Full-Stack AI-Enabled Developer — Course Outline

Course vision, scope, and learning outcomes

- Vision
    - Train engineers to design, implement, and operate production-grade full-stack applications that integrate modern AI capabilities (LLMs, RAG, agents) responsibly and cost-effectively.
    - Emphasize separation of concerns: concepts → implementation → architecture → deployment. Produce portfolio-ready, deployable systems that follow industry best practices in security, observability, and scalability.
- Scope
    - Tech stack: Python (async, typing), FastAPI, PostgreSQL (SQLAlchemy/Alembic), Redis, vector DB (Pinecone/Weaviate/Milvus/FAISS), Next.js (App Router + server/client components), Docker, CI/CD (GitHub Actions), IaC (Terraform/CDK), cloud deployment (AWS/GCP/Azure or managed PaaS), OpenAI/Vertex/Anthropic APIs + LangChain/Semantic Kernel patterns.
    - Topics: REST API design, DB modeling and migrations, auth & security (JWT/OAuth2/OIDC), frontend BFF patterns, RAG pipelines, agents and tool integrations, testing, observability (OpenTelemetry), performance tuning, cost control, and production deployment strategies.
- Measurable Learning Outcomes (by course end)
    - Design API-first systems and author OpenAPI contracts.
    - Implement async, typed Python services with FastAPI and Pydantic.
    - Model data with Postgres + SQLAlchemy, write and apply Alembic migrations, and use Redis for caching.
    - Implement secure auth flows (JWT, OAuth2 with PKCE) and role-based access.
    - Build Next.js frontends using server/client components and BFF integration patterns.
    - Create RAG pipelines: ingestion, embedding, vector index, retrieval, reranking, and prompt assembly with provenance.
    - Orchestrate LLM tool/agent workflows safely using function-calling patterns.
    - Deploy full-stack apps to cloud with Docker, CI/CD, IaC, autoscaling, and observability.
    - Demonstrate responsible AI practices: data minimization, logging/redaction, human-in-the-loop.
    - Produce deployable portfolio projects with architecture docs, metrics dashboards, and cost controls.

Course structure and recommended pacing

- Total suggested duration: 16–20 weeks (can be condensed or extended). Each section includes lessons, objectives, labs, and assessments.
- Delivery modes supported: instructor-led, self-paced, or hybrid. Each section contains a mini-project (hands-on) and checkpoints.

Section-wise curriculum (section title, purpose, lessons with objectives)

Section 0 — Onboarding & Foundations (1 week)

- Purpose
  - Ensure all learners share minimal prerequisites and tooling. Setup local dev environments and course repos.
- Lessons
  - L0.1 — Course orientation & workflow
    - Objectives: Understand course goals, deliverables, repo templates, CI flow, and assessment rubric.
  - L0.2 — Developer tooling & environment
    - Objectives: Setup git, GitHub, Python venv/Poetry, Docker, Node/PNPM/Yarn, VSCode, linters (ruff, eslint), pre-commit, and secrets manager patterns.
  - L0.3 — Web & infra basics refresher
    - Objectives: HTTP basics, status codes, TLS, DNS, basic Docker concepts, cloud account setup (free tier), and cost safety guardrails.
- Deliverable: Working project skeleton cloned and CI pipeline passing; "Hello World" FastAPI + Next.js minimal demo running locally and in Docker.

Section 1 — Python & Web Foundations (2 weeks)

- Purpose
  - Build strong Python fundamentals focused on async, type-safety, packaging, and production readiness required for backend systems.
- Lessons
  - L1.1 — Modern Python idioms & typing
    - Objectives: Use type hints, dataclasses, pydantic BaseModel vs domain models, run mypy/pyright; explain trade-offs of gradual typing.
  - L1.2 — Asyncio, concurrency, and I/O patterns
    - Objectives: Understand event loop, async/await, tasks, run_in_executor, avoid blocking calls, and when to use threads/process pools.
  - L1.3 — Packaging, dependency management & reproducible builds
    - Objectives: Use Poetry or pip-tools, lockfiles, semantic versioning, and multi-env workflows.
  - L1.4 — Code quality: linters, formatting, and testing basics
    - Objectives: Set up ruff/black, pytest, TDD basics, and writing small unit

        tests.

- Mini-Project 1: CLI ETL tool
    - Build a typed async CLI that ingests a CSV or web JSON, does light transformation, stores results to local SQLite or Postgres. Include unit tests, type checks, and packaging (poetry + pyproject). Deliverables: README, tests passing, package artifact.

Section 2 — REST APIs & FastAPI Backend Development (3 weeks)

- Purpose
    - Teach API-first design, FastAPI fundamentals and best practices, DI, validation, background tasks, and OpenAPI generation.
- Lessons
    - L2.1 — REST fundamentals & API-first design
        - Objectives: Design resources, idempotency, status codes, error schemas (RFC 7807), pagination patterns, and versioning strategies.
    - L2.2 — FastAPI basics: routes, Pydantic, and auto-docs
        - Objectives: Build routes, request/response models, automatic OpenAPI docs, and interactive Swagger UI.
    - L2.3 — Dependency Injection, middleware, and lifecycle events
        - Objectives: Implement DB session DI, config injection, startup/shutdown hooks.
    - L2.4 — Background tasks, tasks vs queues, and worker patterns
        - Objectives: Use FastAPI BackgroundTasks for small tasks; integrate Celery/RQ or cloud task queues for durable work.
    - L2.5 — API security basics, rate-limiting, and CORS
        - Objectives: Implement CORS policies, basic rate-limiting middleware, and secure headers.
    - L2.6 — Testing FastAPI services
        - Objectives: Write unit and integration tests using pytest, TestClient, and DB test fixtures.
- Mini-Project 2: CRUD API service
    - Implement a fully-typed FastAPI CRUD service with OpenAPI contract, request/response validation, DI for Postgres session, unit/integration tests, and Dockerfile. Must include structured logs and health endpoints.

Section 3 — Databases & Data Management (2 weeks)

- Purpose

    - Teach relational modeling, async DB integration, migrations, caching, NoSQL overview, and vector store basics for RAG prep.
- Lessons
    - L3.1 — Relational modeling & Postgres essentials
        - Objectives: Normalize/denormalize trade-offs, indices, transactions, isolation, EXPLAIN ANALYZE basics.
    - L3.2 — ORMs and async database access
        - Objectives: Use SQLAlchemy (async guidance) or Prisma/Tortoise, connection pooling, yield-based session DI pattern.
    - L3.3 — Migrations with Alembic
        - Objectives: Author migrations, run in CI/CD, handle backward compatibility strategies.
    - L3.4 — Caching patterns & Redis
        - Objectives: Cache-control headers, Redis for TTL caches and rate-limiting, caching invalidation strategies.
    - L3.5 — Vector DBs & embedding storage primer
        - Objectives: Vector vs metadata storage, index types, metadata filtering, and integration choices (Pinecone/Weaviate/Milvus/FAISS).
- Mini-Project 3: Data layer & ingestion pipeline
    - Implement a small ingestion pipeline that persists structured data to Postgres and caches hot results in Redis; create a vectorization step that stores embeddings in a vector DB (can be emulated locally). Include migrations and DB schema docs.

Section 4 — Authentication, Security & Authorization (2 weeks)

- Purpose
    - Provide practical knowledge to secure APIs and frontends: JWT/OAuth2/OIDC, session handling, RBAC/ABAC, and secrets management.
- Lessons
    - L4.1 — Authentication fundamentals & OAuth2 flows
        - Objectives: Differentiate OAuth flows (Authorization Code + PKCE vs Client Credentials), OIDC, and when to use managed IdP.
    - L4.2 — JWT design and token lifecycle
        - Objectives: Token structure, signing/validation, refresh tokens, revocation strategies, cookie vs storage decisions.
    - L4.3 — Authorization patterns & RBAC/ABAC

        - Objectives: Implement role checks, permission middleware, and fine-grained policy enforcement.
    - L4.4 — Secure coding & OWASP
        - Objectives: Input validation, XSS/CSRF protections, SQL injection countermeasures, dependency scanning.
    - L4.5 — Secrets & key management
        - Objectives: Use secret stores (AWS/GCP/HashiCorp), rotation, and least privilege for API keys and service accounts.
- Mini-Project 4: Secure auth flow
    - Create an auth system that issues short-lived JWTs and refresh tokens (server-side refresh rotation). Integrate with Next.js front-end cookie-based session using HttpOnly secure cookies. Provide middleware for role checks and demonstration of token revocation.

Section 5 — Frontend Development with Next.js (3 weeks)

- Purpose
    - Teach modern React and Next.js (app router, server/client components), BFF patterns, and typed API integration using generated OpenAPI types.
- Lessons
    - L5.1 — React & TypeScript fundamentals for production
        - Objectives: Hooks, Suspense basics, component composition, and typing patterns.
    - L5.2 — Next.js rendering strategies & App Router
        - Objectives: SSR, SSG, ISR, server components, client components, and when to use each for performance and UX trade-offs.
    - L5.3 — API integration & BFF patterns
        - Objectives: BFF role, typed client generation (OpenAPI → TypeScript), caching with React Query or SWR, and global error handling.
    - L5.4 — Edge functions & performance
        - Objectives: Use edge middleware for personalization and A/B testing; static asset optimization and CDNs.
    - L5.5 — Frontend testing & accessibility
        - Objectives: Unit tests with Jest + React Testing Library; E2E with Playwright/Cypress; basic accessibility checks.
- Mini-Project 5: Frontend BFF demo

    - Build a Next.js app that consumes the earlier FastAPI backend via a BFF layer and displays data with client caching and optimistic updates. Include authentication flows and E2E test for critical user flows.

Section 6 — Full-Stack Integration & Mid-Course Project (2 weeks)

- Purpose
    - Integrate frontend and backend into a cohesive product and practice contract-first development with OpenAPI.
- Lessons
    - L6.1 — Contract-first workflows & typed clients
        - Objectives: Author OpenAPI specs, generate server/client stubs, and keep types in sync using CI checks.
    - L6.2 — Error handling, retries, and resilience patterns
        - Objectives: Implement exponential backoff, circuit breakers, graceful degradation, and retries on transient failures.
    - L6.3 — CI/CD basics for full-stack apps
        - Objectives: Pipeline that runs lint/tests, builds images/Next app, runs migrations, deploys to staging, and runs smoke tests.
- Mid-course Project (Integrated)
    - Project: Knowledge Base Q&A (RAG Lite) with BFF
        - Spec:
            - Next.js frontend for uploading documents and asking questions.
            - FastAPI backend that exposes endpoints for uploading docs, indexing (chunking + embed), and question answering via a vector DB + LLM.
            - Use server-side RAG: ingest small corpus (e.g., product docs), store embeddings, retrieve top-K passages, assemble prompt with provenance, call model API, and return answer with sources.
            - Implement auth, logging (structured), minimal observability (metrics), and unit/integration tests.
        - Deliverables:
            - OpenAPI spec, generated TS types used in Next.js, Postgres schema + Alembic migrations, vector DB usage, cost estimation for LLM usage, and README with run/deploy instructions.
        - Evaluation criteria:
            - Functional correctness (ingest & answer), provenance shown, basic security and tests, and a deployed demo (staging).

Section 7 — AI Integration with Cloud APIs (3 weeks)

- Purpose
    - Teach LLM API integration patterns, prompt engineering, batching, streaming, monitoring, cost control, and proxy/BFF strategies.
- Lessons
    - L7.1 — Cloud LLM APIs & selection
        - Objectives: Compare providers (OpenAI, Anthropic, Vertex, Bedrock), model trade-offs, latency/cost/security factors.
    - L7.2 — Prompt engineering & prompt templates
        - Objectives: System/user/assistant roles, few-shot patterns, prompt templates with variables, token budgeting.
    - L7.3 — API integration patterns: streaming, batching, and retries
        - Objectives: Implement streaming responses for UX, batch requests to reduce overhead, handle rate limits and backoff.
    - L7.4 — Proxying & data governance
        - Objectives: Build server-side proxy that centralizes calls, sanitizes inputs, masks PII, and records telemetry.
    - L7.5 — Observability & cost monitoring for model usage
        - Objectives: Instrument token counts, model latencies, usage dashboards, alerts for cost thresholds.
- Lab: Model adapter & cost dashboard
    - Implement an adapter service that abstracts provider selection, supports streaming and function-calling, logs usage metrics, and emits events to metrics backend. Build a simple dashboard showing model calls, tokens per call, and cost estimates.

Section 8 — Advanced AI Patterns (RAG, tools, agents) (3 weeks)

- Purpose
    - Cover RAG at scale, hybrid search, reranking, agent frameworks, function calling, tool orchestration, and safety guardrails.
- Lessons
    - L8.1 — RAG architecture & production patterns
        - Objectives: Build ingestion pipeline (chunking, embeddings, metadata), incremental reindexing, and re-ranking layers.
    - L8.2 — Vector DB design and scaling
        - Objectives: Index configuration (HNSW, IVF, PQ), sharding/replication, filtering by metadata, and maintenance/reindex strategies.

    - L8.3 — Hybrid search & reranking
        - Objectives: Combine lexical (BM25) and semantic retrieval, implement reranker cross-encoder, and improve recall/precision trade-offs.
    - L8.4 — Agents & function-calling orchestration
        - Objectives: Implement function schemas, validate function args, sandbox tool execution, audit logs, and fallbacks.
    - L8.5 — Safety, guardrails & human-in-the-loop
        - Objectives: Limit agent reach, approval flows for destructive actions, red-team test cases, and monitoring suspicious patterns.
- Mini-Project 6: Agent-powered Assistant
    - Build an assistant that can (safely) perform simple actions: query a database, call external APIs, and summarize results. Use function calling with schema validation, execution sandbox, and audit logs. Include a reranker to improve retrieval results.

Section 9 — Testing, Observability & Performance (2 weeks)

- Purpose
    - Teach observability best practices (OpenTelemetry), testing strategies for non-deterministic AI components, and performance tuning for both app and model calls.
- Lessons
    - L9.1 — Testing AI systems
        - Objectives: Unit tests, mocking LLM responses (fixtures), contract tests, property-based tests for pipelines, and CI strategies to avoid cost.
    - L9.2 — Tracing, metrics & structured logging
        - Objectives: Instrument FastAPI/Next/worker services with OpenTelemetry, exports to Jaeger/Prometheus/Grafana, correlate traces for LLM calls.
    - L9.3 — Load testing & performance tuning
        - Objectives: Use Locust/k6 to simulate user flows (including RAG/model calls), identify p95/p99 latencies, and optimize DB queries and event loop blocking.
    - L9.4 — SLOs, alerting & incident playbooks
        - Objectives: Define SLOs for critical APIs, set up alerting thresholds, and write incident response checklists and postmortems.
- Lab: Observability pipeline + load test
    - Add distributed tracing and metrics to mid-course project, create dashboards for

model latency and token usage, perform a load test and present an optimization plan.

Section 10 — DevOps, Deployment & Scaling (2 weeks)

- Purpose
  - Teach containerization, IaC, CI/CD, deployment patterns (serverless vs k8s), autoscaling, secrets management, and cost controls.
- Lessons
  - L10.1 — Containerization & image best practices
    - Objectives: Multi-stage Dockerfiles, minimal base images, non-root users, healthchecks, and multi-arch images.
  - L10.2 — CI/CD pipelines, migrations, and deploy strategies
    - Objectives: GitHub Actions pipeline templates: build/test/lint/build-image/push/deploy, run migrations with safe rollouts, blue/green/canary.
  - L10.3 — Infra as Code & environment parity
    - Objectives: Terraform or CDK for core infra (DB, vector DB, secrets), manage staging and production parity and IaC review process.
  - L10.4 — Autoscaling, serverless caveats, and connection pooling
    - Objectives: Design stateless services, use connection pooling for DBs (pgbouncer), serverless cold start trade-offs, and use spot instances for batch.
  - L10.5 — Cost governance & tagging
    - Objectives: Tag resources, set budgets/alerts, and pick cost-optimized instance families.
- Mini-Project 7: Deploy to cloud
  - Deploy mid-course project to cloud staging using Docker + managed services or serverless (as chosen). Include CI pipeline, IaC definitions, secrets, healthchecks, and scaling rules. Demonstrate canary/blue deployment.

Section 11 — Capstone Preparation & Architecture (1 week)

- Purpose
  - Guide students to design their final capstone: architecture, data flows, security, and testing plan.
- Lessons
  - L11.1 — Capstone architecture & design doc
    - Objectives: Produce an architecture diagram, API contract (OpenAPI), data models, scaling plan, and security checklist.

    - L11.2 — Project planning & milestones
        - Objectives: Break capstone into sprints, define acceptance criteria and deliverables (MVP + stretch).
- Deliverable: Final capstone proposal and project repo scaffold.

Section 12 — Final Capstone Project (3–4 weeks)

- Purpose
    - Deliver a production-ready full-stack AI product that demonstrates all learnings: secure, observable, scalable, and cost-controlled.
- Capstone options (examples)
    - Option A — AI-Powered Customer Support Platform
        - Features: Document ingestion, RAG-based knowledge assistant with source attribution, triage + human-in-the-loop escalation, agent-based ticket actions (create/update), role-based access, analytics dashboard, and deployment to cloud.
    - Option B — Personal Research Assistant for Teams
        - Features: Ingest internal docs (Google Drive/Slack/Confluence connectors), RAG search, multi-doc summarization, scheduled re-indexing pipelines, access controls, and audit logs.
    - Option C — Agent-Assisted Data Ops Dashboard
        - Features: Query data, run safe database transformations via function calling and approval flows, change tracking, and rollback features.
- Capstone Requirements
    - Must include: Next.js frontend, FastAPI backend(s), Postgres with Alembic migrations, Redis cache, vector DB ingestion & retrieval, LLM integration (with cost & telemetry), auth (OIDC or managed IdP), CI/CD pipeline, IaC manifests, observability (traces/metrics/logs), load tests, and security checklist.
- Deliverables
    - Public repo with code + README, architecture doc, deployment scripts + instructions, deployed staging URL, automated tests, and a short demo video (5–10 minutes).
- Evaluation rubric
    - Functionality (30%), Security & Privacy (15%), Observability & Testing (15%), Architecture & Scalability (15%), Code quality & docs (15%), UX & Demo (10%).

Skill progression mapping: Beginner → Intermediate → Advanced

- Beginner (Foundations)

  - Skills: Python basics, HTTP/REST basics, Git, basic SQL, simple scripts, basic React.
  - Outcomes: Build simple APIs and static frontends; run services locally.

- Intermediate (Core engineering)

  - Skills: Async Python, FastAPI, Pydantic, SQLAlchemy/Alembic, Next.js SSR/SSG, Docker, basic LLM API use, JWT/OAuth2, unit + integration testing.
  - Outcomes: Build and deploy a full-stack app with secure auth, typed contracts, basic RAG pipeline and vector DB usage; instrumented with logging and simple metrics.

- Advanced (Production & AI systems)

  - Skills: Agent orchestration, hybrid search & reranking, vector DB scaling, advanced CI/CD and IaC, Kubernetes or serverless patterns at scale, advanced observability (SLOs, tracing), responsible AI governance.
  - Outcomes: Architect and operate production-grade AI-enabled systems that are scalable, cost-aware, secure, and auditable; lead system design and postmortems.

Mapping of modules to skill level

- Beginner: Sections 0–2
- Intermediate: Sections 3–7
- Advanced: Sections 8–12

Hands-on projects (module-level mini projects, mid-course project, final capstone)

- Mini-projects (module level; checklist and deliverable examples)

  - Mini 1 (Foundations): Async CLI ETL tool (tests + packaging)
  - Mini 2 (FastAPI): CRUD API with OpenAPI + Docker
  - Mini 3 (DB): Ingestion pipeline + embeddings stub
  - Mini 4 (Auth): JWT + refresh token implementation and Next.js cookie-based session
  - Mini 5 (Frontend): Next.js BFF with typed clients and optimistic updates
  - Mini 6 (Agents): Function-calling agent with sandboxed tools and audit logs
  - Mini 7 (Deployment): Cloud deployment with CI pipeline

- Mid-course integrated project

  - Knowledge Base Q&A (RAG Lite) as described in Section 6.
  - Goals: full integration of backend + frontend + RAG + auth + deployment to staging; must show provenance and cost controls.

- Final full-stack AI capstone project

  - Requirements listed in Section 12.
  - Emphasis on production readiness: security, observability, SLOs, cost estimation, IaC, and automated tests.

Assessments & evaluation

- Continuous assessments

  - Weekly labs and quizzes measuring theory comprehension and coding skills.

- Project-based assessments
  - Mini-project code review and automated test suite (pass/fail + score).
  - Mid-course integrated project: demo to instructor panel with architecture critique.
  - Final capstone: graded by rubric and peer-review, plus a live demo and recorded walk-through.

- Non-functional evaluations
  - Security checklist: pass all OWASP/API security checklist items relevant to project.
  - Observability: traces and metrics visible in instructor-specified dashboard; token usage metrics implemented.
  - Cost report: produce cost estimation for one month in production and list mitigation strategies.

- Passing criteria
  - Minimum passing: complete mid-course project with working deployment and all unit tests passing; final capstone scoring $\geq$ 70% on rubric.

Portfolio and industry-readiness guidance

- What to include per project
  - README: short pitch, architecture diagram, tech stack, how to run locally, how to deploy, known limitations, and cost/security notes.
  - Architecture doc: data flows, scaling plan, SLOs, and backup/restore strategies.
  - Code quality: modular code, type-checking, tests, CI badges, and linting config.
  - Deployable demo: hosted staging URL or instructions to run in cloud (template IaC or docker-compose).
  - Observability snapshot: screenshots of dashboards (latency, error rate, model token usage) and a postmortem or runbook.
  - Short demo video: 3–5 minute screencast explaining the problem, architecture, and live demo.

- Resume & interview prep suggestions
  - Highlight measurable results: e.g., “Implemented RAG pipeline reducing average support response time by X%” or “Cut model token cost by Y% via caching & reranking”.
  - Prepare architecture whiteboards: be ready to explain trade-offs (model selection, deployment choice, security vs UX).
  - Be ready to discuss incident postmortems and performance tuning examples.

- Project presentation checklist
  - Problem statement and intended users.
  - Data flows and third-party services used.
  - Security & privacy considerations implemented.
  - Observability & SLOs.
  - Cost controls and mitigation strategies.

    - Roadmap & future work.

Responsible AI, governance & ethics integration (required for projects)

- Requirements for projects
    - PII handling: identify PII flows, redact or pseudonymize before sending to third-party LLMs.
    - Logging policy: prompt & outputs logged with redaction rules; retention policy documented.
    - Human-in-the-loop: add manual review path for high-risk outputs or action triggers.
    - Bias & fairness: short test plan for bias detection relevant to domain (e.g., classification fairness tests) and mitigation notes.
    - Legal & compliance checklist: GDPR/CCPA considerations where applicable and use of managed IdP for identity.

Teaching aids & materials to accompany the course

- Starter repos and templates (instructor-provided)
    - FastAPI template: project layout, DI patterns, Alembic, tests, Dockerfile, GitHub Actions pipeline.
    - Next.js TypeScript template: App Router structure, OpenAPI client generation pipeline, React Query/SWR configuration, E2E test scaffold.
    - RAG ingestion template: document chunker, embedding module, vector DB wrapper, reranker sample (cross-encoder stub).
    - Agent sandbox template: function schema definitions, validator, and safe executor.
- Lecture notes & slide outlines
    - For each lesson: key concepts, diagrams, code snippets, pitfalls, and suggested reading from the reference corpus.
- Lab instructions & test harnesses
    - Automated tests for labs (pytest, Playwright), mocked LLM fixtures, and cost-safety flags in CI.
- CI templates
    - GitHub Actions workflows for lint/test/build/deploy; demo deployments to free-tier clouds or instruction for Vercel/Render.
- Assessment rubrics & sample solutions
    - Clear grading rubric for code quality, architecture, security, testing, and deployment.

Common pitfalls, trade-offs & advice (practical tips)

- Avoid blocking calls in async endpoints; choose async DB drivers or run blocking calls in workers.
- Use short-lived model tokens and server-side proxies to avoid client-side secret leakage.
- Always include provenance for outputs from RAG or agents.

- Monitor token usage carefully and cache/reuse responses where safe.
- Start with managed services when team bandwidth is low; evaluate vendor lock-in trade-offs explicitly.
- Implement zero-downtime migration strategies: blue/green, shadow writes, and feature toggles for schema changes.
- For tests, mock LLM APIs and capture deterministic fixtures; use snapshot tests with tolerance thresholds for non-determinism.

Optional advanced extensions (for longer/new cohorts)

- Production-grade vector DB sharding & replication labs
- Implementing privacy-preserving techniques: differential privacy and local/on-prem inference
- Model fine-tuning workflows vs prompt engineering (cost/benefit analysis)
- Service mesh (Istio) and chaos engineering lab

Suggested week-by-week syllabus (compact 16-week cadence)

- Week 1: Onboarding & Foundations + Mini 1
- Week 2–3: Python & Web Foundations + Mini 2
- Week 4–5: FastAPI & REST API module + Mini 3
- Week 6: Databases & Data Management + Mini 4
- Week 7: Auth & Security + Mini 5
- Week 8–9: Next.js Frontend + BFF + Mid-course project kickoff
- Week 10: AI Integration & Cloud LLM APIs + Model adapter lab
- Week 11–12: Advanced AI Patterns (RAG, Agents) + Mini 6
- Week 13: Testing, Observability & Performance + Mini 7
- Week 14: DevOps, Deployment & Scaling
- Week 15: Capstone sprint (design & implementation)
- Week 16: Capstone finalization, demos, and grading

Instructor & classroom guidance (how to teach effectively)

- Use contract-first development: require OpenAPI for each group project.
- Encourage TDD where practical: students write tests first for critical components.
- Provide scaffolded starter repos so students focus on architecture and features.
- Offer recorded demos for complex infra steps (IaC, k8s, vector DB provisioning).
- Use pair programming and code reviews for collaborative learning.
- Provide safety guardrails in cloud accounts: budget alerts, usage caps, and cost-monitoring guides.
- Evaluate both code and architecture — include live Q&A/demos for final assessment.

Next-step deliverables (options I can produce if you want)

- Week-by-week lecture plan with slides and reading lists.
- Starter repositories (FastAPI + Next.js + RAG) with Dockerfile, CI, and deployment manifests.
- Detailed lab and grading rubrics for each mini project and capstone.
- Slide outlines and instructor notes for each module.

Appendix — Quick reference of recommended technologies & resources

- Backend: Python 3.10+, FastAPI, Pydantic, SQLAlchemy (async), Alembic
- Frontend: Next.js (App Router), React, TypeScript, React Query/SWR, Playwright
- Databases: Postgres, Redis, Pinecone/Weaviate/Milvus/FAISS

- AI: OpenAI/Anthropic/Vertex, LangChain, LlamaIndex (optional)
- DevOps: Docker, GitHub Actions, Terraform (or CDK), Kubernetes (optional)
- Observability: OpenTelemetry, Prometheus, Grafana, Jaeger
- Security: OAuth2/OIDC providers (Auth0/Okta/Cognito), Secrets Manager/HashiCorp
- Testing: pytest, pytest-asyncio, Playwright/Cypress, Locust/k6

Concluding notes

- This course converts the provided research corpus into an actionable, hands-on, production-oriented learning path. It balances fundamentals (Python, REST, DBs), frontend UX (Next.js), AI integration (RAG, agents), and production engineering (CI/CD, IaC, observability). The curriculum produces portfolio-ready projects and prepares learners for industry roles as Full-Stack AI engineers or SRE/ML-Infra engineers with strong system design judgment and applied AI knowledge.

If you want, I can next:

- Generate a detailed week-by-week syllabus with lecture slides outlines and lab instructions, or
- Produce the starter repository templates (FastAPI + Next.js + RAG) with Dockerfile, CI, and deployment manifests.

Which deliverable would you like me to create first?