Admissions & Support:
support@mockattempt.com
Phone / WhatsApp: +91 98765 43210
Course Code: TSCR • **Updated:** 01 Nov 2026
https://mockattempt.com/courses/typescript-training

TSCR • FULL-STACK TYPE-SAFE JAVASCRIPT & SYSTEMS **BEGINNER TO STAFF TYPESCRIPT ENGINEER**

100% LIVE INDUSTRY EXPERT COHORT + INTERNSHIP LETTER

TypeScript Training

Master Strict TypeScript 5 from `tsc` Compilation, Type Erasure, `unknown` vs `any` & `satisfies` Inference to `strictNullChecks`, `readonly` Arrays/Tuples, Discriminated Unions, `never` Exhaustiveness, `as const` vs Enums & Typed `Map`/`Set` Collections — with runnable TypeScript 5 programs, 10×50 MCQ Mock Tests per topic, and Spoken Voice + Gemini AI Interviews at ₹2,499.

Duration: 14–16 Weeks • Live + Hands-On Projects (205 Hrs) • **Modules:** 16 • **Live Topics:** 42 • **Labs & Tasks:** 84 • **Mocks / AI Vivas:** 244 / 244 • **Projects:** 16 Module + 1 Live Client Prod

Fixed Non-Negotiable Tuition Fee: Base Fee **₹2,499** + 18% GST (₹450) = **Total Incl. GST: ₹2,949**

Live Batch Tracks: Weekday Live Online Batch (Mon – Fri) (Monday to Friday (5 Live Interactive Classes / Week) • 1.5 Hours / Day (60 Mins Live Architecture & Coding + 30 Mins Hands-On Lab) • 17–19 Weeks Fast-Track) | Weekend Intensive Live Online Batch (Sat – Sun) (Saturday & Sunday (Designed for Working Professionals & College Students) • 3.0 Hours / Day (2 Hrs Deep-Dive Live Coding + 1 Hr Guided Project Build) • 19–21 Weeks Weekend Track)

Core Technologies & Engineering Stack: TypeScript 5.x (`strict: true`) • tsc Compiler, tsx & Node.js Native Strip • Structural Typing & Control-Flow Narrowing • `unknown`, `never` & `satisfies` Operator • Readonly Arrays, Labeled Tuples & Unions • `as const` Objects & String/Const Enums • Typed ES6 `Map<K,V>`, `Set<T>` & Iterators

1. PROGRAM MASTERY OUTCOMES & ELIGIBILITY

End-to-End Engineering Outcomes:

1. Configure rock-solid `tsconfig.json` (`strict: true`, `noUncheckedIndexedAccess`, `exactOptionalPropertyTypes`, `verbatimModuleSyntax`) across full-stack workspaces
2. Eliminate runtime `TypeError` crashes using `unknown` narrowing, `strictNullChecks`, optional chaining (`?.`), nullish coalescing (`??`), and `satisfies`
3. Model impossible-to-misuse domain states with Labeled `readonly` Tuples, Discriminated Unions, `never` exhaustiveness checks, and O(1) `Map` / `Set` structures
4. Clear 200 Timed Mock Exams (10,000 TypeScript MCQs) and 200 Spoken Voice + Gemini AI Full-Stack TypeScript Interviews
5. Strict compiler configuration (`strict: true`, `noUncheckedIndexedAccess`, `exactOptionalPropertyTypes`) with zero `any`.
6. Advanced type-level modeling: Generics, Discriminated Unions, Type Narrowing, Mapped/Conditional Types, and `infer`.
7. Bridging compile-time types and untrusted I/O boundaries using Zod/Valibot runtime schema validation.
8. Shared full-stack API contracts, ESM/CJS package configuration, declaration files (`.d.ts`), and typed error handling.

Eligibility & Prerequisites: Basic JavaScript familiarity (variables, functions, objects, arrays) — teaches static typing and TypeScript 5 compiler rules step-by-step from scratch • Any laptop/PC running Node.js 20/22 LTS, npm/pnpm, `typescript` (`tsc`), and VS Code • No prior TypeScript industry experience required — Module 1 starts from environment setup, core mental models, and plain-English fundamentals. • Basic computer & internet familiarity on a Windows, macOS, or Linux laptop (8 GB+ RAM recommended for smooth live coding labs). • Ideal for B.Tech / BCA / MCA / B.Sc IT students, non-CS beginners, career switchers, and working developers upgrading to 2026 production standards. • Commitment to attend live Weekday or Weekend classes, complete daily hands-on lab checklists, and practice topic-wise Mock Tests & AI Interviews.

2. DETAILED MODULE & TOPIC-WISE SYLLABUS (16 MODULES • 42 LIVE TOPICS)

MODULE 1 OF 16 • 10 LIVE + LAB HOURS • 2 TOPICS • 1 CAPSTONE PROJECT (100 MARKS)

Module 1: TypeScript Foundations, Static Type Checking & Type Erasure vs JavaScript

Understand TypeScript as a typed syntactic superset of JavaScript, compile-time Type Erasure, and why static type checking catches bugs before production.

Objectives: ✓ Explain how TypeScript compiles down to clean standards-compliant ECMAScript with complete compile-time Type Erasure • ✓ Contrast compile-time static structural verification in TypeScript against dynamic runtime duck-typing in JavaScript • ✓ Recognize why untrusted external JSON/API boundaries still require runtime schema validation (e.g. Zod / type guards) after type erasure

1.1 TypeScript 5 Overview — Syntactic Superset, Compile-Time Type Erasure & Zero-Cost Abstractions [50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Purpose: To understand and apply TypeScript 5 Overview — Syntactic Superset, Compile-Time Type Erasure & Zero-Cost Abstractions in real TypeScript systems — specifically: How Type...

1.2 TypeScript vs JavaScript — Static Structural Checking vs Dynamic Coercion Traps [50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Purpose: To understand and apply TypeScript vs JavaScript — Static Structural Checking vs Dynamic Coercion Traps in real TypeScript systems — specifically: Preventing JavaScript's...

Module 1 Hands-On Capstone Project (100 Marks): Zero-Runtime-Overhead Invoice Calculator & Boundary Type Guard — Write a strict TypeScript 5 billing module demonstrating compile-time interface enforcement paired with a runtime `unknown` type-predicate guard for untrusted `JSON.parse` payloads.

MODULE 2 OF 16 • 10 LIVE + LAB HOURS • 2 TOPICS • 1 CAPSTONE PROJECT (100 MARKS)

Module 2: Structural Duck Typing, Generics Overview & Production `tsconfig.json` Setup

Master structural ("duck") typing vs nominal typing, modern TypeScript 5 capabilities, and production `tsconfig.json` compiler flags (`strict`, `noUncheckedIndexedAccess`, `moduleResolution`).

Objectives: ✓ Understand TypeScript's structural type system (shape compatibility) versus Java/C# nominal class inheritance • ✓ Install `typescript`, `tsx`, and `@types/node` cleanly as project `devDependencies` rather than drifting global installs • ✓ Configure `tsconfig.json` with `strict: true`, `noUncheckedIndexedAccess: true`, `target: "ES2022"`, and `verbatimModuleSyntax: true`

2.1 TypeScript Core Features — Structural ("Duck") Typing, Excess Property Checks & Generics [50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Purpose: To understand and apply TypeScript Core Features — Structural ("Duck") Typing, Excess Property Checks & Generics in real TypeScript systems — specifically: Why TypeScript...

2.2 Installing & Configuring TypeScript 5 — Local `devDependencies`, `tsc --init` & `tsconfig.json` Flags [50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Purpose: To understand and apply Installing & Configuring TypeScript 5 — Local `devDependencies`, `tsc --init` & `tsconfig.json` Flags in real TypeScript systems — specifically: S...

Module 2 Hands-On Capstone Project (100 Marks): Structural Repository Adapter & Hardened `tsconfig.json` Validator — Build a structurally typed logger/storage adapter demonstrating shape compatibility across independent classes alongside a `tsconfig.json` strictness auditor.

MODULE 3 OF 16 • 10 LIVE + LAB HOURS • 2 TOPICS • 1 CAPSTONE PROJECT (100 MARKS)

Module 3: First TypeScript Program & The Type Hierarchy (`unknown`, `any`, `never`, `void`)

Compile and run clean `.ts` programs while mastering TypeScript's complete type lattice from Top Types (`unknown`, unsafe `any`) down to primitives, literals, and the Bottom Type (`never`).

Objectives: ✓ Compile `.ts` modules cleanly without global scope collisions (`export {}` module isolation) and source maps • ✓ Replace unsafe `any` escape hatches with type-safe `unknown` top-type narrowing across API and `catch (err: unknown)` blocks • ✓ Distinguish `void` (ignored/empty return) from `never` (functions that throw unconditionally or unreachable control-flow branches)

3.1 First TypeScript Program — Module Scope Isolation, `tsc` Emit Diagnostics & `noEmitOnError` [50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Purpose: To understand and apply First TypeScript Program — Module Scope Isolation, `tsc` Emit Diagnostics & `noEmitOnError` in real TypeScript systems — specifically: Writing, ty...

3.2 TypeScript Type Hierarchy — Primitives (`string` vs `String`), Top Type `unknown`, `any`, `void` & Bottom Type `never` [50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Purpose: To understand and apply TypeScript Type Hierarchy — Primitives (`string` vs `String`), Top Type `unknown`, `any`, `void` & Bottom Type `never` in real TypeScript systems...

Module 3 Hands-On Capstone Project (100 Marks): Safe `unknown` Error Normalizer & Type Lattice Inspector — Implement a zero-`any` error extraction utility (`toErrorMessage(err: unknown): string`) for `try / catch` blocks alongside an unreachable `assertNever(x: never): never` helper.

MODULE 4 OF 16 • 11 LIVE + LAB HOURS • 2 TOPICS • 1 CAPSTONE PROJECT (100 MARKS)

Module 4: Explicit Type Annotations vs Type Inference & The `satisfies` Operator

Know exactly when to write explicit `: Type` annotations (public boundaries, empty collections) versus letting TypeScript infer narrow literal types (`const`, `as const`, `satisfies`).

Objectives: ✓ Annotate public function parameters, return contracts, and initially empty arrays explicitly to catch implementation drift at the source • ✓ Understand `let` widening (`"dark"` -> `string`) versus `const` literal inference (`"dark"`) and contextual callback typing • ✓ Use TypeScript's `satisfies` operator to validate an object against a schema WITHOUT widening away its exact literal keys and value types

4.1 TypeScript Type Annotations — Function Signatures, Return Contracts, Object Shapes & Index Signatures [50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Purpose: To understand and apply TypeScript Type Annotations — Function Signatures, Return Contracts, Object Shapes & Index Signatures in real TypeScript systems — specifically: P...

4.2 TypeScript Type Inference — `const` vs `let` Widening, Contextual Typing & `satisfies` [50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Purpose: To understand and apply TypeScript Type Inference — `const` vs `let` Widening, Contextual Typing & `satisfies` in real TypeScript systems — specifically: How TypeScript's...

Module 4 Hands-On Capstone Project (100 Marks): Route & Theme Config Validator Powered by `satisfies` & Explicit API Return Contracts — Design a configuration map validated with `satisfies Record<string, RouteConfig>` so typos fail compilation while individual route paths retain exact literal autocomplete.

MODULE 5 OF 16 • 11 LIVE + LAB HOURS • 2 TOPICS • 1 CAPSTONE PROJECT (100 MARKS)

Module 5: Type Assertions (`as` vs `as const`) & Strict Nullability (`null` vs `undefined`)

Understand why `as Type` assertions bypass soundness (and never convert values at runtime!), and eliminate billion-dollar null crashes with `strictNullChecks`, `?.`, `??`, and `exactOptionalPropertyTypes`.

Objectives: ✓ Recognize why `val as User` performs ZERO runtime coercion or validation and replace unsafe double assertions (`as unknown as T`) with type guards • ✓ Leverage `as const` const-assertions to freeze deep object/array literals into immutable literal types • ✓ Differentiate `undefined` (uninitialized / omitted) from `null` (intentional absence) and avoid `||` falsy bugs on `0` and `""` by using `??`

5.1 TypeScript Type Assertions — `as T` Unsoundness Risks, Double Assertions & Safe `as const` [50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Purpose: To understand and apply TypeScript Type Assertions — `as T` Unsoundness Risks, Double Assertions & Safe `as const` in real TypeScript systems — specifically: Distinguishi...

5.2 `null` vs `undefined` in Strict TypeScript — `strictNullChecks`, `?.` & Nullish Coalescing `??` [50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Purpose: To understand and apply `null` vs `undefined` in Strict TypeScript — `strictNullChecks`, `?.` & Nullish Coalescing `??` in real TypeScript systems — specifically: Disting...

Module 5 Hands-On Capstone Project (100 Marks): Null-Safe User Preference Merger (`??` vs `||`) & `as const` Permission Matrix — Build a configuration resolver that preserves valid falsy values (`timeoutMs: 0`, `prefix: ""`) using nullish coalescing `??` instead of `||`, paired with an `as const` role tuple.

MODULE 6 OF 16 • 10 LIVE + LAB HOURS • 2 TOPICS • 1 CAPSTONE PROJECT (100 MARKS)

Module 6: Block-Scoped Bindings (`const` / `let` / TDZ) & Immutable `readonly` Arrays

Eliminate `var` hoisting/closure bugs with Temporal Dead Zone (`const` / `let`) rules and prevent accidental in-place mutation bugs with `readonly T[]` and `noUncheckedIndexedAccess`.

Objectives: ✓ Understand `var` function-hoisting closure bugs versus block-scoped `let` / `const` and the Temporal Dead Zone (TDZ) • ✓ Distinguish shallow reference locking (`const arr = []`) from true compile-time immutability (`readonly number[]` / `ReadonlyArray<T>`) • ✓ Guard out-of-bounds array index reads (`arr[99] -> T | undefined`) with `noUncheckedIndexedAccess: true`

6.1 TypeScript Variables — `const` vs `let` vs `var`, Temporal Dead Zone (TDZ) & `Readonly<T>` [50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Purpose: To understand and apply TypeScript Variables — `const` vs `let` vs `var`, Temporal Dead Zone (TDZ) & `Readonly<T>` in real TypeScript systems — specifically: Why modern T...

6.2 TypeScript Arrays — `readonly T[]`, Numeric `.sort()` Trap & `noUncheckedIndexedAccess` [50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Purpose: To understand and apply TypeScript Arrays — `readonly T[]`, Numeric `.sort()` Trap & `noUncheckedIndexedAccess` in real TypeScript systems — specifically: Typing homogene...

Module 6 Hands-On Capstone Project (100 Marks): Non-Mutating Score Sorter (`readonly number[]`) & Safe Bounds Reader — Write pure array analytics functions accepting `readonly number[]` (banning `.sort()` / `.splice()` / `.push()` mutations on caller inputs) and returning safe `number | undefined` bounds res...

MODULE 7 OF 16 • 11 LIVE + LAB HOURS • 2 TOPICS • 1 CAPSTONE PROJECT (100 MARKS)

Module 7: Labeled `readonly` Tuples & Exhaustive Discriminated Union Types

Model fixed-arity positional records with `readonly` Labeled Tuples and build ironclad `Result<T, E>` state machines using Discriminated Unions and `never` exhaustiveness.

Objectives: ✓ Define fixed-length Labeled Tuples (`readonly [status: number, body: string]`) and lock away `.push()` tuple-length bypass bugs with `readonly` • ✓ Model `Result<T, E>` and async state machines with literal discriminant tags (`status: "ok" | "err"`) • ✓ Enforce compile-time exhaustiveness across union branches using `const _exhaustive: never = state`

7.1 TypeScript Tuples — Fixed-Arity Positional Types, Labeled Elements, Rest Tuples & `readonly` Guard [50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Purpose: To understand and apply TypeScript Tuples — Fixed-Arity Positional Types, Labeled Elements, Rest Tuples & `readonly` Guard in real TypeScript systems — specifically: Repr...

7.2 TypeScript Union Types (`A | B`) — Discriminated Unions, Shared Members & `never` Exhaustiveness [50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Purpose: To understand and apply TypeScript Union Types (`A | B`) — Discriminated Unions, Shared Members & `never` Exhaustiveness in real TypeScript systems — specifically: Modeli...

Module 7 Hands-On Capstone Project (100 Marks): Type-Safe `Result<T, E>` Parser Returning `readonly` Coordinate Tuples — Build a GPS coordinate parser returning a discriminated union `{ ok: true; point: readonly [lat: number, lng: number] } | { ok: false; reason: string }` with an exhaustive formatter.

MODULE 8 OF 16 • 11 LIVE + LAB HOURS • 2 TOPICS • 1 CAPSTONE PROJECT (100 MARKS)

Module 8: Enums (`enum` vs `as const` Erasable Syntax) & Typed `Map<K, V>` Dictionaries

Understand numeric reverse-mapping quirks vs string enums vs modern `as const` objects (`erasableSyntaxOnly`), and build O(1) prototype-safe dictionaries with `Map<K, V>`.

Objectives: ✓ Distinguish numeric `enum` reverse-mapping bloat & falsy `0` pitfalls from explicit String Enums and `as const` objects • ✓ Understand why `erasableSyntaxOnly` (Node.js native TS execution) favors `as const` objects + union types over runtime-emitting `enum` • ✓ Choose `Map<K, V>` over plain `Record<string, V>` for dynamic untrusted keys (preventing `__proto__` pollution), non-string keys, and O(1) `.size`

8.1 TypeScript Enums — Numeric Reverse-Mapping Quirks, String Enums & `as const` (`erasableSyntaxOnly`) [50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Purpose: To understand and apply TypeScript Enums — Numeric Reverse-Mapping Quirks, String Enums & `as const` (`erasableSyntaxOnly`) in real TypeScript systems — specifically: Why...

8.2 TypeScript `Map<K, V>` — O(1) Hash Lookups, Insertion Order, Non-String Keys & Prototype Safety [50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Purpose: To understand and apply TypeScript `Map<K, V>` — O(1) Hash Lookups, Insertion Order, Non-String Keys & Prototype Safety in real TypeScript systems — specifically: Using g...

Module 8 Hands-On Capstone Project (100 Marks): Prototype-Safe LRU / Rate-Limit Counter Using `Map<string, Entry>` & `as const` Status Codes — Build an O(1) sliding request counter backed by `Map<string, number>` (maintaining insertion order and immunity to `"__proto__"` keys) paired with an `as const` status map.

MODULE 9 OF 16 • 11 LIVE + LAB HOURS • 2 TOPICS • 1 CAPSTONE PROJECT (100 MARKS)

Module 9: Unique `Set<T>` Algebra & Control-Flow Type Narrowing (`if` / `switch`)

Deduplicate & test membership in O(1) with `Set<T>` (`SameValueZero` equality) and master TypeScript's Control-Flow Analysis across `if/else`, `typeof`, `in`, `instanceof`, and `switch`.

Objectives: ✓ Replace O(N^2) `Array.includes()` loops with O(1) `Set<T>.has()` lookups while understanding reference equality on object elements • ✓ Understand `SameValueZero` equality inside `Set` (`NaN` deduplicates cleanly, whereas two `{ id: 1 }` literals are distinct references) • ✓ Master TypeScript Control-Flow Narrowing (`typeof x === "object" && x !== null`, `"key" in obj`, `Array.isArray()`, and truthiness pitfalls)

9.1 TypeScript `Set<T>` — O(1) Membership, Deduplication, `SameValueZero` & Object Reference Caveats [50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Purpose: To understand and apply TypeScript `Set<T>` — O(1) Membership, Deduplication, `SameValueZero` & Object Reference Caveats in real TypeScript systems — specifically: Elimin...

9.2 Decision Making & Control-Flow Narrowing — `if/else`, `typeof null` Trap, `in`, `Array.isArray` & `switch` [50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Purpose: To understand and apply Decision Making & Control-Flow Narrowing — `if/else`, `typeof null` Trap, `in`, `Array.isArray` & `switch` in real TypeScript systems — specifical...

Module 9 Hands-On Capstone Project (100 Marks): O(N) Permission Intersection Engine (`Set<string>`) & Control-Flow Narrowing Router — Build an O(N + M) role-permission verifier using `Set<string>` alongside a multi-input command router narrowing `string | number | { cmd: string } | null` cleanly.

MODULE 10 OF 16 • 10 LIVE + LAB HOURS • 2 TOPICS • 1 CAPSTONE PROJECT (100 MARKS)

Module 10: Definite Loops (`for`, `for...of`, `for...in`) & `.forEach()` Async Pitfalls

Choose accurately between `for`, `for...of` (values + `await` + `break`), `for...in` (`string` keys), and `.forEach()` — avoiding classic unhandled-Promise `.forEach(async ...)` production bugs!

Objectives: ✓ Contrast `for (const val of items)` (iterable values, supports `break`/`continue`/`return`/`await`) with `for (const key in obj)` (enumerable string keys + `Object.hasOwn`) • ✓ Understand why `keyof T` indexing inside `Object.keys(obj)` widens to `string[]` under structural typing and how `Object.entries` works cleanly • ✓ Eliminate the #1 async TypeScript/Node bug: passing `async` callbacks into `.forEach()` where `await` is completely ignored!

10.1 TypeScript Definite Loops — Classic `for`, `for...of` Iterables vs `for...in` Keys & `Object.keys` Widening [50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Purpose: To understand and apply TypeScript Definite Loops — Classic `for`, `for...of` Iterables vs `for...in` Keys & `Object.keys` Widening in real TypeScript systems — specifica...

10.2 TypeScript `Array.prototype.forEach` — `void` Return Ignorance, No `break`, & The `async` Trap [50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Purpose: To understand and apply TypeScript `Array.prototype.forEach` — `void` Return Ignorance, No `break`, & The `async` Trap in real TypeScript systems — specifically: Understa...

Module 10 Hands-On Capstone Project (100 Marks): Sequential Async Batch Processor (`for...of` + `await`) vs Parallel `Promise.all` Runner — Replace a broken `items.forEach(async item => ...)` pipeline with deterministic sequential `for (const item of items) { await ... }` and bounded `Promise.all(items.map(...))` execution.

MODULE 11 OF 16 • 16 LIVE + LAB HOURS • 4 TOPICS • 1 CAPSTONE PROJECT (100 MARKS)

Type-Level Programming: Conditional Types, infer & Template Literals

Production-grade TypeScript Training engineering module covering Type-Level Programming: Conditional Types, infer & Template Literals with hands-on labs and capstone build.

Objectives: ✓ Distributive Conditional Types & the infer Keyword — Purpose: Extract Promise payloads, function parameters, and API response shapes automatically. • ✓ Recursive Mapped Types & Template Literal Path Parsers — Purpose: Parse \"/orders/:orderId/items/:itemId\" into exact { orderId: string; itemId: string } params at compile time!. • ✓ Branded / Nominal Types (UserId vs OrderId, INRPaise) & const Type Params — Purpose: Make it a compile error to accidentally pass an OrderId where a UserId is expected. • ✓ satisfies Operator vs Type Annotations — Purpose: Validate config objects against a schema while preserving exact literal autocomplete.

11.1 Distributive Conditional Types & the infer Keyword [65 mins • Assignment (100M) • Hands-On Lab (50M) • 2 Mock Sets • 2 AI Vivas]

Purpose: To understand and apply Distributive Conditional Types & the infer Keyword in real TypeScript systems — specifically: Extract Promise payloads, function parameters, and A...

11.2 Recursive Mapped Types & Template Literal Path Parsers [65 mins • Assignment (100M) • Hands-On Lab (50M) • 2 Mock Sets • 2 AI Vivas]

Purpose: To understand and apply Recursive Mapped Types & Template Literal Path Parsers in real TypeScript systems — specifically: Parse \"/orders/:orderId/items/:itemId\" into ex...

11.3 Branded / Nominal Types (UserId vs OrderId, INRPaise) & const Type Params [65 mins • Assignment (100M) • Hands-On Lab (50M) • 2 Mock Sets • 2 AI Vivas]

Purpose: To understand and apply Branded / Nominal Types (UserId vs OrderId, INRPaise) & const Type Params in real TypeScript systems — specifically: Make it a compile error to ac...

11.4 satisfies Operator vs Type Annotations [65 mins • Assignment (100M) • Hands-On Lab (50M) • 2 Mock Sets • 2 AI Vivas]

Purpose: To understand and apply satisfies Operator vs Type Annotations in real TypeScript systems — specifically: Validate config objects against a schema while preserving exact...

Module 11 Hands-On Capstone Project (100 Marks): Compile-Time Typed Router & Event-Bus SDK in TypeScript — Design, implement, test, and document 'Compile-Time Typed Router & Event-Bus SDK in TypeScript' applying every production concept from Module 11 (Type-Level Programming: Conditional Types, i...

MODULE 12 OF 16 • 16 LIVE + LAB HOURS • 4 TOPICS • 1 CAPSTONE PROJECT (100 MARKS)

Runtime Schema Validation (Zod), Result Error Unions & End-to-End RPC

Production-grade TypeScript Training engineering module covering Runtime Schema Validation (Zod), Result Error Unions & End-to-End RPC with hands-on labs and capstone build.

Objectives: ✓ Zod Transforms, Refinements, Discriminated Unions & z.infer — Purpose: Parse untrusted HTTP/JSON/env inputs at system boundaries into guaranteed TypeScript types. • ✓ Typed Result<T, E> Unions vs Untyped Thrown Exceptions — Purpose: Force callers to handle NotFound, QuotaExceeded, and ValidationError explicitly. • ✓ OpenAPI-to-TypeScript Codegen & End-to-End Typed API Clients — Purpose: Catch backend response field renames immediately in frontend CI typechecks. • ✓ Strict TSConfig Hardening (noUncheckedIndexedAccess, exactOptionalPropertyTypes) — Purpose: Eliminate silent undefined array-index bugs across production codebases.

12.1 Zod Transforms, Refinements, Discriminated Unions & z.infer [65 mins • Assignment (100M) • Hands-On Lab (50M) • 2 Mock Sets • 2 AI Vivas]

Purpose: To understand and apply Zod Transforms, Refinements, Discriminated Unions & z.infer in real TypeScript systems — specifically: Parse untrusted HTTP/JSON/env inputs at sys...

12.2 Typed Result<T, E> Unions vs Untyped Thrown Exceptions [65 mins • Assignment (100M) • Hands-On Lab (50M) • 2 Mock Sets • 2 AI Vivas]

Purpose: To understand and apply Typed Result<T, E> Unions vs Untyped Thrown Exceptions in real TypeScript systems — specifically: Force callers to handle NotFound, QuotaExceeded,...

12.3 OpenAPI-to-TypeScript Codegen & End-to-End Typed API Clients [65 mins • Assignment (100M) • Hands-On Lab (50M) • 2 Mock Sets • 2 AI Vivas]

Purpose: To understand and apply OpenAPI-to-TypeScript Codegen & End-to-End Typed API Clients in real TypeScript systems — specifically: Catch backend response field renames immed...

12.4 Strict TSConfig Hardening (noUncheckedIndexedAccess, exactOptionalPropertyTypes) [65 mins • Assignment (100M) • Hands-On Lab (50M) • 2 Mock Sets • 2 AI Vivas]

Purpose: To understand and apply Strict TSConfig Hardening (noUncheckedIndexedAccess, exactOptionalPropertyTypes) in real TypeScript systems — specifically: Eliminate silent undef...

Module 12 Hands-On Capstone Project (100 Marks): Zero-Drift Full-Stack Type Contract & Env Validator — Design, implement, test, and document 'Zero-Drift Full-Stack Type Contract & Env Validator' applying every production concept from Module 12 (Runtime Schema Validation (Zod), Result Error Un...

MODULE 13 OF 16 • 16 LIVE + LAB HOURS • 4 TOPICS • 1 CAPSTONE PROJECT (100 MARKS)

Full-Stack TypeScript Patterns Across React Props, Hooks & Node Repositories

Production-grade TypeScript Training engineering module covering Full-Stack TypeScript Patterns Across React Props, Hooks & Node Repositories with hands-on labs and capstone build.

Objectives: ✓ Polymorphic React Components (as Prop), Forwarded Refs & Discriminated Props — Purpose: Build ergonomic UI components where mutually exclusive props cannot be mixed. • ✓ Typed Express Request/Response Generics & Async Middleware Wrappers — Purpose: Flow validated params, query, body, and auth user types cleanly into controllers. • ✓ Type-Safe SQL/ORM Query Builders (Prisma / Drizzle / Kysely Concepts) — Purpose: Write relational joins where selected column names and nullability are 100% inferred. • ✓ ECMAScript Decorators, Symbols & AsyncIterators for Streaming Pagination — Purpose: Consume paginated partner APIs cleanly via for await (const page of client.list()).

13.1 Polymorphic React Components (as Prop), Forwarded Refs & Discriminated Props [65 mins • Assignment (100M) • Hands-On Lab (50M) • 2 Mock Sets • 2 AI Vivas]

Purpose: To understand and apply Polymorphic React Components (as Prop), Forwarded Refs & Discriminated Props in real TypeScript systems — specifically: Build ergonomic UI compone...

13.2 Typed Express Request/Response Generics & Async Middleware Wrappers [65 mins • Assignment (100M) • Hands-On Lab (50M) • 2 Mock Sets • 2 AI Vivas]

Purpose: To understand and apply Typed Express Request/Response Generics & Async Middleware Wrappers in real TypeScript systems — specifically: Flow validated params, query, body,...

13.3 Type-Safe SQL/ORM Query Builders (Prisma / Drizzle / Kysely Concepts) [65 mins • Assignment (100M) • Hands-On Lab (50M) • 2 Mock Sets • 2 AI Vivas]

Purpose: To understand and apply Type-Safe SQL/ORM Query Builders (Prisma / Drizzle / Kysely Concepts) in real TypeScript systems — specifically: Write relational joins where sele...

13.4 ECMAScript Decorators, Symbols & AsyncIterators for Streaming Pagination [65 mins • Assignment (100M) • Hands-On Lab (50M) • 2 Mock Sets • 2 AI Vivas]

Purpose: To understand and apply ECMAScript Decorators, Symbols & AsyncIterators for Streaming Pagination in real TypeScript systems — specifically: Consume paginated partner APIs...

Module 13 Hands-On Capstone Project (100 Marks): Type-Safe Multi-Tenant Express + React Workspace Toolkit — Design, implement, test, and document 'Type-Safe Multi-Tenant Express + React Workspace Toolkit' applying every production concept from Module 13 (Full-Stack TypeScript Patterns Across React...

MODULE 14 OF 16 • 16 LIVE + LAB HOURS • 4 TOPICS • 1 CAPSTONE PROJECT (100 MARKS)

Monorepo Project References, Dual ESM/CJS Library Publishing & Type Tests

Production-grade TypeScript Training engineering module covering Monorepo Project References, Dual ESM/CJS Library Publishing & Type Tests with hands-on labs and capstone build.

Objectives: ✓ TS Project References (composite: true) & Turborepo/pnpm Workspaces — Purpose: Typecheck 10-package full-stack monorepos in seconds using incremental .tsbuildinfo caches. • ✓ Modern moduleResolution: \"NodeNext\" vs \"Bundler\" & package.json exports — Purpose: Ship libraries with crisp .d.ts types that work in both Node 22 ESM and Vite bundlers. • ✓ Compile-Time Type Assertions (expectTypeOf / @ts-expect-error) — Purpose: Write unit tests for your complex generic types so refactors never widen types to any. • ✓ Type-Aware ESLint Rules (no-floating-promises, no-misused-promises) in CI — Purpose: Catch unawaited async bugs automatically before code review even begins.

14.1 TS Project References (composite: true) & Turborepo/pnpm Workspaces [65 mins • Assignment (100M) • Hands-On Lab (50M) • 2 Mock Sets • 2 AI Vivas]

Purpose: To understand and apply TS Project References (composite: true) & Turborepo/pnpm Workspaces in real TypeScript systems — specifically: Typecheck 10-package full-stack mon...

14.2 Modern moduleResolution: \"NodeNext\" vs \"Bundler\" & package.json exports [65 mins • Assignment (100M) • Hands-On Lab (50M) • 2 Mock Sets • 2 AI Vivas]

Purpose: To understand and apply Modern moduleResolution: \"NodeNext\" vs \"Bundler\" & package.json exports in real TypeScript systems — specifically: Ship libraries with crisp ....

14.3 Compile-Time Type Assertions (expectTypeOf / @ts-expect-error) [65 mins • Assignment (100M) • Hands-On Lab (50M) • 2 Mock Sets • 2 AI Vivas]

Purpose: To understand and apply Compile-Time Type Assertions (expectTypeOf / @ts-expect-error) in real TypeScript systems — specifically: Write unit tests for your complex generi...

14.4 Type-Aware ESLint Rules (no-floating-promises, no-misused-promises) in CI [65 mins • Assignment (100M) • Hands-On Lab (50M) • 2 Mock Sets • 2 AI Vivas]

Purpose: To understand and apply Type-Aware ESLint Rules (no-floating-promises, no-misused-promises) in CI in real TypeScript systems — specifically: Catch unawaited async bugs au...

Module 14 Hands-On Capstone Project (100 Marks): Published Dual ESM+CJS TypeScript Package with Vitest Type-Tests — Design, implement, test, and document 'Published Dual ESM+CJS TypeScript Package with Vitest Type-Tests' applying every production concept from Module 14 (Monorepo Project References, Dual E...

MODULE 15 OF 16 • 14 LIVE + LAB HOURS • 3 TOPICS • 1 CAPSTONE PROJECT (100 MARKS)

Runtime Boundary Validation (Zod), Branded Types & Type-Safe SDK Design

Self-explanatory list-and-bullet production mastery module covering Runtime Boundary Validation (Zod), Branded Types & Type-Safe SDK Design in TypeScript Training, paired with hands-on labs, capstone repository, Topic Mock Tests, and Spoken Gemini AI Viva.

Objectives: ✓ Bridging Compile-Time & Runtime Safety: Schema Inference with Zod / Valibot — Purpose: Guarantee untrusted JSON from APIs, env vars, and forms matches your static TypeScript types at runtime. • ✓ Branded / Nominal Types (UserId vs OrderId) & Exhaustive Discriminated Unions (never) — Purpose: Make mixing up entity IDs or forgetting an error state a compile-time impossibility. • ✓ Template Literal Types, satisfies Operator & Zero-any Generic Builder Patterns — Purpose: Preserve exact literal autocomplete while validating configuration contracts cleanly. • ✓ Build, verify, and document 'End-to-End Type-Safe API Client SDK with Branded IDs, Zod Schemas & Result Unions' with clean Git commits and edge-case protection. • ✓ Clear 100% of Topic Timed Mock Tests and Spoken Voice + Google Gemini AI Technical Interviews.

15.1 Bridging Compile-Time & Runtime Safety: Schema Inference with Zod / Valibot [65 mins • Assignment (100M) • Hands-On Lab (50M) • 2 Mock Sets • 2 AI Vivas]

Purpose: To understand and apply Bridging Compile-Time & Runtime Safety: Schema Inference with Zod / Valibot in real TypeScript systems — specifically: Guarantee untrusted JSON fr...

15.2 Branded / Nominal Types (UserId vs OrderId) & Exhaustive Discriminated Unions (never) [65 mins • Assignment (100M) • Hands-On Lab (50M) • 2 Mock Sets • 2 AI Vivas]

Purpose: To understand and apply Branded / Nominal Types (UserId vs OrderId) & Exhaustive Discriminated Unions (never) in real TypeScript systems — specifically: Make mixing up en...

15.3 Template Literal Types, satisfies Operator & Zero-any Generic Builder Patterns [65 mins • Assignment (100M) • Hands-On Lab (50M) • 2 Mock Sets • 2 AI Vivas]

Purpose: To understand and apply Template Literal Types, satisfies Operator & Zero-any Generic Builder Patterns in real TypeScript systems — specifically: Preserve exact literal a...

Module 15 Hands-On Capstone Project (100 Marks): End-to-End Type-Safe API Client SDK with Branded IDs, Zod Schemas & Result Unions — Architect, implement, test, and document 'End-to-End Type-Safe API Client SDK with Branded IDs, Zod Schemas & Result Unions' applying every bullet-wise engineering concept from Module 15 (Ru...

MODULE 16 OF 16 • 14 LIVE + LAB HOURS • 3 TOPICS • 1 CAPSTONE PROJECT (100 MARKS)

Monorepo tsconfig Project References, Type Performance & TS Interviews

Self-explanatory list-and-bullet production mastery module covering Monorepo tsconfig Project References, Type Performance & TS Interviews in TypeScript Training, paired with hands-on labs, capstone repository, Topic Mock Tests, and Spoken Gemini AI Viva.

Objectives: ✓ Production tsconfig Hardening: strict, noUncheckedIndexedAccess & exactOptionalPropertyTypes — Purpose: Eliminate undefined array-index crashes and subtle optional-property bugs across teams. • ✓ ESM/CJS Module Resolution, Declaration Maps & Project References Build Speed — Purpose: Publish dual-compatible libraries and keep IDE type-checking instant on large repositories. • ✓ Advanced Type-Level Challenges (Infer, Mapped Modifiers) & Senior TS Interviews — Purpose: Solve real-world generic typing puzzles and architecture questions without type assertions. • ✓ Build, verify, and document 'Strict Multi-Package TypeScript Workspace (`noUncheckedIndexedAccess` + Vitest Type Tests)' with clean Git commits and edge-case protection. • ✓ Clear 100% of Topic Timed Mock Tests and Spoken Voice + Google Gemini AI Technical Interviews.

16.1 Production tsconfig Hardening: strict, noUncheckedIndexedAccess & exactOptionalPropertyTypes [65 mins • Assignment (100M) • Hands-On Lab (50M) • 2 Mock Sets • 2 AI Vivas]

Purpose: To understand and apply Production tsconfig Hardening: strict, noUncheckedIndexedAccess & exactOptionalPropertyTypes in real TypeScript systems — specifically: Eliminate...

16.2 ESM/CJS Module Resolution, Declaration Maps & Project References Build Speed [65 mins • Assignment (100M) • Hands-On Lab (50M) • 2 Mock Sets • 2 AI Vivas]

Purpose: To understand and apply ESM/CJS Module Resolution, Declaration Maps & Project References Build Speed in real TypeScript systems — specifically: Publish dual-compatible li...

16.3 Advanced Type-Level Challenges (Infer, Mapped Modifiers) & Senior TS Interviews [65 mins • Assignment (100M) • Hands-On Lab (50M) • 2 Mock Sets • 2 AI Vivas]

Purpose: To understand and apply Advanced Type-Level Challenges (Infer, Mapped Modifiers) & Senior TS Interviews in real TypeScript systems — specifically: Solve real-world generi...

Module 16 Hands-On Capstone Project (100 Marks): Strict Multi-Package TypeScript Workspace (`noUncheckedIndexedAccess` + Vitest Type Tests) — Architect, implement, test, and document 'Strict Multi-Package TypeScript Workspace (`noUncheckedIndexedAccess` + Vitest Type Tests)' applying every bullet-wise engineering concept from Modu...

3. LIVE CLIENT PRODUCTION PROJECT, GLOBAL CERTIFICATIONS & CAREER SCOPE

End-of-Course Live Client Production Project + Official Internship Confirmation Letter

Project: TypeSafe SDK, End-to-End Typed API Contract Library & Event Bus Package • **Confirmed Role:** TypeScript 5 Systems & Full-Stack Type Architecture Intern

Domain & Production Infra: Developer Platform & Multi-Team SaaS Engineering Client (TypeScript 5 Strict Mode + Zod Runtime Schemas +ESM/CJS Build + Vitest CI Pipeline)

- Engineered zero-any TypeScript 5 modules using generics, discriminated unions, mapped types, and template literals.
- Paired compile-time TypeScript inference with Zod runtime validation schemas for bulletproof API boundaries.
- Built type-safe event emitters, result/error monads, and dual ESM/CJS library bundles with declaration maps.
- Enforced strict compiler flags (noUncheckedIndexedAccess, exactOptionalPropertyTypes) with 100% Vitest type tests.

Aligned Global Vendor Certifications: **META-FE** (Meta Certified Front-End & React Developer Credential) • **JSNAD** (OpenJS Node.js Application Developer (JSNAD)) • **ANG-L2** (Certified Mid-to-Senior Angular Enterprise Developer)

Target Job Roles: TypeScript Software Engineer, Frontend Engineer (React/Angular + TS), Full-Stack TypeScript Developer, SDK & Platform UI Engineer

Realistic Annual CTC Scope: **Fresher / Entry-Level TS Developer (0–1 Yr):** ₹5.2 LPA – ₹9.8 LPA **Mid-Level Full-Stack TypeScript Engineer (1–3 Yrs):** ₹10.5 LPA – ₹21.0 LPA **Staff Frontend / Full-Stack TypeScript Architect (3–6+ Yrs):** ₹22.0 LPA – ₹42.0+ LPA

MockAttempt Academy Pvt. Ltd. • Email: support@mockattempt.com • Phone/WhatsApp: +91 98765 43210 • Web: https://mockattempt.com/courses/typescript-training

Try 2 Free Topic-Wise 50-MCQ Mock Tests & Spoken Gemini AI Interviews daily on every topic above before enrolling.