Admissions & Support: support@mockattempt.com
Phone / WhatsApp: +91 95400 07917
Course Code: NBCP • **Updated:** 18 Nov 2026
https://mockattempt.com/training/nodejs-backend-developer-career-program

NBCP • NODE.JS, TYPESCRIPT & EXPRESS BACKEND ENGINEERING **BEGINNER TO JOB-READY**

100% PLACEMENT GUARANTEED* CAREER PROGRAM (6-MO REFUND*)

Node.js Backend Developer – 6-Month Industry-Ready Career Program

100% Backend-Focused 6-Month Career Program: Master JavaScript ES6+, TypeScript 5, Node.js (V8/libuv/Streams), Express.js REST APIs, MongoDB + Mongoose, PostgreSQL + Prisma, JWT/OAuth2/RBAC, Redis, BullMQ, RabbitMQ/Kafka, Socket.IO, Jest/Supertest, Docker, Nginx & CI/CD across 24 modules and 6 mandatory projects.

Duration: 24 Weeks (6 Months) • Beginner to Job-Ready (600 Hrs) • **Modules:** 24 • **Live Topics:** 543 • **Subtopics:** 4,344+ • **Labs & Tasks:** 1086 • **Mocks / AI Vivas:** 543 / 543 • **Projects:** 24 Module + 1 Live Client Prod

All-Inclusive Direct Fee (Incl. 18% GST • Zero Hidden Charges): Total Payable: ₹15,000 (Net Tuition ₹12,712 + Included 18% GST ₹2,288) • **Split Option:** 3 × $53 (0% Installments) • **Effective ROI:** ~₹25/Guided Hr (~₹28/Live Topic)

Faculty: 100% Active Working IT Professionals • **Weekend-Only Live Batch (7:00 AM – 8:00 PM IST Flexible Slots):** Weekend-Only Career Live Batch (Sat – Sun • 2 Hrs/Day • 7:00 AM – 8:00 PM) (Saturday & Sunday Only (2 Hours Live Session / Day) • 07:00 AM – 08:00 PM IST (Students Can Choose Any 2-Hour Weekend Slot Based on Working Professional Trainer Availability) • 28 Weeks (600 Guided Live + Lab Hours))

✓ **Trainer LeetCode & HackerRank Mentorship (JavaScript ES6+, TypeScript 5, Node.js, PostgreSQL SQL & MongoDB): Live trainer support to crack LeetCode & HackerRank problems, optimize complexity, and handle edge cases.**

✓ **6-Month Placement Guarantee* + Online AI Assessment (OA) Support: Complete 100% assigned repositories & score ≥90% across all Topic Mock Tests & Voice AI Interviews → 100+ verified IT job openings within 6 months with active Senior Trainer coaching to crack employer Online AI Assessments (OA), or 100% net tuition refund*.**

Core Technologies & Engineering Stack: JavaScript ES6+ & Strict TypeScript 5 • Node.js 22 LTS (V8, libuv & Event Loop) • Buffers, Streams & Async File Processing • Express.js Layered REST Architecture • OpenAPI 3.1, Swagger UI & Zod Validation • MongoDB 7 & Mongoose 8 ODM • PostgreSQL 16, Advanced SQL & Prisma ORM 5 • ACID Transactions, Locking & Index Optimization • JWT Rotation, OAuth 2.0 PKCE & RBAC • Helmet, Argon2id, Rate Limiting & OWASP API Security • Redis 7 Caching, Locks & Pub/Sub • BullMQ Background Queues & Workers • RabbitMQ, Kafka Fundamentals & Outbox Sagas • WebSockets & Socket.IO Real-Time Services • MinIO S3-Compatible Presigned File Storage • Jest, Vitest & Supertest API Testing • Linux, 3-Stage Dockerfiles & Docker Compose • Nginx, PM2, Let's Encrypt TLS & GitHub Actions CI/CD

1. PROGRAM MASTERY OUTCOMES & ELIGIBILITY

End-to-End Engineering Outcomes:

1. JavaScript, TypeScript 5 & Node.js Runtime Mastery: Write zero-`any` TypeScript 5 backend code with deep fluency in V8 memory, `libuv` event-loop phases, backpressure-safe Streams, and `AsyncLocalStorage`.
2. Production Express.js REST APIs: Architect versioned `/api/v1` `Controller -> Service -> Repository` APIs with `Zod` schemas, RFC 9457 Problem Details error handling, cursor pagination, Pino correlation logs, and OpenAPI 3.1 / Swagger docs.
3. Polyglot Databases (MongoDB + PostgreSQL): Design Mongoose document schemas with aggregation pipelines & ESR indexes alongside 3NF PostgreSQL + Prisma schemas with Window Functions, `EXPLAIN ANALYZE`, and deadlock-free `SELECT ... FOR UPDATE` ACID transactions.
4. Enterprise Auth & OWASP API Security: Implement `Argon2id`, short-lived JWT access + rotated SHA-256 refresh-token family reuse detection, OAuth 2.0 PKCE, RBAC/BOLA guards, Redis Lua rate limiters, magic-byte file checks, and signed webhooks.
5. Redis, Queues, Real-Time & Microservices: Scale backends with Redis Cache-Aside + Single-Flight stampede shields, `BullMQ` retry/DLQ workers, `RabbitMQ`/Kafka Transactional Outbox & Saga compensation, and multi-node `Socket.IO` rooms.
6. Performance, Vendor-Neutral DevOps & System Design: Profile `p95`/`p99` latency with `autocannon`, offload CPU work to `worker_threads`, deploy hardened 3-stage non-root Docker Compose + Nginx + PM2 + GitHub Actions stacks on Linux, and lead 45-minute Backend System Design interviews.
7. Crack LeetCode and HackerRank problems in JavaScript ES6+, TypeScript 5, Node.js, PostgreSQL SQL & MongoDB with live trainer mentorship on algorithmic/query patterns, edge cases, and optimal complexity.
8. Receive active Senior Trainer coaching during your 6-month (180-day) Placement Tenure to crack employer Online AI Assessments (OA), timed coding rounds, and automated screening tests.

9. Automated Testing, Debugging & Quality Gates (JavaScript ES6+ & Strict TypeScript 5, Node.js 22 LTS (V8, libuv & Event Loop), Buffers, Streams & Async File Processing, Express.js Layered REST Architecture, OpenAPI 3.1, Swagger UI & Zod Validation) — Automated JavaScript ES6+ & Strict TypeScript 5, Node.js 22 LTS (V8, libuv & Event Loop), Buffers, Streams & Async File Processing, Express.js Layered REST Architecture, OpenAPI 3.1, Swagger UI & Zod Validation test suites • Linting & static checks • Mentor Git pull-request reviews — ensuring your code never breaks silently when real users hit edge cases.
10. 100% Topic-Wise Timed Mock Tests, Spoken MockAttempt AI Viva & Live Client Production — Verify every single Node.js Backend Developer – 6-Month Industry-Ready Career Program topic through timed MCQ exams and spoken technical interviews before deploying your Live Client Project.
11. Architected and implemented 'Enterprise E-Commerce and Order Management Backend' using JavaScript ES6+ & Strict TypeScript 5, Node.js 22 LTS (V8, libuv & Event Loop), Buffers, Streams & Async File Processing, Express.js Layered REST Architecture, OpenAPI 3.1, Swagger UI & Zod Validation under Senior Working IT Professional code review.
12. Engineered production authentication/RBAC, schema migrations, input validation, caching/security hardening, and clean modular layers.

Eligibility & Prerequisites: No prior backend framework or TypeScript experience required — Month 1 builds JavaScript ES6+, asynchronous execution, strict TypeScript 5, OOP, Git, Linux CLI, and HTTP fundamentals from scratch. • Consistent 25 hours/week commitment (~600 total hours across 24 weeks) completing all 6 mandatory backend projects, 105 JS/TS DSA problems, and 30 Backend Machine-Coding exercises. • Ideal for B.Tech / BCA / MCA / B.Sc IT students, non-CS beginners, career switchers, and working developers upgrading to 2026 production standards. • Dedication to write original code in your GitHub portfolio and complete the mentor-guided Live Client Production Sprint.

2. DETAILED ZERO-TO-EXPERT MODULE, TOPIC & SUBTOPIC SYLLABUS (24 MODULES • 543 LIVE TOPICS • 4,344+ SUBTOPICS)

Progressive Zero-to-Expert Mastery Stages: Modules 1–4: Month 1 – Month 1: JavaScript, TypeScript and Backend Foundations (Weeks 1–4 (Modules 1–4)) (85 Topics / 680 Subtopics) • Modules 5–8: Month 2 – Month 2: Node.js Runtime and Express.js Backend Development (Weeks 5–8 (Modules 5–8)) (92 Topics / 736 Subtopics) • Modules 9–12: Month 3 – Month 3: Database Engineering and ORM / ODM (Weeks 9–12 (Modules 9–12)) (90 Topics / 720 Subtopics) • Modules 13–16: Month 4 – Month 4: API Security, Validation, Advanced Features and Testing (Weeks 13–16 (Modules 13–16)) (95 Topics / 760 Subtopics) • Modules 17–20: Month 5 – Month 5: Redis, Queues, Messaging, WebSockets and Microservices (Weeks 17–20 (Modules 17–20)) (99 Topics / 792 Subtopics) • Modules 21–24: Month 6 – Month 6: Performance, Docker, Linux/Nginx CI/CD, System Design & Capstone (Weeks 21–24 (Modules 21–24)) (82 Topics / 656 Subtopics)

MODULE 1 OF 24 • STAGE 1 • 25 LIVE + LAB HOURS • 15 TOPICS (120 SUBTOPICS) • 1 CAPSTONE PROJECT (100 MARKS)

Module 1: JavaScript Programming Fundamentals

Build rock-solid JavaScript ES6+ programming foundations for server-side engineering — variables, types, strict equality, control flow, strings, functional arrays, objects, functions, lexical scope/hoisting, structured try/catch error handling, and Node inspector debugging.

Objectives: ✓ Write clean, predictable ES6+ JavaScript running on Node.js without implicit coercion bugs or global scope leaks. • ✓ Manipulate arrays, strings, and nested objects fluently using immutable functional patterns (`map`, `filter`, `reduce`). • ✓ Apply lexical scoping, arrow vs regular functions, and custom `Error` handling across modular CLI programs. • ✓ Complete all 4 Week 1 practical builds: Calculator, Student Records Processor, Employee Data Manager, and CLI Utility.

1.1 Introduction to JavaScript and Server-Side Execution Environments [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: ECMAScript specification vs host environments (Node.js server runtime vs browser) • V8 JIT compilation • Running scripts via `node --watch` • Production Backend Pitfall & Guardrail (Introduction to JavaScript and Server-Side Execution Environments) • Hands-On TypeScript / Node.js Implementation (Introduction to JavaScript and Server-Side Execution Environments) • Introduction to JavaScript and Server-Side Execution Environments — Latency, Throughput & Interview Trade-Offs • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Introduction to JavaScript

Purpose: Running high-concurrency JSON APIs, CLI tools, and background workers directly on Linux servers with Node.js.

What Topic Is Next → Next Topic 1.2: Variables: var, let and const

1.2 Variables: var, let and const [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Function-scoped hoisted `var` pitfalls vs block-scoped `let` • `const` • The Temporal Dead Zone (`TDZ`) • `const` binding vs object mutability • Production Backend Pitfall & Guardrail (Variables: var, let and const) • Hands-On TypeScript / Node.js Implementation (Variables: var, let and const) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Variables: var

Purpose: Defaulting 95%+ of backend bindings to `const` and reserving `let` strictly for loop counters or state accumulators.

What Topic Is Next → Next Topic 1.3: Primitive Data Types and Reference Data Types

1.3 Primitive Data Types and Reference Data Types [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: 7 primitives (`string` • `number` • `bigint` • `boolean` • Production Backend Pitfall & Guardrail (Primitive Data Types and Reference Data Types) • Hands-On TypeScript / Node.js Implementation (Primitive Data Types and Reference Data Types) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Primitive Data Types

Purpose: Storing monetary amounts in integer paise/cents (`1500000` paise) or `Decimal` instead of binary IEEE-754 floats!

What Topic Is Next → Next Topic 1.4: Type Conversion and Type Coercion

1.4 Type Conversion and Type Coercion [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Explicit parsing (`Number()` • `Number.parseInt(s • 10)` • `String()` • Production Backend Pitfall & Guardrail (Type Conversion and Type Coercion) • Hands-On TypeScript / Node.js Implementation (Type Conversion and Type Coercion) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Type Conversion

Purpose: Safely parsing HTTP query string `?page=2&limit=20` into validated finite integers (`Number.isInteger`).

What Topic Is Next → Next Topic 1.5: Operators and Expressions

1.5 Operators and Expressions [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Strict equality (`===` • `!==`) • Arithmetic/exponentiation (`**`) • Logical short-circuiting (`&&` • Production Backend Pitfall & Guardrail (Operators and Expressions) • Hands-On TypeScript / Node.js Implementation (Operators and Expressions) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Operators

Purpose: Applying `limit = rawLimit ?? 20` so explicit `0` is preserved while `null`/`undefined` falls back cleanly.

What Topic Is Next → Next Topic 1.6: Conditional Statements and Guard Clauses

1.6 Conditional Statements and Guard Clauses [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `if / else if / else` • Early-return guard clauses flattening nested arrow-antipatterns • Exhaustive `switch` blocks with block scoping • Production Backend Pitfall & Guardrail (Conditional Statements and Guard Clauses) • Hands-On TypeScript / Node.js Implementation (Conditional Statements and Guard Clauses) • Conditional Statements and Guard Clauses — Latency, Throughput & Interview Trade-Offs • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Conditional Statements

Purpose: Validating auth, permissions, and payload preconditions via crisp top-of-function guard returns.

What Topic Is Next → Next Topic 1.7: Loops: for, while, do-while, for...of and for...in

1.7 Loops: for, while, do-while, for...of and for...in [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Indexed `for` • Condition-driven `while`/`do-while` • Iterable `for...of` (`Array` • `Map` • Production Backend Pitfall & Guardrail (Loops: for, while, do-while, for...of and for...in) • Hands-On TypeScript / Node.js Implementation (Loops: for, while, do-while, for...of and for...in) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Loops: for

Purpose: Using `for...of` (unlike `.forEach()`) when sequential `await` or early `break` is required inside backend batch processors!

What Topic Is Next → Next Topic 1.8: Strings and String Methods

1.8 Strings and String Methods [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Immutable UTF-16 strings • `slice` • `trim` • `toLowerCase` • Production Backend Pitfall & Guardrail (Strings and String Methods) • Hands-On TypeScript / Node.js Implementation (Strings and String Methods) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Strings

Purpose: Canonicalizing incoming registration emails (`email.trim().toLowerCase()`) and generating zero-padded invoice codes (`INV-00042`).

What Topic Is Next → Next Topic 1.9: Arrays and Functional Array Methods

1.9 Arrays and Functional Array Methods [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Mutating (`push` • `pop` • `splice` • `sort`) vs non-mutating ES2023+ (`toSorted` • Production Backend Pitfall & Guardrail (Arrays and Functional Array Methods) • Hands-On TypeScript / Node.js Implementation (Arrays and Functional Array Methods) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Arrays

Purpose: Aggregating line-item subtotals, taxes, and active SKUs cleanly with `filter` + `reduce` pipelines.

What Topic Is Next → Next Topic 1.10: Objects and Object Manipulation

1.10 Objects and Object Manipulation [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Object literals • Computed keys `[key]` • Dynamic property lookup (`Object.hasOwn(obj • Key)`) • Production Backend Pitfall & Guardrail (Objects and Object Manipulation) • Hands-On TypeScript / Node.js Implementation (Objects and Object Manipulation) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Objects

Purpose: Using `Object.hasOwn()` or `new Map()` when indexing untrusted user-supplied keys so `key = "__proto__"` never triggers Prototype Pollution!

What Topic Is Next → Next Topic 1.11: Functions, Parameters and Return Values

1.11 Functions, Parameters and Return Values [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: First-class functions • Arity • Default parameter expressions (evaluated per call!) • Rest parameters `(...items)` • Production Backend Pitfall & Guardrail (Functions, Parameters and Return Values) • Hands-On TypeScript / Node.js Implementation (Functions, Parameters and Return Values) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Functions

Purpose: Designing pure, deterministic pricing and tax calculation functions trivial to unit-test in Jest/Vitest.

What Topic Is Next → Next Topic 1.12: Function Declarations, Expressions and Arrow Functions

1.12 Function Declarations, Expressions and Arrow Functions [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Hoisted declarations vs `const fn = function() {}` expressions vs concise `=>` arrow functions (lexical `this` • No `arguments` • Cannot be `new`-constructed • `() => ({ key: 1 })` parenthesized object return) • Production Backend Pitfall & Guardrail (Function Declarations, Expressions and Arrow Functions) • Hands-On TypeScript / Node.js Implementation (Function Declarations, Expressions and Arrow Functions) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Function Declarations

Purpose: Using concise arrow callbacks inside array/Promise pipelines while using named functions/methods for clear stack traces.

What Topic Is Next → Next Topic 1.13: Scope and Hoisting

1.13 Scope and Hoisting [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Global • Module • Function • `{}` block scopes • Production Backend Pitfall & Guardrail (Scope and Hoisting) • Hands-On TypeScript / Node.js Implementation (Scope and Hoisting) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Scope

Purpose: Keeping module state private inside ES module scope without leaking globals across concurrent HTTP requests.

What Topic Is Next → Next Topic 1.14: Error Handling Using try, catch, finally

1.14 Error Handling Using try, catch, finally [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Synchronous exception unwinding • Custom `class DomainError extends Error` with `cause` chaining (`new Error("msg" • { cause: err })`) • `finally` resource cleanup • Production Backend Pitfall & Guardrail (Error Handling Using try, catch, finally) • Hands-On TypeScript / Node.js Implementation (Error Handling Using try, catch, finally) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Error Handling Using try

Purpose: Wrapping JSON parsing and external failures in `DomainError` subclasses preserving `.stack` and `{ cause }` for structured logs.

What Topic Is Next → Next Topic 1.15: Debugging and Developer Tools for Backend JS

1.15 Debugging and Developer Tools for Backend JS [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Structured `console.table`/`console.time` • `node --inspect-brk` • VS Code `launch.json` breakpoints • Conditional breakpoints • Production Backend Pitfall & Guardrail (Debugging and Developer Tools for Backend JS) • Hands-On TypeScript / Node.js Implementation (Debugging and Developer Tools for Backend JS) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Debugging

Purpose: Attaching VS Code's debugger to step through a failing data-transformation loop in 30 seconds instead of littering 40 `console.log`s.

What Topic Is Next → Module 1 Capstone (Week 1 Lab Build: JavaScript ES6+ Payroll, Ledger & Text Analytics CLI Suite) → Next Topic 2.1: Execution Context and the Call Stack

Module 1 Hands-On Capstone Project (100 Marks): Week 1 Lab Build: JavaScript ES6+ Payroll, Ledger & Text Analytics CLI Suite — Build a zero-dependency Node.js ES6+ command-line utility pack implementing immutable payroll tax slabs, array/object grouping pipelines, string/CSV log normalizers, and structured custom Er...

MODULE 2 OF 24 • STAGE 1 • 25 LIVE + LAB HOURS • 22 TOPICS (176 SUBTOPICS) • 1 CAPSTONE PROJECT (100 MARKS)

Module 2: Advanced JavaScript and Asynchronous Programming

Master how JavaScript truly works under the hood — execution contexts, closures, `this` binding, prototypes, OOP classes & `#private` encapsulation, destructuring/spread, CommonJS vs ESM, Promises, `async/await`, `Promise.all`/`allSettled`, microtasks vs macrotasks, async error propagation, and V8 garbage collection.

Objectives: ✓ Trace execution contexts, closures, `this` binding (`call`/`apply`/`bind`), and the prototype chain accurately in interviews and code reviews. • ✓ Design clean OOP domain models using ES2022+ `class`, `#private` fields, inheritance, and composition. • ✓ Orchestrate concurrent and sequential I/O using Promises, `async/await`, and `Promise.all` / `allSettled` / `race` / `any`. • ✓ Predict exact event-loop execution order across synchronous frames, microtasks, and macrotasks while avoiding unhandled rejections and memory leaks.

2.1 Execution Context and the Call Stack [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Global & Function Execution Contexts (Creation Phase vs Execution Phase) • Variable Environment • Lexical Environment • LIFO Call Stack frames • Production Backend Pitfall & Guardrail (Execution Context and the Call Stack) • Hands-On TypeScript / Node.js Implementation (Execution Context and the Call Stack) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Execution Context

Purpose: Reading multi-frame Node.js stack traces bottom-up and converting deep recursion into iterative loops for large payloads.

What Topic Is Next → Next Topic 2.2: Lexical Scope and Closures

2.2 Lexical Scope and Closures [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: How inner functions retain live references to outer lexical environments even after the outer function returns — powering factory functions • Memoization • Rate counters • Middleware configuration! • Production Backend Pitfall & Guardrail (Lexical Scope and Closures) • Hands-On TypeScript / Node.js Implementation (Lexical Scope and Closures) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Lexical Scope

Purpose: Creating configurable Express middleware `rateLimit({ windowMs: 60000, max: 100 })` that closes over options cleanly.

What Topic Is Next → Next Topic 2.3: The this Keyword and Binding Rules

2.3 The this Keyword and Binding Rules [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: 4 `this` binding rules in priority order: (1) `new` binding • (2) explicit `call`/`apply`/`bind` • (3) implicit method `obj.fn()` • (4) default (`undefined` in strict mode!) • Production Backend Pitfall & Guardrail (The this Keyword and Binding Rules) • Hands-On TypeScript / Node.js Implementation (The this Keyword and Binding Rules) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for The this Keyword

Purpose: Preventing the classic Express bug `router.get("/users", userController.list)` where `this` inside `list()` becomes `undefined`!

What Topic Is Next → Next Topic 2.4: call(), apply() and bind()

2.4 call(), apply() and bind() [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Invoking functions immediately with explicit `this` (`fn.call(ctx • B)` vs `fn.apply(ctx • ArgsArray)`) vs creating a permanently bound function (`fn.bind(ctx)`) • Production Backend Pitfall & Guardrail (call(), apply() and bind()) • Hands-On TypeScript / Node.js Implementation (call(), apply() and bind()) • call(), apply() and bind() — Latency, Throughput & Interview Trade-Offs • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for call()

Purpose: Writing timing/audit method wrappers that forward `return origMethod.apply(this, args)` preserving caller context.

What Topic Is Next → Next Topic 2.5: Prototypes and Prototypal Inheritance

2.5 Prototypes and Prototypal Inheritance [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `Object.getPrototypeOf()` • `Fn.prototype` vs instance `[[Prototype]]` • Property lookup delegation along the prototype chain • `Object.create(null)` dictionary maps • Production Backend Pitfall & Guardrail (Prototypes and Prototypal Inheritance) • Hands-On TypeScript / Node.js Implementation (Prototypes and Prototypal Inheritance) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Prototypes

Purpose: Understanding that ES6 `class` methods live on `Class.prototype` (shared memory) whereas instance arrow fields allocate a new closure per object!

What Topic Is Next → Next Topic 2.6: Constructor Functions and ES6 Classes

2.6 Constructor Functions and ES6 Classes [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: How `new` works in 4 steps (create object -> link `[[Prototype]]` -> bind `this` -> return object) • Modern `class` syntax (`constructor` • Instance methods • `static` factory methods • Production Backend Pitfall & Guardrail (Constructor Functions and ES6 Classes) • Hands-On TypeScript / Node.js Implementation (Constructor Functions and ES6 Classes) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Constructor Functions

Purpose: Providing expressive `static fromRow(row)` factory constructors on backend domain entities.

What Topic Is Next → Next Topic 2.7: Object-Oriented Programming in JavaScript

2.7 Object-Oriented Programming in JavaScript [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Modelling cohesive backend entities • Value objects • Services • Repositories with clear responsibilities rather than anemic global helper bags • Production Backend Pitfall & Guardrail (Object-Oriented Programming in JavaScript) • Hands-On TypeScript / Node.js Implementation (Object-Oriented Programming in JavaScript) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Object-Oriented Programming in JavaScript

Purpose: Keeping domain invariants (`order.markPaid(txnId)` rejects invalid state transitions!) inside the `Order` entity.

What Topic Is Next → Next Topic 2.8: Inheritance and Polymorphism

2.8 Inheritance and Polymorphism [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `extends` • `super()` • Method overriding • Polymorphic dispatch across notification/payment strategies (`EmailNotifier` • Production Backend Pitfall & Guardrail (Inheritance and Polymorphism) • Hands-On TypeScript / Node.js Implementation (Inheritance and Polymorphism) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Inheritance

Purpose: Dispatching `await notifier.send(payload)` polymorphically across multiple channels without `if/else` chains.

What Topic Is Next → Next Topic 2.9: Encapsulation (#private Fields) and Abstraction

2.9 Encapsulation (#private Fields) and Abstraction [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: True runtime-enforced ECMAScript `#private` fields & `#private` methods (inaccessible even via `Object.keys` or `JSON.stringify`!) • Exposing narrow public APIs • Production Backend Pitfall & Guardrail (Encapsulation (#private Fields) and Abstraction) • Hands-On TypeScript / Node.js Implementation (Encapsulation (#private Fields) and Abstraction) • Encapsulation (#private Fields) and Abstraction — Latency, Throughput & Interview Trade-Offs • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Encapsulation (#private Fields) • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards

Purpose: Hiding `#apiSecret` inside an `AuthTokenSigner` instance so accidental `JSON.stringify(signer)` or `console.log` cannot leak keys!

What Topic Is Next → Next Topic 2.10: Object and Array Destructuring

2.10 Object and Array Destructuring [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Nested destructuring • Property aliasing (`{ user_id: userId }`) • Default values (`= 20` triggered ONLY on `undefined` • NOT `null`!) • Production Backend Pitfall & Guardrail (Object and Array Destructuring) • Hands-On TypeScript / Node.js Implementation (Object and Array Destructuring) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Object

Purpose: Extracting `{ page = 1, limit = 20, sort = "createdAt" } = req.query` cleanly at controller boundaries.

What Topic Is Next → Next Topic 2.11: Spread and Rest Operators (Shallow vs Deep Clone)

2.11 Spread and Rest Operators (Shallow vs Deep Clone) [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Immutable shallow updates (`{ ...user • Role: "admin" }`) • Rest omission (`const { passwordHash • ...safeUser } = dbUser`!) • Production Backend Pitfall & Guardrail (Spread and Rest Operators (Shallow vs Deep Clone)) • Hands-On TypeScript / Node.js Implementation (Spread and Rest Operators (Shallow vs Deep Clone)) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Spread

Purpose: Stripping sensitive `passwordHash` and `refreshTokenHash` fields cleanly via rest destructuring before returning JSON.

What Topic Is Next → Next Topic 2.12: Template Literals and Tagged Templates

2.12 Template Literals and Tagged Templates [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Multi-line strings • `${expr}` interpolation • How **Tagged Template Literals** (`sql\`SELECT * FROM users WHERE id = \${id}\``) power SQL-injection-safe parameterized queries in Prisma & Postgres! • Production Backend Pitfall & Guardrail (Template Literals and Tagged Templates) • Hands-On TypeScript / Node.js Implementation (Template Literals and Tagged Templates) • Template Literals and Tagged Templates — Latency, Throughput & Interview Trade-Offs • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career

Program Production Runtime Execution & Memory Lifecycle for Template Literals

Purpose: Understanding WHY Prisma's `prisma.$queryRaw\`SELECT ... WHERE id = \${id}\`` separates SQL text from `$1` bind values safely — unlike regular string interpolation!

What Topic Is Next → Next Topic 2.13: Optional Chaining (?.) and Nullish Coalescing (??)

2.13 Optional Chaining (?.) and Nullish Coalescing (??) [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Safe deep access `req.user?.organization?.plan ?? "free"` • Optional method `onComplete?.()` • Optional bracket `items?.[0]` • Production Backend Pitfall & Guardrail (Optional Chaining (?.) and Nullish Coalescing (??)) • Hands-On TypeScript / Node.js Implementation (Optional Chaining (?.) and Nullish Coalescing (??)) • Optional Chaining (?.) and Nullish Coalescing (??) — Latency, Throughput & Interview Trade-Offs • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Optional Chaining (?.)

Purpose: Safely reading optional webhook nested metadata `payload?.data?.object?.metadata?.orderId ?? null` without crashes.

What Topic Is Next → Next Topic 2.14: Modules: CommonJS vs ES Modules Preview

2.14 Modules: CommonJS vs ES Modules Preview [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Synchronous dynamic `require()` (`module.exports`) vs static asynchronous ESM (`import` / `export`) • File extensions in Node ESM • Interoperability (`import pkg from "cjs-pkg"`) • Production Backend Pitfall & Guardrail (Modules: CommonJS vs ES Modules Preview) • Hands-On TypeScript / Node.js Implementation (Modules: CommonJS vs ES Modules Preview) • Modules: CommonJS vs ES Modules Preview — Latency, Throughput & Interview Trade-Offs • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Modules: CommonJS vs ES Modules Preview

Purpose: Writing modern ESM/TypeScript backend modules with clean named exports for tree-shaking and explicit dependency graphs.

What Topic Is Next → Next Topic 2.15: Callbacks and Error-First Callback Patterns

2.15 Callbacks and Error-First Callback Patterns [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Legacy Node `(err • Result) => {}` convention • Pyramid of Doom / Callback Hell • Double-invocation hazards • Production Backend Pitfall & Guardrail (Callbacks and Error-First Callback Patterns) • Hands-On TypeScript / Node.js Implementation (Callbacks and Error-First Callback Patterns) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Callbacks

Purpose: Converting callback-based `crypto.scrypt` or legacy SDK methods into clean awaitable Promises via `util.promisify`.

What Topic Is Next → Next Topic 2.16: Promises and Promise Chaining

2.16 Promises and Promise Chaining [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Promise states (`pending` -> settled `fulfilled` or `rejected` — immutable once settled!) • `.then()` • `.catch()` • `.finally()` • Production Backend Pitfall & Guardrail (Promises and Promise Chaining) • Hands-On TypeScript / Node.js Implementation (Promises and Promise Chaining) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Promises

Purpose: Wrapping event-based streams or timers (`const sleep = (ms) => new Promise((r) => setTimeout(r, ms))`) into clean Promises.

What Topic Is Next → Next Topic 2.17: async and await Execution Mechanics

2.17 async and await Execution Mechanics [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: How `async` always wraps return values in a Promise • How `await` suspends ONLY that async function (freeing the Node.js event loop for other requests!) • Avoiding accidental sequential `await` bottlenecks! • Production Backend Pitfall & Guardrail (async and await Execution Mechanics) • Hands-On TypeScript / Node.js Implementation (async and await Execution Mechanics) • async and await Execution Mechanics — Latency, Throughput & Interview Trade-Offs • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for async

Purpose: Running independent `getUser(id)` and `getPermissions(id)` in parallel (`await Promise.all([...])` in `30ms`) instead of sequential `await` (`30ms + 30ms = 60ms`)!

What Topic Is Next → Next Topic 2.18: Promise Combinators: all, allSettled, race and any

2.18 Promise Combinators: all, allSettled, race and any [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Exact failure semantics: `Promise.all` (fail-fast on 1st reject) • `Promise.allSettled` (waits for all -> `{ status • Value | reason }`) • `Promise.race` (1st settled wins) • Production Backend Pitfall & Guardrail (Promise Combinators: all, allSettled, race and any) • Hands-On TypeScript / Node.js Implementation (Promise Combinators: all, allSettled, race and any) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Promise Combinators

Purpose: Using `Promise.allSettled` when sending batch push notifications to 50 devices so 1 invalid device token doesn't abort the other 49!

What Topic Is Next → Next Topic 2.19: Event Loop and Task Queues

2.19 Event Loop and Task Queues [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: How single-threaded JS achieves high I/O concurrency: Call Stack empties -> drain ALL Microtasks -> pick next Macrotask -> drain Microtasks -> repeat • Production Backend Pitfall & Guardrail (Event Loop and Task Queues) • Hands-On TypeScript / Node.js Implementation (Event Loop and Task Queues) • Event Loop and Task Queues — Latency, Throughput & Interview Trade-Offs • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Event Loop • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards • Diagnostics, Profiling & Telemetry: structured telemetry, profiler traces & CI checks

Purpose: Explaining why a `500ms` synchronous `JSON.parse` or regex freeze delays EVERY concurrent HTTP health check and timer.

What Topic Is Next → Next Topic 2.20: Microtasks vs Macrotasks Ordering

2.20 Microtasks vs Macrotasks Ordering [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Priority order in Node.js: Synchronous Stack -> `process.nextTick` -> Promise Microtasks (`.then` • `await` continuations • `queueMicrotask`) -> Macrotasks (`setTimeout` • `setImmediate` • Production Backend Pitfall & Guardrail (Microtasks vs Macrotasks Ordering) • Hands-On TypeScript / Node.js Implementation (Microtasks vs Macrotasks Ordering) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Microtasks vs Macrotasks Ordering

Purpose: Predicting exact execution order in interviews and avoiding recursive `process.nextTick` I/O starvation!

What Topic Is Next → Next Topic 2.21: Error Propagation in Asynchronous Code

2.21 Error Propagation in Asynchronous Code [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `try / catch` around `await` • Why `return fn()` vs `return await fn()` differs inside `try / catch` blocks (`return fn()` skips local `catch`!) • Handling `unhandledRejection` • Production Backend Pitfall & Guardrail (Error Propagation in Asynchronous Code) • Hands-On TypeScript / Node.js Implementation (Error Propagation in Asynchronous Code) • Error Propagation in Asynchronous Code — Latency, Throughput & Interview Trade-Offs • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Error Propagation in Asynchronous Code

Purpose: Using `return await db.save()` inside a `try { ... } catch (err) { await tx.rollback(); throw err; }` block so errors actually hit `catch`!

What Topic Is Next → Next Topic 2.22: Memory Management and Garbage Collection Fundamentals

2.22 Memory Management and Garbage Collection Fundamentals [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: V8 Young Generation (`Scavenge` Nursery) vs Old Generation (`Mark-Sweep-Compact`) • Root reachability • The 4 classic Node.js memory leaks (unbounded in-memory caches/arrays • Unremoved `EventEmitter` listeners • Production Backend Pitfall & Guardrail (Memory Management and Garbage Collection Fundamentals) • Hands-On TypeScript / Node.js Implementation (Memory Management and Garbage Collection Fundamentals) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Memory Management

Purpose: Bounding in-memory `Map` caches with max size + TTL and using `WeakMap` for ephemeral request metadata.

What Topic Is Next → Module 2 Capstone (Week 2 Lab Build: Async Concurrency Limiter, Retry Runner & Event Loop Tracer) → Next Topic 3.1: TypeScript Introduction, Installation and Build/Run Workflow

Module 2 Hands-On Capstone Project (100 Marks): Week 2 Lab Build: Async Concurrency Limiter, Retry Runner & Event Loop Tracer — Implement a pure JavaScript `pLimit(concurrency)` task pool, exponential-backoff + full-jitter `withRetry(fn)` wrapper, `AbortSignal.timeout()` cancellable HTTP client, and deterministic mic...

MODULE 3 OF 24 • STAGE 1 • 25 LIVE + LAB HOURS • 22 TOPICS (176 SUBTOPICS) • 1 CAPSTONE PROJECT (100 MARKS)

Module 3: TypeScript Complete Fundamentals

Master production TypeScript 5 for backend engineering — strict compiler configuration, type narrowing & custom type guards, discriminated unions, generics, interfaces vs types, OOP classes & abstract contracts, utility types (Partial, Pick, Omit, Record, ReturnType), keyof/typeof/infer, and type-safe API DTO architecture.

Objectives: ✓ Configure strict `tsconfig.json` (`strict: true`, `noUncheckedIndexedAccess: true`) catching `undefined`/null bugs at compile time. • ✓ Model expressive backend domain states using Discriminated Unions, Type Guards, Interfaces, and Generics. • ✓ Leverage built-in Utility Types (`Partial`, `Pick`, `Omit`, `Record`, `Readonly`) and `keyof`/`typeof` without duplicating DTO interfaces. • ✓ Structure clean, fast-compiling multi-folder TypeScript Node.js backend projects.

3.1 TypeScript Introduction, Installation and Build/Run Workflow [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Static structural type system + full type erasure at runtime • Installing `typescript` + `@types/node` • Running via `tsx` in dev vs `tsc` -> `node dist/server.js` in production • Production Backend Pitfall & Guardrail (TypeScript Introduction, Installation and Build/Run Workflow) • Hands-On TypeScript / Node.js Implementation (TypeScript Introduction, Installation and Build/Run Workflow) • TypeScript Introduction, Installation and Build/Run Workflow — Latency, Throughput & Interview Trade-Offs • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for TypeScript Introduction

Purpose: Catching renamed database column typos across 40 service files in 1 second via `tsc --noEmit` in CI before deploying!

What Topic Is Next → Next Topic 3.2: Type Annotations and Type Inference

3.2 Type Annotations and Type Inference [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: When TypeScript infers local constants/variables cleanly vs why explicit parameter • Public function/method return types prevent accidental contract drift • Production Backend Pitfall & Guardrail (Type Annotations and Type Inference) • Hands-On TypeScript / Node.js Implementation (Type Annotations and Type Inference) • Type Annotations and Type Inference — Latency, Throughput & Interview Trade-Offs • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Type Annotations • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards

Purpose: Annotating explicit return types on Service and Repository methods so accidental DB schema leaks fail at the source method!

What Topic Is Next → Next Topic 3.3: Primitive and Special Types: unknown, never, void and any

3.3 Primitive and Special Types: unknown, never, void and any [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Why `unknown` is the type-safe top type (forces narrowing before property access!) vs unsafe `any` (disables type checking!) • `never` for exhaustive checks • `void` • Production Backend Pitfall & Guardrail (Primitive and Special Types: unknown, never, void and any) • Hands-On TypeScript / Node.js Implementation (Primitive and Special Types: unknown, never, void and any) • Primitive and Special Types: unknown, never, void and any — Latency, Throughput & Interview Trade-Offs • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Primitive

Purpose: Typing `catch (err: unknown)` and untrusted `JSON.parse(raw): unknown` safely instead of `any`!

What Topic Is Next → Next Topic 3.4: Typed Arrays, Readonly Arrays, Tuples and Enums vs Const Unions

3.4 Typed Arrays, Readonly Arrays, Tuples and Enums vs Const Unions [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `string[]` • `readonly string[]` (prevents accidental `.push()`/`.sort()` mutation!) • Fixed-position tuples `[number • String]` • Production Backend Pitfall & Guardrail (Typed Arrays, Readonly Arrays, Tuples and Enums vs Const Unions) • Hands-On TypeScript / Node.js Implementation (Typed Arrays, Readonly Arrays, Tuples and Enums vs Const Unions) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Typed Arrays

Purpose: Defining `export const ORDER_STATUSES = ["PENDING", "PAID", "SHIPPED", "CANCELLED"] as const; export type OrderStatus = (typeof ORDER_STATUSES)[number];`!

What Topic Is Next → Next Topic 3.5: Function Types, Call Signatures and Overloads

3.5 Function Types, Call Signatures and Overloads [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Function type aliases `(req: AuthReq) => Promise<ResponseDto>` • Callback typing • Function overload signatures when return type depends on input mode • Production Backend Pitfall & Guardrail (Function Types, Call Signatures and Overloads) • Hands-On TypeScript / Node.js Implementation (Function Types, Call Signatures and Overloads) • Function Types, Call Signatures and Overloads — Latency, Throughput & Interview Trade-Offs • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Function Types

Purpose: Typing higher-order middleware factories and retry wrappers (`withRetry<T>(fn: () => Promise<T>): Promise<T>`).

What Topic Is Next → Next Topic 3.6: Optional and Default Parameters in TypeScript

3.6 Optional and Default Parameters in TypeScript [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `limit?: number` (`number | undefined`) vs `limit = 20` (`number`) • Ordering required params before optional params • `PaginationOptions = { page?: number • Limit?: number }` objects • Production Backend Pitfall & Guardrail (Optional and Default Parameters in TypeScript) • Hands-On TypeScript / Node.js Implementation (Optional and Default Parameters in TypeScript) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Optional

Purpose: Designing clean repository `findMany(filter: UserFilter, options?: PageOptions)` signatures.

What Topic Is Next → Next Topic 3.7: Union and Intersection Types + Discriminated Unions

3.7 Union and Intersection Types + Discriminated Unions [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Tagged **Discriminated Unions** (`type Result<T • E = string> = { ok: true • Data: T } | { ok: false • Error: E }`) where checking `if (!res.ok)` automatically narrows `res.error`! • Production Backend Pitfall & Guardrail (Union and Intersection Types + Discriminated Unions) • Hands-On TypeScript / Node.js Implementation (Union and Intersection Types + Discriminated Unions) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Union

Purpose: Eliminating `null`/`undefined` guesswork across payment/webhook outcomes (`status: "CAPTURED" | "FAILED" | "REFUNDED"`).

What Topic Is Next → Next Topic 3.8: Type Aliases vs Interfaces

3.8 Type Aliases vs Interfaces [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Structural compatibility • `interface` `extends` & declaration merging (needed for augmenting `Express.Request.user`!) vs `type` aliases supporting Unions • Tuples • Mapped Types • Production Backend Pitfall & Guardrail (Type Aliases vs Interfaces) • Hands-On TypeScript / Node.js Implementation (Type Aliases vs Interfaces) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Type Aliases vs Interfaces

Purpose: Using `declare global { namespace Express { interface Request { user?: AuthPayload } } }` to type `req.user` cleanly across middleware!

What Topic Is Next → Next Topic 3.9: Type Narrowing and Custom Type Guards

3.9 Type Narrowing and Custom Type Guards [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Control-flow narrowing (`typeof` • `instanceof` • `"code" in err` • `Array.isArray`) • Production Backend Pitfall & Guardrail (Type Narrowing and Custom Type Guards) • Hands-On TypeScript / Node.js Implementation (Type Narrowing and Custom Type Guards) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Type Narrowing

Purpose: Narrowing PostgreSQL/Prisma error objects (`isPrismaUniqueError(err)`) inside centralized Express error middleware without unsafe `as` casts!

What Topic Is Next → Next Topic 3.10: Literal Types and Template Literal Types

3.10 Literal Types and Template Literal Types [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Exact literal types (`HttpMethod = "GET" | "POST" | "PUT" | "PATCH" | "DELETE"`) • Template literal types (``EventKey = `${Domain}.${Action}` ``) • `satisfies` operator • Production Backend Pitfall & Guardrail (Literal Types and Template Literal Types) • Hands-On TypeScript / Node.js Implementation (Literal Types and Template Literal Types) • Literal Types and Template Literal Types — Latency, Throughput & Interview Trade-Offs • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Literal Types

Purpose: Typing message-broker routing keys (`"order.created" | "order.cancelled" | "payment.captured"`) so typos fail compilation!

What Topic Is Next → Next Topic 3.11: Generics and Generic Constraints

3.11 Generics and Generic Constraints [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Type parameters `<T>` • `<K • V>` • Constraining `<T extends { readonly id: string }>` • Production Backend Pitfall & Guardrail (Generics and Generic Constraints) • Hands-On TypeScript / Node.js Implementation (Generics and Generic Constraints) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Generics

Purpose: Writing ONE type-safe `paginate<T>(items: T[], total: number, page: number, limit: number): PaginatedResponse<T>` used by every endpoint!

What Topic Is Next → Next Topic 3.12: Generic Functions, Interfaces and Classes

3.12 Generic Functions, Interfaces and Classes [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Designing `interface Repository<T • CreateInput • UpdateInput>` implemented identically by `InMemoryRepo` (unit tests) • `PrismaRepo` (PostgreSQL) • Production Backend Pitfall & Guardrail (Generic Functions, Interfaces and Classes) • Hands-On TypeScript / Node.js Implementation (Generic Functions, Interfaces and Classes) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Generic Functions

Purpose: Swapping `InMemoryUserRepository` into unit tests in `1ms` while production injects `PrismaUserRepository`!

What Topic Is Next → Next Topic 3.13: Classes and Access Modifiers in TypeScript

3.13 Classes and Access Modifiers in TypeScript [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Constructor parameter properties (`constructor(private readonly repo: UserRepo) {}`) • `public` / `private` / `protected` / `readonly` • `override` (`noImplicitOverride`) • Production Backend Pitfall & Guardrail (Classes and Access Modifiers in TypeScript) • Hands-On TypeScript / Node.js Implementation (Classes and Access Modifiers in TypeScript) • Classes and Access Modifiers in TypeScript — Latency, Throughput & Interview Trade-Offs • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Classes

Purpose: Writing concise constructor-injected TypeScript Services (`private readonly userRepo: UserRepository`).

What Topic Is Next → Next Topic 3.14: Inheritance and Abstract Classes

3.14 Inheritance and Abstract Classes [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `abstract class BaseRepository<T>` sharing common pagination/logging helpers while enforcing `protected abstract mapRow(row: unknown): T` on concrete subclasses • Production Backend Pitfall & Guardrail (Inheritance and Abstract Classes) • Hands-On TypeScript / Node.js Implementation (Inheritance and Abstract Classes) • Inheritance and Abstract Classes — Latency, Throughput & Interview Trade-Offs • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Inheritance • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards • Diagnostics, Profiling & Telemetry: structured telemetry, profiler traces & CI checks

Purpose: Sharing `AppError` base HTTP error serialization across `BadRequestError`, `UnauthorizedError`, `ForbiddenError`, and `NotFoundError`.

What Topic Is Next → Next Topic 3.15: Interfaces with Classes (Dependency Inversion)

3.15 Interfaces with Classes (Dependency Inversion) [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `class RedisCacheService implements CachePort` • Programming Services against `CachePort` / `MailPort` interfaces so infrastructure adapters remain pluggable • Production Backend Pitfall & Guardrail (Interfaces with Classes (Dependency Inversion)) • Hands-On TypeScript / Node.js Implementation (Interfaces with Classes (Dependency Inversion)) • Interfaces with Classes (Dependency Inversion) — Latency, Throughput & Interview Trade-Offs • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Interfaces with Classes (Dependency Inversion) • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards

Purpose: Testing `AuthService.register()` with a `FakeMailer implements MailPort` capturing OTP emails in memory during Jest tests!

What Topic Is Next → Next Topic 3.16: Utility Types: Partial, Pick, Omit, Record, Required, Readonly

3.16 Utility Types: Partial, Pick, Omit, Record, Required, Readonly [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Deriving `PublicUser = Omit<UserEntity • "passwordHash">` • `UpdateEmployeeDto = Partial<Pick<Employee • "department" | "salary">>` • Production Backend Pitfall & Guardrail (Utility Types: Partial, Pick, Omit, Record, Required, Readonly) • Hands-On TypeScript / Node.js Implementation (Utility Types: Partial, Pick, Omit, Record, Required, Readonly) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Utility Types

Purpose: Preventing security mass-assignment bugs by deriving `UpdateProfileDto = Pick<User, "name" | "phone">` (explicitly excluding `role` & `isVerified`!).

What Topic Is Next → Next Topic 3.17: keyof, typeof and infer Fundamentals

3.17 keyof, typeof and infer Fundamentals [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Type-safe property selectors `<T • K extends keyof T>(obj: T • Key: K): T[K]` • Deriving types from runtime constants via `typeof` • Production Backend Pitfall & Guardrail (keyof, typeof and infer Fundamentals) • Hands-On TypeScript / Node.js Implementation (keyof, typeof and infer Fundamentals) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for keyof

Purpose: Restricting `sortBy: keyof EmployeePublicDto` so callers can NEVER sort on a non-existent or private `passwordHash` column!

What Topic Is Next → Next Topic 3.18: Modules, import type and Avoiding Circular Dependencies

3.18 Modules, import type and Avoiding Circular Dependencies [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `verbatimModuleSyntax` / `import type { User } from "./types.js"` • `.js` specifier resolution in `NodeNext` • Why barrel `index.ts` files cause circular-import `undefined` bugs in large backends • Production Backend Pitfall & Guardrail (Modules, import type and Avoiding Circular Dependencies) • Hands-On TypeScript / Node.js Implementation (Modules, import type and Avoiding Circular Dependencies) • Modules, import type and Avoiding Circular Dependencies — Latency, Throughput & Interview Trade-Offs • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Modules

Purpose: Using `import type` so type-only imports vanish completely from compiled Node.js output with zero runtime module load cost.

What Topic Is Next → Next Topic 3.19: tsconfig.json Configuration for Modern Node.js

3.19 tsconfig.json Configuration for Modern Node.js [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Recommended Node 22 LTS backend `tsconfig.json`: `"target": "ES2022"` • `"module": "NodeNext"` • `"moduleResolution": "NodeNext"` • `"rootDir": "src"` • Production Backend Pitfall & Guardrail (tsconfig.json Configuration for Modern Node.js) • Hands-On TypeScript / Node.js Implementation (tsconfig.json Configuration for Modern Node.js) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for tsconfig.json Configuration for Modern Node.js

Purpose: Generating accurate `.js.map` source maps so production `Pino`/Sentry stack traces point to exact `.ts` line numbers (`node --enable-source-maps`)!

What Topic Is Next → Next Topic 3.20: Strict Type Checking Flags

3.20 Strict Type Checking Flags [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `strict: true` (`strictNullChecks` • `noImplicitAny` • `strictBindCallApply` • `useUnknownInCatchVariables`) PLUS `noUncheckedIndexedAccess: true` (`arr[0]` is `T | undefined` • Production Backend Pitfall & Guardrail (Strict Type Checking Flags) • Hands-On TypeScript / Node.js Implementation (Strict Type Checking Flags) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Strict Type Checking Flags

Purpose: Turning on `noUncheckedIndexedAccess: true` so `const first = rows[0]` forces handling empty query results (`if (!first)`) before `first.id` crashes!

What Topic Is Next → Next Topic 3.21: Type-Safe API Models (Request, Response & Error Contracts)

3.21 Type-Safe API Models (Request, Response & Error Contracts) [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Typing Express `Request<Params • ResBody • ReqBody • ReqQuery>` • Production Backend Pitfall & Guardrail (Type-Safe API Models (Request, Response & Error Contracts)) • Hands-On TypeScript / Node.js Implementation (Type-Safe API Models (Request, Response & Error Contracts)) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Type-Safe API Models (Request, Response & Error Cont...

Purpose: Getting full IDE autocomplete on `req.params.employeeId`, `req.query.department`, and `res.json({ ok: true, data })` inside every controller!

What Topic Is Next → Next Topic 3.22: Production TypeScript Backend Project Structure

3.22 Production TypeScript Backend Project Structure [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Organizing `src/{config • Modules/{employees • Departments} • Shared/{errors • Production Backend Pitfall & Guardrail (Production TypeScript Backend Project Structure) • Hands-On TypeScript / Node.js Implementation (Production TypeScript Backend Project Structure) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Production TypeScript Backend Project Structure

Purpose: Setting up the clean modular foundation used directly in Month 1 Project 1 and Month 2 Project 2!

What Topic Is Next → Module 3 Capstone (Week 3 Lab Build: Strict TypeScript 5 Generic Repository & Result<T, E> Library) → Next Topic 4.1: Git Repository Creation and Identity Configuration

Module 3 Hands-On Capstone Project (100 Marks): Week 3 Lab Build: Strict TypeScript 5 Generic Repository & Result<T, E> Library — Create a strict TypeScript 5 (`strict: true`, `noUncheckedIndexedAccess: true`, zero `any`) library providing discriminated `Result<T, E>`, generic `InMemoryRepository<T extends { id: string...

MODULE 4 OF 24 • STAGE 1 • 25 LIVE + LAB HOURS • 26 TOPICS (208 SUBTOPICS) • 1 CAPSTONE PROJECT (100 MARKS)

Module 4: Git, Linux, Networking and Backend Fundamentals

Master collaborative Git & GitHub pull-request workflows, essential Linux server commands/permissions/processes/SSH/logs, and core TCP/DNS/HTTP/HTTPS/REST/CORS networking — culminating in Month 1 Project 1: TypeScript Employee Management CLI.

Objectives: ✓ Collaborate cleanly using Git branches, rebasing, stashing, conflict resolution, and GitHub Pull Request reviews. • ✓ Navigate Linux filesystems, permissions (`chmod`/`chown`), processes/signals, environment variables, SSH, and log pipelines confidently. • ✓ Explain DNS, TCP/UDP, TLS/HTTPS, raw HTTP request/response headers, status codes, cookies/sessions, REST constraints, and CORS preflights. • ✓ Deliver **Month 1 Project (Project 1): TypeScript-Based Command-Line Employee Management System**.

4.1 Git Repository Creation and Identity Configuration [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Distributed VCS snapshots • `git config --global user.name/email` • `init.defaultBranch main` • `core.autocrlf` cross-OS line-ending hygiene • Production Backend Pitfall & Guardrail (Git Repository Creation and Identity Configuration) • Hands-On TypeScript / Node.js Implementation (Git Repository Creation and Identity Configuration) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Git Repository Creation

Purpose: Preventing noisy `CRLF` vs `LF` diffs when Windows developers push to Linux CI runners.

What Topic Is Next → Next Topic 4.2: Core Git Workflow: init, status, add, commit, log and diff

4.2 Core Git Workflow: init, status, add, commit, log and diff [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Working Tree -> Staging Index -> Local Repo lifecycle • `git add -p` hunk staging • `git diff --staged` • `git log --oneline --graph` • Production Backend Pitfall & Guardrail (Core Git Workflow: init, status, add, commit, log and diff) • Hands-On TypeScript / Node.js Implementation (Core Git Workflow: init, status, add, commit, log and diff) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Core Git Workflow

Purpose: Reviewing `git diff --staged` before every commit to guarantee zero debug logs or accidental secrets enter history.

What Topic Is Next → Next Topic 4.3: Branching, Merging and Rebasing

4.3 Branching, Merging and Rebasing [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Short-lived feature branches (`feat/employee-search`) • Fast-forward vs 3-way merge commits • Interactive `git rebase main` linear history • **NEVER rebasing shared public branches** • Production Backend Pitfall & Guardrail (Branching, Merging and Rebasing) • Hands-On TypeScript / Node.js Implementation (Branching, Merging and Rebasing) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Branching

Purpose: Keeping feature branches cleanly up to date with `main` before opening a Pull Request.

What Topic Is Next → Next Topic 4.4: Undoing Safely: stash, reset and revert

4.4 Undoing Safely: stash, reset and revert [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `git stash push -u` / `pop` • `git restore` • `git reset --soft/--mixed` (local unpushed commits) vs `git revert <sha>` (safe compensating commit on shared branches!) • Production Backend Pitfall & Guardrail (Undoing Safely: stash, reset and revert) • Hands-On TypeScript / Node.js Implementation (Undoing Safely: stash, reset and revert) • Undoing Safely: stash, reset and revert — Latency, Throughput & Interview Trade-Offs • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Undoing Safely

Purpose: Rolling back a faulty merged PR on `main` in 10 seconds using `git revert` without rewriting history.

What Topic Is Next → Next Topic 4.5: Remote Collaboration: clone, fetch, pull and push

4.5 Remote Collaboration: clone, fetch, pull and push [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Origin remotes • Why `git fetch` + inspect is safer than blind `git pull` • `git push -u origin <branch>` • `--force-with-lease` safety • Production Backend Pitfall & Guardrail (Remote Collaboration: clone, fetch, pull and push) • Hands-On TypeScript / Node.js Implementation (Remote Collaboration: clone, fetch, pull and push) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Remote Collaboration

Purpose: Syncing daily upstream changes safely across distributed backend squads.

What Topic Is Next → Next Topic 4.6: GitHub Pull Requests and Engineering Code Reviews

4.6 GitHub Pull Requests and Engineering Code Reviews [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Structuring PR descriptions (Problem • Solution • Test Proof) • Line-by-line review comments • Production Backend Pitfall & Guardrail (GitHub Pull Requests and Engineering Code Reviews) • Hands-On TypeScript / Node.js Implementation (GitHub Pull Requests and Engineering Code Reviews) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for GitHub Pull Requests

Purpose: Merging Project 1–6 features strictly through reviewed GitHub Pull Requests just like real product teams.

What Topic Is Next → Next Topic 4.7: Merge Conflict Resolution

4.7 Merge Conflict Resolution [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Reading `<<<<<<< HEAD` • `=======` • `>>>>>>> branch` conflict markers • Resolving semantic TypeScript/package-lock conflicts • Production Backend Pitfall & Guardrail (Merge Conflict Resolution) • Hands-On TypeScript / Node.js Implementation (Merge Conflict Resolution) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Merge Conflict Resolution

Purpose: Resolving concurrent edits to `employee.service.ts` and regenerating `package-lock.json` cleanly via `npm install`.

What Topic Is Next → Next Topic 4.8: .gitignore Hygiene and Branching Strategies

4.8 .gitignore Hygiene and Branching Strategies [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Node/TS `.gitignore` (`node_modules/` • `dist/` • `coverage/` • `.env` • Production Backend Pitfall & Guardrail (.gitignore Hygiene and Branching Strategies) • Hands-On TypeScript / Node.js Implementation (.gitignore Hygiene and Branching Strategies) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for .gitignore Hygiene

Purpose: Committing a clean `.env.example` template while guaranteeing real `.env` secrets can never be staged.

What Topic Is Next → Next Topic 4.9: Linux Filesystem Hierarchy and Navigation

4.9 Linux Filesystem Hierarchy and Navigation [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: FHS layout (`/etc` config • `/var/log` logs • `/opt` & `/srv` apps • `/tmp` • Production Backend Pitfall & Guardrail (Linux Filesystem Hierarchy and Navigation) • Hands-On TypeScript / Node.js Implementation (Linux Filesystem Hierarchy and Navigation) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Linux Filesystem Hierarchy

Purpose: Knowing immediately where Nginx configs (`/etc/nginx`), app logs (`/var/log`), and systemd units (`/etc/systemd/system`) live on Ubuntu/Debian.

What Topic Is Next → Next Topic 4.10: Linux File and Directory Operations

4.10 Linux File and Directory Operations [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `mkdir -p` • `cp -r` • `mv` • `rm` • Production Backend Pitfall & Guardrail (Linux File and Directory Operations) • Hands-On TypeScript / Node.js Implementation (Linux File and Directory Operations) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Linux File

Purpose: Diagnosing a sudden `ENOSPC: no space left on device` production outage in 10 seconds with `df -h` and `du -sh /var/log/*`!

What Topic Is Next → Next Topic 4.11: Linux Permissions and Ownership (chmod & chown)

4.11 Linux Permissions and Ownership (chmod & chown) [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: User/Group/Other bits (`r=4 • W=2 • X=1`) • `chmod 600 ~/.ssh/id_ed25519` • Production Backend Pitfall & Guardrail (Linux Permissions and Ownership (chmod & chown)) • Hands-On TypeScript / Node.js Implementation (Linux Permissions and Ownership (chmod & chown)) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Linux Permissions

Purpose: Locking production `.env` permissions to `0600` owned strictly by the dedicated non-root `nodeapp` Linux user.

What Topic Is Next → Next Topic 4.12: Linux Process Management and Signals

4.12 Linux Process Management and Signals [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `ps aux` • `htop`/`top` • `ss -tulpn` / `lsof -i :3000` (finding what occupies port `3000`!) • `SIGTERM` (`15` • Production Backend Pitfall & Guardrail (Linux Process Management and Signals) • Hands-On TypeScript / Node.js Implementation (Linux Process Management and Signals) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Linux Process Management

Purpose: Resolving `Error: listen EADDRINUSE: address already in use :::3000` and handling `SIGTERM` cleanly in Node.js.

What Topic Is Next → Next Topic 4.13: Linux Environment Variables and Shell Configuration

4.13 Linux Environment Variables and Shell Configuration [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `export KEY=VAL` • Child-process inheritance • `env`/`printenv` • Systemd `EnvironmentFile=/etc/api/.env` • Production Backend Pitfall & Guardrail (Linux Environment Variables and Shell Configuration) • Hands-On TypeScript / Node.js Implementation (Linux Environment Variables and Shell Configuration) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Linux Environment Variables

Purpose: Injecting 12-Factor configuration (`DATABASE_URL`, `REDIS_URL`, `PORT`) cleanly without hardcoding secrets in code.

What Topic Is Next → Next Topic 4.14: SSH Fundamentals and Key-Based Remote Access

4.14 SSH Fundamentals and Key-Based Remote Access [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `ssh-keygen -t ed25519` • `~/.ssh/authorized_keys` • `~/.ssh/config` host aliases • `scp`/`rsync` file transfer • Production Backend Pitfall & Guardrail (SSH Fundamentals and Key-Based Remote Access) • Hands-On TypeScript / Node.js Implementation (SSH Fundamentals and Key-Based Remote Access) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for SSH Fundamentals

Purpose: Connecting securely to staging and production Linux VPS hosts via Ed25519 SSH keys.

What Topic Is Next → Next Topic 4.15: Logs, Piping and Linux Production Troubleshooting

4.15 Logs, Piping and Linux Production Troubleshooting [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `journalctl -u api -f` • `tail -f` • `grep -E` • `jq` JSON log filtering (`cat app.log | jq "select(.level >= 50)"`) • Production Backend Pitfall & Guardrail (Logs, Piping and Linux Production Troubleshooting) • Hands-On TypeScript / Node.js Implementation (Logs, Piping and Linux Production Troubleshooting) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Logs

Purpose: Filtering structured Pino JSON logs with `jq` to isolate all `5xx` errors for a specific `requestId` in seconds!

What Topic Is Next → Next Topic 4.16: Client-Server Request-Response Architecture

4.16 Client-Server Request-Response Architecture [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Stateless HTTP clients (browsers • Mobile apps • Webhook callers • Microservices) -> Reverse Proxy (Nginx) -> Node.js Application -> Data/Cache Stores • Production Backend Pitfall & Guardrail (Client-Server Request-Response Architecture) • Hands-On TypeScript / Node.js Implementation (Client-Server Request-Response Architecture) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Client-Server Request-Response Architecture

Purpose: Tracing end-to-end latency budgets across DNS + TCP + TLS + Nginx + Express + PostgreSQL + Response serialization.

What Topic Is Next → Next Topic 4.17: DNS and Domain Resolution

4.17 DNS and Domain Resolution [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Recursive resolver -> Root -> TLD -> Authoritative nameserver • `A` • `AAAA` • `CNAME` • Production Backend Pitfall & Guardrail (DNS and Domain Resolution) • Hands-On TypeScript / Node.js Implementation (DNS and Domain Resolution) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Domain Resolution

Purpose: Pointing `api.yourdomain.com` (`A`/`AAAA`) to your Nginx reverse proxy and verifying TTL propagation with `dig +short`.

What Topic Is Next → Next Topic 4.18: IP Addresses, Loopback, Subnets and Ports

4.18 IP Addresses, Loopback, Subnets and Ports [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: IPv4 vs IPv6 • `127.0.0.1` (loopback ONLY!) vs `0.0.0.0` (all interfaces — required inside Docker containers!) • Ephemeral ports vs server ports (`80` • `443` • Production Backend Pitfall & Guardrail (IP Addresses, Loopback, Subnets and Ports) • Hands-On TypeScript / Node.js Implementation (IP Addresses, Loopback, Subnets and Ports) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for IP Addresses

Purpose: Binding PostgreSQL/Redis strictly to `127.0.0.1` or private Docker bridge networks while binding containerized Express to `0.0.0.0:3000` behind Nginx.

What Topic Is Next → Next Topic 4.19: TCP vs UDP Fundamentals

4.19 TCP vs UDP Fundamentals [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: TCP 3-Way Handshake (`SYN -> SYN-ACK -> ACK`) • Ordered reliable byte streams • Flow/congestion control • `Keep-Alive` connection reuse vs connectionless low-overhead UDP (DNS • Production Backend Pitfall & Guardrail (TCP vs UDP Fundamentals) • Hands-On TypeScript / Node.js Implementation (TCP vs UDP Fundamentals) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for TCP vs UDP Fundamentals

Purpose: Understanding why PostgreSQL, MongoDB, Redis, and HTTP/1.1–2 rely on persistent pooled TCP sockets.

What Topic Is Next → Next Topic 4.20: HTTP/1.1, HTTP/2 and HTTPS (TLS Termination)

4.20 HTTP/1.1, HTTP/2 and HTTPS (TLS Termination) [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Plaintext HTTP vs TLS 1.3 encrypted HTTPS (confidentiality • Integrity & X.509 server authentication) • HTTP/2 multiplexing • Terminating TLS at Nginx (`443 -> http://127.0.0.1:3000` with `X-Forwarded-Proto`) • Production Backend Pitfall & Guardrail (HTTP/1.1, HTTP/2 and HTTPS (TLS Termination)) • Hands-On TypeScript / Node.js Implementation (HTTP/1.1, HTTP/2 and HTTPS (TLS Termination)) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for HTTP/1.1

Purpose: Configuring `app.set("trust proxy", 1)` in Express behind Nginx so `req.ip` and `Secure` cookies work accurately!

What Topic Is Next → Next Topic 4.21: HTTP Request and Response Wire Structure

4.21 HTTP Request and Response Wire Structure [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Start-line (`POST /api/v1/users HTTP/1.1`) • Headers (`Host` • `Content-Type` • `Accept` • Production Backend Pitfall & Guardrail (HTTP Request and Response Wire Structure) • Hands-On TypeScript / Node.js Implementation (HTTP Request and Response Wire Structure) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for HTTP Request

Purpose: Debugging malformed webhook or multipart requests by reading raw HTTP headers and payload bytes.

What Topic Is Next → Next Topic 4.22: HTTP Methods, Headers and Status Codes

4.22 HTTP Methods, Headers and Status Codes [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Semantics & idempotency of `GET` • `HEAD` • `OPTIONS` • `POST` • Production Backend Pitfall & Guardrail (HTTP Methods, Headers and Status Codes) • Hands-On TypeScript / Node.js Implementation (HTTP Methods, Headers and Status Codes) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for HTTP Methods

Purpose: Returning `401 Unauthorized` when JWT is missing/expired vs `403 Forbidden` when an authenticated user lacks `ADMIN` role!

What Topic Is Next → Next Topic 4.23: Cookies and Server-Side Sessions

4.23 Cookies and Server-Side Sessions [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `Set-Cookie` attributes (`HttpOnly` blocks JS XSS theft! • `Secure` HTTPS-only • `SameSite=Lax|Strict` CSRF defense • `Path` • Production Backend Pitfall & Guardrail (Cookies and Server-Side Sessions) • Hands-On TypeScript / Node.js Implementation (Cookies and Server-Side Sessions) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Cookies

Purpose: Storing `refreshToken` inside an `HttpOnly; Secure; SameSite=Strict; Path=/api/v1/auth/refresh` cookie rather than XSS-vulnerable `localStorage`!

What Topic Is Next → Next Topic 4.24: JSON Serialization, Dates, BigInt and Payload Pitfalls

4.24 JSON Serialization, Dates, BigInt and Payload Pitfalls [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `JSON.stringify` / `JSON.parse` rules (`undefined`/functions/Symbols omitted • `Date` -> ISO-8601 string • `BigInt` throws `TypeError` without custom serializer! • Circular references throw) • Production Backend Pitfall & Guardrail (JSON Serialization, Dates, BigInt and Payload Pitfalls) • Hands-On TypeScript / Node.js Implementation (JSON Serialization, Dates, BigInt and Payload Pitfalls) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for JSON Serialization

Purpose: Serializing PostgreSQL `BIGINT` IDs as strings (`id.toString()`) so JavaScript clients never lose integer precision past `Number.MAX_SAFE_INTEGER` (`2^53 - 1`)!

What Topic Is Next → Next Topic 4.25: REST Architectural Principles

4.25 REST Architectural Principles [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Statelessness • Uniform Interface • Resource identification via URIs • Manipulation through representations • Production Backend Pitfall & Guardrail (REST Architectural Principles) • Hands-On TypeScript / Node.js Implementation (REST Architectural Principles) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for REST Architectural Principles

Purpose: Designing clean resource-oriented APIs (`POST /orders/{id}/cancellations`) instead of RPC verbs (`GET /cancelOrderNow?id=5`)!

What Topic Is Next → Next Topic 4.26: CORS (Cross-Origin Resource Sharing) Fundamentals

4.26 CORS (Cross-Origin Resource Sharing) Fundamentals [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Browser Same-Origin Policy (`scheme + host + port`) • Simple vs Preflight `OPTIONS` requests • `Access-Control-Allow-Origin` (explicit allowlist!) • `Access-Control-Allow-Credentials: true` • Production Backend Pitfall & Guardrail (CORS (Cross-Origin Resource Sharing) Fundamentals) • Hands-On TypeScript / Node.js Implementation (CORS (Cross-Origin Resource Sharing) Fundamentals) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for CORS (Cross-Origin Resource Sharing) Fundamentals

Purpose: Configuring strict origin allowlists so authorized web frontends call your API with cookies while rogue third-party sites are blocked by browsers.

What Topic Is Next → Module 4 Capstone (Project 1: TypeScript-Based Command-Line Employee Management System) → Next Topic 5.1: Introduction to Node.js and Backend Engineering Strengths

Module 4 Hands-On Capstone Project (100 Marks): Project 1: TypeScript-Based Command-Line Employee Management System — Before touching web frameworks or external databases, a strong backend engineer must master clean TypeScript domain modeling, OOP encapsulation, immutable data transformations, input validat...

MODULE 5 OF 24 • STAGE 2 • 25 LIVE + LAB HOURS • 18 TOPICS (144 SUBTOPICS) • 1 CAPSTONE PROJECT (100 MARKS)

Module 5: Node.js Fundamentals and Runtime Architecture

Understand Node.js from the inside out — V8 JIT compilation, libuv non-blocking I/O & threadpool, all 6 phases of the Node.js event loop, nextTick vs microtasks, timers, process lifecycle, Buffer & globals, environment config, CLI args, debugging, and runtime performance.

Objectives: ✓ Explain V8, C++ bindings, `libuv`, OS async I/O (`epoll`/`kqueue`/`IOCP`), and the `libuv` worker threadpool clearly in senior backend interviews. • ✓ Trace all 6 phases of the Node.js event loop and contrast `setImmediate` vs `setTimeout(..., 0)` inside I/O callbacks. • ✓ Manage `process`, `Buffer`, environment variables, CLI flags, exit codes, and Node inspector profiling cleanly. • ✓ Identify and prevent event-loop blocking anti-patterns before they degrade API tail latency.

5.1 Introduction to Node.js and Backend Engineering Strengths [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Why event-driven non-blocking Node.js excels at I/O-intensive REST APIs • API gateways • Real-time WebSockets • Streaming services (• Production Backend Pitfall & Guardrail (Introduction to Node.js and Backend Engineering Strengths) • Hands-On TypeScript / Node.js Implementation (Introduction to Node.js and Backend Engineering Strengths) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Introduction to Node.js

Purpose: Handling `10,000` concurrent idle/streaming HTTP or WebSocket connections inside `< 150 MB` RAM without 10,000 OS threads!

What Topic Is Next → Next Topic 5.2: Installing Node.js LTS, Version Managers and Lockfile Discipline

5.2 Installing Node.js LTS, Version Managers and Lockfile Discipline [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Active LTS releases (Node 22 LTS) • `.nvmrc` / `"engines": { "node": ">=22.0.0" }` • Semantic versioning (`^` vs `~`) • Deterministic `npm ci` using `package-lock.json` • Production Backend Pitfall & Guardrail (Installing Node.js LTS, Version Managers and Lockfile Discipline) • Hands-On TypeScript / Node.js

Implementation (Installing Node.js LTS, Version Managers and Lockfile Discipline) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Installing Node.js LTS

Purpose: Running `npm ci --omit=dev` inside CI and Docker builds for 100% reproducible dependency versions.

What Topic Is Next → Next Topic 5.3: Node.js Layered Runtime Architecture

5.3 Node.js Layered Runtime Architecture [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: 4-layer stack: Your TypeScript/JS App -> Node.js Standard Library (`node:fs` • `node:http`) -> C++ Node Bindings -> `V8` + `libuv` + `llhttp` + `OpenSSL` + `c-ares` + `zlib` • Production Backend Pitfall & Guardrail (Node.js Layered Runtime Architecture) • Hands-On TypeScript / Node.js Implementation (Node.js Layered Runtime Architecture) • Node.js Layered Runtime Architecture — Latency, Throughput & Interview Trade-Offs • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Node.js Layered Runtime Architecture • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards

Purpose: Understanding why `crypto.createHash` and TLS run at native C/OpenSSL speed despite being invoked from JavaScript.

What Topic Is Next → Next Topic 5.4: V8 JavaScript Engine Internals

5.4 V8 JavaScript Engine Internals [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Parsing AST -> `Ignition` bytecode interpreter -> profiling hot functions -> `Sparkplug`/`Maglev`/`TurboFan` optimizing JIT machine code + Hidden Classes (Shapes) & Inline Caches • Production Backend Pitfall & Guardrail (V8 JavaScript Engine Internals) • Hands-On TypeScript / Node.js Implementation (V8 JavaScript Engine Internals) • V8 JavaScript Engine Internals — Latency, Throughput & Interview Trade-Offs • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for V8 JavaScript Engine Internals • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards • Diagnostics, Profiling & Telemetry: structured telemetry, profiler traces & CI checks

Purpose: Initializing object properties in consistent constructor shapes so V8 Inline Caches stay monomorphic and fast.

What Topic Is Next → Next Topic 5.5: libuv and Asynchronous I/O Delegation

5.5 libuv and Asynchronous I/O Delegation [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: How `libuv` routes TCP/HTTP sockets directly to non-blocking OS kernel multiplexers (`epoll` on Linux • `kqueue` on macOS • `IOCP` on Windows) while delegating blocking file/crypto/DNS calls to its C threadpool • Production Backend Pitfall & Guardrail (libuv and Asynchronous I/O Delegation) • Hands-On TypeScript / Node.js Implementation (libuv and Asynchronous I/O Delegation) • libuv and Asynchronous I/O Delegation — Latency, Throughput & Interview Trade-Offs • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for libuv

Purpose: Explaining accurately in interviews why 500 concurrent PostgreSQL TCP queries do NOT consume the 4-thread `libuv` threadpool!

What Topic Is Next → Next Topic 5.6: Single-Threaded JS Execution Model and Event-Loop Safety

5.6 Single-Threaded JS Execution Model and Event-Loop Safety [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Why shared-memory mutex races disappear in single-threaded JS • YET logical async interleaving races (`read balance -> await -> write balance`) • Main-thread RegEx `ReDoS` blocking remain critical! • Production Backend Pitfall & Guardrail (Single-Threaded JS Execution Model and Event-Loop Safety) • Hands-On TypeScript / Node.js Implementation (Single-Threaded JS Execution Model and Event-Loop Safety) • Single-Threaded JS Execution Model and Event-Loop Safety — Latency, Throughput & Interview Trade-Offs • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Single-Threaded JS Execution Model

Purpose: Guarding against catastrophic backtracking `ReDoS` regexes that freeze the single JS thread for 30 seconds on a 50-char string!

What Topic Is Next → Next Topic 5.7: Event-Driven Architecture in Node.js

5.7 Event-Driven Architecture in Node.js [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Observer pattern across Node servers • Sockets • Streams • Domain buses (`server.on("request")` • Production Backend Pitfall & Guardrail (Event-Driven Architecture in Node.js) • Hands-On TypeScript / Node.js Implementation (Event-Driven Architecture in Node.js) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Event-Driven Architecture in Node.js

Purpose: Decoupling post-registration side effects (audit log, welcome email enqueue) cleanly via typed domain events.

What Topic Is Next → Next Topic 5.8: Non-Blocking vs Blocking I/O in Practice

5.8 Non-Blocking vs Blocking I/O in Practice [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `fs.readFileSync` / `crypto.pbkdf2Sync` / `execSync` (acceptable ONLY once before `server.listen()` at cold boot • NEVER inside request handlers!) vs non-blocking `fs/promises` • Production Backend Pitfall & Guardrail (Non-Blocking vs Blocking I/O in Practice) • Hands-On TypeScript / Node.js Implementation (Non-Blocking vs Blocking I/O in Practice) • Non-Blocking vs Blocking I/O in Practice — Latency, Throughput & Interview Trade-Offs • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Non-Blocking vs Blocking I/O in Practice • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards

Purpose: Replacing `fs.readFileSync` template reads inside a route with cached startup loading or `await fs.readFile()`.

What Topic Is Next → Next Topic 5.9: Node.js Event Loop Phases in Detail

5.9 Node.js Event Loop Phases in Detail [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: All 6 libuv phases in order: (1) `timers` (`setTimeout`/`setInterval`) • (2) `pending callbacks` • (3) `idle • Prepare` • Production Backend Pitfall & Guardrail (Node.js Event Loop Phases in Detail) • Hands-On TypeScript / Node.js Implementation (Node.js Event Loop Phases in Detail) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Node.js Event Loop Phases in Detail

Purpose: Understanding why `setImmediate` ALWAYS fires before `setTimeout(..., 0)` when both are scheduled inside an `fs.readFile` I/O (`poll` phase) callback (`poll` is followed...

What Topic Is Next → Next Topic 5.10: Call Stack, Callback Queue and Microtask Queue

5.10 Call Stack, Callback Queue and Microtask Queue [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Between EVERY event-loop phase transition/callback • Node drains `process.nextTick` queue followed by Promise Microtask queue before moving on • Production Backend Pitfall & Guardrail (Call Stack, Callback Queue and Microtask Queue) • Hands-On TypeScript / Node.js Implementation (Call Stack, Callback Queue and Microtask Queue) • Call Stack, Callback Queue and Microtask Queue — Latency, Throughput & Interview Trade-Offs • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Call Stack • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards

Purpose: Preferring `queueMicrotask()` over `process.nextTick()` in modern Node code unless immediate post-constructor event emission is required.

What Topic Is Next → Next Topic 5.11: Timers, Intervals and Cancellation (AbortSignal & unref)

5.11 Timers, Intervals and Cancellation (AbortSignal & unref) [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `setTimeout` • `setInterval` • `clearInterval` • `timers/promises` (`await setTimeout(100 • Production Backend Pitfall & Guardrail (Timers, Intervals and Cancellation (AbortSignal & unref)) • Hands-On TypeScript / Node.js Implementation (Timers, Intervals and Cancellation (AbortSignal & unref)) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Timers

Purpose: Calling `.unref()` on a periodic in-memory cleanup interval so Jest tests and graceful shutdowns exit cleanly without hanging!

What Topic Is Next → Next Topic 5.12: Node.js Process Lifecycle, Signals and Uncaught Errors

5.12 Node.js Process Lifecycle, Signals and Uncaught Errors [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `beforeExit` vs `exit` (sync only!) • `SIGTERM` / `SIGINT` graceful handlers • `uncaughtException` & `unhandledRejection` (log fatal error • Flush telemetry • Production Backend Pitfall & Guardrail (Node.js Process Lifecycle, Signals and Uncaught Errors) • Hands-On TypeScript / Node.js Implementation (Node.js Process Lifecycle, Signals and Uncaught Errors) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Node.js Process Lifecycle

Purpose: Logging structured fatal context on `uncaughtException` and exiting cleanly rather than limping along with corrupted state.

What Topic Is Next → Next Topic 5.13: Global Objects and Built-In Web-Standard Globals

5.13 Global Objects and Built-In Web-Standard Globals [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Node 22 built-in standards: global `fetch` • `Headers` • `Request` • `Response` • Production Backend Pitfall & Guardrail (Global Objects and Built-In Web-Standard Globals) • Hands-On TypeScript / Node.js Implementation (Global Objects and Built-In Web-Standard Globals) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Global Objects

Purpose: Calling external REST webhooks with native Node `fetch(..., { signal: AbortSignal.timeout(5000) })` with zero extra npm dependencies!

What Topic Is Next → Next Topic 5.14: process, Buffer and Global Hygiene

5.14 process, Buffer and Global Hygiene [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `process.cwd()` • `process.hrtime.bigint()` nanosecond timers • `process.memoryUsage()` • `Buffer.from(str • Production Backend Pitfall & Guardrail (process, Buffer and Global Hygiene) • Hands-On TypeScript / Node.js Implementation (process, Buffer and Global Hygiene) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for process

Purpose: Measuring exact sub-millisecond DB query latency using `process.hrtime.bigint()`.

What Topic Is Next → Next Topic 5.15: Environment Variables and Native --env-file

5.15 Environment Variables and Native --env-file [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Node 20.6+/22 native `node --env-file=.env dist/server.js` • Why all `process.env.*` values are `string | undefined` (`process.env.DEBUG = "false"` is TRUTHY as a string!) • Typed startup parsing • Production Backend Pitfall & Guardrail (Environment Variables and Native --env-file) • Hands-On TypeScript / Node.js Implementation (Environment Variables and Native --env-file) • Environment Variables and Native --env-file — Latency, Throughput & Interview Trade-Offs • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Environment Variables

Purpose: Parsing `ENABLE_CACHE === "true"` explicitly and crashing at boot if `JWT_SECRET` is `< 32` characters.

What Topic Is Next → Next Topic 5.16: Command-Line Arguments (process.argv & util.parseArgs)

5.16 Command-Line Arguments (process.argv & util.parseArgs) [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Native `node:util` `parseArgs({ options • AllowPositionals })` building zero-dependency typed administrative/migration CLI scripts • Production Backend Pitfall & Guardrail (Command-Line Arguments (process.argv & util.parseArgs)) • Hands-On TypeScript / Node.js Implementation (Command-Line Arguments (process.argv & util.parseArgs)) • Command-Line Arguments (process.argv & util.parseArgs) — Latency, Throughput & Interview Trade-Offs • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Command-Line Arguments (process.argv & util.parseArg... • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards

Purpose: Building clean admin maintenance CLIs (`npm run seed -- --tenant acme --dry-run`) without heavy flag dependencies.

What Topic Is Next → Next Topic 5.17: Node.js Debugging and Diagnostic Reports

5.17 Node.js Debugging and Diagnostic Reports [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `node --inspect` • VS Code Attach/Launch configs • `node --trace-warnings` • `process.report.writeReport()` diagnostic JSON dumps on fatal crashes • Production Backend Pitfall & Guardrail (Node.js Debugging and Diagnostic Reports) • Hands-On TypeScript / Node.js Implementation (Node.js Debugging and Diagnostic Reports) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Node.js Debugging

Purpose: Finding the exact line emitting a mysterious `DeprecationWarning` or `UnhandledPromiseRejection` via `--trace-warnings`.

What Topic Is Next → Next Topic 5.18: Node.js Performance Fundamentals (Event-Loop Lag Monitoring)

5.18 Node.js Performance Fundamentals (Event-Loop Lag Monitoring) [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `node:perf_hooks` `monitorEventLoopDelay({ resolution: 20 })` tracking `p50`/`p99` event-loop lag in milliseconds • Keeping JSON payloads & sync work bounded • Production Backend Pitfall & Guardrail (Node.js Performance Fundamentals (Event-Loop Lag Monitoring)) • Hands-On TypeScript / Node.js Implementation (Node.js Performance Fundamentals (Event-Loop Lag Monitoring)) • Node.js Performance Fundamentals (Event-Loop Lag Monitoring) — Latency, Throughput & Interview Trade-Offs • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Node.js Performance Fundamentals (Event-Loop Lag Mon... • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards

Purpose: Exposing `eventLoopLagP99Ms` on `/metrics` to alert immediately when CPU-bound handlers degrade server responsiveness!

What Topic Is Next → Module 5 Capstone (Week 5 Lab Build: Raw Node.js HTTP Router, EventEmitter Bus & Crypto Hash CLI) → Next Topic 6.1: CommonJS Module System Internals

Module 5 Hands-On Capstone Project (100 Marks): Week 5 Lab Build: Raw Node.js HTTP Router, EventEmitter Bus & Crypto Hash CLI — Without Express, build a raw `node:http` JSON API router supporting URL path params, graceful `SIGTERM` connection draining, typed `EventEmitter` audit hooks, and non-blocking `node:crypto`...

MODULE 6 OF 24 • STAGE 2 • 25 LIVE + LAB HOURS • 25 TOPICS (200 SUBTOPICS) • 1 CAPSTONE PROJECT (100 MARKS)

Module 6: Node.js Core Modules and Advanced Programming

Master the Node.js standard library without unnecessary npm bloat — CommonJS/ESM resolution & caching, fs/path/os/http/events/crypto/url, Buffers, Readable/Writable/Transform Streams with pipeline() and backpressure, large-file processing, child_process, worker_threads, cluster, UV_THREADPOOL_SIZE, and graceful shutdown.

Objectives: ✓ Use `node:` prefixed core modules (`fs/promises`, `path`, `crypto`, `events`, `http`, `url`) idiomatically and securely. • ✓ Process multi-gigabyte files and transforms in constant `O(1)` memory using Streams, `pipeline()`, and proper backpressure handling. • ✓ Scale CPU-bound and multi-core workloads using `worker_threads`, `child_process.execFile`/`spawn`, `cluster`, and `UV_THREADPOOL_SIZE`. • ✓ Implement zero-dropped-request graceful process shutdown draining HTTP keep-alive sockets and database pools.

6.1 CommonJS Module System Internals [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: How Node wraps CJS files `(function(exports • Require • Module • __filename • Production Backend Pitfall & Guardrail (CommonJS Module System Internals) • Hands-On TypeScript / Node.js Implementation (CommonJS Module System Internals) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for CommonJS Module System Internals

Purpose: Maintaining and interoperating cleanly with existing CommonJS enterprise packages.

What Topic Is Next → Next Topic 6.2: ES Module System in Node.js

6.2 ES Module System in Node.js [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `"type": "module"` in `package.json` • `.mjs`/`.cjs` explicit extensions • Asynchronous 3-phase loading (`Construction -> Instantiation -> Evaluation`) • Top-Level `await` • Production Backend Pitfall & Guardrail (ES Module System in Node.js) • Hands-On TypeScript / Node.js Implementation (ES Module System in Node.js) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for ES Module System in Node.js

Purpose: Validating config and warming connections cleanly at module startup using native ESM.

What Topic Is Next → Next Topic 6.3: Module Exports, Imports and Subpath Exports

6.3 Module Exports, Imports and Subpath Exports [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Named exports (preferred for refactoring & explicit contracts) vs default exports • `package.json` `"exports"` encapsulating internal private files • Production Backend Pitfall & Guardrail (Module Exports, Imports and Subpath Exports) • Hands-On TypeScript / Node.js Implementation (Module Exports, Imports and Subpath Exports) • Module Exports, Imports and Subpath Exports — Latency, Throughput & Interview Trade-Offs • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Module Exports • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards

Purpose: Enforcing clean public package boundaries in monorepos via `package.json` `"exports"`.

What Topic Is Next → Next Topic 6.4: Module Resolution and Singleton Caching

6.4 Module Resolution and Singleton Caching [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: How Node caches evaluated modules by resolved real file path (case-sensitivity trap: `./User.js` vs `./user.js` on macOS/Windows loads TWICE • Then crashes on Linux!!) • Production Backend Pitfall & Guardrail (Module Resolution and Singleton Caching) • Hands-On TypeScript / Node.js Implementation (Module Resolution and Singleton Caching) • Module Resolution and Singleton Caching — Latency, Throughput & Interview Trade-Offs • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Module Resolution • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards

Purpose: Sharing a singleton `PrismaClient` / `Logger` cleanly across imports while enforcing `forceConsistentCasingInFileNames: true`!

What Topic Is Next → Next Topic 6.5: File System Module (node:fs/promises)

6.5 File System Module (node:fs/promises) [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `readFile` • `writeFile` • Atomic temp-write + `rename` • `mkdir({ recursive: true })` • Production Backend Pitfall & Guardrail (File System Module (node:fs/promises)) • Hands-On TypeScript / Node.js Implementation (File System Module (node:fs/promises)) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for File System Module (node:fs/promises)

Purpose: Writing crash-safe atomic JSON state files (`writeFile(tmp) -> rename(tmp, target)`) and streaming large directories.

What Topic Is Next → Next Topic 6.6: Path Module and Directory Traversal Defense (node:path)

6.6 Path Module and Directory Traversal Defense (node:path) [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `path.join` • `path.resolve` • `path.basename` • `path.extname` • Production Backend Pitfall & Guardrail (Path Module and Directory Traversal Defense (node:path)) • Hands-On TypeScript / Node.js Implementation (Path Module and Directory Traversal Defense (node:path)) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Path Module

Purpose: Blocking `GET /downloads?file=../../.env` path-traversal attacks deterministically!

What Topic Is Next → Next Topic 6.7: OS Module Diagnostics (node:os)

6.7 OS Module Diagnostics (node:os) [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `os.availableParallelism()` (cgroup/container-aware CPU count in Node 19.4+ — better than `os.cpus().length` inside Docker!) • `os.freemem()` • `os.totalmem()` • `os.loadavg()` • Production Backend Pitfall & Guardrail (OS Module Diagnostics (node:os)) • Hands-On TypeScript / Node.js Implementation (OS Module Diagnostics (node:os)) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for OS Module Diagnostics (node:os)

Purpose: Sizing `cluster` / PM2 worker count inside CPU-limited Docker containers accurately via `os.availableParallelism()`!

What Topic Is Next → Next Topic 6.8: Raw HTTP and HTTPS Modules (node:http / node:https)

6.8 Raw HTTP and HTTPS Modules (node:http / node:https) [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Building a framework-free `http.createServer((req • Res) => ...)` understanding `IncomingMessage` (Readable stream!) • `ServerResponse` (Writable stream!) + `headersTimeout` / `requestTimeout` Slowloris protection! • Production Backend Pitfall & Guardrail (Raw HTTP and HTTPS Modules (node:http / node:https)) • Hands-On TypeScript / Node.js Implementation (Raw HTTP and HTTPS Modules (node:http / node:https)) • Raw HTTP and HTTPS Modules (node:http / node:https) — Latency, Throughput & Interview Trade-Offs • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Raw HTTP

Purpose: Seeing exactly what Express wraps under the hood: `req` is simply a Node `Readable` stream of body chunks!

What Topic Is Next → Next Topic 6.9: Events and Typed EventEmitter (node:events)

6.9 Events and Typed EventEmitter (node:events) [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `on` • `once` • `off` • `emit` (synchronous listener invocation!) • Production Backend Pitfall & Guardrail (Events and Typed EventEmitter (node:events)) • Hands-On TypeScript / Node.js Implementation (Events and Typed EventEmitter (node:events)) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Events

Purpose: Awaiting `await once(server, "listening")` cleanly during integration test startup.

What Topic Is Next → Next Topic 6.10: Crypto Module (node:crypto)

6.10 Crypto Module (node:crypto) [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `randomBytes` • `randomUUID` • `createHash("sha256")` • `createHmac("sha256" • Production Backend Pitfall & Guardrail (Crypto Module (node:crypto)) • Hands-On TypeScript / Node.js Implementation (Crypto Module (node:crypto)) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Crypto Module (node:crypto)

Purpose: Verifying Razorpay/Stripe payment webhook HMAC-SHA256 signatures in constant time with `crypto.timingSafeEqual`!

What Topic Is Next → Next Topic 6.11: WHATWG URL Module (node:url)

6.11 WHATWG URL Module (node:url) [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `new URL(input • Base)` • Safe hostname/protocol validation (`url.protocol === "https:"`) • Automatic encoding • Production Backend Pitfall & Guardrail (WHATWG URL Module (node:url)) • Hands-On TypeScript / Node.js Implementation (WHATWG URL Module (node:url)) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for WHATWG URL Module (node:url)

Purpose: Validating webhook callback URLs (`protocol === "https:"` and allowlisted hostname) before outbound fetches.

What Topic Is Next → Next Topic 6.12: Query Strings and URLSearchParams

6.12 Query Strings and URLSearchParams [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Building • Parsing query parameters via standard `URLSearchParams` (`append` repeated keys • `getAll` • Automatic UTF-8 percent-encoding) • Production Backend Pitfall & Guardrail (Query Strings and URLSearchParams) • Hands-On TypeScript / Node.js Implementation (Query Strings and URLSearchParams) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Query Strings

Purpose: Constructing clean outbound pagination/filter query strings without manual `&` / `encodeURIComponent` concatenation bugs.

What Topic Is Next → Next Topic 6.13: Buffer and Binary Data

6.13 Buffer and Binary Data [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Fixed-size raw byte allocations outside V8 JS string heap (`Uint8Array` subclass) • `Buffer.from` • `Buffer.alloc` (zero-filled safe!) vs `Buffer.allocUnsafe` (uninitialized memory leak hazard!) • `utf8` / `hex` / `base64url` • Production Backend Pitfall & Guardrail (Buffer and Binary Data) • Hands-On TypeScript / Node.js Implementation (Buffer and Binary Data) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Buffer

Purpose: Encoding JWT headers/payloads with `Buffer.from(json).toString("base64url")` and inspecting magic file bytes (`%PDF-`).

What Topic Is Next → Next Topic 6.14: Streams Architecture Overview

6.14 Streams Architecture Overview [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Processing unbounded data chunk-by-chunk (`default highWaterMark = 64 KB` for `fs` streams • `16 KB` general • `16` objects in `objectMode`) instead of buffering entire files in RAM • Production Backend Pitfall & Guardrail (Streams Architecture Overview) • Hands-On TypeScript / Node.js Implementation (Streams Architecture Overview) • Streams Architecture Overview — Latency, Throughput & Interview Trade-Offs • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Streams Architecture Overview

Purpose: Exporting 1,000,000 PostgreSQL audit rows directly into an HTTP CSV download response in `< 40 MB` constant RAM!

What Topic Is Next → Next Topic 6.15: Readable Streams and Async Iteration

6.15 Readable Streams and Async Iteration [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Flowing vs Paused modes • `Readable.from(asyncGenerator)` • Modern `for await (const chunk of readable)` clean async iteration • Production Backend Pitfall & Guardrail (Readable Streams and Async Iteration) • Hands-On TypeScript / Node.js Implementation (Readable Streams and Async Iteration) • Readable Streams and Async Iteration — Latency, Throughput & Interview Trade-Offs • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Readable Streams

Purpose: Turning an async paginated database cursor generator into a standard Node `Readable.from(cursorGen())` stream!

What Topic Is Next → Next Topic 6.16: Writable Streams

6.16 Writable Streams [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `writable.write(chunk)` boolean return (`false` means internal buffer reached `highWaterMark` — wait for `"drain"`!) • `writable.end()` • `"finish"` vs `"close"` • Production Backend Pitfall & Guardrail (Writable Streams) • Hands-On TypeScript / Node.js Implementation (Writable Streams) • Writable Streams — Latency, Throughput & Interview Trade-Offs • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Writable Streams

Purpose: Streaming generated reports to disk or HTTP `ServerResponse` safely.

What Topic Is Next → Next Topic 6.17: Duplex and Transform Streams

6.17 Duplex and Transform Streams [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `Duplex` (independent read/write sides • E.g. TCP `net.Socket`) vs `Transform` (output computed from input • E.g. `zlib.createGzip()` • `crypto.createCipheriv` • Production Backend Pitfall & Guardrail (Duplex and Transform Streams) • Hands-On TypeScript / Node.js Implementation (Duplex and Transform Streams) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Duplex

Purpose: Compressing responses on the fly (`createGzip`) and transforming DB objects into CSV lines via a `Transform` stream.

What Topic Is Next → Next Topic 6.18: Piping and stream/promises pipeline()

6.18 Piping and stream/promises pipeline() [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Why legacy `a.pipe(b).pipe(c)` LEAKS file descriptors when `b` or `c` errors/disconnects vs **`await pipeline(a • C)`** (`node:stream/promises`) which destroys ALL streams cleanly on any error or client disconnect! • Production Backend Pitfall & Guardrail (Piping and stream/promises pipeline()) • Hands-On TypeScript / Node.js Implementation (Piping and stream/promises pipeline()) • Piping and stream/promises pipeline() — Latency, Throughput & Interview Trade-Offs • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Piping • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards

Purpose: Using `await pipeline(s3Stream, createGzip(), res)` in Express downloads so mid-download browser tab closes never leak sockets!

What Topic Is Next → Next Topic 6.19: Backpressure Mechanics in Depth

6.19 Backpressure Mechanics in Depth [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: How `pipeline()` automatically pauses a fast `500 MB/s` disk/DB `Readable` whenever a slow `2 MB/s` mobile 4G HTTP `Writable` fills its `highWaterMark` • Resuming on `"drain"` • Production Backend Pitfall & Guardrail (Backpressure Mechanics in Depth) • Hands-On TypeScript / Node.js Implementation (Backpressure Mechanics in Depth) • Backpressure Mechanics in Depth — Latency, Throughput & Interview Trade-Offs • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Backpressure Mechanics in Depth • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards

Purpose: Serving 500 concurrent slow-network file downloads without RAM growing past a few megabytes!

What Topic Is Next → Next Topic 6.20: File Streaming and Multi-GB Large-File Processing

6.20 File Streaming and Multi-GB Large-File Processing [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Combining `createReadStream` • `readline.createInterface({ crlfDelay: Infinity })` • Streaming SHA-256 checksums • `createGzip` for large CSV/log ingestion • Production Backend Pitfall & Guardrail (File Streaming and Multi-GB Large-File Processing) • Hands-On TypeScript / Node.js Implementation (File Streaming and Multi-GB Large-File Processing) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for File Streaming

Purpose: Ingesting a `2 million`-row product CSV import while computing its `SHA-256` digest in a single streaming pass!

What Topic Is Next → Next Topic 6.21: Child Processes (execFile & spawn vs Unsafe exec)

6.21 Child Processes (execFile & spawn vs Unsafe exec) [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `execFile` (no shell spawned — immune to ` • Rm -rf` shell metacharacter injection!) & streaming `spawn` vs why `exec(cmdString)` with user input is a critical RCE vulnerability! • Production Backend Pitfall & Guardrail (Child Processes (execFile & spawn vs Unsafe exec)) • Hands-On TypeScript / Node.js Implementation (Child Processes (execFile & spawn vs Unsafe exec)) • Child Processes (execFile & spawn vs Unsafe exec) — Latency, Throughput & Interview Trade-Offs • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Child Processes (execFile & spawn vs Unsafe exec) • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards

Purpose: Invoking `pg_dump` or `ffmpeg` safely via `spawn("pg_dump", argsArray, { shell: false })` streaming stdout.

What Topic Is Next → Next Topic 6.22: Worker Threads for CPU-Intensive Tasks (node:worker_threads)

6.22 Worker Threads for CPU-Intensive Tasks (node:worker_threads) [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Isolated V8 isolates sharing one OS process (`Worker` • `parentPort` • `workerData` • Transferable `ArrayBuffer`s) • Production Backend Pitfall & Guardrail (Worker Threads for CPU-Intensive Tasks (node:worker_threads)) • Hands-On TypeScript / Node.js Implementation (Worker Threads for CPU-Intensive Tasks (node:worker_threads)) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Worker Threads for CPU-Intensive Tasks (node:worker_...

Purpose: Offloading heavy PDF report generation, CSV parsing, or image resizing to a bounded 4-thread `worker_threads` pool.

What Topic Is Next → Next Topic 6.23: Cluster Module for Multi-Core Scaling (node:cluster)

6.23 Cluster Module for Multi-Core Scaling (node:cluster) [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Primary process forking `os.availableParallelism()` worker processes sharing port `:3000` via OS round-robin accept • Auto-respawning crashed workers with backoff • Vs container horizontal replicas • Production Backend Pitfall & Guardrail (Cluster Module for Multi-Core Scaling (node:cluster)) • Hands-On TypeScript / Node.js Implementation (Cluster Module for Multi-Core Scaling (node:cluster)) • Cluster Module for Multi-Core Scaling (node:cluster) — Latency, Throughput & Interview Trade-Offs • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Cluster Module for Multi-Core Scaling (node:cluster)

Purpose: Utilizing all 8 vCPUs of a bare-metal Linux server via `cluster` or `PM2 -i max`.

What Topic Is Next → Next Topic 6.24: Thread Pools and Concurrency Tuning (UV_THREADPOOL_SIZE)

6.24 Thread Pools and Concurrency Tuning (UV_THREADPOOL_SIZE) [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Tuning `UV_THREADPOOL_SIZE` (default `4` • Max `1024` • MUST be set in env BEFORE first threadpool call!) when heavy `crypto.scrypt` / `bcrypt` / `zlib` / `fs` workloads queue up • Production Backend Pitfall & Guardrail (Thread Pools and Concurrency Tuning (UV_THREADPOOL_SIZE)) • Hands-On TypeScript / Node.js Implementation (Thread Pools and Concurrency Tuning (UV_THREADPOOL_SIZE)) • Thread Pools and Concurrency Tuning (UV_THREADPOOL_SIZE) — Latency, Throughput & Interview Trade-Offs • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Thread Pools

Purpose: Setting `UV_THREADPOOL_SIZE=16` on an Authentication microservice performing heavy concurrent `scrypt`/`argon2` logins.

What Topic Is Next → Next Topic 6.25: Graceful Process Shutdown

6.25 Graceful Process Shutdown [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Complete zero-downtime shutdown sequence on `SIGTERM`: (1) Flip `/readyz` to `503` -> (2) `server.close()` (stop new connections • Finish in-flight HTTP requests) -> (3) Close BullMQ workers -> (4) Disconnect Prisma/Mongoose/Redis -> (5) `process.exit(0)` (with hard `15s` force-exit timer!) • Production Backend Pitfall & Guardrail (Graceful Process Shutdown) • Hands-On TypeScript / Node.js Implementation (Graceful Process Shutdown) • Graceful Process Shutdown — Latency, Throughput & Interview Trade-Offs • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Graceful Process Shutdown • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards

Purpose: Deploying rolling updates under PM2 / Docker / Nginx with `0` `502 Bad Gateway` errors for active shoppers!

What Topic Is Next → Module 6 Capstone (Week 6 Lab Build: Backpressure-Safe 1M-Row CSV Stream Transformer & NDJSON Logger) → Next Topic 7.1: Introduction to Express.js and Minimalist Request Pipeline

Module 6 Hands-On Capstone Project (100 Marks): Week 6 Lab Build: Backpressure-Safe 1M-Row CSV Stream Transformer & NDJSON Logger — Build a constant-RAM (`< 60 MB` RSS) `stream.pipeline()` processor transforming multi-hundred-MB CSV files into validated gzip-compressed `.ndjson.gz` shards with dead-letter row quarantine...

MODULE 7 OF 24 • STAGE 2 • 25 LIVE + LAB HOURS • 26 TOPICS (208 SUBTOPICS) • 1 CAPSTONE PROJECT (100 MARKS)

Module 7: Express.js Complete Backend Development

Build clean, secure, documented HTTP backends with Express.js and TypeScript — routing, typed params/query/body, middleware chains, controllers, centralized 4-arg error handling, async error forwarding, HTTP status semantics, request validation, versioning, pagination/filtering/sorting/search, and OpenAPI/Swagger UI.

Objectives: ✓ Separate `createApp()` (`app.ts`) from `server.listen()` (`server.ts`) so Supertest tests run in-memory without port conflicts. • ✓ Compose application, router, third-party, custom, and 4-argument error-handling middleware in exact execution order. • ✓ Forward rejected async route Promises reliably into centralized RFC-style JSON error envelopes. • ✓ Implement `/api/v1` versioning, pagination, filtering, sorting, search, and interactive Swagger/OpenAPI 3.1 docs.

7.1 Introduction to Express.js and Minimalist Request Pipeline [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: How Express wraps `node:http` with routing tables • `req`/`res` prototype helpers • A linear middleware stack • Production Backend Pitfall & Guardrail (Introduction to Express.js and Minimalist Request Pipeline) • Hands-On TypeScript / Node.js Implementation (Introduction to Express.js and Minimalist Request Pipeline) • Introduction to Express.js and Minimalist Request Pipeline — Latency, Throughput & Interview Trade-Offs • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Introduction to Express.js

Purpose: Building fast, explicit, zero-magic REST APIs where every header, middleware, and query is transparent.

What Topic Is Next → Next Topic 7.2: Express Installation, TypeScript Types and Express 5 Readiness

7.2 Express Installation, TypeScript Types and Express 5 Readiness [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Installing `express` + `@types/express` • Strict `tsconfig.json` • How Express 5 automatically forwards rejected `async` route Promises to error middleware (vs `asyncHandler` wrapper in Express 4!) • Production Backend Pitfall & Guardrail (Express Installation, TypeScript Types and Express 5 Readiness) • Hands-On TypeScript / Node.js Implementation (Express Installation, TypeScript Types and Express 5 Readiness) • Express Installation, TypeScript Types and Express 5 Readiness — Latency, Throughput & Interview Trade-Offs • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Express Installation

Purpose: Knowing BOTH Express 5 native async error forwarding AND Express 4's `asyncHandler` wrapper (since 80% of existing company codebases run Express 4!).

What Topic Is Next → Next Topic 7.3: Creating an HTTP Server Cleanly (app.ts vs server.ts)

7.3 Creating an HTTP Server Cleanly (app.ts vs server.ts) [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Wiring `const server = http.createServer(app)` with explicit `keepAliveTimeout` (`65s` > Nginx/ALB `60s` idle timeout to prevent `502` race conditions!) • `headersTimeout` • Production Backend Pitfall & Guardrail (Creating an HTTP Server Cleanly (app.ts vs server.ts)) • Hands-On TypeScript / Node.js Implementation (Creating an HTTP Server Cleanly (app.ts vs server.ts)) • Creating an HTTP Server Cleanly (app.ts vs server.ts) — Latency, Throughput & Interview Trade-Offs • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Creating an HTTP Server Cleanly (app.ts vs server.ts... • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards

Purpose: Eliminating intermittent `502 Bad Gateway` errors behind Nginx/load balancers caused by Node closing keep-alive sockets at `5s` right as Nginx reuses them!

What Topic Is Next → Next Topic 7.4: Scalable Express Application Folder Structure

7.4 Scalable Express Application Folder Structure [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Feature-first modular layout: `src/modules/employees/{employee.routes.ts • Employee.controller.ts • Employee.service.ts • Employee.repository.ts • Production Backend Pitfall & Guardrail (Scalable Express Application Folder Structure) • Hands-On TypeScript / Node.js Implementation (Scalable Express Application Folder Structure) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Scalable Express Application Folder Structure

Purpose: Scaling cleanly from 5 endpoints to 150 endpoints without touching unrelated modules.

What Topic Is Next → Next Topic 7.5: Routing and Route Parameters (req.params)

7.5 Routing and Route Parameters (req.params) [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Route methods (`get` • `post` • `put` • `patch` • Production Backend Pitfall & Guardrail (Routing and Route Parameters (req.params)) • Hands-On TypeScript / Node.js Implementation (Routing and Route Parameters (req.params)) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Routing

Purpose: Preventing `GET /employees/export` from being captured by `/employees/:id` with `req.params.id === "export"`!

What Topic Is Next → Next Topic 7.6: Query Parameters and Safe Coercion (req.query)

7.6 Query Parameters and Safe Coercion (req.query) [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Why every `req.query.*` arrives as `string | string[] | ParsedQs | undefined` (`?role=a&role=b` becomes an ARRAY `["a" • "b"]` — HTTP Parameter Pollution `HPP`!) • Production Backend Pitfall & Guardrail (Query Parameters and Safe Coercion (req.query)) • Hands-On TypeScript / Node.js Implementation (Query Parameters and Safe Coercion (req.query)) • Query Parameters and Safe Coercion (req.query) — Latency, Throughput & Interview Trade-Offs • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Query Parameters • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards

Purpose: Validating `req.query` with a strict schema so `?email[$ne]=x` or `?id=1&id=2` never reaches business logic unvalidated!

What Topic Is Next → Next Topic 7.7: Express Request and Response Objects

7.7 Express Request and Response Objects [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `req.headers` • `req.get("Idempotency-Key")` • `req.ip` • `res.status(201).location(...).json(...)` • Production Backend Pitfall & Guardrail (Express Request and Response Objects) • Hands-On TypeScript / Node.js Implementation (Express Request and Response Objects) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Express Request

Purpose: Returning `res.status(201).location(\`/api/v1/employees/\${emp.id}\`).json({ data: emp })` on resource creation.

What Topic Is Next → Next Topic 7.8: Request Body Parsing and Payload Size Limits

7.8 Request Body Parsing and Payload Size Limits [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `express.json({ limit: "100kb" • Strict: true })` • `express.urlencoded({ extended: false • Limit: "100kb" })` • Production Backend Pitfall & Guardrail (Request Body Parsing and Payload Size Limits) • Hands-On TypeScript / Node.js Implementation (Request Body Parsing and Payload Size Limits) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Request Body Parsing

Purpose: Blocking `50 MB` JSON DoS payloads with `{ limit: "100kb" }` (`413 Payload Too Large`) and returning clean `400 INVALID_JSON` on broken JSON.

What Topic Is Next → Next Topic 7.9: Middleware Architecture and Execution Order

7.9 Middleware Architecture and Execution Order [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Exact production middleware order: (1) `requestId` + `pino-http` logger -> (2) `helmet` + `cors` -> (3) Rate limiter -> (4) Body parsers -> (5) `/api/v1` Routers -> (6) `404` catch-all -> (7) 4-arg Global Error Handler! • Production Backend Pitfall & Guardrail (Middleware Architecture and Execution Order) • Hands-On TypeScript / Node.js Implementation (Middleware Architecture and Execution Order) • Middleware Architecture and Execution Order — Latency, Throughput & Interview Trade-Offs • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Middleware Architecture • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards • Diagnostics, Profiling & Telemetry: structured telemetry, profiler traces & CI checks

Purpose: Ensuring even `413 Payload Too Large` and `429 Rate Limit` responses carry `X-Request-Id` and security headers!

What Topic Is Next → Next Topic 7.10: Application-Level Middleware (app.use)

7.10 Application-Level Middleware (app.use) [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Cross-cutting concerns (`X-Request-Id` correlation header propagation/generation • Response-time header • Maintenance gate) • Production Backend Pitfall & Guardrail (Application-Level Middleware (app.use)) • Hands-On TypeScript / Node.js Implementation (Application-Level Middleware (app.use)) • Application-Level Middleware (app.use) — Latency, Throughput & Interview Trade-Offs • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Application-Level Middleware (app.use)

Purpose: Accepting incoming upstream `X-Request-Id` (if valid UUID/safe string) or generating `crypto.randomUUID()` and echoing it in `res.setHeader("X-Request-Id", id)`.

What Topic Is Next → Next Topic 7.11: Router-Level Middleware (express.Router)

7.11 Router-Level Middleware (express.Router) [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Modular sub-routers • `Router({ mergeParams: true })` for nested `/departments/:deptId/employees` • Scoping `authenticate` + `authorize("ADMIN")` to protected router trees • Production Backend Pitfall & Guardrail (Router-Level Middleware (express.Router)) • Hands-On TypeScript / Node.js Implementation (Router-Level Middleware (express.Router)) • Router-Level Middleware (express.Router) — Latency, Throughput & Interview Trade-Offs • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Router-Level Middleware (express.Router)

Purpose: Mounting `adminRouter.use(authenticate, requireRole("ADMIN"))` once so zero admin routes can accidentally forget auth!

What Topic Is Next → Next Topic 7.12: Built-In Express Middleware

7.12 Built-In Express Middleware [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `express.json()` • `express.urlencoded()` • `express.raw({ type: "application/json" })` (essential for Stripe/Razorpay webhook HMAC verification!) • Why pure backend APIs rarely serve `express.static` • Production Backend Pitfall & Guardrail (Built-In Express Middleware) • Hands-On TypeScript / Node.js Implementation (Built-In Express Middleware) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Built-In Express Middleware

Purpose: Using `express.raw({ type: "application/json" })` on `/webhooks/payments` BEFORE `express.json()` so exact raw bytes match the gateway HMAC signature!

What Topic Is Next → Next Topic 7.13: Essential Third-Party Middleware

7.13 Essential Third-Party Middleware [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `helmet` (security headers) • `cors` (explicit allowlist) • `compression` (gzip/brotli above threshold) • `express-rate-limit` • Production Backend Pitfall & Guardrail (Essential Third-Party Middleware) • Hands-On TypeScript / Node.js Implementation (Essential Third-Party Middleware) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Essential Third-Party Middleware

Purpose: Hardening headers and structured request logging in 10 lines of battle-tested middleware.

What Topic Is Next → Next Topic 7.14: Authoring Custom Middleware

7.14 Authoring Custom Middleware [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Parameterized middleware factories (`validateBody(schema)` • `requirePermission("employees:write")` • `cacheResponse(ttlSeconds)`) • Production Backend Pitfall & Guardrail (Authoring Custom Middleware) • Hands-On TypeScript / Node.js Implementation (Authoring Custom Middleware) • Authoring Custom Middleware — Latency, Throughput & Interview Trade-Offs • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Authoring Custom Middleware

Purpose: Replacing repetitive `if (!valid) return res.status(400)...` boilerplate inside 40 controllers with declarative `validate({ body: CreateEmpSchema })`!

What Topic Is Next → Next Topic 7.15: Route Handlers and Transport Layer Discipline

7.15 Route Handlers and Transport Layer Discipline [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: What belongs in an Express handler (parse/extract HTTP -> call Service -> map HTTP status & DTO) vs what NEVER belongs in a handler (SQL/Prisma queries • Password hashing • Business math!) • Production Backend Pitfall & Guardrail (Route Handlers and Transport Layer Discipline) • Hands-On TypeScript / Node.js Implementation (Route Handlers and Transport Layer Discipline) • Route Handlers and Transport Layer Discipline — Latency, Throughput & Interview Trade-Offs • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Route Handlers

Purpose: Keeping every controller method `< 20` lines so business logic can be reused from HTTP, BullMQ workers, and CLI scripts!

What Topic Is Next → Next Topic 7.16: Typed Controllers in TypeScript

7.16 Typed Controllers in TypeScript [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Class controllers vs factory object controllers (`createEmployeeController(service)`) avoiding `this` binding pitfalls completely! • Production Backend Pitfall & Guardrail (Typed Controllers in TypeScript) • Hands-On TypeScript / Node.js Implementation (Typed Controllers in TypeScript) • Typed Controllers in TypeScript — Latency, Throughput & Interview Trade-Offs • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Typed Controllers in TypeScript • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards • Diagnostics, Profiling & Telemetry: structured telemetry, profiler traces & CI checks

Purpose: Using closure-based `createEmployeeController(employeeService)` so route registration `router.post("/", ctrl.create)` never loses `this`!

What Topic Is Next → Next Topic 7.17: Error-Handling Middleware (4-Argument Signature)

7.17 Error-Handling Middleware (4-Argument Signature) [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Why Express identifies error middleware STRICTLY by `fn.length === 4` (`(err • Req • Res • Next)` — removing unused `_next` parameter turns it into a normal 3-arg middleware • Production Backend Pitfall & Guardrail (Error-Handling Middleware (4-Argument Signature)) • Hands-On TypeScript / Node.js Implementation (Error-Handling Middleware (4-Argument Signature)) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Error-Handling Middleware (4-Argument Signature)

Purpose: Funneling 100% of operational and unexpected errors through one central error translator.

What Topic Is Next → Next Topic 7.18: Centralized Exception Hierarchy

7.18 Centralized Exception Hierarchy [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `AppError(message • StatusCode • ErrorCode • IsOperational • Production Backend Pitfall & Guardrail (Centralized Exception Hierarchy) • Hands-On TypeScript / Node.js Implementation (Centralized Exception Hierarchy) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-

blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Centralized Exception Hierarchy

Purpose: Throwing `throw new NotFoundError("Employee", id)` inside `EmployeeService` without `EmployeeService` importing `express`!

What Topic Is Next → Next Topic 7.19: Asynchronous Error Handling End-to-End

7.19 Asynchronous Error Handling End-to-End [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Ensuring every async middleware • Route handler • Event callback • Stream pipeline forwards errors to `next(err)` without hanging sockets or crashing the worker • Production Backend Pitfall & Guardrail (Asynchronous Error Handling End-to-End) • Hands-On TypeScript / Node.js Implementation (Asynchronous Error Handling End-to-End) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Asynchronous Error Handling End-to-End

Purpose: Returning deterministic `503` / `500` JSON within milliseconds when a downstream database throws.

What Topic Is Next → Next Topic 7.20: HTTP Status Codes in Express APIs

7.20 HTTP Status Codes in Express APIs [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Consistent mapping: `POST` create -> `201` • `GET`/`PATCH` -> `200` • `DELETE` -> `204` • Async job accepted -> `202` • Production Backend Pitfall & Guardrail (HTTP Status Codes in Express APIs) • Hands-On TypeScript / Node.js Implementation (HTTP Status Codes in Express APIs) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for HTTP Status Codes in Express APIs

Purpose: Allowing frontend/mobile SDKs and API gateways to handle retries and UI states deterministically by status code.

What Topic Is Next → Next Topic 7.21: Standardized JSON Response Envelopes

7.21 Standardized JSON Response Envelopes [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Consistent `{ success: true • Data • Meta? }` • RFC 9457 Problem Details `{ type • Production Backend Pitfall & Guardrail (Standardized JSON Response Envelopes) • Hands-On TypeScript / Node.js Implementation (Standardized JSON Response Envelopes) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Standardized JSON Response Envelopes

Purpose: Giving QA and client developers one predictable error contract including `requestId` for instant log lookup.

What Topic Is Next → Next Topic 7.22: Request Validation Architecture

7.22 Request Validation Architecture [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Validating `params` • `query` • `body` BEFORE controllers run • Stripping unknown fields (preventing mass assignment!) • Production Backend Pitfall & Guardrail (Request Validation Architecture) • Hands-On TypeScript / Node.js Implementation (Request Validation Architecture) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Request Validation Architecture

Purpose: Blocking `{ "email": "not-an-email", "salary": -500, "isAdmin": true }` at the HTTP gate before any service/DB call.

What Topic Is Next → Next Topic 7.23: API Versioning in Express

7.23 API Versioning in Express [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Mounting `app.use("/api/v1" • V1Router)` vs `Accept-Version` header routing • Maintaining `v1` response adapters while introducing breaking schema changes in `/api/v2` • Production Backend Pitfall & Guardrail (API Versioning in Express) • Hands-On TypeScript / Node.js Implementation (API Versioning in Express) • API Versioning in Express — Latency, Throughput & Interview Trade-Offs • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for API Versioning in Express

Purpose: Rolling out `/api/v2/employees` without breaking mobile clients still pinned to `/api/v1/employees`.

What Topic Is Next → Next Topic 7.24: API Pagination Implementation

7.24 API Pagination Implementation [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Implementing bounded `?page=1&limit=20` (`maxLimit = 100`) returning `{ items • Page • Limit • Total • Production Backend Pitfall & Guardrail (API Pagination Implementation) • Hands-On TypeScript / Node.js Implementation (API Pagination Implementation) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for API Pagination Implementation

Purpose: Ensuring no client can ever trigger an unbounded `SELECT *` full-table dump over HTTP.

What Topic Is Next → Next Topic 7.25: Filtering, Sorting and Search Endpoints

7.25 Filtering, Sorting and Search Endpoints [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Whitelisting allowed filter fields (`department` • `status` • `minSalary` • `maxSalary`) • Production Backend Pitfall & Guardrail (Filtering, Sorting and Search Endpoints) • Hands-On TypeScript / Node.js Implementation (Filtering, Sorting and Search Endpoints) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Filtering

Purpose: Powering `GET /api/v1/employees?department=Engineering&minSalary=60000&sortBy=salary&sortOrder=desc&q=ravi`.

What Topic Is Next → Next Topic 7.26: API Documentation Using Swagger and OpenAPI 3.1

7.26 API Documentation Using Swagger and OpenAPI 3.1 [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Authoring OpenAPI 3.1 specs (`paths` • `components.schemas` • `securitySchemes: bearerAuth`) • Serving interactive `/api/docs` via `swagger-ui-express` • Production Backend Pitfall & Guardrail (API Documentation Using Swagger and OpenAPI 3.1) • Hands-On TypeScript / Node.js Implementation (API Documentation Using Swagger and OpenAPI 3.1) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for API Documentation Using Swagger

Purpose: Giving frontend/QA/partner engineers a live interactive "Try it out" OpenAPI spec for every endpoint & error code.

What Topic Is Next → Module 7 Capstone (Week 7 Lab Build: Layered Express.js + TypeScript + Zod Task & Ticket API) → Next Topic 8.1: REST Principles and Resource Modelling

Module 7 Hands-On Capstone Project (100 Marks): Week 7 Lab Build: Layered Express.js + TypeScript + Zod Task & Ticket API — Build a `Controller -> Service -> Repository` Express.js + TypeScript REST API featuring `Zod` `body`/`query`/`params` validation middleware, centralized RFC 9457 Problem Details error handl...

MODULE 8 OF 24 • STAGE 2 • 25 LIVE + LAB HOURS • 23 TOPICS (184 SUBTOPICS) • 1 CAPSTONE PROJECT (100 MARKS)

Module 8: REST API Design and Professional Backend Architecture

Elevate Express routes into clean enterprise backend architecture — resource modeling, idempotency keys, cursor pagination, bulk APIs, Postman automated collections, Controller-Service-Repository layering, constructor Dependency Injection, typed config, Pino structured logging, and Month 2 Project 2.

Objectives: ✓ Design clean, idempotent, evolvable REST resource contracts passing senior API design reviews. • ✓ Architect decoupled `Controller -> Service -> Repository` TypeScript backends using clean constructor Dependency Injection. • ✓ Enforce fail-fast startup environment validation, structured `Pino` JSON logging with `AsyncLocalStorage` correlation IDs, and automated Postman collections. • ✓ Build and defend **Month 2 Project (Project 2): Production-Style Employee Management REST API**.

8.1 REST Principles and Resource Modelling [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Turning imperative business actions (`"promote employee"` • `"deactivate account"` • `"issue refund"`) into clean noun sub-resources (`POST /employees/{id}/promotions` • `POST /orders/{id}/refunds`) • Production Backend Pitfall & Guardrail (REST Principles and Resource Modelling) • Hands-On TypeScript / Node.js Implementation (REST Principles and Resource Modelling) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for REST Principles

Purpose: Keeping audit history first-class because every `Promotion` or `Refund` becomes an addressable resource with its own ID & timestamp!

What Topic Is Next → Next Topic 8.2: REST vs RPC (gRPC / tRPC) Trade-Offs

8.2 REST vs RPC (gRPC / tRPC) Trade-Offs [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: When public/partner/CRUD HTTP resource APIs favor REST + OpenAPI vs when ultra-low-latency internal service-to-service binary calls favor `gRPC` + Protobuf • Production Backend Pitfall & Guardrail (REST vs RPC (gRPC / tRPC) Trade-Offs) • Hands-On TypeScript / Node.js Implementation (REST vs RPC (gRPC / tRPC) Trade-Offs) • REST vs RPC (gRPC / tRPC) Trade-Offs — Latency, Throughput & Interview Trade-Offs • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for REST vs RPC (gRPC / tRPC) Trade-Offs • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards • Diagnostics, Profiling & Telemetry: structured telemetry, profiler traces & CI checks

Purpose: Exposing REST + OpenAPI at the public API Gateway while knowing where internal `gRPC` fits in Month 5 microservices.

What Topic Is Next → Next Topic 8.3: Resource Naming Conventions

8.3 Resource Naming Conventions [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Lowercase kebab-case plural nouns (`/api/v1/purchase-orders` • Never `/getPurchaseOrders` or `/Purchase_Order`) • Consistent `:id` path variables • Query params for filter/sort/page • Production Backend Pitfall & Guardrail (Resource Naming Conventions) • Hands-On TypeScript / Node.js Implementation (Resource Naming Conventions) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Resource Naming Conventions

Purpose: Maintaining a uniform, self-consistent API style guide across 10 microservices.

What Topic Is Next → Next Topic 8.4: HTTP Methods and Idempotency Keys

8.4 HTTP Methods and Idempotency Keys [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Natural idempotency of `GET` • `PUT` • `DELETE` vs non-idempotent `POST` — • Making `POST /orders` or `POST /payments` retry-safe via `Idempotency-Key` header + payload hash! • Production Backend Pitfall & Guardrail (HTTP Methods and Idempotency Keys) • Hands-On TypeScript / Node.js Implementation (HTTP Methods and Idempotency Keys) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for HTTP Methods

Purpose: Preventing double-charging or duplicate employee/order creation when mobile clients retry after a network drop!

What Topic Is Next → Next Topic 8.5: Status Codes and Actionable Error Contracts

8.5 Status Codes and Actionable Error Contracts [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Machine-parsable `error.code` (`EMAIL_ALREADY_EXISTS` • `DEPARTMENT_HAS_ACTIVE_EMPLOYEES`) • Human `message` • `requestId` • Production Backend Pitfall & Guardrail (Status Codes and Actionable Error Contracts) • Hands-On TypeScript / Node.js Implementation (Status Codes and Actionable Error Contracts) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Status Codes

Purpose: Preventing clients from brittle regex-matching English error strings (`if (err.code === "EMAIL_ALREADY_EXISTS")`).

What Topic Is Next → Next Topic 8.6: Request and Response DTO Contracts

8.6 Request and Response DTO Contracts [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Strict separation between `EmployeeCreateRequest` • `EmployeeDbRecord` • `EmployeeResponse` (never exposing internal auto-increment PKs • Soft-delete flags • Production Backend Pitfall & Guardrail (Request and Response DTO Contracts) • Hands-On TypeScript / Node.js Implementation (Request and Response DTO Contracts) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Request

Purpose: Refactoring internal database table columns freely without breaking external API JSON fields!

What Topic Is Next → Next Topic 8.7: API Versioning Strategies in Production

8.7 API Versioning Strategies in Production [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: URI prefix (`/api/v1`) vs Header (`Api-Version: 2026-01-01`) • Shared domain services serving both `v1Controller` • `v2Controller` via thin DTO transformers • Production Backend Pitfall & Guardrail (API Versioning Strategies in Production) • Hands-On TypeScript / Node.js Implementation (API Versioning Strategies in Production) • API Versioning Strategies in Production — Latency, Throughput & Interview Trade-Offs • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for API Versioning Strategies in Production

Purpose: Supporting `v1` and `v2` simultaneously for 6 months without duplicating `EmployeeService` business logic!

What Topic Is Next → Next Topic 8.8: Pagination Strategies: Offset vs Keyset Cursor

8.8 Pagination Strategies: Offset vs Keyset Cursor [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Why `OFFSET 100000 LIMIT 20` scans & discards `100 • 000` rows (`O(offset)` latency + skipped/duplicate rows on concurrent inserts!) vs **Keyset Cursor Pagination** (`WHERE (created_at • Id) < (:cursorTs • :cursorId) ORDER BY created_at DESC • Production Backend Pitfall & Guardrail (Pagination Strategies: Offset vs Keyset Cursor) • Hands-On TypeScript / Node.js Implementation (Pagination Strategies: Offset vs Keyset Cursor) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Pagination Strategies

Purpose: Encoding opaque base64url cursors `{ createdAt, id }` so page 5,000 is just as `2ms` fast as page 1!

What Topic Is Next → Next Topic 8.9: Filtering and Sorting Contract Design

8.9 Filtering and Sorting Contract Design [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Consistent filter query grammar (`status=ACTIVE` • `minSalary=50000` • `joinedAfter=2025-01-01`) • Multi-column sort (`sort=-createdAt • Production Backend Pitfall & Guardrail (Filtering and Sorting Contract Design) • Hands-On TypeScript / Node.js Implementation (Filtering and Sorting Contract Design) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Filtering

Purpose: Preventing unindexed sort columns (`sort=bioText`) from triggering `500 MB` external disk sorts in PostgreSQL/MongoDB!

What Topic Is Next → Next Topic 8.10: Search API Design

8.10 Search API Design [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Prefix/trigram/full-text search vs exact ID/email lookup • Minimum query length (`>= 2` chars) • Highlighting/ranking • Debounced rate limits • Production Backend Pitfall & Guardrail (Search API Design) • Hands-On TypeScript / Node.js Implementation (Search API Design) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Search API Design

Purpose: Supporting instant employee search across `name`, `email`, and `employeeCode` in Project 2.

What Topic Is Next → Next Topic 8.11: Nested Resources and Depth Limits

8.11 Nested Resources and Depth Limits [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Keep nesting `<= 2` levels: `POST /departments/:deptId/employees` & `GET /departments/:deptId/employees` are clear • Whereas individual lookups use flat `GET /employees/:empId`! • Production Backend Pitfall & Guardrail (Nested Resources and Depth Limits) • Hands-On TypeScript / Node.js Implementation (Nested Resources and Depth Limits) • Nested Resources and Depth Limits — Latency, Throughput & Interview Trade-Offs • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Nested Resources • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards

Purpose: Avoiding monstrous `/orgs/:old/branches/:bld/departments/:dId/teams/:tId/employees/:eId` URLs!

What Topic Is Next → Next Topic 8.12: Bulk Operations Design

8.12 Bulk Operations Design [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Bounded batch sizes (`max 100 items` per call!) • All-or-nothing transactional batch (`POST /employees/bulk`) vs partial-success `{ succeeded: [...] • Failed: [{ index • Error }] }` • Production Backend Pitfall & Guardrail (Bulk Operations Design) • Hands-On TypeScript / Node.js Implementation (Bulk Operations Design) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Bulk Operations Design

Purpose: Importing 80 new campus hires in one validated batch call with per-row validation error indices.

What Topic Is Next → Next Topic 8.13: Rate-Limiting Concepts and Standard Headers

8.13 Rate-Limiting Concepts and Standard Headers [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Fixed Window vs Sliding Window vs Token Bucket • Keying by `userId` (authenticated) vs `req.ip` (public/login) • Returning `429 Too Many Requests` + `Retry-After` • Production Backend Pitfall & Guardrail (Rate-Limiting Concepts and Standard Headers) • Hands-On TypeScript / Node.js Implementation (Rate-Limiting Concepts and Standard Headers) • Rate-Limiting Concepts and Standard Headers — Latency, Throughput & Interview Trade-Offs • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Rate-Limiting Concepts

Purpose: Applying strict `5 req / 15 min` limits on `/auth/login` while allowing `120 req / min` on read endpoints.

What Topic Is Next → Next Topic 8.14: API Documentation Standards

8.14 API Documentation Standards [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Documenting authentication schemes • Pagination parameters • Realistic request/response examples • Idempotency headers • Production Backend Pitfall & Guardrail (API Documentation Standards) • Hands-On TypeScript / Node.js Implementation (API Documentation Standards) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for API Documentation Standards

Purpose: Shipping self-service API docs that allow frontend engineers to integrate Project 2 with zero Slack back-and-forth.

What Topic Is Next → Next Topic 8.15: Postman Collections, Environments and Automated Assertions

8.15 Postman Collections, Environments and Automated Assertions [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Exporting version-controlled `postman/collection.json` + `local.environment.json` • Chaining `pm.environment.set("employeeId" • Res.json().data.id)` • Writing `pm.test()` contract checks runnable in CI via Newman/CLI! • Production Backend Pitfall & Guardrail (Postman Collections, Environments and Automated Assertions) • Hands-On TypeScript / Node.js Implementation (Postman Collections, Environments and Automated Assertions) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Postman Collections

Purpose: Submitting a 1-click runnable Postman test collection alongside Project 2 and the Final Capstone.

What Topic Is Next → Next Topic 8.16: Controller-Service-Repository Pattern

8.16 Controller-Service-Repository Pattern [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `Controller` (HTTP status • Headers • DTO mapping) -> `Service` (use-case orchestration • Domain rules • Production Backend Pitfall & Guardrail (Controller-Service-Repository Pattern) • Hands-On TypeScript / Node.js Implementation (Controller-Service-Repository Pattern) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Controller-Service-Repository Pattern

Purpose: Testing 100% of `EmployeeService` business rules in `< 50ms` using an `InMemoryEmployeeRepository` with zero HTTP or DB boot!

What Topic Is Next → Next Topic 8.17: Separation of Concerns

8.17 Separation of Concerns [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Isolating pure domain calculations • External infrastructure adapters (Mail • Storage • Cache) • Production Backend Pitfall & Guardrail (Separation of Concerns) • Hands-On TypeScript / Node.js Implementation (Separation of Concerns) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Separation of Concerns

Purpose: Swapping storage from in-memory JSON (Month 2) to PostgreSQL Prisma + MongoDB Mongoose (Month 3) by replacing ONLY the Repository class!

What Topic Is Next → Next Topic 8.18: Modular Backend Architecture (Bounded Modules)

8.18 Modular Backend Architecture (Bounded Modules) [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Structuring medium-to-large backends as **Modular Monoliths** first: independent domain modules (`employees` • `departments` • `auth`) communicating through public service interfaces rather than reaching into each other's private repositories! • Production Backend Pitfall & Guardrail (Modular Backend Architecture (Bounded Modules)) • Hands-On TypeScript / Node.js Implementation (Modular Backend Architecture (Bounded Modules)) • Modular Backend Architecture (Bounded Modules) — Latency, Throughput & Interview Trade-Offs • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Modular Backend Architecture (Bounded Modules)

Purpose: Keeping module boundaries clean so extracting a microservice in Month 5 is a clean seam rather than a painful untangling.

What Topic Is Next → Next Topic 8.19: Dependency Injection Fundamentals in TypeScript

8.19 Dependency Injection Fundamentals in TypeScript [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Explicit **Composition Root** (`src/container.ts`) wiring `const repo = new ... • Const svc = new EmployeeService(repo • DeptSvc • Logger) • Production Backend Pitfall & Guardrail (Dependency Injection Fundamentals in TypeScript) • Hands-On TypeScript / Node.js Implementation (Dependency Injection Fundamentals in TypeScript) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Dependency Injection Fundamentals in TypeScript

Purpose: Having 100% compile-time verified dependency graphs with zero `reflect-metadata` runtime surprises.

What Topic Is Next → Next Topic 8.20: Configuration Management (12-Factor Fail-Fast Config)

8.20 Configuration Management (12-Factor Fail-Fast Config) [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Validating ALL environment variables at process boot into one frozen `AppConfig` object — crashing immediately in `< 10ms` with clear field errors if `PORT` • `DATABASE_URL` • Or `JWT_SECRET` is invalid! • Production Backend Pitfall & Guardrail (Configuration Management (12-Factor Fail-Fast Config)) • Hands-On TypeScript / Node.js Implementation (Configuration Management (12-Factor Fail-Fast Config)) • Configuration Management (12-Factor Fail-Fast Config) — Latency, Throughput & Interview Trade-Offs • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Configuration Management (12-Factor Fail-Fast Config...

Purpose: Catching a missing `REDIS_URL` at container boot (`CrashLoopBackOff` caught instantly by health check) instead of 3 hours later when the first user triggers password rese...

What Topic Is Next → Next Topic 8.21: Environment-Specific Settings

8.21 Environment-Specific Settings [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Clean behavior differences across `development` (pretty logs • Swagger UI enabled) • `test` (ephemeral DB • Silent/error logs • Production Backend Pitfall & Guardrail (Environment-Specific Settings) • Hands-On TypeScript / Node.js Implementation (Environment-Specific Settings) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Environment-Specific Settings

Purpose: Ensuring `NODE_ENV=production` disables stack-trace error responses and enables V8/Express production optimizations.

What Topic Is Next → Next Topic 8.22: Structured Logging (Pino & AsyncLocalStorage Correlation)

8.22 Structured Logging (Pino & AsyncLocalStorage Correlation) [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Why `console.log` blocks stdout synchronously on large payloads vs ultra-fast asynchronous **`pino`** JSON logging + `AsyncLocalStorage` injecting `requestId` & `userId` into every child log automatically! • Production Backend Pitfall & Guardrail (Structured Logging (Pino & AsyncLocalStorage Correlation)) • Hands-On TypeScript / Node.js Implementation (Structured Logging (Pino & AsyncLocalStorage Correlation)) • Structured Logging (Pino & AsyncLocalStorage Correlation) — Latency, Throughput & Interview Trade-Offs • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Structured Logging (Pino & AsyncLocalStorage Correla... • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards • Diagnostics, Profiling & Telemetry:

structured telemetry, profiler traces & CI checks

Purpose: Tracing `reqId: "9f82..."` across Controller -> Service -> Repository logs in 1 query without passing `reqId` through 10 parameters!

What Topic Is Next → Next Topic 8.23: API Design Reviews and Production Readiness Checklist

8.23 API Design Reviews and Production Readiness Checklist [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: 12-point API review rubric: Naming • Idempotency • Pagination bounds • Whitelisted sort/filter • Production Backend Pitfall & Guardrail (API Design Reviews and Production Readiness Checklist) • Hands-On TypeScript / Node.js Implementation (API Design Reviews and Production Readiness Checklist) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for API Design Reviews

Purpose: Self-reviewing and shipping **Month 2 Project 2: Production-Style Employee Management REST API** cleanly.

What Topic Is Next → Module 8 Capstone (Project 2: Production-Style Employee Management REST API) → Next Topic 9.1: Database Fundamentals: Storage Engines, Durability and Workload Fit

Module 8 Hands-On Capstone Project (100 Marks): Project 2: Production-Style Employee Management REST API — Transform Month 1's domain into a full-fledged, production-structured HTTP service! Build a **Production-Style Employee Management REST API** using **Node.js 22, Express.js, and TypeScript 5...

MODULE 9 OF 24 • STAGE 3 • 25 LIVE + LAB HOURS • 23 TOPICS (184 SUBTOPICS) • 1 CAPSTONE PROJECT (100 MARKS)

Module 9: MongoDB and NoSQL Database Development

Master document-oriented NoSQL engineering with MongoDB — WiredTiger internals, BSON types, CRUD & atomic update operators, cursor pagination, embedded vs referenced schema patterns, ESR compound/TTL/text indexes, multi-stage Aggregation Pipelines ($match, $group, $lookup, $facet), replica-set ACID transactions, and backups.

Objectives: ✓ Choose objectively between MongoDB document modeling (polymorphic product attributes, high-velocity catalogs/events) and PostgreSQL relational tables. • ✓ Model `1:1`, `1:Few` (embedded), `1:Many` / `1:Squillions` (referenced — avoiding unbounded `16 MB` document growth!), and `M:N` relationships cleanly. • ✓ Design `ESR` (`Equality -> Sort -> Range`), Unique, Partial, TTL, and Text indexes verified with `.explain("executionStats")`. • ✓ Author high-performance Aggregation Pipelines (`$match`, `$group`, `$project`, `$sort`, `$lookup`, `$unwind`) and replica-set transactions.

9.1 Database Fundamentals: Storage Engines, Durability and Workload Fit [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: OLTP vs OLAP • Write-ahead journaling • Read/write amplification • Working-set RAM sizing • Production Backend Pitfall & Guardrail (Database Fundamentals: Storage Engines, Durability and Workload Fit) • Hands-On TypeScript / Node.js Implementation (Database Fundamentals: Storage Engines, Durability and Workload Fit) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Database Fundamentals

Purpose: Listing your application's top 10 read/write queries BEFORE designing collections or tables.

What Topic Is Next → Next Topic 9.2: SQL vs NoSQL Architectural Trade-Offs

9.2 SQL vs NoSQL Architectural Trade-Offs [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Relational ACID tables (schema-on-write • Arbitrary ad-hoc joins • Foreign keys) vs Document/Key-Value/Wide-Column stores (schema flexibility • Single-document locality • Production Backend Pitfall & Guardrail (SQL vs NoSQL Architectural Trade-Offs) • Hands-On TypeScript / Node.js Implementation (SQL vs NoSQL Architectural Trade-Offs) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for SQL vs NoSQL Architectural Trade-Offs

Purpose: Storing Electronics (`ramGb`, `screenHz`), Apparel (`size`, `fabric`), and Books (`isbn`, `author`) cleanly in one polymorphic `products` MongoDB collection!

What Topic Is Next → Next Topic 9.3: MongoDB WiredTiger Architecture

9.3 MongoDB WiredTiger Architecture [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: WiredTiger cache (~50% of available RAM) • Document-level concurrency control • Snappy/zstd block compression • `oplog` replication • Production Backend Pitfall & Guardrail (MongoDB WiredTiger Architecture) • Hands-On TypeScript / Node.js Implementation (MongoDB WiredTiger Architecture) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for MongoDB WiredTiger Architecture

Purpose: Sizing RAM so active indexes + hot working set fit inside WiredTiger cache without disk thrashing.

What Topic Is Next → Next Topic 9.4: Databases, Collections and 16 MB BSON Documents

9.4 Databases, Collections and 16 MB BSON Documents [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Dynamic vs JSON-Schema validated collections (`validator: { $jsonSchema: ... }`) • `_id` immutability • The hard `16 MB` BSON document ceiling • Production Backend Pitfall & Guardrail (Databases, Collections and 16 MB BSON Documents) • Hands-On TypeScript / Node.js Implementation (Databases, Collections and 16 MB BSON Documents) • Databases, Collections and 16 MB BSON Documents — Latency, Throughput & Interview Trade-Offs • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Databases

Purpose: Attaching MongoDB `$jsonSchema` collection validation rules as a database-level safety net alongside Mongoose.

What Topic Is Next → Next Topic 9.5: BSON Data Types in Depth

9.5 BSON Data Types in Depth [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: 12-byte `ObjectId` (`4B timestamp + 5B random + 3B counter` — naturally roughly time-ordered!) • `Int32`/`Int64` • `Decimal128` (exact decimal math!) • `ISODate` (`new Date()`) • Production Backend Pitfall & Guardrail (BSON Data Types in Depth) • Hands-On TypeScript / Node.js Implementation (BSON Data Types in Depth) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for BSON Data Types in Depth

Purpose: Using integer paise (`Int64`) or `Decimal128` for prices — NEVER binary `Double` floats!

What Topic Is Next → Next Topic 9.6: MongoDB Local & Docker Installation + mongosh

9.6 MongoDB Local & Docker Installation + mongosh [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Running `mongo:7` in Docker with a single-node replica set (`--replSet rs0` • Required locally to test multi-document transactions & Change Streams!) • `mongosh` • Production Backend Pitfall & Guardrail (MongoDB Local & Docker Installation + mongosh) • Hands-On TypeScript / Node.js Implementation (MongoDB Local & Docker Installation + mongosh) • MongoDB Local & Docker Installation + mongosh — Latency, Throughput & Interview Trade-Offs • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for MongoDB Local

Purpose: Configuring local Docker `mongo:7 --replSet rs0` so transactions work identically in local dev, CI, and production!

What Topic Is Next → Next Topic 9.7: MongoDB Atlas Fundamentals

9.7 MongoDB Atlas Fundamentals [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Replica-set `M0/M10+` clusters • `mongodb+srv://` DNS seedlist connection strings • SCRAM auth • IP/VPC peering allowlists • Production Backend Pitfall & Guardrail (MongoDB Atlas Fundamentals) • Hands-On TypeScript / Node.js Implementation (MongoDB Atlas Fundamentals) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for MongoDB Atlas Fundamentals

Purpose: Understanding `mongodb+srv://` failover across Primary + 2 Secondary replica nodes.

What Topic Is Next → Next Topic 9.8: MongoDB Compass Visual Explain & Schema Profiling

9.8 MongoDB Compass Visual Explain & Schema Profiling [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Inspecting document schema distributions • Index usage hit counts • Visual Explain Plans (`Keys Examined` vs `Docs Examined` ratio!) • Production Backend Pitfall & Guardrail (MongoDB Compass Visual Explain & Schema Profiling) • Hands-On TypeScript / Node.js Implementation (MongoDB Compass Visual Explain & Schema Profiling) • MongoDB Compass Visual Explain & Schema Profiling — Latency, Throughput & Interview Trade-Offs • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for MongoDB Compass Visual Explain

Purpose: Spotting a query scanning `250,000` documents (`totalDocsExamined: 250000`) to return `nReturned: 5` in 2 seconds!

What Topic Is Next → Next Topic 9.9: CRUD Operations and Atomic Update Operators

9.9 CRUD Operations and Atomic Update Operators [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `insertOne` • `insertMany({ ordered: false })` • `find` • `updateOne` (`$set` • Production Backend Pitfall & Guardrail (CRUD Operations and Atomic Update Operators) • Hands-On TypeScript / Node.js Implementation (CRUD Operations and Atomic Update Operators) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for CRUD Operations

Purpose: Atomically decrementing stock `updateOne({ sku, qty: { $gte: n } }, { $inc: { qty: -n } })` — single-document updates in MongoDB are 100% atomic without race conditions!

What Topic Is Next → Next Topic 9.10: Query Operators ($in, $gt/$lte, $elemMatch, $exists)

9.10 Query Operators ($in, $gt/$lte, $elemMatch, $exists) [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Comparison (`$eq` • `$ne` • `$gt` • `$gte` • Production Backend Pitfall & Guardrail (Query Operators ($in, $gt/$lte, $elemMatch, $exists)) • Hands-On TypeScript / Node.js Implementation (Query Operators ($in, $gt/$lte, $elemMatch, $exists)) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Query Operators ($in, $gt/$lte, $elemMatch, $exists)

Purpose: Querying `variants: { $elemMatch: { size: "L", stock: { $gt: 0 } } }` accurately (without `$elemMatch`, size `"L"` out-of-stock + size `"S"` in-stock falsely matches!!).

What Topic Is Next → Next Topic 9.11: Projection and Sorting

9.11 Projection and Sorting [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Inclusion `{ title: 1 • PricePaise: 1 }` vs exclusion `{ internalCost: 0 }` • **Covered Queries** (`totalDocsExamined === 0` when all projected fields live in the index!) • Compound sort direction rules • Production Backend Pitfall & Guardrail (Projection and Sorting) • Hands-On TypeScript / Node.js Implementation (Projection and Sorting) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Projection

Purpose: Projecting only `{ sku: 1, title: 1, pricePaise: 1, thumbnail: 1 }` on catalog grid listings to slash network payload by `80%`.

What Topic Is Next → Next Topic 9.12: Cursor Pagination in MongoDB

9.12 Cursor Pagination in MongoDB [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Why `.skip(50000).limit(20)` walks 50 • 020 index entries vs keyset `{ _id: { $lt: lastObjectId } }` or compound `{ $or: [{ createdAt: { $lt: ts } } • { createdAt: ts • _id: { $lt: id } }] }` • Production Backend Pitfall & Guardrail (Cursor Pagination in MongoDB) • Hands-On TypeScript / Node.js Implementation (Cursor Pagination in MongoDB) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Cursor Pagination in MongoDB

Purpose: Serving infinite-scroll catalog feeds at constant `3ms` latency across millions of products.

What Topic Is Next → Next Topic 9.13: Embedded Documents Pattern

9.13 Embedded Documents Pattern [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Embedding tightly owned • Bounded `1:1` • `1:Few` data (`order.shippingAddress` • `product.dimensions` • Production Backend Pitfall & Guardrail (Embedded Documents Pattern) • Hands-On TypeScript / Node.js Implementation (Embedded Documents Pattern) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Embedded Documents Pattern

Purpose: Snapshotting `shippingAddress` and `lineItems` (`priceAtPurchasePaise`) directly inside an `Order` document so future address/price edits never rewrite historical invoice...

What Topic Is Next → Next Topic 9.14: References, Relationships and Extended Reference Pattern

9.14 References, Relationships and Extended Reference Pattern [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Referencing `categoryId` / `brandId` when cardinality is unbounded (`1:Many` / `1:Squillions` • E.g. • `reviews` referencing `productId`!) + caching `{ id • Name • Production Backend Pitfall & Guardrail (References, Relationships and Extended Reference Pattern) • Hands-On TypeScript / Node.js Implementation (References, Relationships and Extended Reference Pattern) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for References

Purpose: Storing `{ productId: ObjectId(...) }` on each `review` document (instead of `product.reviews[]`) while keeping `{ reviewCount, ratingAvg }` summary counters on `product`...

What Topic Is Next → Next Topic 9.15: One-to-One and One-to-Many Modelling

9.15 One-to-One and One-to-Many Modelling [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Choosing `1:1` embedded (`user.preferences`) vs split collection (large rarely-read `product_raw_audit`) • `1:Few` vs `1:Many` vs `1:Millions` • Production Backend Pitfall & Guardrail (One-to-One and One-to-Many Modelling) • Hands-On TypeScript / Node.js Implementation (One-to-One and One-to-Many Modelling) • One-to-One and One-to-Many Modelling — Latency, Throughput & Interview Trade-Offs • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for One-to-One • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards

Purpose: Splitting rarely accessed `200 KB` changelog history out of the hot `2 KB` `products` working-set document.

What Topic Is Next → Next Topic 9.16: Many-to-Many Modelling in MongoDB

9.16 Many-to-Many Modelling in MongoDB [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Bounded two-way array of IDs (`product.tagIds[]` `<= 20`) with multikey index vs dedicated association collection (`warehouse_stock` with metadata `aisle` • `bin` • `updatedAt`) • Production Backend Pitfall & Guardrail (Many-to-Many Modelling in MongoDB) • Hands-On TypeScript / Node.js Implementation (Many-to-Many Modelling in MongoDB) • Many-to-Many Modelling in MongoDB — Latency, Throughput & Interview Trade-Offs • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Many-to-Many Modelling in MongoDB

Purpose: Using multikey-indexed `categoryIds: ObjectId[]` (`<= 10` categories per product) for instant `{ categoryIds: catId }` lookups.

What Topic Is Next → Next Topic 9.17: Schema Design Principles (Workload-Driven Design)

9.17 Schema Design Principles (Workload-Driven Design) [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: 6 core MongoDB patterns: Embedded • Extended Reference • Subset (keep latest 5 reviews inline + rest in `reviews`!) • Computed (`ratingAvg`) • Production Backend Pitfall & Guardrail (Schema Design Principles (Workload-Driven Design)) • Hands-On TypeScript / Node.js Implementation (Schema Design Principles (Workload-Driven Design)) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Schema Design Principles (Workload-Driven Design)

Purpose: Using `schemaVersion: 2` lazy/background migrations to evolve MongoDB documents with zero downtime.

What Topic Is Next → Next Topic 9.18: Indexes and Compound Indexes (The ESR Rule)

9.18 Indexes and Compound Indexes (The ESR Rule) [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: B-Tree indexes in MongoDB • The golden **ESR Rule** for compound index field order: **1. Equality** (`status: "ACTIVE" • CategoryId: X`) -> **2. Sort** (`pricePaise: 1`) -> **3. Range** (`ratingAvg: { $gte: 4 }`)! • Production Backend Pitfall & Guardrail (Indexes and Compound Indexes (The ESR Rule)) • Hands-On TypeScript / Node.js Implementation (Indexes and Compound Indexes (The ESR Rule)) • Indexes and Compound Indexes (The ESR Rule) — Latency, Throughput & Interview Trade-Offs • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Indexes

Purpose: Ordering `{ tenantId: 1, status: 1, createdAt: -1, pricePaise: 1 }` by `E -> S -> R` so MongoDB avoids in-memory `SORT` stages (`32 MB` RAM sort limit without disk spill)...

What Topic Is Next → Next Topic 9.19: Unique, Partial, TTL and Text Indexes

9.19 Unique, Partial, TTL and Text Indexes [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `{ unique: true }` (`E11000` duplicate guard) • `{ partialFilterExpression: { deletedAt: null } }` (unique active email even with soft deletes!) • `{ expireAfterSeconds: 0 }` TTL cleanup • Weighted `$text` search indexes • Production Backend Pitfall & Guardrail (Unique, Partial, TTL and Text Indexes) • Hands-On TypeScript / Node.js Implementation (Unique, Partial, TTL and Text Indexes) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Unique

Purpose: Auto-deleting expired OTPs, password-reset tokens, and idempotency keys via MongoDB TTL indexes with zero cron code!

What Topic Is Next → Next Topic 9.20: MongoDB Aggregation Pipeline Architecture

9.20 MongoDB Aggregation Pipeline Architecture [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Streaming stage-by-stage transformation pipeline • Pushing `$match` & `$sort` to Stage 1 so B-Tree indexes execute BEFORE heavy `$group`/`$lookup` stages • `allowDiskUse` • Production Backend Pitfall & Guardrail (MongoDB Aggregation Pipeline Architecture) • Hands-On TypeScript / Node.js Implementation (MongoDB Aggregation Pipeline Architecture) • MongoDB Aggregation Pipeline Architecture — Latency, Throughput & Interview Trade-Offs • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for MongoDB Aggregation Pipeline Architecture

Purpose: Computing real-time category sales dashboards, price buckets, and inventory summaries inside MongoDB instead of pulling 500k docs over the network into Node!

What Topic Is Next → Next Topic 9.21: Core Stages: $match, $group, $project and $sort

9.21 Core Stages: $match, $group, $project and $sort [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Filtering (`$match`) • Grouping accumulators (`$sum` • `$avg` • `$min` • Production Backend Pitfall & Guardrail (Core Stages: $match, $group, $project and $sort) • Hands-On TypeScript / Node.js Implementation (Core Stages: $match, $group, $project and $sort) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Core Stages: $match

Purpose: Calculating `{ categoryId, totalSkus, avgPricePaise, outOfStockCount }` in one fast server-side pipeline.

What Topic Is Next → Next Topic 9.22: Relational & Multi-Facet Stages: $lookup, $unwind and $facet

9.22 Relational & Multi-Facet Stages: $lookup, $unwind and $facet [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Correlated/sub-pipeline `$lookup` (always ensure the foreign field is indexed!) • `$unwind` (`preserveNullAndEmptyArrays: true`) • `$facet` returning `{ items • TotalCount • Production Backend Pitfall & Guardrail (Relational & Multi-Facet Stages: $lookup, $unwind and $facet) • Hands-On TypeScript / Node.js Implementation (Relational & Multi-Facet Stages: $lookup, $unwind and $facet) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Relational

Purpose: Powering e-commerce sidebar filter counts (`Brands (12)`, `In Stock (84)`) + paginated items in ONE `$facet` aggregation!

What Topic Is Next → Next Topic 9.23: MongoDB Transactions, Security and Backups

9.23 MongoDB Transactions, Security and Backups [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Multi-document ACID sessions (`session.withTransaction` with automatic transient-transaction retry • `< 60s` max duration) • SCRAM least-privilege DB users • TLS • Production Backend Pitfall & Guardrail (MongoDB Transactions, Security and Backups) • Hands-On TypeScript / Node.js Implementation (MongoDB Transactions, Security and Backups) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for MongoDB Transactions

Purpose: Using `session.withTransaction()` ONLY when atomicity spans 2+ documents/collections while relying on single-document atomicity for 95% of operations.

What Topic Is Next → Module 9 Capstone (Week 9 Lab Build: MongoDB + Mongoose Catalog, Aggregation & ESR Index Benchmark) → Next Topic 10.1: Introduction to Mongoose ODM vs Native MongoDB Driver

Module 9 Hands-On Capstone Project (100 Marks): Week 9 Lab Build: MongoDB + Mongoose Catalog, Aggregation & ESR Index Benchmark — Design a MongoDB 7 + Mongoose 8 product & review backend featuring embedded vs referenced modeling, multi-stage `$match -> $group -> $facet` analytics pipelines, `.lean()` read paths, and `e...

MODULE 10 OF 24 • STAGE 3 • 25 LIVE + LAB HOURS • 22 TOPICS (176 SUBTOPICS) • 1 CAPSTONE PROJECT (100 MARKS)

Module 10: Mongoose ODM and Database Integration

Integrate MongoDB cleanly into TypeScript Node.js backends using Mongoose ODM — strict schemas, custom validators, hooks, lean() read optimization, populate N+1 prevention, sessions/transactions, connection pooling, error mapping, and Repository abstractions.

Objectives: ✓ Define strict, TypeScript-typed Mongoose schemas (`strict: "throw"`, `autoIndex: false` in prod, `sanitizeFilter: true`). • ✓ Avoid classic Mongoose production traps: missing `runValidators: true` on updates, `updateOne` bypassing `pre("save")`, slow hydrated reads without `.lean()`, and N+1 `.populate()` loops. • ✓ Manage connection pools, graceful disconnects, multi-document `startSession()` transactions, and `11000` duplicate-key translation. • ✓ Encapsulate Mongoose models cleanly behind typed `ProductRepository` interfaces.

10.1 Introduction to Mongoose ODM vs Native MongoDB Driver [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: What Mongoose adds (schema enforcement • TypeScript inference • Middleware hooks • Population • Production Backend Pitfall & Guardrail (Introduction to Mongoose ODM vs Native MongoDB Driver) • Hands-On TypeScript / Node.js Implementation (Introduction to Mongoose ODM vs Native MongoDB Driver) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Introduction to Mongoose ODM vs Native MongoDB Drive...

Purpose: Getting structured schema validation on writes + fast `.lean()` POJO performance on reads.

What Topic Is Next → Next Topic 10.2: Connecting Node.js to MongoDB Resiliently

10.2 Connecting Node.js to MongoDB Resiliently [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `mongoose.connect(uri • { maxPoolSize: 20 • ServerSelectionTimeoutMS: 5000 • SocketTimeoutMS: 45000 • Production Backend Pitfall & Guardrail (Connecting Node.js to MongoDB Resiliently) • Hands-On TypeScript / Node.js Implementation (Connecting Node.js to MongoDB Resiliently) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Connecting Node.js to MongoDB Resiliently

Purpose: Failing fast in `5s` (`serverSelectionTimeoutMS: 5000`) when `MONGODB_URI` is unreachable instead of hanging startup for `30s`.

What Topic Is Next → Next Topic 10.3: Mongoose Schema Definition and Strict Mode

10.3 Mongoose Schema Definition and Strict Mode [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `new Schema<IProduct>({ ... } • { strict: "throw" • Timestamps: true • VersionKey: "__v" })` rejecting undeclared fields instead of silently dropping or saving typos • Production Backend Pitfall & Guardrail (Mongoose Schema Definition and Strict Mode) • Hands-On TypeScript / Node.js Implementation (Mongoose Schema Definition and Strict Mode) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Mongoose Schema Definition

Purpose: Catching bugs immediately when a service passes `{ priceInPaise }` instead of `{ pricePaise }` via `strict: "throw"`.

What Topic Is Next → Next Topic 10.4: Models and Hydrated Documents

10.4 Models and Hydrated Documents [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `model<IProduct>("Product" • ProductSchema)` • Model collection pluralization (`collection: "products"` explicit!) • `.isModified()` • Production Backend Pitfall & Guardrail (Models and Hydrated Documents) • Hands-On TypeScript / Node.js Implementation (Models and Hydrated Documents) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Models

Purpose: Normalizing `_id` to string `id` cleanly in `toJSON` / DTO mappers.

What Topic Is Next → Next Topic 10.5: SchemaTypes and Built-In Validation

10.5 SchemaTypes and Built-In Validation [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `String` (`trim` • `lowercase` • `minLength` • `maxLength` • Production Backend Pitfall & Guardrail (SchemaTypes and Built-In Validation) • Hands-On TypeScript / Node.js Implementation (SchemaTypes and Built-In Validation) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for SchemaTypes

Purpose: Enforcing `email: { type: String, required: true, trim: true, lowercase: true }` directly on the schema.

What Topic Is Next → Next Topic 10.6: Required, Conditional and Custom Validators

10.6 Required, Conditional and Custom Validators [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Custom `validate: { validator: (v) => ... • Message: ... }` functions • SKU format checks • Why uniqueness (`unique: true`) is a MongoDB **INDEX** • Production Backend Pitfall & Guardrail (Required, Conditional and Custom Validators) • Hands-On TypeScript / Node.js Implementation (Required, Conditional and Custom Validators) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Required

Purpose: Understanding that `unique: true` throws a MongoServerError `code === 11000` (not `ValidationError`!) so race-condition duplicates are caught atomically by the B-Tree ind...

What Topic Is Next → Next Topic 10.7: Default Values, Immutable Fields and Timestamps

10.7 Default Values, Immutable Fields and Timestamps [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Function defaults (`default: () => randomUUID()` vs static value trap!) • `immutable: true` (`tenantId` • `createdBy` • `sku`) • Production Backend Pitfall & Guardrail (Default Values, Immutable Fields and Timestamps) • Hands-On TypeScript / Node.js Implementation (Default Values, Immutable Fields and Timestamps) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Default Values

Purpose: Marking `tenantId: { type: String, required: true, immutable: true }` so no update bug can ever reassign a document to another tenant!

What Topic Is Next → Next Topic 10.8: Instance Methods, Static Methods and Query Helpers

10.8 Instance Methods, Static Methods and Query Helpers [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Adding domain methods on `schema.methods` (`function` keyword — NEVER arrow `() =>` which loses `this`!) • `schema.statics` • Chainable `schema.query.active()` • Production Backend Pitfall & Guardrail (Instance Methods, Static Methods and Query Helpers) • Hands-On TypeScript / Node.js Implementation (Instance Methods, Static Methods and Query Helpers) • Instance Methods, Static Methods and Query Helpers — Latency, Throughput & Interview Trade-Offs • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Instance Methods

Purpose: Providing `Product.query.active()` while keeping heavy orchestration inside `ProductService`.

What Topic Is Next → Next Topic 10.9: Mongoose Middleware and Hooks Architecture

10.9 Mongoose Middleware and Hooks Architecture [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: 4 distinct middleware types: **Document** (`validate` • `save` • `deleteOne`) • **Query** (`find` • Production Backend Pitfall & Guardrail (Mongoose Middleware and Hooks Architecture) • Hands-On TypeScript / Node.js Implementation (Mongoose Middleware and Hooks Architecture) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Mongoose Middleware

Purpose: Knowing exactly which `this` (`HydratedDocument` vs `Query`) is bound inside each hook category.

What Topic Is Next → Next Topic 10.10: Pre and Post Hooks (and the updateOne Trap)

10.10 Pre and Post Hooks (and the updateOne Trap) [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Why `Model.findOneAndUpdate()` • `Model.updateOne()` trigger **Query** middleware (where `this` is the `Query` • NOT the document!) • NEVER run `pre("save")` document hooks! • Production Backend Pitfall & Guardrail (Pre and Post Hooks (and the updateOne Trap)) • Hands-On TypeScript / Node.js Implementation (Pre and Post Hooks (and the updateOne Trap)) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Post Hooks (and the updateOne Trap)

Purpose: Preventing the classic password-reset bug where `User.updateOne({ _id }, { password })` saves a PLAINTEXT password because `pre("save")` hashing was skipped!!

What Topic Is Next → Next Topic 10.11: CRUD Using Mongoose Safely

10.11 CRUD Using Mongoose Safely [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `Model.create` • `findById` • `find` • `findOneAndUpdate(filter • Production Backend Pitfall & Guardrail (CRUD Using Mongoose Safely) • Hands-On TypeScript / Node.js Implementation (CRUD Using Mongoose Safely) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for CRUD Using Mongoose Safely

Purpose: Enabling `mongoose.set("runValidators", true)` globally at startup so NO `findOneAndUpdate` call can forget schema validation!

What Topic Is Next → Next Topic 10.12: Query Operators and NoSQL Operator Injection Defense

10.12 Query Operators and NoSQL Operator Injection Defense [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Enabling `mongoose.set("sanitizeFilter" • True)` + validating inputs as primitive `string`s in Zod so `{ "email": { "$gt": "" } }` attacker JSON cannot bypass login filters! • Production Backend Pitfall & Guardrail (Query Operators and NoSQL Operator Injection Defense) • Hands-On TypeScript / Node.js Implementation (Query Operators and NoSQL Operator Injection Defense) • Query Operators and NoSQL Operator Injection Defense — Latency, Throughput & Interview Trade-Offs • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Query Operators • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards

Purpose: Neutralizing NoSQL injection (`POST /login` `{ "email": { "$ne": null }, "password": "..." }`) at BOTH Zod validation AND Mongoose `sanitizeFilter` layers!

What Topic Is Next → Next Topic 10.13: Population, References and Preventing N+1 Queries

10.13 Population, References and Preventing N+1 Queries [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `.populate({ path: "categoryId" • Select: "name slug" })` (executes 1 batched `$in` query across the page • NOT N queries — UNLESS you call `await doc.populate()` inside a `for` loop!) • Production Backend Pitfall & Guardrail (Population, References and Preventing N+1 Queries) • Hands-On TypeScript / Node.js Implementation (Population, References and Preventing N+1 Queries) • Population, References and Preventing N+1 Queries — Latency, Throughput & Interview Trade-Offs • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Population

Purpose: Selecting strictly `select: "name slug"` during `.populate()` so full referenced documents aren't over-fetched.

What Topic Is Next → Next Topic 10.14: Virtual Properties

10.14 Virtual Properties [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Computed non-persisted properties (`priceInRupees`) • Virtual populate • Why `.lean()` omits virtuals by default (compute DTO fields in your clean TypeScript DTO mapper instead!) • Production Backend Pitfall & Guardrail (Virtual Properties) • Hands-On TypeScript / Node.js Implementation (Virtual Properties) • Virtual Properties — Latency, Throughput & Interview Trade-Offs • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Virtual Properties

Purpose: Preferring explicit pure TypeScript `toProductDto(leanDoc)` functions that work identically with `.lean()` and unit-test mocks.

What Topic Is Next → Next Topic 10.15: Aggregation Using Mongoose

10.15 Aggregation Using Mongoose [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `ProductModel.aggregate<CategoryStat>([...])` noting Mongoose does NOT auto-cast string `"65f..."` to `Types.ObjectId` inside `.aggregate({ $match })` (cast `new Types.ObjectId(id)` explicitly!) • Production Backend Pitfall & Guardrail (Aggregation Using Mongoose) • Hands-On TypeScript / Node.js Implementation (Aggregation Using Mongoose) • Aggregation Using Mongoose — Latency, Throughput & Interview Trade-Offs • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Aggregation Using Mongoose • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards • Diagnostics, Profiling & Telemetry: structured telemetry, profiler traces & CI checks

Purpose: Building fast inventory/catalog analytics endpoints typed cleanly via `aggregate<ResultType>()`.

What Topic Is Next → Next Topic 10.16: Transactions and ClientSessions in Mongoose

10.16 Transactions and ClientSessions in Mongoose [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `const session = await mongoose.startSession() • Try { await session.withTransaction(async () => { ... }) • } finally { await session.endSession() • Production Backend Pitfall & Guardrail (Transactions and ClientSessions in Mongoose) • Hands-On TypeScript / Node.js Implementation (Transactions and ClientSessions in Mongoose) • Transactions and ClientSessions in Mongoose — Latency, Throughput & Interview Trade-Offs • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Transactions

Purpose: Atomically creating a `CategoryTransferLog` and reassigning `Product` categories inside `withTransaction` with guaranteed `endSession()`.

What Topic Is Next → Next Topic 10.17: Index Configuration and Production Migration Safety

10.17 Index Configuration and Production Migration Safety [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Declaring compound/unique/TTL indexes on schemas • Setting `autoIndex: false` in `NODE_ENV=production` (prevents startup index builds locking/slowing boot!) • Syncing explicitly via `Model.syncIndexes()` / deploy scripts • Production Backend Pitfall & Guardrail (Index Configuration and Production Migration Safety) • Hands-On TypeScript / Node.js Implementation (Index Configuration and Production Migration Safety) • Index Configuration and Production Migration Safety — Latency, Throughput & Interview Trade-Offs • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Index Configuration

Purpose: Running index creation in a controlled CI/deploy step before rolling out new application containers.

What Topic Is Next → Next Topic 10.18: Query Optimization (.lean, .select, Cursor Streaming, Explain)

10.18 Query Optimization (.lean, .select, Cursor Streaming, Explain) [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Combining `.find(filter).select("sku title pricePaise").sort(...).limit(20).lean()` + `.cursor()` async iteration for batch jobs + `.explain("executionStats")` assertions • Production Backend Pitfall & Guardrail (Query Optimization (.lean, .select, Cursor Streaming, Explain)) • Hands-On TypeScript / Node.js Implementation (Query Optimization (.lean, .select, Cursor Streaming, Explain)) • Query Optimization (.lean, .select, Cursor Streaming, Explain) — Latency, Throughput & Interview Trade-Offs • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Query Optimization (.lean, .select, Cursor Streaming... • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards • Diagnostics, Profiling & Telemetry: structured telemetry, profiler traces & CI checks

Purpose: Cutting `GET /api/v1/products` p95 response time from `48ms` down to `6ms` and heap allocations by `70%`!

What Topic Is Next → Next Topic 10.19: Connection Pooling Sizing and Monitoring

10.19 Connection Pooling Sizing and Monitoring [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Sizing `maxPoolSize` (`10–25` per Node process: `totalConnections = replicas * maxPoolSize < Atlas/Mongo limit!`) • `minPoolSize` • Monitoring pool checkout wait times • Production Backend Pitfall & Guardrail (Connection Pooling Sizing and Monitoring) • Hands-On TypeScript / Node.js Implementation (Connection Pooling Sizing and Monitoring) • Connection Pooling Sizing and Monitoring — Latency, Throughput & Interview Trade-Offs • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Connection Pooling Sizing

Purpose: Avoiding connection storms when scaling 4 PM2/Docker workers against a shared MongoDB replica set.

What Topic Is Next → Next Topic 10.20: Mongoose Error Handling Translation

10.20 Mongoose Error Handling Translation [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Mapping `MongoServerError` (`code === 11000` -> `409 Conflict` extracting duplicate key name!) • `Error.CastError` (invalid `24-hex` ObjectId -> `400 Bad Request`) • `Error.ValidationError` -> `422` • `VersionError` (`__v` optimistic concurrency -> `409`) • Production Backend Pitfall & Guardrail

(Mongoose Error Handling Translation) • Hands-On TypeScript / Node.js Implementation (Mongoose Error Handling Translation) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Mongoose Error Handling Translation

Purpose: Returning clean `{ code: "DUPLICATE_SKU", status: 409 }` instead of leaking raw `E11000 duplicate key error collection: db.products index: sku_1` to clients!

What Topic Is Next → Next Topic 10.21: Repository Pattern with Mongoose

10.21 Repository Pattern with Mongoose [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Implementing `class MongooseProductRepository implements ProductRepository` returning clean domain POJOs (`ProductEntity`) so `ProductService` has zero `import mongoose` coupling! • Production Backend Pitfall & Guardrail (Repository Pattern with Mongoose) • Hands-On TypeScript / Node.js Implementation (Repository Pattern with Mongoose) • Repository Pattern with Mongoose — Latency, Throughput & Interview Trade-Offs • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Repository Pattern with Mongoose • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards • Diagnostics, Profiling & Telemetry: structured telemetry, profiler traces & CI checks

Purpose: Keeping `InventoryOrchestratorService` in Project 3 cleanly decoupled across `MongooseProductRepository` and `PrismaStockRepository`!

What Topic Is Next → Next Topic 10.22: TypeScript Integration with Mongoose

10.22 TypeScript Integration with Mongoose [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Explicit `interface ProductDoc` + `Schema<ProductDoc>` + `HydratedDocument<ProductDoc>` • `InferSchemaType<typeof ProductSchema>` • Strict `Types.ObjectId` vs `string` boundary mapping • Production Backend Pitfall & Guardrail (TypeScript Integration with Mongoose) • Hands-On TypeScript / Node.js Implementation (TypeScript Integration with Mongoose) • TypeScript Integration with Mongoose — Latency, Throughput & Interview Trade-Offs • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for TypeScript Integration with Mongoose

Purpose: Achieving zero `any` casts across Mongoose schemas, queries, and repository return types.

What Topic Is Next → Module 10 Capstone (Week 10 Lab Build: Normalized 3NF PostgreSQL Financial Ledger & Window-Function Analytics) → Next Topic 11.1: Relational Database Concepts

Module 10 Hands-On Capstone Project (100 Marks): Week 10 Lab Build: Normalized 3NF PostgreSQL Financial Ledger & Window-Function Analytics — Author a 3NF PostgreSQL 16 schema (`accounts`, `journal_entries`, `ledger_postings`) using `NUMERIC(14,2)`, `CHECK` constraints, `NOT EXISTS` anti-joins, recursive CTE category trees, `DENSE...

MODULE 11 OF 24 • STAGE 3 • 25 LIVE + LAB HOURS • 23 TOPICS (184 SUBTOPICS) • 1 CAPSTONE PROJECT (100 MARKS)

Module 11: PostgreSQL and Complete SQL

Master enterprise relational database engineering with PostgreSQL 16 and complete SQL — MVCC/WAL internals, strict DDL types & constraints, joins, correlated subqueries, CTEs, Window Functions, views/materialized views, composite B-Tree/GIN indexes, ACID isolation levels, FOR UPDATE locks/deadlocks, 3NF normalization, EXPLAIN ANALYZE, and backups.

Objectives: ✓ Design normalized (`3NF`/`BCNF`) PostgreSQL schemas using exact types (`NUMERIC`, `UUID`, `TIMESTAMPTZ`, `JSONB`) and integrity constraints. • ✓ Write complex analytical & transactional SQL using `JOIN`s, `EXISTS`, Recursive CTEs, Window Functions (`ROW_NUMBER`, `DENSE_RANK`, `LAG`), and `ON CONFLICT` upserts. • ✓ Prevent Lost Updates, Dirty/Phantom Reads, and Deadlocks using ACID isolation levels and deterministic lock ordering (`SELECT ... FOR UPDATE`). • ✓ Diagnose slow SQL using `EXPLAIN (ANALYZE, BUFFERS)` and tune composite/partial/covering indexes.

11.1 Relational Database Concepts [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Relations (tables) • Tuples (rows) • Attributes (columns) • Domains • Production Backend Pitfall & Guardrail (Relational Database Concepts) • Hands-On TypeScript / Node.js Implementation (Relational Database Concepts) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Relational Database Concepts

Purpose: Replacing 300 lines of slow Node.js nested-loop aggregation with one 8-line set-based PostgreSQL query running in `3ms`!

What Topic Is Next → Next Topic 11.2: PostgreSQL Architecture (MVCC, Shared Buffers & WAL)

11.2 PostgreSQL Architecture (MVCC, Shared Buffers & WAL) [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Client backend processes • `shared_buffers` page cache • Multi-Version Concurrency Control (**MVCC**: readers never block writers • Writers never block readers!) • Production Backend Pitfall & Guardrail (PostgreSQL Architecture (MVCC, Shared Buffers & WAL)) • Hands-On TypeScript / Node.js Implementation (PostgreSQL Architecture (MVCC, Shared Buffers & WAL)) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for PostgreSQL Architecture (MVCC, Shared Buffers & WAL)

Purpose: Understanding why `UPDATE` in PostgreSQL writes a new tuple version (`xmin`/`xmax`) and why `autovacuum` health is vital on high-churn tables.

What Topic Is Next → Next Topic 11.3: Database, Schema and Table Creation (DDL)

11.3 Database, Schema and Table Creation (DDL) [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `CREATE SCHEMA` • `CREATE TABLE` • `GENERATED ALWAYS AS IDENTITY` (modern SQL standard over legacy `SERIAL`!) • Snake_case identifiers • Production Backend Pitfall & Guardrail (Database, Schema and Table Creation (DDL)) • Hands-On TypeScript / Node.js Implementation (Database, Schema and Table Creation (DDL)) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Database

Purpose: Creating clean relational tables (`warehouses`, `inventory_stock`, `stock_ledger_entries`) with identity/UUID keys.

What Topic Is Next → Next Topic 11.4: PostgreSQL Data Types Selection

11.4 PostgreSQL Data Types Selection [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `BIGINT` / `UUID` (`gen_random_uuid()`) • `NUMERIC(14 • 2)` (exact decimal currency — NEVER `REAL`/`DOUBLE PRECISION`!) • `TEXT` vs `VARCHAR(n)` • Production Backend Pitfall & Guardrail (PostgreSQL Data Types Selection) • Hands-On TypeScript / Node.js Implementation (PostgreSQL Data Types Selection) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for PostgreSQL Data Types Selection

Purpose: Storing all timestamps as UTC `TIMESTAMPTZ` and prices as integer paise (`BIGINT`) or `NUMERIC(12, 2)`.

What Topic Is Next → Next Topic 11.5: Primary Keys and Foreign Keys

11.5 Primary Keys and Foreign Keys [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Surrogate keys (`BIGINT` / `UUIDv7`/`v4`) vs natural keys • Composite primary keys • `REFERENCES ... ON DELETE RESTRICT | CASCADE | SET NULL` • **ALWAYS indexing Foreign Key columns**! • Production Backend Pitfall & Guardrail (Primary Keys and Foreign Keys) • Hands-On TypeScript / Node.js Implementation (Primary Keys and Foreign Keys) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Primary Keys

Purpose: Using `ON DELETE RESTRICT` on `orders.user_id` so financial order history can NEVER be accidentally cascade-wiped!

What Topic Is Next → Next Topic 11.6: Constraints (NOT NULL, UNIQUE, CHECK)

11.6 Constraints (NOT NULL, UNIQUE, CHECK) [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Enforcing domain invariants inside the database engine (`CHECK (qty_on_hand >= 0)` • `CHECK (reserved_qty <= qty_on_hand)` • `UNIQUE (warehouse_id • Product_sku)`) so even buggy scripts cannot corrupt data! • Production Backend Pitfall & Guardrail (Constraints (NOT NULL, UNIQUE, CHECK)) • Hands-On TypeScript / Node.js Implementation (Constraints (NOT NULL, UNIQUE, CHECK)) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Constraints (NOT NULL, UNIQUE, CHECK)

Purpose: Guaranteeing `qty_on_hand >= 0` mathematically inside PostgreSQL even under 1,000 concurrent checkout requests!

What Topic Is Next → Next Topic 11.7: INSERT, SELECT, UPDATE, DELETE and Upserts

11.7 INSERT, SELECT, UPDATE, DELETE and Upserts [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Multi-row `INSERT ... VALUES ... RETURNING *` • Atomic upsert `INSERT ... ON CONFLICT (warehouse_id • Sku) DO UPDATE SET qty_on_hand = inventory_stock.qty_on_hand + EXCLUDED.qty_on_hand RETURNING *` • Guarded `UPDATE`/`DELETE` • Production Backend Pitfall & Guardrail (INSERT, SELECT, UPDATE, DELETE and Upserts) • Hands-On TypeScript / Node.js Implementation (INSERT, SELECT, UPDATE, DELETE and Upserts) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for INSERT

Purpose: Returning inserted/updated rows in 1 network round-trip via `RETURNING *` and performing race-free stock initialization via `ON CONFLICT DO UPDATE`!

What Topic Is Next → Next Topic 11.8: WHERE, ORDER BY and GROUP BY

11.8 WHERE, ORDER BY and GROUP BY [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Sargable predicates (`created_at >= $1 • Created_at < $2` uses index vs `EXTRACT(YEAR FROM created_at) = 2026` breaks index!) • `IS NULL` / `IS NOT NULL` (never `= NULL`!) • `NULLS LAST` • Production Backend Pitfall & Guardrail (WHERE, ORDER BY and GROUP BY) • Hands-On TypeScript / Node.js Implementation (WHERE, ORDER BY and GROUP BY) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for WHERE

Purpose: Writing index-friendly sargable date-range filters (`>= startOfDay AND < nextDay`) for daily sales reports.

What Topic Is Next → Next Topic 11.9: HAVING and Aggregate Functions

11.9 HAVING and Aggregate Functions [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `COUNT(*)` • `COUNT(DISTINCT user_id)` • `COALESCE(SUM(amount) • 0)` • Production Backend Pitfall & Guardrail (HAVING and Aggregate Functions) • Hands-On TypeScript / Node.js Implementation (HAVING and Aggregate Functions) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for HAVING

Purpose: Computing `paid_orders`, `cancelled_orders`, and `gross_revenue` in ONE single table scan using PostgreSQL `FILTER (WHERE ...)`!

What Topic Is Next → Next Topic 11.10: INNER, LEFT, RIGHT and FULL OUTER Joins

11.10 INNER, LEFT, RIGHT and FULL OUTER Joins [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Hash Join vs Nested Loop vs Merge Join • `INNER JOIN` vs `LEFT JOIN` (preserving zero-order customers!) • Why putting a right-table filter in `WHERE` instead of `ON` silently turns a `LEFT JOIN` into an `INNER JOIN`! • Production Backend Pitfall & Guardrail (INNER, LEFT, RIGHT and FULL OUTER Joins) • Hands-On TypeScript / Node.js Implementation (INNER, LEFT, RIGHT and FULL OUTER Joins) • INNER, LEFT, RIGHT and FULL OUTER Joins — Latency, Throughput & Interview Trade-Offs • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for INNER

Purpose: Listing all warehouses alongside their today's shipment counts (including `0` for quiet warehouses via `LEFT JOIN`!).

What Topic Is Next → Next Topic 11.11: Self Joins

11.11 Self Joins [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Joining a table onto itself with mandatory table aliases (`e.manager_id = m.id` employee-manager org trees • Finding duplicate/overlapping shifts) • Production Backend Pitfall & Guardrail (Self Joins) • Hands-On TypeScript / Node.js Implementation (Self Joins) • Self Joins — Latency, Throughput & Interview Trade-Offs • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Self Joins • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards

Purpose: Resolving `employee -> reporting_manager` names and parent->child category labels in 1 query.

What Topic Is Next → Next Topic 11.12: Subqueries, Correlated Subqueries and EXISTS vs NOT IN

11.12 Subqueries, Correlated Subqueries and EXISTS vs NOT IN [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Scalar subqueries • Derived tables • `EXISTS` (short-circuits on 1st match!) • The **deadly `NOT IN` NULL trap** (`WHERE id NOT IN (SELECT parent_id FROM t)` returns **ZERO ROWS** if `parent_id` contains even a single `NULL` — ALWAYS use `NOT EXISTS`!)! • Production Backend Pitfall & Guardrail

(Subqueries, Correlated Subqueries and EXISTS vs NOT IN) • Hands-On TypeScript / Node.js Implementation (Subqueries, Correlated Subqueries and EXISTS vs NOT IN) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Subqueries

Purpose: Finding orphan or never-ordered SKUs fast and safely with `WHERE NOT EXISTS (SELECT 1 FROM order_items oi WHERE oi.sku = p.sku)`.

What Topic Is Next → Next Topic 11.13: Common Table Expressions (WITH & WITH RECURSIVE)

11.13 Common Table Expressions (WITH & WITH RECURSIVE) [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Readable multi-step `WITH` CTE pipelines • Data-modifying CTEs (`WITH moved AS (DELETE ... RETURNING *) INSERT INTO archive ...`) • `WITH RECURSIVE` arbitrary-depth category/org trees • Production Backend Pitfall & Guardrail (Common Table Expressions (WITH & WITH RECURSIVE)) • Hands-On TypeScript / Node.js Implementation (Common Table Expressions (WITH & WITH RECURSIVE)) • Common Table Expressions (WITH & WITH RECURSIVE) — Latency, Throughput & Interview Trade-Offs • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Common Table Expressions (WITH & WITH RECURSIVE)

Purpose: Fetching an entire nested `Electronics > Computers > Laptops > Gaming` category subtree in 1 `WITH RECURSIVE` query!

What Topic Is Next → Next Topic 11.14: Window Functions (OVER, PARTITION BY, ROW_NUMBER, DENSE_RANK, LAG)

11.14 Window Functions (OVER, PARTITION BY, ROW_NUMBER, DENSE_RANK, LAG) [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Ranking (`ROW_NUMBER` • `RANK` • `DENSE_RANK` for Top-N per category!) • Offset (`LAG` • Production Backend Pitfall & Guardrail (Window Functions (OVER, PARTITION BY, ROW_NUMBER, DENSE_RANK, LAG)) • Hands-On TypeScript / Node.js Implementation (Window Functions (OVER, PARTITION BY, ROW_NUMBER, DENSE_RANK, LAG)) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Window Functions (OVER, PARTITION BY, ROW_NUMBER, DE...

Purpose: Solving the #1 SQL interview & dashboard query: `"Top 3 best-selling SKUs per warehouse"` cleanly with `DENSE_RANK() OVER (PARTITION BY warehouse_id ORDER BY units_sold D...

What Topic Is Next → Next Topic 11.15: Views and Materialized Views

11.15 Views and Materialized Views [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Virtual `CREATE VIEW` (query simplification / column security) vs persisted `CREATE MATERIALIZED VIEW` + unique index enabling zero-lock `REFRESH MATERIALIZED VIEW CONCURRENTLY`! • Production Backend Pitfall & Guardrail (Views and Materialized Views) • Hands-On TypeScript / Node.js Implementation (Views and Materialized Views) • Views and Materialized Views — Latency, Throughput & Interview Trade-Offs • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Views • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards • Diagnostics, Profiling & Telemetry: structured telemetry, profiler traces & CI checks

Purpose: Pre-aggregating heavy 30-day revenue dashboards into a Materialized View refreshed every 5 minutes via BullMQ/cron so dashboard APIs respond in `2ms`!

What Topic Is Next → Next Topic 11.16: Indexes and Composite Indexes

11.16 Indexes and Composite Indexes [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: B-Tree leftmost-prefix rule (`(tenant_id • Status • Created_at DESC)` serves `(tenant_id)` • `(tenant_id • Production Backend Pitfall & Guardrail (Indexes and Composite Indexes) • Hands-On TypeScript / Node.js Implementation (Indexes and Composite Indexes) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Indexes

Purpose: Building indexes on live production tables with `CREATE INDEX CONCURRENTLY` so `INSERT`/`UPDATE` traffic is NEVER blocked!

What Topic Is Next → Next Topic 11.17: Transactions and ACID Properties

11.17 Transactions and ACID Properties [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: **Atomicity** (all or nothing) • **Consistency** (constraints satisfied) • **Isolation** (concurrent tx safety) • **Durability** (committed WAL survives power loss) • Production Backend Pitfall & Guardrail (Transactions and ACID Properties) • Hands-On TypeScript / Node.js Implementation (Transactions and ACID Properties) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Transactions

Purpose: Deducting `inventory_stock` AND inserting an immutable `stock_ledger_entries` audit row inside ONE atomic transaction.

What Topic Is Next → Next Topic 11.18: Transaction Isolation Levels

11.18 Transaction Isolation Levels [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Read phenomena (Dirty Read • Non-Repeatable Read • Phantom Read • Serialization Anomaly) • Production Backend Pitfall & Guardrail (Transaction Isolation Levels) • Hands-On TypeScript / Node.js Implementation (Transaction Isolation Levels) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Transaction Isolation Levels

Purpose: Knowing that under default `READ COMMITTED`, two concurrent `SELECT balance` -> `UPDATE balance = $calc` transactions WILL overwrite each other unless `FOR UPDATE` or ato...

What Topic Is Next → Next Topic 11.19: Row-Level Locks (FOR UPDATE / SKIP LOCKED) and Deadlocks

11.19 Row-Level Locks (FOR UPDATE / SKIP LOCKED) and Deadlocks [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Pessimistic row locking `SELECT ... FOR UPDATE` • Job-claiming `FOR UPDATE SKIP LOCKED` • Preventing **Deadlocks (`40P01`)** by ALWAYS locking multiple rows in deterministic sorted ID order (`ORDER BY product_id ASC FOR UPDATE`)! • Production Backend Pitfall & Guardrail (Row-Level Locks (FOR UPDATE / SKIP LOCKED) and Deadlocks) • Hands-On TypeScript / Node.js Implementation (Row-Level Locks (FOR UPDATE / SKIP LOCKED) and Deadlocks) • Row-Level Locks (FOR UPDATE / SKIP LOCKED) and Deadlocks — Latency, Throughput & Interview Trade-Offs • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Row-

Level Locks (FOR UPDATE / SKIP LOCKED)

Purpose: Sorting cart `sku`s alphabetically (`["SKU-A", "SKU-B"]`) BEFORE locking inventory rows so Checkout #1 (`A, B`) and Checkout #2 (`B, A` -> sorted to `A, B`!) can NEVER de...

What Topic Is Next → Next Topic 11.20: Database Normalization (1NF, 2NF, 3NF, BCNF)

11.20 Database Normalization (1NF, 2NF, 3NF, BCNF) [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Eliminating repeating comma-separated columns (`1NF`) • Partial key dependencies (`2NF`) • Transitive non-key dependencies (`3NF`) • Plus when controlled snapshot denormalization (`order_items.unit_price_at_purchase`) is required! • Production Backend Pitfall & Guardrail (Database Normalization (1NF, 2NF, 3NF, BCNF)) • Hands-On TypeScript / Node.js Implementation (Database Normalization (1NF, 2NF, 3NF, BCNF)) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Database Normalization (1NF, 2NF, 3NF, BCNF)

Purpose: Normalizing `product.current_price` (`3NF`) while intentionally snapshotting `order_items.unit_price_paise` so price changes never alter past orders!

What Topic Is Next → Next Topic 11.21: Relationships and ER Diagrams

11.21 Relationships and ER Diagrams [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Drawing Crow's Foot Entity-Relationship Diagrams (`1:1` • `1:N` • `M:N` associative join tables with composite PK/unique + extra attributes) • Optionality (`0..1` vs `1..*`) • Production Backend Pitfall & Guardrail (Relationships and ER Diagrams) • Hands-On TypeScript / Node.js Implementation (Relationships and ER Diagrams) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Relationships

Purpose: Authoring clear Mermaid/Crow's Foot ER diagrams required for Project 3 and the Final Capstone deliverables.

What Topic Is Next → Next Topic 11.22: EXPLAIN (ANALYZE, BUFFERS) and Query Optimization

11.22 EXPLAIN (ANALYZE, BUFFERS) and Query Optimization [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Reading execution trees bottom-up (`Actual Time` • `Rows Removed by Filter` • `Buffers: shared hit` RAM vs `read` disk • `Seq Scan` vs `Index Scan` vs `Index Only Scan` • Production Backend Pitfall & Guardrail (EXPLAIN (ANALYZE, BUFFERS) and Query Optimization) • Hands-On TypeScript / Node.js Implementation (EXPLAIN (ANALYZE, BUFFERS) and Query Optimization) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for EXPLAIN (ANALYZE, BUFFERS)

Purpose: Proving a `420ms` `Seq Scan` dropped to a `0.18ms` `Index Scan` after adding a targeted composite B-Tree index!

What Topic Is Next → Next Topic 11.23: Backup and Restore Fundamentals (pg_dump & pg_restore)

11.23 Backup and Restore Fundamentals (pg_dump & pg_restore) [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Logical backups with custom compressed format `pg_dump -Fc -d app_db -f backup.dump` • `pg_restore --clean --if-exists` • WAL archiving concepts • Regular restore verification drills! • Production Backend Pitfall & Guardrail (Backup and Restore Fundamentals (pg_dump & pg_restore)) • Hands-On TypeScript / Node.js Implementation (Backup and Restore Fundamentals (pg_dump & pg_restore)) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Backup

Purpose: Automating nightly compressed `pg_dump -Fc` backups and testing `pg_restore` into a disposable container weekly.

What Topic Is Next → Module 11 Capstone (Week 11 Lab Build: Prisma Interactive ACID Wallet Transfer & Optimistic Locking Service) → Next Topic 12.1: Introduction to Prisma ORM Architecture

Module 11 Hands-On Capstone Project (100 Marks): Week 11 Lab Build: Prisma Interactive ACID Wallet Transfer & Optimistic Locking Service — Implement a deadlock-free PostgreSQL + Prisma 5 wallet & seat-booking service using sorted `SELECT ... FOR UPDATE` row locking inside `prisma.$transaction`, optimistic `version` compare-and-...

MODULE 12 OF 24 • STAGE 3 • 25 LIVE + LAB HOURS • 22 TOPICS (176 SUBTOPICS) • 1 CAPSTONE PROJECT (100 MARKS)

Module 12: Prisma ORM and Advanced Database Engineering

Combine PostgreSQL relational rigor with end-to-end TypeScript type safety using Prisma ORM — schema modeling, relations, generated PrismaClient, cursor pagination, N+1 prevention, interactive ACID transactions, production migrations, seeding, pooling, and Month 3 Project 3: Multi-Database Inventory Management Backend.

Objectives: ✓ Model normalized PostgreSQL schemas in `schema.prisma` with exact `@db.*` native types, relations, and composite indexes. • ✓ Execute type-safe CRUD, cursor pagination, `select`/`include` projection, and bounded interactive `$transaction` workflows. • ✓ Manage schema migrations safely (`migrate dev` locally vs `migrate deploy` in CI/production) and size connection pools. • ✓ Deliver **Month 3 Project (Project 3): Multi-Database Inventory Management Backend** integrating MongoDB (Mongoose) + PostgreSQL (Prisma).

12.1 Introduction to Prisma ORM Architecture [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Declarative `schema.prisma` -> `prisma generate` -> 100% compile-time type-safe `PrismaClient` query builder + Prisma Migrate DDL engine • Production Backend Pitfall & Guardrail (Introduction to Prisma ORM Architecture) • Hands-On TypeScript / Node.js Implementation (Introduction to Prisma ORM Architecture) • Introduction to Prisma ORM Architecture — Latency, Throughput & Interview Trade-Offs • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Introduction to Prisma ORM Architecture • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards • Diagnostics, Profiling & Telemetry: structured telemetry, profiler traces & CI checks

Purpose: Catching invalid column names, missing required fields, and wrong relation types at `tsc` compile time before running a single query!

What Topic Is Next → Next Topic 12.2: Prisma Installation and Multi-Environment Configuration

12.2 Prisma Installation and Multi-Environment Configuration [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Installing `prisma` (devDependency) + `@prisma/client` (dependency) • Configuring `DATABASE_URL` via `.env` • Separating direct DB URL vs pooled URL when needed • Production Backend Pitfall & Guardrail (Prisma Installation and Multi-Environment Configuration) • Hands-On TypeScript / Node.js Implementation (Prisma Installation and Multi-Environment Configuration) • Prisma Installation and Multi-Environment Configuration — Latency, Throughput &

Interview Trade-Offs • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Prisma Installation

Purpose: Keeping CLI `prisma` out of production runtime dependencies when only `@prisma/client` (+ `migrate deploy` job) is needed.

What Topic Is Next → Next Topic 12.3: Prisma Schema Anatomy and Database Naming (@map & @@map)

12.3 Prisma Schema Anatomy and Database Naming (@map & @@map) [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Idiomatic camelCase TypeScript fields (`qtyOnHand` • `createdAt`) mapped cleanly onto PostgreSQL `snake_case` columns/tables via `@map("qty_on_hand")` • `@@map("inventory_stock")`! • Production Backend Pitfall & Guardrail (Prisma Schema Anatomy and Database Naming (@map & @@map)) • Hands-On TypeScript / Node.js Implementation (Prisma Schema Anatomy and Database Naming (@map & @@map)) • Prisma Schema Anatomy and Database Naming (@map & @@map) — Latency, Throughput & Interview Trade-Offs • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Prisma Schema Anatomy

Purpose: Enjoying idiomatic camelCase in TypeScript AND idiomatic unquoted `snake_case` in `psql`/DBA SQL queries simultaneously!

What Topic Is Next → Next Topic 12.4: Models, Scalar Fields, Enums and Defaults

12.4 Models, Scalar Fields, Enums and Defaults [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `@id @default(uuid())` vs `@default(autoincrement())` • `@updatedAt` • Native Postgres `enum` mapping • Optional `?` (`NULL`) • Production Backend Pitfall & Guardrail (Models, Scalar Fields, Enums and Defaults) • Hands-On TypeScript / Node.js Implementation (Models, Scalar Fields, Enums and Defaults) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Models

Purpose: Declaring composite uniqueness and query indexes right beside model fields so indexes are versioned in Git migrations!

What Topic Is Next → Next Topic 12.5: Native PostgreSQL Data Types (@db.*) and Decimal Precision

12.5 Native PostgreSQL Data Types (@db.*) and Decimal Precision [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `Decimal @db.Decimal(12 • 2)` (`Prisma.Decimal` exact decimal math — never JS `float`!) • `@db.Timestamptz(6)` • `@db.Uuid` • Production Backend Pitfall & Guardrail (Native PostgreSQL Data Types (@db.*) and Decimal Precision) • Hands-On TypeScript / Node.js Implementation (Native PostgreSQL Data Types (@db.*) and Decimal Precision) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Native PostgreSQL Data Types (@db.*)

Purpose: Performing exact rupee/paise arithmetic with `Prisma.Decimal` (or `BigInt` paise) without `0.1 + 0.2 = 0.30000000000000004` rounding drift!

What Topic Is Next → Next Topic 12.6: Modelling Relationships in Prisma (@relation)

12.6 Modelling Relationships in Prisma (@relation) [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Relation fields vs foreign-key scalar fields • Explicit `onDelete: Restrict | Cascade | SetNull` • Disambiguating multiple relations between the same two models (`@relation("SourceWarehouse")` vs `@relation("TargetWarehouse")`) • Production Backend Pitfall & Guardrail (Modelling Relationships in Prisma (@relation)) • Hands-On TypeScript / Node.js Implementation (Modelling Relationships in Prisma (@relation)) • Modelling Relationships in Prisma (@relation) — Latency, Throughput & Interview Trade-Offs • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Modelling Relationships in Prisma (@relation)

Purpose: Modelling `StockTransfer` having BOTH `fromWarehouseId` and `toWarehouseId` pointing to `Warehouse`!

What Topic Is Next → Next Topic 12.7: One-to-One, One-to-Many and Explicit Many-to-Many Relations

12.7 One-to-One, One-to-Many and Explicit Many-to-Many Relations [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `1:1` (`@unique` FK) • `1:N` • Why **Explicit Join Models** (`OrderItem` with `qty` • `unitPricePaise` • Production Backend Pitfall & Guardrail (One-to-One, One-to-Many and Explicit Many-to-Many Relations) • Hands-On TypeScript / Node.js Implementation (One-to-One, One-to-Many and Explicit Many-to-Many Relations) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for One-to-One

Purpose: Using explicit join models (`WarehouseStock`, `OrderItem`, `UserRoleAssignment`) so audit timestamps and quantities are first-class columns.

What Topic Is Next → Next Topic 12.8: Singleton PrismaClient Lifecycle Management

12.8 Singleton PrismaClient Lifecycle Management [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Creating ONE shared `PrismaClient` per Node process (caching on `globalThis` during `tsx watch` dev hot-reloads so dev mode doesn't leak 50 DB pools!) • Calling `$disconnect()` during graceful shutdown • Production Backend Pitfall & Guardrail (Singleton PrismaClient Lifecycle Management) • Hands-On TypeScript / Node.js Implementation (Singleton PrismaClient Lifecycle Management) • Singleton PrismaClient Lifecycle Management — Latency, Throughput & Interview Trade-Offs • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Singleton PrismaClient Lifecycle Management • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards

Purpose: Preventing `FATAL: sorry, too many clients already` inPostgreSQL!

What Topic Is Next → Next Topic 12.9: Type-Safe CRUD Operations with Prisma Client

12.9 Type-Safe CRUD Operations with Prisma Client [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `create` • `createManyAndReturn` • `findUnique` / `findFirst` • `findMany` • Production Backend Pitfall & Guardrail (Type-Safe CRUD Operations with Prisma Client) • Hands-On TypeScript / Node.js Implementation (Type-Safe CRUD Operations with Prisma Client) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Type-Safe CRUD Operations with Prisma Client

Purpose: Using `updateMany({ where: { sku, qtyOnHand: { gte: qty } }, data: { qtyOnHand: { decrement: qty } } })` checking `result.count === 1` for lock-free atomic stock deductio...

What Topic Is Next → Next Topic 12.10: Composable Filtering and Multi-Column Sorting

12.10 Composable Filtering and Multi-Column Sorting [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Building typed `const where: Prisma.InventoryStockWhereInput = { ... }` dynamically (`contains` • `mode: "insensitive"` • `in` • `gte`/`lte` • Production Backend Pitfall & Guardrail (Composable Filtering and Multi-Column Sorting) • Hands-On TypeScript / Node.js Implementation (Composable Filtering and Multi-Column Sorting) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Composable Filtering

Purpose: Constructing clean optional filter queries with 100% TypeScript autocomplete and zero string-concatenated SQL.

What Topic Is Next → Next Topic 12.11: Offset vs Native Cursor Pagination in Prisma

12.11 Offset vs Native Cursor Pagination in Prisma [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `skip` + `take` vs high-scale `cursor: { id: lastId } • Skip: 1 • Take: limit + 1` keyset pagination in Prisma • Production Backend Pitfall & Guardrail (Offset vs Native Cursor Pagination in Prisma) • Hands-On TypeScript / Node.js Implementation (Offset vs Native Cursor Pagination in Prisma) • Offset vs Native Cursor Pagination in Prisma — Latency, Throughput & Interview Trade-Offs • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Offset vs Native Cursor Pagination in Prisma

Purpose: Paginating 2,000,000 stock ledger movements at constant `O(log N)` speed using Prisma `cursor` + `take: limit + 1`.

What Topic Is Next → Next Topic 12.12: Nested Queries, select vs include and Preventing N+1

12.12 Nested Queries, select vs include and Preventing N+1 [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Projecting strictly required columns with `select: { id: true • Email: true }` • Eager-loading relations via `include` (`relationLoadStrategy: "join"`) • Prisma's automatic `findUnique` dataloader batching • Production Backend Pitfall & Guardrail (Nested Queries, select vs include and Preventing N+1) • Hands-On TypeScript / Node.js Implementation (Nested Queries, select vs include and Preventing N+1) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Nested Queries

Purpose: Ensuring `passwordHash` is NEVER even fetched over the wire from PostgreSQL on profile/list queries via explicit `select`!

What Topic Is Next → Next Topic 12.13: Sequential Batch Transactions ($transaction([...]))

12.13 Sequential Batch Transactions ($transaction([...])) [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `await prisma.$transaction([op1 • Op2] • { isolationLevel: Prisma.TransactionIsolationLevel.RepeatableRead })` when operations don't depend on intermediate JS branching • Production Backend Pitfall & Guardrail (Sequential Batch Transactions ($transaction([...]))) • Hands-On TypeScript / Node.js Implementation (Sequential Batch Transactions ($transaction([...]))) • Sequential Batch Transactions ($transaction([...])) — Latency, Throughput & Interview Trade-Offs • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Sequential Batch Transactions ($transaction([...]))

Purpose: Executing a `deleteMany` + `createMany` replacement atomically in one transaction.

What Topic Is Next → Next Topic 12.14: Interactive Transactions ($transaction(async (tx) => ...))

12.14 Interactive Transactions ($transaction(async (tx) => ...)) [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `await prisma.$transaction(async (tx) => { ... } • { maxWait: 3000 • Timeout: 5000 • IsolationLevel: ... })` • Production Backend Pitfall & Guardrail (Interactive Transactions ($transaction(async (tx) => ...))) • Hands-On TypeScript / Node.js Implementation (Interactive Transactions ($transaction(async (tx) => ...))) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Interactive Transactions ($transaction(async (tx) =>...

Purpose: Transferring stock between two warehouses: lock/check source -> decrement source -> increment destination -> write 2 ledger entries atomically via `tx`!

What Topic Is Next → Next Topic 12.15: Database Migrations (migrate dev vs migrate deploy)

12.15 Database Migrations (migrate dev vs migrate deploy) [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Generating reviewable SQL files in `prisma/migrations/` via `npx prisma migrate dev --name ...` • Reviewing destructive `DROP` warnings BEFORE committing • Running non-interactive **`npx prisma migrate deploy`** in CI/production (NEVER `migrate dev` or `db push` in prod!)! • Production Backend Pitfall & Guardrail (Database Migrations (migrate dev vs migrate deploy)) • Hands-On TypeScript / Node.js Implementation (Database Migrations (migrate dev vs migrate deploy)) • Database Migrations (migrate dev vs migrate deploy) — Latency, Throughput & Interview Trade-Offs • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Database Migrations (migrate dev vs migrate deploy)

Purpose: Executing Expand-and-Contract zero-downtime column renames across multi-step SQL migrations.

What Topic Is Next → Next Topic 12.16: Idempotent Database Seeding (prisma/seed.ts)

12.16 Idempotent Database Seeding (prisma/seed.ts) [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Writing re-runnable `upsert`-based seed scripts (`npx prisma db seed`) populating reference roles • Warehouses • Sample SKUs without throwing duplicate errors on second run • Production Backend Pitfall & Guardrail (Idempotent Database Seeding (prisma/seed.ts)) • Hands-On TypeScript / Node.js Implementation (Idempotent Database Seeding (prisma/seed.ts)) • Idempotent Database Seeding (prisma/seed.ts) — Latency, Throughput & Interview Trade-Offs • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Idempotent Database Seeding (prisma/seed.ts)

Purpose: Allowing any developer or CI job to run `prisma migrate deploy && prisma db seed` cleanly multiple times.

What Topic Is Next → Next Topic 12.17: Connection Management and Pool Sizing

12.17 Connection Management and Pool Sizing [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Configuring `?connection_limit=10&pool_timeout=10&connect_timeout=5` in `DATABASE_URL` • Matching `(num_containers * connection_limit) < postgres max_connections` • `PgBouncer` transaction-mode (`?pgbouncer=true`) • Production Backend Pitfall & Guardrail (Connection Management and Pool Sizing) • Hands-On TypeScript / Node.js Implementation (Connection Management and Pool Sizing) • Connection Management and Pool Sizing — Latency, Throughput & Interview Trade-Offs • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Connection Management

Purpose: Preventing PostgreSQL connection exhaustion when running Express API + BullMQ Worker containers simultaneously.

What Topic Is Next → Next Topic 12.18: Query Optimization, $queryRaw Safe Tagged Templates & Logging

12.18 Query Optimization, $queryRaw Safe Tagged Templates & Logging [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Enabling slow-query event logging (`prisma.$on("query" • E => { if (e.duration > 100) ... })`) • Using **`prisma.$queryRaw\`SELECT ... WHERE id = \${id} FOR UPDATE\`** (safe parameterized `$1`!) • NEVER using `$queryRawUnsafe`! • Production Backend Pitfall & Guardrail (Query Optimization, $queryRaw Safe Tagged Templates & Logging) • Hands-On TypeScript / Node.js Implementation (Query Optimization, $queryRaw Safe Tagged Templates & Logging) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Query Optimization

Purpose: Dropping down cleanly to `$queryRaw\`...\`` for `FOR UPDATE` row locks, Window Functions, or `EXPLAIN` while retaining 100% SQL-injection immunity!

What Topic Is Next → Next Topic 12.19: Repository and Service Patterns with Prisma

12.19 Repository and Service Patterns with Prisma [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Passing optional `tx?: Prisma.TransactionClient` (defaulting to `this.prisma`) into repository methods so Services can compose multiple repositories inside ONE ACID transaction cleanly! • Production Backend Pitfall & Guardrail (Repository and Service Patterns with Prisma) • Hands-On TypeScript / Node.js Implementation (Repository and Service Patterns with Prisma) • Repository and Service Patterns with Prisma — Latency, Throughput & Interview Trade-Offs • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Repository • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards • Diagnostics, Profiling & Telemetry: structured telemetry, profiler traces & CI checks

Purpose: Coordinating `stockRepo.decrement(sku, qty, tx)` and `ledgerRepo.recordMovement(entry, tx)` inside one `InventoryService` transaction!

What Topic Is Next → Next Topic 12.20: Advanced TypeScript Integration with Prisma

12.20 Advanced TypeScript Integration with Prisma [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Using `Prisma.InventoryStockGetPayload<{ include: { warehouse: true } }>` • `Prisma.PrismaClientKnownRequestError` (`err.code === "P2002"` unique • `"P2025"` not found • `"P2003"` FK) • Production Backend Pitfall & Guardrail (Advanced TypeScript Integration with Prisma) • Hands-On TypeScript / Node.js Implementation (Advanced TypeScript Integration with Prisma) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Advanced TypeScript Integration with Prisma

Purpose: Translating `P2002` (`409 Conflict`), `P2025` (`404 Not Found`), and `P2003` (`409 Referenced Record`) cleanly in one error mapper!

What Topic Is Next → Next Topic 12.21: Database Integration Testing Strategies

12.21 Database Integration Testing Strategies [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Testing Repositories & APIs against real ephemeral Docker PostgreSQL + MongoDB instances (never mocking SQL queries that pass in mocks • Fail on real DB constraints!) • Fast `TRUNCATE ... RESTART IDENTITY CASCADE` between suites • Production Backend Pitfall & Guardrail (Database Integration Testing Strategies) • Hands-On TypeScript / Node.js Implementation (Database Integration Testing Strategies) • Database Integration Testing Strategies — Latency, Throughput & Interview Trade-Offs • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Database Integration Testing Strategies

Purpose: Catching real `CHECK (qty_on_hand >= 0)`, FK, and unique index bugs in CI before production!

What Topic Is Next → Next Topic 12.22: Comparing Prisma, Mongoose and Direct SQL (Polyglot Synthesis)

12.22 Comparing Prisma, Mongoose and Direct SQL (Polyglot Synthesis) [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: When to use **Prisma** (type-safe relational CRUD/migrations) • **Direct Parameterized SQL / `$queryRaw`** (complex CTEs • Window Functions • `FOR UPDATE SKIP LOCKED` • Production Backend Pitfall & Guardrail (Comparing Prisma, Mongoose and Direct SQL (Polyglot Synthesis)) • Hands-On TypeScript / Node.js Implementation (Comparing Prisma, Mongoose and Direct SQL (Polyglot Synthesis)) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Comparing Prisma

Purpose: Building **Project 3: Multi-Database Inventory Management Backend** combining MongoDB Product Catalog (`sku` master specs) + PostgreSQL Inventory & Ledger (`ACID` stock m...

What Topic Is Next → Module 12 Capstone (Project 3: Multi-Database Inventory Management Backend) → Next Topic 13.1: Authentication vs Authorization

Module 12 Hands-On Capstone Project (100 Marks): Project 3: Multi-Database Inventory Management Backend — Real-world commerce and supply-chain backends frequently combine **polyglot persistence**: document databases excel at flexible, deeply varied product specifications (`laptops` have RAM/CPU,...

MODULE 13 OF 24 • STAGE 4 • 25 LIVE + LAB HOURS • 22 TOPICS (176 SUBTOPICS) • 1 CAPSTONE PROJECT (100 MARKS)

Module 13: Authentication and Authorization

Engineer production-grade Authentication and Authorization in Node.js & TypeScript — Argon2id/bcrypt password hashing, email verification, SHA-256 reset tokens, short-lived JWT access + rotated refresh token families, Redis session cookies, OAuth 2.0/OIDC, RBAC, granular permissions, and resource-ownership BOLA guards.

Objectives: ✓ Hash passwords safely using `Argon2id` / `bcrypt` and implement enumeration-safe login, email verification, and one-time password resets. • ✓ Implement short-lived JWT Access Tokens (`15m`) + rotated `HttpOnly` Refresh Tokens (`7d`) with reuse-detection family revocation. • ✓ Contrast stateless JWTs against Redis-backed stateful sessions and integrate OAuth 2.0 (`PKCE`) / Google OIDC login. • ✓ Enforce Role-Based (`RBAC`), Permission-Based, and Resource-Ownership (`BOLA`) authorization cleanly via Express middleware.

13.1 Authentication vs Authorization [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `Authentication (AuthN)` = verifying identity (`"Who are you?"` -> `401 Unauthorized` when missing/invalid) vs `Authorization (AuthZ)` = verifying permissions (`"What are you allowed to do?"` -> `403 Forbidden`) • Production Backend Pitfall & Guardrail (Authentication vs Authorization) • Hands-On TypeScript / Node.js Implementation (Authentication vs Authorization) • Authentication vs Authorization — Latency, Throughput & Interview Trade-Offs • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Authentication vs Authorization • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards • Diagnostics, Profiling & Telemetry: structured telemetry, profiler traces & CI checks

Purpose: Keeping `authenticate` middleware (populates `req.user`) cleanly separate from `authorize("orders:refund")` middleware.

What Topic Is Next → Next Topic 13.2: User Registration and Enumeration-Safe Login

13.2 User Registration and Enumeration-Safe Login [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Canonicalizing email (`trim().toLowerCase()`) • Constant-message `"Invalid email or password"` responses • Performing a dummy hash compare when user is not found to prevent timing-based email enumeration! • Production Backend Pitfall & Guardrail (User Registration and Enumeration-Safe Login) • Hands-On TypeScript / Node.js Implementation (User Registration and Enumeration-Safe Login) • User Registration and Enumeration-Safe Login — Latency, Throughput & Interview Trade-Offs • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for User Registration

Purpose: Preventing attackers from probing which corporate emails exist by comparing `2ms` (no user) vs `120ms` (`argon2.verify`) response times!

What Topic Is Next → Next Topic 13.3: Password Hashing Fundamentals

13.3 Password Hashing Fundamentals [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Why fast cryptographic hashes (`SHA-256` runs billions/sec on GPUs!) are catastrophic for passwords vs slow • Salted • Adaptive key-derivation functions (`Argon2id` • `bcrypt` • Production Backend Pitfall & Guardrail (Password Hashing Fundamentals) • Hands-On TypeScript / Node.js Implementation (Password Hashing Fundamentals) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Password Hashing Fundamentals

Purpose: Ensuring every user password automatically receives a unique cryptographic random salt embedded in the encoded hash string.

What Topic Is Next → Next Topic 13.4: bcrypt and Argon2id in Node.js

13.4 bcrypt and Argon2id in Node.js [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: OWASP #1 recommended **`argon2` (`argon2id` • `memoryCost: 65536`)** vs `bcrypt` (`cost = 12`) • Async threadpool execution (NEVER `hashSync`/`compareSync`!) • `bcrypt`'s silent `72-byte` UTF-8 truncation limit! • Production Backend Pitfall & Guardrail (bcrypt and Argon2id in Node.js) • Hands-On TypeScript / Node.js Implementation (bcrypt and Argon2id in Node.js) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for bcrypt

Purpose: Using async `await argon2.hash(pw, { type: argon2.argon2id })` without blocking the Node.js event loop.

What Topic Is Next → Next Topic 13.5: Password Reset Workflows

13.5 Password Reset Workflows [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Generating `32-byte` CSPRNG token (`crypto.randomBytes(32).toString("hex")`) • Emailing raw token in link • Storing ONLY `SHA-256(token)` + `expiresAt` (`15m`) + `usedAt` in DB • Revoking all active sessions upon reset! • Production Backend Pitfall & Guardrail (Password Reset Workflows) • Hands-On TypeScript / Node.js Implementation (Password Reset Workflows) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Password Reset Workflows

Purpose: Ensuring a read-only database backup leak STILL cannot compromise active password-reset tokens because only `SHA-256(token)` is stored!

What Topic Is Next → Next Topic 13.6: Email Verification Workflows

13.6 Email Verification Workflows [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Issuing hashed verification tokens on signup • Rate-limiting resend requests (`1 / 60s`) • Marking `emailVerifiedAt = NOW()` • Gating sensitive actions on verified accounts • Production Backend Pitfall & Guardrail (Email Verification Workflows) • Hands-On TypeScript / Node.js Implementation (Email Verification Workflows) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Email Verification Workflows

Purpose: Blocking unverified bot accounts from creating orders or triggering outbound notifications.

What Topic Is Next → Next Topic 13.7: JWT Fundamentals

13.7 JWT Fundamentals [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Compact URL-safe `base64url(header).base64url(payload).base64url(signature)` tokens — **SIGNED (tamper-evident) • NOT ENCRYPTED** (anyone can base64-decode payload!) • Production Backend Pitfall & Guardrail (JWT Fundamentals) • Hands-On TypeScript / Node.js Implementation (JWT Fundamentals) • JWT Fundamentals — Latency, Throughput & Interview Trade-Offs • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for JWT Fundamentals • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards

Purpose: Stateless signature verification across horizontally scaled API servers without hitting PostgreSQL on every read.

What Topic Is Next → Next Topic 13.8: JWT Structure and Strict Signature Verification

13.8 JWT Structure and Strict Signature Verification [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Symmetric `HS256` (`>= 256-bit` high-entropy secret) vs asymmetric `RS256`/`EdDSA` (Auth Service signs with private key • Microservices verify with public key!) • Pinning `algorithms: ["HS256"]` + `issuer` + `audience`! • Production Backend Pitfall & Guardrail (JWT Structure and Strict Signature Verification) • Hands-On TypeScript / Node.js Implementation (JWT Structure and Strict Signature Verification) • JWT Structure and Strict Signature Verification — Latency, Throughput & Interview Trade-Offs • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for JWT Structure

Purpose: Defending against `alg: "none"` bypasses and `RS256 -> HS256` algorithm-confusion attacks by hardcoding `algorithms` in `jwt.verify()`!

What Topic Is Next → Next Topic 13.9: Access Tokens and Refresh Tokens Architecture

13.9 Access Tokens and Refresh Tokens Architecture [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Pairing short-lived stateless `Access Token` (`10–15 min` in `Authorization: Bearer`) with long-lived revocable `Refresh Token` (`7–30 days` in `HttpOnly` cookie • `SHA-256` hash persisted in DB/Redis) • Production Backend Pitfall & Guardrail (Access Tokens and Refresh Tokens Architecture) • Hands-On TypeScript / Node.js Implementation (Access Tokens and Refresh Tokens Architecture) • Access Tokens and Refresh Tokens Architecture — Latency, Throughput & Interview Trade-Offs • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Access Tokens • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards

Purpose: Keeping 99% of API calls fast & stateless while retaining instant server-side session revocation at the next 15-minute refresh!

What Topic Is Next → Next Topic 13.10: Token Expiration, Rotation and Theft Reuse Detection

13.10 Token Expiration, Rotation and Theft Reuse Detection [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: On `POST /auth/refresh`: verify presented refresh token -> if already marked `replacedByTokenId != null` • **THEFT DETECTED: revoke ALL tokens in `familyId` immediately!** -> else mark used & issue fresh pair! • Production Backend Pitfall & Guardrail (Token Expiration, Rotation and Theft Reuse Detection) • Hands-On TypeScript / Node.js Implementation (Token Expiration, Rotation and Theft Reuse Detection) • Token Expiration, Rotation and Theft Reuse Detection — Latency, Throughput & Interview Trade-Offs • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Token Expiration • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards

Purpose: Automatically kicking out both attacker and victim if a stolen refresh token is replayed!

What Topic Is Next → Next Topic 13.11: Immediate Token Revocation Strategies

13.11 Immediate Token Revocation Strategies [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Combining short `15m` access TTL with a lightweight Redis denylist `SET revoked:jti:<jti> 1 EX <remaining_ttl>` (checked in `< 0.5ms`) for critical immediate logouts/bans + `tokenVersion` bump on password change! • Production Backend Pitfall & Guardrail (Immediate Token Revocation Strategies) • Hands-On TypeScript / Node.js Implementation (Immediate Token Revocation Strategies) • Immediate Token Revocation Strategies — Latency, Throughput & Interview Trade-Offs • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Immediate Token Revocation Strategies • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards • Diagnostics, Profiling & Telemetry: structured telemetry, profiler traces & CI checks

Purpose: Invalidating a terminated employee's `15m` access token within `10ms` across all microservices via Redis `jti` check.

What Topic Is Next → Next Topic 13.12: Session-Based Authentication with Redis

13.12 Session-Based Authentication with Redis [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Opaque CSPRNG `sessionId` cookie mapped to Redis JSON (`sess:<sid>`) • Sliding TTL renewal • Session fixation prevention (**regenerate session ID on login!**) • Instant `DEL sess:<sid>` logout • Production Backend Pitfall & Guardrail (Session-Based Authentication with Redis) • Hands-On TypeScript / Node.js Implementation (Session-Based Authentication with Redis) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Session-Based Authentication with Redis

Purpose: Choosing simple, instantly revocable Redis sessions for server-rendered/first-party web apps.

What Topic Is Next → Next Topic 13.13: Cookie-Based Authentication Mechanics

13.13 Cookie-Based Authentication Mechanics [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Automatic browser `Cookie` attachment vs explicit `Authorization` header • Scoping refresh cookies strictly to `Path=/api/v1/auth` so every `/api/v1/products` call doesn't even transmit the refresh cookie over the wire! • Production Backend Pitfall & Guardrail (Cookie-Based Authentication Mechanics) • Hands-On TypeScript / Node.js Implementation (Cookie-Based Authentication Mechanics) • Cookie-Based Authentication Mechanics — Latency, Throughput & Interview Trade-Offs • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Cookie-Based Authentication Mechanics • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards

Purpose: Minimizing refresh-cookie exposure surface by restricting `Path=/api/v1/auth/refresh`.

What Topic Is Next → Next Topic 13.14: Secure Cookie Configuration (__Host- Prefix & Flags)

13.14 Secure Cookie Configuration (__Host- Prefix & Flags) [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `HttpOnly: true` • `Secure: true` • `SameSite: "lax" | "strict"` • The **`__Host-`** cookie name prefix (`__Host-rt` forces browser enforcement of `Secure` • Production Backend Pitfall & Guardrail (Secure Cookie Configuration (__Host- Prefix & Flags)) • Hands-On TypeScript / Node.js Implementation (Secure Cookie Configuration (__Host- Prefix & Flags)) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Secure Cookie Configuration (__Host- Prefix & Flags)

Purpose: Preventing a compromised `blog.example.com` subdomain from overwriting auth cookies on `api.example.com`!

What Topic Is Next → Next Topic 13.15: OAuth 2.0 Fundamentals (Authorization Code + PKCE)

13.15 OAuth 2.0 Fundamentals (Authorization Code + PKCE) [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Roles (`Resource Owner` • `Client` • `Authorization Server` • `Resource Server`) • Production Backend Pitfall & Guardrail (OAuth 2.0 Fundamentals (Authorization Code + PKCE)) • Hands-On TypeScript / Node.js Implementation (OAuth 2.0 Fundamentals (Authorization Code + PKCE)) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for OAuth 2.0 Fundamentals (Authorization Code + PKCE)

Purpose: Delegating login to Google/GitHub without ever handling third-party passwords on your server.

What Topic Is Next → Next Topic 13.16: OpenID Connect (OIDC) Overview

13.16 OpenID Connect (OIDC) Overview [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Why OAuth 2.0 alone is an *authorization* protocol vs **OIDC** (`openid email profile` scopes) adding standardized identity `id_token` JWTs verified against provider `/.well-known/openid-configuration` `JWKS` keys! • Production Backend Pitfall & Guardrail (OpenID Connect (OIDC) Overview) • Hands-On TypeScript / Node.js Implementation (OpenID Connect (OIDC) Overview) • OpenID Connect (OIDC) Overview — Latency, Throughput & Interview Trade-Offs • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for OpenID Connect (OIDC) Overview • Production Hardening & Security: input validation, least-privilege access &

OWASP safeguards • Diagnostics, Profiling & Telemetry: structured telemetry, profiler traces & CI checks

Purpose: Validating `iss`, `aud` (your `GOOGLE_CLIENT_ID`!), `exp`, and `email_verified === true` on OIDC `id_token`s.

What Topic Is Next → Next Topic 13.17: Google Login Backend Integration Fundamentals

13.17 Google Login Backend Integration Fundamentals [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Redirecting to Google Auth URL (`state` • `code_challenge`) -> exchanging `code` server-to-server -> verifying `id_token` via `google-auth-library` / `jose` JWKS -> upserting `oauth_accounts(provider • ProviderAccountId)` -> issuing your own app session/JWT! • Production Backend Pitfall & Guardrail (Google Login Backend Integration Fundamentals) • Hands-On TypeScript / Node.js Implementation (Google Login Backend Integration Fundamentals) • Google Login Backend Integration Fundamentals — Latency, Throughput & Interview Trade-Offs • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Google Login Backend Integration Fundamentals

Purpose: Keying linked accounts by immutable Google `sub` (`providerAccountId`), NOT mutable email addresses!

What Topic Is Next → Next Topic 13.18: Role-Based Access Control (RBAC)

13.18 Role-Based Access Control (RBAC) [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Modelling `CUSTOMER` • `SELLER` • `SUPPORT` • `ADMIN` roles • Production Backend Pitfall & Guardrail (Role-Based Access Control (RBAC)) • Hands-On TypeScript / Node.js Implementation (Role-Based Access Control (RBAC)) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Role-Based Access Control (RBAC)

Purpose: Restricting `/api/v1/admin/coupons` strictly to `ADMIN` users while keeping catalog routes public.

What Topic Is Next → Next Topic 13.19: Permission-Based (Fine-Grained) Authorization

13.19 Permission-Based (Fine-Grained) Authorization [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Decoupling code from role renames: Roles map to **Permissions** (`orders:read` • `orders:refund` • `products:publish`) • Routes check `requirePermissions("orders:refund")`! • Production Backend Pitfall & Guardrail (Permission-Based (Fine-Grained) Authorization) • Hands-On TypeScript / Node.js Implementation (Permission-Based (Fine-Grained) Authorization) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Permission-Based (Fine-Grained) Authorization

Purpose: Introducing a new `FINANCE_AUDITOR` role with `["orders:read", "refunds:approve"]` in DB config with ZERO route code edits!

What Topic Is Next → Next Topic 13.20: Resource-Level Ownership Authorization (Preventing BOLA / IDOR)

13.20 Resource-Level Ownership Authorization (Preventing BOLA / IDOR) [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: **OWASP API1 #1 Vulnerability (`Broken Object Level Authorization`)**: verifying `order.userId === req.user.sub || req.user.role === "ADMIN"` inside the query/policy on EVERY `:id` endpoint! • Production Backend Pitfall & Guardrail (Resource-Level Ownership Authorization (Preventing BOLA / IDOR)) • Hands-On TypeScript / Node.js Implementation (Resource-Level Ownership Authorization (Preventing BOLA / IDOR)) • Resource-Level Ownership Authorization (Preventing BOLA / IDOR) — Latency, Throughput & Interview Trade-Offs • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Resource-Level Ownership Authorization (Preventing B... • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards • Diagnostics, Profiling & Telemetry: structured telemetry, profiler traces & CI checks

Purpose: Scoping `prisma.order.findFirst({ where: { id: orderId, userId: currentUserId } })` so Customer A can NEVER view or cancel Customer B's order!

What Topic Is Next → Next Topic 13.21: Multi-Role and Multi-Tenant Application Design

13.21 Multi-Role and Multi-Tenant Application Design [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Users holding multiple roles or workspace memberships (`organization_members(orgId • UserId • Role)`) • Scoping active `X-Organization-Id` verified against DB/cache membership • Production Backend Pitfall & Guardrail (Multi-Role and Multi-Tenant Application Design) • Hands-On TypeScript / Node.js Implementation (Multi-Role and Multi-Tenant Application Design) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Multi-Role

Purpose: Allowing a Seller to manage multiple merchant storefronts with isolated inventory and staff roles.

What Topic Is Next → Next Topic 13.22: Protected Routes and Composable Auth Middleware Pipeline

13.22 Protected Routes and Composable Auth Middleware Pipeline [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Chaining `authenticate -> requireVerifiedEmail -> requirePermissions("products:write")` cleanly with 100% TypeScript narrowing • Production Backend Pitfall & Guardrail (Protected Routes and Composable Auth Middleware Pipeline) • Hands-On TypeScript / Node.js Implementation (Protected Routes and Composable Auth Middleware Pipeline) • Protected Routes and Composable Auth Middleware Pipeline — Latency, Throughput & Interview Trade-Offs • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Protected Routes • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards • Diagnostics, Profiling & Telemetry: structured telemetry, profiler traces & CI checks

Purpose: Auditing security posture of your entire API by reading declarative route definitions in `*.routes.ts`.

What Topic Is Next → Module 13 Capstone (Week 13 Lab Build: Argon2id + Rotated Refresh-Token Family & RBAC/BOLA Guard Engine) → Next Topic 14.1: OWASP API Security Top 10 Overview

Module 13 Hands-On Capstone Project (100 Marks): Week 13 Lab Build: Argon2id + Rotated Refresh-Token Family & RBAC/BOLA Guard Engine — Build a standalone Express + TypeScript authentication core implementing `Argon2id` password hashing, `15m` JWT access tokens, `HttpOnly` rotated refresh-token families with SHA-256 DB stora...

MODULE 14 OF 24 • STAGE 4 • 25 LIVE + LAB HOURS • 24 TOPICS (192 SUBTOPICS) • 1 CAPSTONE PROJECT (100 MARKS)

Module 14: Backend Security and OWASP

Harden Node.js & Express APIs systematically against the OWASP API Security Top 10 — preventing SQLi, NoSQLi, XSS, CSRF, SSRF, mass assignment, brute-force attacks, malicious uploads, prototype pollution, vulnerable dependencies, PII log leaks, and excessive privileges.

Objectives: ✓ Mitigate all OWASP API Security Top 10 risks with concrete Express, TypeScript, Prisma, Mongoose, and Redis controls. • ✓ Configure `helmet`, strict `cors`, tiered rate limiting, brute-force lockouts, magic-byte upload checks, and payload caps. • ✓ Eliminate SQL injection, NoSQL operator injection, Prototype Pollution, `ReDoS`, and secret/PII leakage in logs and errors. • ✓ Maintain append-only security audit trails and dependency vulnerability scanning in CI.

14.1 OWASP API Security Top 10 Overview [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: API1 BOLA • API2 Broken Auth • API3 Broken Object Property Level Auth (Mass Assignment & Excessive Data Exposure!) • API4 Unrestricted Resource Consumption • Production Backend Pitfall & Guardrail (OWASP API Security Top 10 Overview) • Hands-On TypeScript / Node.js Implementation (OWASP API Security Top 10 Overview) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for OWASP API Security Top 10 Overview

Purpose: Using API3 to catch BOTH incoming mass assignment (`isAdmin: true`) AND outgoing over-exposure (`passwordHash` in JSON)!

What Topic Is Next → Next Topic 14.2: Input Validation and Sanitization

14.2 Input Validation and Sanitization [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Allowlist validation (strict types • Lengths • Regexes • Enums via Zod) OVER blocklist string stripping • Production Backend Pitfall & Guardrail (Input Validation and Sanitization) • Hands-On TypeScript / Node.js Implementation (Input Validation and Sanitization) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Input Validation

Purpose: Rejecting unexpected keys and oversized strings at the HTTP edge before business logic runs.

What Topic Is Next → Next Topic 14.3: SQL Injection Prevention

14.3 SQL Injection Prevention [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Why parameterized placeholders (`$1 • $2` in `pg` • Prisma ORM methods • `prisma.$queryRaw\`...\`` tagged templates) separate SQL AST compilation from data values — making SQLi mathematically impossible — plus whitelisting dynamic `ORDER BY` identifiers! • Production Backend Pitfall & Guardrail (SQL Injection Prevention) • Hands-On TypeScript / Node.js Implementation (SQL Injection Prevention) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for SQL Injection Prevention

Purpose: Safely supporting dynamic sort columns via a strict constant `Record<SortKey, string>` map.

What Topic Is Next → Next Topic 14.4: NoSQL Injection Prevention

14.4 NoSQL Injection Prevention [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Blocking JSON operator objects (`{ "$ne": null }` • `{ "$gt": "" }` • `$where` JS execution!) via Zod `z.string()` validation + `mongoose.set("sanitizeFilter" • True)` • Production Backend Pitfall & Guardrail (NoSQL Injection Prevention) • Hands-On TypeScript / Node.js Implementation (NoSQL Injection Prevention) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for NoSQL Injection Prevention

Purpose: Preventing authentication bypasses on MongoDB login and password-reset endpoints.

What Topic Is Next → Next Topic 14.5: Cross-Site Scripting (XSS) Fundamentals for Backend Engineers

14.5 Cross-Site Scripting (XSS) Fundamentals for Backend Engineers [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Enforcing `Content-Type: application/json • Charset=utf-8` + `X-Content-Type-Options: nosniff` on APIs • Escaping HTML in server-generated emails/PDFs • `HttpOnly` cookies so XSS cannot read tokens • Production Backend Pitfall & Guardrail (Cross-Site Scripting (XSS) Fundamentals for Backend Engineers) • Hands-On TypeScript / Node.js Implementation (Cross-Site Scripting (XSS) Fundamentals for Backend Engineers) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Cross-Site Scripting (XSS) Fundamentals for Backend...

Purpose: HTML-escaping `{{userName}}` inside transactional welcome/invoice emails so `<img src=x onerror=...>` never renders!

What Topic Is Next → Next Topic 14.6: Cross-Site Request Forgery (CSRF) Defense

14.6 Cross-Site Request Forgery (CSRF) Defense [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: When CSRF applies (cookie-authenticated state-changing endpoints • NOT pure `Authorization: Bearer` header APIs!) • Defending cookie endpoints via `SameSite=Lax|Strict` + `Origin`/`Referer` verification + Double-Submit / Synchronizer CSRF tokens • Production Backend Pitfall & Guardrail (Cross-Site Request Forgery (CSRF) Defense) • Hands-On TypeScript / Node.js Implementation (Cross-Site Request Forgery (CSRF) Defense) • Cross-Site Request Forgery (CSRF) Defense — Latency, Throughput & Interview Trade-Offs • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Cross-Site Request Forgery (CSRF) Defense

Purpose: Validating `Origin` header matches your allowed frontend origin on `POST /api/v1/auth/refresh` (which uses an `HttpOnly` cookie!).

What Topic Is Next → Next Topic 14.7: Production CORS Configuration

14.7 Production CORS Configuration [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Dynamic `origin(reqOrigin • Cb)` checking an explicit `Set<string>` allowlist • `credentials: true` • Explicit `methods` & `allowedHeaders` • Production Backend Pitfall & Guardrail (Production CORS Configuration) • Hands-On TypeScript / Node.js Implementation (Production CORS Configuration) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Production CORS Configuration

Purpose: Caching `OPTIONS` preflights for 24 hours (`maxAge: 86400`) to cut browser double-request latency in half!

What Topic Is Next → Next Topic 14.8: Helmet Security Headers

14.8 Helmet Security Headers [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `Strict-Transport-Security` (`HSTS`) • `X-Content-Type-Options: nosniff` • `X-Frame-Options: DENY` • `Referrer-Policy: no-referrer` • Production Backend Pitfall & Guardrail (Helmet Security Headers) • Hands-On TypeScript / Node.js Implementation (Helmet Security Headers) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Helmet Security Headers

Purpose: Setting `default-src 'none'; frame-ancestors 'none'` on pure JSON API servers so browsers refuse to execute scripts or frame API responses.

What Topic Is Next → Next Topic 14.9: Rate Limiting Architecture

14.9 Rate Limiting Architecture [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Multi-layer rate limits: Nginx edge IP burst limit + Express Redis-backed per-user/per-IP sliding windows (`express-rate-limit` + `rate-limit-redis`) • Production Backend Pitfall & Guardrail (Rate Limiting Architecture) • Hands-On TypeScript / Node.js Implementation (Rate Limiting Architecture) • Rate Limiting Architecture — Latency, Throughput & Interview Trade-Offs • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Rate Limiting Architecture • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards • Diagnostics, Profiling & Telemetry: structured telemetry, profiler traces & CI checks

Purpose: Sharing accurate rate-limit counters across all 4 Node/Docker replicas via Redis.

What Topic Is Next → Next Topic 14.10: Brute-Force and Credential-Stuffing Protection

14.10 Brute-Force and Credential-Stuffing Protection [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Dual-key tracking on `/login`: (1) per-IP limit (`30 / 15m` protecting corporate NATs) • (2) per-`email` failure counter (`5 failed -> 15m cooldown` blocking distributed botnets rotating 10 • 000 IPs!) • Production Backend Pitfall & Guardrail (Brute-Force and Credential-Stuffing Protection) • Hands-On TypeScript / Node.js Implementation (Brute-Force and Credential-Stuffing Protection) • Brute-Force and Credential-Stuffing Protection — Latency, Throughput & Interview Trade-Offs • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Brute-Force

Purpose: Stopping a 5,000-IP botnet trying 1 password per IP against `ceo@company.com`!

What Topic Is Next → Next Topic 14.11: Password Security and Policy Standards

14.11 Password Security and Policy Standards [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: NIST SP 800-63B modern rules: length (`>= 8–12` chars up to `128`) • Allowing spaces/passphrases • Screening common breached passwords (`password123` • `qwerty12345`) • Production Backend Pitfall & Guardrail (Password Security and Policy Standards) • Hands-On TypeScript / Node.js Implementation (Password Security and Policy Standards) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Password Security

Purpose: Blocking the top 10,000 leaked passwords while encouraging strong memorable passphrases.

What Topic Is Next → Next Topic 14.12: Secure Token Storage (Hashing Server-Side Secrets)

14.12 Secure Token Storage (Hashing Server-Side Secrets) [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Hashing refresh tokens • API keys (`sk_live_...`) • OTPs • Reset tokens with `SHA-256` BEFORE storing in PostgreSQL/MongoDB — just like GitHub/Stripe show an API key ONCE • Production Backend Pitfall & Guardrail (Secure Token Storage (Hashing Server-Side Secrets)) • Hands-On TypeScript / Node.js Implementation (Secure Token Storage (Hashing Server-Side Secrets)) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Secure Token Storage (Hashing Server-Side Secrets)

Purpose: Storing `{ keyPrefix: "mk_live_8fa2", keyHash: sha256(fullKey) }` so a DB read breach never leaks usable API keys or refresh tokens!

What Topic Is Next → Next Topic 14.13: Secrets Management and Rotation

14.13 Secrets Management and Rotation [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Supporting **Key Ring Arrays** (`JWT_SECRETS = [currentKey • PreviousKey]`: sign with `keys[0]` • Verify trying `keys[0]` then `keys[1]`) so signing keys rotate with ZERO user logouts! • Production Backend Pitfall & Guardrail (Secrets Management and Rotation) • Hands-On TypeScript / Node.js Implementation (Secrets Management and Rotation) • Secrets Management and Rotation — Latency, Throughput & Interview Trade-Offs • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Secrets Management

Purpose: Rotating `JWT_SECRET` or `COOKIE_SECRET` seamlessly without kicking out 50,000 active sessions.

What Topic Is Next → Next Topic 14.14: Hardened Environment Configuration

14.14 Hardened Environment Configuration [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Enforcing minimum secret entropy (`JWT_SECRET.length >= 32`) • Forbidding dev defaults (`"changeme"` • `"secret"`) when `NODE_ENV === "production"` • Freezing config • Production Backend Pitfall & Guardrail (Hardened Environment Configuration) • Hands-On TypeScript / Node.js Implementation (Hardened Environment Configuration) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Hardened Environment Configuration

Purpose: Crashing immediately on startup if someone deploys to production with `JWT_SECRET=secret`!

What Topic Is Next → Next Topic 14.15: File Upload Security Architecture

14.15 File Upload Security Architecture [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Renaming every uploaded file to a random `UUID` (`never` using original `file.originalname` on disk!) • Storing outside webroot in private buckets • `Content-Disposition: attachment` • Separate cookieless download domain • Production Backend Pitfall & Guardrail (File Upload Security Architecture) • Hands-On TypeScript / Node.js Implementation (File Upload Security Architecture) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for File Upload Security Architecture

Purpose: Ensuring an uploaded `invoice.html` or `avatar.svg` (SVGs can contain `<script>` JS!!) can never execute scripts in your app origin!

What Topic Is Next → Next Topic 14.16: Magic-Byte MIME Type Validation

14.16 Magic-Byte MIME Type Validation [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Inspecting actual leading binary file signatures (`%PDF-` = `25 50 44 46` • PNG = `89 50 4E 47` • JPEG = `FF D8 FF`) rather than trusting spoofable `Content-Type` multipart headers! • Production Backend Pitfall & Guardrail (Magic-Byte MIME Type Validation) • Hands-On TypeScript / Node.js Implementation (Magic-Byte MIME Type Validation) • Magic-Byte MIME Type Validation — Latency, Throughput & Interview Trade-Offs • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Magic-Byte MIME Type Validation

Purpose: Blocking an executable/PHP/HTML script renamed to `photo.png` with spoofed `Content-Type: image/png`!

What Topic Is Next → Next Topic 14.17: Request Size, Parameter and Depth Limits

14.17 Request Size, Parameter and Depth Limits [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Bounding `express.json({ limit: "100kb" })` • URL query parameter count (`parameterLimit: 50`) • Multer `{ limits: { fileSize: 5 * 1024 * 1024 • Files: 3 } }` • Production Backend Pitfall & Guardrail (Request Size, Parameter and Depth Limits) • Hands-On TypeScript / Node.js Implementation (Request Size, Parameter and Depth Limits) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Request Size

Purpose: Preventing memory exhaustion and CPU hash-collision attacks on public endpoints.

What Topic Is Next → Next Topic 14.18: API Abuse, Enumeration and SSRF Prevention

14.18 API Abuse, Enumeration and SSRF Prevention [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Using UUIDv4/v7 external IDs instead of sequential `/invoices/1 • Protecting coupon/checkout business flows (`API6`) • Blocking private IP ranges (`127.0.0.0/8` • `10.0.0.0/8` • Production Backend Pitfall & Guardrail (API Abuse, Enumeration and SSRF Prevention) • Hands-On TypeScript / Node.js Implementation (API Abuse, Enumeration and SSRF Prevention) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for API Abuse

Purpose: Stopping attackers from scraping all invoice PDFs by incrementing integers or probing internal Redis/DB ports via webhook test URLs.

What Topic Is Next → Next Topic 14.19: Dependency Vulnerability Scanning

14.19 Dependency Vulnerability Scanning [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Running `npm audit --omit=dev --audit-level=high` in CI • Pinning lockfiles • Disabling untrusted install scripts (`--ignore-scripts` consideration) • Preferring native Node 22 APIs over micro-packages • Production Backend Pitfall & Guardrail (Dependency Vulnerability Scanning) • Hands-On TypeScript / Node.js Implementation (Dependency Vulnerability Scanning) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Dependency Vulnerability Scanning

Purpose: Catching known CVEs in transitive dependencies automatically on GitHub Actions PRs.

What Topic Is Next → Next Topic 14.20: Secure Logging and Redaction

14.20 Secure Logging and Redaction [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Configuring Pino `redact: { paths: [...] • Censor: "[REDACTED]" }` for `authorization` • `cookie` • `set-cookie` • Production Backend Pitfall & Guardrail (Secure Logging and Redaction) • Hands-On TypeScript / Node.js Implementation (Secure Logging and Redaction) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Secure Logging

Purpose: Guaranteeing centralized log aggregators never store plaintext Bearer tokens or user passwords.

What Topic Is Next → Next Topic 14.21: PII Handling and Field-Level Encryption

14.21 PII Handling and Field-Level Encryption [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Classifying PII (`PAN` • `taxId` • `bankAccountNumber`) • Encrypting high-sensitivity fields at rest with `node:crypto` `aes-256-gcm` (unique `12-byte` IV + `16-byte` authTag per value!) • Production Backend Pitfall & Guardrail (PII Handling and Field-Level Encryption) • Hands-On TypeScript / Node.js Implementation (PII Handling and Field-Level Encryption) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for PII Handling

Purpose: Encrypting seller payout account numbers at rest with authenticated `AES-256-GCM` while showing only last 4 digits in APIs.

What Topic Is Next → Next Topic 14.22: Immutable Security Audit Trails

14.22 Immutable Security Audit Trails [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Recording `security_audit_logs` (`actorUserId` • `action` • `targetType` • `targetId` • Production Backend Pitfall & Guardrail (Immutable Security Audit Trails) • Hands-On TypeScript / Node.js Implementation (Immutable Security Audit Trails) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Immutable Security Audit Trails

Purpose: Investigating `"Who changed Order #882 status to REFUNDED and from which IP?"` during incident response.

What Topic Is Next → Next Topic 14.23: Secure Error Responses in Production

14.23 Secure Error Responses in Production [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Logging full `{ err • Stack • Sql • RequestId }` internally to Pino while returning ONLY `{ code: "INTERNAL_ERROR" • Production Backend Pitfall & Guardrail (Secure Error Responses in Production) • Hands-On TypeScript / Node.js Implementation (Secure Error Responses in Production) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Secure Error Responses in Production

Purpose: Giving support engineers exact `requestId` lookup power while giving attackers zero database/file-path reconnaissance!

What Topic Is Next → Next Topic 14.24: Least-Privilege Access Across Stack Layers

14.24 Least-Privilege Access Across Stack Layers [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Running Node as UID `1000` (`node` user • Never `root`) • Connecting the app with a non-superuser PostgreSQL role (`SELECT • INSERT • Production Backend Pitfall & Guardrail (Least-Privilege Access Across Stack Layers) • Hands-On TypeScript / Node.js Implementation (Least-Privilege Access Across Stack Layers) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-

Ready Career Program Production Runtime Execution & Memory Lifecycle for Least-Privilege Access Across Stack Layers

Purpose: Containing blast radius so even a bug cannot drop schemas or modify container root filesystems.

What Topic Is Next → Module 14 Capstone (Week 14 Lab Build: OWASP API Top 10 Hardening Pack (Helmet, Zod, SSRF & Redis Rate Limiter)) → Next Topic 15.1: Advanced Validation Using Zod and Joi

Module 14 Hands-On Capstone Project (100 Marks): Week 14 Lab Build: OWASP API Top 10 Hardening Pack (Helmet, Zod, SSRF & Redis Rate Limiter) — Harden an Express API against NoSQL/SQL injection, mass assignment, prototype pollution, ReDoS, and private-IP SSRF while adding `Helmet`, strict origin allowlist `CORS`, and atomic Redis Lu...

MODULE 15 OF 24 • STAGE 4 • 25 LIVE + LAB HOURS • 24 TOPICS (192 SUBTOPICS) • 1 CAPSTONE PROJECT (100 MARKS)

Module 15: Advanced API Development and Backend Features

Build complete real-world backend product features — Zod/Joi request & response validation, cursor pagination, bulk operations, Multer + S3-compatible MinIO presigned object storage, Nodemailer SMTP templates, cron & background jobs, streaming CSV/Excel/PDF processing, and HMAC-signed idempotent webhooks.

Objectives: ✓ Validate and infer TypeScript types from a single source of truth using `Zod` (and maintain `Joi` schemas) across requests AND outbound responses. • ✓ Handle multipart uploads with `Multer` and offload large files via vendor-neutral S3-compatible (`MinIO`) Presigned URLs. • ✓ Implement transactional email templates, cluster-safe cron schedules, streaming CSV/Excel imports & exports, and server-side PDF invoices. • ✓ Build both inbound and outbound Webhooks with HMAC-SHA256 + timestamp replay verification, retries, and idempotency.

15.1 Advanced Validation Using Zod and Joi [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `Zod` (`z.object` • `.strict()`/`.strip()` • `.refine()` • `.transform()` • Production Backend Pitfall & Guardrail (Advanced Validation Using Zod and Joi) • Hands-On TypeScript / Node.js Implementation (Advanced Validation Using Zod and Joi) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Advanced Validation Using Zod

Purpose: Deriving `type CreateOrderDto = z.infer<typeof CreateOrderSchema>` so runtime validation and TypeScript types NEVER drift apart!

What Topic Is Next → Next Topic 15.2: Request Schema Validation Middleware (params, query, body)

15.2 Request Schema Validation Middleware (params, query, body) [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `z.coerce.number().int().min(1).max(100).default(20)` for query strings • `z.string().uuid()` for params • Formatting `ZodError.issues` into clean `422` field error arrays • Production Backend Pitfall & Guardrail (Request Schema Validation Middleware (params, query, body)) • Hands-On TypeScript / Node.js Implementation (Request Schema Validation Middleware (params, query, body)) • Request Schema Validation Middleware (params, query, body) — Latency, Throughput & Interview Trade-Offs • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Request Schema Validation Middleware (params, query,...

Purpose: Replacing `req.query` string soup with strongly typed `{ page: number; limit: number }` automatically!

What Topic Is Next → Next Topic 15.3: Outbound Response Validation

15.3 Outbound Response Validation [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Running `PublicUserDtoSchema.parse(data)` before `res.json()` (or in tests/staging) as a hard data-leakage firewall guaranteeing `passwordHash` or `secret` can NEVER leave the server! • Production Backend Pitfall & Guardrail (Outbound Response Validation) • Hands-On TypeScript / Node.js Implementation (Outbound Response Validation) • Outbound Response Validation — Latency, Throughput & Interview Trade-Offs • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Outbound Response Validation • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards • Diagnostics, Profiling & Telemetry: structured telemetry, profiler traces & CI checks

Purpose: Preventing OWASP `API3 Excessive Data Exposure` even if a developer accidentally spreads `...dbUser` in a service!

What Topic Is Next → Next Topic 15.4: Centralized API Error Mapping (Zod, Prisma, Mongoose, Multer, JWT)

15.4 Centralized API Error Mapping (Zod, Prisma, Mongoose, Multer, JWT) [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Unified error normalizer converting `ZodError` (`422`) • `Prisma P2002` (`409`) • `Mongoose 11000` (`409`) • `MulterError LIMIT_FILE_SIZE` (`413`) • Production Backend Pitfall & Guardrail (Centralized API Error Mapping (Zod, Prisma, Mongoose, Multer, JWT)) • Hands-On TypeScript / Node.js Implementation (Centralized API Error Mapping (Zod, Prisma, Mongoose, Multer, JWT)) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Centralized API Error Mapping (Zod, Prisma, Mongoose...

Purpose: Keeping Services and Controllers completely free of repetitive `try / catch` error-formatting boilerplate.

What Topic Is Next → Next Topic 15.5: API Versioning Lifecycle & Deprecation Headers

15.5 API Versioning Lifecycle & Deprecation Headers [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Router-level `/api/v1` vs `/api/v2` • RFC 8594 `Sunset` & `Deprecation: true` headers • Tracking per-version traffic metrics before retiring old versions • Production Backend Pitfall & Guardrail (API Versioning Lifecycle & Deprecation Headers) • Hands-On TypeScript / Node.js Implementation (API Versioning Lifecycle & Deprecation Headers) • API Versioning Lifecycle & Deprecation Headers — Latency, Throughput & Interview Trade-Offs • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for API Versioning Lifecycle

Purpose: Communicating a 90-day sunset window clearly to API consumers via standard HTTP headers.

What Topic Is Next → Next Topic 15.6: Offset Pagination vs Opaque Cursor Pagination

15.6 Offset Pagination vs Opaque Cursor Pagination [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Implementing tamper-checked base64url cursors `{ sortValue • Id }` supporting forward (`after`) pagination without `OFFSET` degradation on `10M`-row tables • Production Backend Pitfall & Guardrail (Offset Pagination vs Opaque Cursor Pagination) • Hands-On TypeScript / Node.js Implementation (Offset Pagination vs Opaque Cursor Pagination) • Offset Pagination vs Opaque Cursor Pagination — Latency, Throughput & Interview Trade-Offs • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Offset Pagination vs Opaque Cursor Pagination • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards

Purpose: Serving `/api/v1/orders?limit=25&after=eyJ...` at `< 5ms` regardless of scroll depth.

What Topic Is Next → Next Topic 15.7: Advanced Filtering and Multi-Field Sorting

15.7 Advanced Filtering and Multi-Field Sorting [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Supporting range operators (`price[gte]=1000&price[lte]=5000`) • Multi-select (`status=PAID • SHIPPED`) • Safe multi-sort (`sort=-createdAt • Production Backend Pitfall & Guardrail (Advanced Filtering and Multi-Field Sorting) • Hands-On TypeScript / Node.js Implementation (Advanced Filtering and Multi-Field Sorting) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Advanced Filtering

Purpose: Powering rich e-commerce product discovery filters in the Final Capstone.

What Topic Is Next → Next Topic 15.8: Bulk Insert and Bulk Update Patterns

15.8 Bulk Insert and Bulk Update Patterns [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `prisma.product.createMany({ data • SkipDuplicates })` • Chunking `5 • 000` rows into batches of `500` so single SQL packets stay fast • Production Backend Pitfall & Guardrail (Bulk Insert and Bulk Update Patterns) • Hands-On TypeScript / Node.js Implementation (Bulk Insert and Bulk Update Patterns) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Bulk Insert

Purpose: Updating 2,000 SKU prices in 4 batched `500`-row queries (`40ms` total) instead of 2,000 sequential `await update()` calls (`18 seconds`!)!

What Topic Is Next → Next Topic 15.9: File Upload with Multer

15.9 File Upload with Multer [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Configuring `multer` (`diskStorage` to `/tmp` with `UUID` filename vs `memoryStorage` ONLY for small `< 2 MB` avatars) • `fileFilter` • Strict `limits` • Cleaning up temp files in `finally`! • Production Backend Pitfall & Guardrail (File Upload with Multer) • Hands-On TypeScript / Node.js Implementation (File Upload with Multer) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for File Upload with Multer

Purpose: Accepting validated product images and CSV import files safely.

What Topic Is Next → Next Topic 15.10: Self-Hosted / S3-Compatible Object Storage Fundamentals (MinIO)

15.10 Self-Hosted / S3-Compatible Object Storage Fundamentals (MinIO) [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Why stateless Node containers NEVER store permanent uploads on local container disk (wiped on redeploy & invisible to sibling replicas!) -> storing objects in private S3-compatible buckets (`MinIO` in Docker / any S3-compatible provider) keyed by `{tenant}/{year}/{uuid}.ext` • Production Backend Pitfall & Guardrail (Self-Hosted / S3-Compatible Object Storage Fundamentals (MinIO)) • Hands-On TypeScript / Node.js Implementation (Self-Hosted / S3-Compatible Object Storage Fundamentals (MinIO)) • Self-Hosted / S3-Compatible Object Storage Fundamentals (MinIO) — Latency, Throughput & Interview Trade-Offs • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Self-Hosted / S3-Compatible Object Storage Fundament... • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards • Diagnostics, Profiling & Telemetry: structured telemetry, profiler traces & CI checks

Purpose: Running 100% self-hosted S3-compatible `MinIO` inside Docker Compose for product images and invoice PDFs.

What Topic Is Next → Next Topic 15.11: Presigned URL Concepts (Direct Upload & Time-Limited Download)

15.11 Presigned URL Concepts (Direct Upload & Time-Limited Download) [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Client requests `POST /uploads/presign` (Node checks Auth + quota + MIME/size conditions -> signs a `5-minute` `PUT` URL) -> Client uploads `50 MB` directly to Object Storage (zero Node RAM/bandwidth!) -> `GET` uses `5-minute` RBAC-checked presigned download URL! • Production Backend Pitfall & Guardrail (Presigned URL Concepts (Direct Upload & Time-Limited Download)) • Hands-On TypeScript / Node.js Implementation (Presigned URL Concepts (Direct Upload & Time-Limited Download)) • Presigned URL Concepts (Direct Upload & Time-Limited Download) — Latency, Throughput & Interview Trade-Offs • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Presigned URL Concepts (Direct Upload & Time-Limited... • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards • Diagnostics, Profiling & Telemetry: structured telemetry, profiler traces & CI checks

Purpose: Offloading heavy media transfer completely off your single-threaded Node.js event loop!

What Topic Is Next → Next Topic 15.12: Email Services and Transactional Templates

15.12 Email Services and Transactional Templates [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Designing clean HTML + plain-text fallback email templates (Welcome • Verify Email • Password Reset • Order Receipt) with HTML-escaped variables • Production Backend Pitfall & Guardrail (Email Services and Transactional Templates) • Hands-On TypeScript / Node.js Implementation (Email Services and Transactional Templates) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Email Services

Purpose: Rendering crisp, XSS-safe order confirmation receipts with line-item totals.

What Topic Is Next → Next Topic 15.13: SMTP Integration with Nodemailer

15.13 SMTP Integration with Nodemailer [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Configuring `nodemailer.createTransport({ pool: true • MaxConnections: 5 • ConnectionTimeout: 5000 })` • Testing locally against `Mailpit` / `MailHog` SMTP container (zero real spam during dev!) • Production Backend Pitfall & Guardrail (SMTP Integration with Nodemailer) • Hands-On TypeScript / Node.js Implementation (SMTP Integration with Nodemailer) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js

Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for SMTP Integration with Nodemailer

Purpose: Testing real SMTP delivery locally in `Mailpit` web UI (`localhost:8025`) while decoupling slow `800ms` SMTP handshakes from HTTP latency!

What Topic Is Next → Next Topic 15.14: End-to-End Password Reset Email Workflows

15.14 End-to-End Password Reset Email Workflows [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Combining rate limit -> hashed token DB write -> queued reset email -> token verification -> Argon2 password update -> session revocation -> confirmation security alert email! • Production Backend Pitfall & Guardrail (End-to-End Password Reset Email Workflows) • Hands-On TypeScript / Node.js Implementation (End-to-End Password Reset Email Workflows) • End-to-End Password Reset Email Workflows — Latency, Throughput & Interview Trade-Offs • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for End-to-End Password Reset Email Workflows • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards • Diagnostics, Profiling & Telemetry: structured telemetry, profiler traces & CI checks

Purpose: Sending a `"Your password was just changed"` security notification email immediately after reset completion.

What Topic Is Next → Next Topic 15.15: Scheduled Tasks Architecture

15.15 Scheduled Tasks Architecture [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Use cases: expiring unpaid `15-minute` inventory reservations • Nightly ledger reconciliations • Purging expired tokens • Refreshing materialized views • Production Backend Pitfall & Guardrail (Scheduled Tasks Architecture) • Hands-On TypeScript / Node.js Implementation (Scheduled Tasks Architecture) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Scheduled Tasks Architecture

Purpose: Releasing unpaid reserved checkout stock back to `qtyOnHand` automatically after 15 minutes.

What Topic Is Next → Next Topic 15.16: Cron Jobs and Multi-Instance Distributed Lock Safety

15.16 Cron Jobs and Multi-Instance Distributed Lock Safety [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Cron 5-field expressions (`*/5 * * * *`) • Explicit `timezone: "UTC"` • Why `node-cron` inside 4 API containers fires **4 TIMES** unless moved to BullMQ Repeatable Jobs or guarded by a Redis `SET NX PX` lock! • Production Backend Pitfall & Guardrail (Cron Jobs and Multi-Instance Distributed Lock Safety) • Hands-On TypeScript / Node.js Implementation (Cron Jobs and Multi-Instance Distributed Lock Safety) • Cron Jobs and Multi-Instance Distributed Lock Safety — Latency, Throughput & Interview Trade-Offs • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Cron Jobs

Purpose: Preventing 4 containers from sending 4 duplicate `"Daily Summary"` emails to every customer at 09:00 AM!

What Topic Is Next → Next Topic 15.17: Background Task Architecture Preview

15.17 Background Task Architecture Preview [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Returning `202 Accepted` + `{ jobId • StatusUrl: "/api/v1/jobs/..." }` for operations taking `> 500ms` (bulk imports • PDF generation • Webhook fan-out) • Production Backend Pitfall & Guardrail (Background Task Architecture Preview) • Hands-On TypeScript / Node.js Implementation (Background Task Architecture Preview) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Background Task Architecture Preview

Purpose: Keeping 99% of HTTP API endpoints `< 100ms` fast while long tasks report progress cleanly.

What Topic Is Next → Next Topic 15.18: Streaming CSV Import and Export (CSV Injection Defense!)

15.18 Streaming CSV Import and Export (CSV Injection Defense!) [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Streaming CSV rows via `csv-parse` / `fast-csv` + `pipeline()` in `O(1)` memory • Defending against **CSV Formula Injection** (prefixing leading `=` • `+` • `-` • Production Backend Pitfall & Guardrail (Streaming CSV Import and Export (CSV Injection Defense!)) • Hands-On TypeScript / Node.js Implementation (Streaming CSV Import and Export (CSV Injection Defense!)) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Streaming CSV Import

Purpose: Streaming 100,000 orders directly to `res` (`Content-Type: text/csv`) while neutralizing `=HYPERLINK(...)` spreadsheet attacks!

What Topic Is Next → Next Topic 15.19: Excel (.xlsx) Streaming Processing Fundamentals

15.19 Excel (.xlsx) Streaming Processing Fundamentals [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Using `exceljs` streaming `WorkbookWriter` / `WorkbookReader` instead of loading entire multi-sheet workbooks into DOM memory • Production Backend Pitfall & Guardrail (Excel (.xlsx) Streaming Processing Fundamentals) • Hands-On TypeScript / Node.js Implementation (Excel (.xlsx) Streaming Processing Fundamentals) • Excel (.xlsx) Streaming Processing Fundamentals — Latency, Throughput & Interview Trade-Offs • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Excel (.xlsx) Streaming Processing Fundamentals • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards • Diagnostics, Profiling & Telemetry: structured telemetry, profiler traces & CI checks

Purpose: Generating formatted multi-sheet financial reconciliation `.xlsx` exports without spiking V8 heap.

What Topic Is Next → Next Topic 15.20: PDF Generation Fundamentals (pdfkit)

15.20 PDF Generation Fundamentals (pdfkit) [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Generating lightweight vector GST tax invoices & receipts server-side using streaming `pdfkit` (`doc.pipe(writeStream)`) rather than spawning a `400 MB` headless Chromium browser! • Production Backend Pitfall & Guardrail (PDF Generation Fundamentals (pdfkit)) • Hands-On TypeScript / Node.js Implementation (PDF Generation Fundamentals (pdfkit)) • PDF Generation Fundamentals (pdfkit) — Latency, Throughput & Interview Trade-Offs • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for PDF Generation Fundamentals (pdfkit) • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards • Diagnostics, Profiling & Telemetry: structured telemetry, profiler traces & CI checks

Purpose: Generating a `25 KB` GST order invoice PDF in `12ms` using `pdfkit` inside a background worker.

What Topic Is Next → Next Topic 15.21: Webhook Creation (Outbound & Inbound Architecture)

15.21 Webhook Creation (Outbound & Inbound Architecture) [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Inbound webhooks (receiving payment events from Razorpay/Stripe) vs Outbound webhooks (notifying merchant URLs of `order.shipped` with `X-Webhook-Id` • `X-Webhook-Timestamp` • `X-Webhook-Signature`) • Production Backend Pitfall & Guardrail (Webhook Creation (Outbound & Inbound Architecture))

• Hands-On TypeScript / Node.js Implementation (Webhook Creation (Outbound & Inbound Architecture)) • Webhook Creation (Outbound & Inbound Architecture) — Latency, Throughput & Interview Trade-Offs • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Webhook Creation (Outbound & Inbound Architecture)

Purpose: Responding `200 OK` within `< 200ms` after verifying signature & persisting raw webhook event, then processing asynchronously!

What Topic Is Next → Next Topic 15.22: Webhook Signature Validation + Replay-Window Defense

15.22 Webhook Signature Validation + Replay-Window Defense [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Signing `${timestamp}.${rawBody}` with HMAC-SHA256 • Verifying `Math.abs(nowSec - timestamp) <= 300` (`5-minute` replay window!) • Comparing digests with `crypto.timingSafeEqual`! • Production Backend Pitfall & Guardrail (Webhook Signature Validation + Replay-Window Defense) • Hands-On TypeScript / Node.js Implementation (Webhook Signature Validation + Replay-Window Defense) • Webhook Signature Validation + Replay-Window Defense — Latency, Throughput & Interview Trade-Offs • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Webhook Signature Validation + Replay-Window Defense

Purpose: Blocking BOTH forged webhook payloads AND replayed captured valid webhooks older than 5 minutes!

What Topic Is Next → Next Topic 15.23: Webhook Retries and Consumer Idempotency

15.23 Webhook Retries and Consumer Idempotency [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Why payment gateways deliver webhooks **At-Least-Once** (same `payment.captured` event can arrive 3 times!) • Guaranteeing exactly-once business effect via `UNIQUE (provider • Provider_event_id)` inside the same DB transaction as order status update! • Production Backend Pitfall & Guardrail (Webhook Retries and Consumer Idempotency) • Hands-On TypeScript / Node.js Implementation (Webhook Retries and Consumer Idempotency) • Webhook Retries and Consumer Idempotency — Latency, Throughput & Interview Trade-Offs • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Webhook Retries

Purpose: Ensuring 3 duplicate `payment.captured` webhook deliveries mark Order `#901` `PAID` and ship items **EXACTLY ONCE**!

What Topic Is Next → Next Topic 15.24: API Documentation and Contract Governance

15.24 API Documentation and Contract Governance [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Generating OpenAPI 3.1 JSON directly from `Zod` schemas (`zod-to-openapi`) • Exporting client SDK types • Running CI diffs catching accidental breaking contract changes • Production Backend Pitfall & Guardrail (API Documentation and Contract Governance) • Hands-On TypeScript / Node.js Implementation (API Documentation and Contract Governance) • API Documentation and Contract Governance — Latency, Throughput & Interview Trade-Offs • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for API Documentation

Purpose: Guaranteeing `/api/docs` is always 100% truthful because it compiles from the exact runtime Zod schemas.

What Topic Is Next → Module 15 Capstone (Week 15 Lab Build: MinIO Presigned Upload Service + Supertest & Testcontainers Suite) → Next Topic 16.1: Software Testing Fundamentals & the Backend Testing Trophy

Module 15 Hands-On Capstone Project (100 Marks): Week 15 Lab Build: MinIO Presigned Upload Service + Supertest & Testcontainers Suite — Build a secure file upload & invoice attachment service verifying magic-byte file signatures (`file-type`) and S3-compatible `MinIO` presigned `PUT`/`GET` URLs — backed by an `>= 85%` `Vites...

MODULE 16 OF 24 • STAGE 4 • 25 LIVE + LAB HOURS • 25 TOPICS (200 SUBTOPICS) • 1 CAPSTONE PROJECT (100 MARKS)

Module 16: Testing and Quality Engineering

Ship bug-resistant Node.js & TypeScript services with automated Quality Engineering — Jest/Vitest, Supertest HTTP integration testing, TDD, real-DB test isolation, external adapter mocking, AuthN/AuthZ/BOLA security tests, ESLint flat config, Prettier, strict tsc, and Month 4 Project 4: Secure Authentication and Authorization Service.

Objectives: ✓ Write fast unit tests (`Vitest`/`Jest`) for domain logic and comprehensive `Supertest` HTTP + real-DB integration tests for APIs. • ✓ Test authentication edge cases, refresh-token rotation/theft detection, RBAC, negative BOLA attempts, and concurrent race conditions. • ✓ Enforce `ESLint` (`no-floating-promises`), `Prettier`, `tsc --noEmit`, and `>= 80%` branch coverage gates automatically. • ✓ Deliver **Month 4 Project (Project 4): Secure Authentication and Authorization Service**.

16.1 Software Testing Fundamentals & the Backend Testing Trophy [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Static Type/Lint checks (`tsc` + `ESLint`) -> Fast Pure Unit Tests -> High-Confidence API + Real-DB Integration Tests (`Supertest`) -> Focused Critical-Path E2E Flows • Production Backend Pitfall & Guardrail (Software Testing Fundamentals & the Backend Testing Trophy) • Hands-On TypeScript / Node.js Implementation (Software Testing Fundamentals & the Backend Testing Trophy) • Software Testing Fundamentals & the Backend Testing Trophy — Latency, Throughput & Interview Trade-Offs • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Software Testing Fundamentals • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards • Diagnostics, Profiling & Telemetry: structured telemetry, profiler traces & CI checks

Purpose: Investing heavily in `Supertest + Real Test DB` integration tests where middleware, Zod, Service, and SQL constraints are verified together!

What Topic Is Next → Next Topic 16.2: Unit Testing Pure Domain Logic and Services

16.2 Unit Testing Pure Domain Logic and Services [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Arrange-Act-Assert (`AAA`) structure • Testing pure pricing/coupon/token calculators • Services injected with lightweight `InMemoryRepository` fakes in `< 5ms` per test • Production Backend Pitfall & Guardrail (Unit Testing Pure Domain Logic and Services) • Hands-On TypeScript / Node.js Implementation (Unit Testing Pure Domain Logic and Services) • Unit Testing Pure Domain Logic and Services — Latency, Throughput & Interview Trade-Offs • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Unit Testing Pure Domain Logic

Purpose: Testing 30 coupon discount boundary rules (expired, minCart, maxCap, firstOrderOnly) in `40ms` total.

What Topic Is Next → Next Topic 16.3: Integration Testing Repositories and Persistence

16.3 Integration Testing Repositories and Persistence [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Verifying actual Prisma/PostgreSQL • Mongoose/MongoDB queries • Unique/check constraints • Cascade rules • Production Backend Pitfall & Guardrail (Integration Testing Repositories and Persistence) • Hands-On TypeScript / Node.js Implementation (Integration Testing Repositories and Persistence) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Integration Testing Repositories

Purpose: Proving `CHECK (qty_on_hand >= 0)` and `11000`/`P2002` unique email errors map cleanly to `409 Conflict`.

What Topic Is Next → Next Topic 16.4: HTTP API Testing Principles

16.4 HTTP API Testing Principles [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Asserting HTTP status code • `Content-Type` • Security/rate/location headers • Response JSON schema • Production Backend Pitfall & Guardrail (HTTP API Testing Principles) • Hands-On TypeScript / Node.js Implementation (HTTP API Testing Principles) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for HTTP API Testing Principles

Purpose: Asserting not only that `POST /register` returned `201`, but ALSO that `dbUser.passwordHash !== plainPassword` and starts with `$argon2id$`!

What Topic Is Next → Next Topic 16.5: End-to-End Backend Flow Testing

16.5 End-to-End Backend Flow Testing [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Multi-step stateful user journeys (`Register -> Capture OTP from FakeMailer -> Verify Email -> Login -> Access Protected Route -> Rotate Refresh Cookie -> Logout -> Verify Old Refresh Rejected`) • Production Backend Pitfall & Guardrail (End-to-End Backend Flow Testing) • Hands-On TypeScript / Node.js Implementation (End-to-End Backend Flow Testing) • End-to-End Backend Flow Testing — Latency, Throughput & Interview Trade-Offs • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for End-to-End Backend Flow Testing • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards • Diagnostics, Profiling & Telemetry: structured telemetry, profiler traces & CI checks

Purpose: Proving the entire authentication lifecycle works cohesively across cookies, headers, Redis, and PostgreSQL.

What Topic Is Next → Next Topic 16.6: Test-Driven Development (TDD) Fundamentals

16.6 Test-Driven Development (TDD) Fundamentals [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `Red` (write failing test specifying exact behavior) -> `Green` (write minimal clean code to pass) -> `Refactor` (clean up with green safety net) • Production Backend Pitfall & Guardrail (Test-Driven Development (TDD) Fundamentals) • Hands-On TypeScript / Node.js Implementation (Test-Driven Development (TDD) Fundamentals) • Test-Driven Development (TDD) Fundamentals — Latency, Throughput & Interview Trade-Offs • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Test-Driven Development (TDD) Fundamentals • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards • Diagnostics, Profiling & Telemetry: structured telemetry, profiler traces & CI checks

Purpose: Using TDD on tricky security/financial logic (refresh-token theft detection, prorated refund math, webhook idempotency).

What Topic Is Next → Next Topic 16.7: Jest and Vitest Configuration for TypeScript + ESM

16.7 Jest and Vitest Configuration for TypeScript + ESM [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Configuring `vitest.config.ts` / `jest.config.ts` (`ts-jest` / native ESM) • Separating fast `test:unit` from DB-backed `test:integration` (`maxWorkers: 1` or per-worker DB schema!) • Environment setup • Production Backend Pitfall & Guardrail (Jest and Vitest Configuration for TypeScript + ESM) • Hands-On TypeScript / Node.js Implementation (Jest and Vitest Configuration for TypeScript + ESM) • Jest and Vitest Configuration for TypeScript + ESM — Latency, Throughput & Interview Trade-Offs • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Jest

Purpose: Running unit tests in `< 1s` on file save while running isolated DB integration suites cleanly.

What Topic Is Next → Next Topic 16.8: Node.js Testing Best Practices & Handle Cleanup

16.8 Node.js Testing Best Practices & Handle Cleanup [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Closing Prisma (`$disconnect`) • Mongoose (`disconnect`) • Redis (`quit`) in `afterAll` • Using `--detectOpenHandles` • Production Backend Pitfall & Guardrail (Node.js Testing Best Practices & Handle Cleanup) • Hands-On TypeScript / Node.js Implementation (Node.js Testing Best Practices & Handle Cleanup) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Node.js Testing Best Practices

Purpose: Ensuring CI test pipelines exit in `< 10s` with zero hung open socket warnings.

What Topic Is Next → Next Topic 16.9: Supertest Express HTTP Testing in Depth

16.9 Supertest Express HTTP Testing in Depth [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `request(app).post("/api/v1/auth/login").set("Authorization" • ...).send(...)` • `request.agent(app)` for automatic `Set-Cookie` / `Cookie` jar persistence across refresh/CSRF calls! • Production Backend Pitfall & Guardrail (Supertest Express HTTP Testing in Depth) • Hands-On TypeScript / Node.js Implementation (Supertest Express HTTP Testing in Depth) • Supertest Express HTTP Testing in Depth — Latency, Throughput & Interview Trade-Offs • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Supertest Express HTTP Testing in Depth

Purpose: Testing `HttpOnly; SameSite=Strict` refresh-token rotation cookies accurately with `supertest.agent(app)`.

What Topic Is Next → Next Topic 16.10: Assertions and Expressive Matchers

16.10 Assertions and Expressive Matchers [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `toEqual` • `toMatchObject` • `expect.any(String)` • `expect.stringMatching()` • Production Backend Pitfall & Guardrail (Assertions and Expressive Matchers) • Hands-On TypeScript / Node.js Implementation (Assertions and Expressive Matchers) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Assertions

Purpose: Using `toMatchObject` with `expect.any(String)` for dynamic UUIDs and ISO timestamps.

What Topic Is Next → Next Topic 16.11: Test Lifecycle Hooks (beforeAll, beforeEach, afterEach, afterAll)

16.11 Test Lifecycle Hooks (beforeAll, beforeEach, afterEach, afterAll) [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Connecting DB pools once in `beforeAll` • Resetting table rows + `vi.restoreAllMocks()` in `beforeEach`/`afterEach` • Disconnecting pools in `afterAll` • Production Backend Pitfall & Guardrail (Test Lifecycle Hooks (beforeAll, beforeEach, afterEach, afterAll)) • Hands-On TypeScript / Node.js Implementation (Test Lifecycle Hooks (beforeAll, beforeEach, afterEach, afterAll)) • Test Lifecycle Hooks (beforeAll, beforeEach, afterEach, afterAll) — Latency, Throughput & Interview Trade-Offs • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Test Lifecycle Hooks (beforeAll, beforeEach, afterEa...

Purpose: Keeping 100 DB integration tests fast (`~3s` total) by connecting once and truncating cleanly per test.

What Topic Is Next → Next Topic 16.12: Test Factories and Data Builders

16.12 Test Factories and Data Builders [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Creating composable `buildUserOverrides({ role = "USER" • IsVerified = true } = {})` factories generating unique emails/SKUs (`crypto.randomUUID()`) per call • Production Backend Pitfall & Guardrail (Test Factories and Data Builders) • Hands-On TypeScript / Node.js Implementation (Test Factories and Data Builders) • Test Factories and Data Builders — Latency, Throughput & Interview Trade-Offs • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Test Factories • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards

Purpose: Making each test readable in 3 lines by highlighting ONLY the fields relevant to that test (`createTestUser({ isVerified: false })`).

What Topic Is Next → Next Topic 16.13: Mocking, Stubbing and Fake Timers

16.13 Mocking, Stubbing and Fake Timers [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Using `vi.useFakeTimers()` / `jest.useFakeTimers()` + `advanceTimersByTime(16 * 60 * 1000)` to test `15-minute` token expiry • `15-minute` rate-limit resets in `1 millisecond` (never `await sleep(60000)` in tests!)! • Production Backend Pitfall & Guardrail (Mocking, Stubbing and Fake Timers) • Hands-On TypeScript / Node.js Implementation (Mocking, Stubbing and Fake Timers) • Mocking, Stubbing and Fake Timers — Latency, Throughput & Interview Trade-Offs • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Mocking • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards

Purpose: Testing `15m` access-token expiration and `7d` refresh-token expiration deterministically in `2ms`!

What Topic Is Next → Next Topic 16.14: Dependency Mocking Boundaries (What to Mock vs Keep Real)

16.14 Dependency Mocking Boundaries (What to Mock vs Keep Real) [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Golden rule: **Mock external third-party network boundaries** (SMTP • OAuth token endpoint • Payment Gateway HTTP) • **DO NOT mock your own database • Production Backend Pitfall & Guardrail (Dependency Mocking Boundaries (What to Mock vs Keep Real)) • Hands-On TypeScript / Node.js Implementation (Dependency Mocking Boundaries (What to Mock vs Keep Real)) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Dependency Mocking Boundaries (What to Mock vs Keep...

Purpose: Injecting `FakeMailService` (`sentEmails: SentMail[]`) to inspect the exact password-reset link sent during `POST /forgot-password`!

What Topic Is Next → Next Topic 16.15: Database Testing & Fast Table Cleanup

16.15 Database Testing & Fast Table Cleanup [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Querying `pg_tables` (`schemaname = 'public'` excluding `_prisma_migrations`) to execute a single `TRUNCATE TABLE ... RESTART IDENTITY CASCADE` in `< 8ms` between integration tests! • Production Backend Pitfall & Guardrail (Database Testing & Fast Table Cleanup) • Hands-On TypeScript / Node.js Implementation (Database Testing & Fast Table Cleanup) • Database Testing & Fast Table Cleanup — Latency, Throughput & Interview Trade-Offs • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Database Testing • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards • Diagnostics, Profiling & Telemetry: structured telemetry, profiler traces & CI checks

Purpose: Resetting 15 relational tables cleanly in one PostgreSQL round-trip without re-running slow DDL migrations.

What Topic Is Next → Next Topic 16.16: Authentication Testing Matrix

16.16 Authentication Testing Matrix [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Systematic tests: valid signup/login • Duplicate email (`409`) • Wrong password (`401`) • Unverified login policy • Production Backend Pitfall & Guardrail (Authentication Testing Matrix) • Hands-On TypeScript / Node.js Implementation (Authentication Testing Matrix) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Authentication Testing Matrix

Purpose: Guaranteeing zero regressions in Project 4's authentication core.

What Topic Is Next → Next Topic 16.17: Authorization and BOLA Security Testing

16.17 Authorization and BOLA Security Testing [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Matrix testing `(Anonymous • Customer A • Customer B • Support • Production Backend Pitfall & Guardrail (Authorization and BOLA Security Testing) • Hands-On TypeScript / Node.js Implementation (Authorization and BOLA Security Testing) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Authorization

Purpose: Proving mathematically via table-driven `it.each(...)` tests that horizontal (`Customer A -> Customer B`) and vertical (`Customer -> Admin`) privilege escalations fail!

What Topic Is Next → Next Topic 16.18: Negative, Boundary and Concurrency Race Testing

16.18 Negative, Boundary and Concurrency Race Testing [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Firing `await Promise.all(Array.from({ length: 10 } • () => request(app).post("/coupons/redeem")))` on a `maxUses = 1` coupon or `qty = 1` item — proving **1 succeeds (`200`) • 9 fail (`409`)**! • Production Backend Pitfall & Guardrail (Negative, Boundary and Concurrency Race Testing) • Hands-On

TypeScript / Node.js Implementation (Negative, Boundary and Concurrency Race Testing) • Negative, Boundary and Concurrency Race Testing — Latency, Throughput & Interview Trade-Offs • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Negative

Purpose: Catching check-then-act race conditions in stock deduction, coupon redemption, and refresh rotation BEFORE launch!

What Topic Is Next → Next Topic 16.19: API Contract Testing Fundamentals

16.19 API Contract Testing Fundamentals [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Validating every `Supertest` response JSON against your published OpenAPI 3.1 response schema so undocumented status codes or missing required fields fail tests automatically • Production Backend Pitfall & Guardrail (API Contract Testing Fundamentals) • Hands-On TypeScript / Node.js Implementation (API Contract Testing Fundamentals) • API Contract Testing Fundamentals — Latency, Throughput & Interview Trade-Offs • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for API Contract Testing Fundamentals • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards • Diagnostics, Profiling & Telemetry: structured telemetry, profiler traces & CI checks

Purpose: Catching accidental breaking response changes before frontend/mobile apps break.

What Topic Is Next → Next Topic 16.20: Test Coverage Gates and Branch Coverage

16.20 Test Coverage Gates and Branch Coverage [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Measuring Statements • Branches (`if/else` • `??` • `?:`) • Production Backend Pitfall & Guardrail (Test Coverage Gates and Branch Coverage) • Hands-On TypeScript / Node.js Implementation (Test Coverage Gates and Branch Coverage) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Test Coverage Gates

Purpose: Failing CI if someone adds a new payment/auth branch with `0%` test coverage.

What Topic Is Next → Next Topic 16.21: ESLint Static Analysis for Async TypeScript

16.21 ESLint Static Analysis for Async TypeScript [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: ESLint v9 flat config (`eslint.config.mjs`) + `typescript-eslint` type-aware rules: **`@typescript-eslint/no-floating-promises`** (catches forgotten `await`!) • `no-misused-promises` (Express async callback trap!) • `no-explicit-any`! • Production Backend Pitfall & Guardrail (ESLint Static Analysis for Async TypeScript) • Hands-On TypeScript / Node.js Implementation (ESLint Static Analysis for Async TypeScript) • ESLint Static Analysis for Async TypeScript — Latency, Throughput & Interview Trade-Offs • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for ESLint Static Analysis for Async TypeScript

Purpose: Catching forgotten `await`s on database writes and Promise-returning guards at lint time!

What Topic Is Next → Next Topic 16.22: Prettier Code Formatting

16.22 Prettier Code Formatting [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `.prettierrc` + `eslint-config-prettier` + `prettier --check .` in CI eliminating bikeshedding formatting debates in Pull Requests • Production Backend Pitfall & Guardrail (Prettier Code Formatting) • Hands-On TypeScript / Node.js Implementation (Prettier Code Formatting) • Prettier Code Formatting — Latency, Throughput & Interview Trade-Offs • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Prettier Code Formatting • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards • Diagnostics, Profiling & Telemetry: structured telemetry, profiler traces & CI checks

Purpose: Keeping diffs crisp and uniform across every contributor.

What Topic Is Next → Next Topic 16.23: TypeScript Strict Compiler Checks in CI

16.23 TypeScript Strict Compiler Checks in CI [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Running `tsc --noEmit` in CI (because fast test runners like `vitest`/`swc`/`esbuild` transpile TS WITHOUT typechecking!) • Production Backend Pitfall & Guardrail (TypeScript Strict Compiler Checks in CI) • Hands-On TypeScript / Node.js Implementation (TypeScript Strict Compiler Checks in CI) • TypeScript Strict Compiler Checks in CI — Latency, Throughput & Interview Trade-Offs • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for TypeScript Strict Compiler Checks in CI • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards • Diagnostics, Profiling & Telemetry: structured telemetry, profiler traces & CI checks

Purpose: Preventing broken types from slipping into `main` just because `vitest` skipped full `tsc` checking!

What Topic Is Next → Next Topic 16.24: Backend Code Review Standards

16.24 Backend Code Review Standards [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Reviewing PRs systematically for: BOLA ownership • SQL/NoSQL parameterization • Transaction boundaries • N+1 queries • Production Backend Pitfall & Guardrail (Backend Code Review Standards) • Hands-On TypeScript / Node.js Implementation (Backend Code Review Standards) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Backend Code Review Standards

Purpose: Catching architectural and security defects before merge.

What Topic Is Next → Next Topic 16.25: Continuous Test Execution & Project 4 Delivery

16.25 Continuous Test Execution & Project 4 Delivery [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Automating `lint + typecheck + unit + integration (with Postgres/Redis service containers)` on every push • Completing **Project 4: Secure Authentication • Authorization Service** • Production Backend Pitfall & Guardrail (Continuous Test Execution & Project 4 Delivery) • Hands-On TypeScript / Node.js Implementation (Continuous Test Execution & Project 4 Delivery) • Continuous Test Execution & Project 4 Delivery — Latency, Throughput & Interview Trade-Offs • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Continuous Test Execution

Purpose: Delivering Project 4 with a green badge and `>= 80%` verified test coverage.

What Topic Is Next → Module 16 Capstone (Project 4: Secure Authentication and Authorization Service) → Next Topic 17.1: Redis Architecture and Docker Setup

Module 16 Hands-On Capstone Project (100 Marks): Project 4: Secure Authentication and Authorization Service — Broken authentication, token theft, missing BOLA checks, and brute-force credential stuffing account for the majority of severe backend breaches. Build a standalone, reusable **Secure Authen...

MODULE 17 OF 24 • STAGE 5 • 25 LIVE + LAB HOURS • 21 TOPICS (168 SUBTOPICS) • 1 CAPSTONE PROJECT (100 MARKS)

Module 17: Redis and Backend Performance Optimization

Slash API latency and coordinate distributed Node.js workers using Redis 7 — Strings/Hashes/Lists/Sets/Sorted Sets, SCAN & pipelining, ioredis + atomic Lua scripts, Cache-Aside & invalidation, TTL jitter, stampede single-flight/mutex protection, Redis rate limiting, distributed locks, RDB/AOF persistence, and autocannon benchmarking.

Objectives: ✓ Choose optimal Redis data structures (`String`, `Hash`, `List`, `Set`, `Sorted Set`) and namespace keys with explicit TTLs. • ✓ Implement Cache-Aside, deterministic write invalidation, TTL jitter, and Single-Flight / Mutex cache-stampede protection. • ✓ Build atomic Lua-backed Sliding-Window rate limiters and safe owner-token Distributed Locks in `ioredis`. • ✓ Configure `RDB`/`AOF` persistence & `maxmemory-policy` and prove `5–10x` latency/RPS improvements with `autocannon`.

17.1 Redis Architecture and Docker Setup [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Why single-threaded in-memory Redis executes `100 • 000+ ops/sec` at sub-millisecond latency (zero context-switching/lock overhead + multiplexed `epoll` I/O) • `redis:7-alpine` • `redis-cli` • Production Backend Pitfall & Guardrail (Redis Architecture and Docker Setup) • Hands-On TypeScript / Node.js Implementation (Redis Architecture and Docker Setup) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Redis Architecture

Purpose: Serving hot catalog, session, and rate-limit lookups in `0.3ms` instead of `12ms` disk DB queries.

What Topic Is Next → Next Topic 17.2: Redis Data Types Overview

17.2 Redis Data Types Overview [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Choosing the right native structure (`String` • `Hash` • `List` • `Set` • Production Backend Pitfall & Guardrail (Redis Data Types Overview) • Hands-On TypeScript / Node.js Implementation (Redis Data Types Overview) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Redis Data Types Overview

Purpose: Using `HINCRBY cart:u42 item:sku9 1` to increment 1 cart item atomically without reading/writing the whole cart JSON!

What Topic Is Next → Next Topic 17.3: Strings, Hashes, Lists and Sets

17.3 Strings, Hashes, Lists and Sets [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `SET key val EX 300 NX` (atomic set-if-not-exists + expiry!) • `INCR`/`DECR` • `HSET`/`HMGET` • `LPUSH`/`RPOP` • Production Backend Pitfall & Guardrail (Strings, Hashes, Lists and Sets) • Hands-On TypeScript / Node.js Implementation (Strings, Hashes, Lists and Sets) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Strings

Purpose: Using `SET idemp:<key> "PROCESSING" EX 60 NX` to guarantee two simultaneous duplicate POSTs cannot both execute!

What Topic Is Next → Next Topic 17.4: Sorted Sets (ZSET) for Leaderboards & Sliding Windows

17.4 Sorted Sets (ZSET) for Leaderboards & Sliding Windows [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `ZADD key score member` • `ZINCRBY` • `ZREVRANGE` top-K rankings in `O(log N + K)` • `ZREMRANGEBYSCORE` pruning expired timestamps • Production Backend Pitfall & Guardrail (Sorted Sets (ZSET) for Leaderboards & Sliding Windows) • Hands-On TypeScript / Node.js Implementation (Sorted Sets (ZSET) for Leaderboards & Sliding Windows) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Sorted Sets (ZSET) for Leaderboards

Purpose: Maintaining real-time `"Top 10 Trending Products"` (`ZINCRBY trending:2026-11-18 1 sku`) AND exact sliding-window rate limiters!

What Topic Is Next → Next Topic 17.5: Redis Commands, Expiration, SCAN and Pipelining

17.5 Redis Commands, Expiration, SCAN and Pipelining [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `EXPIRE`/`PEXPIRE` • `TTL`/`PTTL` • Cursor-based `SCAN 0 MATCH prefix:* COUNT 100` • `UNLINK` (async non-blocking memory free vs blocking `DEL` on huge keys!) • Production Backend Pitfall & Guardrail (Redis Commands, Expiration, SCAN and Pipelining) • Hands-On TypeScript / Node.js Implementation (Redis Commands, Expiration, SCAN and Pipelining) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Redis Commands

Purpose: Using `pipeline()` to execute 15 Redis commands in `0.6ms` (`1` RTT) instead of `15 * 0.6ms = 9ms` sequential!

What Topic Is Next → Next Topic 17.6: Key Naming Conventions and Memory Modelling

17.6 Key Naming Conventions and Memory Modelling [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Colon-delimited hierarchical keys `{service}:{version}:{entity}:{id}` (`catalog:v1:product:SKU-101` — bump `v1 -> v2` on DTO schema changes to avoid deserialization crashes during rolling deploys!) • Production Backend Pitfall & Guardrail (Key Naming Conventions and Memory Modelling) • Hands-On TypeScript / Node.js Implementation (Key Naming Conventions and Memory Modelling) • Key Naming Conventions and Memory Modelling — Latency, Throughput & Interview Trade-Offs • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Key Naming Conventions • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards • Diagnostics, Profiling & Telemetry: structured telemetry, profiler traces & CI checks

Purpose: Preventing new `v2` containers from crashing on cached `v1` JSON shapes during a zero-downtime deploy!

What Topic Is Next → Next Topic 17.7: Node.js Redis Integration (ioredis & Lua Atomicity)

17.7 Node.js Redis Integration (ioredis & Lua Atomicity) [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `new Redis(url • { maxRetriesPerRequest: 3 • EnableReadyCheck: true • LazyConnect: true })` • Production Backend Pitfall & Guardrail (Node.js Redis Integration (ioredis & Lua Atomicity)) • Hands-On TypeScript / Node.js Implementation (Node.js Redis Integration (ioredis & Lua Atomicity)) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program

Production Runtime Execution & Memory Lifecycle for Node.js Redis Integration (ioredis & Lua Atomicity)

Purpose: Running `GET + compare + DEL` or `ZREMRANGEBYSCORE + ZCARD + ZADD + PEXPIRE` as one indivisible atomic Lua operation.

What Topic Is Next → Next Topic 17.8: Caching Fundamentals (When to Cache & Hit-Ratio Math)

17.8 Caching Fundamentals (When to Cache & Hit-Ratio Math) [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Effective latency `L_eff = hit_rate * L_cache + (1 - hit_rate) * L_db` (`95%` hit @ `0.5ms` + `5%` miss @ `40ms` = `2.47ms` avg!) • Read-heavy high-fanout candidates vs write-heavy/strictly-real-time non-candidates • Production Backend Pitfall & Guardrail (Caching Fundamentals (When to Cache & Hit-Ratio Math)) • Hands-On TypeScript / Node.js Implementation (Caching Fundamentals (When to Cache & Hit-Ratio Math)) • Caching Fundamentals (When to Cache & Hit-Ratio Math) — Latency, Throughput & Interview Trade-Offs • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Caching Fundamentals (When to Cache & Hit-Ratio Math... • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards

Purpose: Caching Product Details, Category Trees, and User Permissions (`98%` read ratio) while NOT caching live available stock counts blindly.

What Topic Is Next → Next Topic 17.9: Cache-Aside (Lazy Loading) Pattern

17.9 Cache-Aside (Lazy Loading) Pattern [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Read path: `1) Check Redis -> Hit? Return parsed -> 2) Miss? Query DB -> 3) Populate Redis w/ TTL -> Return` + caching negative `"__NULL__"` sentinels (`short 30s TTL`) to block **Cache Penetration** attacks on non-existent IDs! • Production Backend Pitfall & Guardrail (Cache-Aside (Lazy Loading) Pattern) • Hands-On TypeScript / Node.js Implementation (Cache-Aside (Lazy Loading) Pattern) • Cache-Aside (Lazy Loading) Pattern — Latency, Throughput & Interview Trade-Offs • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Cache-Aside (Lazy Loading) Pattern • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards • Diagnostics, Profiling & Telemetry: structured telemetry, profiler traces & CI checks

Purpose: Stopping an attacker requesting `/products/fake-id-1..100000` from bypassing Redis and hammering MongoDB/Postgres (`Cache Penetration`)!

What Topic Is Next → Next Topic 17.10: Read-Through, Write-Through and Write-Behind Concepts

17.10 Read-Through, Write-Through and Write-Behind Concepts [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Comparing **Cache-Aside** (only requested items cached) • **Write-Through** (write updates DB + Cache synchronously) • **Write-Behind** (write hits Redis fast -> async batch flushes to DB • E.g. • Production Backend Pitfall & Guardrail (Read-Through, Write-Through and Write-Behind Concepts) • Hands-On TypeScript / Node.js Implementation (Read-Through, Write-Through and Write-Behind Concepts) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Read-Through

Purpose: Using Write-Behind (`HINCRBY product:views sku 1` -> flushed to Postgres every 60s by BullMQ) so 10,000 product pageviews/sec cause `0` Postgres row lock contention!

What Topic Is Next → Next Topic 17.11: Deterministic Cache Invalidation

17.11 Deterministic Cache Invalidation [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Why **deleting (`UNLINK`/`DEL`)** the cache key AFTER the DB transaction commits beats overwriting the cache value (avoids out-of-order concurrent write overwrite races!) • Production Backend Pitfall & Guardrail (Deterministic Cache Invalidation) • Hands-On TypeScript / Node.js Implementation (Deterministic Cache Invalidation) • Deterministic Cache Invalidation — Latency, Throughput & Interview Trade-Offs • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Deterministic Cache Invalidation • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards • Diagnostics, Profiling & Telemetry: structured telemetry, profiler traces & CI checks

Purpose: Calling `await redis.unlink(RedisKeys.product(sku))` right after `await productRepo.update(...)` commits.

What Topic Is Next → Next Topic 17.12: TTL Strategies and Expiry Jitter (Preventing Cache Avalanche)

17.12 TTL Strategies and Expiry Jitter (Preventing Cache Avalanche) [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Why warming 10 • 000 product keys at boot with exact `EX 3600` causes all 10 • 000 keys to expire at the EXACT same second 1 hour later (`Cache Avalanche`!) -> adding random jitter `baseTtl + randomInt(0 • BaseTtl * 0.15)`! • Production Backend Pitfall & Guardrail (TTL Strategies and Expiry Jitter (Preventing Cache Avalanche)) • Hands-On TypeScript / Node.js Implementation (TTL Strategies and Expiry Jitter (Preventing Cache Avalanche)) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for TTL Strategies

Purpose: Spreading expirations smoothly across a 5–10 minute window so DB load stays flat.

What Topic Is Next → Next Topic 17.13: Database Query Caching

17.13 Database Query Caching [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Hashing normalized query filter objects (`sha256(canonicalJson)`) into versioned cache keys (`catalog:v1:list:ver_${version}:${filterHash}`) bumped atomically via `INCR catalog:v1:version` on catalog writes! • Production Backend Pitfall & Guardrail (Database Query Caching) • Hands-On TypeScript / Node.js Implementation (Database Query Caching) • Database Query Caching — Latency, Throughput & Interview Trade-Offs • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Database Query Caching • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards • Diagnostics, Profiling & Telemetry: structured telemetry, profiler traces & CI checks

Purpose: Invalidating 500 cached filter/page combinations in `O(1)` (`1` `INCR` command!) without scanning keys!

What Topic Is Next → Next Topic 17.14: Session and Auth Metadata Storage in Redis

17.14 Session and Auth Metadata Storage in Redis [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Storing `sess:${sid}` • User permission sets (`auth:v1:perms:${userId}` `TTL 5m`) • OTP counters • `revoked:jti:${jti}` with automatic sub-millisecond expiry • Production Backend Pitfall & Guardrail (Session and Auth Metadata Storage in Redis) • Hands-On TypeScript / Node.js Implementation (Session and Auth Metadata Storage in Redis) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Session

Purpose: Resolving `req.user` fine-grained permissions in `0.4ms` from Redis instead of joining 3 RBAC SQL tables on every API call!

What Topic Is Next → Next Topic 17.15: API Response Caching Middleware

17.15 API Response Caching Middleware [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Caching serialized JSON + `ETag` (`If-None-Match` -> `304 Not Modified` saving bandwidth!) strictly on `GET` public/tenant-scoped endpoints (`NEVER` cache personalized responses under a shared public key!) • Production Backend Pitfall & Guardrail (API Response Caching Middleware) • Hands-On TypeScript / Node.js Implementation (API Response Caching Middleware) • API Response Caching Middleware — Latency, Throughput & Interview Trade-Offs • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for API Response Caching Middleware • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards • Diagnostics, Profiling & Telemetry: structured telemetry, profiler traces & CI checks

Purpose: Returning `304 Not Modified` in `0.8ms` with `0` body bytes when mobile clients re-fetch unchanged category trees.

What Topic Is Next → Next Topic 17.16: Redis-Based Sliding-Window & Token-Bucket Rate Limiting

17.16 Redis-Based Sliding-Window & Token-Bucket Rate Limiting [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Atomic Lua Sliding-Window Log (`ZREMRANGEBYSCORE` old hits -> `ZCARD` check `< limit` -> `ZADD now reqUuid` -> `PEXPIRE`) preventing the `2x` boundary burst flaw of naive Fixed-Window counters! • Production Backend Pitfall & Guardrail (Redis-Based Sliding-Window & Token-Bucket Rate Limiting) • Hands-On TypeScript / Node.js Implementation (Redis-Based Sliding-Window & Token-Bucket Rate Limiting) • Redis-Based Sliding-Window & Token-Bucket Rate Limiting — Latency, Throughput & Interview Trade-Offs • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Redis-Based Sliding-Window • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards • Diagnostics, Profiling & Telemetry: structured telemetry, profiler traces & CI checks

Purpose: Enforcing exact `100 requests / rolling 60s` fairness across all API instances in `1` atomic Lua round-trip.

What Topic Is Next → Next Topic 17.17: Distributed Locks Fundamentals

17.17 Distributed Locks Fundamentals [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Acquiring `SET lock:${resource} ${ownerUuid} PX ${ttlMs} NX` • Releasing ONLY via atomic Lua `if redis.call("get" • KEYS[1]) == ARGV[1] then return redis.call("del" • KEYS[1]) else return 0 end`! • Production Backend Pitfall & Guardrail (Distributed Locks Fundamentals) • Hands-On TypeScript / Node.js Implementation (Distributed Locks Fundamentals) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Distributed Locks Fundamentals

Purpose: Coordinating exclusive invoice sequence generation or scheduled cron execution across multiple workers.

What Topic Is Next → Next Topic 17.18: Cache Consistency Patterns

17.18 Cache Consistency Patterns [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Understanding why distributed Cache + DB is eventually consistent unless serialized • Handling stale reads safely • NEVER treating ephemeral cache as the authoritative balance before charging money! • Production Backend Pitfall & Guardrail (Cache Consistency Patterns) • Hands-On TypeScript / Node.js Implementation (Cache Consistency Patterns) • Cache Consistency Patterns — Latency, Throughput & Interview Trade-Offs • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Cache Consistency Patterns

Purpose: Reading catalog display prices from Redis cache, then ALWAYS re-reading authoritative `price_paise` from PostgreSQL/MongoDB at `POST /orders` checkout!

What Topic Is Next → Next Topic 17.19: Cache Stampede (Thundering Herd) Prevention

17.19 Cache Stampede (Thundering Herd) Prevention [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: When a hot homepage key expires under `5 • 000 RPS` • `500` requests simultaneously miss Redis • Slam PostgreSQL! Defenses: (1) **In-Process Single-Flight Promise Map** (`inflight.get(key)`) + (2) **Redis Mutex Lock** + (3) **Early Probabilistic Refresh (XFetch)**! • Production Backend Pitfall & Guardrail (Cache Stampede (Thundering Herd) Prevention) • Hands-On TypeScript / Node.js Implementation (Cache Stampede (Thundering Herd) Prevention) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Cache Stampede (Thundering Herd) Prevention

Purpose: Coalescing 200 concurrent cache misses for `product:iphone-16` into **EXACTLY ONE** DB query while the other 199 await the same Promise!

What Topic Is Next → Next Topic 17.20: Redis Persistence (RDB vs AOF) and Eviction Policies

17.20 Redis Persistence (RDB vs AOF) and Eviction Policies [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `RDB` point-in-time fork snapshots vs `AOF` (`appendfsync everysec`) durability • Why **Cache Redis** uses `maxmemory-policy allkeys-lru` whereas **BullMQ Queue Redis** MUST use **`noeviction`** (evicting job keys silently drops background jobs!!)! • Production Backend Pitfall & Guardrail (Redis Persistence (RDB vs AOF) and Eviction Policies) • Hands-On TypeScript / Node.js Implementation (Redis Persistence (RDB vs AOF) and Eviction Policies) • Redis Persistence (RDB vs AOF) and Eviction Policies — Latency, Throughput & Interview Trade-Offs • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Redis Persistence (RDB vs AOF) • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards

Purpose: Running separate logical/instance configs (`allkeys-lru` for cache vs `noeviction` + `AOF` for BullMQ queues/sessions)!

What Topic Is Next → Next Topic 17.21: Performance Benchmarking Before & After Redis

17.21 Performance Benchmarking Before & After Redis [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Running reproducible load benchmarks with `autocannon -c 100 -d 15` measuring `Req/Sec` • `p50` • `p95` • `p99` latency • Production Backend Pitfall & Guardrail (Performance Benchmarking Before & After Redis) • Hands-On TypeScript / Node.js Implementation (Performance Benchmarking Before & After Redis) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Performance Benchmarking Before

Purpose: Documenting `GET /api/v1/products`: `380 RPS / p99 210ms` (DB direct) -> `4,900 RPS / p99 14ms` (Redis Cache-Aside + Single-Flight) in your portfolio README!

What Topic Is Next → Module 17 Capstone (Week 17 Lab Build: Redis 7 Cache-Aside, Single-Flight Stampede Shield & Redlock Service) → Next Topic 18.1: Synchronous vs Asynchronous Processing

Module 17 Hands-On Capstone Project (100 Marks): Week 17 Lab Build: Redis 7 Cache-Aside, Single-Flight Stampede Shield & Redlock Service — Implement a typed `ioredis` caching layer featuring Cache-Aside with jittered TTLs, in-process `Map<string, Promise<T>>` Single-Flight stampede coalescing, tag/version invalidation, ZSET lea...

MODULE 18 OF 24 • STAGE 5 • 25 LIVE + LAB HOURS • 31 TOPICS (248 SUBTOPICS) • 1 CAPSTONE PROJECT (100 MARKS)

Module 18: Background Jobs, Queues and Messaging Systems

Decouple slow, failure-prone, and cross-service workflows asynchronously using BullMQ (Redis queues, delayed & repeatable jobs, concurrency), RabbitMQ (AMQP direct/fanout/topic/headers exchanges, manual acks, DLQs), and Apache Kafka fundamentals (brokers, partitions, consumer groups, offsets).

Objectives: ✓ Design idempotent, retry-safe background worker pipelines with exponential backoff and Dead-Letter Queues (`DLQ`). • ✓ Build production `BullMQ` Producers, Workers, Delayed Jobs, Repeatable Schedules, and concurrency/rate-limited processors. • ✓ Publish and consume persistent `RabbitMQ` messages across `direct`, `fanout`, `topic`, and `headers` exchanges with manual `ack` and `DLX`/`DLQ` routing. • ✓ Explain `Apache Kafka` brokers, partitioned ordering keys, replication, consumer groups, and offset replayability.

18.1 Synchronous vs Asynchronous Processing [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Keeping user-facing HTTP critical path minimal (`validate + write order = 25ms`) while pushing non-blocking side effects (invoice PDF • Email • Analytics • Webhook delivery) onto durable queues • Production Backend Pitfall & Guardrail (Synchronous vs Asynchronous Processing) • Hands-On TypeScript / Node.js Implementation (Synchronous vs Asynchronous Processing) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Synchronous vs Asynchronous Processing

Purpose: Ensuring checkout succeeds in `30ms` even when the external SMTP or PDF service is temporarily slow.

What Topic Is Next → Next Topic 18.2: Background Job Architecture

18.2 Background Job Architecture [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Producer (API container enqueues compact JSON `{ orderId }` — NEVER `10 MB` blobs!) -> Durable Broker -> Independent Worker Processes -> Result/Audit Store • Production Backend Pitfall & Guardrail (Background Job Architecture) • Hands-On TypeScript / Node.js Implementation (Background Job Architecture) • Background Job Architecture — Latency, Throughput & Interview Trade-Offs • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Background Job Architecture • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards • Diagnostics, Profiling & Telemetry: structured telemetry, profiler traces & CI checks

Purpose: Scaling CPU-heavy PDF Worker containers (`4` replicas) independently of HTTP API containers (`2` replicas)!

What Topic Is Next → Next Topic 18.3: Job Queues and Isolated Worker Pools

18.3 Job Queues and Isolated Worker Pools [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Separating fast high-priority queues (`otp-emails` SLA `< 3s`) from slow bulk queues (`monthly-csv-exports`) so 10 • 000 CSV exports NEVER delay login OTPs (`Head-of-Line Blocking` prevention!) • Production Backend Pitfall & Guardrail (Job Queues and Isolated Worker Pools) • Hands-On TypeScript / Node.js Implementation (Job Queues and Isolated Worker Pools) • Job Queues and Isolated Worker Pools — Latency, Throughput & Interview Trade-Offs • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Job Queues • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards

Purpose: Dedicating isolated queues by SLA tier: `critical-realtime`, `default-transactional`, and `bulk-low-priority`.

What Topic Is Next → Next Topic 18.4: Scheduled and Recurring Jobs

18.4 Scheduled and Recurring Jobs [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Cron-driven recurring maintenance vs per-entity delayed timers (`check unpaid order #901 in 15m`) • Production Backend Pitfall & Guardrail (Scheduled and Recurring Jobs) • Hands-On TypeScript / Node.js Implementation (Scheduled and Recurring Jobs) • Scheduled and Recurring Jobs — Latency, Throughput & Interview Trade-Offs • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Scheduled • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards • Diagnostics, Profiling & Telemetry: structured telemetry, profiler traces & CI checks

Purpose: Enqueuing a precise `delay: 15 * 60 * 1000` expiry check per order rather than polling the entire orders table every second.

What Topic Is Next → Next Topic 18.5: Job Retries and Exponential Backoff

18.5 Job Retries and Exponential Backoff [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `attempts: 5` • `backoff: { type: "exponential" • Delay: 2000 }` (`2s -> 4s -> 8s -> 16s -> 32s`) vs failing immediately (`UnrecoverableError`) on permanent `400` invalid-email errors! • Production Backend Pitfall & Guardrail (Job Retries and Exponential Backoff) • Hands-On TypeScript / Node.js Implementation (Job Retries and Exponential Backoff) • Job Retries and Exponential Backoff — Latency, Throughput & Interview Trade-Offs • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Job Retries

Purpose: Surviving a 20-second SMTP outage automatically while skipping pointless retries on permanent validation bugs.

What Topic Is Next → Next Topic 18.6: Failed Jobs and Dead-Letter Handling (DLQ)

18.6 Failed Jobs and Dead-Letter Handling (DLQ) [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Routing poison-pill messages that exhaust all `attempts` into a retained Dead-Letter Queue (`DLQ`) with error stack • Attempt count • Admin replay tooling • Production Backend Pitfall & Guardrail (Failed Jobs and Dead-Letter Handling (DLQ)) • Hands-On TypeScript / Node.js Implementation (Failed Jobs and Dead-Letter Handling (DLQ)) • Failed Jobs and Dead-Letter Handling (DLQ) — Latency, Throughput & Interview Trade-Offs • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Failed Jobs

Purpose: Preventing 1 malformed message from blocking the queue forever AND preserving it so engineers can fix the bug and replay it!

What Topic Is Next → Next Topic 18.7: Idempotent Job Processing

18.7 Idempotent Job Processing [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Why EVERY queue guarantees **At-Least-Once** delivery (worker crash right after sending email / charging card but right BEFORE `ack` replays the job!) -> enforcing deterministic `jobId` + DB idempotency state! • Production Backend Pitfall & Guardrail (Idempotent Job Processing) • Hands-On TypeScript / Node.js Implementation (Idempotent Job Processing) • Idempotent Job Processing — Latency, Throughput & Interview Trade-Offs • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Idempotent Job Processing • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards • Diagnostics, Profiling & Telemetry: structured telemetry, profiler traces & CI checks

Purpose: Checking `if (order.invoiceSentAt !== null) return;` + atomic DB flag so re-running `generate-invoice` twice is a safe no-op!

What Topic Is Next → Next Topic 18.8: Job Queue Monitoring and Operational Health

18.8 Job Queue Monitoring and Operational Health [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Tracking `waiting` • `active` • `delayed` • `failed` counts • Production Backend Pitfall & Guardrail (Job Queue Monitoring and Operational Health) • Hands-On TypeScript / Node.js Implementation (Job Queue Monitoring and Operational Health) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Job Queue Monitoring

Purpose: Alerting when `q-critical-otp` oldest waiting job age exceeds `5 seconds` (scale workers!).

What Topic Is Next → Next Topic 18.9: BullMQ Installation and Architecture

18.9 BullMQ Installation and Architecture [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Modern TypeScript-native `bullmq` (`Redis Streams` + atomic Lua scripts replacing legacy `bull`) • Separate `ioredis` connections for `Queue` vs blocking `Worker` (`maxRetriesPerRequest: null` required on Worker connection!) • Production Backend Pitfall & Guardrail (BullMQ Installation and Architecture) • Hands-On TypeScript / Node.js Implementation (BullMQ Installation and Architecture) • BullMQ Installation and Architecture — Latency, Throughput & Interview Trade-Offs • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for BullMQ Installation • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards

Purpose: Setting `maxRetriesPerRequest: null` on the BullMQ Worker's dedicated `ioredis` connection as required by blocking `BZPOPMIN`/`XREADGROUP` commands.

What Topic Is Next → Next Topic 18.10: BullMQ Queues and Workers

18.10 BullMQ Queues and Workers [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Typed `new Queue<OrderEmailPayload • EmailResult • "send-receipt">(name • Opts)` • Production Backend Pitfall & Guardrail (BullMQ Queues and Workers) • Hands-On TypeScript / Node.js Implementation (BullMQ Queues and Workers) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for BullMQ Queues

Purpose: Draining active BullMQ jobs gracefully during container rolling deploys via `await worker.close()` before exiting.

What Topic Is Next → Next Topic 18.11: BullMQ Producers and Deduplicated Job IDs

18.11 BullMQ Producers and Deduplicated Job IDs [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `queue.add("send-receipt" • Payload • { jobId: \`receipt-\${orderId}\` })` — passing a deterministic `jobId` prevents duplicate waiting jobs for the same order automatically! • Production Backend Pitfall & Guardrail (BullMQ Producers and Deduplicated Job IDs) • Hands-On TypeScript / Node.js Implementation (BullMQ Producers and Deduplicated Job IDs) • BullMQ Producers and Deduplicated Job IDs — Latency, Throughput & Interview Trade-Offs • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for BullMQ Producers

Purpose: Ensuring 3 rapid status updates only enqueue 1 deduplicated indexing job via deterministic `jobId`.

What Topic Is Next → Next Topic 18.12: BullMQ Job Options and Memory Hygiene

18.12 BullMQ Job Options and Memory Hygiene [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `attempts` • `backoff` • `priority` (`1` = highest!) • **`removeOnComplete: { age: 3600 • Production Backend Pitfall & Guardrail (BullMQ Job Options and Memory Hygiene) • Hands-On TypeScript / Node.js Implementation (BullMQ Job Options and Memory Hygiene) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for BullMQ Job Options

Purpose: Keeping Redis memory flat over months of processing millions of jobs by bounding `removeOnComplete`!

What Topic Is Next → Next Topic 18.13: Delayed Jobs in BullMQ

18.13 Delayed Jobs in BullMQ [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `queue.add("expire-unpaid-order" • { orderId } • { delay: 15 * 60 * 1000 • JobId: \`expire-\${orderId}\` })` promoted automatically from ZSET to waiting stream when due • Production Backend Pitfall & Guardrail (Delayed Jobs in BullMQ) • Hands-On TypeScript / Node.js Implementation (Delayed Jobs in BullMQ) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Delayed Jobs in BullMQ

Purpose: Auto-releasing reserved flash-sale inventory if checkout payment isn't completed within 15 minutes.

What Topic Is Next → Next Topic 18.14: Repeatable / Job Schedulers in BullMQ

18.14 Repeatable / Job Schedulers in BullMQ [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `queue.upsertJobScheduler("nightly-reconcile" • { pattern: "0 2 * * *" • Tz: "Asia/Kolkata" } • { name: "reconcile" })` running cluster-safely ONCE per cron tick across 10 workers! • Production Backend Pitfall & Guardrail (Repeatable / Job Schedulers in BullMQ) • Hands-On TypeScript / Node.js Implementation (Repeatable / Job Schedulers in BullMQ) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Repeatable / Job Schedulers in BullMQ

Purpose: Replacing duplicate-firing `node-cron` with cluster-coordinated BullMQ Job Schedulers.

What Topic Is Next → Next Topic 18.15: Worker Concurrency and Outbound Rate Limiters

18.15 Worker Concurrency and Outbound Rate Limiters [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `concurrency: 15` (parallel async I/O jobs per worker) + `limiter: { max: 14 • Duration: 1000 }` (pacing outbound calls to respect third-party SMTP/SMS `14 emails/sec` provider limits!) • Production Backend Pitfall & Guardrail (Worker Concurrency and Outbound Rate Limiters) • Hands-On TypeScript / Node.js Implementation (Worker Concurrency and Outbound Rate Limiters) • Worker Concurrency and Outbound Rate Limiters — Latency, Throughput & Interview Trade-Offs • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Worker Concurrency • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards

Purpose: Blasting 50,000 order updates into BullMQ instantly while Workers smoothly drain them at `14 req/sec` without triggering vendor `429` bans!

What Topic Is Next → Next Topic 18.16: BullMQ Job Events and Progress Reporting

18.16 BullMQ Job Events and Progress Reporting [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `await job.updateProgress(45)` • Worker `.on("completed")` / `.on("failed")` / `.on("error")` (ALWAYS attach `.on("error")` on `Worker` & `Queue`!) • Cross-process `QueueEvents` • Production Backend Pitfall & Guardrail (BullMQ Job Events and Progress Reporting) • Hands-On TypeScript / Node.js Implementation (BullMQ Job Events and Progress Reporting) • BullMQ Job Events and Progress Reporting — Latency, Throughput & Interview Trade-Offs • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for BullMQ Job Events

Purpose: Updating CSV import progress (`0% -> 100%`) polled by the client status endpoint or pushed via Socket.IO!

What Topic Is Next → Next Topic 18.17: Failed Job Management & Bull Board

18.17 Failed Job Management & Bull Board [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Mounting `@bull-board/express` behind strict `ADMIN` RBAC • Inspecting stacktraces (`job.failedReason` • `job.stacktrace`) • `job.retry()` • Production Backend Pitfall & Guardrail (Failed Job Management & Bull Board) • Hands-On TypeScript / Node.js Implementation (Failed Job Management & Bull Board) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Failed Job Management

Purpose: Giving operations engineers a secure admin UI to inspect and replay failed jobs after an external vendor outage.

What Topic Is Next → Next Topic 18.18: RabbitMQ AMQP 0-9-1 Messaging Architecture

18.18 RabbitMQ AMQP 0-9-1 Messaging Architecture [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: TCP Connection (`1` per process!) -> Multiplexed lightweight `Channel`s -> `Exchange` (receives from Producer) -> `Bindings` -> `Queue` (buffers messages) -> `Consumer` (`amqplib`) • Production Backend Pitfall & Guardrail (RabbitMQ AMQP 0-9-1 Messaging Architecture) • Hands-On TypeScript / Node.js Implementation (RabbitMQ AMQP 0-9-1 Messaging Architecture) • RabbitMQ AMQP 0-9-1 Messaging Architecture — Latency, Throughput & Interview Trade-Offs • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for RabbitMQ AMQP 0-9-1 Messaging Architecture • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards • Diagnostics, Profiling & Telemetry: structured telemetry, profiler traces & CI checks

Purpose: Decoupling polyglot microservices (`Order Service` in Node -> `Analytics` in Python -> `Notification` in Node) via rich exchange routing.

What Topic Is Next → Next Topic 18.19: Exchanges and Queues (`durable: true`)

18.19 Exchanges and Queues (`durable: true`) [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Why Producers NEVER publish directly to a Queue in AMQP — Producers publish to an **Exchange** • Exchanges route copies to bound Queues (`assertExchange(... • { durable: true })` • `assertQueue(... • Production Backend Pitfall & Guardrail (Exchanges and Queues (`durable: true`)) • Hands-On TypeScript / Node.js Implementation (Exchanges and Queues (`durable: true`)) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Exchanges

Purpose: Adding a new `FraudAuditQueue` subscriber to `order.events` exchange tomorrow with ZERO changes to `OrderService`!

What Topic Is Next → Next Topic 18.20: Direct, Fanout, Topic and Headers Exchanges

18.20 Direct, Fanout, Topic and Headers Exchanges [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `direct` (exact routingKey match) • `fanout` (broadcast copy to ALL bound queues • E.g. • `user.deleted`!) • Production Backend Pitfall & Guardrail (Direct, Fanout, Topic and Headers Exchanges) • Hands-On TypeScript / Node.js Implementation (Direct, Fanout, Topic and Headers Exchanges) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Direct

Purpose: Broadcasting `order.paid` via `topic` exchange so `inventory-queue` (`order.paid`), `email-queue` (`order.*`), and `audit-queue` (`#`) each get their own independent copy...

What Topic Is Next → Next Topic 18.21: Routing Keys and Topic Wildcards

18.21 Routing Keys and Topic Wildcards [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Designing dot-delimited routing taxonomies `<domain>.<entity>.<event>.<region>` (`commerce.order.created.in`) • Matching `commerce.order.*.*` vs `commerce.#` • Production Backend Pitfall & Guardrail (Routing Keys and Topic Wildcards) • Hands-On TypeScript / Node.js Implementation (Routing Keys and Topic Wildcards) • Routing Keys and Topic Wildcards — Latency, Throughput & Interview Trade-Offs • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Routing Keys • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards

Purpose: Subscribing regional tax services only to `commerce.order.paid.in` while global analytics subscribes to `commerce.order.#`.

What Topic Is Next → Next Topic 18.22: Producers, Publisher Confirms and Consumer Prefetch

18.22 Producers, Publisher Confirms and Consumer Prefetch [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Using `createConfirmChannel()` (waits for broker disk persistence ack!) + checking `channel.publish()` boolean backpressure + setting **`channel.prefetch(10)`** on Consumers! • Production Backend Pitfall & Guardrail (Producers, Publisher Confirms and Consumer Prefetch) • Hands-On TypeScript / Node.js Implementation (Producers, Publisher Confirms and Consumer Prefetch) • Producers, Publisher Confirms and Consumer Prefetch — Latency, Throughput &

Interview Trade-Offs • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Producers • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards • Diagnostics, Profiling & Telemetry: structured telemetry, profiler traces & CI checks

Purpose: Setting `channel.prefetch(10)` so RabbitMQ doesn't dump `50,000` unacked messages into Worker #1's RAM while Worker #2 sits idle!

What Topic Is Next → Next Topic 18.23: Manual Acknowledgments (ack / nack)

18.23 Manual Acknowledgments (ack / nack) [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Setting `{ noAck: false }` • Calling `channel.ack(msg)` ONLY AFTER DB commit succeeds • Calling `channel.nack(msg • False • Production Backend Pitfall & Guardrail (Manual Acknowledgments (ack / nack)) • Hands-On TypeScript / Node.js Implementation (Manual Acknowledgments (ack / nack)) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Manual Acknowledgments (ack / nack)

Purpose: Guaranteeing zero message loss if a Node worker crashes mid-processing (RabbitMQ re-delivers unacked messages to another healthy worker!).

What Topic Is Next → Next Topic 18.24: RabbitMQ Retry Strategies (TTL Delay Queues)

18.24 RabbitMQ Retry Strategies (TTL Delay Queues) [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Implementing non-blocking delayed retries in RabbitMQ using Wait Queues (`x-message-ttl: 5000` • `x-dead-letter-exchange: main.ex`) tracking retry count in `msg.properties.headers["x-retry-count"]` • Production Backend Pitfall & Guardrail (RabbitMQ Retry Strategies (TTL Delay Queues)) • Hands-On TypeScript / Node.js Implementation (RabbitMQ Retry Strategies (TTL Delay Queues)) • RabbitMQ Retry Strategies (TTL Delay Queues) — Latency, Throughput & Interview Trade-Offs • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for RabbitMQ Retry Strategies (TTL Delay Queues) • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards

Purpose: Retrying transient webhook/email failures cleanly at `5s -> 30s -> 120s` before dead-lettering.

What Topic Is Next → Next Topic 18.25: Dead-Letter Exchanges (DLX) and Dead-Letter Queues (DLQ)

18.25 Dead-Letter Exchanges (DLX) and Dead-Letter Queues (DLQ) [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Configuring `arguments: { "x-dead-letter-exchange": "orders.dlx" • "x-dead-letter-routing-key": "orders.failed" }` on primary queues so rejected/expired messages land safely in `orders.dlq` • Production Backend Pitfall & Guardrail (Dead-Letter Exchanges (DLX) and Dead-Letter Queues (DLQ)) • Hands-On TypeScript / Node.js Implementation (Dead-Letter Exchanges (DLX) and Dead-Letter Queues (DLQ)) • Dead-Letter Exchanges (DLX) and Dead-Letter Queues (DLQ) — Latency, Throughput & Interview Trade-Offs • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Dead-Letter Exchanges (DLX) • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards

Purpose: Inspecting and replaying dead-lettered order events via RabbitMQ Management UI (`:15672`) or replay scripts.

What Topic Is Next → Next Topic 18.26: Apache Kafka Architecture (Distributed Commit Log)

18.26 Apache Kafka Architecture (Distributed Commit Log) [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Why Kafka is an **immutable append-only event log** (consumers pull & advance offsets • Reading a message does NOT delete it!) vs RabbitMQ queue deletion on `ack` • KRaft metadata mode (no ZooKeeper!) • `kafkajs` • Production Backend Pitfall & Guardrail (Apache Kafka Architecture (Distributed Commit Log)) • Hands-On TypeScript / Node.js Implementation (Apache Kafka Architecture (Distributed Commit Log)) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Apache Kafka Architecture (Distributed Commit Log)

Purpose: Allowing Payments, Fraud Detection, Warehouse Fulfillment, and Data Warehouse to each read the SAME `orders.events` stream at their own independent pace!

What Topic Is Next → Next Topic 18.27: Kafka Brokers and Topics

18.27 Kafka Brokers and Topics [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Multi-broker clusters • Topic retention (`retention.ms = 7 days` or `compact` log compaction keeping latest value per key) • Sequential disk I/O + OS pagecache zero-copy speed • Production Backend Pitfall & Guardrail (Kafka Brokers and Topics) • Hands-On TypeScript / Node.js Implementation (Kafka Brokers and Topics) • Kafka Brokers and Topics — Latency, Throughput & Interview Trade-Offs • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Kafka Brokers

Purpose: Replaying the last 48 hours of `payment-events` into a newly deployed Analytics microservice by resetting its consumer group offset!

What Topic Is Next → Next Topic 18.28: Partitions, Ordering Keys and Replication

18.28 Partitions, Ordering Keys and Replication [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Why **ordering in Kafka is guaranteed ONLY WITHIN a single Partition** -> hashing **`key: orderId`** (`murmur2(key) % numPartitions`) ensures `OrderCreated -> OrderPaid -> OrderShipped` for `order_42` ALWAYS land in the SAME partition in exact order!! • Production Backend Pitfall & Guardrail (Partitions, Ordering Keys and Replication) • Hands-On TypeScript / Node.js Implementation (Partitions, Ordering Keys and Replication) • Partitions, Ordering Keys and Replication — Latency, Throughput & Interview Trade-Offs • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Partitions • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards • Diagnostics, Profiling & Telemetry: structured telemetry, profiler traces & CI checks

Purpose: Passing `key: order.id` on every Kafka publish so `OrderShipped` is NEVER processed before `OrderCreated` for that order!

What Topic Is Next → Next Topic 18.29: Kafka Producers (acks, Idempotence & Batching)

18.29 Kafka Producers (acks, Idempotence & Batching) [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `acks: -1` (`all` in-sync replicas `ISR` confirm!) • `idempotent: true` producer sequence numbers preventing duplicate appends on network retry • `linger.ms` / `snappy` batching • Production Backend Pitfall & Guardrail (Kafka Producers (acks, Idempotence & Batching)) • Hands-On TypeScript / Node.js Implementation (Kafka Producers (acks, Idempotence & Batching)) • Kafka Producers (acks, Idempotence & Batching) — Latency, Throughput & Interview Trade-Offs • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Kafka Producers (acks, Idempotence & Batching)

Purpose: Publishing financial `PaymentCaptured` events with `acks: -1` and `idempotent: true` so broker leader failover loses zero committed events.

What Topic Is Next → Next Topic 18.30: Consumer Groups and Parallelism Ceiling

18.30 Consumer Groups and Parallelism Ceiling [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Within ONE `groupId` (`inventory-svc-group`) • Each Partition is assigned to AT MOST ONE consumer instance (`12 partitions` + `4 consumers` = `3 partitions each` • `12 partitions` + `15 consumers` = **`3 consumers sit 100% IDLE`**!) • While a DIFFERENT `groupId` (`email-svc-group`) receives a full independent stream! • Production Backend Pitfall & Guardrail (Consumer Groups and Parallelism Ceiling) • Hands-On TypeScript / Node.js Implementation (Consumer Groups and Parallelism Ceiling) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Consumer Groups

Purpose: Sizing topic partition count (`e.g., 12`) to match peak horizontal consumer scaling.

What Topic Is Next → Next Topic 18.31: Consumer Offsets, Commits and Event-Driven Concepts

18.31 Consumer Offsets, Commits and Event-Driven Concepts [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `__consumer_offsets` topic • Committing `offset + 1` ONLY AFTER processing + idempotent DB write succeeds (`autoCommit: false`) • Consumer lag monitoring (`latestOffset - committedOffset`) • Schema evolution (additive optional fields) • Production Backend Pitfall & Guardrail (Consumer Offsets, Commits and Event-Driven Concepts) • Hands-On TypeScript / Node.js Implementation (Consumer Offsets, Commits and Event-Driven Concepts) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Consumer Offsets

Purpose: Preventing silent message loss when a consumer crashes mid-handler by disabling blind premature `autoCommit`.

What Topic Is Next → Module 18 Capstone (Week 18 Lab Build: BullMQ Retry/DLQ Worker + RabbitMQ Topic Exchange & Kafka Partition Lab) → Next Topic 19.1: Real-Time Application Architecture

Module 18 Hands-On Capstone Project (100 Marks): Week 18 Lab Build: BullMQ Retry/DLQ Worker + RabbitMQ Topic Exchange & Kafka Partition Lab — Build a fault-tolerant background processing pipeline combining `BullMQ` (exponential backoff, deterministic `jobId` deduplication, DLQ inspection, PDF invoice generation) with a `RabbitMQ`...

MODULE 19 OF 24 • STAGE 5 • 25 LIVE + LAB HOURS • 20 TOPICS (160 SUBTOPICS) • 1 CAPSTONE PROJECT (100 MARKS)

Module 19: WebSockets and Real-Time Backend Development

Build authenticated, horizontally scalable real-time backends using WebSockets and Socket.IO — HTTP 101 upgrade internals, rooms & namespaces, private/group chat, Redis-backed online presence, JWT handshake & room RBAC, reconnection catch-up, @socket.io/redis-adapter multi-node scaling, and event-driven notification hubs.

Objectives: ✓ Contrast Short Polling, Long Polling, SSE, and full-duplex WebSockets across latency, directionality, and server overhead. • ✓ Build strictly typed `Socket.IO` servers in TypeScript supporting rooms, namespaces, private/group messaging, and presence. • ✓ Enforce JWT handshake authentication, per-room ownership authorization (`BOLA`), and Zod payload validation on WebSocket events. • ✓ Scale `Socket.IO` across multiple Node/Docker instances seamlessly using `@socket.io/redis-adapter`.

19.1 Real-Time Application Architecture [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Bidirectional low-latency use cases (chat • Collaborative cursors • Live bidding) vs unidirectional server-push (`SSE` for live order/ticker feeds) • Stateful TCP socket memory budgets • Production Backend Pitfall & Guardrail (Real-Time Application Architecture) • Hands-On TypeScript / Node.js Implementation (Real-Time Application Architecture) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Real-Time Application Architecture

Purpose: Pushing instant `"Your order #9042 is out for delivery!"` toast updates to connected web/mobile clients in `< 15ms`.

What Topic Is Next → Next Topic 19.2: Short Polling, Long Polling and SSE vs WebSockets

19.2 Short Polling, Long Polling and SSE vs WebSockets [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Short Polling (high wasted overhead) • HTTP Long Polling (held request) • Server-Sent Events (`text/event-stream` • HTTP/2 friendly one-way server->client) • Production Backend Pitfall & Guardrail (Short Polling, Long Polling and SSE vs WebSockets) • Hands-On TypeScript / Node.js Implementation (Short Polling, Long Polling and SSE vs WebSockets) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Short Polling

Purpose: Choosing `SSE` for simple one-way status streams and `WebSocket`/`Socket.IO` for bidirectional rooms, typing indicators, and acknowledgments.

What Topic Is Next → Next Topic 19.3: WebSocket Protocol (RFC 6455 Handshake & Framing)

19.3 WebSocket Protocol (RFC 6455 Handshake & Framing) [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `GET / HTTP/1.1` + `Upgrade: websocket` + `Sec-WebSocket-Key` -> `101 Switching Protocols` + `Sec-WebSocket-Accept` • Lightweight binary/text frames • Opcode `0x9` Ping / `0xA` Pong • Close codes (`1000` • Production Backend Pitfall & Guardrail (WebSocket Protocol (RFC 6455 Handshake & Framing)) • Hands-On TypeScript / Node.js Implementation (WebSocket Protocol (RFC 6455 Handshake & Framing)) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for WebSocket Protocol (RFC 6455 Handshake & Framing)

Purpose: Inspecting the `101 Switching Protocols` handshake in DevTools / `curl` to debug proxy issues.

What Topic Is Next → Next Topic 19.4: Managing Stateful WebSocket Connections & Heartbeats

19.4 Managing Stateful WebSocket Connections & Heartbeats [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Detecting half-open "zombie" TCP connections (mobile tunnel drop without FIN packet!) via `pingInterval` (`25s`) & `pingTimeout` (`20s`) • Bounding `maxHttpBufferSize` (`e.g. 64 KB` • Default `1 MB`!) • Production Backend Pitfall & Guardrail (Managing Stateful WebSocket Connections & Heartbeats) • Hands-On TypeScript / Node.js Implementation (Managing Stateful WebSocket Connections & Heartbeats) • Managing Stateful WebSocket Connections & Heartbeats — Latency, Throughput & Interview Trade-Offs • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Managing Stateful WebSocket Connections

Purpose: Reclaiming dead socket memory automatically within 45 seconds when mobile clients lose signal.

What Topic Is Next → Next Topic 19.5: Socket.IO Architecture vs Raw ws

19.5 Socket.IO Architecture vs Raw ws [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: What `Socket.IO` adds on top of WebSockets: automatic reconnection with backoff • Packet buffering • Acknowledgments (`emitWithAck`) • Rooms • Production Backend Pitfall & Guardrail (Socket.IO Architecture vs Raw ws) • Hands-On TypeScript / Node.js Implementation (Socket.IO Architecture vs Raw ws) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Socket.IO Architecture vs Raw ws

Purpose: Getting battle-tested rooms, acks, and Redis multi-server broadcast out of the box.

What Topic Is Next → Next Topic 19.6: Typed Server-Side Socket.IO Integration in TypeScript

19.6 Typed Server-Side Socket.IO Integration in TypeScript [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Attaching `new Server<ClientToServerEvents • ServerToClientEvents • InterServerEvents • SocketData>(httpServer • Production Backend Pitfall & Guardrail (Typed Server-Side Socket.IO Integration in TypeScript) • Hands-On TypeScript / Node.js Implementation (Typed Server-Side Socket.IO Integration in TypeScript) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Typed Server-Side Socket.IO Integration in TypeScrip...

Purpose: Getting 100% compile-time checking on event names and payload shapes (`socket.emit("order:updated", payload)`)!

What Topic Is Next → Next Topic 19.7: Connection and Disconnection Lifecycle Events

19.7 Connection and Disconnection Lifecycle Events [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `io.on("connection" • Socket => ...)` • `socket.on("disconnecting")` (`socket.rooms` is STILL populated here!) vs `socket.on("disconnect" • Reason => ...)` (`socket.rooms` is already emptied!) • Production Backend Pitfall & Guardrail (Connection and Disconnection Lifecycle Events) • Hands-On TypeScript / Node.js Implementation (Connection and Disconnection Lifecycle Events) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Connection

Purpose: Reading `socket.rooms` inside `"disconnecting"` (not `"disconnect"`!) when notifying room peers that a user left.

What Topic Is Next → Next Topic 19.8: Targeted Emits, Broadcasting and Acknowledgments

19.8 Targeted Emits, Broadcasting and Acknowledgments [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `socket.emit` (sender only) • `socket.broadcast.emit` (everyone EXCEPT sender) • `io.to(room).emit` (all in room) • Callback/Promise acknowledgments (`socket.timeout(5000).emitWithAck(...)`) • Production Backend Pitfall & Guardrail (Targeted Emits, Broadcasting and Acknowledgments) • Hands-On TypeScript / Node.js Implementation (Targeted Emits, Broadcasting and Acknowledgments) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Targeted Emits

Purpose: Confirming client receipt of critical trade/order alerts via `ack` callbacks.

What Topic Is Next → Next Topic 19.9: Rooms and Namespaces

19.9 Rooms and Namespaces [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Auto-joining every authenticated socket into `socket.join(\`user:\${socket.data.userId}\`)` (supports 3 open browser tabs + mobile app effortlessly!) • Resource rooms `order:${orderId}` • Isolated `/admin` Namespaces (`io.of("/admin")`) • Production Backend Pitfall & Guardrail (Rooms and Namespaces) • Hands-On TypeScript / Node.js Implementation (Rooms and Namespaces) • Rooms and Namespaces — Latency, Throughput & Interview Trade-Offs • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Rooms

Purpose: Pushing notifications to ALL open tabs/devices of User `u_42` with a single `io.to("user:u_42").emit(...)` call — zero manual `Map<userId, socketId>` bugs!

What Topic Is Next → Next Topic 19.10: Private 1-to-1 Messaging Architecture

19.10 Private 1-to-1 Messaging Architecture [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Persist-First flow: `1) Validate Zod -> 2) Authorize conversation membership -> 3) Persist message in PostgreSQL/MongoDB with clientIdempKey -> 4) Emit to user:${recipientId} & user:${senderId} -> 5) Ack messageId` • Production Backend Pitfall & Guardrail (Private 1-to-1 Messaging Architecture) • Hands-On TypeScript / Node.js Implementation (Private 1-to-1 Messaging Architecture) • Private 1-to-1 Messaging Architecture — Latency, Throughput & Interview Trade-Offs • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Private 1-to-1 Messaging Architecture • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards • Diagnostics, Profiling & Telemetry: structured telemetry, profiler traces & CI checks

Purpose: Ensuring messages sent while recipient is briefly offline in an elevator are safely stored in DB and fetched on reconnect!

What Topic Is Next → Next Topic 19.11: Group / Channel Messaging

19.11 Group / Channel Messaging [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Channel membership rooms (`channel:${channelId}`) • Typing indicators (`volatile.emit` — droppable if congested!) • Read-cursor watermarks (`lastReadMessageId`) • Bounded pagination • Production Backend Pitfall & Guardrail (Group / Channel Messaging) • Hands-On TypeScript / Node.js Implementation (Group / Channel Messaging) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Group / Channel Messaging

Purpose: Sending ephemeral `"Ravi is typing..."` events with `socket.to(room).volatile.emit(...)` so transient network lag never queues stale typing packets!

What Topic Is Next → Next Topic 19.12: Online and Offline Presence Tracking

19.12 Online and Offline Presence Tracking [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Multi-device accurate presence in Redis: `SADD presence:user:${uid} ${socket.id}` on connect + `SREM` on disconnect (`user is offline ONLY when SCARD === 0`!) + heartbeat TTL fallback for crashed nodes! • Production Backend Pitfall & Guardrail (Online and Offline Presence Tracking) • Hands-On TypeScript / Node.js Implementation (Online and Offline Presence Tracking) • Online and Offline Presence Tracking — Latency, Throughput & Interview Trade-Offs • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Online • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards • Diagnostics, Profiling & Telemetry: structured telemetry, profiler traces & CI checks

Purpose: Keeping User A `"Online"` while Tab 1 remains open even after Tab 2 closes (`SCARD === 1 > 0`)!

What Topic Is Next → Next Topic 19.13: Authentication for WebSockets (io.use Middleware)

19.13 Authentication for WebSockets (io.use Middleware) [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Verifying `socket.handshake.auth.token` (or `HttpOnly` cookie on handshake — NEVER `?token=jwt` in URL query string which leaks into Nginx/proxy access logs!!) inside `io.use(async (socket • Next) => ...)` + periodic token-expiry re-check! • Production Backend Pitfall & Guardrail (Authentication for WebSockets (io.use Middleware)) • Hands-On TypeScript / Node.js Implementation (Authentication for WebSockets (io.use Middleware)) • Authentication for WebSockets (io.use Middleware) — Latency, Throughput & Interview Trade-Offs • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Authentication for WebSockets (io.use Middleware) • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards

Purpose: Rejecting unauthenticated sockets BEFORE `"connection"` ever fires (`next(new Error("UNAUTHORIZED"))`).

What Topic Is Next → Next Topic 19.14: Authorization for Rooms (Preventing Real-Time BOLA)

19.14 Authorization for Rooms (Preventing Real-Time BOLA) [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Validating DB ownership inside `socket.on("order:subscribe" • Async ({ orderId } • Ack) => { const ok = await orderSvc.canView(socket.data.userId • OrderId) • Production Backend Pitfall & Guardrail (Authorization for Rooms (Preventing Real-Time BOLA)) • Hands-On TypeScript / Node.js Implementation (Authorization for Rooms (Preventing Real-Time BOLA)) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Authorization for Rooms (Preventing Real-Time BOLA)

Purpose: Blocking an attacker from emitting `socket.emit("join_room", "order:competitor_id")` to spy on live orders!

What Topic Is Next → Next Topic 19.15: Reconnection Strategies and Missed-Event Recovery

19.15 Reconnection Strategies and Missed-Event Recovery [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Randomized reconnection jitter (`reconnectionDelay` • `randomizationFactor: 0.5` preventing `10 • 000` clients reconnecting at the exact same millisecond after a deploy!) • Socket.IO Connection State Recovery • Production Backend Pitfall & Guardrail (Reconnection Strategies and Missed-Event Recovery) • Hands-On TypeScript / Node.js Implementation (Reconnection Strategies and Missed-Event Recovery) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Reconnection Strategies

Purpose: Staggering client reconnects after a rolling deploy so CPU/auth doesn’t spike.

What Topic Is Next → Next Topic 19.16: WebSocket Error Handling, Zod Validation & Event Rate Limiting

19.16 WebSocket Error Handling, Zod Validation & Event Rate Limiting [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Wrapping every `socket.on(...)` listener in a safe `withSocketGuard(schema • RateLimiter • Handler)` because an unhandled `async` rejection inside `socket.on()` crashes Node (Express `errorMiddleware` does NOT cover Socket.IO events!)! • Production Backend Pitfall & Guardrail (WebSocket Error Handling, Zod Validation & Event Rate Limiting) • Hands-On TypeScript / Node.js Implementation (WebSocket Error Handling, Zod Validation & Event Rate Limiting) • WebSocket Error Handling, Zod Validation & Event Rate Limiting — Latency, Throughput & Interview Trade-Offs • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for WebSocket Error Handling

Purpose: Validating incoming socket payloads with Zod and returning structured `{ ok: false, code: "VALIDATION_ERROR" }` acks without crashing.

What Topic Is Next → Next Topic 19.17: Horizontal Scaling Challenge for Stateful WebSockets

19.17 Horizontal Scaling Challenge for Stateful WebSockets [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Why scaling to `N` Node containers breaks in-memory `io.to("user:42").emit(...)` + why Socket.IO HTTP long-polling fallback requires sticky sessions (or `transports: ["websocket"]` direct upgrade!) • Production Backend Pitfall & Guardrail (Horizontal Scaling Challenge for Stateful WebSockets) • Hands-On TypeScript / Node.js Implementation (Horizontal Scaling Challenge for Stateful WebSockets) • Horizontal Scaling Challenge for Stateful WebSockets — Latency, Throughput & Interview Trade-Offs • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Horizontal Scaling Challenge for Stateful WebSockets • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards • Diagnostics, Profiling & Telemetry: structured telemetry, profiler traces & CI checks

Purpose: Configuring `transports: ["websocket"]` (or Nginx `ip_hash` sticky sessions) + Redis Pub/Sub adapter.

What Topic Is Next → Next Topic 19.18: Redis Adapter Fundamentals (@socket.io/redis-adapter)

19.18 Redis Adapter Fundamentals (@socket.io/redis-adapter) [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Creating dedicated `pubClient` • `subClient = pubClient.duplicate()` • Attaching `io.adapter(createAdapter(pubClient • SubClient))` so `io.to("user:42").emit(...)` fans out across ALL Node containers via Redis Pub/Sub! • Production Backend Pitfall & Guardrail (Redis Adapter Fundamentals (@socket.io/redis-adapter)) • Hands-On TypeScript / Node.js Implementation (Redis Adapter Fundamentals (@socket.io/redis-adapter)) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Redis Adapter Fundamentals (@socket.io/redis-adapter...

Purpose: Allowing a BullMQ Worker container (with `0` client sockets!) to push live events to any user on any API container via `@socket.io/redis-emitter`!

What Topic Is Next → Next Topic 19.19: Real-Time Notification Service Architecture

19.19 Real-Time Notification Service Architecture [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: End-to-end flow: Domain Service emits `OrderShipped` -> BullMQ/RabbitMQ Consumer persists `notifications` DB row (`isRead: false`) -> publishes via Redis Emitter to `user:${userId}` room -> unread badge updates in `< 20ms`! • Production Backend Pitfall & Guardrail (Real-Time Notification Service Architecture) • Hands-On TypeScript / Node.js Implementation (Real-Time Notification Service Architecture) • Real-Time Notification Service Architecture — Latency, Throughput & Interview Trade-Offs • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Real-Time Notification Service Architecture • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards • Diagnostics, Profiling & Telemetry: structured telemetry, profiler traces & CI checks

Purpose: Building the real-time Order Status Notification module used directly in the Month 6 Enterprise E-Commerce Capstone!

What Topic Is Next → Next Topic 19.20: Event-Driven Real-Time Design Patterns

19.20 Event-Driven Real-Time Design Patterns [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Out-of-order frame protection (`version` / sequence numbers on state updates so v2 doesn't overwrite v3!) • Backpressure coalescing • Graceful socket drain on deploy • Production Backend Pitfall & Guardrail (Event-Driven Real-Time Design Patterns) • Hands-On TypeScript / Node.js Implementation (Event-Driven Real-Time Design Patterns) • Event-Driven Real-Time Design Patterns — Latency, Throughput & Interview Trade-Offs • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Event-Driven Real-Time Design Patterns

Purpose: Including `updatedAt` / monotonic `version` on `order:status` payloads so clients ignore late out-of-order frames.

What Topic Is Next → Module 19 Capstone (Week 19 Lab Build: Multi-Node Authenticated Socket.IO Real-Time Notification & Room Server) → Next Topic 20.1: Monolithic vs Microservices Architecture

Module 19 Hands-On Capstone Project (100 Marks): Week 19 Lab Build: Multi-Node Authenticated Socket.IO Real-Time Notification & Room Server — Build a JWT-authenticated `Socket.IO` server using `@socket.io/redis-adapter` across 2 Node instances, auto-joining `user:${userId}` multi-tab rooms, enforcing server-side BOLA room-subscrip...

MODULE 20 OF 24 • STAGE 5 • 25 LIVE + LAB HOURS • 27 TOPICS (216 SUBTOPICS) • 1 CAPSTONE PROJECT (100 MARKS)

Module 20: Microservices and Distributed Backend Architecture

Design and build resilient distributed Node.js microservice systems — Modular Monolith vs Microservices, DDD bounded contexts, API Gateway, REST & gRPC communication, Database-per-Service, Transactional Outbox & Saga orchestration/choreography, idempotency, retries/jitter, Circuit Breakers, correlation tracing, service-to-service auth, and Month 5 Project 5.

Objectives: ✓ Define clean DDD Bounded Contexts and choose pragmatically between a Modular Monolith and Microservices. • ✓ Enforce Database-per-Service isolation and coordinate cross-service workflows using the **Saga Pattern** (compensating transactions) + **Transactional Outbox**. • ✓ Protect synchronous inter-service calls with strict timeouts, jittered retries, Circuit Breakers, Bulkheads, and scoped service JWTs. • ✓ Deliver **Month 5 Project (Project 5): 7-Service E-Commerce Microservices Backend**.

20.1 Monolithic vs Microservices Architecture [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Comparing Deployment simplicity • ACID single-DB joins • `0ms` in-process calls (Monolith) vs Independent team/service scaling • Fault isolation • Production Backend Pitfall & Guardrail (Monolithic vs Microservices Architecture) • Hands-On TypeScript / Node.js Implementation (Monolithic vs Microservices Architecture) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Monolithic vs Microservices Architecture

Purpose: Articulating mature architectural trade-offs instead of claiming `"microservices are always better"`.

What Topic Is Next → Next Topic 20.2: Modular Monolith Architecture

20.2 Modular Monolith Architecture [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Starting new products as a strictly modularized single deployable (`modules/users` • `modules/catalog` • `modules/orders` with zero cross-schema SQL joins!) • Extracting only proven bottleneck/independent services • Production Backend Pitfall & Guardrail (Modular Monolith Architecture) • Hands-On TypeScript / Node.js Implementation (Modular Monolith Architecture) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Modular Monolith Architecture

Purpose: Moving fast on Day 1 while keeping clean module seams ready for painless extraction into microservices.

What Topic Is Next → Next Topic 20.3: Microservice Principles (Single Responsibility & Loose Coupling)

20.3 Microservice Principles (Single Responsibility & Loose Coupling) [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Independent deployability • Backward-compatible API/event contracts • Autonomous data ownership • Resilience when peer services are down • Production Backend Pitfall & Guardrail (Microservice Principles (Single Responsibility & Loose Coupling)) • Hands-On TypeScript / Node.js Implementation (Microservice Principles (Single Responsibility & Loose Coupling)) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Microservice Principles (Single Responsibility & Loo...

Purpose: Ensuring `CatalogService` & `CartService` keep serving shoppers even while `NotificationService` restarts!

What Topic Is Next → Next Topic 20.4: Defining Clean Service Boundaries

20.4 Defining Clean Service Boundaries [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Splitting by **Business Capability** (`User` • `Product Catalog` • `Inventory` • `Cart` • Production Backend Pitfall & Guardrail (Defining Clean Service Boundaries) • Hands-On TypeScript / Node.js Implementation (Defining Clean Service Boundaries) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Defining Clean Service Boundaries

Purpose: Structuring all 7 services of Month 5 Project 5 around cohesive business capabilities.

What Topic Is Next → Next Topic 20.5: Domain-Driven Design (DDD) Bounded Contexts Fundamentals

20.5 Domain-Driven Design (DDD) Bounded Contexts Fundamentals [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Why `"Product"` means `{ title • Description • Images • Specs }` inside **Catalog Context** (`MongoDB`) • Production Backend Pitfall & Guardrail (Domain-Driven Design (DDD) Bounded Contexts Fundamentals) • Hands-On TypeScript / Node.js Implementation (Domain-Driven Design (DDD) Bounded Contexts Fundamentals) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Domain-Driven Design (DDD) Bounded Contexts Fundamen...

Purpose: Eliminating bloated 90-column God-Models by giving each Bounded Context its own lean focused representation!

What Topic Is Next → Next Topic 20.6: Service-to-Service Communication Overview

20.6 Service-to-Service Communication Overview [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Choosing **Synchronous Request/Response** (`REST` / `gRPC` for immediate read queries) vs **Asynchronous Event-Driven** (`RabbitMQ` / `Kafka` for state changes & multi-step workflows) • Production Backend Pitfall & Guardrail (Service-to-Service Communication Overview) • Hands-On TypeScript / Node.js Implementation (Service-to-Service Communication Overview) • Service-to-Service Communication Overview — Latency, Throughput & Interview Trade-Offs • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Service-to-Service Communication Overview • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards • Diagnostics, Profiling & Telemetry: structured telemetry, profiler traces & CI checks

Purpose: Using sync REST/gRPC when `CartService` validates a live SKU price, and async events for `OrderCreated -> ReserveStock -> ProcessPayment -> Notify`!

What Topic Is Next → Next Topic 20.7: REST-Based Inter-Service Communication

20.7 REST-Based Inter-Service Communication [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Internal HTTP clients using pooled `fetch` / `undici` keep-alive connections • Mandatory `AbortSignal.timeout(3000)` • Propagating `X-Correlation-Id` • Strict Zod response validation (`OWASP API10`) • Production Backend Pitfall & Guardrail (REST-Based Inter-Service Communication) • Hands-On TypeScript / Node.js Implementation (REST-Based Inter-Service Communication) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for REST-Based Inter-Service Communication

Purpose: Validating internal responses from `product-svc` so unexpected schema drift fails cleanly instead of corrupting `cart-svc`.

What Topic Is Next → Next Topic 20.8: gRPC and Protocol Buffers Fundamentals

20.8 gRPC and Protocol Buffers Fundamentals [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Contract-first `.proto` IDL schemas • Strongly typed generated TypeScript stubs • Compact binary Protobuf serialization (`3–7x` smaller than JSON) • Multiplexed HTTP/2 streams • Production Backend Pitfall & Guardrail (gRPC and Protocol Buffers Fundamentals) • Hands-On TypeScript / Node.js Implementation (gRPC and Protocol Buffers Fundamentals) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for gRPC

Purpose: Understanding why high-QPS internal `Order -> Inventory` stock checks often adopt `gRPC` while edge Gateway stays REST+JSON.

What Topic Is Next → Next Topic 20.9: API Gateway Architecture

20.9 API Gateway Architecture [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Single external entrypoint (`gateway` / `Nginx` + Gateway service) handling TLS • CORS • Edge Rate Limiting • JWT verification (forwarding verified `X-User-Id`/`X-User-Role` or internal JWT) • Production Backend Pitfall & Guardrail (API Gateway Architecture) • Hands-On TypeScript / Node.js Implementation (API Gateway Architecture) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for API Gateway Architecture

Purpose: Shielding internal microservices on a private Docker network so clients ONLY talk to the Gateway on port `80/443`.

What Topic Is Next → Next Topic 20.10: Service Discovery

20.10 Service Discovery [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Client-side vs Server-side discovery • How Docker Compose internal DNS (`http://inventory-svc:3003`) • Container orchestrator Services resolve healthy container IPs automatically without hardcoded IPs! • Production Backend Pitfall & Guardrail (Service Discovery) • Hands-On TypeScript / Node.js Implementation (Service Discovery) • Service Discovery — Latency, Throughput & Interview Trade-Offs • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Service Discovery

Purpose: Configuring `INVENTORY_SVC_URL=http://inventory-svc:3003` via environment variables.

What Topic Is Next → Next Topic 20.11: Centralized and Validated Service Configuration

20.11 Centralized and Validated Service Configuration [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Shared typed config validation pattern per service (`PORT` • `SERVICE_NAME` • `LOG_LEVEL` • Own `DATABASE_URL` • Production Backend Pitfall & Guardrail (Centralized and Validated Service Configuration) • Hands-On TypeScript / Node.js Implementation (Centralized and Validated Service Configuration) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Centralized

Purpose: Ensuring every microservice stamps `{ service: config.SERVICE_NAME }` onto every log line and trace span.

What Topic Is Next → Next Topic 20.12: Database-per-Service Principle

20.12 Database-per-Service Principle [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `user-svc` owns `users_db` (Postgres) • `product-svc` owns `catalog_db` (MongoDB) • `cart-svc` owns `cart_redis` (Redis) • `inventory-svc` owns `inventory_db` (Postgres) • Production Backend Pitfall & Guardrail (Database-per-Service Principle) • Hands-On TypeScript / Node.js Implementation (Database-per-Service Principle) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Database-per-Service Principle

Purpose: Allowing `product-svc` to scale MongoDB independently while `order-svc` runs ACID PostgreSQL migrations without coordinating downtime!

What Topic Is Next → Next Topic 20.13: Event-Driven Microservice Architecture

20.13 Event-Driven Microservice Architecture [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Publishing immutable past-tense **Domain Events** (`OrderCreated` • `StockReserved` • `StockInsufficient` • `PaymentSucceeded` • Production Backend Pitfall & Guardrail (Event-Driven Microservice Architecture) • Hands-On TypeScript / Node.js Implementation (Event-Driven Microservice Architecture) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Event-Driven Microservice Architecture

Purpose: Decoupling `OrderService` from knowing how `NotificationService` sends emails or SMS.

What Topic Is Next → Next Topic 20.14: Message Brokers in Microservices (RabbitMQ vs Kafka)

20.14 Message Brokers in Microservices (RabbitMQ vs Kafka) [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: When RabbitMQ topic exchanges shine (task routing • Flexible bindings • Per-message TTL/DLQ) vs when Kafka shines (high-throughput ordered event streaming • Multi-day replay) • Production Backend Pitfall & Guardrail (Message Brokers in Microservices (RabbitMQ vs Kafka)) • Hands-On TypeScript / Node.js Implementation (Message Brokers in Microservices (RabbitMQ vs Kafka)) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Message Brokers in Microservices (RabbitMQ vs Kafka)

Purpose: Implementing Project 5's event bus cleanly over RabbitMQ topic exchanges (or Kafka topics) behind an `EventBusPort` interface!

What Topic Is Next → Next Topic 20.15: Synchronous vs Asynchronous Communication Matrix

20.15 Synchronous vs Asynchronous Communication Matrix [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Queries needing immediate read response (`GET /cart`) -> Sync HTTP/gRPC + Cache • Cross-service multi-step state mutations (`Checkout Saga`) -> Async Events / Commands! • Production Backend Pitfall & Guardrail (Synchronous vs Asynchronous Communication Matrix) • Hands-On TypeScript / Node.js Implementation (Synchronous vs Asynchronous Communication Matrix) • Synchronous vs Asynchronous Communication Matrix — Latency, Throughput & Interview Trade-Offs • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Synchronous vs Asynchronous Communication Matrix • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards

Purpose: Returning `202 / 201 { orderId, status: "PENDING_CONFIRMATION" }` immediately while the Saga reserves stock & captures payment asynchronously (pushing live update via Soc...

What Topic Is Next → Next Topic 20.16: Why Distributed 2PC Transactions Fail & the Dual-Write Trap

20.16 Why Distributed 2PC Transactions Fail & the Dual-Write Trap [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Why cross-service `XA / 2PC` locks kill availability • Why naive `await db.saveOrder() • Await broker.publish("OrderCreated") • ` loses events if the process crashes between lines 1 • Production Backend Pitfall & Guardrail (Why Distributed 2PC Transactions Fail & the Dual-Write Trap) • Hands-On TypeScript / Node.js Implementation (Why Distributed 2PC Transactions Fail & the Dual-Write Trap) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Why Distributed 2PC Transactions Fail

Purpose: Solving Dual-Write atomically via the **Transactional Outbox** (`INSERT INTO orders` + `INSERT INTO outbox_events` in ONE single local Postgres transaction, relayed to Ra...

What Topic Is Next → Next Topic 20.17: Saga Pattern Fundamentals (Choreography, Orchestration & Compensations)

20.17 Saga Pattern Fundamentals (Choreography, Orchestration & Compensations) [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Forward steps + **Compensating Transactions** on failure: `1) Create Order (PENDING) -> 2) Reserve Stock -> 3) Charge Payment -> [IF PAYMENT FAILS] -> Compensate: Release Stock -> Mark Order PAYMENT_FAILED`! • Production Backend Pitfall & Guardrail (Saga Pattern Fundamentals (Choreography, Orchestration & Compensations)) • Hands-On TypeScript / Node.js Implementation (Saga Pattern Fundamentals (Choreography, Orchestration & Compensations)) • Saga Pattern Fundamentals (Choreography, Orchestration & Compensations) — Latency, Throughput & Interview Trade-Offs • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Saga Pattern Fundamentals (Choreography, Orchestrati... • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards • Diagnostics, Profiling & Telemetry: structured telemetry, profiler traces & CI checks

Purpose: Guaranteeing reserved stock NEVER stays leaked/locked when a customer's credit card declines in Step 3!

What Topic Is Next → Next Topic 20.18: Eventual Consistency in Practice

20.18 Eventual Consistency in Practice [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Read-your-own-writes UX • Maintaining local read projections (`order-svc` listens to `ProductUpdated` or snapshots SKU summary) • Handling out-of-order event arrival • Production Backend Pitfall & Guardrail (Eventual Consistency in Practice) • Hands-On TypeScript / Node.js Implementation (Eventual Consistency in Practice) • Eventual Consistency in Practice — Latency, Throughput & Interview Trade-Offs • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Eventual Consistency in Practice

Purpose: Returning `status: "PROCESSING"` immediately on order creation and transitioning cleanly to `"CONFIRMED"` within `~150ms`.

What Topic Is Next → Next Topic 20.19: End-to-End Idempotency in Distributed Systems

20.19 End-to-End Idempotency in Distributed Systems [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Recording `processed_events(event_id PRIMARY KEY • Processed_at)` INSIDE the same local DB transaction as the consumer's state update (`Inventory` stock deduction)! • Production Backend Pitfall & Guardrail (End-to-End Idempotency in Distributed Systems) • Hands-On TypeScript / Node.js Implementation (End-to-End Idempotency in Distributed Systems) • End-to-End Idempotency in Distributed Systems — Latency, Throughput & Interview Trade-Offs • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for End-to-End Idempotency in Distributed Systems • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards

Purpose: Guaranteeing that if `OrderCreated` (`evt_99`) is delivered 3 times by RabbitMQ/Kafka, `InventoryService` deducts stock **ONCE**!

What Topic Is Next → Next Topic 20.20: Retries and Exponential Backoff with Full Jitter

20.20 Retries and Exponential Backoff with Full Jitter [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Why synchronized fixed-interval retries (`100` callers all retrying at exact `+2.000s` • `+4.000s`) hammer a recovering service back down vs **Full Jitter** `random(0 • Min(cap • Base * 2^attempt))`! • Production Backend Pitfall & Guardrail (Retries and Exponential Backoff with Full Jitter) • Hands-On TypeScript / Node.js Implementation (Retries and Exponential Backoff with Full Jitter) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Retries

Purpose: Smoothing recovery load across time when an internal service restarts.

What Topic Is Next → Next Topic 20.21: Circuit Breakers (CLOSED -> OPEN -> HALF_OPEN)

20.21 Circuit Breakers (CLOSED -> OPEN -> HALF_OPEN) [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Tracking rolling failure rate: when downstream `payment-svc` hits `>= 5` consecutive timeouts/5xx • Trip **OPEN** (fail fast in `0.1ms` for `15s` cooldown without tying up sockets!) -> test 1 probe in **HALF_OPEN** -> **CLOSED** on success! • Production Backend Pitfall & Guardrail (Circuit Breakers (CLOSED -> OPEN -> HALF_OPEN)) • Hands-On TypeScript / Node.js Implementation (Circuit Breakers (CLOSED -> OPEN -> HALF_OPEN)) • Circuit Breakers (CLOSED -> OPEN -> HALF_OPEN) — Latency, Throughput & Interview Trade-Offs • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Circuit Breakers (CLOSED -> OPEN -> HALF_OPEN) • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards

Purpose: Preventing a dead optional Recommendation/Analytics service from dragging down Checkout!

What Topic Is Next → Next Topic 20.22: Timeouts and Bulkhead Isolation

20.22 Timeouts and Bulkhead Isolation [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Propagating deadline budgets (`AbortSignal.timeout`) • **Bulkheads** (separate connection/concurrency pools per downstream dependency so a slow `ReportingService` cannot starve `InventoryService` HTTP sockets!) • Production Backend Pitfall & Guardrail (Timeouts and Bulkhead Isolation) • Hands-On TypeScript / Node.js Implementation (Timeouts and Bulkhead Isolation) • Timeouts and Bulkhead Isolation — Latency, Throughput & Interview Trade-Offs • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Timeouts • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards

Purpose: Capping concurrent outbound calls to `recommendation-svc` at `max = 10` (`Bulkhead`) so `inventory-svc` retains full capacity.

What Topic Is Next → Next Topic 20.23: Distributed Tracing Fundamentals (OpenTelemetry)

20.23 Distributed Tracing Fundamentals (OpenTelemetry) [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: W3C `traceparent` header (`00-<traceId>-<spanId>-01`) propagating one `traceId` across HTTP calls • RabbitMQ/Kafka message headers (`OpenTelemetry` Node SDK -> `Jaeger`) • Production Backend Pitfall & Guardrail (Distributed Tracing Fundamentals (OpenTelemetry)) • Hands-On TypeScript / Node.js Implementation (Distributed Tracing Fundamentals (OpenTelemetry)) • Distributed Tracing Fundamentals (OpenTelemetry) — Latency, Throughput & Interview Trade-Offs • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Distributed Tracing Fundamentals (OpenTelemetry) • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards

Purpose: Viewing one visual waterfall flamegraph spanning `Gateway -> OrderService -> RabbitMQ -> InventoryService -> PaymentService`!

What Topic Is Next → Next Topic 20.24: Correlation IDs Across Async Flows

20.24 Correlation IDs Across Async Flows [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Generating `correlationId` at API Gateway entry • Storing in `AsyncLocalStorage` • Forwarding in every outbound HTTP call • Outbox row • Production Backend Pitfall & Guardrail (Correlation IDs Across Async Flows) • Hands-On TypeScript / Node.js Implementation (Correlation IDs Across Async Flows) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Correlation IDs Across Async Flows

Purpose: Grepping `"correlationId":"cor_882a"` across all 7 service logs and seeing the complete chronological story of Order `#9042`!

What Topic Is Next → Next Topic 20.25: Centralized Structured Logging

20.25 Centralized Structured Logging [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Standardizing Pino JSON log schema (`timestamp` • `level` • `service` • `env` • Production Backend Pitfall & Guardrail (Centralized Structured Logging) • Hands-On TypeScript / Node.js Implementation (Centralized Structured Logging) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Centralized Structured Logging

Purpose: Writing logs strictly to `stdout` inside Docker containers so container runtimes collect & rotate logs centrally.

What Topic Is Next → Next Topic 20.26: Microservice Security (Zero-Trust Internal Network)

20.26 Microservice Security (Zero-Trust Internal Network) [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Why `"it's on the internal Docker network"` is NOT sufficient auth: validating caller identity • Scoping `aud: "inventory-svc"` • Network segmentation (DBs unreachable from unrelated services) • Rate-limiting internal callers • Production Backend Pitfall & Guardrail (Microservice Security (Zero-Trust Internal Network)) • Hands-On TypeScript / Node.js Implementation (Microservice Security (Zero-Trust Internal Network)) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Microservice Security (Zero-Trust Internal Network)

Purpose: Ensuring a SSRF bug in a peripheral service cannot call `payment-svc` internal refund endpoints unauthenticated.

What Topic Is Next → Next Topic 20.27: Service-to-Service Authentication (Asymmetric JWT / HMAC & Project 5)

20.27 Service-to-Service Authentication (Asymmetric JWT / HMAC & Project 5) [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Signing short-lived internal tokens (`iss: "order-svc"` • `aud: "inventory-svc"` • `exp: +60s`) verified offline via `RS256`/`Ed25519` public keys (`JWKS`) — • Delivering **Project 5: E-Commerce Microservices Backend**! • Production Backend Pitfall & Guardrail (Service-to-Service Authentication (Asymmetric JWT / HMAC & Project 5)) • Hands-On TypeScript / Node.js Implementation (Service-to-Service Authentication (Asymmetric JWT / HMAC & Project 5)) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Service-to-Service Authentication (Asymmetric JWT /...

Purpose: Verifying tokens across all 7 services using asymmetric public keys so only `auth-svc` holds the private user-signing key!

What Topic Is Next → Module 20 Capstone (Project 5: E-Commerce Microservices Backend) → Next Topic 21.1: Node.js CPU Profiling and Flamegraphs

Module 20 Hands-On Capstone Project (100 Marks): Project 5: E-Commerce Microservices Backend — When engineering orgs and domain throughput outgrow a single deployment unit, backend engineers must decompose bounded contexts cleanly without accidentally building a fragile "distributed m...

MODULE 21 OF 24 • STAGE 6 • 25 LIVE + LAB HOURS • 22 TOPICS (176 SUBTOPICS) • 1 CAPSTONE PROJECT (100 MARKS)

Module 21: Advanced Node.js Performance and Scalability

Diagnose and eliminate real-world Node.js production bottlenecks — CPU flamegraphs, event-loop lag, memory leaks & V8 heap snapshots, worker_threads/cluster sizing, DB/HTTP connection pooling, index/query tuning, L1+L2 caching, autocannon/k6 load testing, graceful load shedding, health/readiness probes, and high availability.

Objectives: ✓ Capture and read V8 `--cpu-prof` flamegraphs and `--heapsnapshot` retainer trees to pinpoint CPU bottlenecks and memory leaks. • ✓ Right-size V8 `--max-old-space-size`, `libuv` threadpools, HTTP `undici` keep-alive pools, and PostgreSQL/MongoDB connection pools. • ✓ Stress-test APIs with `autocannon` / `k6`, enforce backpressure/load-shedding, and separate `/healthz` (liveness) from `/readyz` (readiness). • ✓ Design stateless horizontally scalable Node.js architectures with zero-downtime rolling reloads.

21.1 Node.js CPU Profiling and Flamegraphs [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `node --cpu-prof dist/server.js` (or `inspector` `Profiler.start`) • Reading wide flat plateaus on Flamegraphs (wide bar = CPU time spent IN that function!) • `clinic flame` • Production Backend Pitfall & Guardrail (Node.js CPU Profiling and Flamegraphs) • Hands-On TypeScript / Node.js Implementation (Node.js CPU Profiling and Flamegraphs) • Node.js CPU Profiling and Flamegraphs — Latency, Throughput & Interview Trade-Offs • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Node.js CPU Profiling

Purpose: Spotting in 30 seconds that an accidental synchronous `zod` schema re-compilation inside a tight loop consumed `68%` of CPU!

What Topic Is Next → Next Topic 21.2: Measuring and Alerting on Event-Loop Delay

21.2 Measuring and Alerting on Event-Loop Delay [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `monitorEventLoopDelay` `p50`/`p95`/`p99` (`healthy < 20ms` • `degraded > 100ms`) • Why high event-loop lag delays `setTimeout` timeouts • Health checks • Production Backend Pitfall & Guardrail (Measuring and Alerting on Event-Loop Delay) • Hands-On TypeScript / Node.js Implementation (Measuring and Alerting on Event-Loop Delay) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Measuring

Purpose: Returning `503 Service Unavailable` (`load shedding`) temporarily if `eventLoopLagP99 > 250ms` to let in-flight work drain instead of spiraling into total timeout death!

What Topic Is Next → Next Topic 21.3: CPU-Bound vs I/O-Bound Workloads

21.3 CPU-Bound vs I/O-Bound Workloads [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Classifying JSON serialization of 50k rows • Image/PDF rendering • Regex • Compression • Production Backend Pitfall & Guardrail (CPU-Bound vs I/O-Bound Workloads) • Hands-On TypeScript / Node.js Implementation (CPU-Bound vs I/O-Bound Workloads) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for CPU-Bound vs I/O-Bound Workloads

Purpose: Streaming/paginating JSON responses (`<= 100` items) so `JSON.stringify` never blocks the main thread for `80ms`.

What Topic Is Next → Next Topic 21.4: Eliminating Hidden Blocking Operations (ReDoS & Sync Calls)

21.4 Eliminating Hidden Blocking Operations (ReDoS & Sync Calls) [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Auditing vulnerable nested-quantifier regexes (`/^(a+)+$/` -> `O(2^n)` CPU freeze on `"aaaaaaaaaaaaaaaaaaaaaaaaX"`!) • Replacing with linear regexes/`re2`/bounded string lengths (`<= 256` chars before regex!) • Production Backend Pitfall & Guardrail (Eliminating Hidden Blocking Operations (ReDoS & Sync Calls)) • Hands-On TypeScript / Node.js Implementation (Eliminating Hidden Blocking Operations (ReDoS & Sync Calls)) • Eliminating Hidden Blocking Operations (ReDoS & Sync Calls) — Latency, Throughput & Interview Trade-Offs • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Eliminating Hidden Blocking Operations (ReDoS & Sync... • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards

Purpose: Capping email/slug/search length BEFORE running any regular expression so `ReDoS` input sizes are impossible!

What Topic Is Next → Next Topic 21.5: Memory Leaks and V8 Heap Snapshots

21.5 Memory Leaks and V8 Heap Snapshots [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Taking TWO heap snapshots (`v8.writeHeapSnapshot()` at `t=1m` after warmup • `t=10m` after 10 • 000 requests) -> diffing `Delta > 0` allocated objects & inspecting the **Retainers** path back to GC roots! • Production Backend Pitfall & Guardrail (Memory Leaks and V8 Heap Snapshots) • Hands-On TypeScript / Node.js Implementation (Memory Leaks and V8 Heap Snapshots) • Memory Leaks and V8 Heap Snapshots — Latency, Throughput & Interview Trade-Offs • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Memory Leaks

Purpose: Pinpointing an unremoved `EventEmitter` listener or unbounded `Map` cache growing `20 MB` every 1,000 requests.

What Topic Is Next → Next Topic 21.6: Garbage Collection Fundamentals & Container Memory Tuning

21.6 Garbage Collection Fundamentals & Container Memory Tuning [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Scavenge (fast `< 2ms` nursery copy) vs Major Mark-Sweep-Compact • Setting `NODE_OPTIONS="--max-old-space-size=384"` inside a `512Mi` Docker container (leaving headroom for RSS buffers • C++ OpenSSL/Prisma & stack so Linux `OOMKiller` doesn't `SIGKILL 137` the container!) • Production Backend Pitfall & Guardrail (Garbage Collection Fundamentals & Container Memory Tuning) • Hands-On TypeScript / Node.js Implementation (Garbage Collection Fundamentals & Container Memory Tuning) • Garbage Collection Fundamentals & Container Memory Tuning — Latency, Throughput & Interview Trade-Offs • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Garbage Collection Fundamentals

Purpose: Sizing `--max-old-space-size` to `~75%` of container memory limit (`384 MB` for `512 MB` container; `768 MB` for `1024 MB`).

What Topic Is Next → Next Topic 21.7: Worker Threads Pool Sizing

21.7 Worker Threads Pool Sizing [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Sizing fixed worker-thread pools (`Math.max(1 • Os.availableParallelism() - 1)`) with bounded input queues (`maxQueue`) rejecting with `503` on overload • Production Backend Pitfall & Guardrail (Worker Threads Pool Sizing) • Hands-On TypeScript / Node.js Implementation (Worker Threads Pool Sizing) •

Worker Threads Pool Sizing — Latency, Throughput & Interview Trade-Offs • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Worker Threads Pool Sizing • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards

Purpose: Keeping 1 CPU core free for the main HTTP event loop while worker threads crunch CSV/crypto tasks.

What Topic Is Next → Next Topic 21.8: Child Processes for Isolated Heavy External Tools

21.8 Child Processes for Isolated Heavy External Tools [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Streaming `stdin`/`stdout` to external CLI tools (`pg_dump` • Image/document tools) with hard `timeout` • `killSignal` • Concurrency semaphore caps • Production Backend Pitfall & Guardrail (Child Processes for Isolated Heavy External Tools) • Hands-On TypeScript / Node.js Implementation (Child Processes for Isolated Heavy External Tools) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Child Processes for Isolated Heavy External Tools

Purpose: Preventing a hung external subprocess from leaking zombie processes (`cgroups` / explicit `kill`).

What Topic Is Next → Next Topic 21.9: Cluster Module vs Container Replicas

21.9 Cluster Module vs Container Replicas [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: When `node:cluster` / `PM2 -i max` wins (single Linux VPS) vs when `1` single-process Node container × `N` Docker/orchestrator replicas wins (clean per-process cgroup limits • Independent restarts • Simpler metrics) • Production Backend Pitfall & Guardrail (Cluster Module vs Container Replicas) • Hands-On TypeScript / Node.js Implementation (Cluster Module vs Container Replicas) • Cluster Module vs Container Replicas — Latency, Throughput & Interview Trade-Offs • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Cluster Module vs Container Replicas

Purpose: Deploying `PM2` cluster mode on a `4-vCPU` Linux VPS vs `replicas: 4` single-process containers behind Nginx.

What Topic Is Next → Next Topic 21.10: Process Management and Crash Auto-Healing

21.10 Process Management and Crash Auto-Healing [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Restart backoff delays (`exp_backoff_restart_delay`) • Max memory restart thresholds (`max_memory_restart: "450M"` safety net while fixing leaks) • Crash-loop alerts • Production Backend Pitfall & Guardrail (Process Management and Crash Auto-Healing) • Hands-On TypeScript / Node.js Implementation (Process Management and Crash Auto-Healing) • Process Management and Crash Auto-Healing — Latency, Throughput & Interview Trade-Offs • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Process Management

Purpose: Preventing a broken DB credential from restarting Node `1,000 times/minute` pinning CPU at `100%`!

What Topic Is Next → Next Topic 21.11: Connection Pooling Across DB, Redis and Outbound HTTP

21.11 Connection Pooling Across DB, Redis and Outbound HTTP [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: End-to-end pool math: PostgreSQL (`connection_limit`) • MongoDB (`maxPoolSize`) • Redis (`ioredis`) • Node's built-in `undici` global `fetch` connection poolkeep-alive! • Production Backend Pitfall & Guardrail (Connection Pooling Across DB, Redis and Outbound HTTP) • Hands-On TypeScript / Node.js Implementation (Connection Pooling Across DB, Redis and Outbound HTTP) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Connection Pooling Across DB

Purpose: Ensuring `replicas (4) * pgPool (12) + workers (2) * pgPool (6) = 60` stays safely under PostgreSQL `max_connections = 100`!

What Topic Is Next → Next Topic 21.12: Database Query Optimization at Scale

21.12 Database Query Optimization at Scale [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Eliminating `SELECT *` • Batching `IN (...)` lookups • Pushing aggregations into SQL/Mongo • Read-replica routing for heavy analytics reports • Production Backend Pitfall & Guardrail (Database Query Optimization at Scale) • Hands-On TypeScript / Node.js Implementation (Database Query Optimization at Scale) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Database Query Optimization at Scale

Purpose: Setting PostgreSQL `statement_timeout = '5s'` on the `app_runtime` role so one pathological query is cancelled automatically instead of locking up the pool!

What Topic Is Next → Next Topic 21.13: Index Optimization & Write-Amplification Balance

21.13 Index Optimization & Write-Amplification Balance [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Inspecting `pg_stat_user_indexes` (`idx_scan = 0` unused indexes slow down every `INSERT`/`UPDATE` • Waste RAM!) • Covering `INCLUDE` indexes enabling `Index Only Scan` (`Heap Fetches: 0`) • Production Backend Pitfall & Guardrail (Index Optimization & Write-Amplification Balance) • Hands-On TypeScript / Node.js Implementation (Index Optimization & Write-Amplification Balance) • Index Optimization & Write-Amplification Balance — Latency, Throughput & Interview Trade-Offs • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Index Optimization

Purpose: Dropping 6 unused redundant single-column indexes while adding 1 surgical covering index that speeds up writes by `25%` and reads by `10x`!

What Topic Is Next → Next Topic 21.14: API Response Optimization (Compression, ETag & Projection)

21.14 API Response Optimization (Compression, ETag & Projection) [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Selective `brotli`/`gzip` above `1 KB` (preferring compression at **Nginx** reverse proxy so Node's main thread/threadpool saves `zlib` CPU!) • `ETag` `304` • Sparse DTOs • Production Backend Pitfall & Guardrail (API Response Optimization (Compression, ETag & Projection)) • Hands-On TypeScript / Node.js Implementation (API Response Optimization (Compression, ETag & Projection)) • API Response Optimization (Compression, ETag & Projection) — Latency, Throughput & Interview Trade-Offs • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for API Response Optimization (Compression, ETag & Proje...

Purpose: Offloading `gzip`/`brotli` compression to Nginx in C so Node workers spend 100% of CPU on business logic!

What Topic Is Next → Next Topic 21.15: Pagination at Scale (Deep Offset Elimination)

21.15 Pagination at Scale (Deep Offset Elimination) [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Enforcing max offset caps (`page <= 100` for classic numbered pages) + switching feeds • Audit logs • Exports to `O(log N)` keyset cursors (`WHERE (created_at • Id) < ($1 • Production Backend Pitfall & Guardrail (Pagination at Scale (Deep Offset Elimination)) • Hands-On TypeScript / Node.js Implementation (Pagination at Scale (Deep Offset Elimination)) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Pagination at Scale (Deep Offset Elimination)

Purpose: Blocking scraper bots requesting `?page=95000` (`OFFSET 1900000`) from bringing down PostgreSQL!

What Topic Is Next → Next Topic 21.16: Multi-Tier Caching Strategies (L1 Memory + L2 Redis)

21.16 Multi-Tier Caching Strategies (L1 Memory + L2 Redis) [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Tiny bounded in-process `L1 LRU` (`max: 500` items • `TTL: 5s` for ultra-hot static config/categories -> `0.002ms` zero network!) backed by `L2 Redis` (`TTL: 5m` -> `0.4ms`) + Redis Pub/Sub L1 invalidation • Production Backend Pitfall & Guardrail (Multi-Tier Caching Strategies (L1 Memory + L2 Redis)) • Hands-On TypeScript / Node.js Implementation (Multi-Tier Caching Strategies (L1 Memory + L2 Redis)) • Multi-Tier Caching Strategies (L1 Memory + L2 Redis) — Latency, Throughput & Interview Trade-Offs • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Multi-Tier Caching Strategies (L1 Memory + L2 Redis) • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards

Purpose: Absorbing `20,000 RPS` flash-sale category-banner reads right inside bounded `5-second` L1 LRU memory!

What Topic Is Next → Next Topic 21.17: Load Testing Fundamentals (autocannon & k6)

21.17 Load Testing Fundamentals (autocannon & k6) [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Warm-up phase -> Ramp-up (`10 -> 50 -> 200 -> 500` VUs) -> Sustained plateau -> Spike test • Tracking `p95` • `p99` • Error rate `< 0.1%` • Production Backend Pitfall & Guardrail (Load Testing Fundamentals (autocannon & k6)) • Hands-On TypeScript / Node.js Implementation (Load Testing Fundamentals (autocannon & k6)) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Load Testing Fundamentals (autocannon & k6)

Purpose: Proving your Capstone handles `300` concurrent users with `0%` errors and `p95 < 80ms` before launch!

What Topic Is Next → Next Topic 21.18: Concurrency Handling and Load Shedding

21.18 Concurrency Handling and Load Shedding [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Bounding max in-flight requests per worker • Shedding excess load early with fast `503 Retry-After: 2` instead of accepting 10 • 000 requests that all time out after 30s (`Little's Law: L = λ * W`)! • Production Backend Pitfall & Guardrail (Concurrency Handling and Load Shedding) • Hands-On TypeScript / Node.js Implementation (Concurrency Handling and Load Shedding) • Concurrency Handling and Load Shedding — Latency, Throughput & Interview Trade-Offs • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Concurrency Handling

Purpose: Keeping `90%` of users fast during a `5x` traffic surge instead of `100%` of users timing out!

What Topic Is Next → Next Topic 21.19: Horizontal vs Vertical Scaling

21.19 Horizontal vs Vertical Scaling [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Vertical (`2 vCPU -> 8 vCPU` • Simplest up to hardware ceiling • Single point of failure) vs Horizontal (`N` stateless API containers across `2+` hosts behind Load Balancer — requires zero local state: sessions/locks/sockets in Redis • Files in MinIO!) • Production Backend Pitfall & Guardrail (Horizontal vs Vertical Scaling) • Hands-On TypeScript / Node.js Implementation (Horizontal vs Vertical Scaling) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Horizontal vs Vertical Scaling

Purpose: Verifying 100% statelessness so scaling `docker compose up --scale api=4` works with zero code changes!

What Topic Is Next → Next Topic 21.20: Graceful Shutdown Under Active Load

21.20 Graceful Shutdown Under Active Load [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Handling keep-alive idle connections vs active in-flight requests (`server.closeIdleConnections()` in Node 18.2+!) • Failing `/readyz` for `3–5s` FIRST so Nginx stops sending new requests BEFORE `server.close()`! • Production Backend Pitfall & Guardrail (Graceful Shutdown Under Active Load) • Hands-On TypeScript / Node.js Implementation (Graceful Shutdown Under Active Load) • Graceful Shutdown Under Active Load — Latency, Throughput & Interview Trade-Offs • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Graceful Shutdown Under Active Load • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards

Purpose: Achieving literally `0` failed requests (`0.00%` error rate) while running `autocannon -c 50` during a rolling deploy!

What Topic Is Next → Next Topic 21.21: Health Checks (/healthz Liveness vs /readyz Readiness)

21.21 Health Checks (/healthz Liveness vs /readyz Readiness) [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: CRITICAL distinction: **`GET /healthz` (Liveness)** checks ONLY that the Node event loop responds (`200 OK` • NEVER queries Postgres/Redis!) vs **`GET /readyz` (Readiness)** pings Postgres + Mongo + Redis (`503` stops traffic routing WITHOUT restarting containers when DB blips!)! • Production Backend Pitfall & Guardrail (Health Checks (/healthz Liveness vs /readyz Readiness)) • Hands-On TypeScript / Node.js Implementation (Health Checks (/healthz Liveness vs /readyz Readiness)) • Health Checks (/healthz Liveness vs /readyz Readiness) — Latency, Throughput & Interview Trade-Offs • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Health Checks (/healthz Liveness vs /readyz Readines... • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards

Purpose: Preventing all 10 API containers from entering a restart death-spiral just because PostgreSQL did a 5-second failover!

What Topic Is Next → Next Topic 21.22: High Availability Fundamentals

21.22 High Availability Fundamentals [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Eliminating Single Points of Failure (`SPOF`): redundant Load Balancers • `>= 2` API instances • PostgreSQL Primary + Streaming Standby Replica • MongoDB 3-node Replica Set • Production Backend Pitfall & Guardrail (High Availability Fundamentals) • Hands-On TypeScript / Node.js Implementation (High

Availability Fundamentals) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for High Availability Fundamentals

Purpose: Designing an HA e-commerce topology where any single process/node crash causes zero outage.

What Topic Is Next → Module 21 Capstone (Week 21 Lab Build: Autocannon Load Benchmark, Clinic.js Flamegraph & Worker-Thread Offloader) → Next Topic 22.1: Linux Server Filesystems and Production Directory Layout

Module 21 Hands-On Capstone Project (100 Marks): Week 21 Lab Build: Autocannon Load Benchmark, Clinic.js Flamegraph & Worker-Thread Offloader — Profile an intentionally slow/leaky Node API with `autocannon`, `--inspect` heap snapshots, and `perf_hooks.monitorEventLoopDelay()`, eliminate `O(N)` event-loop stalls via `worker_threads`...

MODULE 22 OF 24 • STAGE 6 • 25 LIVE + LAB HOURS • 26 TOPICS (208 SUBTOPICS) • 1 CAPSTONE PROJECT (100 MARKS)

Module 22: Docker, Linux, CI/CD and Production Deployment

Package, harden, automate, and deploy production Node.js backends on Linux servers and containers — Linux sysadmin/SSH/ufw/systemd, multi-stage non-root Dockerfiles, multi-container Docker Compose (Node.js, PostgreSQL, MongoDB, Redis, MinIO), Nginx reverse proxy, PM2, DNS/Let's Encrypt HTTPS, GitHub Actions CI/CD, server monitoring, backups, and live troubleshooting.

Objectives: ✓ Administer and harden Linux backend servers (`users`, `chmod`, `systemd`, `ufw` firewall, SSH keys, `journalctl` logs). • ✓ Author optimized multi-stage non-root `Dockerfile`s and health-checked `Docker Compose` stacks (`Node.js`, `PostgreSQL`, `MongoDB`, `Redis`, `MinIO`, `Nginx`). • ✓ Configure `Nginx` reverse proxying, `PM2` zero-downtime reloads, Domain/DNS records, and automated Let's Encrypt TLS/HTTPS. • ✓ Automate `GitHub Actions` CI/CD pipelines, secrets management, Prometheus/health monitoring, automated backups, and production incident recovery.

22.1 Linux Server Filesystems and Production Directory Layout [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Structuring `/srv/backend` or `/opt/app` • `/etc/nginx/sites-available` • `/var/log/nginx` • Swap sizing • Production Backend Pitfall & Guardrail (Linux Server Filesystems and Production Directory Layout) • Hands-On TypeScript / Node.js Implementation (Linux Server Filesystems and Production Directory Layout) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Linux Server Filesystems

Purpose: Catching `"No space left on device"` when `df -h` shows `40%` free disk because millions of tiny temp files exhausted `100%` of inodes (`df -i`)!

What Topic Is Next → Next Topic 22.2: Linux Users, Groups and Least-Privilege Permissions

22.2 Linux Users, Groups and Least-Privilege Permissions [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Creating dedicated unprivileged system user `useradd --system --create-home --shell /usr/sbin/nologin nodeapp` • Disabling root SSH login • Scoping `sudo` strictly where needed • Production Backend Pitfall & Guardrail (Linux Users, Groups and Least-Privilege Permissions) • Hands-On TypeScript / Node.js Implementation (Linux Users, Groups and Least-Privilege Permissions) • Linux Users, Groups and Least-Privilege Permissions — Latency, Throughput & Interview Trade-Offs • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Linux Users

Purpose: Running Node.js and PM2 strictly under `nodeapp` so an application bug cannot touch `/etc` or other services.

What Topic Is Next → Next Topic 22.3: Linux Process Management and systemd Service Units

22.3 Linux Process Management and systemd Service Units [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Writing `/etc/systemd/system/backend-api.service` (`Restart=on-failure` • `RestartSec=5` • `User=nodeapp` • `EnvironmentFile=/etc/backend/api.env` • Production Backend Pitfall & Guardrail (Linux Process Management and systemd Service Units) • Hands-On TypeScript / Node.js Implementation (Linux Process Management and systemd Service Units) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Linux Process Management

Purpose: Auto-starting your backend on Linux host reboot and supervising crashes natively via the kernel init system.

What Topic Is Next → Next Topic 22.4: Linux Networking Commands and Firewall Hardening (ufw)

22.4 Linux Networking Commands and Firewall Hardening (ufw) [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `ss -tulpn` • `ip addr` • `curl -I http://127.0.0.1:3000/healthz` • `nc -zv` • Production Backend Pitfall & Guardrail (Linux Networking Commands and Firewall Hardening (ufw)) • Hands-On TypeScript / Node.js Implementation (Linux Networking Commands and Firewall Hardening (ufw)) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Linux Networking Commands

Purpose: Blocking external internet access to internal ports `3000` (Express), `5432` (Postgres), `27017` (Mongo), and `6379` (Redis)!

What Topic Is Next → Next Topic 22.5: SSH Hardening and Secure Remote Administration

22.5 SSH Hardening and Secure Remote Administration [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Ed25519 SSH keys • `PasswordAuthentication no` • `PermitRootLogin no` • SSH bastion/port forwarding (`ssh -L 5433:127.0.0.1:5432 deploy@server` to inspect prod DB safely without opening port `5432`!) • Production Backend Pitfall & Guardrail (SSH Hardening and Secure Remote Administration) • Hands-On TypeScript / Node.js Implementation (SSH Hardening and Secure Remote Administration) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for SSH Hardening

Purpose: Tunneling `psql` or Compass securely over an encrypted SSH local port-forward (`ssh -L`) while keeping DB ports `100%` closed to the internet!

What Topic Is Next → Next Topic 22.6: Production Server Environment Variable Management

22.6 Production Server Environment Variable Management [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Storing `/etc/backend/api.env` (`chmod 0600` • `chown root:nodeapp` or `nodeapp:nodeapp`) outside the Git working directory so `git clean` or redeploys never touch or expose secrets • Production Backend Pitfall & Guardrail (Production Server Environment Variable Management) • Hands-On TypeScript / Node.js Implementation (Production Server Environment Variable Management) • Production Server Environment Variable Management — Latency, Throughput & Interview Trade-Offs • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month

Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Production Server Environment Variable Management • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards

Purpose: Keeping runtime secrets decoupled from release tarballs and Docker images.

What Topic Is Next → Next Topic 22.7: Server Logs, logrotate and Diagnostics

22.7 Server Logs, logrotate and Diagnostics [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `journalctl -u backend-api --since "10 min ago"` • Configuring `/etc/logrotate.d/` (`daily` • `rotate 14` • `compress` • Production Backend Pitfall & Guardrail (Server Logs, logrotate and Diagnostics) • Hands-On TypeScript / Node.js Implementation (Server Logs, logrotate and Diagnostics) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Server Logs

Purpose: Running `dmesg -T | grep -i "killed process"` to prove instantly whether Linux `OOMKiller` killed a crashed Node worker!

What Topic Is Next → Next Topic 22.8: Linux Server Tuning (File Descriptors & TCP Backlog)

22.8 Linux Server Tuning (File Descriptors & TCP Backlog) [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Raising `ulimit -n` (`nofile = 65535` • Default `1024` crashes Node with `EMFILE: too many open files` at ~900 concurrent WebSockets!) • `somaxconn` • NTP time sync (critical for JWT `exp` & webhook timestamps!) • Production Backend Pitfall & Guardrail (Linux Server Tuning (File Descriptors & TCP Backlog)) • Hands-On TypeScript / Node.js Implementation (Linux Server Tuning (File Descriptors & TCP Backlog)) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Linux Server Tuning (File Descriptors & TCP Backlog)

Purpose: Supporting `10,000+` concurrent Socket.IO connections on one Linux server by raising `LimitNOFILE=65535` and `chrony` NTP sync.

What Topic Is Next → Next Topic 22.9: Containerization Concepts vs Virtual Machines

22.9 Containerization Concepts vs Virtual Machines [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Linux kernel `namespaces` (isolated PID • Net • Mount • User views) + `cgroups` (CPU/RAM caps) sharing the host kernel (`~20ms` boot • Production Backend Pitfall & Guardrail (Containerization Concepts vs Virtual Machines) • Hands-On TypeScript / Node.js Implementation (Containerization Concepts vs Virtual Machines) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Containerization Concepts vs Virtual Machines

Purpose: Eliminating `"works on my Windows laptop, breaks on Ubuntu prod"` forever via immutable container artifacts.

What Topic Is Next → Next Topic 22.10: Docker Engine Architecture

22.10 Docker Engine Architecture [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Docker CLI -> `dockerd` Daemon -> `containerd` -> `runc` OCI runtime • Layered copy-on-write UnionFS image layers • OCI registries (`GHCR` / Docker Hub) • Production Backend Pitfall & Guardrail (Docker Engine Architecture) • Hands-On TypeScript / Node.js Implementation (Docker Engine Architecture) • Docker Engine Architecture — Latency, Throughput & Interview Trade-Offs • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Docker Engine Architecture

Purpose: Ordering `Dockerfile` instructions from least-frequently changing (`COPY package*.json` -> `RUN npm ci`) to most-frequently changing (`COPY src ./src`) for `10x` faster c...

What Topic Is Next → Next Topic 22.11: Docker Images and Containers Lifecycle

22.11 Docker Images and Containers Lifecycle [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Immutable tagged Images (`ghcr.io/org/api:git-sha` • NEVER deploy mutable `:latest` tags in production!) vs ephemeral Containers (`docker run --rm` • `docker ps` • `docker logs -f` • Production Backend Pitfall & Guardrail (Docker Images and Containers Lifecycle) • Hands-On TypeScript / Node.js Implementation (Docker Images and Containers Lifecycle) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Docker Images

Purpose: Tagging every CI build with its immutable Git commit SHA (`api:a1b2c3d`) so rolling back to yesterday's release takes 5 seconds!

What Topic Is Next → Next Topic 22.12: Authoring Production Dockerfiles and .dockerignore

22.12 Authoring Production Dockerfiles and .dockerignore [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `FROM node:22-bookworm-slim` (avoid `alpine` `musl` libc surprises with `prisma`/`argon2`/`sharp` native binaries unless tested — `bookworm-slim` uses standard `glibc`!) • `.dockerignore` (`node_modules` • `.git` • `.env*` • Production Backend Pitfall & Guardrail (Authoring Production Dockerfiles and .dockerignore) • Hands-On TypeScript / Node.js Implementation (Authoring Production Dockerfiles and .dockerignore) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Authoring Production Dockerfiles

Purpose: Using **exec form** `CMD ["node", "dist/server.js"]` instead of `CMD ["npm", "start"]` (`npm` swallows `SIGTERM` and causes a 10-second `SIGKILL` timeout on every deploy!...

What Topic Is Next → Next Topic 22.13: Building and Running Hardened Containers

22.13 Building and Running Hardened Containers [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Running as built-in non-root `USER node` (UID `1000`) • `--read-only` root filesystem (`--tmpfs /tmp`) • `--cap-drop=ALL` • Resource limits (`mem_limit: 512m` • Production Backend Pitfall & Guardrail (Building and Running Hardened Containers) • Hands-On TypeScript / Node.js Implementation (Building and Running Hardened Containers) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Building

Purpose: Ensuring a compromised dependency cannot write binaries to `/app` or consume all host RAM.

What Topic Is Next → Next Topic 22.14: Docker Volumes and Persistent Database Storage

22.14 Docker Volumes and Persistent Database Storage [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Named volumes (`pg_data:/var/lib/postgresql/data` • `mongo_data:/data/db` • `redis_data:/data` • `minio_data:/data`) vs bind mounts (dev only!) • Production Backend Pitfall & Guardrail (Docker Volumes and Persistent Database Storage) • Hands-On TypeScript / Node.js Implementation (Docker Volumes

and Persistent Database Storage) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Docker Volumes

Purpose: Persisting PostgreSQL, MongoDB, Redis AOF, and MinIO objects safely across container image upgrades.

What Topic Is Next → Next Topic 22.15: Container Networking and Internal DNS Isolation

22.15 Container Networking and Internal DNS Isolation [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Custom isolated Docker networks: `edge_net` (`nginx <-> api`) vs `backend_net` (`api + worker <-> postgres + mongo + redis + minio` with `internal: true` so DB containers have ZERO external exposure!) • Production Backend Pitfall & Guardrail (Container Networking and Internal DNS Isolation) • Hands-On TypeScript / Node.js Implementation (Container Networking and Internal DNS Isolation) • Container Networking and Internal DNS Isolation — Latency, Throughput & Interview Trade-Offs • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Container Networking • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards • Diagnostics, Profiling & Telemetry: structured telemetry, profiler traces & CI checks

Purpose: Placing `postgres`, `mongo`, and `redis` on `backend_net` so `nginx` cannot even route TCP packets directly to databases.

What Topic Is Next → Next Topic 22.16: Docker Compose Orchestration

22.16 Docker Compose Orchestration [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Modern `compose.yaml` (`healthcheck` • `depends_on: { postgres: { condition: service_healthy } }` • `restart: unless-stopped` • `logging: { driver: "json-file" • Production Backend Pitfall & Guardrail (Docker Compose Orchestration) • Hands-On TypeScript / Node.js Implementation (Docker Compose Orchestration) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Docker Compose Orchestration

Purpose: Waiting for `pg_isready` (`service_healthy`) BEFORE starting the `migrate` one-shot container and `api` service — eliminates boot-time `"Connection refused"` crashes!

What Topic Is Next → Next Topic 22.17: Multi-Container Backend Stack (Node + Postgres + Mongo + Redis + MinIO)

22.17 Multi-Container Backend Stack (Node + Postgres + Mongo + Redis + MinIO) [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Assembling the complete polyglot backend stack (`nginx` • `migrate` • `api` • `worker` • Production Backend Pitfall & Guardrail (Multi-Container Backend Stack (Node + Postgres + Mongo + Redis + MinIO)) • Hands-On TypeScript / Node.js Implementation (Multi-Container Backend Stack (Node + Postgres + Mongo + Redis + MinIO)) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Multi-Container Backend Stack (Node + Postgres + Mon...

Purpose: Running the exact Final Capstone production topology locally and on any Linux server.

What Topic Is Next → Next Topic 22.18: Docker Image Optimization and Multi-Stage Builds

22.18 Docker Image Optimization and Multi-Stage Builds [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: 3-stage `Dockerfile`: Stage 1 `deps` (`npm ci`) -> Stage 2 `builder` (`npx prisma generate && npm run build` + `npm prune --omit=dev`) -> Stage 3 `runtime` (copies ONLY production `node_modules` • Generated Prisma client & `dist/` — zero TypeScript compiler • Zero `src/` • Zero test files!) • Production Backend Pitfall & Guardrail (Docker Image Optimization and Multi-Stage Builds) • Hands-On TypeScript / Node.js Implementation (Docker Image Optimization and Multi-Stage Builds) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Docker Image Optimization

Purpose: Shrinking your Node API image from `1.15 GB` (with devDependencies & build tools) down to `~165 MB` with minimal attack surface!

What Topic Is Next → Next Topic 22.19: Nginx Reverse Proxy Configuration

22.19 Nginx Reverse Proxy Configuration [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `upstream backend_api { server 127.0.0.1:3001 • Server 127.0.0.1:3002 • Keepalive 32 • `X-Real-IP` / `X-Forwarded-For` / `X-Forwarded-Proto` • Production Backend Pitfall & Guardrail (Nginx Reverse Proxy Configuration) • Hands-On TypeScript / Node.js Implementation (Nginx Reverse Proxy Configuration) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Nginx Reverse Proxy Configuration

Purpose: Terminating HTTPS, compressing JSON, buffering slow mobile uploads, and load-balancing across Node workers cleanly.

What Topic Is Next → Next Topic 22.20: PM2 Process Manager for Bare-Metal Linux Deployments

22.20 PM2 Process Manager for Bare-Metal Linux Deployments [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `ecosystem.config.cjs` (`instances: "max"` • `exec_mode: "cluster"` • `wait_ready: true` • `listen_timeout: 8000` • Production Backend Pitfall & Guardrail (PM2 Process Manager for Bare-Metal Linux Deployments) • Hands-On TypeScript / Node.js Implementation (PM2 Process Manager for Bare-Metal Linux Deployments) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for PM2 Process Manager for Bare-Metal Linux Deployments

Purpose: Deploying zero-downtime rolling reloads (`pm2 reload ecosystem.config.cjs --update-env` — NOT `pm2 restart` which drops active connections!) on a Linux VPS.

What Topic Is Next → Next Topic 22.21: Domain and DNS Configuration

22.21 Domain and DNS Configuration [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Configuring `A` (IPv4) & `AAAA` (IPv6) records for `api.yourdomain.com` • `CAA` records restricting certificate issuance to Let's Encrypt • Lowering DNS `TTL` (`300s`) 24 hours before IP migrations • Production Backend Pitfall & Guardrail (Domain and DNS Configuration) • Hands-On TypeScript / Node.js Implementation (Domain and DNS Configuration) • Domain and DNS Configuration — Latency, Throughput & Interview Trade-Offs • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Domain

Purpose: Routing `api.example.com` and `ws.example.com` cleanly to your Nginx reverse proxy.

What Topic Is Next → Next Topic 22.22: SSL Certificates and HTTPS (Let's Encrypt & Certbot)

22.22 SSL Certificates and HTTPS (Let's Encrypt & Certbot) [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Automating free trusted TLS 1.2/1.3 certificates via `certbot --nginx -d api.example.com` • HTTP `80 -> 301 HTTPS 443` redirect • `HSTS` • OCSP stapling • Production Backend Pitfall & Guardrail (SSL Certificates and HTTPS (Let's Encrypt & Certbot)) • Hands-On TypeScript / Node.js Implementation (SSL Certificates and HTTPS (Let's Encrypt & Certbot)) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for SSL Certificates

Purpose: Achieving an `A+` SSL Labs TLS grade with automated zero-touch 90-day certificate renewals.

What Topic Is Next → Next Topic 22.23: GitHub Actions CI/CD Workflows

22.23 GitHub Actions CI/CD Workflows [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Authoring `.github/workflows/ci-cd.yml` with `npm ci` caching • `postgres:16` + `mongo:7` + `redis:7` GitHub Actions `services:` containers • Running `lint -> typecheck -> prisma migrate deploy -> test -> docker build`! • Production Backend Pitfall & Guardrail (GitHub Actions CI/CD Workflows) • Hands-On TypeScript / Node.js Implementation (GitHub Actions CI/CD Workflows) • GitHub Actions CI/CD Workflows — Latency, Throughput & Interview Trade-Offs • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for GitHub Actions CI/CD Workflows

Purpose: Running real PostgreSQL, MongoDB, and Redis integration tests automatically on every GitHub Pull Request.

What Topic Is Next → Next Topic 22.24: Automated Deployment Workflows & Safe Rollouts

22.24 Automated Deployment Workflows & Safe Rollouts [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Push to `main` -> CI passes -> Build & push GHCR image (`:git-sha`) -> Deploy over SSH (`docker compose pull && docker compose run --rm migrate && docker compose up -d --remove-orphans`) -> Post-deploy `/readyz` smoke check -> Auto-rollback to previous SHA if `/readyz` fails! • Production Backend Pitfall & Guardrail (Automated Deployment Workflows & Safe Rollouts) • Hands-On TypeScript / Node.js Implementation (Automated Deployment Workflows & Safe Rollouts) • Automated Deployment Workflows & Safe Rollouts — Latency, Throughput & Interview Trade-Offs • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Automated Deployment Workflows • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards • Diagnostics, Profiling & Telemetry: structured telemetry, profiler traces & CI checks

Purpose: Shipping 5 safe deployments a day with automated health verification and instant SHA rollback.

What Topic Is Next → Next Topic 22.25: Production Observability & Server Health Monitoring

22.25 Production Observability & Server Health Monitoring [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Exposing Prometheus `/metrics` (`prom-client`: HTTP request duration histograms by route/status • Event-loop lag • Active handles • Heap bytes • Production Backend Pitfall & Guardrail (Production Observability & Server Health Monitoring) • Hands-On TypeScript / Node.js Implementation (Production Observability & Server Health Monitoring) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Production Observability

Purpose: Getting a Telegram/Slack/Pager alert within 30 seconds if `5xx` rate exceeds `1%`, disk passes `80%`, or `/readyz` fails.

What Topic Is Next → Next Topic 22.26: Backup, Disaster Recovery and Production Troubleshooting

22.26 Backup, Disaster Recovery and Production Troubleshooting [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Automated encrypted off-host `pg_dump` + `mongodump` schedules with retention pruning • Monthly restore drills • The 6-step Incident Runbook (`Check /readyz -> Check recent deploy diff -> Check Pino 5xx logs -> Check DB pg_stat_activity locks -> Check disk/RAM/OOM -> Rollback first • Root-cause second!`) • Production Backend Pitfall & Guardrail (Backup, Disaster Recovery and Production Troubleshooting) • Hands-On TypeScript / Node.js Implementation (Backup, Disaster Recovery and Production Troubleshooting) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Backup

Purpose: Restoring service in `< 3 minutes` during a live outage by rolling back cleanly before investigating root causes.

What Topic Is Next → Module 22 Capstone (Week 22 Lab Build: 3-Stage Non-Root Docker Compose + Nginx + PM2 + GitHub Actions CI/CD Stack) → Next Topic 23.1: Interview Mastery: Closures, Prototypes, Promises & Event Loop

Module 22 Hands-On Capstone Project (100 Marks): Week 22 Lab Build: 3-Stage Non-Root Docker Compose + Nginx + PM2 + GitHub Actions CI/CD Stack — Containerize and deploy a polyglot Node.js + PostgreSQL + MongoDB + Redis + MinIO + Nginx stack using a `~165 MB` 3-stage non-root `Dockerfile`, dual-network `compose.yaml` with `service_hea...

MODULE 23 OF 24 • STAGE 6 • 25 LIVE + LAB HOURS • 18 TOPICS (144 SUBTOPICS) • 1 CAPSTONE PROJECT (100 MARKS)

Module 23: Backend System Design and Technical Interviews

Crack Node.js & Backend Software Engineer interviews across all 4 rounds — deep JS/TS & Node/Express/DB/Security oral vivas, 10 classic Backend System Design whiteboard problems, and 10 timed Node.js/TypeScript Backend Machine-Coding exercises.

Objectives: ✓ Answer rapid-fire JS/TS, V8/libuv/Streams, MongoDB/PostgreSQL/Prisma, and Auth/OWASP oral interview questions with production depth. • ✓ Lead 45-minute Backend System Design rounds covering capacity math, schema design, caching, queues, sharding, and failure modes across all 10 classic systems. • ✓ Complete timed 60–75 minute Node.js + TypeScript Backend Machine-Coding challenges cleanly under interview pressure. • ✓ Clear structured mock technical interviews ahead of Week 24 Capstone Defense.

23.1 Interview Mastery: Closures, Prototypes, Promises & Event Loop [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Lexical closures & memory retainers • `this` & prototype delegation • Promise combinators (`all` vs `allSettled` vs `any`) • `return await` inside `try/catch` • Production Backend Pitfall & Guardrail (Interview Mastery: Closures, Prototypes, Promises & Event Loop) • Hands-On TypeScript / Node.js Implementation (Interview Mastery: Closures, Prototypes, Promises & Event Loop) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Interview Mastery

Purpose: Clearing Round 1 JavaScript internals live-debugging and event-loop trace interviews effortlessly.

What Topic Is Next → Next Topic 23.2: Interview Mastery: TypeScript Interfaces, Generics & Strict Narrowing

23.2 Interview Mastery: TypeScript Interfaces, Generics & Strict Narrowing [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `unknown` vs `any` vs `never` • Discriminated unions • Custom type predicates (`x is T`) • `keyof`/`typeof`/`infer` • Production Backend Pitfall & Guardrail (Interview Mastery: TypeScript Interfaces, Generics & Strict Narrowing) • Hands-On TypeScript / Node.js Implementation (Interview Mastery: TypeScript Interfaces, Generics & Strict Narrowing) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Interview Mastery

Purpose: Writing zero-`any` generic helpers live during TypeScript pair-programming interviews.

What Topic Is Next → Next Topic 23.3: Interview Mastery: Node.js V8, libuv, Streams, Workers & Memory

23.3 Interview Mastery: Node.js V8, libuv, Streams, Workers & Memory [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: V8 JIT & GC generations • `libuv` epoll vs 4-thread pool (`UV_THREADPOOL_SIZE`) • `Buffer` & Stream backpressure (`pipeline()`) • `worker_threads` vs `cluster` • Production Backend Pitfall & Guardrail (Interview Mastery: Node.js V8, libuv, Streams, Workers & Memory) • Hands-On TypeScript / Node.js Implementation (Interview Mastery: Node.js V8, libuv, Streams, Workers & Memory) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Interview Mastery

Purpose: Explaining crisply how a single Node process handles 10k sockets AND when to offload CPU work to `worker_threads`.

What Topic Is Next → Next Topic 23.4: Interview Mastery: Express Middleware, Routing & Controller Architecture

23.4 Interview Mastery: Express Middleware, Routing & Controller Architecture [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Middleware execution stack • 4-arg `(err • Req • Res • Production Backend Pitfall & Guardrail (Interview Mastery: Express Middleware, Routing & Controller Architecture) • Hands-On TypeScript / Node.js Implementation (Interview Mastery: Express Middleware, Routing & Controller Architecture) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Interview Mastery

Purpose: Walking interviewers cleanly through request lifecycle from TCP accept to Pino correlation log & response.

What Topic Is Next → Next Topic 23.5: Interview Mastery: MongoDB Aggregation, PostgreSQL SQL, Indexes & ACID

23.5 Interview Mastery: MongoDB Aggregation, PostgreSQL SQL, Indexes & ACID [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: MongoDB `$match`/`$group`/`$lookup` + `ESR` compound indexes + `.lean()` • PostgreSQL Joins • `NOT EXISTS` • Window Functions (`DENSE_RANK` • Production Backend Pitfall & Guardrail (Interview Mastery: MongoDB Aggregation, PostgreSQL SQL, Indexes & ACID) • Hands-On TypeScript / Node.js Implementation (Interview Mastery: MongoDB Aggregation, PostgreSQL SQL, Indexes & ACID) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Interview Mastery

Purpose: Solving live SQL Window-Function + MongoDB Aggregation + concurrency-race questions back-to-back.

What Topic Is Next → Next Topic 23.6: Interview Mastery: JWT Rotation, OAuth2, RBAC & OWASP API Security

23.6 Interview Mastery: JWT Rotation, OAuth2, RBAC & OWASP API Security [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Argon2id vs bcrypt • `15m` Access + `7d` rotated `HttpOnly` Refresh family reuse detection • OAuth2 PKCE + OIDC `email_verified` • RBAC + BOLA query scoping • Production Backend Pitfall & Guardrail (Interview Mastery: JWT Rotation, OAuth2, RBAC & OWASP API Security) • Hands-On TypeScript / Node.js Implementation (Interview Mastery: JWT Rotation, OAuth2, RBAC & OWASP API Security) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Interview Mastery

Purpose: Defending every security design decision of your Auth & E-Commerce portfolio projects.

What Topic Is Next → Next Topic 23.7: System Design #1: High-Scale URL Shortener (10k Reads/sec)

23.7 System Design #1: High-Scale URL Shortener (10k Reads/sec) [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Capacity math (`100:1` read:write) • Base62 (`[0-9a-zA-Z]` • `7` chars = `62^7 = 3.5 trillion` URLs!) over Snowflake/Range-Allocated IDs (zero collision!) • `302` (analytics) vs `301` (CDN cached) • Production Backend Pitfall & Guardrail (System Design #1: High-Scale URL Shortener (10k Reads/sec)) • Hands-On TypeScript / Node.js Implementation (System Design #1: High-Scale URL Shortener (10k Reads/sec)) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for System Design #1

Purpose: Leading a crisp 40-minute URL Shortener design from QPS math to DB index & cache eviction.

What Topic Is Next → Next Topic 23.8: System Design #2: Centralized Authentication & SSO Service

23.8 System Design #2: Centralized Authentication & SSO Service [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Multi-app OIDC/OAuth2 issuer • `Ed25519`/`RS256` key rotation via cached `/.well-known/jwks.json` • Redis rate-limit + `jti` revocation broadcast • Refresh-token family table • Production Backend Pitfall & Guardrail (System Design #2: Centralized Authentication & SSO Service) • Hands-On TypeScript / Node.js Implementation (System Design #2: Centralized Authentication & SSO Service) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for System Design #2

Purpose: Designing an Auth microservice where 20 downstream APIs verify tokens locally in `0.05ms` with zero network hop!

What Topic Is Next → Next Topic 23.9: System Design #3: Flash-Sale E-Commerce & Order Management System

23.9 System Design #3: Flash-Sale E-Commerce & Order Management System [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Handling `50 • 000` users clicking Buy on `500` iPhones at `12:00:00`: CDN/Redis catalog cache -> Lua atomic stock pre-decrement gate (blocks 49 • 500 oversells in `0.2ms` before Postgres!) -> Order Queue -> Saga Payment + `15m` unpaid expiry release! • Production Backend Pitfall & Guardrail (System Design #3: Flash-Sale E-Commerce & Order Management System) • Hands-On TypeScript / Node.js Implementation (System Design #3: Flash-Sale E-Commerce & Order Management System) • System Design #3: Flash-Sale E-Commerce & Order Management System — Latency, Throughput & Interview Trade-Offs • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for System Design #3

Purpose: Explaining how to survive `100x` flash-sale checkout spikes without melting PostgreSQL row locks.

What Topic Is Next → Next Topic 23.10: System Design #4: Multi-Channel Notification Service (Email, SMS, Push, In-App)

23.10 System Design #4: Multi-Channel Notification Service (Email, SMS, Push, In-App) [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Unified `POST /notify` API + Idempotency Key -> User Preference/Opt-Out filter -> Channel-isolated Priority Queues (`otp-sms` vs `promo-email`) -> Provider Adapters with Circuit Breaker failover (`Vendor A -> Vendor B`) + Delivery status webhook tracker • Production Backend Pitfall & Guardrail (System Design #4: Multi-Channel Notification Service (Email, SMS, Push, In-App)) • Hands-On TypeScript / Node.js Implementation (System Design #4: Multi-Channel Notification Service (Email, SMS, Push, In-App)) • System Design #4: Multi-Channel Notification Service (Email, SMS, Push, In-App) — Latency, Throughput & Interview Trade-Offs • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for System Design #4 • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards • Diagnostics, Profiling & Telemetry: structured telemetry, profiler traces & CI checks

Purpose: Guaranteeing P0 OTP SMS delivery in `< 3s` even when Marketing enqueues `2 million` promotional emails!

What Topic Is Next → Next Topic 23.11: System Design #5: Real-Time Chat Backend (1-on-1 & Group)

23.11 System Design #5: Real-Time Chat Backend (1-on-1 & Group) [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Stateful WebSocket Gateway fleet + Redis Presence/Routing Registry (`userId -> gatewayNodeId`) + Message Service + Partitioned/Time-Sorted Message Store (`(conversation_id • Message_id DESC)`) + Offline Push Notification fallback • Production Backend Pitfall & Guardrail (System Design #5: Real-Time Chat Backend (1-on-1 & Group)) • Hands-On TypeScript / Node.js Implementation (System Design #5: Real-Time Chat Backend (1-on-1 & Group)) • System Design #5: Real-Time Chat Backend (1-on-1 & Group) — Latency, Throughput & Interview Trade-Offs • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for System Design #5 • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards

Purpose: Designing WhatsApp/Slack-style delivery (`Sent ✓ -> Delivered ✓✓ -> Read`) and bounded fan-out (`max 500` group size).

What Topic Is Next → Next Topic 23.12: System Design #6: Object File Storage & Media Processing Pipeline

23.12 System Design #6: Object File Storage & Media Processing Pipeline [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Metadata DB (`files` table: `id` • `ownerId` • `sha256` • `status` • Production Backend Pitfall & Guardrail (System Design #6: Object File Storage & Media Processing Pipeline) • Hands-On TypeScript / Node.js Implementation (System Design #6: Object File Storage & Media Processing Pipeline) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for System Design #6

Purpose: Uploading `2 GB` videos resumably in `5 MB` chunks without routing bytes through Express.

What Topic Is Next → Next Topic 23.13: System Design #7: Distributed Job Scheduling System

23.13 System Design #7: Distributed Job Scheduling System [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Durable `scheduled_jobs` table (`next_run_at` indexed) -> Partitioned Scheduler Pollers using `SELECT ... WHERE next_run_at <= NOW() • Status = 'READY' FOR UPDATE SKIP LOCKED LIMIT 100` -> Worker Queue -> Heartbeat lease renewal + Dead-worker recovery! • Production Backend Pitfall & Guardrail (System Design #7: Distributed Job Scheduling System) • Hands-On TypeScript / Node.js Implementation (System Design #7: Distributed Job Scheduling System) • System Design #7: Distributed Job Scheduling System — Latency, Throughput & Interview Trade-Offs • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for System Design #7 • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards

Purpose: Executing millions of user-configured scheduled webhooks/reports reliably across N worker nodes with zero duplicate claims!

What Topic Is Next → Next Topic 23.14: System Design #8: Distributed Rate Limiter & API Gateway

23.14 System Design #8: Distributed Rate Limiter & API Gateway [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: API Gateway placement • Token Bucket vs Sliding-Window Counter (`curr_min * 1 + prev_min * overlap_ratio` — `O(1)` memory using just 2 integers!) • Redis Lua atomicity • Local 100ms batch-sync for ultra-high RPS • Production Backend Pitfall & Guardrail (System Design #8: Distributed Rate Limiter & API Gateway) • Hands-On TypeScript / Node.js Implementation (System Design #8: Distributed Rate Limiter & API Gateway) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for System Design #8

Purpose: Designing a `< 1ms` gateway rate limiter handling `50,000 RPS` with standard `RateLimit-*` & `Retry-After` headers.

What Topic Is Next → Next Topic 23.15: System Design #9: Centralized Log Ingestion & Distributed Caching

23.15 System Design #9: Centralized Log Ingestion & Distributed Caching [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Non-blocking stdout JSON -> Shipper agent -> Kafka buffer (absorbs `10x` incident log storms!) -> Bulk indexer -> Hot (`7d` searchable) vs Cold (`90d` compressed archive) tiers • PLUS Distributed Cache consistent hashing ring (`virtual nodes`) minimizing key remapping (`1/N`) when cache nodes join/leave! • Production Backend Pitfall & Guardrail (System Design #9: Centralized Log Ingestion & Distributed Caching) • Hands-On TypeScript / Node.js Implementation (System Design #9: Centralized Log Ingestion & Distributed Caching) • System Design #9: Centralized Log Ingestion & Distributed Caching — Latency, Throughput & Interview Trade-Offs • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for System Design #9 • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards

Purpose: Explaining why `hash(key) % N` invalidates ~`80%` of keys when `N` changes from `4` to `5`, whereas **Consistent Hashing** remaps only `~1/5` (`20%`)!

What Topic Is Next → Next Topic 23.16: Machine Coding #1–#3: Auth API, Employee CRUD & Dynamic RBAC Engine

23.16 Machine Coding #1–#3: Auth API, Employee CRUD & Dynamic RBAC Engine [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Timed 60-min builds: (1) User Registration/Login + Rotated Refresh JWT API • (2) Validated Employee CRUD + Paginated Search/Filter/Sort API • (3) Role + Granular Permission + Resource-Ownership Middleware Engine with Supertest suites • Production Backend Pitfall & Guardrail (Machine Coding #1–#3: Auth API, Employee CRUD & Dynamic RBAC Engine) • Hands-On TypeScript / Node.js Implementation (Machine Coding #1–#3: Auth API, Employee CRUD & Dynamic RBAC Engine) • Machine Coding #1–#3: Auth API, Employee CRUD & Dynamic RBAC Engine — Latency, Throughput & Interview Trade-Offs • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Machine Coding #1–#3

Purpose: Writing clean, tested layered TypeScript + Express APIs from a blank folder in under 60 minutes.

What Topic Is Next → Next Topic 23.17: Machine Coding #4–#6: Paginated Search, Safe File Upload & Email Notification Worker

23.17 Machine Coding #4–#6: Paginated Search, Safe File Upload & Email Notification Worker [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Timed 60-min builds: (4) Keyset Cursor + Faceted Search API • (5) Multer + Magic-Byte + Checksum File Upload & Presigned URL Service • (6) BullMQ Templated Email Notification Service with Exponential Backoff & Idempotency • Production Backend Pitfall & Guardrail (Machine Coding #4–#6: Paginated Search, Safe File Upload & Email Notification Worker) • Hands-On TypeScript / Node.js Implementation (Machine Coding #4–#6: Paginated Search, Safe File Upload & Email Notification Worker) • Machine Coding #4–#6: Paginated Search, Safe File Upload & Email Notification Worker — Latency, Throughput & Interview Trade-Offs • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Machine Coding #4–#6

Purpose: Demonstrating production async & I/O engineering fluency in live coding rounds.

What Topic Is Next → Next Topic 23.18: Machine Coding #7–#10: Redis Cache Layer, Job Processor, Signed Webhook & Order Workflow

23.18 Machine Coding #7–#10: Redis Cache Layer, Job Processor, Signed Webhook & Order Workflow [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Timed 75-min builds: (7) Cache-Aside + Single-Flight Stampede Guard + Invalidation Layer • (8) Priority Retry + DLQ Background Job Processor • (9) HMAC-SHA256 + Timestamp + Idempotent Webhook Processor • (10) Transactional Order Checkout -> Stock Reserve -> Payment Workflow! • Production Backend Pitfall & Guardrail (Machine Coding #7–#10: Redis Cache Layer, Job Processor, Signed Webhook & Order Workflow) • Hands-On TypeScript / Node.js Implementation (Machine Coding #7–#10: Redis Cache Layer, Job Processor, Signed Webhook & Order Workflow) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Machine Coding #7–#10

Purpose: Completing all 10 Backend Machine-Coding exercises directly preparing for the Week 24 Final Enterprise Capstone!

What Topic Is Next → Module 23 Capstone (Week 23 Lab Build: 10 System Design Whiteboards + Timed Backend Machine-Coding Portfolio) → Next Topic 24.1: Capstone Blueprint & Modular Bounded-Context Setup

Module 23 Hands-On Capstone Project (100 Marks): Week 23 Lab Build: 10 System Design Whiteboards + Timed Backend Machine-Coding Portfolio — Deliver architecture specs & capacity calculations for all 10 Backend System Design problems (URL Shortener, Auth/SSO, Flash-Sale Order Engine, Notification Hub, Chat, Object Storage, Job Sc...

MODULE 24 OF 24 • STAGE 6 • 25 LIVE + LAB HOURS • 16 TOPICS (128 SUBTOPICS) • 1 CAPSTONE PROJECT (100 MARKS)

Module 24: Final Backend Capstone and Placement Readiness

Architect, build, test, document, containerize, and defend the Mandatory Final Capstone — Enterprise E-Commerce and Order Management Backend (TypeScript, Node.js, Express.js, PostgreSQL + Prisma, MongoDB + Mongoose, JWT/OAuth/RBAC, Redis, BullMQ, RabbitMQ/Kafka, Socket.IO, Jest/Supertest, Swagger & Docker Compose + Nginx) — and complete final placement readiness.

Objectives: ✓ Deliver all 24 functional requirements of the **Enterprise E-Commerce and Order Management Backend** cleanly in strict TypeScript. • ✓ Integrate PostgreSQL (`Prisma` ACID orders/inventory/coupons), MongoDB (`Mongoose` catalog), Redis (caching & rate limits), BullMQ (jobs), RabbitMQ/Kafka (events), MinIO (files), and Socket.IO (real-time order updates). • ✓ Ship complete OpenAPI/Swagger docs, Postman collection, `Jest + Supertest` suite, ER & Architecture diagrams, and multi-container `Docker Compose + Nginx` deployment. • ✓ Pass Mock Technical Interview #3 and defend your Final Capstone live in a rigorous technical viva.

24.1 Capstone Blueprint & Modular Bounded-Context Setup [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Structuring `src/modules/{auth • Users • Catalog • Inventory • Production Backend Pitfall & Guardrail (Capstone Blueprint & Modular Bounded-Context Setup) • Hands-On TypeScript / Node.js Implementation (Capstone Blueprint & Modular Bounded-Context Setup) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Capstone Blueprint

Purpose: Establishing clean module boundaries before writing business endpoints.

What Topic Is Next → Next Topic 24.2: Capstone Auth, Rotated Refresh Tokens & Granular RBAC

24.2 Capstone Auth, Rotated Refresh Tokens & Granular RBAC [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Implementing registration • Email verification • Password reset • `Argon2id` • Production Backend Pitfall & Guardrail (Capstone Auth, Rotated Refresh Tokens & Granular RBAC) • Hands-On TypeScript / Node.js Implementation (Capstone Auth, Rotated Refresh Tokens & Granular RBAC) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Capstone Auth

Purpose: Securing customer profiles, seller product management, and admin coupon/order controls.

What Topic Is Next → Next Topic 24.3: Capstone Polyglot Data Layer (MongoDB Catalog + PostgreSQL Ledger)

24.3 Capstone Polyglot Data Layer (MongoDB Catalog + PostgreSQL Ledger) [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Modelling polymorphic Product & Category specs/attributes in **MongoDB (`Mongoose`)** with `ESR` + text indexes • Alongside **PostgreSQL (`Prisma`)** Users • Inventory Stock • Coupons • Production Backend Pitfall & Guardrail (Capstone Polyglot Data Layer (MongoDB Catalog + PostgreSQL Ledger)) • Hands-On TypeScript / Node.js Implementation (Capstone Polyglot Data Layer (MongoDB Catalog + PostgreSQL Ledger)) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Capstone Polyglot Data Layer (MongoDB Catalog + Post...

Purpose: Demonstrating production polyglot persistence where each store handles its ideal workload.

What Topic Is Next → Next Topic 24.4: Capstone Product Search, Faceted Filters, Pagination & Redis Caching

24.4 Capstone Product Search, Faceted Filters, Pagination & Redis Caching [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Building `GET /api/v1/products` with keyword search • Category/price/rating/in-stock filters • Whitelisted sorting • Cursor/offset pagination • Production Backend Pitfall & Guardrail (Capstone Product Search, Faceted Filters, Pagination & Redis Caching) • Hands-On TypeScript / Node.js Implementation (Capstone Product Search, Faceted Filters, Pagination & Redis Caching) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Capstone Product Search

Purpose: Delivering `< 15ms` p95 product discovery throughput.

What Topic Is Next → Next Topic 24.5: Capstone Shopping Cart & Coupon/Discount Engine

24.5 Capstone Shopping Cart & Coupon/Discount Engine [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Redis Hash / DB-synced Shopping Cart APIs (`add` • `updateQty` • `remove` • `clear` • Production Backend Pitfall & Guardrail (Capstone Shopping Cart & Coupon/Discount Engine) • Hands-On TypeScript / Node.js Implementation (Capstone Shopping Cart & Coupon/Discount Engine) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Capstone Shopping Cart

Purpose: Calculating exact itemized subtotal, coupon discount, 18% GST, and payable net total server-side.

What Topic Is Next → Next Topic 24.6: Capstone Race-Free Inventory Reservation & Order Creation

24.6 Capstone Race-Free Inventory Reservation & Order Creation [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Sorted-SKU `SELECT ... FOR UPDATE` / conditional atomic `qtyOnHand >= qty` stock reservation • Snapshotting `title` + `unitPricePaise` on `OrderItem` • `Idempotency-Key` protection on `POST /api/v1/orders` • `15m` BullMQ unpaid reservation expiry! • Production Backend Pitfall & Guardrail (Capstone Race-Free Inventory Reservation & Order Creation) • Hands-On TypeScript / Node.js Implementation (Capstone Race-Free Inventory Reservation & Order Creation) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Capstone Race-Free Inventory Reservation

Purpose: Guaranteeing zero overselling even when 20 buyers click Checkout on the last unit simultaneously!

What Topic Is Next → Next Topic 24.7: Capstone Sandbox Payment Integration & Signed Webhook Processing

24.7 Capstone Sandbox Payment Integration & Signed Webhook Processing [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Creating sandbox payment intents (`orderId` • `amountPaise` • `currency: "INR"`) • Verifying inbound `payment.captured` / `payment.failed` webhooks via `express.raw` + `HMAC-SHA256` (`timingSafeEqual`) + `5m` timestamp window + `providerEventId` unique idempotency! • Production Backend Pitfall & Guardrail (Capstone Sandbox Payment Integration & Signed Webhook Processing) • Hands-On TypeScript / Node.js Implementation (Capstone Sandbox Payment Integration & Signed Webhook Processing) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Capstone Sandbox Payment Integration

Purpose: Transitioning orders to `PAID` reliably even if the user closes their browser tab before redirect!

What Topic Is Next → Next Topic 24.8: Capstone Order Status State Machine & Refund Handling

24.8 Capstone Order Status State Machine & Refund Handling [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Enforcing strict legal state transitions (`PENDING_PAYMENT -> PAID -> CONFIRMED -> PACKED -> SHIPPED -> DELIVERED` • `CANCELLED` • `REFUNDED`) with audit history timestamps • Automatic stock restock on cancellation! • Production Backend Pitfall & Guardrail (Capstone Order Status State Machine & Refund Handling) • Hands-On TypeScript / Node.js Implementation (Capstone Order Status State Machine & Refund Handling) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Capstone Order Status State Machine

Purpose: Preventing invalid jumps like `PENDING_PAYMENT -> DELIVERED` or cancelling an already `DELIVERED` order.

What Topic Is Next → Next Topic 24.9: Capstone File Uploads & S3-Compatible Object Storage (MinIO)

24.9 Capstone File Uploads & S3-Compatible Object Storage (MinIO) [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Multer + magic-byte validation (`PNG`/`JPEG`/`WEBP` product images • `PDF` GST invoices) stored in private Dockerized `MinIO` buckets with presigned upload/download URLs • Production Backend Pitfall & Guardrail (Capstone File Uploads & S3-Compatible Object Storage (MinIO)) • Hands-On TypeScript / Node.js Implementation (Capstone File Uploads & S3-Compatible Object Storage (MinIO)) • Capstone File Uploads & S3-Compatible Object Storage (MinIO) — Latency, Throughput & Interview Trade-Offs • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Capstone File Uploads • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards

Purpose: Serving private customer GST invoice PDFs exclusively via `5-minute` ownership-verified presigned URLs.

What Topic Is Next → Next Topic 24.10: Capstone BullMQ Background Jobs & RabbitMQ/Kafka Domain Events

24.10 Capstone BullMQ Background Jobs & RabbitMQ/Kafka Domain Events [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Publishing `OrderPaid` • `OrderShipped` • `OrderCancelled` domain events via Outbox -> RabbitMQ/Kafka + BullMQ workers generating PDF invoices • Sending transactional emails • Production Backend Pitfall & Guardrail (Capstone BullMQ Background Jobs & RabbitMQ/Kafka Domain Events) • Hands-On TypeScript / Node.js Implementation (Capstone BullMQ Background Jobs & RabbitMQ/Kafka Domain Events) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Capstone BullMQ Background Jobs

Purpose: Keeping checkout `< 40ms` fast while invoices, emails, and analytics process reliably in the background.

What Topic Is Next → Next Topic 24.11: Capstone Real-Time Socket.IO Order & Admin Notifications

24.11 Capstone Real-Time Socket.IO Order & Admin Notifications [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: JWT-authenticated Socket.IO (`@socket.io/redis-adapter`) pushing live `order:status_changed` events to `user:${buyerId}` • `inventory:low_stock` alerts to `/admin` namespace! • Production Backend Pitfall & Guardrail (Capstone Real-Time Socket.IO Order & Admin Notifications) • Hands-On TypeScript / Node.js Implementation (Capstone Real-Time Socket.IO Order & Admin Notifications) • Capstone Real-Time Socket.IO Order & Admin Notifications — Latency, Throughput & Interview Trade-Offs • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Capstone Real-Time Socket.IO Order • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards

Purpose: Giving buyers instant live order tracking updates the moment warehouse staff mark an order `SHIPPED`.

What Topic Is Next → Next Topic 24.12: Capstone Rate Limiting, Centralized Errors & Structured Observability

24.12 Capstone Rate Limiting, Centralized Errors & Structured Observability [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Tiered Redis rate limiters (`/auth` • `/orders` • `/webhooks` • `/products`) • Production Backend Pitfall & Guardrail (Capstone Rate Limiting, Centralized Errors & Structured Observability) • Hands-On TypeScript / Node.js Implementation (Capstone Rate Limiting, Centralized Errors & Structured Observability) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Capstone Rate Limiting

Purpose: Making the entire Capstone observable and OWASP-hardened for production review.

What Topic Is Next → Next Topic 24.13: Capstone Swagger / OpenAPI 3.1 & Automated Postman Collection

24.13 Capstone Swagger / OpenAPI 3.1 & Automated Postman Collection [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Serving complete `/api/docs` OpenAPI 3.1 specification + exporting `postman/ecommerce-capstone.collection.json` covering Auth -> Catalog -> Cart -> Coupon -> Order -> Webhook -> Admin Ship flows • Production Backend Pitfall & Guardrail (Capstone Swagger / OpenAPI 3.1 & Automated Postman Collection) • Hands-On TypeScript / Node.js Implementation (Capstone Swagger / OpenAPI 3.1 & Automated Postman Collection) • Capstone Swagger / OpenAPI 3.1 & Automated Postman Collection — Latency, Throughput & Interview Trade-Offs • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Capstone Swagger / OpenAPI 3.1 • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards • Diagnostics, Profiling & Telemetry: structured telemetry, profiler traces & CI checks

Purpose: Allowing evaluators and hiring managers to test all 24 functional requirements in 2 minutes!

What Topic Is Next → Next Topic 24.14: Capstone Automated Test Suite (Jest / Vitest & Supertest)

24.14 Capstone Automated Test Suite (Jest / Vitest & Supertest) [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Unit tests (pricing • Coupons • State machine • HMAC) + Real-DB Supertest integration suites (Auth rotation • Production Backend Pitfall & Guardrail (Capstone Automated Test Suite (Jest / Vitest & Supertest)) • Hands-On TypeScript / Node.js Implementation (Capstone Automated Test Suite (Jest / Vitest & Supertest)) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Capstone Automated Test Suite (Jest / Vitest & Super...

Purpose: Proving production reliability automatically in GitHub Actions CI.

What Topic Is Next → Next Topic 24.15: Capstone Multi-Container Docker Compose & Nginx Deployment

24.15 Capstone Multi-Container Docker Compose & Nginx Deployment [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Multi-stage non-root `Dockerfile` • `compose.yaml` (`nginx` • `migrate` • `api` • Production Backend Pitfall & Guardrail (Capstone Multi-Container Docker Compose & Nginx Deployment) • Hands-On TypeScript / Node.js Implementation (Capstone Multi-Container Docker Compose & Nginx Deployment) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Capstone Multi-Container Docker Compose

Purpose: Demonstrating live containerized deployment (`https` / Nginx reverse-proxied) during Capstone presentation.

What Topic Is Next → Next Topic 24.16: GitHub Portfolio Polish, Mock Interviews & Final Capstone Viva

24.16 GitHub Portfolio Polish, Mock Interviews & Final Capstone Viva [55 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Polishing all 6 mandatory GitHub repositories (architecture diagrams • ER diagrams • Benchmark tables • Setup guides) • Production Backend Pitfall & Guardrail (GitHub Portfolio Polish, Mock Interviews & Final Capstone Viva) • Hands-On TypeScript / Node.js Implementation (GitHub Portfolio Polish, Mock Interviews & Final Capstone Viva) • Idiomatic NBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Node.js Backend Developer – 6-Month Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for GitHub Portfolio Polish

Purpose: Graduating job-ready for Node.js Backend Developer, TypeScript Backend Engineer, Express API Developer, and Backend Software Engineer roles!

What Topic Is Next → Module 24 Capstone → End-of-Course Live Client Project: Enterprise E-Commerce and Order Management Backend

Module 24 Hands-On Capstone Project (100 Marks): Project 6 (Final Capstone): Enterprise E-Commerce and Order Management Backend — Architect, implement, secure, load-test, document, and containerize a complete, production-ready **Enterprise E-Commerce and Order Management Backend** in **TypeScript 5, Node.js 22, and Exp...

3. LIVE CLIENT PRODUCTION PROJECT & CAREER SCOPE

End-of-Course Live Client Production Project + Official Internship Confirmation Letter

Project: Enterprise E-Commerce and Order Management Backend • **Confirmed Role:** Node.js Backend Developer Project Intern

Domain & Production Infra: Live Enterprise Node.js, TypeScript & Express Backend Engineering Production Engagement (TypeScript, Node.js, Express.js, PostgreSQL + Prisma, MongoDB + Mongoose, JWT/OAuth/RBAC, Sandbox Payments & Signed Webhooks, Redis, BullMQ, RabbitMQ/Kafka, Socket.IO, Jest/Supertest, Docker Compose & Nginx)

- Architected and implemented 'Enterprise E-Commerce and Order Management Backend' using JavaScript ES6+ & Strict TypeScript 5, Node.js 22 LTS (V8, libuv & Event Loop), Buffers, Streams & Async File Processing, Express.js Layered REST Architecture, OpenAPI 3.1, Swagger UI & Zod Validation under Senior Working IT Professional code review.
- Engineered production authentication/RBAC, schema migrations, input validation, caching/security hardening, and clean modular layers.
- Completed automated unit/integration test suites, Docker/IaC packaging, and Git pull-request review sprints.
- Deployed and defended the complete production build with observability, runbooks, and 1-on-1 architecture viva sign-off.

Target Job Roles: Node.js Backend Developer – 6-Month Industry-Ready Career Program Developer, Associate Node.js, TypeScript & Express Backend Engineering Engineer, Software Development Engineer (SDE-1)

Realistic Annual CTC Scope: **Fresher / Entry-Level (0–1 Yr):** ₹4.5 – ₹8.5 LPA • $55K–$85K Global **Mid-Level Product Engineer (1–3 Yrs):** ₹9.5 – ₹19.0 LPA • $85K–$130K Global **Senior / Multi-Stack Specialist (3–6+ Yrs):** ₹20.0 – ₹38.0+ LPA • $130K–$185K+ Global

Optional Worthwhile Global Certifications (100% Exam-Success Mentorship Available): **Professional React, TypeScript & Modern Frontend Specialist (FE-PRO):** ₹9.0 – ₹22.0 LPA • $80K–$130K Global • **OpenJS Node.js Application Developer (JSNAD):** ₹12.0 LPA – ₹27.0 LPA (Hands-On Coding Exam) • **OpenJS Node.js Services Developer (JSNSD):** ₹14.0 LPA – ₹30.0 LPA (API & Security Exam)

MockAttempt Academy Pvt. Ltd. • Email: support@mockattempt.com • Phone/WhatsApp: +91 95400 07917 • Web: https://mockattempt.com/training/nodejs-backend-developer-career-program

Try 2 Free Topic-Wise Timed Topic Mock Tests & Spoken MockAttempt AI Interviews daily on every topic above before enrolling.