SOFTWARE DEVELOPMENT AND ENGINEERING PRACTICES

Building REST APIs with ASP.NET Core Web API

Level: Practitioner • 2 days (expandable to 3) • Virtual, In-person

Overview

APIs are the contracts your software makes with the outside world, and unlike code, a contract is hard to change once someone depends on it. Getting an ASP.NET Core Web API to return JSON takes an afternoon; designing resources and status codes that clients can rely on, handling errors consistently, securing the endpoints, and documenting it all so consumers do not have to guess is where real API work lives.

This is a hands-on, practitioner course. It begins with REST design fundamentals, because the framework can only implement decisions you have already made well, then layers up through controllers and routing, models and validation, error handling, security, and OpenAPI documentation. Rather than tour every feature of ASP.NET Core, it goes deep on the path from a blank project to a well-designed, secured, documented API, and you build that API throughout the course. Every module ends with a lab and builds on the one before.

Who Should Attend

- C# developers who need to build or maintain HTTP APIs
- Back-end developers moving from older ASP.NET or WCF to ASP.NET Core
- Full-stack developers who want their APIs to be as solid as their front ends

Developers new to C# should take *Object-Oriented Programming with C#* first.

Prerequisites

- Working proficiency in C#
- Basic understanding of HTTP: requests, responses, and status codes
- Comfortable with Visual Studio or VS Code and running a .NET project locally

What You Will Learn

- Design RESTful resources, routes, and status codes that clients can depend on
- Build API endpoints with ASP.NET Core controllers, routing, and model binding
- Validate input and shape responses with DTOs instead of leaking internal models
- Handle errors consistently with problem details and centralized exception handling
- Secure an API with authentication and authorization, including JWT bearer tokens
- Document an API with OpenAPI and manage change through versioning

Course Outline

Day one: from REST design to working endpoints

- REST and API Design Fundamentals
  - Resources, verbs, and status codes: the grammar of HTTP APIs
  - Designing URLs and payloads clients can predict

  - What makes an API easy to consume, and the design smells that make it painful
  - Lab: design the resource model and endpoint contract for the API you will build all course
- Controllers, Routing, and Model Binding
  - Project anatomy: the request pipeline, controllers, and dependency injection
  - Attribute routing, route parameters, and query strings
  - Model binding from route, query, and body
  - Lab: implement the core CRUD endpoints for your designed resources
- Models, DTOs, and Validation
  - Why DTOs: separating your API contract from your internal model
  - Validating input with data annotations and returning useful validation errors
  - Mapping between entities and DTOs without drowning in boilerplate
  - Lab: add DTOs and validation to your endpoints and prove bad input is rejected cleanly

Day two: making it dependable

- Persistence and the Service Layer
  - Wiring in a database with Entity Framework Core, kept at arm's length from controllers
  - Async endpoints and why they matter for API throughput
  - Lab: replace in-memory data with EF Core persistence behind a service layer
- Error Handling and Security
  - Consistent errors with ProblemDetails and centralized exception handling
  - Authentication with JWT bearer tokens, and authorization with policies and roles
  - The API security basics: HTTPS, CORS, and not trusting the client
  - Lab: secure your API with JWT authentication and verify both the happy path and the rejections
- Documentation, Versioning, and Shipping
  - OpenAPI and Swagger UI: documentation that stays true because it is generated
  - Versioning strategies and evolving an API without breaking clients
  - Testing endpoints and a first look at deployment
  - Lab: publish OpenAPI documentation for your API and walk another attendee through consuming it

Extended Version

The three-day version keeps the same gradient and adds depth on production API concerns:

- Minimal APIs and when to choose them over controllers
- Performance and resilience: caching, pagination done well, and rate limiting
- Integration testing the API with WebApplicationFactory
- A capstone: extend the course API with a new secured, versioned, documented resource, end to end (deeper data access work continues in *Data Access with Entity Framework Core*)