Admissions & Support:
support@mockattempt.com
Phone / WhatsApp: +91 95400 07917
Course Code: FLSK • **Updated:** 04 Nov 2026
https://mockattempt.com/courses/flask-training

FLSK • PYTHON WEB MICROSERVICES & REST API ENGINEERING | **BEGINNER TO PRINCIPAL FLASK 3 & PYTHON BACKEND ARCHITECT**

100% LIVE INDUSTRY EXPERT COHORT + INTERNSHIP LETTER

Flask Training

Master Python Flask 3 from Application Factories (create_app), Dynamic Routing, url_for(), Blueprints & Jinja2 Auto-Escaping to Signed Sessions, Flask-WTF CSRF, SQLite/SQLAlchemy 2.0 ORM, REST APIs, Flask-Login/JWT, Pytest test_client() & Gunicorn — with runnable Python programs, 386 Topic Mock Tests (17,860 MCQs), and Spoken Voice + MockAttempt AI Interviews at ₹1,799 (Incl. 18% GST).

Duration: 15–17 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

All-Inclusive Direct Fee (Incl. 18% GST • Zero Hidden Charges): Total Payable: ₹1,799 (Net Tuition ₹1,525 + Included 18% GST ₹274) • **Split Option:** 100% Credit to Job Track • **Effective ROI:** ~₹10/Guided Hr (~₹34/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 & Flask 3.1 • Werkzeug WSGI & Jinja2 • Blueprints & Flask-WTF • Flask-SQLAlchemy 2.0 & Alembic • REST APIs & Pydantic v2 • Flask-Login & JWT Auth • Pytest, Celery & Gunicorn

1. PROGRAM MASTERY OUTCOMES & ELIGIBILITY

End-to-End Engineering Outcomes:

1. Architect scalable Flask 3 apps using the Application Factory pattern (create_app), Blueprints, and thread-safe Werkzeug Context Locals (current_app, g, request)
2. Build secure SSR & API flows with Jinja2 XSS auto-escaping, itsdangerous signed sessions, secure_filename uploads, and Flask-WTF CSRF protection
3. Model relational schemas cleanly using Flask-SQLAlchemy (SQLAlchemy 2.0 Mapped columns), prevent N+1 queries with selectinload, and manage Alembic migrations
4. Clear 386 Timed Mock Exams (17,500 Flask MCQs) and 350 Spoken Voice + MockAttempt AI Python Backend Technical Interviews
5. Scalable Application Factory (`create_app`) and modular Blueprints cleanly separating routes, services, and models.
6. Crack LeetCode and HackerRank problems in Python 3.12 with live trainer mentorship on algorithmic/query patterns, edge cases, and optimal complexity.
7. SQLAlchemy 2.0 typed ORM queries, Alembic schema migrations, and connection-pooled transactional sessions.
8. Strict request/response validation with Marshmallow/Pydantic, JWT authentication, and custom CLI commands.

Eligibility & Prerequisites: Basic Python functions, decorators & dictionaries — teaches HTTP, WSGI, routing, and Flask 3 step-by-step from scratch • Any laptop/PC running Python 3.10+ (`venv`), pip, and VS Code / PyCharm • No prior Flask 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 (14 Topics / 112 Subtopics) • Modules 5–8: Stage 2: Applied Engineering, Data & API Workflows (14 Topics / 112 Subtopics) • Modules 9–12: Stage 3: Internals, Concurrency, Security & Performance (13 Topics / 104 Subtopics) • Modules 13–16: Stage 4: Production Architecture, Testing, CI/CD & Expert Mastery (12 Topics / 96 Subtopics)

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

Module 1: Flask 3 Microframework Architecture, WSGI (`Werkzeug`), `Jinja2` & Application Factory

Understand Armin Ronacher's Pallets stack (`Werkzeug` WSGI + `Jinja2` + `Click` + `itsdangerous`), run isolated `venv` Flask 3 apps cleanly, compare Flask vs Django vs FastAPI, and structure scalable apps with `create_app()` + Context Locals.

Objectives: ✓ Flask 3 Microframework Architecture — PEP-3333 `WSGI`, `Werkzeug`, `Jinja2` & `Click` Core — Why Flask is called a "micro" framework (minimal core with zero forced ORM/form lock-in, infinitely extensible!) built on the Pallets ecosystem: **`Werkzeug`** (WSGI HTTP routing/wrappers), **`Jinja2`** (templating), **`itsdangerous`** (cryptographic signing), and **`Click`** (`flask` CLI). • ✓ First Flask 3 Application — `Flask(__name__)`, `flask --app run --debug` CLI & Virtual Environments — Why passing `__name__` tells Flask exactly where relative `templates/` and `static/` folders live, modern Flask 3 CLI (`flask --app app run --debug`, replacing deprecated `FLASK_ENV=development` removed in Flask 2.3/3.0!), and `python -m venv .venv`. • ✓ Flask vs Django vs FastAPI — Architectural Trade-Offs, ORM Freedom & Microservice Fit — Choosing the right Python web framework: **Flask** (unopinionated WSGI + async view support, custom SQLAlchemy/MongoDB/SQLite freedom, minimal footprint) vs **Django** (batteries-included Admin, monolithic ORM, rigid layout) vs **FastAPI** (ASGI async-first OpenAPI). • ✓ Application Factory (`create_app`) & Context Locals (`current_app`, `g`, `request`, `session`) — Solving circular imports (`app <-> models <-> views`) with **`def create_app(test_config=None):`** + `ext.init_app(app)`, and how Flask 3 uses Python's `contextvars` to isolate **Application Context** (`current_app`, `g`) and **Request Context** (`request`, `session`) safely across concurrent threads/coroutines!. • ✓ Build, verify, and ship Module Capstone: Multi-Environment Flask 3 Application Factory (`create_app`) & Context Telemetry Starter.

1.1 Flask 3 Microframework Architecture — PEP-3333 `WSGI`, `Werkzeug`, `Jinja2` & `Click` Core

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

Subtopics Covered: PEP-3333 WSGI protocol & Werkzeug HTTP & routing toolkit • Core Syntax, CLI & Production APIs: from flask import Flask; app = Flask(__name__) # WSGI callable app(environ, start_response) • Deep-Dive Mechanics: Jinja2 template engine & Never name your file flask.py! • Advanced Patterns: Itsdangerous HMAC signing & Composable microframework philosophy • Production Failure Mode & Pitfall Guard • Hands-On Live Coding Lab: Verifying Flask 3 WSGI Callable & Test Client Dispatch • Idiomatic FLSK Implementation: Application Factory Blueprints, Pydantic/Marshmallow schemas & Gunicorn worker... • Flask 3 WSGI Request/App Context & SQLAlchemy 2 Session Execution & Memory Lifecycle for Flask 3 Microframework Architecture

Purpose: To understand and apply Flask 3 Microframework Architecture — PEP-3333 `WSGI`, `Werkzeug`, `Jinja2` & `Click` Core in real Flask systems — specifically: Why Flask is call...

What Topic Is Next → Next Topic 1.2: First Flask 3 Application — `Flask(__name__)`, `flask --app run --debug` CLI & Virtual Environments

1.2 First Flask 3 Application — `Flask(__name__)`, `flask --app run --debug` CLI & Virtual Environments

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

Subtopics Covered: Flask(__name__) resource root & Flask --app run --debug CLI • Core Syntax, CLI & Production APIs: python -m venv .venv && pip install Flask && flask --app app run --debug --port 5000 • Deep-Dive Mechanics: FLASK_ENV removed in Flask 3! & NEVER enable debug=True in prod! • Advanced Patterns: Isolated python -m venv & Host 127.0.0.1 vs 0.0.0.0 • Production Failure Mode & Pitfall Guard • Hands-On Live Coding Lab: Inspecting `app.root_path`, Config Defaults & Route Map • Idiomatic FLSK Implementation: Application Factory Blueprints, Pydantic/Marshmallow schemas & Gunicorn worker... • Flask 3 WSGI Request/App Context & SQLAlchemy 2 Session Execution & Memory Lifecycle for First Flask 3 Application

Purpose: To understand and apply First Flask 3 Application — `Flask(__name__)`, `flask --app run --debug` CLI & Virtual Environments in real Flask systems — specifically: Why pass...

What Topic Is Next → Next Topic 1.3: Flask vs Django vs FastAPI — Architectural Trade-Offs, ORM Freedom & Microservice Fit

1.3 Flask vs Django vs FastAPI — Architectural Trade-Offs, ORM Freedom & Microservice Fit

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

Subtopics Covered: Microframework vs Batteries-Included & ORM-agnostic (SQLAlchemy / MongoDB) • Core Syntax, CLI & Production APIs: Flask (Composable WSGI + SQLAlchemy/NoSQL choice) vs Django (Monolithic MTV + Built-in Admin... • Deep-Dive Mechanics: Flask[async] coroutine views & Explicit extension composition • Advanced Patterns: Lightweight Docker footprint & Custom microservice suitability • Production Failure Mode & Pitfall Guard • Hands-On Live Coding Lab: Framework Architecture Selector Matrix in Python • Idiomatic FLSK Implementation: Application Factory Blueprints, Pydantic/Marshmallow schemas & Gunicorn worker... • Flask 3 WSGI Request/App Context & SQLAlchemy 2 Session Execution & Memory Lifecycle for Flask vs Django vs FastAPI

Purpose: To understand and apply Flask vs Django vs FastAPI — Architectural Trade-Offs, ORM Freedom & Microservice Fit in real Flask systems — specifically: Choosing the right Pyt...

What Topic Is Next → Next Topic 1.4: Application Factory (`create_app`) & Context Locals (`current_app`, `g`, `request`, `session`)

1.4 Application Factory (`create_app`) & Context Locals (`current_app`, `g`, `request`, `session`)

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

Subtopics Covered: Create_app() Application Factory & Ext.init_app(app) two-phase init • Core Syntax, CLI & Production APIs: def create_app(cfg=None) -> Flask: ... db.init_app(app) | from flask import current_app, g • Deep-Dive Mechanics: Application Context (current_app, g) & Request Context (request, session) • Advanced Patterns: Python contextvars isolation & RuntimeError outside context • Production Failure Mode & Pitfall Guard • Hands-On Live Coding Lab: Clean `create_app()` Factory Storing Request-Scoped Metadata in `flask.g` • Idiomatic FLSK Implementation: Application Factory Blueprints, Pydantic/Marshmallow schemas & Gunicorn worker... • Flask 3 WSGI Request/App Context & SQLAlchemy 2 Session Execution & Memory Lifecycle for Application Factory (`create_app`)

Purpose: To understand and apply Application Factory (`create_app`) & Context Locals (`current_app`, `g`, `request`, `session`) in real Flask systems — specifically: Solving circu...

What Topic Is Next → Module 1 Capstone (Multi-Environment Flask 3 Application Factory (`create_app`) & Context Telemetry Starter) → Next Topic 2.1: Flask Dynamic Routing — `@app.route()`, Typed URL Converters (`<int>`, `<uuid>`, `<path>`) & Trailing Slashes

Module 1 Hands-On Capstone Project (100 Marks): Multi-Environment Flask 3 Application Factory (`create_app`) & Context Telemetry Starter — Build an extensible `create_app(config_object)` Flask 3 package supporting `DevConfig`, `TestConfig`, and `ProdConfig`, injecting a per-request correlation UUID into `flask.g.request_id`, an...

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

Module 2: Dynamic URL Routing, Converter Rules, Reverse `url_for()` & Modular `Blueprints`

Master Werkzeug's `Map` router, typed URL variable rules (`<int:id>`, `<uuid:token>`, `<path:subpath>`), canonical trailing-slash behavior, reverse URL building with `url_for()`, `Blueprint` modularization, and lifecycle hooks.

Objectives: ✓ Flask Dynamic Routing — `@app.route()`, Typed URL Converters (`<int>`, `<uuid>`, `<path>`) & Trailing Slashes — Mapping URLs via `@app.route()` (and `app.add_url_rule()`), built-in Werkzeug converters (`string`, `int`, `float`, `path`, `uuid`), custom `BaseConverter` subclasses, and the critical difference between `/projects/` (canonical folder `308` redirect) and `/about` (strict `404` on `/about/`!). • ✓ Reverse URL Building with `url_for()` — Query Params, `_external=True` & Static Cache Busting — Why `url_for("endpoint_name", **kwargs)` always beats hardcoded URL strings: automatic percent-escaping of special characters, unknown kwargs appended automatically as `?query=params`!, `_external=True` absolute URLs for emails, and `url_for("static", filename="app.css")`. • ✓ Modular Scaling with Flask `Blueprint`s — `url_prefix`, Domain Packages & Nested Blueprints — Decomposing a growing Flask codebase from one giant `app.py` into clean feature **`Blueprint`** modules (`Blueprint("api", __name__, url_prefix="/api/v1")`), blueprint-specific templates/static folders, and Flask 2.0+ **Nested Blueprints** (`parent_bp.register_blueprint(child_bp)`). • ✓ Request Lifecycle Hooks — `@before_request`, `@after_request` & `@teardown_appcontext` — Intercepting requests cleanly: **`@app.before_request`** (auth guard / timer start — returning a Response short-circuits the view!), **`@app.after_request`** (`def hook(response): return response` — MUST return `response`!), and **`@app.teardown_appcontext`** (guaranteed DB/resource cleanup even on unhandled exceptions!). • ✓ Build, verify, and ship Module Capstone: Multi-Blueprint Academy Router (`auth`, `catalog`, `api_v1`) with Custom Slug Converter.

2.1 Flask Dynamic Routing — `@app.route()`, Typed URL Converters (`<int>`, `<uuid>`, `<path>`) & Trailing Slashes

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

Subtopics Covered: <int:id>, <uuid:id>, <path:p> converters & Trailing slash 308 vs 404 rule! • Core Syntax, CLI & Production APIs: @app.get("/courses/<string:slug>/modules/<int:mod_no>") • Deep-Dive Mechanics: Unique endpoint function names & App.url_map inspection • Advanced Patterns: Custom Werkzeug BaseConverter & Strict_slashes=False option • Production Failure Mode & Pitfall Guard • Hands-On Live Coding Lab: Typed `<int:>` & `<path:>` Converters Tested with `test_client()` • Idiomatic FLSK Implementation: Application Factory Blueprints, Pydantic/Marshmallow schemas & Gunicorn worker... • Flask 3 WSGI Request/App Context & SQLAlchemy 2 Session Execution & Memory Lifecycle for Flask Dynamic Routing

Purpose: To understand and apply Flask Dynamic Routing — `@app.route()`, Typed URL Converters (`<int>`, `<uuid>`, `<path>`) & Trailing Slashes in real Flask systems — specifically...

What Topic Is Next → Next Topic 2.2: Reverse URL Building with `url_for()` — Query Params, `_external=True` & Static Cache Busting

2.2 Reverse URL Building with `url_for()` — Query Params, `_external=True` & Static Cache Busting

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

Subtopics Covered: Url_for(endpoint, **values) & Extra kwargs become ?query_params! • Core Syntax, CLI & Production APIs: url_for("courses.show", slug="flask-training", ref="banner", _external=False) • Deep-Dive Mechanics: Automatic URL percent-encoding & _external=True for emails/webhooks • Advanced Patterns: Url_for("static", filename=...) & Blueprint.endpoint namespace • Production Failure Mode & Pitfall Guard • Hands-On Live Coding Lab: Building Encoded Dynamic & Query-String URLs via `url_for()` • Idiomatic FLSK Implementation: Application Factory Blueprints, Pydantic/Marshmallow schemas & Gunicorn worker... • Flask 3 WSGI Request/App Context & SQLAlchemy 2 Session Execution & Memory Lifecycle for Reverse URL Building with `url_for()`

Purpose: To understand and apply Reverse URL Building with `url_for()` — Query Params, `_external=True` & Static Cache Busting in real Flask systems — specifically: Why `url_for("...

What Topic Is Next → Next Topic 2.3: Modular Scaling with Flask `Blueprint`s — `url_prefix`, Domain Packages & Nested Blueprints

2.3 Modular Scaling with Flask `Blueprint`s — `url_prefix`, Domain Packages & Nested Blueprints

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

Subtopics Covered: Blueprint("name", __name__, url_prefix=...) & App.register_blueprint(bp) • Core Syntax, CLI & Production APIs: api_bp = Blueprint("api", __name__, url_prefix="/api/v1"); app.register_blueprint(api_bp) • Deep-Dive Mechanics: Import routes BEFORE register_blueprint! & Nested child blueprints • Advanced Patterns: Per-blueprint before_request & errorhandler & Endpoint prefix bp_name.view_fn • Production Failure Mode & Pitfall Guard • Hands-On Live Coding Lab: Registering a Versioned API `Blueprint` Inside `create_app()` • Idiomatic FLSK Implementation: Application Factory Blueprints, Pydantic/Marshmallow schemas & Gunicorn worker... • Flask 3 WSGI Request/App Context & SQLAlchemy 2 Session Execution & Memory Lifecycle for Modular Scaling with Flask `Blueprint`s

Purpose: To understand and apply Modular Scaling with Flask `Blueprint`s — `url_prefix`, Domain Packages & Nested Blueprints in real Flask systems — specifically: Decomposing a gr...

What Topic Is Next → Next Topic 2.4: Request Lifecycle Hooks — `@before_request`, `@after_request` & `@teardown_appcontext`

2.4 Request Lifecycle Hooks — `@before_request`, `@after_request` & `@teardown_appcontext`

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

Subtopics Covered: Before_request short-circuit auth & After_request MUST return response! • Core Syntax, CLI & Production APIs: @app.before_request | @app.after_request (must return response!) | @app.teardown_appcontext • Deep-Dive Mechanics: Teardown_appcontext guaranteed cleanup & Before_first_request REMOVED in Flask 3! • Advanced Patterns: Security headers injection & Request latency timing via g.start • Production Failure Mode & Pitfall Guard • Hands-On Live Coding Lab: Injecting Security Headers & Timing via `@before_request` + `@after_request` • Idiomatic FLSK Implementation: Application Factory Blueprints, Pydantic/Marshmallow schemas & Gunicorn worker... • Flask 3 WSGI Request/App Context & SQLAlchemy 2 Session Execution & Memory Lifecycle for Request Lifecycle Hooks

Purpose: To understand and apply Request Lifecycle Hooks — `@before_request`, `@after_request` & `@teardown_appcontext` in real Flask systems — specifically: Intercepting requests...

What Topic Is Next → Module 2 Capstone (Multi-Blueprint Academy Router (`auth`, `catalog`, `api_v1`) with Custom Slug Converter) → Next Topic 3.1: Flask HTTP Methods — `@app.get` / `@app.post` Shorthands, `405 Method Not Allowed` & Post-Redirect-Get

Module 2 Hands-On Capstone Project (100 Marks): Multi-Blueprint Academy Router (`auth`, `catalog`, `api_v1`) with Custom Slug Converter — Structure a Flask 3 app with three `Blueprint`s (`url_prefix="/auth"`, `"/courses"`, `"/api/v1"`), register a custom `Werkzeug` `BaseConverter` validating lowercase `[a-z0-9-]+` slugs, and g...

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

Module 3: HTTP Verbs, The `request` Proxy Object, `jsonify` Responses & Content Negotiation

Handle `GET`, `POST`, `PUT`, `PATCH`, and `DELETE` cleanly with Flask 2+/3+ route shorthands, extract query/form/JSON payloads safely from `flask.request` without `KeyError` crashes, and craft rich HTTP `Response` tuples & streaming responses.

Objectives: ✓ Flask HTTP Methods — `@app.get` / `@app.post` Shorthands, `405 Method Not Allowed` & Post-Redirect-Get — Why `@app.route("/login", methods=["GET", "POST"])` defaults to `GET` (+ implicit `HEAD` & `OPTIONS`!) when `methods=` is omitted, modern Flask 2+/3+ `@app.get()` / `@app.post()` / `@app.delete()` decorators, automatic `405 Method Not Allowed`, and the **Post-Redirect-Get (PRG)** pattern. • ✓ The `flask.request` Object — `args`, `form`, `get_json(silent=True)`, `MultiDict` & Proxy IP Safety — Inspecting incoming HTTP data accurately: `request.args` (`ImmutableMultiDict` with `.get("page", 1, type=int)` & `.getlist("tag")`!), `request.form`, `request.get_json(silent=True)`, `request.headers`, and why `request.remote_addr` behind Nginx requires **`ProxyFix`**!. • ✓ Constructing `Response`s — Dict Auto-JSON, `jsonify()`, `make_response()` & Streaming SSE Generators — How Flask 2.2+/3.x converts returned `dict` and `list` objects directly to `application/json` via `app.json.response()`, `(body, status, headers)` response tuples, `make_response()` for attaching cookies/ETags, and `stream_with_context()` for real-time AI/CSV streaming!. • ✓ Build, verify, and ship Module Capstone: Content-Negotiated Lead Intake Endpoint (JSON + HTML Form PRG) with Safe `request` Parsing.

3.1 Flask HTTP Methods — `@app.get` / `@app.post` Shorthands, `405 Method Not Allowed` & Post-Redirect-Get [50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Subtopics Covered: @app.get / @app.post / @app.delete & Never mutate DB state in GET! • Core Syntax, CLI & Production APIs: @app.get("/items") | @app.post("/items") | @app.delete("/items/<int:id>") • Deep-Dive Mechanics: 405 Method Not Allowed auto-handling & Automatic HEAD & OPTIONS support • Advanced Patterns: Post-Redirect-Get (PRG) pattern & PUT idempotent vs POST non-idempotent • Production Failure Mode & Pitfall Guard • Hands-On Live Coding Lab: Clean `@app.get` & `@app.post` Separation Verifying `405` on Unsupported Verbs • Idiomatic FLSK Implementation: Application Factory Blueprints, Pydantic/Marshmallow schemas & Gunicorn worker... • Flask 3 WSGI Request/App Context & SQLAlchemy 2 Session Execution & Memory Lifecycle for Flask HTTP Methods

Purpose: To understand and apply Flask HTTP Methods — `@app.get` / `@app.post` Shorthands, `405 Method Not Allowed` & Post-Redirect-Get in real Flask systems — specifically: Why `...

What Topic Is Next → Next Topic 3.2: The `flask.request` Object — `args`, `form`, `get_json(silent=True)`, `MultiDict` & Proxy IP Safety

3.2 The `flask.request` Object — `args`, `form`, `get_json(silent=True)`, `MultiDict` & Proxy IP Safety [50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Subtopics Covered: Request.args.get(key, default, type=int) & MultiDict .getlist("tags") • Core Syntax, CLI & Production APIs: page = request.args.get("page", default=1, type=int); payload = request.get_json(silent=True... • Deep-Dive Mechanics: Request.get_json(silent=True) or {} & BadRequestKeyError 400 on missing [key] • Advanced Patterns: ProxyFix for X-Forwarded-For IP & Request.headers case-insensitive lookup • Production Failure Mode & Pitfall Guard • Hands-On Live Coding Lab: Type-Safe Query `args`, `.getlist()`, and Safe JSON Parsing • Idiomatic FLSK Implementation: Application Factory Blueprints, Pydantic/Marshmallow schemas & Gunicorn worker... • Flask 3 WSGI Request/App Context & SQLAlchemy 2 Session Execution & Memory Lifecycle for The `flask.request` Object

Purpose: To understand and apply The `flask.request` Object — `args`, `form`, `get_json(silent=True)`, `MultiDict` & Proxy IP Safety in real Flask systems — specifically: Inspecti...

What Topic Is Next → Next Topic 3.3: Constructing `Response`s — Dict Auto-JSON, `jsonify()`, `make_response()` & Streaming SSE Generators

3.3 Constructing `Response`s — Dict Auto-JSON, `jsonify()`, `make_response()` & Streaming SSE Generators [50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Subtopics Covered: Dict & list auto-JSON in Flask 3 & (body, status, headers) tuple • Core Syntax, CLI & Production APIs: return {"ok": True}, 200, {"Cache-Control": "no-store"} | stream_with_context(generate()) • Deep-Dive Mechanics: Make_response() header/cookie control & Stream_with_context() SSE/CSV streaming • Advanced Patterns: DefaultJSONProvider (replaces json_encoder) & Cannot return bare int from view • Production Failure Mode & Pitfall Guard • Hands-On Live Coding Lab: Returning Custom Status + Headers Tuple and Streaming Chunks • Idiomatic FLSK Implementation: Application Factory Blueprints, Pydantic/Marshmallow schemas & Gunicorn worker... • Flask 3 WSGI Request/App Context & SQLAlchemy 2 Session Execution & Memory Lifecycle for Constructing `Response`s

Purpose: To understand and apply Constructing `Response`s — Dict Auto-JSON, `jsonify()`, `make_response()` & Streaming SSE Generators in real Flask systems — specifically: How Fla...

What Topic Is Next → Module 3 Capstone (Content-Negotiated Lead Intake Endpoint (JSON + HTML Form PRG) with Safe `request` Parsing) → Next Topic 4.1: `Jinja2` Templates — `render_template()`, Layout Inheritance (`{% extends %}`), Filters & Macros

Module 3 Hands-On Capstone Project (100 Marks): Content-Negotiated Lead Intake Endpoint (JSON + HTML Form PRG) with Safe `request` Parsing — Build a `/leads` endpoint accepting both `application/json` (`request.get_json(silent=True)`) and HTML Form `POST` (`request.form.get()`), enforcing `415 Unsupported Media Type` on malformed...

MODULE 4 OF 16 • STAGE 1 • 10 LIVE + LAB HOURS • 3 TOPICS (24 SUBTOPICS) • 1 CAPSTONE PROJECT (100 MARKS)

Module 4: `Jinja2` Template Engine — Inheritance, Macros, Auto-Escaping XSS Defense & Flash Messaging

Render clean server-side HTML5 UIs with `Jinja2` (`render_template`, `{% extends %}`, `{% macro %}`), surface one-time toast alerts via `flash()` + `get_flashed_messages()`, and block XSS/SSTI attacks cleanly.

Objectives: ✓ `Jinja2` Templates — `render_template()`, Layout Inheritance (`{% extends %}`), Filters & Macros — Separating Python logic from HTML presentation in `templates/`: expressions (`{{ var }}`), control blocks (`{% for %}`, `{% if %}`), DRY layout inheritance (`{% extends "base.html" %}` + `{{ super() }}`), built-in filters (`|default`, `|length`, `|tojson`), and `{% macro %}` components. • ✓ Flask Message Flashing — `flash(msg, category)` & `get_flashed_messages(with_categories=True)` — Storing one-time feedback banners in the signed `session` at the end of a `POST` request (`flash("Profile updated!", "success")`) and popping/clearing them automatically on the very next `GET` render via `get_flashed_messages(with_categories=True)`. • ✓ `@app.context_processor`, Custom Filters & Blocking SSTI (`render_template_string` RCE) / XSS — Sharing common template variables (`@app.context_processor def inject_globals(): return {"brand": "MockAttempt"}`), registering `@app.template_filter("inr")`, and understanding the critical difference between **XSS** (`|safe`) and **Server-Side Template Injection (SSTI)**!. • ✓ Build, verify, and ship Module Capstone: Jinja2 Course Portal with Base Layout Inheritance, Reusable Card Macros & Categorized Flash Toasts.

4.1 `Jinja2` Templates — `render_template()`, Layout Inheritance (`{% extends %}`), Filters & Macros [50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Subtopics Covered: Render_template("view.html", **ctx) & {% extends %} & {% block %} inheritance • Core Syntax, CLI & Production APIs: render_template("courses/detail.html", course=course_obj, offer_fee=999) • Deep-Dive Mechanics: {% macro %} reusable UI components & {{ data | tojson }} safe JS embedding • Advanced Patterns: Bytecode-compiled template cache & MarkupSafe automatic HTML escaping • Production Failure Mode & Pitfall Guard • Hands-On Live Coding Lab: Rendering Escaped `Jinja2` Expressions & `|tojson` Safely • Idiomatic FLSK Implementation: Application Factory Blueprints, Pydantic/Marshmallow schemas & Gunicorn worker... • Flask 3 WSGI Request/App Context & SQLAlchemy 2 Session Execution & Memory Lifecycle for `Jinja2` Templates

Purpose: To understand and apply `Jinja2` Templates — `render_template()`, Layout Inheritance (`{% extends %}`), Filters & Macros in real Flask systems — specifically: Separating...

What Topic Is Next → Next Topic 4.2: Flask Message Flashing — `flash(msg, category)` & `get_flashed_messages(with_categories=True)`

4.2 Flask Message Flashing — `flash(msg, category)` & `get_flashed_messages(with_categories=True)` [50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Subtopics Covered: Flash(message, category) & Get_flashed_messages(with_categories=True) • Core Syntax, CLI & Production APIs: flash("Batch unlocked!", "success") -> {% for cat, msg in get_flashed_messages(with_categori... • Deep-Dive Mechanics: Requires app.secret_key! & Popped once on next request • Advanced Patterns: 4 KB cookie size awareness & Category_filter=["error"] option • Production Failure Mode & Pitfall Guard • Hands-On Live Coding Lab: Testing One-Time `flash()` Delivery & Automatic Session Pop • Idiomatic FLSK Implementation: Application Factory Blueprints, Pydantic/Marshmallow schemas & Gunicorn worker... • Flask 3 WSGI Request/App Context & SQLAlchemy 2 Session Execution & Memory Lifecycle for Flask Message Flashing

Purpose: To understand and apply Flask Message Flashing — `flash(msg, category)` & `get_flashed_messages(with_categories=True)` in real Flask systems — specifically: Storing one-t...

What Topic Is Next → Next Topic 4.3: `@app.context_processor`, Custom Filters & Blocking SSTI (`render_template_string` RCE) / XSS

4.3 `@app.context_processor`, Custom Filters & Blocking SSTI (`render_template_string` RCE) / XSS [50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Subtopics Covered: SSTI RCE via f-string template! & Never use |safe on untrusted HTML • Core Syntax, CLI & Production APIs: @app.context_processor | @app.template_filter("currency") | markupsafe.escape() • Deep-Dive Mechanics: @app.context_processor dict return & @app.template_filter custom formatter • Advanced Patterns: Keep context_processor fast (no slow SQL) & Bleach / NH3 HTML sanitizer when needed • Production Failure Mode & Pitfall Guard • Hands-On Live Coding Lab: Registering a Custom `@app.template_filter("inr")` & Context Processor • Idiomatic FLSK Implementation: Application Factory Blueprints, Pydantic/Marshmallow schemas & Gunicorn worker... • Flask 3 WSGI Request/App Context & SQLAlchemy 2 Session Execution & Memory Lifecycle for `@app.context_processor`

Purpose: To understand and apply `@app.context_processor`, Custom Filters & Blocking SSTI (`render_template_string` RCE) / XSS in real Flask systems — specifically: Sharing common...

What Topic Is Next → Module 4 Capstone (Jinja2 Course Portal with Base Layout Inheritance, Reusable Card Macros & Categorized Flash Toasts) → Next Topic 5.1: Flask HTTP Cookies — `request.cookies`, `resp.set_cookie()`, `HttpOnly`, `Secure` & `SameSite`

Module 4 Hands-On Capstone Project (100 Marks): Jinja2 Course Portal with Base Layout Inheritance, Reusable Card Macros & Categorized Flash Toasts — Build an SSR student dashboard using a shared `base.html` template, a reusable `{% macro course_card(course) %}` component, `@app.context_processor` current-year injection, and `flash("Enrol...

MODULE 5 OF 16 • STAGE 2 • 12 LIVE + LAB HOURS • 4 TOPICS (32 SUBTOPICS) • 1 CAPSTONE PROJECT (100 MARKS)

Module 5: Client Cookies, `itsdangerous` Signed Sessions, Server-Side Redis Sessions & File Uploads

Harden HTTP cookies (`HttpOnly`, `Secure`, `SameSite`), understand Flask's `itsdangerous` signed `session` vs Redis-backed `Flask-Session`, lock down multipart file uploads with `secure_filename` + `MAX_CONTENT_LENGTH`, and send emails with `Flask-Mail`.

Objectives: ✓ Flask HTTP Cookies — `request.cookies`, `resp.set_cookie()`, `HttpOnly`, `Secure` & `SameSite` — Reading cookies via `request.cookies.get("theme")` and setting security-hardened cookies on `Response` objects (`set_cookie`, `delete_cookie`) with **`httponly=True`** (blocks JS `document.cookie` XSS theft!), **`secure=True`** (HTTPS only), and **`samesite="Lax"`** (CSRF defense!). • ✓ Flask `session` Internals — `itsdangerous` Signing, `SECRET_KEY_FALLBACKS` & Server-Side Redis Sessions — How Flask's built-in `session` serializes dicts via `itsdangerous.URLSafeTimedSerializer` signed with `SECRET_KEY`, `PERMANENT_SESSION_LIFETIME`, mutating nested lists (`session.modified = True`!), zero-downtime key rotation (`SECRET_KEY_FALLBACKS` in Flask 2.3/3.x!), and when to switch to Redis `Flask-Session`. • ✓ Secure File Uploads — `request.files`, `secure_filename()`, `MAX_CONTENT_LENGTH` (`413`) & Storage Isolation — Handling `enctype="multipart/form-data"` uploads via `request.files["resume"]`, capping upload size globally with **`app.config["MAX_CONTENT_LENGTH"]`** (auto-raises `413 Request Entity Too Large`!), sanitizing names with `werkzeug.utils.secure_filename`, and storing outside `static/`. • ✓ Transactional Email with `Flask-Mail` — SMTP/TLS Configuration, Bulk Connections & Async Queueing — Configuring `MAIL_SERVER`, `MAIL_PORT=587`, `MAIL_USE_TLS=True`, and env-sourced credentials (`MAIL_USERNAME`, `MAIL_PASSWORD`), composing plain + HTML multipart `Message`s, `mail.connect()` connection reuse for batch sends, and `MAIL_SUPPRESS_SEND` in tests. • ✓ Build, verify, and ship Module Capstone: Hardened Assignment Upload Vault (`secure_filename` + `413` Guard) & Signed Session Cart.

5.1 Flask HTTP Cookies — `request.cookies`, `resp.set_cookie()`, `HttpOnly`, `Secure` & `SameSite` [50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Subtopics Covered: Httponly=True XSS cookie defense & Secure=True HTTPS-only flag • Core Syntax, CLI & Production APIs: resp = make_response(...); resp.set_cookie("pref", "dark", max_age=86400, httponly=True, sec... • Deep-Dive Mechanics: Samesite="Lax" or "Strict" & Never trust unsigned raw cookies! • Advanced Patterns: Max_age vs session-lifetime cookie & Delete_cookie matching path/domain • Production Failure Mode & Pitfall Guard • Hands-On Live Coding Lab: Setting & Reading a Hardened `HttpOnly` + `SameSite=Lax` Cookie • Idiomatic FLSK Implementation: Application Factory Blueprints, Pydantic/Marshmallow schemas & Gunicorn worker... • Flask 3 WSGI Request/App Context & SQLAlchemy 2 Session Execution & Memory Lifecycle for Flask HTTP Cookies

Purpose: To understand and apply Flask HTTP Cookies — `request.cookies`, `resp.set_cookie()`, `HttpOnly`, `Secure` & `SameSite` in real Flask systems — specifically: Reading cooki...

What Topic Is Next → Next Topic 5.2: Flask `session` Internals — `itsdangerous` Signing, `SECRET_KEY_FALLBACKS` & Server-Side Redis Sessions

5.2 Flask `session` Internals — `itsdangerous` Signing, `SECRET_KEY_FALLBACKS` & Server-Side Redis Sessions [50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Subtopics Covered: SIGNED not encrypted (readable!) & Secrets.token_hex(32) SECRET_KEY • Core Syntax, CLI & Production APIs: app.config.update(SECRET_KEY=os.environ["SECRET_KEY"], SESSION_COOKIE_HTTPONLY=True, PERMANE... • Deep-Dive Mechanics: Session.modified = True on nested list! & SECRET_KEY_FALLBACKS key rotation • Advanced Patterns: PERMANENT_SESSION_LIFETIME & Redis Flask-Session for instant revoke • Production Failure Mode & Pitfall Guard • Hands-On Live Coding Lab: Persisting & Mutating Nested Session Lists Safely with `session.modified` • Idiomatic FLSK Implementation: Application Factory Blueprints, Pydantic/Marshmallow schemas & Gunicorn worker... • Flask 3 WSGI Request/App Context & SQLAlchemy 2 Session Execution & Memory Lifecycle for Flask `session` Internals

Purpose: To understand and apply Flask `session` Internals — `itsdangerous` Signing, `SECRET_KEY_FALLBACKS` & Server-Side Redis Sessions in real Flask systems — specifically: How...

What Topic Is Next → Next Topic 5.3: Secure File Uploads — `request.files`, `secure_filename()`, `MAX_CONTENT_LENGTH` (`413`) & Storage Isolation

5.3 Secure File Uploads — `request.files`, `secure_filename()`, `MAX_CONTENT_LENGTH` (`413`) & Storage Isolation [50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Subtopics Covered: Werkzeug.utils.secure_filename()! & MAX_CONTENT_LENGTH -> 413 error • Core Syntax, CLI & Production APIs: app.config["MAX_CONTENT_LENGTH"] = 5 * 1024 * 1024; safe_name = f"{uuid4().hex}_{secure_file... • Deep-Dive Mechanics: UUID prefix prevents collisions & Allowlist extensions (.pdf, .png) • Advanced Patterns: Store outside public static/ folder & Send_from_directory path-safe download • Production Failure Mode & Pitfall Guard • Hands-On Live Coding Lab: Testing Multipart Upload Validation & `MAX_CONTENT_LENGTH` (`413`) Guard • Idiomatic FLSK Implementation: Application Factory Blueprints, Pydantic/Marshmallow schemas & Gunicorn worker... • Flask 3 WSGI Request/App Context & SQLAlchemy 2 Session Execution & Memory Lifecycle for Secure File Uploads

Purpose: To understand and apply Secure File Uploads — `request.files`, `secure_filename()`, `MAX_CONTENT_LENGTH` (`413`) & Storage Isolation in real Flask systems — specifically:...

What Topic Is Next → Next Topic 5.4: Transactional Email with `Flask-Mail` — SMTP/TLS Configuration, Bulk Connections & Async Queueing

5.4 Transactional Email with `Flask-Mail` — SMTP/TLS Configuration, Bulk Connections & Async Queueing [50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Subtopics Covered: MAIL_USE_TLS=True on port 587 & Never hardcode SMTP passwords! • Core Syntax, CLI & Production APIs: msg = Message(subject="Welcome", recipients=[user.email], body="...", html=render_template(.... • Deep-Dive Mechanics: Offload SMTP to Celery/RQ worker! & Mail.record_messages() test outbox • Advanced Patterns: Multipart text + HTML fallback & Mail.connect() bulk send reuse • Production Failure Mode & Pitfall Guard • Hands-On Live Coding Lab: Validating SMTP Header Safety & Multipart Receipt Builder • Idiomatic FLSK Implementation: Application Factory Blueprints, Pydantic/Marshmallow schemas & Gunicorn worker... • Flask 3 WSGI Request/App Context & SQLAlchemy 2 Session Execution & Memory Lifecycle for Transactional Email with `Flask-Mail`

Purpose: To understand and apply Transactional Email with `Flask-Mail` — SMTP/TLS Configuration, Bulk Connections & Async Queueing in real Flask systems — specifically: Configurin...

What Topic Is Next → Module 5 Capstone (Hardened Assignment Upload Vault (`secure_filename` + `413` Guard) & Signed Session Cart) → Next Topic 6.1: Flask `redirect()` (`302` / `303` / `301`), Open-Redirect Defense & Early `abort()` HTTP Exceptions

Module 5 Hands-On Capstone Project (100 Marks): Hardened Assignment Upload Vault (`secure_filename` + `413` Guard) & Signed Session Cart — Build a student submission portal enforcing `MAX_CONTENT_LENGTH = 5 * 1024 * 1024` (`5 MB`), validating `.pdf`/`.zip` extensions + magic headers, renaming files with `uuid4().hex + "_" + sec...

MODULE 6 OF 16 • STAGE 2 • 10 LIVE + LAB HOURS • 3 TOPICS (24 SUBTOPICS) • 1 CAPSTONE PROJECT (100 MARKS)

Module 6: Redirects, `abort()` HTTP Exceptions, Custom Error Handlers & `Flask-WTF` CSRF Forms

Control HTTP flows safely with `redirect()` (blocking Open Redirect phishing!) and `abort()`, unify JSON/HTML error responses via `@app.errorhandler`, and build CSRF-hardened validated forms with `Flask-WTF` (`WTForms`).

Objectives: ✓ Flask `redirect()` (`302` / `303` / `301`), Open-Redirect Defense & Early `abort()` HTTP Exceptions — Issuing redirects via `redirect(url_for("dashboard"), code=302)` (`303 See Other` after `POST`, `301`/`308` permanent SEO moves), halting execution cleanly by raising Werkzeug `HTTPException` via **`abort(404)`** / **`abort(403)`**, and validating `?next=` URLs!. • ✓ `Flask-WTF` & `WTForms` — `FlaskForm`, `CSRFProtect`, `validate_on_submit()` & Custom Validators — Declarative Python form classes (`class EnrollForm(FlaskForm)`), automatic hidden `{{ form.hidden_tag() }}` **CSRF synchronization token** verification, built-in validators (`DataRequired`, `Email`, `Length`, `NumberRange`), and custom `def validate_email(self, field):` hooks. • ✓ Global `@app.errorhandler` JSON/HTML Contracts & Structured Production Logging — Replacing default Werkzeug HTML error pages on API endpoints with `@app.errorhandler(HTTPException)` returning structured JSON (`{"code": e.code, "name": e.name, "description": e.description}`), catching unexpected `Exception`s safely without swallowing `HTTPException`, and `app.logger`. • ✓ Build, verify, and ship Module Capstone: CSRF-Protected Counseling Inquiry Form (`Flask-WTF`) & Dual JSON/HTML Error Handler.

6.1 Flask `redirect()` (`302` / `303` / `301`), Open-Redirect Defense & Early `abort()` HTTP Exceptions [50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Subtopics Covered: Redirect(url_for(...)) & Open Redirect ?next= validation! • Core Syntax, CLI & Production APIs: from flask import abort, redirect, url_for; if not item: abort(404, description="Course not... • Deep-Dive Mechanics: Abort(404) raises HTTPException & 401 Unauthenticated vs 403 Forbidden • Advanced Patterns: 302/303 temporary vs 301/308 permanent & Never catch HTTPException as generic 500! • Production Failure Mode & Pitfall Guard • Hands-On Live Coding Lab: Safe Relative Redirect Validator & `abort(404)` Guard • Idiomatic FLSK Implementation: Application Factory Blueprints, Pydantic/Marshmallow schemas & Gunicorn worker... • Flask 3 WSGI Request/App Context & SQLAlchemy 2 Session Execution & Memory Lifecycle for Flask `redirect()` (`302` / `303` / `301`)

Purpose: To understand and apply Flask `redirect()` (`302` / `303` / `301`), Open-Redirect Defense & Early `abort()` HTTP Exceptions in real Flask systems — specifically: Issuing...

What Topic Is Next → Next Topic 6.2: `Flask-WTF` & `WTForms` — `FlaskForm`, `CSRFProtect`, `validate_on_submit()` & Custom Validators

6.2 `Flask-WTF` & `WTForms` — `FlaskForm`, `CSRFProtect`, `validate_on_submit()` & Custom Validators [50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Subtopics Covered: FlaskForm & CSRFProtect(app) & {{ form.hidden_tag() }} required! • Core Syntax, CLI & Production APIs: CSRFProtect(app) + class EnrollForm(FlaskForm): ... if form.validate_on_submit(): ... • Deep-Dive Mechanics: Form.validate_on_submit() & DataRequired vs InputRequired • Advanced Patterns: Custom validate_<fieldname> method & WTF_CSRF_ENABLED=False only in unit tests • Production Failure Mode & Pitfall Guard • Hands-On Live Coding Lab: Custom `WTForms` / Python Validator Logic Rejecting Disposable Domains • Idiomatic FLSK Implementation: Application Factory Blueprints, Pydantic/Marshmallow schemas & Gunicorn worker... • Flask 3 WSGI Request/App Context & SQLAlchemy 2 Session Execution & Memory Lifecycle for `Flask-WTF`

Purpose: To understand and apply `Flask-WTF` & `WTForms` — `FlaskForm`, `CSRFProtect`, `validate_on_submit()` & Custom Validators in real Flask systems — specifically: Declarative...

What Topic Is Next → Next Topic 6.3: Global `@app.errorhandler` JSON/HTML Contracts & Structured Production Logging

6.3 Global `@app.errorhandler` JSON/HTML Contracts & Structured Production Logging [50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Subtopics Covered: @app.errorhandler(HTTPException) & Pass through HTTPException in Exception handler! • Core Syntax, CLI & Production APIs: @app.errorhandler(HTTPException) def handle_http_exc(e: HTTPException): return {"error": e.n... • Deep-Dive Mechanics: JSON vs HTML error content negotiation & Never leak Python tracebacks in prod JSON • Advanced Patterns: App.logger.exception() server logging & Correlation X-Request-ID header • Production Failure Mode & Pitfall Guard • Hands-On Live Coding Lab: Global JSON `@app.errorhandler(HTTPException)` Preserving Exact Status Codes • Idiomatic FLSK Implementation: Application Factory Blueprints, Pydantic/Marshmallow schemas & Gunicorn worker... • Flask 3 WSGI Request/App Context & SQLAlchemy 2 Session Execution & Memory Lifecycle for Global `@app.errorhandler` JSON/HTML Contracts

Purpose: To understand and apply Global `@app.errorhandler` JSON/HTML Contracts & Structured Production Logging in real Flask systems — specifically: Replacing default Werkzeug HT...

What Topic Is Next → Module 6 Capstone (CSRF-Protected Counseling Inquiry Form (`Flask-WTF`) & Dual JSON/HTML Error Handler) → Next Topic 7.1: Flask + `sqlite3` Integration — Request-Scoped `g.db`, `sqlite3.Row`, WAL Mode & Parameterized `?` SQL

Module 6 Hands-On Capstone Project (100 Marks): CSRF-Protected Counseling Inquiry Form (`Flask-WTF`) & Dual JSON/HTML Error Handler — Implement a `LeadInquiryForm(FlaskForm)` enforcing CSRF tokens, phone regex validation, and safe `?next=` post-login redirect validation (`urlsplit` host check), paired with a content-negoti...

MODULE 7 OF 16 • STAGE 2 • 14 LIVE + LAB HOURS • 4 TOPICS (32 SUBTOPICS) • 1 CAPSTONE PROJECT (100 MARKS)

Module 7: Relational Persistence — `SQLite3`, `Flask-SQLAlchemy` 3.1 (SQLAlchemy 2.0) & `Flask-Migrate`

Persist relational data cleanly from lightweight per-request `sqlite3` (`g.db` + `teardown_appcontext`) to modern typed `Flask-SQLAlchemy` 3.1 + **SQLAlchemy 2.0** (`Mapped`, `mapped_column`, `db.select`), solve ORM `N+1` queries (`selectinload`), and run Alembic `Flask-Migrate` migrations.

Objectives: ✓ Flask + `sqlite3` Integration — Request-Scoped `g.db`, `sqlite3.Row`, WAL Mode & Parameterized `?` SQL — Opening a lazy per-request `sqlite3` connection cached on **`g.db`**, enabling dict-like column access via `conn.row_factory = sqlite3.Row`, closing deterministically in `@app.teardown_appcontext`, enabling `PRAGMA foreign_keys = ON` + `journal_mode = WAL`, and `?` placeholders. • ✓ `Flask-SQLAlchemy` 3.1 & SQLAlchemy 2.0 ORM — `Mapped`, `mapped_column`, `db.select()` & Transactions — Binding `SQLAlchemy(model_class=Base)` via `db.init_app(app)`, declaring Python type-annotated models (`id: Mapped[int] = mapped_column(primary_key=True)`), executing composable **SQLAlchemy 2.0** queries (`db.session.scalars(db.select(Course).where(...))`), and atomic `db.session.commit()` / `rollback()`. • ✓ SQLAlchemy Relationships (`ForeignKey`, `relationship`) & Crushing ORM `N+1` Queries (`selectinload`) — Bidirectional `back_populates` (`Course.modules` <-> `Module.course`), indexing foreign key columns (`index=True`), why default lazy loading triggers **`N+1` SQL storms** inside loops/JSON serializers, and fixing `N+1` via **`selectinload()`** (for `1-to-N` collections) and **`joinedload()`** (for `N-to-1` parents)!. • ✓ Schema Migrations with `Flask-Migrate` (`Alembic`) & Production Connection Pool (`pool_pre_ping`) Tuning — Replacing destructive `db.create_all()` with version-controlled **`Flask-Migrate` (`Alembic`)** (`flask db init`, `flask db migrate -m "..."`, `flask db upgrade` / `downgrade`), reviewing auto-generated migration scripts, and configuring SQLAlchemy `QueuePool` (`pool_size`, `max_overflow`, `pool_pre_ping=True`). • ✓ Build, verify, and ship Module Capstone: SQLAlchemy 2.0 Typed Academy ORM (`Course` <-> `Module` <-> `Attempt`) with Zero-`N+1` Eager Loading.

7.1 Flask + `sqlite3` Integration — Request-Scoped `g.db`, `sqlite3.Row`, WAL Mode & Parameterized `?` SQL [50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Subtopics Covered: Per-request g.db + teardown_appcontext & Sqlite3.Row dict-style column access • Core Syntax, CLI & Production APIs: g.db = sqlite3.connect(db_path); g.db.row_factory = sqlite3.Row; cur.execute("SELECT * FROM... • Deep-Dive Mechanics: ? parameterized SQL placeholders! & Never share sqlite3 conn across threads • Advanced Patterns: PRAGMA foreign_keys=ON & WAL mode & Single-argument tuple (code,) comma! • Production Failure Mode & Pitfall Guard • Hands-On Live Coding Lab: Per-Request `sqlite3` Connection in `flask.g` with Automatic Teardown Cleanup • Idiomatic FLSK Implementation: Application Factory Blueprints, Pydantic/Marshmallow schemas & Gunicorn worker... • Flask 3 WSGI Request/App Context & SQLAlchemy 2 Session Execution & Memory Lifecycle for Flask + `sqlite3` Integration

Purpose: To understand and apply Flask + `sqlite3` Integration — Request-Scoped `g.db`, `sqlite3.Row`, WAL Mode & Parameterized `?` SQL in real Flask systems — specifically: Openi...

What Topic Is Next → Next Topic 7.2: `Flask-SQLAlchemy` 3.1 & SQLAlchemy 2.0 ORM — `Mapped`, `mapped_column`, `db.select()` & Transactions

7.2 `Flask-SQLAlchemy` 3.1 & SQLAlchemy 2.0 ORM — `Mapped`, `mapped_column`, `db.select()` & Transactions [50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Subtopics Covered: SQLAlchemy 2.0 Mapped & mapped_column & Db.session.scalars(db.select(...)) • Core Syntax, CLI & Production APIs: stmt = db.select(Course).where(Course.code == "FLSK"); course = db.session.scalar(stmt) • Deep-Dive Mechanics: Db.init_app(app) factory binding & Db.session.rollback() on IntegrityError! • Advanced Patterns: Avoid legacy Model.query API & Db.get_or_404(Course, id) helper • Production Failure Mode & Pitfall Guard • Hands-On Live Coding Lab: SQLAlchemy 2.0 Declarative `Mapped` Model Definition Pattern • Idiomatic FLSK Implementation: Application Factory Blueprints, Pydantic/Marshmallow schemas & Gunicorn worker... • Flask 3 WSGI Request/App Context & SQLAlchemy 2 Session Execution & Memory Lifecycle for `Flask-SQLAlchemy` 3.1

Purpose: To understand and apply `Flask-SQLAlchemy` 3.1 & SQLAlchemy 2.0 ORM — `Mapped`, `mapped_column`, `db.select()` & Transactions in real Flask systems — specifically: Bindin...

What Topic Is Next → Next Topic 7.3: SQLAlchemy Relationships (`ForeignKey`, `relationship`) & Crushing ORM `N+1` Queries (`selectinload`)

7.3 SQLAlchemy Relationships (`ForeignKey`, `relationship`) & Crushing ORM `N+1` Queries (`selectinload`)

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

Subtopics Covered: Selectinload() for 1-to-N collections! & Joinedload() for N-to-1 parent • Core Syntax, CLI & Production APIs: db.select(Course).options(selectinload(Course.modules)) • Deep-Dive Mechanics: Back_populates bidirectional typing & Explicit index=True on ForeignKey! • Advanced Patterns: Lazy="raise" catches N+1 in tests & Cascade="all, delete-orphan" • Production Failure Mode & Pitfall Guard • Hands-On Live Coding Lab: Comparing `N+1` Lazy Round-Trips vs `selectinload()` Constant 2-Query Batching • Idiomatic FLSK Implementation: Application Factory Blueprints, Pydantic/Marshmallow schemas & Gunicorn worker... • Flask 3 WSGI Request/App Context & SQLAlchemy 2 Session Execution & Memory Lifecycle for SQLAlchemy Relationships (`ForeignKey`, `relationshi...

Purpose: To understand and apply SQLAlchemy Relationships (`ForeignKey`, `relationship`) & Crushing ORM `N+1` Queries (`selectinload`) in real Flask systems — specifically: Bidire...

What Topic Is Next → Next Topic 7.4: Schema Migrations with `Flask-Migrate` (`Alembic`) & Production Connection Pool (`pool_pre_ping`) Tuning

7.4 Schema Migrations with `Flask-Migrate` (`Alembic`) & Production Connection Pool (`pool_pre_ping`) Tuning

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

Subtopics Covered: Flask-Migrate (Alembic) versioned DDL & Never rely on db.create_all() in prod • Core Syntax, CLI & Production APIs: Migrate(app, db) + SQLALCHEMY_ENGINE_OPTIONS = {"pool_pre_ping": True, "pool_recycle": 1800,... • Deep-Dive Mechanics: Review auto-generated Alembic scripts! & Pool_pre_ping=True stale-socket fix! • Advanced Patterns: Pool_size & max_overflow per Gunicorn worker & Expand-contract zero-downtime migrations • Production Failure Mode & Pitfall Guard • Hands-On Live Coding Lab: Calculating Total DB Connections Across Multi-Worker Gunicorn + SQLAlchemy Pools • Idiomatic FLSK Implementation: Application Factory Blueprints, Pydantic/Marshmallow schemas & Gunicorn worker... • Flask 3 WSGI Request/App Context & SQLAlchemy 2 Session Execution & Memory Lifecycle for Schema Migrations with `Flask-Migrate` (`Alembic`)

Purpose: To understand and apply Schema Migrations with `Flask-Migrate` (`Alembic`) & Production Connection Pool (`pool_pre_ping`) Tuning in real Flask systems — specifically: Rep...

What Topic Is Next → Module 7 Capstone (SQLAlchemy 2.0 Typed Academy ORM (`Course` <-> `Module` <-> `Attempt`) with Zero-`N+1` Eager Loading) → Next Topic 8.1: Class-Based REST APIs — `flask.views.MethodView`, `flask-smorest` & OpenAPI 3 / Swagger UI

Module 7 Hands-On Capstone Project (100 Marks): SQLAlchemy 2.0 Typed Academy ORM (`Course` <-> `Module` <-> `Attempt`) with Zero-`N+1` Eager Loading — Model `Course`, `Module`, and `MockAttempt` using `Flask-SQLAlchemy` 3.1 (`DeclarativeBase`, `Mapped[...]`, `ForeignKey` indexes), load nested course trees in 2 constant queries via `selecti...

MODULE 8 OF 16 • STAGE 2 • 11 LIVE + LAB HOURS • 3 TOPICS (24 SUBTOPICS) • 1 CAPSTONE PROJECT (100 MARKS)

Module 8: Production RESTful JSON APIs, `Pydantic` / `Marshmallow` Validation, CORS & Rate Limiting

Architect OpenAPI-documented REST microservices using `MethodView` / `flask-smorest`, validate JSON contracts strictly with `Pydantic v2` / `Marshmallow`, and protect public APIs with scoped `Flask-CORS`, `Flask-Limiter` (`429`), and `Flask-Caching`.

Objectives: ✓ Class-Based REST APIs — `flask.views.MethodView`, `flask-smorest` & OpenAPI 3 / Swagger UI — Grouping `get()`, `post()`, `patch()`, and `delete()` methods cleanly on a `MethodView` subclass (`CourseAPI.as_view("course_api")`), and using modern **`flask-smorest`** or **`APIFlask`** (instead of unmaintained legacy `Flask-RESTPlus`!) to auto-generate `/openapi.json` + Swagger UI. • ✓ API Schema Validation — `Pydantic v2` / `Marshmallow`, Mass-Assignment Defense & Cursor Pagination — Validating untrusted `request.get_json()` dicts against strict schemas (`Pydantic v2` Rust-backed `pydantic-core` or `Marshmallow` `Schema`), rejecting unknown fields (`extra="forbid"`), formatting `422 Unprocessable Entity` field errors, and scaling reads with Cursor (`WHERE id > :after_id LIMIT 20`) pagination. • ✓ `Flask-CORS`, Redis Rate Limiting (`Flask-Limiter` `429`) & `Flask-Caching` Acceleration — Allowing cross-origin React/Next.js SPA frontends safely via `Flask-CORS` (explicit origin allowlist!), stopping credential-stuffing & scraping attacks with **`Flask-Limiter`** (`429 Too Many Requests` + `Retry-After` backed by Redis!), and caching read-heavy views with `Flask-Caching`. • ✓ Build, verify, and ship Module Capstone: OpenAPI-Documented Flask 3 REST API with `Pydantic` Validation & Redis Rate Limiting.

8.1 Class-Based REST APIs — `flask.views.MethodView`, `flask-smorest` & OpenAPI 3 / Swagger UI

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

Subtopics Covered: Flask.views.MethodView class views & Decorators = [...] uniform guards • Core Syntax, CLI & Production APIs: class CourseAPI(MethodView): def get(self, course_id): ...; app.add_url_rule("/api/courses/<... • Deep-Dive Mechanics: APIFlask / flask-smorest OpenAPI 3 & Avoid deprecated reqparse / RESTPlus • Advanced Patterns: Semantic HTTP REST status codes & Resource-oriented plural nouns • Production Failure Mode & Pitfall Guard • Hands-On Live Coding Lab: Clean `MethodView` REST Resource Serving `GET` & `DELETE` (`204`) • Idiomatic FLSK Implementation: Application Factory Blueprints, Pydantic/Marshmallow schemas & Gunicorn worker... • Flask 3 WSGI Request/App Context & SQLAlchemy 2 Session Execution & Memory Lifecycle for Class-Based REST APIs

Purpose: To understand and apply Class-Based REST APIs — `flask.views.MethodView`, `flask-smorest` & OpenAPI 3 / Swagger UI in real Flask systems — specifically: Grouping `get()`,...

What Topic Is Next → Next Topic 8.2: API Schema Validation — `Pydantic v2` / `Marshmallow`, Mass-Assignment Defense & Cursor Pagination

8.2 API Schema Validation — `Pydantic v2` / `Marshmallow`, Mass-Assignment Defense & Cursor Pagination

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

Subtopics Covered: Never unpack User(**request.json)! & Pydantic extra="forbid" strict DTO • Core Syntax, CLI & Production APIs: Pydantic BaseModel(model_config=ConfigDict(extra="forbid", str_strip_whitespace=True)) | Mar... • Deep-Dive Mechanics: 422 field-level validation errors & Marshmallow dump_only / load_only • Advanced Patterns: Keyset cursor pagination vs OFFSET & Cap page size <= 100 • Production Failure Mode & Pitfall Guard • Hands-On Live Coding Lab: Strict Whitelist Payload Sanitizer Blocking Mass-Assignment Keys • Idiomatic FLSK Implementation: Application Factory Blueprints, Pydantic/Marshmallow schemas & Gunicorn worker... • Flask 3 WSGI Request/App Context & SQLAlchemy 2 Session Execution & Memory Lifecycle for API Schema Validation

Purpose: To understand and apply API Schema Validation — `Pydantic v2` / `Marshmallow`, Mass-Assignment Defense & Cursor Pagination in real Flask systems — specifically: Validatin...

What Topic Is Next → Next Topic 8.3: `Flask-CORS`, Redis Rate Limiting (`Flask-Limiter` `429`) & `Flask-Caching` Acceleration

8.3 `Flask-CORS`, Redis Rate Limiting (`Flask-Limiter` `429`) & `Flask-Caching` Acceleration [50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Subtopics Covered: Explicit CORS origin allowlist! & Flask-Limiter Redis storage_uri! • Core Syntax, CLI & Production APIs: CORS(app, resources={r"/api/*": {"origins": ALLOWED_ORIGINS}}) + Limiter(get_remote_address,... • Deep-Dive Mechanics: 429 Too Many Requests + Retry-After & @cache.cached(query_string=True)! • Advanced Patterns: Cache invalidation on write & ProxyFix accurate client IP • Production Failure Mode & Pitfall Guard • Hands-On Live Coding Lab: Simulating Sliding-Window Rate Limiter Returning `429` + `Retry-After` • Idiomatic FLSK Implementation: Application Factory Blueprints, Pydantic/Marshmallow schemas & Gunicorn worker... • Flask 3 WSGI Request/App Context & SQLAlchemy 2 Session Execution & Memory Lifecycle for `Flask-CORS`

Purpose: To understand and apply `Flask-CORS`, Redis Rate Limiting (`Flask-Limiter` `429`) & `Flask-Caching` Acceleration in real Flask systems — specifically: Allowing cross-orig...

What Topic Is Next → Module 8 Capstone (OpenAPI-Documented Flask 3 REST API with `Pydantic` Validation & Redis Rate Limiting) → Next Topic 9.1: Password Hashing (`werkzeug.security` `scrypt`) & Stateful Browser Auth (`Flask-Login`)

Module 8 Hands-On Capstone Project (100 Marks): OpenAPI-Documented Flask 3 REST API with `Pydantic` Validation & Redis Rate Limiting — Build `/api/v1/courses` and `/api/v1/attempts` `MethodView` resources validating payloads with `Pydantic` models, enforcing cursor pagination (`?after_id=...&limit=20`), scoping `CORS(app, r...

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

Module 9: Authentication & Security — `Werkzeug` Scrypt Hashing, `Flask-Login`, `JWT` & OWASP Hardening

Hash passwords securely with `werkzeug.security` (`scrypt`) / `Argon2`, build stateful browser auth with `Flask-Login` or stateless API auth with `Flask-JWT-Extended` (Access + Refresh rotation + Redis blocklist), and enforce HSTS/CSP security headers.

Objectives: ✓ Password Hashing (`werkzeug.security` `scrypt`) & Stateful Browser Auth (`Flask-Login`) — Why Werkzeug 3 upgraded `generate_password_hash()` default to memory-hard **`scrypt`** (`scrypt:32768:8:1$salt$hash`), timing-safe `check_password_hash()`, `Flask-Login`'s `@login_manager.user_loader`, `UserMixin`, `@login_required`, and regenerating session state on privilege change. • ✓ Stateless Token Auth with `Flask-JWT-Extended` — Access/Refresh Rotation, `jti` Blocklist & Claims — Issuing short-lived (15 min) `create_access_token(identity=str(user.id))` + rotating `create_refresh_token()`, storing tokens in `HttpOnly` cookies (with CSRF double-submit) vs `Authorization: Bearer`, and revoking compromised/logged-out tokens via `@jwt.token_in_blocklist_loader` in Redis!. • ✓ OWASP Flask Hardening — `Flask-Talisman` (`HSTS`, `CSP`), Subprocess/Pickle RCE & `TRUSTED_HOSTS` — Enforcing HTTPS & browser defense-in-depth headers via `Flask-Talisman` (`Strict-Transport-Security`, `Content-Security-Policy` nonces, `X-Frame-Options: DENY`), Flask 3.1's **`config["TRUSTED_HOSTS"]`** against Host-Header poisoning!, and banning `pickle.loads` / `shell=True`. • ✓ Build, verify, and ship Module Capstone: Role-Based Auth Service (`Werkzeug scrypt` + `JWT` Refresh Rotation + Redis Token Blocklist).

9.1 Password Hashing (`werkzeug.security` `scrypt`) & Stateful Browser Auth (`Flask-Login`) [50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Subtopics Covered: Generate_password_hash (scrypt) & Check_password_hash constant-time • Core Syntax, CLI & Production APIs: from werkzeug.security import generate_password_hash, check_password_hash | @login_required • Deep-Dive Mechanics: @functools.wraps(fn) on decorators! & Flask-Login @user_loader & UserMixin • Advanced Patterns: Prevent user enumeration timing & SESSION_PROTECTION = "strong" • Production Failure Mode & Pitfall Guard • Hands-On Live Coding Lab: Werkzeug `scrypt` Password Hashing & `@functools.wraps` RBAC Decorator • Idiomatic FLSK Implementation: Application Factory Blueprints, Pydantic/Marshmallow schemas & Gunicorn worker... • Flask 3 WSGI Request/App Context & SQLAlchemy 2 Session Execution & Memory Lifecycle for Password Hashing (`werkzeug.security` `scrypt`)

Purpose: To understand and apply Password Hashing (`werkzeug.security` `scrypt`) & Stateful Browser Auth (`Flask-Login`) in real Flask systems — specifically: Why Werkzeug 3 upgra...

What Topic Is Next → Next Topic 9.2: Stateless Token Auth with `Flask-JWT-Extended` — Access/Refresh Rotation, `jti` Blocklist & Claims

9.2 Stateless Token Auth with `Flask-JWT-Extended` — Access/Refresh Rotation, `jti` Blocklist & Claims [50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Subtopics Covered: Short-lived Access + Refresh rotation & Redis jti blocklist revocation • Core Syntax, CLI & Production APIs: create_access_token(identity=uid, additional_claims={"role": "student"}) + @jwt_required() • Deep-Dive Mechanics: HttpOnly cookie vs localStorage XSS & JWT payload is Base64URL readable! • Advanced Patterns: Separate JWT_SECRET_KEY from SECRET_KEY & Explicit algorithms=["HS256"] • Production Failure Mode & Pitfall Guard • Hands-On Live Coding Lab: Constant-Time HMAC Bearer Token Verification Helper • Idiomatic FLSK Implementation: Application Factory Blueprints, Pydantic/Marshmallow schemas & Gunicorn worker... • Flask 3 WSGI Request/App Context & SQLAlchemy 2 Session Execution & Memory Lifecycle for Stateless Token Auth with `Flask-JWT-Extended`

Purpose: To understand and apply Stateless Token Auth with `Flask-JWT-Extended` — Access/Refresh Rotation, `jti` Blocklist & Claims in real Flask systems — specifically: Issuing s...

What Topic Is Next → Next Topic 9.3: OWASP Flask Hardening — `Flask-Talisman` (`HSTS`, `CSP`), Subprocess/Pickle RCE & `TRUSTED_HOSTS`

9.3 OWASP Flask Hardening — `Flask-Talisman` (`HSTS`, `CSP`), Subprocess/Pickle RCE & `TRUSTED_HOSTS` [50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Subtopics Covered: App.config["TRUSTED_HOSTS"] (Flask 3)! & Host Header password-reset poisoning • Core Syntax, CLI & Production APIs: app.config["TRUSTED_HOSTS"] = ["mockattempt.com", ".mockattempt.com"] + Talisman(app, conten... • Deep-Dive Mechanics: Flask-Talisman HSTS & CSP nonces & NEVER use pickle on untrusted data! • Advanced Patterns: Yaml.safe_load() & shell=False & X-Content-Type-Options: nosniff • Production Failure Mode & Pitfall Guard • Hands-On Live Coding Lab: Verifying Flask 3 `TRUSTED_HOSTS` Blocking Spoofed `Host` Headers • Idiomatic FLSK Implementation: Application Factory Blueprints, Pydantic/Marshmallow schemas & Gunicorn worker... • Flask 3 WSGI Request/App Context & SQLAlchemy 2 Session Execution & Memory Lifecycle for OWASP Flask Hardening

Purpose: To understand and apply OWASP Flask Hardening — `Flask-Talisman` (`HSTS`, `CSP`), Subprocess/Pickle RCE & `TRUSTED_HOSTS` in real Flask systems — specifically: Enforcing...

What Topic Is Next → Module 9 Capstone (Role-Based Auth Service (`Werkzeug scrypt` + `JWT` Refresh Rotation + Redis Token Blocklist)) → Next Topic 10.1: Async Background Jobs with `Celery` 5 + `Redis` — `@shared_task`, Flask `AppContext` & `202 Accepted`

Module 9 Hands-On Capstone Project (100 Marks): Role-Based Auth Service (`Werkzeug scrypt` + `JWT` Refresh Rotation + Redis Token Blocklist) — Build `/api/auth/login`, `/api/auth/refresh`, and `/api/auth/logout` endpoints hashing passwords with `werkzeug.security.generate_password_hash`, issuing 15-minute Access JWTs + `HttpOnly` R...

MODULE 10 OF 16 • STAGE 3 • 14 LIVE + LAB HOURS • 4 TOPICS (32 SUBTOPICS) • 1 CAPSTONE PROJECT (100 MARKS)

Module 10: Background Workers (`Celery` + `Redis`), `Pytest` `test_client()` & `Gunicorn` + `Docker` Ops

Offload slow PDF/Email/AI workflows to `Celery` + `Redis`, achieve high-confidence CI with `pytest` fixtures + `app.test_client()`, and deploy production Flask behind `Gunicorn` + `Nginx` (`ProxyFix`) inside hardened multi-stage non-root Docker containers.

Objectives: ✓ Async Background Jobs with `Celery` 5 + `Redis` — `@shared_task`, Flask `AppContext` & `202 Accepted` — Binding Celery 5 to `create_app()` via a custom `FlaskTask` subclass (`with flask_app.app_context(): return self.run(*args, **kwargs)`), passing **primitive IDs** (`user_id: int`, never pickled SQLAlchemy models!) to `.delay()`, idempotent retries (`autoretry_for`), and `AsyncResult(id)` polling. • ✓ Automated Flask Testing with `pytest` — `app.test_client()`, Session Transactions & CLI Runner — Writing fast deterministic `pytest` fixtures (`@pytest.fixture def client(app): return app.test_client()`), inspecting/seeding signed sessions via `with client.session_transaction() as sess:`, testing Click commands via `app.test_cli_runner()`, and `TESTING=True` exception propagation. • ✓ Production WSGI Deployment — `Gunicorn` (`gthread` / `gevent`), `ProxyFix` & `Nginx` Reverse Proxy — Why Werkzeug's `flask run` dev server explicitly warns **"Do not use it in a production deployment"**, running **`gunicorn "app:create_app()" --workers 4 --threads 2 --timeout 30`**, recycling memory leaks cleanly with `--max-requests 1000 --max-requests-jitter 100`, and `Nginx` TLS/static termination + `ProxyFix`!. • ✓ Cloud-Native Flask — Multi-Stage Non-Root `Dockerfile`, `/healthz` Readiness Probes & Graceful `SIGTERM` — Building lean `python:3.12-slim` multi-stage Docker images (`PYTHONDONTWRITEBYTECODE=1`, `PYTHONUNBUFFERED=1`), running as unprivileged `USER appuser`, separating `/livez` (process alive) from `/readyz` (`SELECT 1` DB check), and handling Kubernetes `SIGTERM` graceful worker drain. • ✓ Build, verify, and ship Module Capstone: Production-Ready Dockerized Flask 3 + Gunicorn + Celery Microservice with `pytest` CI Suite.

10.1 Async Background Jobs with `Celery` 5 + `Redis` — `@shared_task`, Flask `AppContext` & `202 Accepted`

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

Subtopics Covered: Commit DB BEFORE task.delay(id)! & Pass primitive IDs not ORM objects • Core Syntax, CLI & Production APIs: @shared_task(bind=True, autoretry_for=(TimeoutError,), retry_backoff=True, max_retries=3) de... • Deep-Dive Mechanics: FlaskTask with app.app_context() & 202 Accepted + AsyncResult polling • Advanced Patterns: Idempotent tasks & retry_backoff & JSON task serializer (never pickle) • Production Failure Mode & Pitfall Guard • Hands-On Live Coding Lab: `202 Accepted` Job Enqueue & Status Polling Endpoint Pattern • Idiomatic FLSK Implementation: Application Factory Blueprints, Pydantic/Marshmallow schemas & Gunicorn worker... • Flask 3 WSGI Request/App Context & SQLAlchemy 2 Session Execution & Memory Lifecycle for Async Background Jobs with `Celery` 5 + `Redis`

Purpose: To understand and apply Async Background Jobs with `Celery` 5 + `Redis` — `@shared_task`, Flask `AppContext` & `202 Accepted` in real Flask systems — specifically: Bindin...

What Topic Is Next → Next Topic 10.2: Automated Flask Testing with `pytest` — `app.test_client()`, Session Transactions & CLI Runner

10.2 Automated Flask Testing with `pytest` — `app.test_client()`, Session Transactions & CLI Runner

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

Subtopics Covered: Pytest create_app(TESTING=True) fixture & In-memory WSGI app.test_client() • Core Syntax, CLI & Production APIs: with client.session_transaction() as sess: sess["user_id"] = 1 | res = client.post("/api/..... • Deep-Dive Mechanics: Client.session_transaction() helper! & App.test_cli_runner() Click tests • Advanced Patterns: Monkeypatch / responses HTTP mocking & Isolated per-test DB rollback • Production Failure Mode & Pitfall Guard • Hands-On Live Coding Lab: Testing Protected Session State Directly via `client.session_transaction()` • Idiomatic FLSK Implementation: Application Factory Blueprints, Pydantic/Marshmallow schemas & Gunicorn worker... • Flask 3 WSGI Request/App Context & SQLAlchemy 2 Session Execution & Memory Lifecycle for Automated Flask Testing with `pytest`

Purpose: To understand and apply Automated Flask Testing with `pytest` — `app.test_client()`, Session Transactions & CLI Runner in real Flask systems — specifically: Writing fast...

What Topic Is Next → Next Topic 10.3: Production WSGI Deployment — `Gunicorn` (`gthread` / `gevent`), `ProxyFix` & `Nginx` Reverse Proxy

10.3 Production WSGI Deployment — `Gunicorn` (`gthread` / `gevent`), `ProxyFix` & `Nginx` Reverse Proxy

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

Subtopics Covered: NEVER run flask run in production! & Gunicorn "app:create_app()" factory • Core Syntax, CLI & Production APIs: gunicorn "app:create_app()" -w 4 --threads 2 --max-requests 1000 --max-requests-jitter 100 -... • Deep-Dive Mechanics: ProxyFix x_for, x_proto, x_host! & Nginx slow-client buffering & TLS • Advanced Patterns: --max-requests + --max-requests-jitter & Gthread vs sync vs gevent workers • Production Failure Mode & Pitfall Guard • Hands-On Live Coding Lab: Wrapping Flask with Werkzeug `ProxyFix` Behind Reverse Proxy Headers • Idiomatic FLSK Implementation: Application Factory Blueprints, Pydantic/Marshmallow schemas & Gunicorn worker... • Flask 3 WSGI Request/App Context & SQLAlchemy 2 Session Execution & Memory Lifecycle for Production WSGI Deployment

Purpose: To understand and apply Production WSGI Deployment — `Gunicorn` (`gthread` / `gevent`), `ProxyFix` & `Nginx` Reverse Proxy in real Flask systems — specifically: Why Werkz...

What Topic Is Next → Next Topic 10.4: Cloud-Native Flask — Multi-Stage Non-Root `Dockerfile`, `/healthz` Readiness Probes & Graceful `SIGTERM`

10.4 Cloud-Native Flask — Multi-Stage Non-Root `Dockerfile`, `/healthz` Readiness Probes & Graceful `SIGTERM`

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

Subtopics Covered: Exec form CMD ["gunicorn", ...] SIGTERM! & PYTHONUNBUFFERED=1 instant container logs • Core Syntax, CLI & Production APIs: ENV PYTHONUNBUFFERED=1 | USER appuser | GET /livez -> 200, GET /readyz -> SELECT 1 DB check • Deep-Dive Mechanics: Non-root USER appuser security & /livez vs /readyz DB SELECT 1 probe • Advanced Patterns: Multi-stage wheel build slim image & Pinned requirements.txt / uv.lock • Production Failure Mode & Pitfall Guard • Hands-On Live Coding Lab: Separated Kubernetes `/livez` & `/readyz` Health Probe Endpoints in Flask • Idiomatic FLSK Implementation: Application Factory Blueprints, Pydantic/Marshmallow schemas & Gunicorn worker... • Flask 3 WSGI Request/App Context & SQLAlchemy 2 Session Execution & Memory Lifecycle for Cloud-Native Flask

Purpose: To understand and apply Cloud-Native Flask — Multi-Stage Non-Root `Dockerfile`, `/healthz` Readiness Probes & Graceful `SIGTERM` in real Flask systems — specifically: Bui...

What Topic Is Next → Module 10 Capstone (Production-Ready Dockerized Flask 3 + Gunicorn + Celery Microservice with `pytest` CI Suite) → Next Topic 11.1: create_app() Factory, Extensions Init & Multi-Env Config Classes

Module 10 Hands-On Capstone Project (100 Marks): Production-Ready Dockerized Flask 3 + Gunicorn + Celery Microservice with `pytest` CI Suite — Package a `create_app()` Flask 3 service with a `pytest` suite (`>= 90%` coverage using `test_client()`), an async `@shared_task` certificate PDF generator returning `202 Accepted` + `/tasks...

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

Application Factory Pattern, Modular Blueprints & Pydantic/Marshmallow

Production-grade Flask Training engineering module covering Application Factory Pattern, Modular Blueprints & Pydantic/Marshmallow with hands-on labs and capstone build.

Objectives: ✓ create_app() Factory, Extensions Init & Multi-Env Config Classes — Eliminate circular imports and instantiate isolated Flask apps cleanly for Pytest. • ✓ Domain Blueprints, Request/Response Schema Validation & Error JSON — Validate incoming JSON/query payloads strictly and return uniform error envelopes. • ✓ Custom Flask CLI Commands for Seeding, Maintenance & Audits — Automate operational admin tasks cleanly via flask <command> scripts. • ✓ Build, verify, and ship Module Capstone: Modular Multi-Blueprint SaaS Backend with Validated Schemas.

11.1 create_app() Factory, Extensions Init & Multi-Env Config Classes

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

Subtopics Covered: Plain • Production Implementation of Extensions Init • Integration with Domain Blueprints, Request/Response Schema Validation & Error JSON • Live Debugging, Telemetry & Technical Interview Trade • What Is create_app() Factory, Extensions Init & Multi • In Flask Training, create_app() Factory, Extensions Init & Multi • Even if you come from a non • Runnable Code Implementation (PYTHON)

Purpose: To understand and apply create_app() Factory, Extensions Init & Multi-Env Config Classes in real Flask systems — specifically: Eliminate circular imports and instantiate...

What Topic Is Next → Next Topic 11.2: Domain Blueprints, Request/Response Schema Validation & Error JSON

11.2 Domain Blueprints, Request/Response Schema Validation & Error JSON

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

Subtopics Covered: Plain • Production Implementation of Request/Response Schema Validation • Integration with Custom Flask CLI Commands for Seeding, Maintenance & Audits • Live Debugging, Telemetry & Technical Interview Trade • What Is Domain Blueprints, Request/Response Schema Validation & Error JSON & Why Industry Teams Use It • In Flask Training, Domain Blueprints, Request/Response Schema Validation & Error JSON exists for one clear real • Even if you come from a non • Runnable Code Implementation (PYTHON)

Purpose: To understand and apply Domain Blueprints, Request/Response Schema Validation & Error JSON in real Flask systems — specifically: Validate incoming JSON/query payloads str...

What Topic Is Next → Next Topic 11.3: Custom Flask CLI Commands for Seeding, Maintenance & Audits

11.3 Custom Flask CLI Commands for Seeding, Maintenance & Audits

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

Subtopics Covered: Plain • Production Implementation of Maintenance • Integration with create_app() Factory, Extensions Init & Multi • Live Debugging, Telemetry & Technical Interview Trade • What Is Custom Flask CLI Commands for Seeding, Maintenance & Audits & Why Industry Teams Use It • In Flask Training, Custom Flask CLI Commands for Seeding, Maintenance & Audits exists for one clear real • Even if you come from a non • Runnable Code Implementation (PYTHON)

Purpose: To understand and apply Custom Flask CLI Commands for Seeding, Maintenance & Audits in real Flask systems — specifically: Automate operational admin tasks cleanly via fla...

What Topic Is Next → Module 11 Capstone (Modular Multi-Blueprint SaaS Backend with Validated Schemas) → Next Topic 12.1: SQLAlchemy 2.0 Mapped[] Type-Safe Models & Unit-of-Work Sessions

Module 11 Hands-On Capstone Project (100 Marks): Modular Multi-Blueprint SaaS Backend with Validated Schemas — Design, implement, test, and document 'Modular Multi-Blueprint SaaS Backend with Validated Schemas' applying every production concept from Module 11 (Application Factory Pattern, Modular Blu...

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

SQLAlchemy 2.0 ORM, Alembic Migrations & Query Optimization

Production-grade Flask Training engineering module covering SQLAlchemy 2.0 ORM, Alembic Migrations & Query Optimization with hands-on labs and capstone build.

Objectives: ✓ SQLAlchemy 2.0 Mapped[] Type-Safe Models & Unit-of-Work Sessions — Prevent DetachedInstanceError bugs and manage transactional commits/rollbacks cleanly. • ✓ selectinload / joinedload Eager Loading & Composite Indexes — Eliminate N+1 ORM query storms on nested REST API serialization. • ✓ Alembic Autogenerate Reviews & Zero-Downtime Schema Upgrades — Version every table and index migration safely across staging and production. • ✓ Build, verify, and ship Module Capstone: High-Performance SQLAlchemy 2.0 Repository & Alembic Pipeline.

12.1 SQLAlchemy 2.0 Mapped[] Type-Safe Models & Unit-of-Work Sessions

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

Subtopics Covered: Plain • Production Implementation of Unit • Integration with selectinload / joinedload Eager Loading & Composite Indexes • Live Debugging, Telemetry & Technical Interview Trade • What Is SQLAlchemy 2.0 Mapped[] Type • In Flask Training, SQLAlchemy 2.0 Mapped[] Type • Even if you come from a non • Runnable Code Implementation (PYTHON)

Purpose: To understand and apply SQLAlchemy 2.0 Mapped[] Type-Safe Models & Unit-of-Work Sessions in real Flask systems — specifically: Prevent DetachedInstanceError bugs and mana...

What Topic Is Next → Next Topic 12.2: selectinload / joinedload Eager Loading & Composite Indexes

12.2 selectinload / joinedload Eager Loading & Composite Indexes [65 mins • Assignment (100M) • Hands-On Lab (50M) • 2 Mock Sets • 2 AI Vivas]

Subtopics Covered: Plain • Production Implementation of Composite Indexes • Integration with Alembic Autogenerate Reviews & Zero • Live Debugging, Telemetry & Technical Interview Trade • What Is selectinload / joinedload Eager Loading & Composite Indexes & Why Industry Teams Use It • In Flask Training, selectinload / joinedload Eager Loading & Composite Indexes exists for one clear real • Even if you come from a non • Runnable Code Implementation (PYTHON)

Purpose: To understand and apply selectinload / joinedload Eager Loading & Composite Indexes in real Flask systems — specifically: Eliminate N+1 ORM query storms on nested REST AP...

What Topic Is Next → Next Topic 12.3: Alembic Autogenerate Reviews & Zero-Downtime Schema Upgrades

12.3 Alembic Autogenerate Reviews & Zero-Downtime Schema Upgrades [65 mins • Assignment (100M) • Hands-On Lab (50M) • 2 Mock Sets • 2 AI Vivas]

Subtopics Covered: Plain • Production Implementation of Zero • Integration with SQLAlchemy 2.0 Mapped[] Type • Live Debugging, Telemetry & Technical Interview Trade • What Is Alembic Autogenerate Reviews & Zero • In Flask Training, Alembic Autogenerate Reviews & Zero • Even if you come from a non • Runnable Code Implementation (PYTHON)

Purpose: To understand and apply Alembic Autogenerate Reviews & Zero-Downtime Schema Upgrades in real Flask systems — specifically: Version every table and index migration safely...

What Topic Is Next → Module 12 Capstone (High-Performance SQLAlchemy 2.0 Repository & Alembic Pipeline) → Next Topic 13.1: JWT Access/Refresh Rotation, RBAC Decorators, CSRF & Talisman Headers

Module 12 Hands-On Capstone Project (100 Marks): High-Performance SQLAlchemy 2.0 Repository & Alembic Pipeline — Design, implement, test, and document 'High-Performance SQLAlchemy 2.0 Repository & Alembic Pipeline' applying every production concept from Module 12 (SQLAlchemy 2.0 ORM, Alembic Migrations...

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

JWT/Session Security, Rate Limiting, Redis Caching & Celery Workers

Production-grade Flask Training engineering module covering JWT/Session Security, Rate Limiting, Redis Caching & Celery Workers with hands-on labs and capstone build.

Objectives: ✓ JWT Access/Refresh Rotation, RBAC Decorators, CSRF & Talisman Headers — Protect Flask web and API routes against CSRF, XSS, and brute-force attacks. • ✓ Flask-Limiter Sliding Windows & Redis Response/Fragment Caching — Block abusive IP bursts and serve hot dashboard metrics in milliseconds. • ✓ Celery + Redis Background Workers, Retries & Periodic Beat Schedules — Offload slow report generation, webhooks, and emails out of Gunicorn request workers. • ✓ Build, verify, and ship Module Capstone: Hardened Flask API with Celery Async Tasks & Redis Throttling.

13.1 JWT Access/Refresh Rotation, RBAC Decorators, CSRF & Talisman Headers [65 mins • Assignment (100M) • Hands-On Lab (50M) • 2 Mock Sets • 2 AI Vivas]

Subtopics Covered: Plain • Production Implementation of RBAC Decorators • Integration with Flask • Live Debugging, Telemetry & Technical Interview Trade • What Is JWT Access/Refresh Rotation, RBAC Decorators, CSRF & Talisman Headers & Why Industry Teams Use It • In Flask Training, JWT Access/Refresh Rotation, RBAC Decorators, CSRF & Talisman Headers exists for one clear real • Even if you come from a non • Runnable Code Implementation (PYTHON)

Purpose: To understand and apply JWT Access/Refresh Rotation, RBAC Decorators, CSRF & Talisman Headers in real Flask systems — specifically: Protect Flask web and API routes again...

What Topic Is Next → Next Topic 13.2: Flask-Limiter Sliding Windows & Redis Response/Fragment Caching

13.2 Flask-Limiter Sliding Windows & Redis Response/Fragment Caching [65 mins • Assignment (100M) • Hands-On Lab (50M) • 2 Mock Sets • 2 AI Vivas]

Subtopics Covered: Plain • Production Implementation of Redis Response/Fragment Caching • Integration with Celery + Redis Background Workers, Retries & Periodic Beat Schedules • Live Debugging, Telemetry & Technical Interview Trade • What Is Flask • In Flask Training, Flask • Even if you come from a non • Runnable Code Implementation (PYTHON)

Purpose: To understand and apply Flask-Limiter Sliding Windows & Redis Response/Fragment Caching in real Flask systems — specifically: Block abusive IP bursts and serve hot dashbo...

What Topic Is Next → Next Topic 13.3: Celery + Redis Background Workers, Retries & Periodic Beat Schedules

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

Subtopics Covered: Plain • Production Implementation of Retries • Integration with JWT Access/Refresh Rotation, RBAC Decorators, CSRF & Talisman Headers • Live Debugging, Telemetry & Technical Interview Trade • What Is Celery + Redis Background Workers, Retries & Periodic Beat Schedules & Why Industry Teams Use It • In Flask Training, Celery + Redis Background Workers, Retries & Periodic Beat Schedules exists for one clear real • Even if you come from a non • Runnable Code Implementation (PYTHON)

Purpose: To understand and apply Celery + Redis Background Workers, Retries & Periodic Beat Schedules in real Flask systems — specifically: Offload slow report generation, webhook...

What Topic Is Next → Module 13 Capstone (Hardened Flask API with Celery Async Tasks & Redis Throttling) → Next Topic 14.1: Pytest App/Client Fixtures, Auth Helpers & Rollback Test Transactions

Module 13 Hands-On Capstone Project (100 Marks): Hardened Flask API with Celery Async Tasks & Redis Throttling — Design, implement, test, and document 'Hardened Flask API with Celery Async Tasks & Redis Throttling' applying every production concept from Module 13 (JWT/Session Security, Rate Limiting, R...

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

Pytest Client Suites, Gunicorn + Nginx Docker Stack & Observability

Production-grade Flask Training engineering module covering Pytest Client Suites, Gunicorn + Nginx Docker Stack & Observability with hands-on labs and capstone build.

Objectives: ✓ Pytest App/Client Fixtures, Auth Helpers & Rollback Test Transactions — Verify 100% of Flask blueprints and edge cases in fast, isolated test suites. • ✓ WSGI Production Serving with Gunicorn Workers/Threads & Nginx Proxy — Replace Flask's single-threaded dev server with hardened multi-worker production serving. • ✓ Structured Request-ID Logging, Sentry Alerts & /healthz Probes — Monitor live latency, exceptions, and DB connectivity in cloud production. • ✓ Build, verify, and ship Module Capstone: Containerized Production Flask Service Behind Nginx TLS.

14.1 Pytest App/Client Fixtures, Auth Helpers & Rollback Test Transactions

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

Subtopics Covered: Plain • Production Implementation of Auth Helpers • Integration with WSGI Production Serving with Gunicorn Workers/Threads & Nginx Proxy • Live Debugging, Telemetry & Technical Interview Trade • What Is Pytest App/Client Fixtures, Auth Helpers & Rollback Test Transactions & Why Industry Teams Use It • In Flask Training, Pytest App/Client Fixtures, Auth Helpers & Rollback Test Transactions exists for one clear real • Even if you come from a non • Runnable Code Implementation (PYTHON)

Purpose: To understand and apply Pytest App/Client Fixtures, Auth Helpers & Rollback Test Transactions in real Flask systems — specifically: Verify 100% of Flask blueprints and ed...

What Topic Is Next → Next Topic 14.2: WSGI Production Serving with Gunicorn Workers/Threads & Nginx Proxy

14.2 WSGI Production Serving with Gunicorn Workers/Threads & Nginx Proxy

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

Subtopics Covered: Plain • Production Implementation of Nginx Proxy • Integration with Structured Request • Live Debugging, Telemetry & Technical Interview Trade • What Is WSGI Production Serving with Gunicorn Workers/Threads & Nginx Proxy & Why Industry Teams Use It • In Flask Training, WSGI Production Serving with Gunicorn Workers/Threads & Nginx Proxy exists for one clear real • Even if you come from a non • Runnable Code Implementation (PYTHON)

Purpose: To understand and apply WSGI Production Serving with Gunicorn Workers/Threads & Nginx Proxy in real Flask systems — specifically: Replace Flask's single-threaded dev serv...

What Topic Is Next → Next Topic 14.3: Structured Request-ID Logging, Sentry Alerts & /healthz Probes

14.3 Structured Request-ID Logging, Sentry Alerts & /healthz Probes

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

Subtopics Covered: Plain • Production Implementation of Sentry Alerts • Integration with Pytest App/Client Fixtures, Auth Helpers & Rollback Test Transactions • Live Debugging, Telemetry & Technical Interview Trade • What Is Structured Request • In Flask Training, Structured Request • Even if you come from a non • Runnable Code Implementation (PYTHON)

Purpose: To understand and apply Structured Request-ID Logging, Sentry Alerts & /healthz Probes in real Flask systems — specifically: Monitor live latency, exceptions, and DB conn...

What Topic Is Next → Module 14 Capstone (Containerized Production Flask Service Behind Nginx TLS) → Next Topic 15.1: Pydantic/Marshmallow Request-Response Schemas & Uniform JSON Error Handlers

Module 14 Hands-On Capstone Project (100 Marks): Containerized Production Flask Service Behind Nginx TLS — Design, implement, test, and document 'Containerized Production Flask Service Behind Nginx TLS' applying every production concept from Module 14 (Pytest Client Suites, Gunicorn + Nginx Docke...

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

Production Flask API Architecture: Schemas, Caching & Background Workers

Self-explanatory list-and-bullet production mastery module covering Production Flask API Architecture: Schemas, Caching & Background Workers in Flask Training, paired with hands-on labs, capstone repository, Topic Mock Tests, and Spoken MockAttempt AI Viva.

Objectives: ✓ Pydantic/Marshmallow Request-Response Schemas & Uniform JSON Error Handlers — Purpose: Validate incoming JSON payloads strictly and return clean HTTP status contracts across Blueprints. • ✓ Redis Caching, Sliding-Window Rate Limiting & Idempotent POST Endpoints — Purpose: Protect Flask APIs from brute-force traffic bursts and duplicate form/payment submissions. • ✓ Offloading Slow Emails, PDFs & Webhooks to Background Redis Queue Workers — Purpose: Keep Flask HTTP response times under 50ms while heavy jobs execute asynchronously. • ✓ Build, verify, and document 'Blueprint-Modular Flask Service with Redis Rate Limits & Celery/RQ Workers' with clean Git commits and edge-case protection. • ✓ Clear 100% of Topic Timed Mock Tests and Spoken Voice + MockAttempt AI Technical Interviews.

15.1 Pydantic/Marshmallow Request-Response Schemas & Uniform JSON Error Handlers

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

Subtopics Covered: Security Hygiene & Data Safety Rules for Pydantic/Marshmallow Request • Performance Optimization, Complexity & Latency Impact of Pydantic/Marshmallow Request • Live Debugging, Stack • Runnable Code Implementation (PYTHON) • Hands • Idiomatic FLSK Implementation: Application Factory Blueprints, Pydantic/Marshmallow schemas & Gunicorn worker... • Flask 3 WSGI Request/App Context & SQLAlchemy 2 Session Execution & Memory Lifecycle for Pydantic/Marshmallow Request-Response Schemas • Production Hardening & Security: CSRF/XSS protection, JWT scopes & parameterized SQL

Purpose: To understand and apply Pydantic/Marshmallow Request-Response Schemas & Uniform JSON Error Handlers in real Flask systems — specifically: Validate incoming JSON payloads...

What Topic Is Next → Next Topic 15.2: Redis Caching, Sliding-Window Rate Limiting & Idempotent POST Endpoints

15.2 Redis Caching, Sliding-Window Rate Limiting & Idempotent POST Endpoints

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

Subtopics Covered: Security Hygiene & Data Safety Rules for Redis Caching, Sliding • Performance Optimization, Complexity & Latency Impact of Redis Caching, Sliding • Live Debugging, Stack • Runnable Code Implementation (PYTHON) • Hands • Idiomatic FLSK Implementation: Application Factory Blueprints, Pydantic/Marshmallow schemas & Gunicorn worker... • Flask 3 WSGI Request/App Context & SQLAlchemy 2 Session Execution & Memory Lifecycle for Redis Caching • Production Hardening & Security: CSRF/XSS protection, JWT scopes & parameterized SQL

Purpose: To understand and apply Redis Caching, Sliding-Window Rate Limiting & Idempotent POST Endpoints in real Flask systems — specifically: Protect Flask APIs from brute-force...

What Topic Is Next → Next Topic 15.3: Offloading Slow Emails, PDFs & Webhooks to Background Redis Queue Workers

15.3 Offloading Slow Emails, PDFs & Webhooks to Background Redis Queue Workers [65 mins • Assignment (100M) • Hands-On Lab (50M) • 2 Mock Sets • 2 AI Vivas]

Subtopics Covered: Security Hygiene & Data Safety Rules for Offloading Slow Emails, PDFs & Webhooks to Background Redis Queue Workers • Performance Optimization, Complexity & Latency Impact of Offloading Slow Emails, PDFs & Webhooks to Background Redi... • Live Debugging, Stack • Runnable Code Implementation (PYTHON) • Hands • Idiomatic FLSK Implementation: Application Factory Blueprints, Pydantic/Marshmallow schemas & Gunicorn worker... • Flask 3 WSGI Request/App Context & SQLAlchemy 2 Session Execution & Memory Lifecycle for Offloading Slow Emails • Production Hardening & Security: CSRF/XSS protection, JWT scopes & parameterized SQL

Purpose: To understand and apply Offloading Slow Emails, PDFs & Webhooks to Background Redis Queue Workers in real Flask systems — specifically: Keep Flask HTTP response times und...

What Topic Is Next → Module 15 Capstone (Blueprint-Modular Flask Service with Redis Rate Limits & Celery/RQ Workers) → Next Topic 16.1: SQLAlchemy Session Lifecycle, Alembic Safe Migrations & N+1 Eager Loading

Module 15 Hands-On Capstone Project (100 Marks): Blueprint-Modular Flask Service with Redis Rate Limits & Celery/RQ Workers — Architect, implement, test, and document 'Blueprint-Modular Flask Service with Redis Rate Limits & Celery/RQ Workers' applying every bullet-wise engineering concept from Module 15 (Productio...

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

Flask Security Hardening, Gunicorn Containerization & Backend Interviews

Self-explanatory list-and-bullet production mastery module covering Flask Security Hardening, Gunicorn Containerization & Backend Interviews in Flask Training, paired with hands-on labs, capstone repository, Topic Mock Tests, and Spoken MockAttempt AI Viva.

Objectives: ✓ SQLAlchemy Session Lifecycle, Alembic Safe Migrations & N+1 Eager Loading — Purpose: Prevent connection leaks and schema drift across development, staging, and production. • ✓ OWASP Security Headers, CSRF/CORS Policies, Secret Rotation & Gunicorn Tuning — Purpose: Size Gunicorn workers/threads properly and harden cookies and headers for HTTPS production. • ✓ Flask Application Context Internals & Python Backend Interview Walkthroughs — Purpose: Explain app/request context locals, WSGI vs ASGI, and clean factory testing in interviews. • ✓ Build, verify, and document 'Non-Root Dockerized Gunicorn + Flask Stack with pytest Coverage Gate' with clean Git commits and edge-case protection. • ✓ Clear 100% of Topic Timed Mock Tests and Spoken Voice + MockAttempt AI Technical Interviews.

16.1 SQLAlchemy Session Lifecycle, Alembic Safe Migrations & N+1 Eager Loading [65 mins • Assignment (100M) • Hands-On Lab (50M) • 2 Mock Sets • 2 AI Vivas]

Subtopics Covered: Security Hygiene & Data Safety Rules for SQLAlchemy Session Lifecycle, Alembic Safe Migrations & N+1 Eager Loading • Performance Optimization, Complexity & Latency Impact of SQLAlchemy Session Lifecycle, Alembic Safe Migrations & N+... • Live Debugging, Stack • Runnable Code Implementation (PYTHON) • Hands • Idiomatic FLSK Implementation: Application Factory Blueprints, Pydantic/Marshmallow schemas & Gunicorn worker... • Flask 3 WSGI Request/App Context & SQLAlchemy 2 Session Execution & Memory Lifecycle for SQLAlchemy Session Lifecycle • Production Hardening & Security: CSRF/XSS protection, JWT scopes & parameterized SQL

Purpose: To understand and apply SQLAlchemy Session Lifecycle, Alembic Safe Migrations & N+1 Eager Loading in real Flask systems — specifically: Prevent connection leaks and schem...

What Topic Is Next → Next Topic 16.2: OWASP Security Headers, CSRF/CORS Policies, Secret Rotation & Gunicorn Tuning

16.2 OWASP Security Headers, CSRF/CORS Policies, Secret Rotation & Gunicorn Tuning [65 mins • Assignment (100M) • Hands-On Lab (50M) • 2 Mock Sets • 2 AI Vivas]

Subtopics Covered: Security Hygiene & Data Safety Rules for OWASP Security Headers, CSRF/CORS Policies, Secret Rotation & Gunicorn Tun... • Performance Optimization, Complexity & Latency Impact of OWASP Security Headers, CSRF/CORS Policies, Secret Rotatio... • Live Debugging, Stack • Runnable Code Implementation (PYTHON) • Hands • Idiomatic FLSK Implementation: Application Factory Blueprints, Pydantic/Marshmallow schemas & Gunicorn worker... • Flask 3 WSGI Request/App Context & SQLAlchemy 2 Session Execution & Memory Lifecycle for OWASP Security Headers • Production Hardening & Security: CSRF/XSS protection, JWT scopes & parameterized SQL

Purpose: To understand and apply OWASP Security Headers, CSRF/CORS Policies, Secret Rotation & Gunicorn Tuning in real Flask systems — specifically: Size Gunicorn workers/threads...

What Topic Is Next → Next Topic 16.3: Flask Application Context Internals & Python Backend Interview Walkthroughs

16.3 Flask Application Context Internals & Python Backend Interview Walkthroughs [65 mins • Assignment (100M) • Hands-On Lab (50M) • 2 Mock Sets • 2 AI Vivas]

Subtopics Covered: Security Hygiene & Data Safety Rules for Flask Application Context Internals & Python Backend Interview Walkthrough... • Performance Optimization, Complexity & Latency Impact of Flask Application Context Internals & Python Backend Inter... • Live Debugging, Stack • Runnable Code Implementation (PYTHON) • Hands • Idiomatic FLSK Implementation: Application Factory Blueprints, Pydantic/Marshmallow schemas & Gunicorn worker... • Flask 3 WSGI Request/App Context & SQLAlchemy 2 Session Execution & Memory Lifecycle for Flask Application Context Internals • Production Hardening & Security: CSRF/XSS protection, JWT scopes & parameterized SQL

Purpose: To understand and apply Flask Application Context Internals & Python Backend Interview Walkthroughs in real Flask systems — specifically: Explain app/request context loca...

What Topic Is Next → Module 16 Capstone → End-of-Course Live Client Project: LeadPulse Lightweight Webhook Gateway, Partner API & Admin Micro-Portal

Module 16 Hands-On Capstone Project (100 Marks): Non-Root Dockerized Gunicorn + Flask Stack with pytest Coverage Gate — Architect, implement, test, and document 'Non-Root Dockerized Gunicorn + Flask Stack with pytest Coverage Gate' applying every bullet-wise engineering concept from Module 16 (Flask Security...

3. LIVE CLIENT PRODUCTION PROJECT & CAREER SCOPE

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

Project: LeadPulse Lightweight Webhook Gateway, Partner API & Admin Micro-Portal • **Confirmed Role:** Python Flask API & Web Engineering Intern

Domain & Production Infra: MarTech SaaS & CRM Integration Client (Python 3.12 + Flask 3 Blueprints + SQLAlchemy 2 + Marshmallow + Gunicorn + Nginx TLS)

- Structured modular Flask 3 applications using the Application Factory pattern, Blueprints, and environment configs.
- Built RESTful partner endpoints with Marshmallow/Pydantic request serialization, JWT auth, and rate limiting.
- Managed relational persistence with SQLAlchemy 2 ORM, Alembic/Flask-Migrate migrations, and Redis caching.
- Deployed multi-worker Gunicorn + Nginx HTTPS containers with pytest test suites to live client production.

Target Job Roles: Python Flask Backend Developer, Junior Python Web & API Engineer, ML API Integration Engineer, Backend Microservices Associate

Realistic Annual CTC Scope: **Fresher / Entry-Level (0–1 Yr):** ₹4.5 LPA – ₹8.5 LPA **Mid-Level Python Web Engineer (1–3 Yrs):** ₹9.0 LPA – ₹17.0 LPA **Senior Python Backend / AI Platform Lead (3–6+ Yrs):** ₹18.0 LPA – ₹34.0+ LPA

MockAttempt Academy Pvt. Ltd. • Email: support@mockattempt.com • Phone/WhatsApp: +91 95400 07917 • Web: https://mockattempt.com/courses/flask-training
Try 2 Free Topic-Wise Timed Topic Mock Tests & Spoken MockAttempt AI Interviews daily on every topic above before enrolling.