Track 2: Python + React — Detailed Curriculum (29 Hours Core)

> *Total with AI + Agile + Project: 58 hours*
>
> *Backend: Python 3.13+ with FastAPI*
>
> *Frontend: React 19 with TypeScript*
>
> *Database: PostgreSQL + MongoDB*
>
> *Target: Final-year BE/BTech/BCA/MCA students*

Module 1: Development Setup & Git (2 Hours)

Session 1 (2 hrs) — Dev Environment & Version Control

Topics:

- Installing Python 3.13+, VS Code with Python extensions
- Virtual environments — `venv`, `pip`, `requirements.txt`
- Node.js, npm/yarn, Create React App / Vite setup
- PostgreSQL installation & pgAdmin
- Git fundamentals refresher — init, add, commit, branch, merge
- GitHub account setup, SSH keys
- Pull request workflow — branch → commit → push → PR → review → merge
- `.gitignore` for Python/React projects
- GitHub Copilot installation and activation

Hands-on:

- Create a virtual environment, install FastAPI and Uvicorn
- Create a GitHub repo, make a branch, submit a PR, merge it
- First Copilot interaction — let it autocomplete a Python function

---

Module 2: Python & FastAPI Backend (11 Hours)

Session 2 (2 hrs) — Python Refresher & FastAPI Introduction

Topics:

- Python features students may not know well — type hints, dataclasses, f-strings, walrus operator
- List/dict comprehensions, generators, context managers
- Decorators and how they work
- FastAPI introduction — why FastAPI (async, auto-docs, type-safe)

- Project structure — routers, services, models, schemas
- First endpoint — `@app.get("/")`, path parameters, query parameters
- Automatic Swagger UI (`/docs`) and ReDoc (`/redoc`)
- Running with Uvicorn — `uvicorn main:app --reload`

Hands-on:

- Create a FastAPI project with a "Hello World" endpoint
- Use type hints and comprehensions in a utility function
- Explore auto-generated Swagger docs

Session 3 (2 hrs) — REST API Fundamentals & Basic Security

Topics:

- REST principles — resources, HTTP methods (GET, POST, PUT, DELETE), status codes
- Pydantic models — request/response schemas, validation, serialization
- `BaseModel`, `Field()`, `@validator`, nested models
- Path parameters, query parameters, request body
- Response models — `response_model`, status codes, `HTTPException`
- Error handling — custom exception handlers, structured error responses
- Basic API security — why APIs need protection, authentication vs authorization
- JWT overview — what tokens are, how they secure APIs (detailed setup in Module 5)
- API testing with Postman and built-in `/docs`

Hands-on:

- Build a complete CRUD API for a `Student` resource
- Pydantic schemas — `StudentCreate`, `StudentResponse`, `StudentUpdate`
- Add validation — name required, email format, age range
- Custom error responses for 404, 422 scenarios
- Test all endpoints in Swagger UI and Postman

Session 4 (2 hrs) — Database Integration with SQLAlchemy

Topics:

- SQLAlchemy 2.0 setup — `sqlalchemy`, `asyncpg` / `psycopg2`
- Database connection — engine, session, dependency injection in FastAPI
- ORM models — `DeclarativeBase`, `Mapped`, `mapped_column`
- Relationships — `relationship()`, `ForeignKey`, one-to-many, many-to-one
- Alembic migrations — init, autogenerate, upgrade, downgrade
- CRUD operations with SQLAlchemy — async queries
- Repository pattern — separating DB logic from business logic

Hands-on:

- Connect Student API to PostgreSQL via SQLAlchemy
- Add a `Course` model with relationship to Student
- Create and run Alembic migration
- Implement repository pattern for Student CRUD
- Verify data in pgAdmin

Session 5 (2 hrs) — Service Layer Design & Dependency Injection

Topics:

- Service layer design — why separate from routers
- Dependency injection in FastAPI — `Depends()`, factory functions, nested dependencies
- Repository pattern — abstracting database access, interface-like ABCs
- Project structure best practices — feature-based organization
- Type hints for services — `Protocol` classes, abstract base classes
- Error handling in services — custom exceptions, mapping to HTTP responses

Hands-on:

- Refactor Student API to proper service/repository pattern
- Implement dependency injection for services with `Depends()`
- Create repository ABC and concrete implementation
- Reorganize project into feature-based structure

Session 6 (2 hrs) — Pagination, Filtering & Data Mapping

Topics:

- Pagination — offset/limit, `Page` response model, cursor-based pagination intro
- Sorting — dynamic `order_by` with query parameters, multi-column sort
- Filtering — query parameters, dynamic filters, search across fields
- Pydantic mapping — model-to-schema conversion, `model_validate`, `from_attributes`
- Response envelope — standardized API response format (`ApiResponse[T]`)
- Query builder pattern — composing complex queries from filter parameters

Hands-on:

- Add pagination and sorting to list endpoints
- Add dynamic filtering (by name, course, date range)
- Implement standardized response envelope
- Build a search endpoint that combines multiple filters
- Create reusable query builder for common filter patterns

Session 7 (2 hrs) — Testing & Code Quality

Topics:

- Why testing matters — confidence, refactoring safety, CI/CD readiness
- Testing with `pytest` — fixtures, `parametrize`, async tests
- `httpx.AsyncClient` for testing FastAPI endpoints
- Mocking with `unittest.mock` — patching services and repos
- Test organization — unit vs integration tests, naming conventions
- Code coverage basics — what to measure, what coverage is meaningful

Hands-on:

- Write 5 unit tests with pytest — service tests with mocked repo
- Write 2 integration tests with `httpx.AsyncClient`
- Test CRUD operations end-to-end

- Practice test-first: write a test, then implement the feature

Session 8 (1 hr) — API Documentation & Best Practices

Topics:

- FastAPI auto-docs — customizing Swagger UI metadata, tags, descriptions
- API versioning — URL prefix strategy (`/api/v1/`)
- CORS configuration — `CORSMiddleware` for frontend integration
- Environment variables — `python-dotenv`, `pydantic-settings`
- Logging — `logging` module, structured logs
- Project structure best practices — feature-based organization

Hands-on:

- Customize Swagger UI with proper descriptions and tags
- Configure CORS for `http://localhost:5173` (Vite dev server)
- Move config to environment variables with `pydantic-settings`

Module 3: Database Deep Dive (3 Hours)

Session 9 (2 hrs) — PostgreSQL Mastery

Topics:

- Schema design principles — normalization (1NF, 2NF, 3NF), when to denormalize
- Data types — VARCHAR, INTEGER, BOOLEAN, TIMESTAMP, JSONB, UUID
- Constraints — PRIMARY KEY, FOREIGN KEY, UNIQUE, CHECK, NOT NULL
- Indexes — B-tree, when to create, covering indexes, `EXPLAIN ANALYZE`
- Complex queries — JOINs (INNER, LEFT, RIGHT), subqueries, CTEs (`WITH`)
- Aggregate functions — GROUP BY, HAVING, window functions intro
- Transactions — ACID properties, isolation levels

Hands-on:

- Design a schema for the capstone project (3-4 tables with relationships)
- Write 5 complex queries with JOINs and aggregations
- Create indexes and compare `EXPLAIN ANALYZE` before/after
- Practice with CTEs for readable complex queries

Session 10 (1 hr) — Advanced ORM Patterns

Topics:

- N+1 query problem — what it is, how to detect (`echo=True`), how to fix
- Eager loading — `joinedload()`, `selectinload()`
- Projections — selecting specific columns for performance
- Raw SQL when ORM is overkill — `text()` queries
- Alembic advanced — data migrations, custom scripts

Hands-on:

- Identify and fix N+1 query in Student-Course relationship
- Implement eager loading with `selectinload`
- Compare ORM vs raw SQL for a complex report query

Module 4: React Frontend (9 Hours)

Session 11 (2 hrs) — React Fundamentals

Topics:

- React architecture — components, JSX, virtual DOM
- Project setup — Vite + React + TypeScript (`npm create vite@latest`)
- TypeScript essentials for React — types, interfaces, generics, union types
- Functional components — props, children, default props
- JSX deep dive — expressions, conditional rendering, lists with `key`
- Event handling — `onClick`, `onChange`, `onSubmit`, event types in TypeScript
- Styling — CSS modules, Tailwind CSS setup

Hands-on:

- Create React project with Vite + TypeScript
- Build 3 components — Header, StudentList, StudentCard
- Render a list of students with conditional styling
- Set up Tailwind CSS

Session 12 (2 hrs) — Routing & State with Hooks

Topics:

- React Router v6 — `BrowserRouter`, `Routes`, `Route`, `Link`, `useNavigate`
- Route parameters — `useParams`, dynamic routes
- Nested routes and layouts — `<Outlet>`
- Protected routes — auth wrapper component
- React Hooks — `useState`, `useEffect`, `useContext`, `useRef`, `useMemo`
- Custom hooks — `useAuth`, `useLocalStorage`, `useFetch`
- Component lifecycle with hooks

Hands-on:

- Set up routing: Dashboard, Students, Student Detail, Login
- Create a protected route wrapper
- Build `useAuth` custom hook with context
- Implement `useLocalStorage` for persisting token

Session 13 (2 hrs) — Forms & State Management

Topics:

- Controlled components — form inputs with `useState`
- Form libraries — React Hook Form (register, handleSubmit, errors)
- Validation with Zod — schema definition, integration with React Hook Form
- Complex forms — nested objects, arrays, dynamic fields
- Context API — `createContext`, `useContext`, Provider pattern
- When to use Context vs prop drilling vs state management
- `useReducer` for complex state logic

Hands-on:

- Build a student registration form with React Hook Form + Zod validation
- Display inline validation errors with proper UX
- Create AuthContext for managing login state across components
- Use `useReducer` for a multi-step form

Session 14 (2 hrs) — HTTP & API Integration

Topics:

- `fetch` API and `axios` — GET, POST, PUT, DELETE
- Async/await patterns in React — loading, error, data states
- Custom `useFetch` / `useApi` hook
- Axios interceptors — attaching JWT token, handling 401 refresh
- Error boundaries — `ErrorBoundary` component, fallback UI
- Environment variables — `.env`, `VITE_API_URL`
- TanStack Query (React Query) intro — `useQuery`, `useMutation`, caching

Hands-on:

- Connect React to FastAPI backend
- Implement JWT auth flow — login, store token, attach to requests via interceptor
- Build Student CRUD pages — list (with search/filter), create form, edit, delete
- Add loading spinners and error toast (react-hot-toast)

Session 15 (1 hr) — UI Polish & Responsive Design

Topics:

- Tailwind CSS patterns — responsive prefixes, dark mode, animations
- Component library options — shadcn/ui, Headless UI
- Table component — sorting, filtering, pagination with TanStack Table
- Charts — Recharts or Chart.js for dashboard widgets
- Build optimization — `npm run build`, code splitting, lazy loading with `React.lazy`
- Deployment basics — building for production, static hosting

Hands-on:

- Polish the student management app with Tailwind
- Add a dashboard with 2 chart widgets (students per course, enrollment trends)

- Verify responsive design on mobile viewport
- Run production build

Module 5: Authentication & Authorization (2 Hours)

Session 16 (2 hrs) — Full Authentication & Authorization

> *Taught after frontend so students can implement the complete auth flow end-to-end (backend + frontend).*

Topics:

- Why authentication matters — stateless APIs, tokens vs sessions
- JWT (JSON Web Tokens) — structure (header, payload, signature), how they work
- `python-jose` for JWT handling — encoding, decoding, expiration
- Password hashing with `passlib` and `bcrypt`
- FastAPI security — `OAuth2PasswordBearer`, `Depends()`
- Dependency injection for current user — `get_current_user()`
- Role-based access — checking user roles in dependencies
- Protecting endpoints — public vs authenticated vs admin-only
- Frontend auth flow — login page, token storage, route guards, interceptors

Hands-on:

- Add user registration and login endpoints to FastAPI backend
- Implement JWT token generation and validation
- Create `get_current_user` dependency
- Protect Student CRUD — only authenticated users can create/update/delete
- Add ADMIN role — only admins can delete students
- Connect React login page to backend auth API
- Test full flow: register → login → get token → access protected routes

---

Module 6: AI Tools for Developers (15 Hours)

> *See AI-TOOLS-MODULE.md for detailed curriculum.*

> *This module is shared across all three tracks.*

Module 7: Agile & Scrum Workshop (2 Hours)

> *See CAPSTONE-PROJECT.md for the Agile workshop session.*
>
> *Teaches Scrum framework, user stories, estimation, GitHub Projects setup.*
>
> *Shared across all three tracks.*

Module 8: Capstone Project Sprint (12 Hours)

> *See CAPSTONE-PROJECT.md for project guidelines and sprint structure.*
>
> *Track 2 stack: Python (FastAPI) + React + PostgreSQL + AI features*
>
> *Run in 2 real Agile sprints with standups, feature branches, PR reviews, and retrospective.*

Track 2 Summary

Module	Hours	Key Technologies
Dev Setup & Git	2	Python 3.13, VS Code, venv, Git, GitHub, Copilot
Python & FastAPI	13	FastAPI, Pydantic, SQLAlchemy 2.0, pytest
Database	3	PostgreSQL, Alembic
React	9	React 19, TypeScript, React Router, React Hook Form, Zod
Authentication	2	JWT, python-jose, passlib, role-based access
AI Tools	15	Copilot, ChatGPT, Prompt Eng, OpenAI API, RAG, MCP
Agile & Scrum	2	Scrum, user stories, estimation, GitHub Projects
Capstone Project	12	Full-stack app in Agile sprints with AI features
Total	58	