predev. Academy

TRACK ROADMAP

.NET Backend Development

Build, test, and deploy a production REST API with C# 14, ASP.NET Core, EF Core, and PostgreSQL on your own Linux server.

Web Backend

DURATION	SESSIONS	SCHEDULE	LEVEL	LANGUAGE	FORMAT
10 weeks	20 live sessions, 60 hours	Twice a week, 3 hours per session, plus 1 hour of online office hours every week	Beginner, after Tech Compass	Egyptian Arabic, English technical terms	In person or online, live

Who it is for

- CS and IT students who want a backend job in .NET after graduation
- Fresh graduates who know some programming but have never shipped a real API
- Career switchers who finished Tech Compass and chose backend development
- Junior developers in other stacks who want to move to C# and ASP.NET Core, the stack many Egyptian banks, enterprises, and Gulf clients hire for

Before day 1

- Tech Compass completed, or basic programming: variables, conditions, loops, functions
- Laptop with at least 8 GB RAM (16 GB recommended), SSD, Windows 11, macOS, or Linux
- Installed before day 1: .NET 10 SDK, Visual Studio Code with the C# Dev Kit extension (or JetBrains Rider), Git, Docker Desktop, Postman or Bruno
- Accounts: GitHub (with a profile photo and real name), a Claude or Cursor account for week 8
- Comfortable reading simple English technical documentation

By the end you can

- Explain what happens in an HTTP request and pick the right method, status code, and JSON shape for an endpoint
- Write clean C# 14 using classes, interfaces, generics, LINQ, records, async/await, and nullable reference types
- Design a relational schema and write SQL queries with joins, grouping, and indexes on PostgreSQL and SQL Server
- Build a REST API with ASP.NET Core using controllers or Minimal APIs, validation, error handling, and OpenAPI docs
- Model data with EF Core 10, run migrations, write efficient queries, and use transactions
- Secure an API with ASP.NET Core Identity, JWT access and refresh tokens, and role and policy authorization
- Add background jobs, Redis caching, and structured logging with Serilog
- Write unit tests with xUnit and integration tests with WebApplicationFactory
- Containerize an API with Docker and deploy it to a Linux VPS through Coolify with a domain and HTTPS
- Use AI coding tools to move faster while reviewing every change and keeping tests green

Tech stack

LANGUAGE & RUNTIME

.NET 10 LTS / C# 14	Long term support release used for new enterprise projects, with modern C# features like records and nullable reference types
.NET CLI	Create, build, run, and test projects from the terminal, the same way CI and servers do

WEB FRAMEWORK

ASP.NET Core 10 Web API	Controllers and Minimal APIs, middleware, dependency injection, and configuration built in
FluentValidation	Readable validation rules kept out of controllers
OpenAPI with Scalar	Built-in API documentation that frontend and mobile teams can test against

DATA

Entity Framework Core 10	The standard ORM in .NET: models, migrations, relations, and LINQ queries
PostgreSQL 17	Free, fast, and the default database for most new APIs and VPS deployments
SQL Server 2022	Still the most requested database in Egyptian enterprise and banking job posts
Redis 8	Caching hot reads and sharing state between instances

SECURITY

ASP.NET Core Identity	Users, password hashing, roles, and lockout without writing it from scratch
JWT bearer tokens	Stateless auth for mobile apps and single page frontends

OPERATIONS

Serilog	Structured logs you can search when something breaks in production
Hosted services and Hangfire	Run emails, reminders, and cleanup work outside the request
Docker	Package the API and its dependencies the same way on every machine
Coolify on a Linux VPS	Deploy containers with HTTPS, environment variables, and databases on a server you control

QUALITY & TOOLING

xUnit and WebApplicationFactory	Unit tests and integration tests that call the real API in memory
Git and GitHub	Version control, pull requests, and the portfolio employers look at
Claude Code or Cursor	AI coding tools used with review and tests, the way teams use them today

The running project: Mawid: appointment booking API

Mawid is the backend for a booking app where clinics, salons, and training centers publish services and staff schedules, and customers book, reschedule, and cancel appointments. It mirrors the booking and operations systems predev. Solutions builds for clients, and it covers the problems every real backend faces: users and roles, related data, double booking, reminders, caching, and deployment.

- Registration, login, JWT access tokens and refresh tokens with ASP.NET Core Identity

- Roles: customer, business owner, staff, admin, with policy rules such as owners only editing their own business
- Businesses, branches, services, and staff with working hours
- Availability endpoint that calculates free time slots per staff member and service
- Bookings with create, reschedule, cancel, and status changes, protected against double booking with a transaction and a unique constraint
- Search, filtering, sorting, and pagination on businesses and bookings
- Validation with FluentValidation and consistent ProblemDetails error responses
- Reminder emails and daily cleanup as background jobs
- Redis caching for business listings and availability, with cache invalidation on change
- Structured request and error logging with Serilog
- OpenAPI document with a Scalar UI
- Unit and integration test suite run with dotnet test
- Docker image and docker compose setup with PostgreSQL and Redis
- Live deployment on a Linux VPS through Coolify with a domain and HTTPS

Phase overview

WEEKS 1-3

Foundations

Understand how the web works and write solid C# 14 and SQL before touching a framework.

WEEKS 4-7

Core framework

Build the Mawid booking API step by step with ASP.NET Core, EF Core, Identity, caching, jobs, logging, tests, and Docker.

WEEKS 8

Building with AI

Use AI coding tools to speed up real work on Mawid while staying in control of quality and security.

WEEKS 9

Capstone

Deploy Mawid to a real server, then build and deploy your own API from a brief.

WEEKS 10

Career

Present your work, prepare for .NET interviews, and start applying for jobs and freelance work.

Week by week

Foundations · Weeks 1-3

Week 1 How the web works and first steps in C#

Session 1 HTTP, JSON, and the client server model

- Clients, servers, DNS, IP addresses, and ports
- HTTP requests and responses: methods, URLs, headers, body
- Status codes by family: 2xx, 3xx, 4xx, 5xx and when to use each
- JSON syntax and how APIs use it
- REST basics: resources, nouns in URLs, statelessness
- Testing real public APIs with Postman and curl

In-class lab: Call three public APIs with Postman and curl, read the headers and status codes, then write a one page API contract for a simple to do list: endpoints, methods, request bodies, and expected status codes.

Session 2 .NET 10 setup, the CLI, and C# basics

- What .NET is: SDK, runtime, NuGet, LTS releases
- dotnet new, build, run, and the structure of a console project
- Variables, built-in types, string interpolation, var
- Conditions, loops, switch expressions
- Methods, parameters, return values
- Git basics: init, add, commit, push to GitHub

In-class lab: Build a console expense tracker that reads amounts and categories from the user, prints totals per category, and push it to a new GitHub repository with clear commits.

Homework: Solve 6 beginner problems on loops and conditions in C# (provided list), extend the expense tracker with a monthly budget warning, and push everything to GitHub with at least 5 meaningful commits.

Week 2 Object oriented C#, collections, and LINQ

Session 1 Classes, interfaces, and generics

- Classes, objects, constructors, properties
- Encapsulation and access modifiers
- Inheritance and when composition is the better choice
- Interfaces and programming against abstractions
- Generics: generic classes and methods, constraints
- Reference types and value types, structs

In-class lab: Model a small library system with Book, Member, and Loan classes and an ILoanPolicy interface with two implementations, then write a generic Repository<T> that stores items in memory.

Session 2 Collections, LINQ, and Git branching

- List, Dictionary, HashSet and when to use each
- LINQ: Where, Select, OrderBy, GroupBy, Any, First, Sum
- Deferred execution and why it matters later with EF Core
- Problem solving method: understand, example, plan, code, test
- Git branches, merging, and pull requests on GitHub
- Writing a useful README

In-class lab: Load a JSON file of 500 orders and answer 8 business questions with LINQ (top customers, revenue per month, most sold product). Work on a feature branch and open a pull request for review.

Homework: Finish the LINQ questions, add 4 more of your own, and solve 5 array and string problems on LeetCode or HackerRank. Review one classmate's pull request and leave at least 3 comments.

Week 3 Modern C# and relational databases

Session 1 async/await, records, nullability, and exceptions

- Tasks, async/await, and why servers need async I/O
- HttpClient basics and calling an API from C#
- Records, init properties, and immutability
- Nullable reference types and reading compiler warnings
- Exceptions: try/catch/finally, custom exceptions, when not to throw
- Pattern matching on types and properties

In-class lab: Write an async console app that fetches data from a public API with HttpClient, maps it to records, handles network errors and null fields, and prints a summary report.

Session 2 SQL and relational design

- Tables, primary keys, foreign keys, constraints
- One to many and many to many relations, normalization basics
- SELECT, WHERE, JOIN, GROUP BY, HAVING, ORDER BY
- INSERT, UPDATE, DELETE, and transactions
- Indexes and reading a simple query plan
- PostgreSQL and SQL Server side by side in Docker

In-class lab: Design the first database schema for Mawid (users, businesses, services, staff, bookings) on paper, create it in PostgreSQL running in Docker, seed sample data, and write 10 queries such as bookings per business per day.

Homework: Run the same schema on SQL Server and note the differences. Write 10 more SQL queries from the exercise sheet, and add a short ERD image of your Mawid schema to the repository README.

Week 4 ASP.NET Core Web API fundamentals

Session 1 First API: routing, controllers, and Minimal APIs

- Creating a Web API project and reading Program.cs
- Controllers and Minimal APIs: same job, two styles
- Routing, route parameters, query strings, and model binding
- Returning results: Ok, Created, NotFound, NoContent
- The middleware pipeline and the order it runs in
- Built-in OpenAPI document and the Scalar UI

In-class lab: Create the Mawid solution and build CRUD endpoints for businesses and services with an in memory store, once with controllers and once with Minimal APIs, then test them in Scalar and Postman.

Session 2 Dependency injection, configuration, and validation

- Dependency injection: services, lifetimes (singleton, scoped, transient)
- appsettings.json, environments, user secrets, and environment variables
- The options pattern with strongly typed settings
- DTOs and keeping entities out of responses
- Validation with FluentValidation
- Global error handling and ProblemDetails responses

In-class lab: Move logic into services registered with DI, add request DTOs with FluentValidation rules, add a global exception handler that returns ProblemDetails, and load business settings through the options pattern.

Homework: Add staff and working hours endpoints with full validation and correct status codes. Write a Postman collection that covers success and failure cases for every endpoint.

Project milestone: Mawid API skeleton: business, service, and staff endpoints with validation, error handling, and OpenAPI docs.

Week 5 Entity Framework Core 10

Session 1 DbContext, entities, relations, and migrations

- DbContext and DbSet, connecting to PostgreSQL
- Entity configuration with conventions and the Fluent API
- One to many and many to many relations, owned types
- Migrations: add, update, rollback, and reviewing generated SQL
- Seeding data for development
- Switching the provider to SQL Server

In-class lab: Replace the in memory store with EF Core and PostgreSQL. Configure all Mawid entities and relations, create the first migration, review its SQL, and seed sample businesses and staff.

Session 2 Querying, tracking, and transactions

- LINQ to SQL: Where, Include, projection with Select
- Change tracking and AsNoTracking for read only queries
- The N+1 problem and how to spot it in the logs
- Pagination, filtering, and sorting
- Transactions and unique constraints to prevent double booking
- Concurrency tokens and handling conflicts

In-class lab: Build the booking endpoints and the availability endpoint. Prevent double booking with a transaction plus a unique index, then prove it by sending parallel requests from a small script.

Homework: Add search, filtering, and pagination to the business list, add a reschedule endpoint, and fix any N+1 queries you find using the EF Core logs.

Project milestone: Mawid persists to PostgreSQL: migrations, relations, bookings, availability, and double booking protection.

Week 6 Authentication, authorization, and architecture

Session 1 ASP.NET Core Identity and JWT

- Authentication and authorization: what each one answers
- ASP.NET Core Identity with EF Core: users, password hashing, lockout
- Issuing and validating JWT access tokens
- Refresh tokens: storage, rotation, and revocation
- Protecting endpoints and reading the current user
- Common mistakes: secrets in code, long lived tokens, missing HTTPS

In-class lab: Add register, login, refresh, and logout endpoints to Mawid with Identity and JWT, then make booking endpoints use the logged in user instead of a user id from the request.

Session 2 Roles, policies, and a practical layered structure

- Role based authorization: customer, owner, staff, admin
- Policy based authorization and resource checks (owners edit only their business)
- Splitting the solution into Api, Application, Domain, and Infrastructure projects
- Where business rules live and keeping controllers thin
- When extra layers and patterns add cost without benefit
- Code review checklist for API pull requests

In-class lab: Refactor Mawid into a layered solution, move booking rules into the application layer, and add role and policy rules so a business owner cannot touch another business's data. Verify with Postman as four different users.

Homework: Add an admin endpoint to suspend a business and a staff endpoint to mark bookings as completed or no show, each with the right policy. Write a short security note in the README listing what is protected and how.

Project milestone: Mawid is secured: Identity, JWT with refresh tokens, roles, policies, and a clean layered solution.

Week 7 Background jobs, caching, logging, testing, and Docker

Session 1 Background jobs, Redis caching, and Serilog

- Hosted services and BackgroundService for simple recurring work
- Hangfire for scheduled and retryable jobs with a dashboard
- Caching with IDistributedCache and Redis, expiry, and invalidation
- Output caching for public endpoints
- Structured logging with Serilog: sinks, enrichers, request logging
- Correlation ids to follow one request through the logs

In-class lab: Schedule booking reminder emails with Hangfire, add a nightly cleanup hosted service, cache business listings and availability in Redis with invalidation on change, and add Serilog request logging.

Session 2 Testing with xUnit and shipping with Docker

- Unit tests with xUnit: arrange, act, assert, theories
- Integration tests with WebApplicationFactory
- Real databases in tests with Testcontainers for PostgreSQL
- Testing authenticated endpoints
- Writing a Dockerfile for an ASP.NET Core API, multi stage builds
- docker compose with the API, PostgreSQL, and Redis

In-class lab: Write unit tests for the availability calculation and integration tests for register, login, create booking, and double booking. Then containerize Mawid and run the full stack with docker compose.

Homework: Reach at least 15 passing tests covering the main booking rules, add a health check endpoint, and document how to run the project locally with docker compose in the README.

Project milestone: Mawid is production ready: background jobs, Redis caching, structured logs, test suite, and a working docker compose setup.

Building with AI · Weeks 8

Week 8 Building with AI the right way

Session 1 AI coding tools in a real codebase

- How tools like Claude Code and Cursor read a codebase and make changes
- Giving context: project instructions file, conventions, and small focused tasks
- Planning a feature with AI before writing code
- Reading AI generated diffs line by line before accepting
- Common AI mistakes in .NET: wrong package versions, old APIs, invented methods
- Keeping secrets and client data out of prompts

In-class lab: Write a project instructions file for Mawid, then use Claude Code or Cursor to add a reviews feature (customers rate completed bookings). Review every diff, reject what is wrong, and record what you changed by hand.

Session 2 Tests as a safety net and when not to trust AI

- Writing tests first and asking AI to make them pass
- Using AI to write tests, then checking that the tests can fail
- Security review of AI code: authorization checks, SQL injection, mass assignment
- Debugging with AI: giving logs and stack traces, verifying the fix
- Tasks to keep for yourself: auth, money, data migrations, architecture decisions
- Being honest in interviews about how you use AI

In-class lab: Take three AI generated pull requests prepared by the instructor, each with a hidden bug or security hole. Find the problem, write a failing test that proves it, then fix it with or without AI.

Homework: Use AI to add one more Mawid feature of your choice (for example business opening exceptions or booking export to CSV) with tests. Submit a short review log: what the AI did well, what you rejected, and why.

Project milestone: Mawid gains an AI assisted feature delivered with tests and a written review log.

Capstone · Weeks 9

Week 9 Real deployment and capstone build

Session 1 Deploying to a Linux VPS with Coolify

- Linux VPS basics: SSH keys, users, firewall, updates
- Coolify: projects, applications from a GitHub repository, Dockerfile builds
- PostgreSQL and Redis as managed services in Coolify
- Environment variables, secrets, and running migrations on deploy
- Domains, DNS records, and automatic HTTPS
- Health checks, logs, and rolling back a bad deploy

In-class lab: Deploy Mawid to the class VPS through Coolify with PostgreSQL, Redis, a subdomain, and HTTPS. Run migrations, test the live API from Postman, then receive and break down your capstone brief into tasks.

Session 2 Capstone studio: build, review, deploy

- Turning a brief into endpoints, a schema, and a task list
- Scoping: what to cut so it ships on time
- Short code review rounds with the instructor and peers
- Writing a README with setup steps, API overview, and live URL
- Preparing a 5 minute demo

In-class lab: Studio session: build your capstone API with check ins every hour, get one code review from the instructor, and finish with a first deploy of the capstone on Coolify.

Homework: Finish the capstone: all must have requirements, tests passing, deployed with HTTPS, README complete. Submit the GitHub link and live URL before the end of week 10.

Project milestone: Mawid is live on a VPS, and the capstone API is deployed with HTTPS.

Week 10 Career week and demo day

Session 1 CV, LinkedIn, GitHub, and freelancing

- A one page backend developer CV that passes ATS filters
- LinkedIn profile: headline, about section, featured projects
- GitHub portfolio: pinned repositories, READMEs, live links, clean history
- Where .NET jobs are in Egypt and remote: LinkedIn, Wuzzuf, company career pages
- Freelancing first steps: Mostaql, Upwork, scoping and pricing a small API job
- Writing a short and specific job application message

In-class lab: Rewrite your CV and LinkedIn profile with live feedback, polish your GitHub profile with Mawid and the capstone pinned, and send 3 real applications in class.

Session 2 Mock interviews and demo day

- Common .NET interview questions: DI lifetimes, async, EF Core tracking, middleware
- Live coding: a small LINQ or problem solving task
- System design basics for juniors: explaining your API end to end
- Behavioral questions and talking about your projects
- Demo day: present your capstone to guests from predev. Solutions and partner companies

In-class lab: Do a 30 minute mock technical interview in pairs with instructor feedback, then present your capstone live on demo day: problem, demo, architecture, and one thing you would change.

Homework: Apply to at least 10 relevant jobs or freelance projects in the next two weeks, keep a simple tracker, and share progress in office hours.

Capstone

Each student gets a brief for a small real product and builds its REST API alone in week 9: data model, endpoints, auth, validation, tests, and a live deployment on a Linux VPS through Coolify. The brief describes users and needs, and the student decides the endpoints and schema.

Requirements

- ASP.NET Core 10 Web API on .NET 10 with EF Core 10 and PostgreSQL
- At least 5 related entities with migrations
- Registration and login with ASP.NET Core Identity and JWT, plus at least two roles with policy checks
- Validation with FluentValidation and ProblemDetails error responses
- Pagination and filtering on at least one list endpoint
- At least one background job or Redis cache
- Structured logging with Serilog
- At least 10 tests including integration tests with WebApplicationFactory
- OpenAPI document with a UI
- Dockerfile and live deployment on a VPS through Coolify with HTTPS
- README with setup steps, API overview, and live URL

Example ideas

- Pharmacy stock and order API with low stock alerts for a small pharmacy chain
- Gym membership API with subscriptions, class bookings, and attendance check in
- Home services marketplace API (plumbers, electricians) with requests, quotes, and ratings
- Training center API for courses, groups, attendance, and installment payments
- Restaurant ordering API with menus, branches, orders, and delivery status
- Property rental API for listings, viewing requests, and landlord dashboards

Capstone rubric

Works end to end and is deployed with HTTPS	**25%**
API design: endpoints, status codes, validation, errors	**15%**
Data model and EF Core usage	**15%**
Authentication, authorization, and security basics	**15%**
Tests	**15%**
Code quality, structure, and README	**10%**
Demo and explanation	**5%**

How you are assessed

Weekly homework Submitted on GitHub before the next week's first session, graded on correctness, commits, and code clarity	**20%**
Mawid project milestones Checked in weeks 4-8 against the milestone list, with a short code review each week	**30%**
Capstone project Graded with the capstone rubric	**35%**
Mock interview and demo day Technical answers, live coding, and how clearly you explain your work	**10%**
Attendance and participation At least 18 of 20 sessions, pull request reviews for classmates, office hours activity	**5%**

Skills checklist

Web and HTTP

- [ ] Read and write HTTP requests and responses
- [ ] Choose correct methods and status codes
- [ ] Design REST endpoints and JSON payloads

C#

- [ ] Classes, interfaces, generics, records
- [ ] Collections and LINQ
- [ ] async/await and HttpClient
- [ ] Nullable reference types and exception handling

Databases

- [ ] Design a normalized relational schema
- [ ] Write SQL with joins, grouping, and indexes
- [ ] Work with PostgreSQL and SQL Server

ASP.NET Core

- [ ] Controllers and Minimal APIs, routing, model binding
- [ ] Middleware pipeline and global error handling
- [ ] Dependency injection and the options pattern
- [ ] Validation with FluentValidation
- [ ] OpenAPI documentation

EF Core

- [ ] Entities, relations, and migrations
- [ ] Efficient queries, projection, no tracking reads
- [ ] Transactions and concurrency handling

Security

- [ ] ASP.NET Core Identity
- [ ] JWT access and refresh tokens
- [ ] Role and policy authorization
- [ ] Secrets management with environment variables

Production

- [ ] Background jobs with hosted services and Hangfire
- [ ] Redis caching and invalidation
- [ ] Structured logging with Serilog
- [ ] Docker and docker compose
- [ ] Deploy on a Linux VPS with Coolify

Quality and workflow

- [ ] Unit tests with xUnit and integration tests with WebApplicationFactory
- [ ] Git branches, pull requests, and code review
- [ ] Reviewing and testing AI generated code

After graduation

Months 1-3 after graduation

- Apply for junior .NET backend roles and internships every week
- Add one more deployed API to your portfolio
- Practice problem solving 3 times a week on LeetCode
- Read the ASP.NET Core and EF Core docs sections you used in class in full

Months 4-6

- Learn CI/CD with GitHub Actions: build, test, and deploy on push
- Learn messaging with RabbitMQ or Azure Service Bus
- Learn real time features with SignalR
- Contribute small fixes to an open source .NET project

Months 7-12: toward mid-level

- Study performance: profiling, query tuning, and load testing
- Learn a cloud platform (Azure or AWS) and prepare for a fundamentals certificate
- Learn observability with OpenTelemetry
- Read about domain driven design and modular monoliths, and apply only what your project needs
- Mentor a newer student and lead code reviews

Where this track leads

Junior .NET Backend Developer | ASP.NET Core Developer | C# Software Engineer | API Developer | Junior Full Stack Developer (.NET with a frontend team) | Freelance API developer on Mostaql and Upwork | Remote .NET developer for Gulf and European companies

Resources

- C# documentation: https://learn.microsoft.com/en-us/dotnet/csharp/
- .NET documentation: https://learn.microsoft.com/en-us/dotnet/
- ASP.NET Core documentation: https://learn.microsoft.com/en-us/aspnet/core/
- Entity Framework Core documentation: https://learn.microsoft.com/en-us/ef/core/
- ASP.NET Core Identity: https://learn.microsoft.com/en-us/aspnet/core/security/authentication/identity
- MDN HTTP guide: https://developer.mozilla.org/en-US/docs/Web/HTTP
- PostgreSQL documentation: https://www.postgresql.org/docs/
- SQL Server documentation: https://learn.microsoft.com/en-us/sql/sql-server/
- FluentValidation documentation: https://docs.fluentvalidation.net/
- Redis documentation: https://redis.io/docs/
- Serilog: https://serilog.net/
- Hangfire documentation: https://docs.hangfire.io/
- xUnit.net: https://xunit.net/
- Testcontainers for .NET: https://dotnet.testcontainers.org/
- Docker documentation: https://docs.docker.com/
- Coolify documentation: https://coolify.io/docs/
- Git documentation: https://git-scm.com/doc
- GitHub Docs: https://docs.github.com/
- Claude Code documentation: https://docs.claude.com/en/docs/claude-code/overview
- Cursor documentation: https://docs.cursor.com/