Admissions & Support:
support@mockattempt.com
Phone / WhatsApp: +91 98765 43210
Course Code: DJNG • **Updated:** 02 Nov 2026
https://mockattempt.com/courses/django-training

DJNG • FULL-STACK PYTHON WEB & REST API ENGINEERING | **BEGINNER TO PRINCIPAL DJANGO 5 & DRF ARCHITECT**

100% LIVE INDUSTRY EXPERT COHORT + INTERNSHIP LETTER

Django Training

Master Python Django 5 from MVT Architecture, URL Dispatching, Generic CBVs & DTL Templates to Relational Models, Migrations, ORM N+1 Optimization (select_related/prefetch_related/F), ModelForms, AbstractUser Auth, Middleware, DRF & Production Deployment — with Python programs, 386 Topic Mock Tests (17,860 MCQs), and Voice + MockAttempt AI Interviews at ₹3,999 + 18% GST.

Duration: 16–18 Weeks • Live + Hands-On Projects (205 Hrs) • **Modules:** 16 • **Live Topics:** 53 • **Subtopics:** 424+ • **Labs & Tasks:** 106 • **Mocks / AI Vivas:** 386 / 386 • **Projects:** 16 Module + 1 Live Client Prod

Fixed Non-Negotiable Tuition Fee: Base Fee **₹3,999** + 18% GST (₹720) = **Total Incl. GST: ₹4,719** • **Effective ROI:** ~₹20/Guided Hr (~₹75/Live Topic)

Faculty: 100% Active Working IT Professionals • **Live Batch Tracks (7:00 AM – 8:00 PM IST Flexible Slots):** Weekday Live Online Batch (Mon – Fri • 1 Hr/Day • 7:00 AM – 8:00 PM) (Monday to Friday (1 Hour Live Session / Day) • 07:00 AM – 08:00 PM IST (Students Can Choose Any 1-Hour Slot Based on Trainer Availability) • 17–19 Weeks Fast-Track) | Weekend Live Online Batch (Sat – Sun • 2 Hrs/Day • 7:00 AM – 8:00 PM) (Saturday & Sunday (2 Hours Live Session / Day) • 07:00 AM – 08:00 PM IST (Students Can Choose Any 2-Hour Slot Based on Trainer Availability) • 19–21 Weeks Weekend Track)

✓ **Trainer LeetCode & HackerRank Mentorship (Python 3.12): Live trainer support to crack LeetCode & HackerRank problems, optimize complexity, and handle edge cases.**

Core Technologies & Engineering Stack: Python 3.12 & Django 5.1 • MVT, URLs & Generic CBVs • Django ORM & Migrations • DTL, WhiteNoise & Bootstrap • ModelForms, CSRF & Admin • Custom AbstractUser & Middleware • Django REST Framework & Gunicorn

1. PROGRAM MASTERY OUTCOMES & ELIGIBILITY

End-to-End Engineering Outcomes:

1. Architect clean modular Django 5 projects following MVT boundaries, namespaced URLs, Function/Class-Based Views, and early Custom User Models (AUTH_USER_MODEL)
2. Eliminate ORM N+1 query storms using select_related() and prefetch_related(), prevent lost-update race conditions with F() expressions and transaction.atomic(), and manage zero-downtime migrations
3. Secure applications with CSRF tokens, ModelForm explicit field allowlists, PBKDF2/Argon2 password hashing, DRF ModelSerializer/JWT APIs, and manage.py check --deploy hardening
4. Clear 386 Timed Mock Exams (17,500 Django MCQs) and 350 Spoken Voice + MockAttempt AI Python Django Technical Interviews
5. Custom `AbstractUser` authentication, domain-aligned Django apps, and service-layer business encapsulation.
6. Crack LeetCode and HackerRank problems in Python 3.12 with live trainer mentorship on algorithmic/query patterns, edge cases, and optimal complexity.
7. Django REST Framework ViewSets, ModelSerializers, SimpleJWT auth, filtering, cursor pagination, and throttling.
8. ORM query optimization using `select_related`, `prefetch_related`, `F()`/`Q()` expressions, and DB indexes.

Eligibility & Prerequisites: Basic Python classes, functions & dictionaries — teaches MVT, HTTP, SQL ORM, and Django 5 step-by-step from scratch • Any laptop/PC running Python 3.10+ (`venv`), pip, and VS Code / PyCharm • No prior Django industry experience required — Module 1 starts from environment setup, core mental models, and plain-English fundamentals. • 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 ZERO-TO-EXPERT MODULE, TOPIC & SUBTOPIC SYLLABUS (16 MODULES • 53 LIVE TOPICS • 424+ SUBTOPICS)

Progressive Zero-to-Expert Mastery Stages: Modules 1–4: Stage 1: Day-1 Foundations & Core Mental Models (15 Topics / 120 Subtopics) • Modules 5–8: Stage 2: Applied Engineering, Data & API Workflows (15 Topics / 120 Subtopics) • Modules 9–12: Stage 3: Internals, Concurrency, Security & Performance (11

Topics / 88 Subtopics) • Modules 13–16: Stage 4: Production Architecture, Testing, CI/CD & Expert Mastery (12 Topics / 96 Subtopics)

MODULE 1 OF 16 • STAGE 1 • 11 LIVE + LAB HOURS • 4 TOPICS (32 SUBTOPICS) • 1 CAPSTONE PROJECT (100 MARKS)

Module 1: Django 5 Architecture, MVT Design Pattern, `venv` Setup & Project vs App Structure

Understand Django 5's "Batteries-Included for Perfectionists with Deadlines" philosophy, contrast MVT (`Model`-`View`-`Template`) against classic MVC, set up isolated `venv` tooling, and structure modular projects (`config/` + domain `apps/`).

Objectives: ✓ Django 5 Overview — Batteries-Included Philosophy, DRY Principle, WSGI/ASGI & Built-In Security — Why Instagram, Pinterest, Disqus, and Mozilla scaled on Django: built-in ORM, Migrations, Admin, Auth, Forms, Caching, and out-of-the-box defenses against **SQL Injection, XSS, CSRF, Clickjacking, and Host Header Poisoning** — plus dual `wsgi.py` & `asgi.py` entrypoints!. • ✓ Django MVT (`Model` - `View` - `Template`) Architecture vs Classic MVC & Full Request Lifecycle — Mapping MVT to MVC accurately: Django's **`Model`** = Data + Business Domain, Django's **`Template`** = Presentation Layer (MVC's "View"!), and Django's **`View`** = Request Handler deciding *which* data is shown (MVC's "Controller"!), while Django's URL Dispatcher acts as the Front Controller!. • ✓ Virtual Environments (`venv` / `uv`), Installing Django 5, `django-admin` & `manage.py` CLI — Isolating per-project dependencies (`python -m venv .venv`), pinning deterministic versions (`pip freeze` / `requirements.lock` / `uv.lock`), and understanding why `python manage.py` wraps `django-admin` by pre-setting `DJANGO_SETTINGS_MODULE`!. • ✓ Django Project vs Reusable Apps (`startapp`) — `INSTALLED_APPS`, `AppConfig` & `DEFAULT_AUTO_FIELD` — A **Project** is the whole site configuration (`settings.py`, root `urls.py`, `asgi.py`), while an **App** (`python manage.py startapp catalog`) is a focused domain module (`models.py`, `views.py`, `admin.py`, `migrations/`, `tests.py`) registered in `INSTALLED_APPS`!. • ✓ Build, verify, and ship Module Capstone: 12-Factor Django 5 Multi-App Workspace (`catalog`, `accounts`, `assessments`) & Env Settings.

1.1 Django 5 Overview — Batteries-Included Philosophy, DRY Principle, WSGI/ASGI & Built-In Security [50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Subtopics Covered: Batteries-Included web framework & DRY single source of truth • Core Syntax, CLI & Production APIs: django.get_version() | config/wsgi.py (Gunicorn) & config/asgi.py (Uvicorn) • Deep-Dive Mechanics: Dual WSGI & ASGI support & Built-in CSRF/XSS/SQLi defenses • Advanced Patterns: Async ORM aget() & afilter() & LTS release cadence • Production Failure Mode & Pitfall Guard • Hands-On Live Coding Lab: Inspecting Environment-Driven Django 5 Security Settings Matrix • Idiomatic DJNG Implementation: select_related/prefetch_related QuerySets, DRF ViewSets & atomic transactions • Django 5 ORM/ASGI Pipeline, PostgreSQL & Celery Workers Execution & Memory Lifecycle for Django 5 Overview

Purpose: To understand and apply Django 5 Overview — Batteries-Included Philosophy, DRY Principle, WSGI/ASGI & Built-In Security in real Django systems — specifically: Why Instagr...

What Topic Is Next → Next Topic 1.2: Django MVT (`Model` - `View` - `Template`) Architecture vs Classic MVC & Full Request Lifecycle

1.2 Django MVT (`Model` - `View` - `Template`) Architecture vs Classic MVC & Full Request Lifecycle [50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Subtopics Covered: MVT vs MVC mapping! & Django View = MVC Controller • Core Syntax, CLI & Production APIs: URLconf (urls.py) -> View (views.py) -> Model/ORM (models.py) -> Template (templates/*.html) • Deep-Dive Mechanics: Django Template = MVC View & Thin Views + Domain Services • Advanced Patterns: URLconf Front Controller & End-to-end request/response onion • Production Failure Mode & Pitfall Guard • Hands-On Live Coding Lab: Clean Domain Service + Thin View MVT Separation Pattern • Idiomatic DJNG Implementation: select_related/prefetch_related QuerySets, DRF ViewSets & atomic transactions • Django 5 ORM/ASGI Pipeline, PostgreSQL & Celery Workers Execution & Memory Lifecycle for Django MVT (`Model` - `View` - `Template`) Architect...

Purpose: To understand and apply Django MVT (`Model` - `View` - `Template`) Architecture vs Classic MVC & Full Request Lifecycle in real Django systems — specifically: Mapping MVT...

What Topic Is Next → Next Topic 1.3: Virtual Environments (`venv` / `uv`), Installing Django 5, `django-admin` & `manage.py` CLI

1.3 Virtual Environments (`venv` / `uv`), Installing Django 5, `django-admin` & `manage.py` CLI [50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Subtopics Covered: Isolated python -m venv .venv & Django-admin startproject config . • Core Syntax, CLI & Production APIs: python -m venv .venv && pip install "Django>=5.1,<5.2" && django-admin startproject config . • Deep-Dive Mechanics: Manage.py sets DJANGO_SETTINGS_MODULE & Pinned requirements.txt / uv.lock • Advanced Patterns: Manage.py check system framework & Custom management commands • Production Failure Mode & Pitfall Guard • Hands-On Live Coding Lab: How `manage.py` Bootstraps `DJANGO_SETTINGS_MODULE` Cleanly • Idiomatic DJNG Implementation: select_related/prefetch_related QuerySets, DRF ViewSets & atomic transactions • Django 5 ORM/ASGI Pipeline, PostgreSQL & Celery Workers Execution & Memory Lifecycle for Virtual Environments (`venv` / `uv`)

Purpose: To understand and apply Virtual Environments (`venv` / `uv`), Installing Django 5, `django-admin` & `manage.py` CLI in real Django systems — specifically: Isolating per-p...

What Topic Is Next → Next Topic 1.4: Django Project vs Reusable Apps (`startapp`) — `INSTALLED_APPS`, `AppConfig` & `DEFAULT_AUTO_FIELD`

1.4 Django Project vs Reusable Apps (`startapp`) — `INSTALLED_APPS`, `AppConfig` & `DEFAULT_AUTO_FIELD` [50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Subtopics Covered: Project (site config) vs App (domain) & Register AppConfig in INSTALLED_APPS! • Core Syntax, CLI & Production APIs: python manage.py startapp catalog -> INSTALLED_APPS = [..., "catalog.apps.CatalogConfig"] • Deep-Dive Mechanics: BigAutoField 64-bit primary keys & Templates/<app_name>/... namespacing! • Advanced Patterns: AppConfig.ready() signal registration & Low coupling between sibling apps • Production Failure Mode & Pitfall Guard • Hands-On Live Coding Lab: Validating Template Namespacing Path Convention Across `INSTALLED_APPS` • Idiomatic DJNG Implementation: select_related/prefetch_related QuerySets, DRF ViewSets & atomic transactions • Django 5 ORM/ASGI Pipeline, PostgreSQL & Celery Workers Execution & Memory Lifecycle for Django Project vs Reusable Apps (`startapp`)

Purpose: To understand and apply Django Project vs Reusable Apps (`startapp`) — `INSTALLED_APPS`, `AppConfig` & `DEFAULT_AUTO_FIELD` in real Django systems — specifically: A **Pro...

What Topic Is Next → Module 1 Capstone (12-Factor Django 5 Multi-App Workspace (`catalog`, `accounts`, `assessments`) & Env Settings) → Next Topic 2.1: Django URL Dispatcher — `urlpatterns`, `path()` Converters (`<int>`, `<slug>`, `<uuid>`), `include()` & Namespaces

Module 1 Hands-On Capstone Project (100 Marks): 12-Factor Django 5 Multi-App Workspace (`catalog`, `accounts`, `assessments`) & Env Settings — Bootstrap a clean Django 5.1 repository (`django-admin startproject config .`) reading `SECRET_KEY`, `DEBUG`, and `ALLOWED_HOSTS` strictly from

environment variables, registering `AppConfig`...

MODULE 2 OF 16 • STAGE 1 • 12 LIVE + LAB HOURS • 4 TOPICS (32 SUBTOPICS) • 1 CAPSTONE PROJECT (100 MARKS)

Module 2: URL Dispatcher (`urls.py`), Function & Class-Based Views, Shortcuts & Safe Redirects

Route clean REST/HTML URLs with `path()`, converters (`<int:pk>`, `<slug:slug>`), and `app_name` namespaces, compare Function-Based Views (`FBV`) against Generic Class-Based Views (`CBV`), and use `get_object_or_404` & `redirect` safely.

Objectives: ✓ Django URL Dispatcher — `urlpatterns`, `path()` Converters (`<int>`, `<slug>`, `<uuid>`), `include()` & Namespaces — Top-to-bottom sequential matching in `urlpatterns`, built-in path converters (`int`, `str`, `slug`, `uuid`, `path`), delegating sub-trees via `path("courses/", include("catalog.urls"))`, `app_name` URL namespaces, and `reverse()` / `reverse_lazy()`. • ✓ Function-Based Views (`FBV`) — `HttpRequest`, `HttpResponse`, `JsonResponse`, Decorators & `async def` Views — Writing explicit Python view functions `def detail(request: HttpRequest, slug: str) -> HttpResponse:`, HTTP verb guards (`@require_GET`, `@require_POST`, `@require_http_methods`), returning `JsonResponse({"items": [...]})`, and Django 5 native `async def` coroutine views!. • ✓ Class-Based Generic Views (`CBV`) — `ListView`, `DetailView`, `CreateView`, Mixins & MRO Order — Eliminating repetitive CRUD boilerplate via `View`, `TemplateView`, `ListView` (`paginate_by`), `DetailView`, `CreateView`, `UpdateView`, and `DeleteView`, overriding `get_queryset()` / `get_context_data()`, and `LoginRequiredMixin`!. • ✓ Django Shortcuts — `render()`, `get_object_or_404()`, `redirect()` & Open-Redirect Protection — Clean view ergonomics: `render(request, tpl, ctx, status=200)`, `get_object_or_404(Course.objects.filter(is_published=True), slug=slug)` (converting `DoesNotExist` into `Http404`!), `redirect("catalog:detail", slug=slug)`, and `url_has_allowed_host_and_scheme()`. • ✓ Build, verify, and ship Module Capstone: Namespaced Course Catalog (`catalog:list`, `catalog:detail`) with Paginated `ListView` & JSON Endpoint.

2.1 Django URL Dispatcher — `urlpatterns`, `path()` Converters (`<int>`, `<slug>`, `<uuid>`), `include()` & Namespaces [50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Subtopics Covered: Top-to-bottom urlpatterns order! & <int:pk>, <slug:slug>, <uuid:id> • Core Syntax, CLI & Production APIs: app_name = "catalog"; urlpatterns = [path("<slug:slug>/", views.detail, name="detail")] -> r... • Deep-Dive Mechanics: Include() & app_name namespaces & Reverse() vs reverse_lazy() in CBVs! • Advanced Patterns: APPEND_SLASH canonical URLs & Register_converter custom converter • Production Failure Mode & Pitfall Guard • Hands-On Live Coding Lab: Simulating Top-to-Bottom `urlpatterns` Specificity Matching • Idiomatic DJNG Implementation: select_related/prefetch_related QuerySets, DRF ViewSets & atomic transactions • Django 5 ORM/ASGI Pipeline, PostgreSQL & Celery Workers Execution & Memory Lifecycle for Django URL Dispatcher

Purpose: To understand and apply Django URL Dispatcher — `urlpatterns`, `path()` Converters (`<int>`, `<slug>`, `<uuid>`), `include()` & Namespaces in real Django systems — specif...

What Topic Is Next → Next Topic 2.2: Function-Based Views (`FBV`) — `HttpRequest`, `HttpResponse`, `JsonResponse`, Decorators & `async def` Views

2.2 Function-Based Views (`FBV`) — `HttpRequest`, `HttpResponse`, `JsonResponse`, Decorators & `async def` Views [50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Subtopics Covered: Explicit request: HttpRequest first arg & @require_GET & @require_POST guards • Core Syntax, CLI & Production APIs: @require_GET def api_pricing(request): return JsonResponse({"code": "DJNG", "offer_inr": 999... • Deep-Dive Mechanics: JsonResponse root dict convention & Async def views + native aget()/acount() • Advanced Patterns: Never block ASGI event loop & Clean early-return readability • Production Failure Mode & Pitfall Guard • Hands-On Live Coding Lab: Building a Safe Envelope `JsonResponse` Payload Structure • Idiomatic DJNG Implementation: select_related/prefetch_related QuerySets, DRF ViewSets & atomic transactions • Django 5 ORM/ASGI Pipeline, PostgreSQL & Celery Workers Execution & Memory Lifecycle for Function-Based Views (`FBV`)

Purpose: To understand and apply Function-Based Views (`FBV`) — `HttpRequest`, `HttpResponse`, `JsonResponse`, Decorators & `async def` Views in real Django systems — specifically...

What Topic Is Next → Next Topic 2.3: Class-Based Generic Views (`CBV`) — `ListView`, `DetailView`, `CreateView`, Mixins & MRO Order

2.3 Class-Based Generic Views (`CBV`) — `ListView`, `DetailView`, `CreateView`, Mixins & MRO Order [50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Subtopics Covered: LoginRequiredMixin MUST be leftmost (MRO)! & View.as_view() per-request instance • Core Syntax, CLI & Production APIs: class CourseListView(LoginRequiredMixin, ListView): model = Course; paginate_by = 12 • Deep-Dive Mechanics: Override get_queryset() against IDOR! & Super().get_context_data(**kwargs) • Advanced Patterns: ListView built-in paginate_by & Reverse_lazy() for success_url • Production Failure Mode & Pitfall Guard • Hands-On Live Coding Lab: Verifying Python MRO Leftmost Mixin Dispatch Order (`LoginRequiredMixin` First) • Idiomatic DJNG Implementation: select_related/prefetch_related QuerySets, DRF ViewSets & atomic transactions • Django 5 ORM/ASGI Pipeline, PostgreSQL & Celery Workers Execution & Memory Lifecycle for Class-Based Generic Views (`CBV`)

Purpose: To understand and apply Class-Based Generic Views (`CBV`) — `ListView`, `DetailView`, `CreateView`, Mixins & MRO Order in real Django systems — specifically: Eliminating...

What Topic Is Next → Next Topic 2.4: Django Shortcuts — `render()`, `get_object_or_404()`, `redirect()` & Open-Redirect Protection

2.4 Django Shortcuts — `render()`, `get_object_or_404()`, `redirect()` & Open-Redirect Protection [50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Subtopics Covered: Get_object_or_404(QuerySet, **lookup) & DoesNotExist 500 vs Http404 404! • Core Syntax, CLI & Production APIs: course = get_object_or_404(Course, slug=slug, is_published=True); return render(request, "ca... • Deep-Dive Mechanics: Redirect() accepts view name or Model & Model.get_absolute_url() integration • Advanced Patterns: Url_has_allowed_host_and_scheme()! & Permanent=True (301) vs default (302) • Production Failure Mode & Pitfall Guard • Hands-On Live Coding Lab: Simulating `get_object_or_404` Behavior vs Unhandled `DoesNotExist` • Idiomatic DJNG Implementation: select_related/prefetch_related QuerySets, DRF ViewSets & atomic transactions • Django 5 ORM/ASGI Pipeline, PostgreSQL & Celery Workers Execution & Memory Lifecycle for Django Shortcuts

Purpose: To understand and apply Django Shortcuts — `render()`, `get_object_or_404()`, `redirect()` & Open-Redirect Protection in real Django systems — specifically: Clean view er...

What Topic Is Next → Module 2 Capstone (Namespaced Course Catalog (`catalog:list`, `catalog:detail`) with Paginated `ListView` & JSON Endpoint) → Next Topic 3.1: Django Template Language (`DTL`) — Inheritance, Tags, Filters, `json_script` & Auto-Escaping

Module 2 Hands-On Capstone Project (100 Marks): Namespaced Course Catalog (`catalog:list`, `catalog:detail`) with Paginated `ListView` & JSON Endpoint — Build an app-namespaced `catalog/urls.py` (`app_name = "catalog"`) serving a paginated `CourseListView(ListView)` (`paginate_by = 12`), a slug-resolved `CourseDetailView` (`get_object_or_404...

MODULE 3 OF 16 • STAGE 1 • 10 LIVE + LAB HOURS • 3 TOPICS (24 SUBTOPICS) • 1 CAPSTONE PROJECT (100 MARKS)

Module 3: Django Template Language (`DTL`), Static Assets (`collectstatic` / `WhiteNoise`) & Responsive UI

Build fast XSS-safe HTML interfaces with Django Template Language (`DTL`), serve fingerprinted immutable CSS/JS bundles in production via `collectstatic` + `ManifestStaticFilesStorage` (`WhiteNoise` / S3 CDN), and compose reusable UI components.

Objectives: ✓ Django Template Language (`DTL`) — Inheritance, Tags, Filters, `json_script` & Auto-Escaping — Intentional logic-light DTL design (no arbitrary Python calls inside `{{ }}`!), layout inheritance (`{% extends "base.html" %}`), dot-lookup order (`dict[k]` -> `attr` -> `method()` -> `list[i]`), auto-escaping, and **`{{ value|json_script:"data-id" }}`**!. • ✓ Static Files in Production — `STATIC_URL`, `STATIC_ROOT`, `collectstatic` & `WhiteNoise` Caching — How `{% load static %}` + `{% static "css/app.css" %}` works, why `STATICFILES_DIRS` (extra source dirs) must NEVER equal `STATIC_ROOT` (compiled output target of `python manage.py collectstatic --noinput`!), and immutable MD5/SHA hash fingerprinting via `WhiteNoise` / CloudFront!. • ✓ Component Partials (`{% include ... only %}`), Custom `@register.simple_tag` & Responsive Form Styling — Modularizing Bootstrap 5 / Tailwind UIs with isolated `{% include "components/_badge.html" with label="Live" only %}`, building custom `templatetags/` (`@register.simple_tag`, `@register.filter`), and styling Django forms cleanly (`django-crispy-forms` / custom widget attrs). • ✓ Build, verify, and ship Module Capstone: Fingerprinted Production Static Pipeline (`ManifestStaticFilesStorage`) & DTL Course Shell.

3.1 Django Template Language (`DTL`) — Inheritance, Tags, Filters, `json_script` & Auto-Escaping

[50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Subtopics Covered: {% extends %} & {% block %} & {{ data|json_script:"id" }} XSS safety! • Core Syntax, CLI & Production APIs: {% extends "base.html" %} | {% url "catalog:detail" course.slug %} | {{ data|json_script:"in... • Deep-Dive Mechanics: Never use |safe / mark_safe on user text & Dot-lookup dict -> attr -> method • Advanced Patterns: Cached.Loader automatic when DEBUG=False & {% empty %} clause inside {% for %} • Production Failure Mode & Pitfall Guard • Hands-On Live Coding Lab: Safe `<script type="application/json">` Escaping Matching Django's `json_script` Filter • Idiomatic DJNG Implementation: select_related/prefetch_related QuerySets, DRF ViewSets & atomic transactions • Django 5 ORM/ASGI Pipeline, PostgreSQL & Celery Workers Execution & Memory Lifecycle for Django Template Language (`DTL`)

Purpose: To understand and apply Django Template Language (`DTL`) — Inheritance, Tags, Filters, `json_script` & Auto-Escaping in real Django systems — specifically: Intentional lo...

What Topic Is Next → Next Topic 3.2: Static Files in Production — `STATIC_URL`, `STATIC_ROOT`, `collectstatic` & `WhiteNoise` Caching

3.2 Static Files in Production — `STATIC_URL`, `STATIC_ROOT`, `collectstatic` & `WhiteNoise` Caching

[50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Subtopics Covered: Collectstatic --noinput CI step! & STATIC_ROOT vs STATICFILES_DIRS • Core Syntax, CLI & Production APIs: python manage.py collectstatic --noinput + WhiteNoise CompressedManifestStaticFilesStorage • Deep-Dive Mechanics: STATIC_ROOT vs MEDIA_ROOT separation & WhiteNoise Manifest hash cache-busting • Advanced Patterns: {% load static %} & {% static %} & Cache-Control immutable 1-year headers • Production Failure Mode & Pitfall Guard • Hands-On Live Coding Lab: Simulating Content-Hash Asset Fingerprinting (`ManifestStaticFilesStorage`) • Idiomatic DJNG Implementation: select_related/prefetch_related QuerySets, DRF ViewSets & atomic transactions • Django 5 ORM/ASGI Pipeline, PostgreSQL & Celery Workers Execution & Memory Lifecycle for Static Files in Production

Purpose: To understand and apply Static Files in Production — `STATIC_URL`, `STATIC_ROOT`, `collectstatic` & `WhiteNoise` Caching in real Django systems — specifically: How `{% lo...

What Topic Is Next → Next Topic 3.3: Component Partials (`{% include ... only %}`), Custom `@register.simple_tag` & Responsive Form Styling

3.3 Component Partials (`{% include ... only %}`), Custom `@register.simple_tag` & Responsive Form Styling

[50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Subtopics Covered: {% include ... with ... only %} isolation & @register.simple_tag & @register.filter • Core Syntax, CLI & Production APIs: {% include "catalog/_card.html" with course=course only %} | @register.simple_tag • Deep-Dive Mechanics: No hidden SQL inside loop tags! & Bootstrap 5 / crispy-forms error states • Advanced Patterns: Accessible aria-invalid form feedback & Context_processors global nav metadata • Production Failure Mode & Pitfall Guard • Hands-On Live Coding Lab: Formatting Indian Rupee Badges Cleanly in a Custom Template Filter Helper • Idiomatic DJNG Implementation: select_related/prefetch_related QuerySets, DRF ViewSets & atomic transactions • Django 5 ORM/ASGI Pipeline, PostgreSQL & Celery Workers Execution & Memory Lifecycle for Component Partials (`{% include ... only %}`)

Purpose: To understand and apply Component Partials (`{% include ... only %}`), Custom `@register.simple_tag` & Responsive Form Styling in real Django systems — specifically: Modu...

What Topic Is Next → Module 3 Capstone (Fingerprinted Production Static Pipeline (`ManifestStaticFilesStorage`) & DTL Course Shell) → Next Topic 4.1: Django Models (`models.Model`) — Field Types, `TextChoices`, `Meta.indexes` & `CheckConstraint`

Module 3 Hands-On Capstone Project (100 Marks): Fingerprinted Production Static Pipeline (`ManifestStaticFilesStorage`) & DTL Course Shell — Build a responsive DTL base layout (`base.html`), reusable course card partial (`{% include "catalog/_card.html" with course=c only %}`), safe JS data passing via `{{ payload|json_script:"co...

MODULE 4 OF 16 • STAGE 1 • 13 LIVE + LAB HOURS • 4 TOPICS (32 SUBTOPICS) • 1 CAPSTONE PROJECT (100 MARKS)

Module 4: Models, Relational Fields, `on_delete` Integrity, Migrations & Database Pooling

Design normalized database schemas with `models.Model`, `Meta.constraints` (`CheckConstraint`, `UniqueConstraint`), `ForeignKey` / `ManyToManyField` (`through`), safe `on_delete=PROTECT` rules, and zero-downtime `makemigrations` / `migrate` workflows.

Objectives: ✓ Django Models (`models.Model`) — Field Types, `TextChoices`, `Meta.indexes` & `CheckConstraint` — Mapping Python classes to SQL tables: `CharField`, `SlugField(unique=True)`, `DecimalField` (exact base-10 decimal!), Django 5 **`GeneratedField`** (database-computed columns!) & `db_default`, `models.TextChoices` enums, `Meta.indexes`, and `Meta.constraints`. • ✓ Relational Fields — `ForeignKey` (`related_name`), `ManyToManyField` (`through=`)

& `OneToOneField` — Modeling `Many-to-One` (`ForeignKey` — Django automatically creates a B-Tree index on `course_id`!), `Many-to-Many` (`ManyToManyField` with explicit `through="Enrollment"` when extra columns like `enrolled_at` or `role` are needed!), and `OneToOneField`. • ✓ `on_delete` Referential Rules — `CASCADE` vs `PROTECT` vs `RESTRICT` vs `SET_NULL` — Why Django 2.0+ made `on_delete` a **mandatory positional argument** on `ForeignKey` & `OneToOneField`: contrasting destructive `models.CASCADE`, safe financial/audit **`models.PROTECT`** / `models.RESTRICT`, `models.SET_NULL(null=True)`, and soft-deletion patterns. • ✓ Django Migrations & Production DB Connectivity — `makemigrations`, `RunPython` & `CONN_MAX_AGE` — Generating declarative schema diffs (`makemigrations`), inspecting exact SQL (`sqlmigrate`), failing CI if model changes lack migrations (`makemigrations --check --dry-run`), writing historical `apps.get_model()` `RunPython` data migrations, and persistent DB connections (`CONN_MAX_AGE = 600` / `psycopg` pool!). • ✓ Build, verify, and ship Module Capstone: Relational Academy Schema (`Course`, `Batch`, `Enrollment` `through` Model) with `CheckConstraint`s.

4.1 Django Models (`models.Model`) — Field Types, `TextChoices`, `Meta.indexes` & `CheckConstraint` [50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Subtopics Covered: DecimalField over FloatField for money & Meta.constraints CheckConstraint! • Core Syntax, CLI & Production APIs: class Course(models.Model): code = models.CharField(max_length=16, unique=True); fee = model... • Deep-Dive Mechanics: Validators only run in full_clean()! & Avoid null=True on plain CharField • Advanced Patterns: Models.TextChoices typed enums & Django 5 db_default & GeneratedField • Production Failure Mode & Pitfall Guard • Hands-On Live Coding Lab: Validating `Decimal` Exact Money Arithmetic vs Binary `float` Drift • Idiomatic DJNG Implementation: select_related/prefetch_related QuerySets, DRF ViewSets & atomic transactions • Django 5 ORM/ASGI Pipeline, PostgreSQL & Celery Workers Execution & Memory Lifecycle for Django Models (`models.Model`)

Purpose: To understand and apply Django Models (`models.Model`) — Field Types, `TextChoices`, `Meta.indexes` & `CheckConstraint` in real Django systems — specifically: Mapping Pyt...

What Topic Is Next → Next Topic 4.2: Relational Fields — `ForeignKey` (`related_name`), `ManyToManyField` (`through=`) & `OneToOneField`

4.2 Relational Fields — `ForeignKey` (`related_name`), `ManyToManyField` (`through=`) & `OneToOneField` [50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Subtopics Covered: Batch.course_id (0 extra SQL) vs .course.id! & Explicit related_name readability • Core Syntax, CLI & Production APIs: course = models.ForeignKey(Course, on_delete=models.PROTECT, related_name="batches") • Deep-Dive Mechanics: ManyToManyField(through="Enrollment") & ForeignKey auto db_index=True • Advanced Patterns: OneToOneField vs AbstractUser & Symmetrical=False self-referential M2M • Production Failure Mode & Pitfall Guard • Hands-On Live Coding Lab: Zero-Query `course_id` Attribute Access Pattern on Model Instances • Idiomatic DJNG Implementation: select_related/prefetch_related QuerySets, DRF ViewSets & atomic transactions • Django 5 ORM/ASGI Pipeline, PostgreSQL & Celery Workers Execution & Memory Lifecycle for Relational Fields

Purpose: To understand and apply Relational Fields — `ForeignKey` (`related_name`), `ManyToManyField` (`through=`) & `OneToOneField` in real Django systems — specifically: Modelin...

What Topic Is Next → Next Topic 4.3: `on_delete` Referential Rules — `CASCADE` vs `PROTECT` vs `RESTRICT` vs `SET_NULL`

4.3 `on_delete` Referential Rules — `CASCADE` vs `PROTECT` vs `RESTRICT` vs `SET_NULL` [50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Subtopics Covered: PROTECT blocks accidental parent wipes! & CASCADE dangerous on Orders/Invoices! • Core Syntax, CLI & Production APIs: user = models.ForeignKey(settings.AUTH_USER_MODEL, on_delete=models.PROTECT) • Deep-Dive Mechanics: RESTRICT vs PROTECT subtleties & SET_NULL requires null=True • Advanced Patterns: Django Python-emulated cascade signals & Is_archived soft-delete alternative • Production Failure Mode & Pitfall Guard • Hands-On Live Coding Lab: Simulating `on_delete=PROTECT` Guarding Paid Invoices Against Deletion • Idiomatic DJNG Implementation: select_related/prefetch_related QuerySets, DRF ViewSets & atomic transactions • Django 5 ORM/ASGI Pipeline, PostgreSQL & Celery Workers Execution & Memory Lifecycle for `on_delete` Referential Rules

Purpose: To understand and apply `on_delete` Referential Rules — `CASCADE` vs `PROTECT` vs `RESTRICT` vs `SET_NULL` in real Django systems — specifically: Why Django 2.0+ made `on...

What Topic Is Next → Next Topic 4.4: Django Migrations & Production DB Connectivity — `makemigrations`, `RunPython` & `CONN_MAX_AGE`

4.4 Django Migrations & Production DB Connectivity — `makemigrations`, `RunPython` & `CONN_MAX_AGE` [50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Subtopics Covered: Apps.get_model() inside RunPython! & Makemigrations --check in CI! • Core Syntax, CLI & Production APIs: python manage.py makemigrations --check && python manage.py migrate --noinput | CONN_MAX_AGE... • Deep-Dive Mechanics: Sqlmigrate DDL inspection & CONN_MAX_AGE & CONN_HEALTH_CHECKS! • Advanced Patterns: Never edit applied prod migrations & 3-step nullable -> backfill -> NOT NULL • Production Failure Mode & Pitfall Guard • Hands-On Live Coding Lab: Safe Historical `RunPython` Data Migration Pattern Using `apps.get_model` • Idiomatic DJNG Implementation: select_related/prefetch_related QuerySets, DRF ViewSets & atomic transactions • Django 5 ORM/ASGI Pipeline, PostgreSQL & Celery Workers Execution & Memory Lifecycle for Django Migrations

Purpose: To understand and apply Django Migrations & Production DB Connectivity — `makemigrations`, `RunPython` & `CONN_MAX_AGE` in real Django systems — specifically: Generating...

What Topic Is Next → Module 4 Capstone (Relational Academy Schema (`Course`, `Batch`, `Enrollment` `through` Model) with `CheckConstraint`s) → Next Topic 5.1: Django ORM `QuerySet` Mastery — Lazy Chaining, `exists()`, `values_list()`, `annotate()` & `bulk_create()`

Module 4 Hands-On Capstone Project (100 Marks): Relational Academy Schema (`Course`, `Batch`, `Enrollment` `through` Model) with `CheckConstraint`s — Model `Course` -> `Batch` (`on_delete=models.PROTECT`) and `Batch` <-> `User` via an explicit `Enrollment` `through` table storing `enrolled_at`, `fee_paid` (`DecimalField`), and a `UniqueCo...

MODULE 5 OF 16 • STAGE 2 • 13 LIVE + LAB HOURS • 3 TOPICS (24 SUBTOPICS) • 1 CAPSTONE PROJECT (100 MARKS)

Module 5: High-Performance Django ORM — Lazy `QuerySet`s, `select_related`/`prefetch_related` & `F()`/`Q()`

Write blazing-fast ORM queries: master lazy `QuerySet` evaluation & caching, crush `N+1` bottlenecks with `select_related()` and `prefetch_related(Prefetch(...))`, and execute race-free atomic SQL arithmetic using `F()`, `Q()`, `annotate()`, and `transaction.atomic()`.

Objectives: ✓ Django ORM `QuerySet` Mastery — Lazy Chaining, `exists()`, `values_list()`, `annotate()` & `bulk_create()` — Composable `filter()`, `exclude()`, field lookups (`__icontains`, `__gte`, `__in`, `__date`), pushing math to SQL via `aggregate()` & `annotate(Count(...))`, memory-safe `.iterator(chunk_size=2000)`, `only()`/`defer()`, and `bulk_create()` / `bulk_update()`. • ✓ `select_related()` vs `prefetch_related()` — Crushing the Django ORM

`N+1` Query Bottleneck — The #1 Django interview & production performance topic: **`select_related("author", "category")`** performs a single SQL `LEFT/INNER JOIN` for forward `ForeignKey` & `OneToOneField`; **`prefetch_related("tags", Prefetch(...))`** runs a separate `WHERE id IN (...)` query and joins in Python for `ManyToMany` & reverse `ForeignKey`s!. • ✓ Atomic SQL with `F()` Expressions, Complex `Q()` Boolean Filters & `transaction.atomic()` — Referencing database columns on the right-hand side of queries/updates with **`F("views") + 1`** (`filter(mrp__gt=F("discount_price"))`), combining `OR` / `NOT` conditions with **`Q(code="DJNG") | ~Q(is_archived=True)`**, and coordinating multi-step writes with `transaction.atomic()` + `on_commit()`. • ✓ Build, verify, and ship Module Capstone: 2-Query Analytics Leaderboard (`annotate` + `select_related` + `Prefetch`) & Atomic Seat Booking Engine.

5.1 Django ORM `QuerySet` Mastery — Lazy Chaining, `exists()`, `values_list()`, `annotate()` & `bulk_create()` [50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Subtopics Covered: Lazy QuerySet evaluation & cache & .exists() vs bool(qs) & .count() vs len() • Core Syntax, CLI & Production APIs: Course.objects.filter(is_published=True).annotate(module_total=Count("modules")).values("cod... • Deep-Dive Mechanics: Annotate() per-row vs aggregate() summary & Values_list("id", flat=True) • Advanced Patterns: .iterator(chunk_size=2000) OOM safety & Bulk_create(batch_size=1000) • Production Failure Mode & Pitfall Guard • Hands-On Live Coding Lab: Demonstrating Lazy Chaining & Single-Pass Aggregation Semantics • Idiomatic DJNG Implementation: select_related/prefetch_related QuerySets, DRF ViewSets & atomic transactions • Django 5 ORM/ASGI Pipeline, PostgreSQL & Celery Workers Execution & Memory Lifecycle for Django ORM `QuerySet` Mastery

Purpose: To understand and apply Django ORM `QuerySet` Mastery — Lazy Chaining, `exists()`, `values_list()`, `annotate()` & `bulk_create()` in real Django systems — specifically:...

What Topic Is Next → Next Topic 5.2: `select_related()` vs `prefetch_related()` — Crushing the Django ORM `N+1` Query Bottleneck

5.2 `select_related()` vs `prefetch_related()` — Crushing the Django ORM `N+1` Query Bottleneck [50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Subtopics Covered: Select_related = SQL JOIN (FK / 1-to-1) & Prefetch_related = SQL IN + Python join (M2M / rev FK) • Core Syntax, CLI & Production APIs: Topic.objects.select_related("module__course").prefetch_related(Prefetch("mock_tests", query... • Deep-Dive Mechanics: Prefetch(lookup, queryset=..., to_attr=...) & Never .filter() on prefetched manager in loop! • Advanced Patterns: Module__course multi-hop JOIN & Django-debug-toolbar & assertNumQueries • Production Failure Mode & Pitfall Guard • Hands-On Live Coding Lab: Comparing SQL Query Count With vs Without `select_related` + `prefetch_related` • Idiomatic DJNG Implementation: select_related/prefetch_related QuerySets, DRF ViewSets & atomic transactions • Django 5 ORM/ASGI Pipeline, PostgreSQL & Celery Workers Execution & Memory Lifecycle for `select_related()` vs `prefetch_related()`

Purpose: To understand and apply `select_related()` vs `prefetch_related()` — Crushing the Django ORM `N+1` Query Bottleneck in real Django systems — specifically: The #1 Django i...

What Topic Is Next → Next Topic 5.3: Atomic SQL with `F()` Expressions, Complex `Q()` Boolean Filters & `transaction.atomic()`

5.3 Atomic SQL with `F()` Expressions, Complex `Q()` Boolean Filters & `transaction.atomic()` [50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Subtopics Covered: F() atomic SQL column arithmetic! & Refresh_from_db() after instance F() save! • Core Syntax, CLI & Production APIs: Course.objects.filter(pk=pk).update(enroll_count=F("enroll_count") + 1) | Q(title__icontains... • Deep-Dive Mechanics: Q() | (OR), & (AND), ~ (NOT) filters & Transaction.atomic() & select_for_update() • Advanced Patterns: Transaction.on_commit() Celery safety! & Case / When conditional SQL expressions • Production Failure Mode & Pitfall Guard • Hands-On Live Coding Lab: Simulating Concurrent Lost-Update Bug vs Atomic `F()` Counter Increment • Idiomatic DJNG Implementation: select_related/prefetch_related QuerySets, DRF ViewSets & atomic transactions • Django 5 ORM/ASGI Pipeline, PostgreSQL & Celery Workers Execution & Memory Lifecycle for Atomic SQL with `F()` Expressions

Purpose: To understand and apply Atomic SQL with `F()` Expressions, Complex `Q()` Boolean Filters & `transaction.atomic()` in real Django systems — specifically: Referencing datab...

What Topic Is Next → Module 5 Capstone (2-Query Analytics Leaderboard (`annotate` + `select_related` + `Prefetch`) & Atomic Seat Booking Engine) → Next Topic 6.1: Django Admin Mastery — `ModelAdmin`, `list_select_related`, `autocomplete_fields`, `Inlines` & Actions

Module 5 Hands-On Capstone Project (100 Marks): 2-Query Analytics Leaderboard (`annotate` + `select_related` + `Prefetch`) & Atomic Seat Booking Engine — Optimize a slow 152-query cohort leaderboard down to **2 SQL queries** (`assertNumQueries(2)`) using `select_related("course")`, `Prefetch("topics", queryset=...)`, and `annotate(avg_score=A...

MODULE 6 OF 16 • STAGE 2 • 11 LIVE + LAB HOURS • 4 TOPICS (32 SUBTOPICS) • 1 CAPSTONE PROJECT (100 MARKS)

Module 6: Django Admin Customization, `Form`s, `ModelForm`s & Multi-Field Validation

Supercharge operations with a fast `N+1`-free `ModelAdmin`, build CSRF-protected `forms.Form` & `forms.ModelForm` classes with explicit field allowlists, and enforce field + cross-field `clean()` validation rules.

Objectives: ✓ Django Admin Mastery — `ModelAdmin`, `list_select_related`, `autocomplete_fields`, `Inlines` & Actions — Registering models with `@admin.register(Course)`, configuring `list_display`, `list_filter`, `search_fields`, `prepopulated_fields = {"slug": ("title",)}`, `TabularInline`, bulk `@admin.action`s, and preventing Admin `N+1` / dropdown hangs!. • ✓ Django `forms.Form` — Bound vs Unbound Lifecycle, `{% csrf_token %}`, Widgets & `cleaned_data` — Unbound (`EnrollForm()` on `GET`) vs Bound (`EnrollForm(request.POST)` on `POST`), how `form.is_valid()` normalizes Python types (`Decimal`, `date`, stripped `str`) into **`form.cleaned_data`**, HTML Widgets (`forms.TextInput`), and `CsrfViewMiddleware`. • ✓ `forms.ModelForm` — Explicit `Meta.fields` (Blocking Mass-Assignment!), `commit=False` & `save_m2m()` — Auto-generating form fields & model uniqueness checks directly from a `Model`, why `Meta.fields` MUST explicitly list safe editable columns, setting `obj.author = request.user` via **`form.save(commit=False)`**, and calling **`form.save_m2m()`** afterwards!. • ✓ Multi-Stage Form Validation — `clean_<field>()`, Cross-Field `clean()` & `ValidationError` Codes — Execution order during `form.is_valid()`: `to_python()` -> `validate()` -> `run_validators()` -> **`clean_<fieldname>()`** -> **`form.clean()`** (for cross-field invariants like `end_date > start_date` or `discount_price <= price`!) -> `Model.post_clean()`. • ✓ Build, verify, and ship Module Capstone: Optimized Operations `ModelAdmin` (`list_select_related` + `autocomplete_fields`) & `ModelForm` Studio.

6.1 Django Admin Mastery — `ModelAdmin`, `list_select_related`, `autocomplete_fields`, `Inlines` & Actions [50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Subtopics Covered: List_select_related prevents Admin N+1! & Autocomplete_fields on large ForeignKeys! • Core Syntax, CLI & Production APIs: @admin.register(Batch) class BatchAdmin(admin.ModelAdmin): list_display = ("code", "course")... • Deep-Dive Mechanics: TabularInline & StackedInline & Prepopulated_fields slug helper • Advanced Patterns: @admin.action bulk QuerySet update & Change default /admin/ URL in prod • Production Failure Mode & Pitfall Guard • Hands-On Live Coding Lab: Bulk Admin Action Publishing Selected Courses in 1 `QuerySet.update()` • Idiomatic DJNG Implementation: select_related/prefetch_related QuerySets, DRF ViewSets & atomic transactions • Django 5 ORM/ASGI Pipeline, PostgreSQL & Celery Workers Execution & Memory Lifecycle for Django Admin Mastery

Purpose: To understand and apply Django Admin Mastery — `ModelAdmin`, `list_select_related`, `autocomplete_fields`, `Inlines` & Actions in real Django systems — specifically: Regi...

What Topic Is Next → Next Topic 6.2: Django `forms.Form` — Bound vs Unbound Lifecycle, `{% csrf_token %}`, Widgets & `cleaned_data`

6.2 Django `forms.Form` — Bound vs Unbound Lifecycle, `{% csrf_token %}`, Widgets & `cleaned_data`

[50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Subtopics Covered: Always read form.cleaned_data! & Unbound (GET) vs Bound (POST) form • Core Syntax, CLI & Production APIs: form = ContactForm(request.POST or None); if request.method == "POST" and form.is_valid(): d... • Deep-Dive Mechanics: {% csrf_token %} & X-CSRFToken AJAX & Never abuse @csrf_exempt on web views • Advanced Patterns: Typed fields (DecimalField, DateField) & Post-Redirect-Get after is_valid() • Production Failure Mode & Pitfall Guard • Hands-On Live Coding Lab: Simulating `is_valid()` Normalization into Typed `cleaned_data` • Idiomatic DJNG Implementation: select_related/prefetch_related QuerySets, DRF ViewSets & atomic transactions • Django 5 ORM/ASGI Pipeline, PostgreSQL & Celery Workers Execution & Memory Lifecycle for Django `forms.Form`

Purpose: To understand and apply Django `forms.Form` — Bound vs Unbound Lifecycle, `{% csrf_token %}`, Widgets & `cleaned_data` in real Django systems — specifically: Unbound (`En...

What Topic Is Next → Next Topic 6.3: `forms.ModelForm` — Explicit `Meta.fields` (Blocking Mass-Assignment!), `commit=False` & `save_m2m()`

6.3 `forms.ModelForm` — Explicit `Meta.fields` (Blocking Mass-Assignment!), `commit=False` & `save_m2m()`

[50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Subtopics Covered: Explicit Meta.fields allowlist! & Never use fields = "__all__" publicly! • Core Syntax, CLI & Production APIs: obj = form.save(commit=False); obj.student = request.user; obj.save(); form.save_m2m() • Deep-Dive Mechanics: Form.save(commit=False) owner binding & Call form.save_m2m() after obj.save()! • Advanced Patterns: Instance=existing_obj for updates & Meta.widgets & labels customization • Production Failure Mode & Pitfall Guard • Hands-On Live Coding Lab: Demonstrating `commit=False` + `save_m2m()` Ordering Requirement • Idiomatic DJNG Implementation: select_related/prefetch_related QuerySets, DRF ViewSets & atomic transactions • Django 5 ORM/ASGI Pipeline, PostgreSQL & Celery Workers Execution & Memory Lifecycle for `forms.ModelForm`

Purpose: To understand and apply `forms.ModelForm` — Explicit `Meta.fields` (Blocking Mass-Assignment!), `commit=False` & `save_m2m()` in real Django systems — specifically: Auto-...

What Topic Is Next → Next Topic 6.4: Multi-Stage Form Validation — `clean_<field>()`, Cross-Field `clean()` & `ValidationError` Codes

6.4 Multi-Stage Form Validation — `clean_<field>()`, Cross-Field `clean()` & `ValidationError` Codes

[50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Subtopics Covered: Always return value in clean_<field>()! & Use .get() inside cross-field clean()! • Core Syntax, CLI & Production APIs: def clean_slug(self): ...; def clean(self): cleaned = super().clean(); if ...: self.add_erro... • Deep-Dive Mechanics: Self.add_error(field, ValidationError) & ValidationError symbolic code="..." • Advanced Patterns: Non_field_errors display in templates & RegexValidator & FileExtensionValidator • Production Failure Mode & Pitfall Guard • Hands-On Live Coding Lab: Safe Cross-Field Pricing Validation (`discount_price <= price`) Using `.get()` • Idiomatic DJNG Implementation: select_related/prefetch_related QuerySets, DRF ViewSets & atomic transactions • Django 5 ORM/ASGI Pipeline, PostgreSQL & Celery Workers Execution & Memory Lifecycle for Multi-Stage Form Validation

Purpose: To understand and apply Multi-Stage Form Validation — `clean_<field>()`, Cross-Field `clean()` & `ValidationError` Codes in real Django systems — specifically: Execution...

What Topic Is Next → Module 6 Capstone (Optimized Operations `ModelAdmin` (`list_select_related` + `autocomplete_fields`) & `ModelForm` Studio) → Next Topic 7.1: Custom User Models (`AbstractUser` vs `AbstractBaseUser`) — Why `AUTH_USER_MODEL` Must Come First!

Module 6 Hands-On Capstone Project (100 Marks): Optimized Operations `ModelAdmin` (`list_select_related` + `autocomplete_fields`) & `ModelForm` Studio — Configure a `CourseAdmin` + `ModuleInline` eliminating Admin `N+1` dropdown stalls on 100k-user ForeignKeys via `autocomplete_fields = ["instructor"]` and `list_select_related = ("category",...

MODULE 7 OF 16 • STAGE 2 • 12 LIVE + LAB HOURS • 4 TOPICS (32 SUBTOPICS) • 1 CAPSTONE PROJECT (100 MARKS)

Module 7: Authentication — `AbstractUser` Custom User Model, Registration, Sessions & Signed Cookies

Configure `AUTH_USER_MODEL` (`AbstractUser`) BEFORE your very first `migrate`, implement secure registration/login/RBAC flows with Argon2/PBKDF2 hashing, choose fast Redis/Cached-DB session backends, and sign tamper-proof cookies.

Objectives: ✓ Custom User Models (`AbstractUser` vs `AbstractBaseUser`) — Why `AUTH_USER_MODEL` Must Come First! — Official Django documentation's #1 project-setup recommendation: create `class User(AbstractUser): pass` and set **`AUTH_USER_MODEL = "accounts.User"`** BEFORE your first `manage.py migrate`! Comparing `AbstractUser` (keeps username/first_name/permissions) vs `AbstractBaseUser` (minimal email-only login). • ✓ Authentication Flows — `UserCreationForm`, `authenticate()`, `login()`, Django 5 `POST` Logout & Argon2 — Verifying credentials via `authenticate(request, username=..., password=...)`, cycling the session key on `login(request, user)` (preventing **Session Fixation**!), Django 5's `POST`-only `LogoutView`, `AUTH_PASSWORD_VALIDATORS`, and upgrading to **`Argon2PasswordHasher`**. • ✓ Django Session Engine — DB vs `cached_db` / Redis Backends, `HttpOnly`, `Secure` & Session Cleanup — How `SessionMiddleware` maps an opaque 32-char `sessionid` cookie to server-side `request.session` state, choosing **`cached_db`** or **Redis** over pure SQL `db` sessions on high-traffic sites, and running scheduled `python manage.py clearsessions`!. • ✓ Django Cookies & Cryptographic Signing — `set_signed_cookie()`, `get_signed_cookie()` & `SameSite` — Reading `request.COOKIES.get(...)`, setting tamper-evident cookies via **`response.set_signed_cookie("promo", "INR999", salt="checkout")`** + `request.get_signed_cookie(..., max_age=3600)` powered by `django.core.signing.TimestampSigner`, and `CSRF_COOKIE_SECURE`. • ✓ Build, verify, and ship Module Capstone: Email-First Custom `AbstractUser` Portal with Argon2 Hashing & Session Security.

7.1 Custom User Models (`AbstractUser` vs `AbstractBaseUser`) — Why `AUTH_USER_MODEL` Must Come First!

[50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Subtopics Covered: Set AUTH_USER_MODEL BEFORE 1st migrate! & AbstractUser vs AbstractBaseUser • Core Syntax, CLI & Production APIs: AUTH_USER_MODEL = "accounts.User" | ForeignKey(settings.AUTH_USER_MODEL, ...) | User = get_u... • Deep-Dive Mechanics: Settings.AUTH_USER_MODEL in ForeignKey! & Get_user_model() in views/forms/tests • Advanced Patterns: CustomUserManager create_user/superuser & USERNAME_FIELD = "email" • Production Failure Mode & Pitfall Guard • Hands-On Live Coding Lab: Normalized Email-Login Custom User Manager Helper • Idiomatic DJNG Implementation: select_related/prefetch_related QuerySets, DRF ViewSets & atomic transactions • Django 5 ORM/ASGI Pipeline, PostgreSQL & Celery Workers Execution & Memory Lifecycle for Custom User Models (`AbstractUser` vs `AbstractBaseU...

Purpose: To understand and apply Custom User Models (`AbstractUser` vs `AbstractBaseUser`) — Why `AUTH_USER_MODEL` Must Come First! in real Django systems — specifically: Official...

What Topic Is Next → Next Topic 7.2: Authentication Flows — `UserCreationForm`, `authenticate()`, `login()`, Django 5 `POST` Logout & Argon2

7.2 Authentication Flows — `UserCreationForm`, `authenticate()`, `login()`, Django 5 `POST` Logout & Argon2

[50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Subtopics Covered: Django 5 LogoutView requires POST! & Login() rotates session key (anti-fixation) • Core Syntax, CLI & Production APIs: user = authenticate(request, username=u, password=p); if user: login(request, user) | @login... • Deep-Dive Mechanics: Argon2PasswordHasher automatic upgrade & Authenticate() + @login_required • Advanced Patterns: PermissionRequiredMixin & Groups RBAC & AUTH_PASSWORD_VALIDATORS checks • Production Failure Mode & Pitfall Guard • Hands-On Live Coding Lab: Simulating Transparent Password Hasher Upgrade Check on Login • Idiomatic DJNG Implementation: select_related/prefetch_related QuerySets, DRF ViewSets & atomic transactions • Django 5 ORM/ASGI Pipeline, PostgreSQL & Celery Workers Execution & Memory Lifecycle for Authentication Flows

Purpose: To understand and apply Authentication Flows — `UserCreationForm`, `authenticate()`, `login()`, Django 5 `POST` Logout & Argon2 in real Django systems — specifically: Ver...

What Topic Is Next → Next Topic 7.3: Django Session Engine — DB vs `cached_db` / Redis Backends, `HttpOnly`, `Secure` & Session Cleanup

7.3 Django Session Engine — DB vs `cached_db` / Redis Backends, `HttpOnly`, `Secure` & Session Cleanup

[50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Subtopics Covered: Opaque server-side sessionid default! & Cached_db / Redis session backend • Core Syntax, CLI & Production APIs: SESSION_ENGINE = "django.contrib.sessions.backends.cached_db" | SESSION_COOKIE_HTTPONLY = Tr... • Deep-Dive Mechanics: Schedule manage.py clearsessions! & SESSION_COOKIE_HTTPONLY = True • Advanced Patterns: SESSION_COOKIE_SECURE = True (HTTPS) & SESSION_COOKIE_AGE & EXPIRE_AT_BROWSER_CLOSE • Production Failure Mode & Pitfall Guard • Hands-On Live Coding Lab: Pruning Expired Session Records Logic (`clearsessions` Equivalent) • Idiomatic DJNG Implementation: select_related/prefetch_related QuerySets, DRF ViewSets & atomic transactions • Django 5 ORM/ASGI Pipeline, PostgreSQL & Celery Workers Execution & Memory Lifecycle for Django Session Engine

Purpose: To understand and apply Django Session Engine — DB vs `cached_db` / Redis Backends, `HttpOnly`, `Secure` & Session Cleanup in real Django systems — specifically: How `Ses...

What Topic Is Next → Next Topic 7.4: Django Cookies & Cryptographic Signing — `set_signed_cookie()`, `get_signed_cookie()` & `SameSite`

7.4 Django Cookies & Cryptographic Signing — `set_signed_cookie()`, `get_signed_cookie()` & `SameSite`

[50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Subtopics Covered: Set_signed_cookie & get_signed_cookie & Pass default=None for tamper safety! • Core Syntax, CLI & Production APIs: resp.set_signed_cookie("ref", "MAA", salt="promo", httponly=True, secure=True, samesite="Lax... • Deep-Dive Mechanics: Salt="..." prevents cookie replay! & TimestampSigner max_age expiry • Advanced Patterns: SESSION_COOKIE_SAMESITE = "Lax" & CSRF_COOKIE_SECURE = True • Production Failure Mode & Pitfall Guard • Hands-On Live Coding Lab: Domain-Salted HMAC Cookie Signer & Replay Rejector • Idiomatic DJNG Implementation: select_related/prefetch_related QuerySets, DRF ViewSets & atomic transactions • Django 5 ORM/ASGI Pipeline, PostgreSQL & Celery Workers Execution & Memory Lifecycle for Django Cookies

Purpose: To understand and apply Django Cookies & Cryptographic Signing — `set_signed_cookie()`, `get_signed_cookie()` & `SameSite` in real Django systems — specifically: Reading...

What Topic Is Next → Module 7 Capstone (Email-First Custom `AbstractUser` Portal with Argon2 Hashing & Session Security) → Next Topic 8.1: Django Middleware Onion — `__call__(request)`, Ordering Rules & `@sync_and_async_middleware`

Module 7 Hands-On Capstone Project (100 Marks): Email-First Custom `AbstractUser` Portal with Argon2 Hashing & Session Security — Create `accounts.User(AbstractUser)` adding `student_code` and unique lowercase `email`, configure `PASSWORD_HASHERS` with `Argon2PasswordHasher` primary, enforce `POST`-only `/logout/` (Dja...

MODULE 8 OF 16 • STAGE 2 • 12 LIVE + LAB HOURS • 4 TOPICS (32 SUBTOPICS) • 1 CAPSTONE PROJECT (100 MARKS)

Module 8: Middleware Onion Pipeline, `HttpRequest`/`JsonResponse`, Exception Handling & Secure Uploads

Master Django's WSGI/ASGI Middleware onion order, inspect `HttpRequest` / stream `JsonResponse` & `FileResponse`, build custom `handler404`/`handler500` & `PermissionDenied` boundaries, and harden `MEDIA_ROOT` / S3 file uploads.

Objectives: ✓ Django Middleware Onion — `__call__(request)`, Ordering Rules & `@sync_and_async_middleware` — How Django wraps the view in a nested Middleware **onion** (top-to-bottom on request; bottom-to-top on response!), optional `process_view` / `process_exception` / `process_template_response` hooks, and writing dual WSGI/ASGI middleware without sync-to-async thread-hop penalties!. • ✓ `HttpRequest` & `HttpResponse` — `QueryDict.getlist()`, `request.headers`, `JsonResponse` & `FileResponse` — Inspecting `request.method`, case-insensitive `request.headers["Authorization"]`, multi-valued `request.GET.getlist("tag")`, parsing JSON bodies (`json.loads(request.body)`), returning `JsonResponse` (`safe=True` dict default!), and zero-copy `FileResponse(as_attachment=True)`. • ✓ Django Exception Handling — `Http404`, `PermissionDenied`, Custom `handler500` & Sentry Logging — How Django maps built-in exceptions to HTTP statuses (`Http404` -> `404`, `PermissionDenied` -> `403`, `SuspiciousOperation` -> `400`, unhandled `Exception` -> `500`), configuring root `urls.py` `handler404` / `handler500`, and structured `LOGGING` dictionaries. • ✓ Secure File Uploads (`FileField` / `ImageField`) — `<form enctype="multipart/form-data">`, `chunks()` & S3 — Binding `Form(request.POST, request.FILES)`, streaming large uploads safely via `for chunk in uploaded_file.chunks():`, randomized `upload_to` paths, `FileExtensionValidator` + `Pillow` verification, and `django-storages` S3 / R2 object storage. • ✓ Build, verify, and ship Module Capstone: Request-ID Correlation + Latency Middleware & Validated S3/`FileField` Assignment Uploader.

8.1 Django Middleware Onion — `__call__(request)`, Ordering Rules & `@sync_and_async_middleware` [50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Subtopics Covered: Top-down request / bottom-up response onion & SecurityMiddleware -> Session -> Csrf -> Auth • Core Syntax, CLI & Production APIs: class TimingMiddleware: def __init__(self, get_response): self.get_response = get_response;... • Deep-Dive Mechanics: Never store request state on self! & __init__ runs once at startup! • Advanced Patterns: Process_exception & process_view hooks & @sync_and_async_middleware ASGI support • Production Failure Mode & Pitfall Guard • Hands-On Live Coding Lab: Tracing Django's Top-Down Request / Bottom-Up Response Middleware Onion • Idiomatic DJNG Implementation: select_related/prefetch_related QuerySets, DRF ViewSets & atomic transactions • Django 5 ORM/ASGI Pipeline, PostgreSQL & Celery Workers Execution & Memory Lifecycle for Django Middleware Onion

Purpose: To understand and apply Django Middleware Onion — `__call__(request)`, Ordering Rules & `@sync_and_async_middleware` in real Django systems — specifically: How Django wra...

What Topic Is Next → Next Topic 8.2: `HttpRequest` & `HttpResponse` — `QueryDict.getlist()`, `request.headers`, `JsonResponse` & `FileResponse`

8.2 `HttpRequest` & `HttpResponse` — `QueryDict.getlist()`, `request.headers`, `JsonResponse` & `FileResponse` [50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Subtopics Covered: QueryDict.getlist() vs .get()! & Case-insensitive request.headers[...] • Core Syntax, CLI & Production APIs: tags = request.GET.getlist("tag") | return JsonResponse({"items": rows}, status=200) • Deep-Dive Mechanics: JsonResponse({"results": ...}) envelope & FileResponse OS streaming chunks • Advanced Patterns: SECURE_PROXY_SSL_HEADER behind Nginx & HttpResponse(status=204) empty body • Production Failure Mode & Pitfall Guard • Hands-On Live Coding Lab: Multi-Valued `QueryDict.getlist()` vs `.get()` Behavior Simulation • Idiomatic DJNG Implementation: select_related/prefetch_related QuerySets, DRF ViewSets & atomic transactions • Django 5 ORM/ASGI Pipeline, PostgreSQL & Celery Workers Execution & Memory Lifecycle for `HttpRequest`

Purpose: To understand and apply `HttpRequest` & `HttpResponse` — `QueryDict.getlist()`, `request.headers`, `JsonResponse` & `FileResponse` in real Django systems — specifically:...

What Topic Is Next → Next Topic 8.3: Django Exception Handling — `Http404`, `PermissionDenied`, Custom `handler500` & Sentry Logging

8.3 Django Exception Handling — `Http404`, `PermissionDenied`, Custom `handler500` & Sentry Logging [50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Subtopics Covered: Http404 (404) & PermissionDenied (403) & SuspiciousOperation / BadRequest (400) • Core Syntax, CLI & Production APIs: from django.core.exceptions import PermissionDenied, BadRequest | handler404 = "core.views.c... • Deep-Dive Mechanics: Keep handler500 DB-free & static-safe! & Never bare except: or leak tracebacks • Advanced Patterns: Course.DoesNotExist vs MultipleObjectsReturned & Structured LOGGING + Sentry SDK • Production Failure Mode & Pitfall Guard • Hands-On Live Coding Lab: Mapping Domain Exceptions to Safe Client HTTP Status Envelopes • Idiomatic DJNG Implementation: select_related/prefetch_related QuerySets, DRF ViewSets & atomic transactions • Django 5 ORM/ASGI Pipeline, PostgreSQL & Celery Workers Execution & Memory Lifecycle for Django Exception Handling

Purpose: To understand and apply Django Exception Handling — `Http404`, `PermissionDenied`, Custom `handler500` & Sentry Logging in real Django systems — specifically: How Django...

What Topic Is Next → Next Topic 8.4: Secure File Uploads (`FileField` / `ImageField`) — `<form enctype="multipart/form-data">`, `chunks()` & S3

8.4 Secure File Uploads (`FileField` / `ImageField`) — `<form enctype="multipart/form-data">`, `chunks()` & S3 [50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Subtopics Covered: Enctype="multipart/form-data" + request.FILES & Iterate uploaded_file.chunks() (64 KB)! • Core Syntax, CLI & Production APIs: form = UploadForm(request.POST, request.FILES) | for chunk in f.chunks(): dest.write(chunk) • Deep-Dive Mechanics: UUID upload_to path prevents collisions & SVG/HTML upload Stored XSS hazard! • Advanced Patterns: DATA_UPLOAD_MAX_MEMORY_SIZE DoS guard & Django-storages S3 / presigned URLs • Production Failure Mode & Pitfall Guard • Hands-On Live Coding Lab: Generating Collision-Proof UUID `upload_to` Storage Keys • Idiomatic DJNG Implementation: select_related/prefetch_related QuerySets, DRF ViewSets & atomic transactions • Django 5 ORM/ASGI Pipeline, PostgreSQL & Celery Workers Execution & Memory Lifecycle for Secure File Uploads (`FileField` / `ImageField`)

Purpose: To understand and apply Secure File Uploads (`FileField` / `ImageField`) — `<form enctype="multipart/form-data">`, `chunks()` & S3 in real Django systems — specifically:...

What Topic Is Next → Module 8 Capstone (Request-ID Correlation + Latency Middleware & Validated S3/`FileField` Assignment Uploader) → Next Topic 9.1: Production Django CRUD Architecture — Ownership-Scoped Queries, PRG & `POST`-Only Delete

Module 8 Hands-On Capstone Project (100 Marks): Request-ID Correlation + Latency Middleware & Validated S3/`FileField` Assignment Uploader — Implement a sync/async-compatible `RequestCorrelationMiddleware` injecting `X-Request-ID` & `X-Response-Time-Ms` headers, structured `400`/`403`/`404`/`500` JSON/HTML error handlers, and a v...

MODULE 9 OF 16 • STAGE 3 • 12 LIVE + LAB HOURS • 3 TOPICS (24 SUBTOPICS) • 1 CAPSTONE PROJECT (100 MARKS)

Module 9: End-to-End CRUD Architecture, Django REST Framework (DRF) Serializers & Streaming CSV Exports

Build complete transactional CRUD applications, design hypermedia/REST APIs with Django REST Framework (`ModelSerializer`, `HyperlinkedModelSerializer`, `ViewSet`, Permissions & Throttling), and stream million-row CSV reports.

Objectives: ✓ Production Django CRUD Architecture — Ownership-Scoped Queries, PRG & `POST`-Only Delete — Combining `ListView` (`paginate_by = 20`), `DetailView`, `CreateView`, `UpdateView`, and `DeleteView` with `LoginRequiredMixin`, scoping `get_object_or_404(Task, pk=pk, owner=request.user)` to block **IDOR** (`OWASP API1`), and flash `django.contrib.messages`. • ✓ Django REST Framework (DRF) — `ModelSerializer`, `URLField`, `HyperlinkedModelSerializer` & `ViewSet`s — Building clean JSON APIs with DRF: `serializers.ModelSerializer`, `serializers.URLField` validation, **`HyperlinkedIdentityField` / `HyperlinkedRelatedField`** (HATEOAS resource URLs!), `validate_<field>()` / `validate()`, `ModelViewSet` + `DefaultRouter`, Permissions, Throttling, and `drf-spectacular` OpenAPI!. • ✓ Streaming CSV Generation — `csv.writer`, `StreamingHttpResponse` (`Echo` Buffer) & Formula-Injection Defense — Official Django pattern for exporting large datasets: small CSVs via `HttpResponse(content_type="text/csv")` vs huge million-row exports via **`StreamingHttpResponse`** paired with a lightweight `Echo` pseudo-buffer (`def write(self, value): return value`) and `queryset.values_list().iterator(chunk_size=2000)`!. • ✓ Build, verify, and ship Module Capstone: DRF Course & Mock-Attempt REST API (`ModelViewSet` + `HyperlinkedIdentityField`) + Streaming CSV Exporter.

9.1 Production Django CRUD Architecture — Ownership-Scoped Queries, PRG & `POST`-Only Delete [50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Subtopics Covered: Scope queries by owner=request.user (anti-IDOR)! & NEVER delete records on HTTP GET! • Core Syntax, CLI & Production APIs: task = get_object_or_404(Task, pk=pk, owner=request.user) | messages.success(request, "Saved... • Deep-Dive Mechanics: Post-Redirect-Get + contrib.messages & Paginator with deterministic -id tie-breaker • Advanced Patterns: ModelForm instance=obj for Update & Atomic multi-table CRUD writes • Production Failure Mode & Pitfall Guard • Hands-On Live Coding Lab: Anti-IDOR Ownership-Scoped Lookup Helper • Idiomatic DJNG Implementation: select_related/prefetch_related QuerySets, DRF ViewSets & atomic transactions • Django 5 ORM/ASGI Pipeline, PostgreSQL & Celery Workers Execution & Memory Lifecycle for Production Django CRUD Architecture

Purpose: To understand and apply Production Django CRUD Architecture — Ownership-Scoped Queries, PRG & `POST`-Only Delete in real Django systems — specifically: Combining `ListVie...

What Topic Is Next → Next Topic 9.2: Django REST Framework (DRF) — `ModelSerializer`, `URLField`, `HyperlinkedModelSerializer` & `ViewSet`s

9.2 Django REST Framework (DRF) — `ModelSerializer`, `URLField`, `HyperlinkedModelSerializer` & `ViewSet`s [50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Subtopics Covered: Pass context={"request": request} for Hyperlinked URLs! & HyperlinkedModelSerializer & URLField • Core Syntax, CLI & Production APIs: class CourseSerializer(serializers.HyperlinkedModelSerializer): url = serializers.Hyperlinke... • Deep-Dive Mechanics: Optimize ViewSet.get_queryset() with select_related! & Explicit Meta.fields & read_only_fields • Advanced Patterns: IsAuthenticated + custom object Permission & AnonRateThrottle / UserRateThrottle & SimpleJWT • Production Failure Mode & Pitfall Guard • Hands-On Live Coding Lab: Simulating DRF `HyperlinkedIdentityField` Absolute URL Resolution with `request` Context • Idiomatic DJNG Implementation: select_related/prefetch_related QuerySets, DRF ViewSets & atomic transactions • Django 5 ORM/ASGI Pipeline, PostgreSQL & Celery Workers Execution & Memory Lifecycle for Django REST Framework (DRF)

Purpose: To understand and apply Django REST Framework (DRF) — `ModelSerializer`, `URLField`, `HyperlinkedModelSerializer` & `ViewSet`s in real Django systems — specifically: Buil...

What Topic Is Next → Next Topic 9.3: Streaming CSV Generation — `csv.writer`, `StreamingHttpResponse` (`Echo` Buffer) & Formula-Injection Defense

9.3 Streaming CSV Generation — `csv.writer`, `StreamingHttpResponse` (`Echo` Buffer) & Formula-Injection Defense [50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Subtopics Covered: StreamingHttpResponse + Echo pseudo-buffer! & Values_list().iterator(chunk_size=2000) O(1) RAM • Core Syntax, CLI & Production APIs: class Echo: def write(self, value): return value | StreamingHttpResponse((writer.writerow(r)... • Deep-Dive Mechanics: Content-Disposition: attachment; filename=... & CSV Formula Injection (=, +, -, @) defense! • Advanced Patterns: Csv.writer proper quoting & escaping & Celery + S3 presigned link for 10M+ rows • Production Failure Mode & Pitfall Guard • Hands-On Live Coding Lab: Django Official `Echo` Streaming CSV Generator + Formula-Injection Sanitizer • Idiomatic DJNG Implementation: select_related/prefetch_related QuerySets, DRF ViewSets & atomic transactions • Django 5 ORM/ASGI Pipeline, PostgreSQL & Celery Workers Execution & Memory Lifecycle for Streaming CSV Generation

Purpose: To understand and apply Streaming CSV Generation — `csv.writer`, `StreamingHttpResponse` (`Echo` Buffer) & Formula-Injection Defense in real Django systems — specifically...

What Topic Is Next → Module 9 Capstone (DRF Course & Mock-Attempt REST API (`ModelViewSet` + `HyperlinkedIdentityField`) + Streaming CSV Exporter) → Next Topic 10.1: Django Fixtures & Fast Testing — `dumpdata`, `loaddata`, Natural Keys & `TestCase.setUpTestData()`

Module 9 Hands-On Capstone Project (100 Marks): DRF Course & Mock-Attempt REST API (`ModelViewSet` + `HyperlinkedIdentityField`) + Streaming CSV Exporter — Build `/api/v1/courses/` & `/api/v1/attempts/` DRF endpoints with `ModelSerializer` explicit `fields`, `HyperlinkedIdentityField` detail URLs, `IsAuthenticatedOrReadOnly` + custom `IsOwner`...

MODULE 10 OF 16 • STAGE 3 • 11 LIVE + LAB HOURS • 2 TOPICS (16 SUBTOPICS) • 1 CAPSTONE PROJECT (100 MARKS)

Module 10: Seed Fixtures, `TestCase` / `pytest-django` Automation & Production WSGI/ASGI Server Deployment

Manage reference data & fast factory tests (`dumpdata`/`loaddata`, `factory_boy`, `TestCase.setUpTestData`), pass `manage.py check --deploy`, and deploy Django 5 behind Apache (`mod_wsgi`) or Nginx + Gunicorn/Uvicorn with HTTPS, HSTS & PostgreSQL.

Objectives: ✓ Django Fixtures & Fast Testing — `dumpdata`, `loaddata`, Natural Keys & `TestCase.setUpTestData()` — Serializing initial lookup catalogs with `manage.py dumpdata catalog.Category --natural-foreign --natural-primary --indent 2`, loading fixtures (`loaddata`), why **`factory_boy` + `setUpTestData()`** beats brittle JSON fixtures in large test suites, and how `loaddata` interacts with `post_save` signals (`kwargs["raw"]`)!. • ✓ Production Deployment — Apache `mod_wsgi` (`WSGIDaemonProcess`), Nginx + Gunicorn/Uvicorn & `check --deploy` — Serving `config/wsgi.py` & `config/asgi.py` in production: Apache **`mod_wsgi` Daemon Mode** (`WSGIDaemonProcess ... python-home=/srv/venv python-path=/srv/app`) vs **Nginx + Gunicorn / Uvicorn**, mapping `Alias /static/ /srv/staticfiles/`, `CSRF_TRUSTED_ORIGINS`, HSTS, and passing `manage.py check --deploy`!. • ✓ Build, verify, and ship Module Capstone: Hardened Production Django 5 Deployment Blueprint (`check --deploy` Clean + Gunicorn/Apache Config + CI Suite).

10.1 Django Fixtures & Fast Testing — `dumpdata`, `loaddata`, Natural Keys & `TestCase.setUpTestData()` [50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Subtopics Covered: --natural-foreign --natural-primary! & Exclude -e contenttypes -e auth.Permission! • Core Syntax, CLI & Production APIs: python manage.py dumpdata catalog --natural-foreign --natural-primary -o fixtures/catalog.js... • Deep-Dive Mechanics: Check kwargs.get("raw") in post_save signals! & TestCase.setUpTestData() 10x test speedup! • Advanced Patterns: Factory_boy / model_bakery over brittle JSON & Data Migrations vs loaddata in CI/CD • Production Failure Mode & Pitfall Guard • Hands-On Live Coding Lab: Signal `raw=True` Guard During `loaddata` Fixture Deserialization • Idiomatic DJNG Implementation: select_related/prefetch_related QuerySets, DRF ViewSets & atomic transactions • Django 5 ORM/ASGI Pipeline, PostgreSQL & Celery Workers Execution & Memory Lifecycle for Django Fixtures

Purpose: To understand and apply Django Fixtures & Fast Testing — `dumpdata`, `loaddata`, Natural Keys & `TestCase.setUpTestData()` in real Django systems — specifically: Serializ...

What Topic Is Next → Next Topic 10.2: Production Deployment — Apache `mod_wsgi` (`WSGIDaemonProcess`), Nginx + Gunicorn/Uvicorn & `check --deploy`

10.2 Production Deployment — Apache `mod_wsgi` (`WSGIDaemonProcess`), Nginx + Gunicorn/Uvicorn & `check --deploy` [50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Subtopics Covered: Manage.py check --deploy CI gate! & CSRF_TRUSTED_ORIGINS = ["https://..."] (Django 4+) • Core Syntax, CLI & Production APIs: python manage.py check --deploy && python manage.py collectstatic --noinput && gunicorn conf... • Deep-Dive Mechanics: SECURE_PROXY_SSL_HEADER behind Nginx/ALB & Apache WSGIDaemonProcess + python-home venv • Advanced Patterns: Nginx Alias /static/ + Gunicorn/Uvicorn workers & --max-requests + --max-requests-jitter • Production Failure Mode & Pitfall Guard • Hands-On Live Coding Lab: Production `check --deploy` Security Settings Auditor • Idiomatic DJNG Implementation: select_related/prefetch_related QuerySets, DRF ViewSets & atomic transactions • Django 5 ORM/ASGI Pipeline, PostgreSQL & Celery Workers Execution & Memory Lifecycle for Production Deployment

Purpose: To understand and apply Production Deployment — Apache `mod_wsgi` (`WSGIDaemonProcess`), Nginx + Gunicorn/Uvicorn & `check --deploy` in real Django systems — specifically...

What Topic Is Next → Module 10 Capstone (Hardened Production Django 5 Deployment Blueprint (`check --deploy` Clean + Gunicorn/Apache Config + CI Suite)) → Next Topic 11.1: ModelSerializers, Field-Level/Object Validation & Read/Write Splits

Module 10 Hands-On Capstone Project (100 Marks): Hardened Production Django 5 Deployment Blueprint (`check --deploy` Clean + Gunicorn/Apache Config + CI Suite) — Prepare a production `settings/prod.py` passing `python manage.py check --deploy` with 0 warnings, write a `TestCase` suite using `setUpTestData` + `assertNumQueries`, export natural-key ref...

MODULE 11 OF 16 • STAGE 3 • 14 LIVE + LAB HOURS • 3 TOPICS (24 SUBTOPICS) • 1 CAPSTONE PROJECT (100 MARKS)

Django REST Framework (DRF): ViewSets, Serializers, RBAC & Throttling

Production-grade Django Training engineering module covering Django REST Framework (DRF): ViewSets, Serializers, RBAC & Throttling with hands-on labs and capstone build.

Objectives: ✓ ModelSerializers, Field-Level/Object Validation & Read/Write Splits — Expose clean versioned REST endpoints with strict input sanitization. • ✓ Custom BasePermission Classes, SimpleJWT Rotation & Scoped Throttles — Enforce tenant-level row isolation and per-plan API rate limits cleanly. • ✓ django-filter, CursorPagination & drf-spectacular OpenAPI 3 Docs — Deliver fast keyset-paginated search endpoints with interactive Swagger schemas. • ✓ Build, verify, and ship Module Capstone: Enterprise Multi-Tenant REST API Built with Django & DRF.

11.1 ModelSerializers, Field-Level/Object Validation & Read/Write Splits [65 mins • Assignment (100M) • Hands-On Lab (50M) • 2 Mock Sets • 2 AI Vivas]

Subtopics Covered: Plain • Production Implementation of Field • Integration with Custom BasePermission Classes, SimpleJWT Rotation & Scoped Throttles • Live Debugging, Telemetry & Technical Interview Trade • What Is ModelSerializers, Field • In Django Training, ModelSerializers, Field • Even if you come from a non • Runnable Code Implementation (PYTHON)

Purpose: To understand and apply ModelSerializers, Field-Level/Object Validation & Read/Write Splits in real Django systems — specifically: Expose clean versioned REST endpoints w...

What Topic Is Next → Next Topic 11.2: Custom BasePermission Classes, SimpleJWT Rotation & Scoped Throttles

11.2 Custom BasePermission Classes, SimpleJWT Rotation & Scoped Throttles [65 mins • Assignment (100M) • Hands-On Lab (50M) • 2 Mock Sets • 2 AI Vivas]

Subtopics Covered: Plain • Production Implementation of SimpleJWT Rotation • Integration with django • Live Debugging, Telemetry & Technical Interview Trade • What Is Custom BasePermission Classes, SimpleJWT Rotation & Scoped Throttles & Why Industry Teams Use It • In Django Training, Custom BasePermission Classes, SimpleJWT Rotation & Scoped Throttles exists for one clear real • Even if you come from a non • Runnable Code Implementation (PYTHON)

Purpose: To understand and apply Custom BasePermission Classes, SimpleJWT Rotation & Scoped Throttles in real Django systems — specifically: Enforce tenant-level row isolation and...

What Topic Is Next → Next Topic 11.3: django-filter, CursorPagination & drf-spectacular OpenAPI 3 Docs

11.3 django-filter, CursorPagination & drf-spectacular OpenAPI 3 Docs [65 mins • Assignment (100M) • Hands-On Lab (50M) • 2 Mock Sets • 2 AI Vivas]

Subtopics Covered: Plain • Production Implementation of CursorPagination • Integration with ModelSerializers, Field • Live Debugging, Telemetry & Technical Interview Trade • What Is django • In Django Training, django • Even if you come from a non • Runnable Code Implementation (PYTHON)

Purpose: To understand and apply django-filter, CursorPagination & drf-spectacular OpenAPI 3 Docs in real Django systems — specifically: Deliver fast keyset-paginated search endpo...

What Topic Is Next → Module 11 Capstone (Enterprise Multi-Tenant REST API Built with Django & DRF) → Next Topic 12.1: select_related vs prefetch_related, Prefetch() & values/only()

Module 11 Hands-On Capstone Project (100 Marks): Enterprise Multi-Tenant REST API Built with Django & DRF — Design, implement, test, and document 'Enterprise Multi-Tenant REST API Built with Django & DRF' applying every production concept from Module 11 (Django REST Framework (DRF): ViewSets, Seri...

MODULE 12 OF 16 • STAGE 3 • 14 LIVE + LAB HOURS • 3 TOPICS (24 SUBTOPICS) • 1 CAPSTONE PROJECT (100 MARKS)

Django ORM Performance: select_related, Subqueries, Locking & Migrations

Production-grade Django Training engineering module covering Django ORM Performance: select_related, Subqueries, Locking & Migrations with hands-on labs and capstone build.

Objectives: ✓ select_related vs prefetch_related, Prefetch() & values/only() — Cut 300-query DRF list endpoints down to 2 indexed PostgreSQL queries. • ✓ F() Atomic Expressions, Q() Filters, Window Functions & Subquery() — Increment counters race-free in SQL and compute complex analytics without Python loops. • ✓ transaction.atomic(), select_for_update() & Safe Data Migrations — Guarantee ACID wallet/stock updates and split schema vs data migrations safely. • ✓ Build, verify, and ship Module Capstone: Zero-N+1 Financial & Inventory Engine in Django ORM.

12.1 select_related vs prefetch_related, Prefetch() & values/only() [65 mins • Assignment (100M) • Hands-On Lab (50M) • 2 Mock Sets • 2 AI Vivas]

Subtopics Covered: Plain • Production Implementation of Prefetch() • Integration with F() Atomic Expressions, Q() Filters, Window Functions & Subquery() • Live Debugging, Telemetry & Technical Interview Trade • What Is select_related vs prefetch_related, Prefetch() & values/only() & Why Industry Teams Use It • In Django Training, select_related vs prefetch_related, Prefetch() & values/only() exists for one clear real • Even if you come from a non • Runnable Code Implementation (PYTHON)

Purpose: To understand and apply select_related vs prefetch_related, Prefetch() & values/only() in real Django systems — specifically: Cut 300-query DRF list endpoints down to 2 i...

What Topic Is Next → Next Topic 12.2: F() Atomic Expressions, Q() Filters, Window Functions & Subquery()

12.2 F() Atomic Expressions, Q() Filters, Window Functions & Subquery() [65 mins • Assignment (100M) • Hands-On Lab (50M) • 2 Mock Sets • 2 AI Vivas]

Subtopics Covered: Plain • Production Implementation of Q() Filters • Integration with transaction.atomic(), select_for_update() & Safe Data Migrations • Live Debugging, Telemetry & Technical Interview Trade • What Is F() Atomic Expressions, Q() Filters, Window Functions & Subquery() & Why Industry Teams Use It • In Django Training, F() Atomic Expressions, Q() Filters, Window Functions & Subquery() exists for one clear real • Even if you come from a non • Runnable Code Implementation (PYTHON)

Purpose: To understand and apply F() Atomic Expressions, Q() Filters, Window Functions & Subquery() in real Django systems — specifically: Increment counters race-free in SQL and...

What Topic Is Next → Next Topic 12.3: transaction.atomic(), select_for_update() & Safe Data Migrations

12.3 transaction.atomic(), select_for_update() & Safe Data Migrations [65 mins • Assignment (100M) • Hands-On Lab (50M) • 2 Mock Sets • 2 AI Vivas]

Subtopics Covered: Plain • Production Implementation of select_for_update() • Integration with select_related vs prefetch_related, Prefetch() & values/only() • Live Debugging, Telemetry & Technical Interview Trade • What Is transaction.atomic(), select_for_update() & Safe Data Migrations & Why Industry Teams Use It • In Django Training, transaction.atomic(), select_for_update() & Safe Data Migrations exists for one clear real • Even if you come from a non • Runnable Code Implementation (PYTHON)

Purpose: To understand and apply transaction.atomic(), select_for_update() & Safe Data Migrations in real Django systems — specifically: Guarantee ACID wallet/stock updates and sp...

What Topic Is Next → Module 12 Capstone (Zero-N+1 Financial & Inventory Engine in Django ORM) → Next Topic 13.1: transaction.on_commit() + Celery Idempotent Tasks & Retry Policies

Module 12 Hands-On Capstone Project (100 Marks): Zero-N+1 Financial & Inventory Engine in Django ORM — Design, implement, test, and document 'Zero-N+1 Financial & Inventory Engine in Django ORM' applying every production concept from Module 12 (Django ORM Performance: select_related, Subqueri...

MODULE 13 OF 16 • STAGE 4 • 14 LIVE + LAB HOURS • 3 TOPICS (24 SUBTOPICS) • 1 CAPSTONE PROJECT (100 MARKS)

Celery Distributed Queues, Redis Caching & Django Channels WebSockets

Production-grade Django Training engineering module covering Celery Distributed Queues, Redis Caching & Django Channels WebSockets with hands-on labs and capstone build.

Objectives: ✓ transaction.on_commit() + Celery Idempotent Tasks & Retry Policies — Ensure Celery workers never pick up a task before its PostgreSQL row actually commits!. • ✓ Per-View/Low-Level Redis Cache Keys & Signal-Free Explicit Invalidation — Accelerate heavy catalog/dashboard reads without hidden Django signal spaghetti. • ✓ ASGI Django Channels + Redis Channel Layer for Live WebSockets — Push real-time order statuses and chat alerts directly to authenticated browsers. • ✓ Build, verify, and ship Module Capstone: Real-Time Notification & Async Report Platform on Django.

13.1 transaction.on_commit() + Celery Idempotent Tasks & Retry Policies [65 mins • Assignment (100M) • Hands-On Lab (50M) • 2 Mock Sets • 2 AI Vivas]

Subtopics Covered: Plain • Production Implementation of Retry Policies • Integration with Per • Live Debugging, Telemetry & Technical Interview Trade • What Is transaction.on_commit() + Celery Idempotent Tasks & Retry Policies & Why Industry Teams Use It • In Django Training, transaction.on_commit() + Celery Idempotent Tasks & Retry Policies exists for one clear real • Even if you come from a non • Runnable Code Implementation (PYTHON)

Purpose: To understand and apply transaction.on_commit() + Celery Idempotent Tasks & Retry Policies in real Django systems — specifically: Ensure Celery workers never pick up a ta...

What Topic Is Next → Next Topic 13.2: Per-View/Low-Level Redis Cache Keys & Signal-Free Explicit Invalidation

13.2 Per-View/Low-Level Redis Cache Keys & Signal-Free Explicit Invalidation [65 mins • Assignment (100M) • Hands-On Lab (50M) • 2 Mock Sets • 2 AI Vivas]

Subtopics Covered: Plain • Production Implementation of Signal • Integration with ASGI Django Channels + Redis Channel Layer for Live WebSockets • Live Debugging, Telemetry & Technical Interview Trade • What Is Per • In Django Training, Per • Even if you come from a non • Runnable Code Implementation (PYTHON)

Purpose: To understand and apply Per-View/Low-Level Redis Cache Keys & Signal-Free Explicit Invalidation in real Django systems — specifically: Accelerate heavy catalog/dashboard...

What Topic Is Next → Next Topic 13.3: ASGI Django Channels + Redis Channel Layer for Live WebSockets

13.3 ASGI Django Channels + Redis Channel Layer for Live WebSockets [65 mins • Assignment (100M) • Hands-On Lab (50M) • 2 Mock Sets • 2 AI Vivas]

Subtopics Covered: Plain • Production Implementation of Edge • Integration with transaction.on_commit() + Celery Idempotent Tasks & Retry Policies • Live Debugging, Telemetry & Technical Interview Trade • What Is ASGI Django Channels + Redis Channel Layer for Live WebSockets & Why Industry Teams Use It • In Django Training, ASGI Django Channels + Redis Channel Layer for Live WebSockets exists for one clear real • Even if you come from a non • Runnable Code Implementation (PYTHON)

Purpose: To understand and apply ASGI Django Channels + Redis Channel Layer for Live WebSockets in real Django systems — specifically: Push real-time order statuses and chat alert...

What Topic Is Next → Module 13 Capstone (Real-Time Notification & Async Report Platform on Django) → Next Topic 14.1: manage.py check --deploy: HSTS, CSRF_TRUSTED_ORIGINS, Secure Cookies & S3

Module 13 Hands-On Capstone Project (100 Marks): Real-Time Notification & Async Report Platform on Django — Design, implement, test, and document 'Real-Time Notification & Async Report Platform on Django' applying every production concept from Module 13 (Celery Distributed Queues, Redis Caching &...

MODULE 14 OF 16 • STAGE 4 • 14 LIVE + LAB HOURS • 3 TOPICS (24 SUBTOPICS) • 1 CAPSTONE PROJECT (100 MARKS)

Django Security Checklist, pytest-django CI & Gunicorn/Uvicorn Docker

Production-grade Django Training engineering module covering Django Security Checklist, pytest-django CI & Gunicorn/Uvicorn Docker with hands-on labs and capstone build.

Objectives: ✓ manage.py check --deploy: HSTS, CSRF_TRUSTED_ORIGINS, Secure Cookies & S3 — Pass enterprise OWASP security reviews and serve media via signed S3/CloudFront URLs. • ✓ pytest-django, FactoryBoy Fixtures & Query-Count Regression Tests — Assert exact max SQL query counts (django_assert_num_queries) on every pull request. • ✓ Multi-Stage Docker, WhiteNoise/CDN Static Assets & Zero-Downtime Release — Ship production WSGI/ASGI Django containers with health checks and Sentry monitoring. • ✓ Build, verify, and ship Module Capstone: Hardened Containerized Django Production Deployment.

14.1 manage.py check --deploy: HSTS, CSRF_TRUSTED_ORIGINS, Secure Cookies & S3 [65 mins • Assignment (100M) • Hands-On Lab (50M) • 2 Mock Sets • 2 AI Vivas]

Subtopics Covered: Plain • Production Implementation of CSRF_TRUSTED_ORIGINS • Integration with pytest • Live Debugging, Telemetry & Technical Interview Trade • What Is manage.py check • In Django Training, manage.py check • Even if you come from a non • Runnable Code Implementation (PYTHON)

Purpose: To understand and apply manage.py check --deploy: HSTS, CSRF_TRUSTED_ORIGINS, Secure Cookies & S3 in real Django systems — specifically: Pass enterprise OWASP security re...

What Topic Is Next → Next Topic 14.2: pytest-django, FactoryBoy Fixtures & Query-Count Regression Tests

14.2 pytest-django, FactoryBoy Fixtures & Query-Count Regression Tests [65 mins • Assignment (100M) • Hands-On Lab (50M) • 2 Mock Sets • 2 AI Vivas]

Subtopics Covered: Plain • Production Implementation of FactoryBoy Fixtures • Integration with Multi • Live Debugging, Telemetry & Technical Interview Trade • What Is pytest • In Django Training, pytest • Even if you come from a non • Runnable Code Implementation (PYTHON)

Purpose: To understand and apply pytest-django, FactoryBoy Fixtures & Query-Count Regression Tests in real Django systems — specifically: Assert exact max SQL query counts (django...

What Topic Is Next → Next Topic 14.3: Multi-Stage Docker, WhiteNoise/CDN Static Assets & Zero-Downtime Release

14.3 Multi-Stage Docker, WhiteNoise/CDN Static Assets & Zero-Downtime Release [65 mins • Assignment (100M) • Hands-On Lab (50M) • 2 Mock Sets • 2 AI Vivas]

Subtopics Covered: Plain • Production Implementation of WhiteNoise/CDN Static Assets • Integration with manage.py check • Live Debugging, Telemetry & Technical Interview Trade • What Is Multi • In Django Training, Multi • Even if you come from a non • Runnable Code Implementation (PYTHON)

Purpose: To understand and apply Multi-Stage Docker, WhiteNoise/CDN Static Assets & Zero-Downtime Release in real Django systems — specifically: Ship production WSGI/ASGI Django c...

What Topic Is Next → Module 14 Capstone (Hardened Containerized Django Production Deployment) → Next Topic 15.1: Eliminating ORM N+1 Queries with select_related, prefetch_related & Subquery/Exists

Module 14 Hands-On Capstone Project (100 Marks): Hardened Containerized Django Production Deployment — Design, implement, test, and document 'Hardened Containerized Django Production Deployment' applying every production concept from Module 14 (Django Security Checklist, pytest-django CI & Gu...

MODULE 15 OF 16 • STAGE 4 • 14 LIVE + LAB HOURS • 3 TOPICS (24 SUBTOPICS) • 1 CAPSTONE PROJECT (100 MARKS)

High-Scale Django 5 & DRF: Query Optimization, Transactions & Celery

Self-explanatory list-and-bullet production mastery module covering High-Scale Django 5 & DRF: Query Optimization, Transactions & Celery in Django Training, paired with hands-on labs, capstone repository, Topic Mock Tests, and Spoken MockAttempt AI Viva.

Objectives: ✓ Eliminating ORM N+1 Queries with select_related, prefetch_related & Subquery/Exists — Purpose: Cut API SQL query counts from hundreds down to 2–3 indexed queries verified via django-debug-toolbar. • ✓ transaction.atomic(), select_for_update() Row Locking & transaction.on_commit() — Purpose: Prevent race conditions on inventory/wallet updates and enqueue Celery tasks safely post-commit. • ✓ Idempotent Celery Workers, Retries, Periodic Beat Schedules & Redis Caching — Purpose: Run reliable background workflows and cache expensive DRF serializer responses. • ✓ Build, verify, and document 'Zero-N+1 Django REST Service with Atomic Ledger & Celery Beat Automation' with clean Git commits and edge-case protection. • ✓ Clear 100% of Topic Timed Mock Tests and Spoken Voice + MockAttempt AI Technical Interviews.

15.1 Eliminating ORM N+1 Queries with select_related, prefetch_related & Subquery/Exists [65 mins • Assignment (100M) • Hands-On Lab (50M) • 2 Mock Sets • 2 AI Vivas]

Subtopics Covered: Security Hygiene & Data Safety Rules for Eliminating ORM N+1 Queries with select_related, prefetch_related & Subque... • Performance Optimization, Complexity & Latency Impact of Eliminating ORM N+1 Queries with select_related, prefetch_... • Live Debugging, Stack • Runnable Code Implementation (PYTHON) • Hands • Idiomatic DJNG Implementation: select_related/prefetch_related QuerySets, DRF ViewSets & atomic transactions • Django 5 ORM/ASGI Pipeline, PostgreSQL & Celery Workers Execution & Memory Lifecycle for Eliminating ORM N+1 Queries with select_related • Production Hardening & Security: DRF object-level permissions, SimpleJWT rotation & CSRF/SQLi guards

Purpose: To understand and apply Eliminating ORM N+1 Queries with select_related, prefetch_related & Subquery/Exists in real Django systems — specifically: Cut API SQL query count...

What Topic Is Next → Next Topic 15.2: transaction.atomic(), select_for_update() Row Locking & transaction.on_commit()

15.2 transaction.atomic(), select_for_update() Row Locking & transaction.on_commit() [65 mins • Assignment (100M) • Hands-On Lab (50M) • 2 Mock Sets • 2 AI Vivas]

Subtopics Covered: Security Hygiene & Data Safety Rules for transaction.atomic(), select_for_update() Row Locking & transaction.on_com... • Performance Optimization, Complexity & Latency Impact of transaction.atomic(), select_for_update() Row Locking & tr... • Live Debugging, Stack • Runnable Code Implementation (PYTHON) • Hands • Idiomatic DJNG Implementation: select_related/prefetch_related QuerySets, DRF ViewSets & atomic transactions • Django 5 ORM/ASGI Pipeline, PostgreSQL & Celery Workers Execution & Memory Lifecycle for transaction.atomic() • Production Hardening & Security: DRF object-level permissions, SimpleJWT rotation & CSRF/SQLi guards

Purpose: To understand and apply transaction.atomic(), select_for_update() Row Locking & transaction.on_commit() in real Django systems — specifically: Prevent race conditions on...

What Topic Is Next → Next Topic 15.3: Idempotent Celery Workers, Retries, Periodic Beat Schedules & Redis Caching

15.3 Idempotent Celery Workers, Retries, Periodic Beat Schedules & Redis Caching [65 mins • Assignment (100M) • Hands-On Lab (50M) • 2 Mock Sets • 2 AI Vivas]

Subtopics Covered: Security Hygiene & Data Safety Rules for Idempotent Celery Workers, Retries, Periodic Beat Schedules & Redis Cachin... • Performance Optimization, Complexity & Latency Impact of Idempotent Celery Workers, Retries, Periodic Beat Schedule... • Live Debugging, Stack • Runnable Code Implementation (PYTHON) • Hands • Idiomatic DJNG Implementation: select_related/prefetch_related QuerySets, DRF ViewSets & atomic transactions • Django 5 ORM/ASGI Pipeline, PostgreSQL & Celery Workers Execution & Memory Lifecycle for Idempotent Celery Workers • Production Hardening & Security: DRF object-level permissions, SimpleJWT rotation & CSRF/SQLi guards

Purpose: To understand and apply Idempotent Celery Workers, Retries, Periodic Beat Schedules & Redis Caching in real Django systems — specifically: Run reliable background workflo...

What Topic Is Next → Module 15 Capstone (Zero-N+1 Django REST Service with Atomic Ledger & Celery Beat Automation) → Next Topic 16.1: Zero-Downtime Django Migrations (SeparateDatabaseAndState) & Custom Indexes

Module 15 Hands-On Capstone Project (100 Marks): Zero-N+1 Django REST Service with Atomic Ledger & Celery Beat Automation — Architect, implement, test, and document 'Zero-N+1 Django REST Service with Atomic Ledger & Celery Beat Automation' applying every bullet-wise engineering concept from Module 15 (High-Scale...

MODULE 16 OF 16 • STAGE 4 • 14 LIVE + LAB HOURS • 3 TOPICS (24 SUBTOPICS) • 1 CAPSTONE PROJECT (100 MARKS)

Django 5 Async Views, Security Checklist & Senior Backend Interviews

Self-explanatory list-and-bullet production mastery module covering Django 5 Async Views, Security Checklist & Senior Backend Interviews in Django Training, paired with hands-on labs, capstone repository, Topic Mock Tests, and Spoken MockAttempt AI Viva.

Objectives: ✓ Zero-Downtime Django Migrations (SeparateDatabaseAndState) & Custom Indexes — Purpose: Evolve large production PostgreSQL tables safely without breaking running web workers. • ✓ Django Production Security Audit: ALLOWED_HOSTS, HSTS, CSRF, Permissions & Throttling — Purpose: Enforce object-level DRF permissions, scoped rate throttles, and full HTTPS security posture. • ✓ Django MVT/DRF Architecture Trade-Offs & Python System Design Interviews — Purpose: Defend serializer optimization, multi-tenant scoping, and scaling strategies in interviews. • ✓ Build, verify, and document 'Hardened ASGI Django Deployment (`manage.py check --deploy`) & pytest-django Suite' with clean Git commits and edge-case protection. • ✓ Clear 100% of Topic Timed Mock Tests and Spoken Voice + MockAttempt AI Technical Interviews.

16.1 Zero-Downtime Django Migrations (SeparateDatabaseAndState) & Custom Indexes [65 mins • Assignment (100M) • Hands-On Lab (50M) • 2 Mock Sets • 2 AI Vivas]

Subtopics Covered: Security Hygiene & Data Safety Rules for Zero • Performance Optimization, Complexity & Latency Impact of Zero • Live Debugging, Stack • Runnable Code Implementation (PYTHON) • Hands • Idiomatic DJNG Implementation: select_related/prefetch_related QuerySets, DRF ViewSets & atomic transactions • Django 5 ORM/ASGI Pipeline, PostgreSQL & Celery Workers Execution & Memory Lifecycle for Zero-Downtime Django Migrations (SeparateDatabaseAnd... • Production Hardening & Security: DRF object-level permissions, SimpleJWT rotation & CSRF/SQLi guards

Purpose: To understand and apply Zero-Downtime Django Migrations (SeparateDatabaseAndState) & Custom Indexes in real Django systems — specifically: Evolve large production Postgre...

What Topic Is Next → Next Topic 16.2: Django Production Security Audit: ALLOWED_HOSTS, HSTS, CSRF, Permissions & Throttling

16.2 Django Production Security Audit: ALLOWED_HOSTS, HSTS, CSRF, Permissions & Throttling [65 mins • Assignment (100M) • Hands-On Lab (50M) • 2 Mock Sets • 2 AI Vivas]

Subtopics Covered: Security Hygiene & Data Safety Rules for Django Production Security Audit • Performance Optimization, Complexity & Latency Impact of Django Production Security Audit • Live Debugging, Stack • Runnable Code Implementation (PYTHON) • Hands • Idiomatic DJNG Implementation: select_related/prefetch_related QuerySets, DRF ViewSets & atomic transactions • Django 5 ORM/ASGI Pipeline, PostgreSQL & Celery Workers Execution & Memory Lifecycle for Django Production Security Audit • Production Hardening & Security: DRF object-level permissions, SimpleJWT rotation & CSRF/SQLi guards

Purpose: To understand and apply Django Production Security Audit: ALLOWED_HOSTS, HSTS, CSRF, Permissions & Throttling in real Django systems — specifically: Enforce object-level...

What Topic Is Next → Next Topic 16.3: Django MVT/DRF Architecture Trade-Offs & Python System Design Interviews

16.3 Django MVT/DRF Architecture Trade-Offs & Python System Design Interviews [65 mins • Assignment (100M) • Hands-On Lab (50M) • 2 Mock Sets • 2 AI Vivas]

Subtopics Covered: Security Hygiene & Data Safety Rules for Django MVT/DRF Architecture Trade • Performance Optimization, Complexity & Latency Impact of Django MVT/DRF Architecture Trade • Live Debugging, Stack • Runnable Code Implementation (PYTHON) • Hands • Idiomatic DJNG Implementation: select_related/prefetch_related QuerySets, DRF ViewSets & atomic transactions • Django 5 ORM/ASGI Pipeline, PostgreSQL & Celery Workers Execution & Memory

Lifecycle for Django MVT/DRF Architecture Trade-Offs • Production Hardening & Security: DRF object-level permissions, SimpleJWT rotation & CSRF/SQLi guards

Purpose: To understand and apply Django MVT/DRF Architecture Trade-Offs & Python System Design Interviews in real Django systems — specifically: Defend serializer optimization, mu...

What Topic Is Next → Module 16 Capstone → End-of-Course Live Client Project: EduOps Multi-Role Enterprise Portal, REST API & Celery Notification Platform

Module 16 Hands-On Capstone Project (100 Marks): Hardened ASGI Django Deployment (`manage.py check --deploy`) & pytest-django Suite — Architect, implement, test, and document 'Hardened ASGI Django Deployment (`manage.py check --deploy`) & pytest-django Suite' applying every bullet-wise engineering concept from Module 16 (D...

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

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

Project: EduOps Multi-Role Enterprise Portal, REST API & Celery Notification Platform • **Confirmed Role:** Python Django 5 & DRF Full-Lifecycle Backend Intern

Domain & Production Infra: Corporate Compliance & Workforce Operations Client (Django 5 + Django REST Framework + PostgreSQL + Redis + Celery + Gunicorn HTTPS)

- Designed custom user models, granular object permissions, and optimized QuerySets (select_related / prefetch_related).
- Developed Django REST Framework ViewSets, nested serializers, SimpleJWT authentication, and custom throttling.
- Offloaded PDF invoice generation and transactional emails to asynchronous Celery + Redis worker queues.
- Hardened CSRF/XSS/SQLi defenses and deployed production Django services behind Nginx TLS.

Aligned Global Vendor Certifications: **PCAP-31-03** (PCAP™ – Certified Associate in Python Programming) • **PCPP1-32-101** (PCPP1™ – Certified Professional in Python Programming 1) • **EDB-PG16** (EDB Certified PostgreSQL 16 Associate)

Target Job Roles: Django Backend / Full-Stack Developer, Python & Django REST (DRF) Engineer, Software Engineer — Python Web, Backend Product Developer

Realistic Annual CTC Scope: **Fresher / Entry-Level (0–1 Yr):** ₹5.0 LPA – ₹9.2 LPA **Mid-Level Django / DRF Product Engineer (1–3 Yrs):** ₹9.5 LPA – ₹19.0 LPA **Senior Python / Full-Stack Architect (3–6+ Yrs):** ₹19.0 LPA – ₹36.0+ LPA

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

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