SOFTWARE DEVELOPMENT AND ENGINEERING PRACTICES

Building Web Applications with ASP.NET Core MVC

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

Overview

Server-rendered web applications never went away: a huge share of business software is forms, lists, and workflows, and ASP.NET Core MVC remains one of the most productive and maintainable ways to build them. The framework is mature and well documented, but it rewards developers who understand the pattern behind it. When you know why responsibilities are split between models, views, and controllers, the framework feels obvious; when you do not, every feature feels like ceremony.

This is a hands-on, practitioner course. It starts with the MVC pattern itself and how a request actually travels through an ASP.NET Core application, because everything else in the framework hangs off that mental model. From there it layers up: controllers and routing, Razor views and layouts, models with binding and validation, data access, and finally the security and deployment work that turns a project into an application. Rather than cover every framework feature, it goes deep on the workflow you will use daily, building one application across the whole course. Every module ends with a lab and builds on the one before.

Who Should Attend

- C# developers building or maintaining server-rendered web applications
- Developers moving from ASP.NET Framework, Web Forms, or older MVC to ASP.NET Core
- Back-end developers who want a productive way to put a real UI on their systems

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

Prerequisites

- Working proficiency in C#
- Basic HTML and CSS
- Comfortable running a .NET project locally in Visual Studio or VS Code

What You Will Learn

- Explain the MVC pattern and how a request flows through the ASP.NET Core pipeline
- Build controllers and actions, and configure routing that maps URLs to code
- Create Razor views with layouts, partials, and tag helpers
- Bind and validate user input with view models and data annotations
- Read and write data through Entity Framework Core from an MVC application
- Secure an application with authentication and authorization and prepare it for deployment

Course Outline

Day one: the pattern, and the request path through it

- The MVC Pattern and the ASP.NET Core Pipeline
  - Why separate models, views, and controllers, and what goes wrong when you do not

  - Anatomy of a project: Program.cs, middleware, and dependency injection
  - Following one request from URL to rendered page
  - Lab: create the course project, trace a request through it, and add your first controller and view
- Controllers and Routing
  - Actions, action results, and passing data to views
  - Convention-based and attribute routing, route parameters, and defaults
  - Redirects, status codes, and the post-redirect-get pattern
  - Lab: build the listing and detail pages for the application's core entity
- Razor Views
  - Razor syntax: mixing C# and HTML without making a mess
  - Layouts, sections, partial views, and view components
  - Tag helpers for forms, links, and clean markup
  - Lab: build a shared layout and refactor your pages into partials and components

Day two: input, data, and making it real

- Models, Binding, and Validation
  - View models versus domain models, and why the separation pays off
  - Model binding from forms and routes
  - Server-side validation with data annotations, and surfacing errors in the view
  - Lab: build the create and edit forms with validation that fails gracefully
- Working with Data
  - Wiring in Entity Framework Core: DbContext, entities, and migrations
  - Repositories and services: keeping data access out of controllers
  - Lab: connect the application to a real database and complete the full CRUD workflow
- Security and Shipping
  - Authentication with ASP.NET Core Identity, and authorization on controllers and actions
  - The web security basics MVC handles for you, and the ones it does not: CSRF, XSS, and overposting
  - Configuration, environments, and a first deployment
  - Lab: add login and role-based access to the application and deploy it

Extended Version

The three-day version keeps the same gradient and adds depth and polish:

- Richer UI: client-side validation, AJAX with fetch, and progressive enhancement
- Deeper Entity Framework Core work: relationships, paging, and query performance (continued fully in *Data Access with Entity Framework Core*)
- Testing MVC applications: unit testing controllers and integration testing pages
- A capstone: add a complete new feature area to the course application, from route to database to secured UI