Admissions & Support:
support@mockattempt.com
Phone / WhatsApp: +91 95400 07917
Course Code: FAPI • **Updated:** 05 Nov 2026
https://mockattempt.com/courses/fastapi-training

FAPI • ASYNC PYTHON APIS & MICROSERVICES | **BEGINNER TO PRINCIPAL FASTAPI ARCHITECT**

100% LIVE INDUSTRY EXPERT COHORT + INTERNSHIP LETTER

FastAPI Training

Master Python FastAPI from Starlette ASGI Event Loops, Path/Query/Body Schemas & Rust-powered Pydantic v2 Validations to Response Model Secret Filtering, Dependency Injection (`yield`), OAuth2 JWT Security, Middleware, CORS, SQLAlchemy 2.0 & APIRouter Microservices — with runnable Python 3.12 programs, 244 Topic Mock Tests (10,440 MCQs), and Spoken Voice + MockAttempt AI Interviews at ₹2,999 (Incl. 18% GST).

Duration: 15–17 Weeks • Live + Hands-On Projects (205 Hrs) • **Modules:** 16 • **Live Topics:** 42 • **Subtopics:** 336+ • **Labs & Tasks:** 84 • **Mocks / AI Vivas:** 244 / 244 • **Projects:** 16 Module + 1 Live Client Prod

All-Inclusive Direct Fee (Incl. 18% GST • Zero Hidden Charges): Total Payable: ₹2,999 (Net Tuition ₹2,542 + Included 18% GST ₹457) • **Split Option:** 100% Credit to Job Track • **Effective ROI:** ~₹15/Guided Hr (~₹71/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 & FastAPI • Starlette ASGI & Uvicorn / uvloop • Pydantic v2 (pydantic-core Rust) • Typing Annotated & Dependency Injection • OAuth2 Bearer JWT & Passlib/Argon2 • SQLAlchemy 2.0 Async & Alembic • APIRouter, Pytest & HTTPX TestClient

1. PROGRAM MASTERY OUTCOMES & ELIGIBILITY

End-to-End Engineering Outcomes:

1. Architect non-blocking ASGI services on Uvicorn/Starlette understanding async def event-loop coroutines versus def worker threadpools
2. Enforce ironclad request/response contracts with Rust-accelerated Pydantic v2 models, Annotated Query/Path/Field constraints, and response_model secret stripping
3. Compose modular, testable microservices using FastAPI Depends() with yield cleanup, OAuth2PasswordBearer JWT security, CORS, and SQLAlchemy 2.0 sessions
4. Clear 244 Timed Mock Exams (10,000 FastAPI MCQs) and 200 Spoken Voice + MockAttempt AI Python FastAPI Interviews
5. True `async/await` request pipelines using FastAPI routers, lifespan events, and modular dependency injection (`Depends`).
6. Crack LeetCode and HackerRank problems in Python 3.12 with live trainer mentorship on algorithmic/query patterns, edge cases, and optimal complexity.
7. Ultra-fast Pydantic v2 Rust-core data validation, custom validators, discriminated unions, and `pydantic-settings`.
8. Async SQLAlchemy 2.0 + `asyncpg` non-blocking database pools, Alembic migrations, and OAuth2/JWT security scopes.

Eligibility & Prerequisites: Basic Python 3 syntax, functions, and type hints (`str`, `int`, `list`, `dict`) — covers async/await, ASGI, and Pydantic v2 from scratch • Any laptop/PC running Python 3.10+, `fastapi[standard]`, Uvicorn, and VS Code / PyCharm • No prior FastAPI 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 • 42 LIVE TOPICS • 336+ SUBTOPICS)

Progressive Zero-to-Expert Mastery Stages: Modules 1–4: Stage 1: Day-1 Foundations & Core Mental Models (8 Topics / 64 Subtopics) • Modules 5–8: Stage 2: Applied Engineering, Data & API Workflows (8 Topics / 64 Subtopics) • Modules 9–12: Stage 3: Internals, Concurrency, Security & Performance (12 Topics / 96

Subtopics) • Modules 13–16: Stage 4: Production Architecture, Testing, CI/CD & Expert Mastery (14 Topics / 112 Subtopics)

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

Module 1: ASGI Architecture, Uvicorn Event Loop & Typed Path Parameters

Boot high-concurrency FastAPI applications on Starlette/Uvicorn, generate OpenAPI 3.1 Swagger/ReDoc specs automatically, and parse typed/Enum path parameters.

Objectives: ✓ FastAPI First Steps — Starlette ASGI Core, Uvicorn Event Loop & OpenAPI 3.1 Auto-Docs — How FastAPI unites Starlette ASGI routing with Pydantic v2 type introspection to deliver Node/Go-class throughput and zero-config OpenAPI 3.1 Swagger UI. • ✓ FastAPI Path Parameters — Type Coercion, Route Declaration Order, StrEnum & Path Converters — Extracting typed URL segments (`int`, `UUID`, `StrEnum`, `{file_path:path}`) with automatic Pydantic v2 parsing, coercion, and structured 422 Unprocessable Entity errors. • ✓ Build, verify, and ship Module Capstone: OpenAPI-Documented Service Health & Enum-Constrained Catalog Router.

1.1 FastAPI First Steps — Starlette ASGI Core, Uvicorn Event Loop & OpenAPI 3.1 Auto-Docs

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

Subtopics Covered: ASGI vs legacy synchronous WSGI & Starlette + Pydantic v2 foundation • Core Syntax, CLI & Production APIs: app = FastAPI() + @app.get('/') + TestClient(app) • Deep-Dive Mechanics: Async def vs def threadpool execution & OpenAPI 3.1 /openapi.json generation • Advanced Patterns: Swagger UI (/docs) and ReDoc (/redoc) & Uvicorn uvloop server architecture • Production Failure Mode & Pitfall Guard • Hands-On Live Coding Lab: Minimal Async FastAPI App & TestClient Verification • Idiomatic FAPI Implementation: async/await non-blocking endpoints, typed Pydantic contracts & grounded RAG • Asyncio Event Loop, FastAPI ASGI, Pydantic v2 & pgvector/PyTorch Execution & Memory Lifecycle for FastAPI First Steps

Purpose: To understand and apply FastAPI First Steps — Starlette ASGI Core, Uvicorn Event Loop & OpenAPI 3.1 Auto-Docs in real FastAPI systems — specifically: How FastAPI unites S...

What Topic Is Next → Next Topic 1.2: FastAPI Path Parameters — Type Coercion, Route Declaration Order, StrEnum & Path Converters

1.2 FastAPI Path Parameters — Type Coercion, Route Declaration Order, StrEnum & Path Converters

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

Subtopics Covered: Typed URL path coercion & Top-to-bottom route matching order • Core Syntax, CLI & Production APIs: @app.get('/courses/{tier}/{course_id}') + class Tier(StrEnum) • Deep-Dive Mechanics: StrEnum predefined path values & Starlette :path slash converter • Advanced Patterns: Automatic HTTP 422 validation errors & OpenAPI path parameter schema • Production Failure Mode & Pitfall Guard • Hands-On Live Coding Lab: Route Ordering & StrEnum Path Validation Program • Idiomatic FAPI Implementation: async/await non-blocking endpoints, typed Pydantic contracts & grounded RAG • Asyncio Event Loop, FastAPI ASGI, Pydantic v2 & pgvector/PyTorch Execution & Memory Lifecycle for FastAPI Path Parameters

Purpose: To understand and apply FastAPI Path Parameters — Type Coercion, Route Declaration Order, StrEnum & Path Converters in real FastAPI systems — specifically: Extracting typ...

What Topic Is Next → Module 1 Capstone (OpenAPI-Documented Service Health & Enum-Constrained Catalog Router) → Next Topic 2.1: FastAPI Query Parameters — Defaults, Optional Unions (`str | None`), Booleans & Required Flags

Module 1 Hands-On Capstone Project (100 Marks): OpenAPI-Documented Service Health & Enum-Constrained Catalog Router — Create a FastAPI service exposing async health metadata and typed/Enum resource routes where static paths resolve safely before dynamic {item_id} catch-all segments.

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

Module 2: Query Parameters & Pydantic v2 Request Body Parsing

Combine required/optional query parameters with Rust-backed Pydantic v2 BaseModel JSON payloads for clean pagination, filtering, and creation endpoints.

Objectives: ✓ FastAPI Query Parameters — Defaults, Optional Unions (`str | None`), Booleans & Required Flags — How function parameters not present in the URL path template are automatically bound from URL query strings (`?skip=0&limit=20&published=true`) with smart boolean coercion. • ✓ FastAPI Request Body — Pydantic v2 `BaseModel`, `model_dump()` & Path + Query + Body Mixing — Validating incoming JSON payloads into strongly typed Python objects via Pydantic v2 (`pydantic-core` in Rust) while seamlessly mixing URL path and query parameters. • ✓ Build, verify, and ship Module Capstone: Paginated Search & Pydantic v2 Course Enrollment Endpoint.

2.1 FastAPI Query Parameters — Defaults, Optional Unions (`str | None`), Booleans & Required Flags

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

Subtopics Covered: Implicit query parameter resolution & Required vs default-valued parameters • Core Syntax, CLI & Production APIs: def list_items(skip: int = 0, limit: int = 10, tag: str | None = None) • Deep-Dive Mechanics: Python 3.10+ str | None = None union & Smart bool coercion (1, true, on, yes) • Advanced Patterns: Combining path and query parameters & 422 missing query error payload • Production Failure Mode & Pitfall Guard • Hands-On Live Coding Lab: Typed Pagination & Boolean Query Coercion Endpoint • Idiomatic FAPI Implementation: async/await non-blocking endpoints, typed Pydantic contracts & grounded RAG • Asyncio Event Loop, FastAPI ASGI, Pydantic v2 & pgvector/PyTorch Execution & Memory Lifecycle for FastAPI Query Parameters

Purpose: To understand and apply FastAPI Query Parameters — Defaults, Optional Unions (`str | None`), Booleans & Required Flags in real FastAPI systems — specifically: How functio...

What Topic Is Next → Next Topic 2.2: FastAPI Request Body — Pydantic v2 `BaseModel`, `model_dump()` & Path + Query + Body Mixing

2.2 FastAPI Request Body — Pydantic v2 `BaseModel`, `model_dump()` & Path + Query + Body Mixing

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

Subtopics Covered: Pydantic v2 BaseModel inheritance & Rust pydantic-core validation engine • Core Syntax, CLI & Production APIs: class Payload(BaseModel): ... + payload.model_dump(exclude_unset=True) • Deep-Dive Mechanics: Model_dump(exclude_unset=True) PATCH pattern & Simultaneous Path + Query + Body binding • Advanced Patterns: Automatic JSON Schema requestBody docs & Type-safe IDE autocomplete • Production Failure Mode & Pitfall Guard • Hands-On Live Coding Lab: Mixed Path, Query & Pydantic v2 Request Body Handler • Idiomatic FAPI Implementation: async/await non-blocking endpoints, typed Pydantic contracts & grounded RAG • Asyncio Event Loop, FastAPI ASGI, Pydantic v2 & pgvector/PyTorch Execution & Memory Lifecycle for FastAPI Request Body

Purpose: To understand and apply FastAPI Request Body — Pydantic v2 `BaseModel`, `model_dump()` & Path + Query + Body Mixing in real FastAPI systems — specifically: Validating inc...

What Topic Is Next → Module 2 Capstone (Paginated Search & Pydantic v2 Course Enrollment Endpoint) → Next Topic 3.1: FastAPI Query Validations — `Annotated[str, Query()]`, `min_length`, `pattern`, Lists & Aliases

Module 2 Hands-On Capstone Project (100 Marks): Paginated Search & Pydantic v2 Course Enrollment Endpoint — Build an API endpoint suite supporting offset/limit/boolean query filtering alongside a Pydantic v2 BaseModel POST endpoint that computes net discounted pricing.

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

Module 3: `Annotated` Metadata & Declarative Query / Path Validations

Adopt modern `typing.Annotated` syntax with `Query(...)` and `Path(...)` to enforce string length, regex patterns, multi-value lists, and numeric `ge`/`le`/`gt`/`lt` bounds.

Objectives: ✓ FastAPI Query Validations — `Annotated[str, Query()]`, `min_length`, `pattern`, Lists & Aliases — Enforcing declarative length limits, regex patterns, repeated list parameters (`?tag=py&tag=api`), and hyphenated query aliases cleanly via `Annotated` and `Query()`. • ✓ FastAPI Path Validations — `Annotated[int, Path(gt=0, le=10000)]` & Numeric Boundary Rules — Constraining URL path IDs and numeric route parameters with `gt`, `ge`, `lt`, and `le` rules while eliminating Python default-argument ordering conflicts via `Annotated`. • ✓ Build, verify, and ship Module Capstone: Hardened Search Filter & Numeric Boundary Guard API.

3.1 FastAPI Query Validations — `Annotated[str, Query()]`, `min_length`, `pattern`, Lists & Aliases

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

Subtopics Covered: Typing.Annotated metadata pattern & Min_length and max_length guards • Core Syntax, CLI & Production APIs: Annotated[str | None, Query(min_length=3, max_length=40, pattern=r'^[a-z0-9-]+$', alias='ite... • Deep-Dive Mechanics: Query(pattern=...) regex validation & Multi-value list[str] query binding • Advanced Patterns: Hyphenated query alias mapping & Deprecated=True OpenAPI parameter flag • Production Failure Mode & Pitfall Guard • Hands-On Live Coding Lab: Annotated Regex, Alias & Multi-Tag Query Validator • Idiomatic FAPI Implementation: async/await non-blocking endpoints, typed Pydantic contracts & grounded RAG • Asyncio Event Loop, FastAPI ASGI, Pydantic v2 & pgvector/PyTorch Execution & Memory Lifecycle for FastAPI Query Validations

Purpose: To understand and apply FastAPI Query Validations — `Annotated[str, Query()]`, `min_length`, `pattern`, Lists & Aliases in real FastAPI systems — specifically: Enforcing...

What Topic Is Next → Next Topic 3.2: FastAPI Path Validations — `Annotated[int, Path(gt=0, le=10000)]` & Numeric Boundary Rules

3.2 FastAPI Path Validations — `Annotated[int, Path(gt=0, le=10000)]` & Numeric Boundary Rules

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

Subtopics Covered: Path(gt, ge, lt, le) numeric bounds & Annotated parameter-order freedom • Core Syntax, CLI & Production APIs: item_id: Annotated[int, Path(title='Item ID', ge=1, le=100_000)] • Deep-Dive Mechanics: Always-required URL segment invariant & Float gt/lt boundary constraints • Advanced Patterns: OpenAPI parameter title and description & Early 422 rejection before DB hit • Production Failure Mode & Pitfall Guard • Hands-On Live Coding Lab: Numeric Path Boundary Guard (`ge=1`, `le=500`) • Idiomatic FAPI Implementation: async/await non-blocking endpoints, typed Pydantic contracts & grounded RAG • Asyncio Event Loop, FastAPI ASGI, Pydantic v2 & pgvector/PyTorch Execution & Memory Lifecycle for FastAPI Path Validations

Purpose: To understand and apply FastAPI Path Validations — `Annotated[int, Path(gt=0, le=10000)]` & Numeric Boundary Rules in real FastAPI systems — specifically: Constraining UR...

What Topic Is Next → Module 3 Capstone (Hardened Search Filter & Numeric Boundary Guard API) → Next Topic 4.1: FastAPI Body Fields — `pydantic.Field()`, `default_factory`, `ConfigDict` & OpenAPI Examples

Module 3 Hands-On Capstone Project (100 Marks): Hardened Search Filter & Numeric Boundary Guard API — Build an `Annotated` catalog filter endpoint enforcing regex SKU codes, bounded page sizes (`ge=1, le=100`), aliased query keys (`item-tag`), and positive integer resource IDs (`gt=0`).

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

Module 4: Rich Pydantic v2 `Field()` Constraints & Deeply Nested JSON Models

Model complex enterprise JSON trees using `pydantic.Field()`, `ConfigDict(extra="forbid")`, `HttpUrl`, `set[str]` deduplication, and deeply nested sub-models.

Objectives: ✓ FastAPI Body Fields — `pydantic.Field()`, `default_factory`, `ConfigDict` & OpenAPI Examples — Enforcing attribute-level numeric/string validation, safe mutable defaults (`default_factory`), strict unknown-field rejection (`extra="forbid"`), and rich OpenAPI schema examples inside Pydantic models. • ✓ FastAPI Nested Models — Sub-Models, `list[SubModel]`, `set[str]` Deduplication & `HttpUrl` — Modeling arbitrarily deep JSON trees with nested `BaseModel` compositions, automatic `set[str]` duplicate elimination, `HttpUrl` parsing, and exact error-path reporting. • ✓ Build, verify, and ship Module Capstone: Nested E-Commerce Checkout Schema with Strict Field & Cross-Field Validation.

4.1 FastAPI Body Fields — `pydantic.Field()`, `default_factory`, `ConfigDict` & OpenAPI Examples

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

Subtopics Covered: Pydantic.Field() attribute constraints & Field(default_factory=...) safe defaults • Core Syntax, CLI & Production APIs: Field(gt=0, max_digits=10) + ConfigDict(extra='forbid', str_strip_whitespace=True) • Deep-Dive Mechanics: ConfigDict(extra='forbid') hardening & Str_strip_whitespace normalization • Advanced Patterns: Body(embed=True) single-key wrapper & Examples OpenAPI schema enrichment • Production Failure Mode & Pitfall Guard • Hands-On Live Coding Lab: Strict `ConfigDict(extra="forbid")` & `Field()` Payload Guard • Idiomatic FAPI Implementation: async/await non-blocking endpoints, typed Pydantic contracts & grounded RAG • Asyncio Event Loop, FastAPI ASGI, Pydantic v2 & pgvector/PyTorch Execution & Memory Lifecycle for FastAPI Body Fields

Purpose: To understand and apply FastAPI Body Fields — `pydantic.Field()`, `default_factory`, `ConfigDict` & OpenAPI Examples in real FastAPI systems — specifically: Enforcing att...

What Topic Is Next → Next Topic 4.2: FastAPI Nested Models — Sub-Models, `list[SubModel]`, `set[str]` Deduplication & `HttpUrl`

4.2 FastAPI Nested Models — Sub-Models, `list[SubModel]`, `set[str]` Deduplication & `HttpUrl` [50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Subtopics Covered: Composed BaseModel sub-hierarchies & Typed list[SubModel] collection validation • Core Syntax, CLI & Production APIs: class CourseBundle(BaseModel): tags: set[str] = set(); lessons: list[LessonSpec] • Deep-Dive Mechanics: Set[str] automatic deduplication & Pydantic HttpUrl special network type • Advanced Patterns: Deep JSON loc error pinpointing & Top-level list[Model] request bodies • Production Failure Mode & Pitfall Guard • Hands-On Live Coding Lab: Deeply Nested Curriculum Model with `set[str]` Tag Deduplication • Idiomatic FAPI Implementation: async/await non-blocking endpoints, typed Pydantic contracts & grounded RAG • Asyncio Event Loop, FastAPI ASGI, Pydantic v2 & pgvector/PyTorch Execution & Memory Lifecycle for FastAPI Nested Models

Purpose: To understand and apply FastAPI Nested Models — Sub-Models, `list[SubModel]`, `set[str]` Deduplication & `HttpUrl` in real FastAPI systems — specifically: Modeling arbitr...

What Topic Is Next → Module 4 Capstone (Nested E-Commerce Checkout Schema with Strict Field & Cross-Field Validation) → Next Topic 5.1: FastAPI Cookie & Header Parameters — Underscore-to-Hyphen Conversion, Duplicate Headers & Models

Module 4 Hands-On Capstone Project (100 Marks): Nested E-Commerce Checkout Schema with Strict Field & Cross-Field Validation — Design a nested Pydantic v2 Order payload containing CustomerAddress, list[LineItem], unique set[str] promo tags, and Field(gt=0) quantities with extra="forbid" mass-assignment protection.

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

Module 5: Cookie / Header Extraction & `response_model` Output Filtering

Extract HTTP Cookies and hyphenated Headers cleanly (including Pydantic header/cookie models) and guarantee zero password-hash leaks using `response_model` output schemas.

Objectives: ✓ FastAPI Cookie & Header Parameters — Underscore-to-Hyphen Conversion, Duplicate Headers & Models — Extracting typed HTTP `Cookie()` and `Header()` values (including `X-Token` hyphen mapping, multi-value header lists, and grouped Pydantic Header/Cookie models). • ✓ FastAPI `response_model` — Secret Stripping, Input/Output DTO Separation & `exclude_unset` — Using `@app.post(..., response_model=UserPublic)` to validate, serialize, document, and strip sensitive fields (`password_hash`, internal tokens) before JSON bytes leave the server. • ✓ Build, verify, and ship Module Capstone: Header-Traced User Registration API with Zero-Leak `response_model` Separation.

5.1 FastAPI Cookie & Header Parameters — Underscore-to-Hyphen Conversion, Duplicate Headers & Models [50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Subtopics Covered: Header() underscore-to-hyphen mapping & Case-insensitive RFC-9110 HTTP headers • Core Syntax, CLI & Production APIs: x_trace_id: Annotated[str | None, Header()] = None + session_id: Annotated[str | None, Cooki... • Deep-Dive Mechanics: Cookie() typed extraction & Multi-value list[str] duplicate headers • Advanced Patterns: Pydantic BaseModel for grouped Headers/Cookies & Convert_underscores=False opt-out • Production Failure Mode & Pitfall Guard • Hands-On Live Coding Lab: Typed `Header()` Hyphen Mapping & `Cookie()` Extractor • Idiomatic FAPI Implementation: async/await non-blocking endpoints, typed Pydantic contracts & grounded RAG • Asyncio Event Loop, FastAPI ASGI, Pydantic v2 & pgvector/PyTorch Execution & Memory Lifecycle for FastAPI Cookie

Purpose: To understand and apply FastAPI Cookie & Header Parameters — Underscore-to-Hyphen Conversion, Duplicate Headers & Models in real FastAPI systems — specifically: Extractin...

What Topic Is Next → Next Topic 5.2: FastAPI `response_model` — Secret Stripping, Input/Output DTO Separation & `exclude_unset`

5.2 FastAPI `response_model` — Secret Stripping, Input/Output DTO Separation & `exclude_unset` [50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Subtopics Covered: Outbound secret-stripping firewall & UserIn vs UserInDB vs UserPublic split • Core Syntax, CLI & Production APIs: @app.post('/users', response_model=UserPublic, response_model_exclude_unset=True) • Deep-Dive Mechanics: Response_model priority over return annotation & Response_model_exclude_unset=True • Advanced Patterns: ConfigDict(from_attributes=True) ORM mode & Response union / RedirectResponse return types • Production Failure Mode & Pitfall Guard • Hands-On Live Coding Lab: Zero-Leak `response_model` Password Stripping Verification • Idiomatic FAPI Implementation: async/await non-blocking endpoints, typed Pydantic contracts & grounded RAG • Asyncio Event Loop, FastAPI ASGI, Pydantic v2 & pgvector/PyTorch Execution & Memory Lifecycle for FastAPI `response_model`

Purpose: To understand and apply FastAPI `response_model` — Secret Stripping, Input/Output DTO Separation & `exclude_unset` in real FastAPI systems — specifically: Using `@app.pos...

What Topic Is Next → Module 5 Capstone (Header-Traced User Registration API with Zero-Leak `response_model` Separation) → Next Topic 6.1: FastAPI Response Status Codes — `fastapi.status` Constants, `201 Created`, `204 No Content` & Dynamic Status

Module 5 Hands-On Capstone Project (100 Marks): Header-Traced User Registration API with Zero-Leak `response_model` Separation — Implement a registration endpoint that reads X-Correlation-Id headers and session cookies while guaranteeing hashed_password and internal audit fields never leave the HTTP response.

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

Module 6: Semantic HTTP Status Codes, Multipart Forms & Streaming File Uploads

Return RFC-9110 semantic HTTP status codes (`status.HTTP_201_CREATED`, `204_NO_CONTENT`) and process `Form()` fields + spooled `UploadFile` streams safely without RAM exhaustion.

Objectives: ✓ FastAPI Response Status Codes — `fastapi.status` Constants, `201 Created`, `204 No Content` & Dynamic Status — Communicating exact REST semantics using `status_code=status.HTTP_201_CREATED`, `204` empty-body deletions, and dynamic upsert status codes (`200` vs `201`) via `Response`. • ✓ FastAPI Forms & Files — `Form()`, `UploadFile`, `SpooledTemporaryFile` & Multipart Streaming — Handling `multipart/form-data` form fields alongside async `UploadFile` spooled file handles (`python-multipart`) without exhausting worker RAM. • ✓ Build, verify, and ship Module Capstone: Chunk-Streamed Resume Uploader with MIME & Size Quota Enforcement.

6.1 FastAPI Response Status Codes — `fastapi.status` Constants, `201 Created`, `204 No Content` & Dynamic Status [50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Subtopics Covered: Fastapi.status named constants & 201 Created resource creation contract • Core Syntax, CLI & Production APIs: @app.post('/items', status_code=status.HTTP_201_CREATED) + response.status_code = ... • Deep-Dive Mechanics: 204 No Content bodyless DELETE rule & 202 Accepted async task acknowledgement • Advanced Patterns: Dynamic Response.status_code override & OpenAPI status code documentation • Production Failure Mode & Pitfall Guard • Hands-On Live Coding Lab: Dynamic Upsert (`201` vs `200`) & `204` Delete Status Controller • Idiomatic FAPI Implementation: async/await non-blocking endpoints, typed Pydantic contracts & grounded RAG • Asyncio Event Loop, FastAPI ASGI, Pydantic v2 & pgvector/PyTorch Execution & Memory Lifecycle for FastAPI Response Status Codes

Purpose: To understand and apply FastAPI Response Status Codes — `fastapi.status` Constants, `201 Created`, `204 No Content` & Dynamic Status in real FastAPI systems — specificall...

What Topic Is Next → Next Topic 6.2: FastAPI Forms & Files — `Form()`, `UploadFile`, `SpooledTemporaryFile` & Multipart Streaming

6.2 FastAPI Forms & Files — `Form()`, `UploadFile`, `SpooledTemporaryFile` & Multipart Streaming [50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Subtopics Covered: UploadFile vs raw bytes File() & SpooledTemporaryFile RAM-to-disk spill • Core Syntax, CLI & Production APIs: title: Annotated[str, Form()], doc: UploadFile = File(...) • Deep-Dive Mechanics: Async read/write/seek/close methods & Form() urlencoded and multipart parsing • Advanced Patterns: Python-multipart parser requirement & MIME content_type + byte-quota checks • Production Failure Mode & Pitfall Guard • Hands-On Live Coding Lab: Multipart `Form()` + `UploadFile` Streaming & Size Verifier • Idiomatic FAPI Implementation: async/await non-blocking endpoints, typed Pydantic contracts & grounded RAG • Asyncio Event Loop, FastAPI ASGI, Pydantic v2 & pgvector/PyTorch Execution & Memory Lifecycle for FastAPI Forms

Purpose: To understand and apply FastAPI Forms & Files — `Form()`, `UploadFile`, `SpooledTemporaryFile` & Multipart Streaming in real FastAPI systems — specifically: Handling `mul...

What Topic Is Next → Module 6 Capstone (Chunk-Streamed Resume Uploader with MIME & Size Quota Enforcement) → Next Topic 7.1: FastAPI Error Handling — `HTTPException`, Custom Headers & `@app.exception_handler`

Module 6 Hands-On Capstone Project (100 Marks): Chunk-Streamed Resume Uploader with MIME & Size Quota Enforcement — Build a `201 Created` multipart upload endpoint accepting candidate metadata via `Form()` and a PDF via `UploadFile`, rejecting non-PDF MIME types (`415`) or files exceeding quota (`413`) wi...

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

Module 7: Structured Error Handling & FastAPI Dependency Injection (`Depends` + `yield`)

Return consistent domain error envelopes via `HTTPException` and `@app.exception_handler` while structuring reusable auth, pagination, and `yield` resource lifecycles with `Depends()`.

Objectives: ✓ FastAPI Error Handling — `HTTPException`, Custom Headers & `@app.exception_handler` — Aborting invalid requests cleanly from any route or sub-dependency via `raise HTTPException(...)` and unifying domain/validation errors with custom exception handlers. • ✓ FastAPI Dependencies (`Depends`) — Sub-Dependency DAGs, Per-Request Caching, `yield` & Overrides — Powering authentication, RBAC, DB sessions, and shared pagination through FastAPI's declarative Dependency Injection graph (`Depends`, `yield` cleanup, and `dependency_overrides`). • ✓ Build, verify, and ship Module Capstone: Domain Exception Envelope & `Depends` Resource Lifecycle Framework.

7.1 FastAPI Error Handling — `HTTPException`, Custom Headers & `@app.exception_handler` [50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Subtopics Covered: Raise HTTPException(status_code, detail, headers) & Custom @app.exception_handler registration • Core Syntax, CLI & Production APIs: raise HTTPException(status_code=404, detail=...) + @app.exception_handler(DomainError) • Deep-Dive Mechanics: RequestValidationError custom formatting & Starlette HTTPException vs fastapi HTTPException • Advanced Patterns: Safe 500 error masking without stack leaks & Structured JSON error contracts • Production Failure Mode & Pitfall Guard • Hands-On Live Coding Lab: Custom Domain Exception Handler & `HTTPException` Guard • Idiomatic FAPI Implementation: async/await non-blocking endpoints, typed Pydantic contracts & grounded RAG • Asyncio Event Loop, FastAPI ASGI, Pydantic v2 & pgvector/PyTorch Execution & Memory Lifecycle for FastAPI Error Handling

Purpose: To understand and apply FastAPI Error Handling — `HTTPException`, Custom Headers & `@app.exception_handler` in real FastAPI systems — specifically: Aborting invalid reque...

What Topic Is Next → Next Topic 7.2: FastAPI Dependencies (`Depends`) — Sub-Dependency DAGs, Per-Request Caching, `yield` & Overrides

7.2 FastAPI Dependencies (`Depends`) — Sub-Dependency DAGs, Per-Request Caching, `yield` & Overrides [50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Subtopics Covered: Depends() declarative DI graph & Per-request dependency caching (use_cache=True) • Core Syntax, CLI & Production APIs: CommonsDep = Annotated[dict, Depends(common_params)] + app.dependency_overrides[...] • Deep-Dive Mechanics: Generator dependencies with try/yield/finally & Router-level and app-level dependencies=[...] • Advanced Patterns: Callable class dependencies & App.dependency_overrides test isolation • Production Failure Mode & Pitfall Guard • Hands-On Live Coding Lab: `yield` Lifecycle Dependency & `dependency_overrides` Test Swap • Idiomatic FAPI Implementation: async/await non-blocking endpoints, typed Pydantic contracts & grounded RAG • Asyncio Event Loop, FastAPI ASGI, Pydantic v2 & pgvector/PyTorch Execution & Memory Lifecycle for FastAPI Dependencies (`Depends`)

Purpose: To understand and apply FastAPI Dependencies (`Depends`) — Sub-Dependency DAGs, Per-Request Caching, `yield` & Overrides in real FastAPI systems — specifically: Powering...

What Topic Is Next → Module 7 Capstone (Domain Exception Envelope & `Depends` Resource Lifecycle Framework) → Next Topic 8.1: FastAPI Security — `OAuth2PasswordBearer`, JWT Verification, Argon2 Hashing & RBAC Scopes

Module 7 Hands-On Capstone Project (100 Marks): Domain Exception Envelope & `Depends` Resource Lifecycle Framework — Create a FastAPI service featuring custom domain exception handlers (returning RFC-style error JSON with correlation IDs) and a `yield`-backed database unit-of-work dependency overridable in...

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

Module 8: OAuth2 + JWT Security & ASGI HTTP Middleware

Secure endpoints with `OAuth2PasswordBearer`, Argon2/Bcrypt password hashing, short-lived JWT Bearer tokens, RBAC `Security()` scopes, and non-blocking ASGI Middleware.

Objectives: ✓ FastAPI Security — `OAuth2PasswordBearer`, JWT Verification, Argon2 Hashing & RBAC Scopes — First-class OpenAPI security schemes (`OAuth2PasswordBearer`, `SecurityScopes`, APIKeyHeader) integrated directly into `Depends()` and Swagger UI's interactive Authorize modal. • ✓ FastAPI Middleware — `@app.middleware("http")`, Execution Onion Order & Pure ASGI Middleware — Intercepting every incoming `Request` and outgoing `Response` via `call_next(request)` or pure ASGI middleware (`TrustedHostMiddleware`, `GZipMiddleware`, timing & trace headers). • ✓ Build, verify, and ship Module Capstone: OAuth2 JWT Bearer Auth Guard & Request Telemetry Middleware.

8.1 FastAPI Security — `OAuth2PasswordBearer`, JWT Verification, Argon2 Hashing & RBAC Scopes

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

Subtopics Covered: OAuth2PasswordBearer scheme + Swagger Authorize & OAuth2PasswordRequestForm token endpoint • Core Syntax, CLI & Production APIs: oauth2_scheme = OAuth2PasswordBearer(tokenUrl='token') + Security(get_user, scopes=['admin']... • Deep-Dive Mechanics: JWT exp, sub, and scope claim verification & Argon2id / Bcrypt salted password hashing • Advanced Patterns: SecurityScopes fine-grained RBAC & 401 WWW-Authenticate: Bearer RFC compliance • Production Failure Mode & Pitfall Guard • Hands-On Live Coding Lab: `OAuth2PasswordBearer` Token Extractor & Role Guard • Idiomatic FAPI Implementation: async/await non-blocking endpoints, typed Pydantic contracts & grounded RAG • Asyncio Event Loop, FastAPI ASGI, Pydantic v2 & pgvector/PyTorch Execution & Memory Lifecycle for FastAPI Security

Purpose: To understand and apply FastAPI Security — `OAuth2PasswordBearer`, JWT Verification, Argon2 Hashing & RBAC Scopes in real FastAPI systems — specifically: First-class Open...

What Topic Is Next → Next Topic 8.2: FastAPI Middleware — `@app.middleware("http")`, Execution Onion Order & Pure ASGI Middleware

8.2 FastAPI Middleware — `@app.middleware("http")`, Execution Onion Order & Pure ASGI Middleware

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

Subtopics Covered: @app.middleware('http') call_next pipeline & LIFO onion registration execution order • Core Syntax, CLI & Production APIs: @app.middleware('http') async def add_timing(request: Request, call_next): ... • Deep-Dive Mechanics: Time.perf_counter() latency instrumentation & Pure ASGI middleware vs BaseHTTPMiddleware • Advanced Patterns: TrustedHostMiddleware host-header defense & GZipMiddleware payload compression • Production Failure Mode & Pitfall Guard • Hands-On Live Coding Lab: HTTP Telemetry & Security Header Middleware Verification • Idiomatic FAPI Implementation: async/await non-blocking endpoints, typed Pydantic contracts & grounded RAG • Asyncio Event Loop, FastAPI ASGI, Pydantic v2 & pgvector/PyTorch Execution & Memory Lifecycle for FastAPI Middleware

Purpose: To understand and apply FastAPI Middleware — `@app.middleware("http")`, Execution Onion Order & Pure ASGI Middleware in real FastAPI systems — specifically: Intercepting...

What Topic Is Next → Module 8 Capstone (OAuth2 JWT Bearer Auth Guard & Request Telemetry Middleware) → Next Topic 9.1: FastAPI CORS — `CORSMiddleware`, `OPTIONS` Preflight Handshakes & Credentialed Origin Rules

Module 8 Hands-On Capstone Project (100 Marks): OAuth2 JWT Bearer Auth Guard & Request Telemetry Middleware — Build an OAuth2 Bearer-protected API that verifies signed token claims, enforces role-based access dependencies, and attaches `X-Process-Time-Ms` + `X-Content-Type-Options: nosniff` headers...

MODULE 9 OF 16 • STAGE 3 • 12 LIVE + LAB HOURS • 2 TOPICS (16 SUBTOPICS) • 1 CAPSTONE PROJECT (100 MARKS)

Module 9: Cross-Origin Resource Sharing (`CORSMiddleware`) & SQL Databases (`SQLAlchemy 2.0`)

Configure browser CORS preflight policies safely without credentialed wildcard vulnerabilities and integrate SQLAlchemy 2.0 / SQLModel transactional sessions via `Depends`.

Objectives: ✓ FastAPI CORS — `CORSMiddleware`, `OPTIONS` Preflight Handshakes & Credentialed Origin Rules — Managing browser Same-Origin Policy preflight `OPTIONS` requests, `Access-Control-Allow-Origin`, `allow_credentials`, and `expose_headers` cleanly via `CORSMiddleware`. • ✓ FastAPI SQL Databases — SQLAlchemy 2.0 / SQLModel, `Session` per Request & `from_attributes=True` — Connecting FastAPI to PostgreSQL/SQLite using SQLAlchemy 2.0 `Session` / `AsyncSession` `yield` dependencies, parameterized ORM queries, and `from_attributes=True` DTO projection. • ✓ Build, verify, and ship Module Capstone: CORS-Hardened SPA Backend with SQLAlchemy 2.0 Transactional Repository.

9.1 FastAPI CORS — `CORSMiddleware`, `OPTIONS` Preflight Handshakes & Credentialed Origin Rules

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

Subtopics Covered: Same-Origin Policy (protocol + host + port) & OPTIONS preflight handshake mechanics • Core Syntax, CLI & Production APIs: app.add_middleware(CORSMiddleware, allow_origins=[...], allow_credentials=True, allow_method... • Deep-Dive Mechanics: Allow_origins explicit whitelist vs wildcard & Allow_credentials=True wildcard prohibition • Advanced Patterns: Expose_headers for custom response headers & Max_age preflight browser caching • Production Failure Mode & Pitfall Guard • Hands-On Live Coding Lab: `CORSMiddleware` Preflight `OPTIONS` Verification • Idiomatic FAPI Implementation: async/await non-blocking endpoints, typed Pydantic contracts & grounded RAG • Asyncio Event Loop, FastAPI ASGI, Pydantic v2 & pgvector/PyTorch Execution & Memory Lifecycle for FastAPI CORS

Purpose: To understand and apply FastAPI CORS — `CORSMiddleware`, `OPTIONS` Preflight Handshakes & Credentialed Origin Rules in real FastAPI systems — specifically: Managing brows...

What Topic Is Next → Next Topic 9.2: FastAPI SQL Databases — SQLAlchemy 2.0 / SQLModel, `Session` per Request & `from_attributes=True`

9.2 FastAPI SQL Databases — SQLAlchemy 2.0 / SQLModel, `Session` per Request & `from_attributes=True`

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

Subtopics Covered: Per-request Session yield dependency & SQLAlchemy 2.0 select() parameterized queries • Core Syntax, CLI & Production APIs: SessionDep = Annotated[Session, Depends(get_db)] + ConfigDict(from_attributes=True) • Deep-Dive Mechanics: ConfigDict(from_attributes=True) ORM mapping & AsyncSession +

asyncpg non-blocking I/O • Advanced Patterns: MissingGreenlet lazy-load prevention (selectinload) & Alembic schema migrations in production • Production Failure Mode & Pitfall Guard • Hands-On Live Coding Lab: In-Memory SQLite + `from_attributes=True` FastAPI CRUD Verification • Idiomatic FAPI Implementation: async/await non-blocking endpoints, typed Pydantic contracts & grounded RAG • Asyncio Event Loop, FastAPI ASGI, Pydantic v2 & pgvector/PyTorch Execution & Memory Lifecycle for FastAPI SQL Databases

Purpose: To understand and apply FastAPI SQL Databases — SQLAlchemy 2.0 / SQLModel, `Session` per Request & `from_attributes=True` in real FastAPI systems — specifically: Connecti...

What Topic Is Next → Module 9 Capstone (CORS-Hardened SPA Backend with SQLAlchemy 2.0 Transactional Repository) → Next Topic 10.1: Bigger FastAPI Applications — `APIRouter`, Versioned Prefixes, Router Dependencies & `lifespan`

Module 9 Hands-On Capstone Project (100 Marks): CORS-Hardened SPA Backend with SQLAlchemy 2.0 Transactional Repository — Build a FastAPI CRUD API configured with explicit origin whitelisting on `CORSMiddleware` and backed by an isolated per-request SQLite/SQLAlchemy 2.0 session dependency.

MODULE 10 OF 16 • STAGE 3 • 12 LIVE + LAB HOURS • 2 TOPICS (16 SUBTOPICS) • 1 CAPSTONE PROJECT (100 MARKS)

Module 10: Modular Microservice Architecture (`APIRouter`) & `BackgroundTasks`

Scale large FastAPI codebases cleanly across domain packages using `APIRouter`, lifespan context managers, and non-blocking post-response `BackgroundTasks`.

Objectives: ✓ Bigger FastAPI Applications — `APIRouter`, Versioned Prefixes, Router Dependencies & `lifespan` — Organizing enterprise FastAPI codebases into clean domain slices (`routers/`, `schemas/`, `services/`, `dependencies.py`) using `APIRouter` and `@asynccontextmanager` `lifespan`. • ✓ FastAPI `BackgroundTasks` — Post-Response Execution, Session Safety & Celery / SQS Trade-Offs — Returning fast `202 Accepted` HTTP responses immediately while scheduling lightweight in-process audit logs, webhook pings, or email dispatches via `BackgroundTasks`. • ✓ Build, verify, and ship Module Capstone: Multi-Router Versioned API (`/api/v1`) with Lifespan Pool & Post-Response Audit Worker.

10.1 Bigger FastAPI Applications — `APIRouter`, Versioned Prefixes, Router Dependencies & `lifespan`

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

Subtopics Covered: APIRouter modular domain slicing & App.include_router() prefix and tag composition • Core Syntax, CLI & Production APIs: router = APIRouter(prefix='/v1/billing', tags=['billing'], dependencies=[Depends(...)]) + ap... • Deep-Dive Mechanics: Router-level dependencies=[Depends(...)] & @asynccontextmanager lifespan startup/shutdown • Advanced Patterns: API versioning (/api/v1, /api/v2) & Preventing circular imports via schemas/deps split • Production Failure Mode & Pitfall Guard • Hands-On Live Coding Lab: Versioned `APIRouter` Composition with Shared Header Guard • Idiomatic FAPI Implementation: async/await non-blocking endpoints, typed Pydantic contracts & grounded RAG • Asyncio Event Loop, FastAPI ASGI, Pydantic v2 & pgvector/PyTorch Execution & Memory Lifecycle for Bigger FastAPI Applications

Purpose: To understand and apply Bigger FastAPI Applications — `APIRouter`, Versioned Prefixes, Router Dependencies & `lifespan` in real FastAPI systems — specifically: Organizing...

What Topic Is Next → Next Topic 10.2: FastAPI `BackgroundTasks` — Post-Response Execution, Session Safety & Celery / SQS Trade-Offs

10.2 FastAPI `BackgroundTasks` — Post-Response Execution, Session Safety & Celery / SQS Trade-Offs

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

Subtopics Covered: Post-response in-process task execution & BackgroundTasks injection in routes & dependencies • Core Syntax, CLI & Production APIs: background_tasks.add_task(write_audit_log, email, event='ENROLLED') • Deep-Dive Mechanics: Fresh DB session rule inside background workers & Async vs sync background callable handling • Advanced Patterns: In-memory volatility on pod SIGKILL & BackgroundTasks vs Celery/ARQ/AWS SQS boundary • Production Failure Mode & Pitfall Guard • Hands-On Live Coding Lab: Post-Response `BackgroundTasks` Audit Trail Execution • Idiomatic FAPI Implementation: async/await non-blocking endpoints, typed Pydantic contracts & grounded RAG • Asyncio Event Loop, FastAPI ASGI, Pydantic v2 & pgvector/PyTorch Execution & Memory Lifecycle for FastAPI `BackgroundTasks`

Purpose: To understand and apply FastAPI `BackgroundTasks` — Post-Response Execution, Session Safety & Celery / SQS Trade-Offs in real FastAPI systems — specifically: Returning fa...

What Topic Is Next → Module 10 Capstone (Multi-Router Versioned API (`/api/v1`) with Lifespan Pool & Post-Response Audit Worker) → Next Topic 11.1: AsyncEngine, async_sessionmaker & Scoped Per-Request Yield Dependencies

Module 10 Hands-On Capstone Project (100 Marks): Multi-Router Versioned API (`/api/v1`) with Lifespan Pool & Post-Response Audit Worker — Structure a multi-module FastAPI service mounting `users` and `enrollments` `APIRouter` instances under `/api/v1` with router-level auth dependencies and a `BackgroundTasks` post-checkout au...

MODULE 11 OF 16 • STAGE 3 • 16 LIVE + LAB HOURS • 4 TOPICS (32 SUBTOPICS) • 1 CAPSTONE PROJECT (100 MARKS)

Async SQLAlchemy 2.0, Connection Pooling, Alembic & Keyset Pagination

Production-grade FastAPI Training engineering module covering Async SQLAlchemy 2.0, Connection Pooling, Alembic & Keyset Pagination with hands-on labs and capstone build.

Objectives: ✓ AsyncEngine, async_sessionmaker & Scoped Per-Request Yield Dependencies — Purpose: Run non-blocking asyncpg queries cleanly without session leaks or blocking the event loop. • ✓ Eager Loading (selectinload) in Async Mode to Prevent MissingGreenlet — Purpose: Avoid implicit lazy-load crashes in async SQLAlchemy while eliminating N+1 queries. • ✓ Alembic Async Migrations & Keyset Cursor Pagination — Purpose: Evolve PostgreSQL schemas safely and paginate millions of rows in constant O(1) index time. • ✓ Transactional Outbox & Atomic SELECT ... FOR UPDATE Locks — Purpose: Guarantee race-free balance updates and reliable downstream event publishing.

11.1 AsyncEngine, async_sessionmaker & Scoped Per-Request Yield Dependencies

[65 mins • Assignment (100M) • Hands-On Lab (50M) • 2 Mock Sets • 2 AI Vivas]

Subtopics Covered: Plain • Production Implementation of async_sessionmaker • Integration with Eager Loading (selectinload) in Async Mode to Prevent MissingGreenlet • Live Debugging, Telemetry & Technical Interview Trade • What Is AsyncEngine, async_sessionmaker & Scoped Per • In FastAPI Training,

AsyncEngine, async_sessionmaker & Scoped Per • Even if you come from a non • Runnable Code Implementation (PYTHON)

Purpose: To understand and apply AsyncEngine, async_sessionmaker & Scoped Per-Request Yield Dependencies in real FastAPI systems — specifically: Run non-blocking asyncpg queries c...

What Topic Is Next → Next Topic 11.2: Eager Loading (selectinload) in Async Mode to Prevent MissingGreenlet

11.2 Eager Loading (selectinload) in Async Mode to Prevent MissingGreenlet

[65 mins • Assignment (100M) • Hands-On Lab (50M) • 2 Mock Sets • 2 AI Vivas]

Subtopics Covered: Plain • Production Implementation of Edge • Integration with Alembic Async Migrations & Keyset Cursor Pagination • Live Debugging, Telemetry & Technical Interview Trade • What Is Eager Loading (selectinload) in Async Mode to Prevent MissingGreenlet & Why Industry Teams Use It • In FastAPI Training, Eager Loading (selectinload) in Async Mode to Prevent MissingGreenlet exists for one clear rea... • Even if you come from a non • Runnable Code Implementation (PYTHON)

Purpose: To understand and apply Eager Loading (selectinload) in Async Mode to Prevent MissingGreenlet in real FastAPI systems — specifically: Avoid implicit lazy-load crashes in...

What Topic Is Next → Next Topic 11.3: Alembic Async Migrations & Keyset Cursor Pagination

11.3 Alembic Async Migrations & Keyset Cursor Pagination

[65 mins • Assignment (100M) • Hands-On Lab (50M) • 2 Mock Sets • 2 AI Vivas]

Subtopics Covered: Plain • Production Implementation of Keyset Cursor Pagination • Integration with Transactional Outbox & Atomic SELECT ... FOR UPDATE Locks • Live Debugging, Telemetry & Technical Interview Trade • What Is Alembic Async Migrations & Keyset Cursor Pagination & Why Industry Teams Use It • In FastAPI Training, Alembic Async Migrations & Keyset Cursor Pagination exists for one clear real • Even if you come from a non • Runnable Code Implementation (PYTHON)

Purpose: To understand and apply Alembic Async Migrations & Keyset Cursor Pagination in real FastAPI systems — specifically: Evolve PostgreSQL schemas safely and paginate millions...

What Topic Is Next → Next Topic 11.4: Transactional Outbox & Atomic SELECT ... FOR UPDATE Locks

11.4 Transactional Outbox & Atomic SELECT ... FOR UPDATE Locks

[65 mins • Assignment (100M) • Hands-On Lab (50M) • 2 Mock Sets • 2 AI Vivas]

Subtopics Covered: Plain • Production Implementation of Atomic SELECT ... FOR UPDATE Locks • Integration with AsyncEngine, async_sessionmaker & Scoped Per • Live Debugging, Telemetry & Technical Interview Trade • What Is Transactional Outbox & Atomic SELECT ... FOR UPDATE Locks & Why Industry Teams Use It • In FastAPI Training, Transactional Outbox & Atomic SELECT ... • FOR UPDATE Locks exists for one clear real • Runnable Code Implementation (PYTHON)

Purpose: To understand and apply Transactional Outbox & Atomic SELECT ... FOR UPDATE Locks in real FastAPI systems — specifically: Guarantee race-free balance updates and reliable...

What Topic Is Next → Module 11 Capstone (Sub-20ms Async PostgreSQL Data Layer for FastAPI) → Next Topic 12.1: OAuth2PasswordBearer + Security() Granular Permission Scopes

Module 11 Hands-On Capstone Project (100 Marks): Sub-20ms Async PostgreSQL Data Layer for FastAPI — Design, implement, test, and document 'Sub-20ms Async PostgreSQL Data Layer for FastAPI' applying every production concept from Module 11 (Async SQLAlchemy 2.0, Connection Pooling, Alembic &...

MODULE 12 OF 16 • STAGE 3 • 16 LIVE + LAB HOURS • 4 TOPICS (32 SUBTOPICS) • 1 CAPSTONE PROJECT (100 MARKS)

OAuth2 Scopes, Rotating JWTs, Redis Rate Limiting & Idempotency Middleware

Production-grade FastAPI Training engineering module covering OAuth2 Scopes, Rotating JWTs, Redis Rate Limiting & Idempotency Middleware with hands-on labs and capstone build.

Objectives: ✓ OAuth2PasswordBearer + Security() Granular Permission Scopes — Purpose: Document and enforce exact read/write scopes right inside OpenAPI & Swagger UI. • ✓ Asymmetric RS256 JWT Verification & Redis Refresh Token Rotation — Purpose: Validate tokens fast across replicas and revoke stolen sessions immediately. • ✓ Sliding-Window Redis Rate Limiter & Idempotency-Key Middleware — Purpose: Prevent brute-force floods and duplicate payment execution under mobile retries. • ✓ Strict Pydantic v2 Validators, CORS Allowlists & TrustedHost Hardening — Purpose: Reject malformed payloads in Rust-powered microseconds before hitting business code.

12.1 OAuth2PasswordBearer + Security() Granular Permission Scopes

[65 mins • Assignment (100M) • Hands-On Lab (50M) • 2 Mock Sets • 2 AI Vivas]

Subtopics Covered: Plain • Production Implementation of Edge • Integration with Asymmetric RS256 JWT Verification & Redis Refresh Token Rotation • Live Debugging, Telemetry & Technical Interview Trade • What Is OAuth2PasswordBearer + Security() Granular Permission Scopes & Why Industry Teams Use It • In FastAPI Training, OAuth2PasswordBearer + Security() Granular Permission Scopes exists for one clear real • Even if you come from a non • Runnable Code Implementation (PYTHON)

Purpose: To understand and apply OAuth2PasswordBearer + Security() Granular Permission Scopes in real FastAPI systems — specifically: Document and enforce exact read/write scopes...

What Topic Is Next → Next Topic 12.2: Asymmetric RS256 JWT Verification & Redis Refresh Token Rotation

12.2 Asymmetric RS256 JWT Verification & Redis Refresh Token Rotation

[65 mins • Assignment (100M) • Hands-On Lab (50M) • 2 Mock Sets • 2 AI Vivas]

Subtopics Covered: Plain • Production Implementation of Redis Refresh Token Rotation • Integration with Sliding • Live Debugging, Telemetry & Technical Interview Trade • What Is Asymmetric RS256 JWT Verification & Redis Refresh Token Rotation & Why Industry Teams Use It • In FastAPI Training, Asymmetric RS256 JWT Verification & Redis Refresh Token Rotation exists for one clear real • Even if you come from a non • Runnable Code Implementation (PYTHON)

Purpose: To understand and apply Asymmetric RS256 JWT Verification & Redis Refresh Token Rotation in real FastAPI systems — specifically: Validate tokens fast across replicas and...

What Topic Is Next → Next Topic 12.3: Sliding-Window Redis Rate Limiter & Idempotency-Key Middleware

12.3 Sliding-Window Redis Rate Limiter & Idempotency-Key Middleware

[65 mins • Assignment (100M) • Hands-On Lab (50M) • 2 Mock Sets • 2 AI Vivas]

Subtopics Covered: Plain • Production Implementation of Idempotency • Integration with Strict Pydantic v2 Validators, CORS Allowlists & TrustedHost Hardening • Live Debugging, Telemetry & Technical Interview Trade • What Is Sliding • In FastAPI Training, Sliding • Even if you come from a non • Runnable Code Implementation

(PYTHON)

Purpose: To understand and apply Sliding-Window Redis Rate Limiter & Idempotency-Key Middleware in real FastAPI systems — specifically: Prevent brute-force floods and duplicate pa...

What Topic Is Next → Next Topic 12.4: Strict Pydantic v2 Validators, CORS Allowlists & TrustedHost Hardening

12.4 Strict Pydantic v2 Validators, CORS Allowlists & TrustedHost Hardening [65 mins • Assignment (100M) • Hands-On Lab (50M) • 2 Mock Sets • 2 AI Vivas]

Subtopics Covered: Plain • Production Implementation of CORS Allowlists • Integration with OAuth2PasswordBearer + Security() Granular Permission Scopes • Live Debugging, Telemetry & Technical Interview Trade • What Is Strict Pydantic v2 Validators, CORS Allowlists & TrustedHost Hardening & Why Industry Teams Use It • In FastAPI Training, Strict Pydantic v2 Validators, CORS Allowlists & TrustedHost Hardening exists for one clear re... • Even if you come from a non • Runnable Code Implementation (PYTHON)

Purpose: To understand and apply Strict Pydantic v2 Validators, CORS Allowlists & TrustedHost Hardening in real FastAPI systems — specifically: Reject malformed payloads in Rust-p...

What Topic Is Next → Module 12 Capstone (Zero-Trust FinTech FastAPI Gateway with Redis Guards) → Next Topic 13.1: FastAPI Lifespan Context Manager for Shared HTTPX, Redis & ML Pools

Module 12 Hands-On Capstone Project (100 Marks): Zero-Trust FinTech FastAPI Gateway with Redis Guards — Design, implement, test, and document 'Zero-Trust FinTech FastAPI Gateway with Redis Guards' applying every production concept from Module 12 (OAuth2 Scopes, Rotating JWTs, Redis Rate Limiti...

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

WebSockets, SSE Streaming, ARQ/Celery Queues & Lifespan Resource Pools

Production-grade FastAPI Training engineering module covering WebSockets, SSE Streaming, ARQ/Celery Queues & Lifespan Resource Pools with hands-on labs and capstone build.

Objectives: ✓ FastAPI Lifespan Context Manager for Shared HTTPX, Redis & ML Pools — Purpose: Initialize and tear down connection pools cleanly once per worker process. • ✓ Authenticated WebSockets & Server-Sent Events (StreamingResponse) — Purpose: Stream live notifications, stock ticks, and GenAI tokens with minimal memory overhead. • ✓ BackgroundTasks vs Distributed Redis Worker Queues (ARQ / Celery) — Purpose: Know when lightweight in-process tasks suffice vs durable retryable worker queues. • ✓ Resilient Outbound HTTPX AsyncClient with Timeouts & Circuit Fallbacks — Purpose: Call partner webhooks safely without letting slow external APIs stall Uvicorn.

13.1 FastAPI Lifespan Context Manager for Shared HTTPX, Redis & ML Pools [65 mins • Assignment (100M) • Hands-On Lab (50M) • 2 Mock Sets • 2 AI Vivas]

Subtopics Covered: Plain • Production Implementation of Redis • Integration with Authenticated WebSockets & Server • Live Debugging, Telemetry & Technical Interview Trade • What Is FastAPI Lifespan Context Manager for Shared HTTPX, Redis & ML Pools & Why Industry Teams Use It • In FastAPI Training, FastAPI Lifespan Context Manager for Shared HTTPX, Redis & ML Pools exists for one clear real • Even if you come from a non • Runnable Code Implementation (PYTHON)

Purpose: To understand and apply FastAPI Lifespan Context Manager for Shared HTTPX, Redis & ML Pools in real FastAPI systems — specifically: Initialize and tear down connection po...

What Topic Is Next → Next Topic 13.2: Authenticated WebSockets & Server-Sent Events (StreamingResponse)

13.2 Authenticated WebSockets & Server-Sent Events (StreamingResponse) [65 mins • Assignment (100M) • Hands-On Lab (50M) • 2 Mock Sets • 2 AI Vivas]

Subtopics Covered: Plain • Production Implementation of Server • Integration with BackgroundTasks vs Distributed Redis Worker Queues (ARQ / Celery) • Live Debugging, Telemetry & Technical Interview Trade • What Is Authenticated WebSockets & Server • In FastAPI Training, Authenticated WebSockets & Server • Even if you come from a non • Runnable Code Implementation (PYTHON)

Purpose: To understand and apply Authenticated WebSockets & Server-Sent Events (StreamingResponse) in real FastAPI systems — specifically: Stream live notifications, stock ticks,...

What Topic Is Next → Next Topic 13.3: BackgroundTasks vs Distributed Redis Worker Queues (ARQ / Celery)

13.3 BackgroundTasks vs Distributed Redis Worker Queues (ARQ / Celery) [65 mins • Assignment (100M) • Hands-On Lab (50M) • 2 Mock Sets • 2 AI Vivas]

Subtopics Covered: Plain • Production Implementation of Edge • Integration with Resilient Outbound HTTPX AsyncClient with Timeouts & Circuit Fallbacks • Live Debugging, Telemetry & Technical Interview Trade • What Is BackgroundTasks vs Distributed Redis Worker Queues (ARQ / Celery) & Why Industry Teams Use It • In FastAPI Training, BackgroundTasks vs Distributed Redis Worker Queues (ARQ / Celery) exists for one clear real • Even if you come from a non • Runnable Code Implementation (PYTHON)

Purpose: To understand and apply BackgroundTasks vs Distributed Redis Worker Queues (ARQ / Celery) in real FastAPI systems — specifically: Know when lightweight in-process tasks s...

What Topic Is Next → Next Topic 13.4: Resilient Outbound HTTPX AsyncClient with Timeouts & Circuit Fallbacks

13.4 Resilient Outbound HTTPX AsyncClient with Timeouts & Circuit Fallbacks [65 mins • Assignment (100M) • Hands-On Lab (50M) • 2 Mock Sets • 2 AI Vivas]

Subtopics Covered: Plain • Production Implementation of Circuit Fallbacks • Integration with FastAPI Lifespan Context Manager for Shared HTTPX, Redis & ML Pools • Live Debugging, Telemetry & Technical Interview Trade • What Is Resilient Outbound HTTPX AsyncClient with Timeouts & Circuit Fallbacks & Why Industry Teams Use It • In FastAPI Training, Resilient Outbound HTTPX AsyncClient with Timeouts & Circuit Fallbacks exists for one clear re... • Even if you come from a non • Runnable Code Implementation (PYTHON)

Purpose: To understand and apply Resilient Outbound HTTPX AsyncClient with Timeouts & Circuit Fallbacks in real FastAPI systems — specifically: Call partner webhooks safely withou...

What Topic Is Next → Module 13 Capstone (Real-Time Streaming & Async Background Processing Service) → Next Topic 14.1: pytest-asyncio + httpx.AsyncClient + Dependency Overrides

Module 13 Hands-On Capstone Project (100 Marks): Real-Time Streaming & Async Background Processing Service — Design, implement, test, and document 'Real-Time Streaming & Async Background Processing Service' applying every production concept from Module 13 (WebSockets, SSE Streaming, ARQ/Celery Queu...

MODULE 14 OF 16 • STAGE 4 • 16 LIVE + LAB HOURS • 4 TOPICS (32 SUBTOPICS) • 1 CAPSTONE PROJECT (100 MARKS)

Pytest-Asyncio Integration Suites, OpenTelemetry & Gunicorn/Uvicorn Docker

Production-grade FastAPI Training engineering module covering Pytest-Asyncio Integration Suites, OpenTelemetry & Gunicorn/Uvicorn Docker with hands-on labs and capstone build.

Objectives: ✓ pytest-asyncio + httpx.AsyncClient + Dependency Overrides — Purpose: Test every async endpoint, auth guard, and DB rollback in seconds. • ✓ RFC 9457 Problem-Details Exception Handlers & Correlation Request IDs — Purpose: Return predictable JSON error contracts linked directly to structured JSON logs. • ✓ OpenTelemetry Auto-Instrumentation & Prometheus /metrics Endpoint — Purpose: Measure p50/p95/p99 route latency and async DB query spans in Grafana. • ✓ Multi-Stage uv/pip Dockerfiles, Uvicorn Worker Tuning & Graceful Shutdown — Purpose: Deploy lean non-root FastAPI containers that drain active requests cleanly on deploy.

14.1 pytest-asyncio + httpx.AsyncClient + Dependency Overrides [65 mins • Assignment (100M) • Hands-On Lab (50M) • 2 Mock Sets • 2 AI Vivas]

Subtopics Covered: Plain • Production Implementation of Edge • Integration with RFC 9457 Problem • Live Debugging, Telemetry & Technical Interview Trade • What Is pytest • In FastAPI Training, pytest • Even if you come from a non • Runnable Code Implementation (PYTHON)

Purpose: To understand and apply pytest-asyncio + httpx.AsyncClient + Dependency Overrides in real FastAPI systems — specifically: Test every async endpoint, auth guard, and DB ro...

What Topic Is Next → Next Topic 14.2: RFC 9457 Problem-Details Exception Handlers & Correlation Request IDs

14.2 RFC 9457 Problem-Details Exception Handlers & Correlation Request IDs [65 mins • Assignment (100M) • Hands-On Lab (50M) • 2 Mock Sets • 2 AI Vivas]

Subtopics Covered: Plain • Production Implementation of Correlation Request IDs • Integration with OpenTelemetry Auto • Live Debugging, Telemetry & Technical Interview Trade • What Is RFC 9457 Problem • In FastAPI Training, RFC 9457 Problem • Even if you come from a non • Runnable Code Implementation (PYTHON)

Purpose: To understand and apply RFC 9457 Problem-Details Exception Handlers & Correlation Request IDs in real FastAPI systems — specifically: Return predictable JSON error contra...

What Topic Is Next → Next Topic 14.3: OpenTelemetry Auto-Instrumentation & Prometheus /metrics Endpoint

14.3 OpenTelemetry Auto-Instrumentation & Prometheus /metrics Endpoint [65 mins • Assignment (100M) • Hands-On Lab (50M) • 2 Mock Sets • 2 AI Vivas]

Subtopics Covered: Plain • Production Implementation of Prometheus /metrics Endpoint • Integration with Multi • Live Debugging, Telemetry & Technical Interview Trade • What Is OpenTelemetry Auto • In FastAPI Training, OpenTelemetry Auto • Even if you come from a non • Runnable Code Implementation (PYTHON)

Purpose: To understand and apply OpenTelemetry Auto-Instrumentation & Prometheus /metrics Endpoint in real FastAPI systems — specifically: Measure p50/p95/p99 route latency and as...

What Topic Is Next → Next Topic 14.4: Multi-Stage uv/pip Dockerfiles, Uvicorn Worker Tuning & Graceful Shutdown

14.4 Multi-Stage uv/pip Dockerfiles, Uvicorn Worker Tuning & Graceful Shutdown [65 mins • Assignment (100M) • Hands-On Lab (50M) • 2 Mock Sets • 2 AI Vivas]

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

Purpose: To understand and apply Multi-Stage uv/pip Dockerfiles, Uvicorn Worker Tuning & Graceful Shutdown in real FastAPI systems — specifically: Deploy lean non-root FastAPI con...

What Topic Is Next → Module 14 Capstone (CI-Tested Containerized FastAPI Production Microservice) → Next Topic 15.1: Event-Loop Safety: async def vs def Threadpool Routing & Lifespan Connection Pools

Module 14 Hands-On Capstone Project (100 Marks): CI-Tested Containerized FastAPI Production Microservice — Design, implement, test, and document 'CI-Tested Containerized FastAPI Production Microservice' applying every production concept from Module 14 (Pytest-Asyncio Integration Suites, OpenTelem...

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

Async FastAPI Concurrency Patterns, Lifespan Resources & Streaming APIs

Self-explanatory list-and-bullet production mastery module covering Async FastAPI Concurrency Patterns, Lifespan Resources & Streaming APIs in FastAPI Training, paired with hands-on labs, capstone repository, Topic Mock Tests, and Spoken MockAttempt AI Viva.

Objectives: ✓ Event-Loop Safety: async def vs def Threadpool Routing & Lifespan Connection Pools — Purpose: Never block the Uvicorn asyncio loop with synchronous CPU/IO calls under high concurrency. • ✓ Server-Sent Events (SSE), StreamingResponse & WebSocket Real-Time Channels — Purpose: Stream live AI tokens, progress updates, and telemetry feeds cleanly to web and mobile clients. • ✓ Redis Sliding-Window Rate Limiting, Idempotency Middleware & RFC 7807 Errors — Purpose: Standardize validation errors and protect public FastAPI routes from abusive traffic. • ✓ Build, verify, and document 'Token-Streaming & Rate-Limited Async FastAPI Gateway with Shared Lifespan Pools' with clean Git commits and edge-case protection. • ✓ Clear 100% of Topic Timed Mock Tests and Spoken Voice + MockAttempt AI Technical Interviews.

15.1 Event-Loop Safety: async def vs def Threadpool Routing & Lifespan Connection Pools [65 mins • Assignment (100M) • Hands-On Lab (50M) • 2 Mock Sets • 2 AI Vivas]

Subtopics Covered: Security Hygiene & Data Safety Rules for Event • Performance Optimization, Complexity & Latency Impact of Event • Live Debugging, Stack • Runnable Code Implementation (PYTHON) • Hands • Idiomatic FAPI Implementation: async/await non-blocking endpoints, typed Pydantic contracts & grounded RAG •

Asyncio Event Loop, FastAPI ASGI, Pydantic v2 & pgvector/PyTorch Execution & Memory Lifecycle for Event-Loop Safety • Production Hardening & Security: OAuth2 scopes, prompt-injection guardrails & token-rate quotas

Purpose: To understand and apply Event-Loop Safety: async def vs def Threadpool Routing & Lifespan Connection Pools in real FastAPI systems — specifically: Never block the Uvicorn...

What Topic Is Next → Next Topic 15.2: Server-Sent Events (SSE), StreamingResponse & WebSocket Real-Time Channels

15.2 Server-Sent Events (SSE), StreamingResponse & WebSocket Real-Time Channels [65 mins • Assignment (100M) • Hands-On Lab (50M) • 2 Mock Sets • 2 AI Vivas]

Subtopics Covered: Security Hygiene & Data Safety Rules for Server • Performance Optimization, Complexity & Latency Impact of Server • Live Debugging, Stack • Runnable Code Implementation (PYTHON) • Hands • Idiomatic FAPI Implementation: async/await non-blocking endpoints, typed Pydantic contracts & grounded RAG • Asyncio Event Loop, FastAPI ASGI, Pydantic v2 & pgvector/PyTorch Execution & Memory Lifecycle for Server-Sent Events (SSE) • Production Hardening & Security: OAuth2 scopes, prompt-injection guardrails & token-rate quotas

Purpose: To understand and apply Server-Sent Events (SSE), StreamingResponse & WebSocket Real-Time Channels in real FastAPI systems — specifically: Stream live AI tokens, progress...

What Topic Is Next → Next Topic 15.3: Redis Sliding-Window Rate Limiting, Idempotency Middleware & RFC 7807 Errors

15.3 Redis Sliding-Window Rate Limiting, Idempotency Middleware & RFC 7807 Errors [65 mins • Assignment (100M) • Hands-On Lab (50M) • 2 Mock Sets • 2 AI Vivas]

Subtopics Covered: Security Hygiene & Data Safety Rules for Redis Sliding • Performance Optimization, Complexity & Latency Impact of Redis Sliding • Live Debugging, Stack • Runnable Code Implementation (PYTHON) • Hands • Idiomatic FAPI Implementation: async/await non-blocking endpoints, typed Pydantic contracts & grounded RAG • Asyncio Event Loop, FastAPI ASGI, Pydantic v2 & pgvector/PyTorch Execution & Memory Lifecycle for Redis Sliding-Window Rate Limiting • Production Hardening & Security: OAuth2 scopes, prompt-injection guardrails & token-rate quotas

Purpose: To understand and apply Redis Sliding-Window Rate Limiting, Idempotency Middleware & RFC 7807 Errors in real FastAPI systems — specifically: Standardize validation errors...

What Topic Is Next → Module 15 Capstone (Token-Streaming & Rate-Limited Async FastAPI Gateway with Shared Lifespan Pools) → Next Topic 16.1: Async SQLAlchemy 2.0 Sessions, Alembic Async Migrations & Avoiding MissingGreenlet

Module 15 Hands-On Capstone Project (100 Marks): Token-Streaming & Rate-Limited Async FastAPI Gateway with Shared Lifespan Pools — Architect, implement, test, and document 'Token-Streaming & Rate-Limited Async FastAPI Gateway with Shared Lifespan Pools' applying every bullet-wise engineering concept from Module 15 (Asyn...

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

Async SQLAlchemy 2.0, pytest-asyncio Suites & Production Uvicorn/Gunicorn

Self-explanatory list-and-bullet production mastery module covering Async SQLAlchemy 2.0, pytest-asyncio Suites & Production Uvicorn/Gunicorn in FastAPI Training, paired with hands-on labs, capstone repository, Topic Mock Tests, and Spoken MockAttempt AI Viva.

Objectives: ✓ Async SQLAlchemy 2.0 Sessions, Alembic Async Migrations & Avoiding MissingGreenlet — Purpose: Write clean non-blocking database repositories with explicit eager loading and transaction scopes. • ✓ Dependency Override Testing with httpx.AsyncClient & pytest-asyncio Fixtures — Purpose: Test auth scopes, validation failures, and DB rollbacks rapidly in isolated async suites. • ✓ Production Uvicorn Workers, OpenTelemetry Tracing & FastAPI Interview Mastery — Purpose: Tune reverse-proxy headers, graceful shutdowns, and Pydantic v2 serialization speed. • ✓ Build, verify, and document 'Containerized Async FastAPI + PostgreSQL Microservice with 95% Route Coverage' with clean Git commits and edge-case protection. • ✓ Clear 100% of Topic Timed Mock Tests and Spoken Voice + MockAttempt AI Technical Interviews.

16.1 Async SQLAlchemy 2.0 Sessions, Alembic Async Migrations & Avoiding MissingGreenlet [65 mins • Assignment (100M) • Hands-On Lab (50M) • 2 Mock Sets • 2 AI Vivas]

Subtopics Covered: Security Hygiene & Data Safety Rules for Async SQLAlchemy 2.0 Sessions, Alembic Async Migrations & Avoiding Missing... • Performance Optimization, Complexity & Latency Impact of Async SQLAlchemy 2.0 Sessions, Alembic Async Migrations &.... • Live Debugging, Stack • Runnable Code Implementation (PYTHON) • Hands • Idiomatic FAPI Implementation: async/await non-blocking endpoints, typed Pydantic contracts & grounded RAG • Asyncio Event Loop, FastAPI ASGI, Pydantic v2 & pgvector/PyTorch Execution & Memory Lifecycle for Async SQLAlchemy 2.0 Sessions • Production Hardening & Security: OAuth2 scopes, prompt-injection guardrails & token-rate quotas

Purpose: To understand and apply Async SQLAlchemy 2.0 Sessions, Alembic Async Migrations & Avoiding MissingGreenlet in real FastAPI systems — specifically: Write clean non-blockin...

What Topic Is Next → Next Topic 16.2: Dependency Override Testing with httpx.AsyncClient & pytest-asyncio Fixtures

16.2 Dependency Override Testing with httpx.AsyncClient & pytest-asyncio Fixtures [65 mins • Assignment (100M) • Hands-On Lab (50M) • 2 Mock Sets • 2 AI Vivas]

Subtopics Covered: Security Hygiene & Data Safety Rules for Dependency Override Testing with httpx.AsyncClient & pytest • Performance Optimization, Complexity & Latency Impact of Dependency Override Testing with httpx.AsyncClient & pytes... • Live Debugging, Stack • Runnable Code Implementation (PYTHON) • Hands • Idiomatic FAPI Implementation: async/await non-blocking endpoints, typed Pydantic contracts & grounded RAG • Asyncio Event Loop, FastAPI ASGI, Pydantic v2 & pgvector/PyTorch Execution & Memory Lifecycle for Dependency Override Testing with httpx.AsyncClient • Production Hardening & Security: OAuth2 scopes, prompt-injection guardrails & token-rate quotas

Purpose: To understand and apply Dependency Override Testing with httpx.AsyncClient & pytest-asyncio Fixtures in real FastAPI systems — specifically: Test auth scopes, validation...

What Topic Is Next → Next Topic 16.3: Production Uvicorn Workers, OpenTelemetry Tracing & FastAPI Interview Mastery

16.3 Production Uvicorn Workers, OpenTelemetry Tracing & FastAPI Interview Mastery [65 mins • Assignment (100M) • Hands-On Lab (50M) • 2 Mock Sets • 2 AI Vivas]

Subtopics Covered: Security Hygiene & Data Safety Rules for Production Uvicorn Workers, OpenTelemetry Tracing & FastAPI Interview Mast... • Performance Optimization, Complexity & Latency Impact of Production Uvicorn Workers, OpenTelemetry Tracing & FastAP... • Live Debugging, Stack • Runnable Code Implementation (PYTHON) • Hands • Idiomatic FAPI Implementation: async/await non-blocking endpoints, typed Pydantic contracts & grounded RAG • Asyncio Event Loop, FastAPI ASGI, Pydantic v2 & pgvector/PyTorch Execution & Memory Lifecycle for Production Uvicorn Workers • Production Hardening & Security: OAuth2 scopes, prompt-injection guardrails & token-rate quotas

Purpose: To understand and apply Production Uvicorn Workers, OpenTelemetry Tracing & FastAPI Interview Mastery in real FastAPI systems — specifically: Tune reverse-proxy headers,...

What Topic Is Next → Module 16 Capstone → End-of-Course Live Client Project: FinPulse Async High-Concurrency Payment & Partner Webhook Microservice

Module 16 Hands-On Capstone Project (100 Marks): Containerized Async FastAPI + PostgreSQL Microservice with 95% Route Coverage — Architect, implement, test, and document 'Containerized Async FastAPI + PostgreSQL Microservice with 95% Route Coverage' applying every bullet-wise engineering concept from Module 16 (Async...

3. LIVE CLIENT PRODUCTION PROJECT & CAREER SCOPE

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

Project: FinPulse Async High-Concurrency Payment & Partner Webhook Microservice • **Confirmed Role:** Python FastAPI Backend Engineering Intern

Domain & Production Infra: Digital Payments & API Platform Client (FastAPI + Pydantic v2 + Async SQLAlchemy + Uvicorn/Gunicorn Docker Production)

- Developed async FastAPI endpoints with strict Pydantic v2 schema validation, dependency injection, and OpenAPI 3.1 docs.
- Implemented OAuth2/JWT security scopes, async database session pooling, and idempotent background task pipelines.
- Wrote comprehensive pytest-asyncio integration suites achieving high endpoint coverage and sub-30ms p95 latency.
- Containerized and deployed the microservice behind reverse-proxy TLS to live client production.

Target Job Roles: Python FastAPI Backend Engineer, Async Microservices Developer, AI / LLM Backend API Engineer, Software Engineer — Python APIs

Realistic Annual CTC Scope: **Fresher / Entry-Level (0–1 Yr):** ₹5.5 LPA – ₹10.0 LPA **Mid-Level FastAPI / AI Backend Engineer (1–3 Yrs):** ₹10.5 LPA – ₹21.5 LPA **Senior Python Microservices / GenAI Architect (3–6+ Yrs):** ₹22.0 LPA – ₹42.0+ LPA

MockAttempt Academy Pvt. Ltd. • Email: support@mockattempt.com • Phone/WhatsApp: +91 95400 07917 • Web: https://mockattempt.com/courses/fastapi-training

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