ASP.NET Core Web Development on .NET Core 8

Duration 40 hrs

Module 1: Exploring ASP.NET Core

Microsoft ASP.NET Core web technologies can help you create and host dynamic, powerful, and extensible web applications. ASP.NET Core, is an open-source, cross-platform framework built on .NET, that allows you to build web applications. You can develop and run ASP.NET Core web applications on Windows, macOS, Linux, or any other platform that supports it. ASP.NET Core supports an agile, test-driven development cycle. It also allows you to use the latest HTML standards and front-end frameworks such as Angular, React, and more.

Lessons

Introducing Microsoft Web Technologies

- Getting Started with Razor Pages in ASP.NET Core
- Introducing ASP.NET Core MVC

After completing this module, students will be able to:

- Understand the variety of technologies available in the Microsoft web stack.
- Describe the different programming models available for developers in ASP.NET.
- Describe the role of ASP.NET Core in the web technologies stack, and how to use ASP.NET Core to build web applications.

Module 2: Designing ASP.NET Core Web Applications

Microsoft ASP.NET Core is a programming model that you can use to create powerful and complex web applications. However, all complex development projects, and large projects in particular, can be challenging and intricate to fully understand. Without a complete understanding of the purposes of a project, you cannot develop an effective solution to the customer’s problem. You need to know how to identify a set of business needs, and then make technology choices and plan the web application to meet those needs. The planning phase assures stakeholders that you understand their requirements and communicates the functionality of the web application, its user interface, structure, and data storage to the developers.

Lessons

- Development Methodologies
- Planning in the Project Design Phase

- Choosing between Razor Pages and MVC
- Designing Models, Controllers and Views

After completing this module, students will be able to:

- Plan the overall architecture of an ASP.NET Core MVC web application and consider aspects such as state management.
- Plan the models, controllers, and views that are required to implement a given set of functional requirements.

Module 3: Using Razor Pages and Middleware

ASP.NET Core is a framework that allows us to build many kinds of applications. In this module we'll first look in more detail at ASP.NET Razor Pages, as a quick way of building a web application that doesn't require the complexity of the MVC model. Then we will look at middleware, which has a particular meaning in the context of the ASP.NET Core request pipeline, and potentially allows multiple separate requests to be handled in a completely different fashion and receive separate responses. You will learn how to leverage the ASP.NET Core framework to handle requests and responses via existing, and custom middleware, and how to configure services for use in middleware and throughout other parts of the application, such as controllers. We will also look at Services; classes that expose functionality which you can later use throughout different parts of the application. This is achieved without having to keep track of scope manually in each individual location, or instantiate any dependencies, by using Dependency Injection. Dependency Injection is a technique used by ASP.NET Core that allows us to add dependencies into the code without having to worry about instantiating objects, keeping them in memory, or passing along required dependencies.

Lessons

- Using Razor Pages
- Configuring Middleware
- Configuring Services

After completing this module, students will be able to:

- Build a simple web application using Razor Pages.
- Use existing middleware to set up an ASP.NET Core application.
- Understand the basic principles behind Dependency Injection, and how it is used in ASP.NET Core.

Module 4: Developing Controllers

ASP.NET Core MVC is a framework for building web applications by using the Model-View-Controller (MVC) architectural pattern. The Controller is essentially responsible for processing a web request by interacting with the model and then passing the results to the view. The model represents the business layern, and may include data objects, application logic, and business rules. The View uses the data that it receives from the controller to produce the HTML or other output that is sent back to the browser. In this module we will focus on developing controllers, specialized classes which are central to MVC applications. Understanding how controllers work is crucial to being able to create the appropriate model objects, manipulate them, and pass them to the appropriate views. Controllers have several methods that are called ‘actions’. When an MVC application receives a request, it finds which controller and action should handle the request. It determines this by using Uniform Resource Locator (URL) routing; another very important concept necessary for developing MVC applications. We will also see how to maximize the reuse of code in controllers by writing action filters.

Lessons

- Writing Controllers and Actions
- Configuring Routes
- Writing Action Filters

After completing this module, students will be able to:

- Add a controller to a web application that responds to user actions that are specified in the project design.
- Add routes to the ASP.NET Core routing engine and ensure that URLs are user-friendly in an MVC web application.
- Write code in action filters that runs before or after a controller action.

Module 5: Developing Views

Views are one of the three major components of the Model-View-Controller (MVC) programming model. You can define the user interface for your web application by creating views; a combination of HTML markup and C# code that runs on a web server. To create a view, you need to know how to write the HTML markup and C# code and use the various helper classes that are built into MVC. You also need to know how to create partial views and view components, which render sections of HTML that can be reused in your web application. We will also look in more detail at Razor markup syntax for embedding .NET based code into webpages.

Lessons

- Creating Views with Razor Syntax
- Using HTML Helpers and Tag Helpers
- Reusing Code in Views

After completing this module, students will be able to:

- Create an MVC view and add Razor markup to it to display data to users.
- Use HTML helpers and tag helpers in a view.
- Reuse Razor markup in multiple locations throughout an application.

Module 6: Developing Models

Most web applications interact with various types of data or objects. An e-commerce application, for example, manages products, shopping carts, customers, and orders. A social networking application might help manage users, status updates, comments, photos, and videos. A blog is used to manage blog entries, comments, categories, and tags. When you write a Model-View-Controller (MVC) web application, you create an MVC model to model the data for your web application. Within this model, you create a model class for each type of object. The model class describes the properties of each type of object and can include business logic that matches business processes. Therefore, the model is a fundamental building-block in an MVC application. We will also look at validation of user input.

Lessons

- Creating MVC Models
- Working with Forms
- Validating User Input

After completing this module, students will be able to:

- Add a model to an MVC application and write code in it to implement the business logic.
- Use display and edit data annotations.
- Validate user input with data annotations.

Module 7: Using Entity Framework Core in ASP.NET Core

Web applications often require a data store for dynamic information, for example to create a web application that changes continually in response to user input, administrative actions, and publishing events. The data store is usually a database,

but other types of data stores are also used. In Model-View-Controller (MVC) applications, you can create a model that implements data access logic and business logic. Alternatively, you can separate business logic from data access logic by using a repository class that a controller can use to read from or write to an underlying data store. When you write an ASP.NET application you can use the Entity Framework Core (EF Core) and Language Integrated Query (LINQ) technologies, which make data access code very quick to write and simple to understand. In this module, you will see how to build a database-driven website in ASP.NET Core using Entity Framework.

Lessons

- Introduction to Entity Framework Core
- Working with Entity Framework Core
- Using Entity Framework Core Database Providers

After completing this module, students will be able to:

- Connect an application to a database to access and store data.
- Explain EF Core.
- Work with Entity Framework Core.

Module 8: Using Layouts, CSS and JavaScript in ASP.NET Core

While building web applications, you should apply a consistent look and feel to the application. You would typically include consistent header and footer sections and navigation controls in all the views. Microsoft ASP.NET Core uses special templates called layouts to achieve this, along with cascading style sheets (CSS) to enhance the appearance and usability of your web application. You can also create interactive HTML elements by using JavaScript to provide client-side code in your web application, along with client-side JavaScript libraries.

Lessons

- Using Layouts
- Using CSS
- Using JavaScript

After completing this module, students will be able to:

- Apply a consistent layout to ASP.NET Core MVC applications.
- Add JavaScript code to your web application.
- Use CSS stylesheets.

Module 9: Client-Side Development

When creating an application, it is important to know how to develop both client-side and server-side code for the application. In this module, you are going to learn client-side tools that will allow you to create complex web applications on any scale, including using the Bootstrap CSS framework to style your web application. You will learn how to use Sass, a CSS pre-processor that adds code-like features such as variables, nested rules, and functions, that improve the maintainability of complex CSS stylesheets. You will learn responsive design principles that allow you to adapt your web application based on the capabilities of the web browser or device using CSS media queries, and how to use a responsive grid system. Next, you will learn how to set up the gulp task runner and use it to compile Sass files during the build and perform bundling and minification of CSS and JavaScript files, and how to set up a watcher task to automatically compile Sass files as you write your code. Finally, we'll introduce the Blazor framework for building an interactive client-side web UI with .NET.

Lessons

- Responsive Web Design
- Using Front-end Development Tools
- Looking at ASP.NET Core Blazor

After completing this module, students will be able to:

- Use Bootstrap and SASS in a Microsoft ASP.NET Core application.
- Use front-end development tools.
- Ensure that a web application displays correctly on devices with different screen sizes.
- Understand ASP.NET Core Blazor applications.

Module 10: Managing Security

Web applications are normally delivered through a web browser, by means of the public Internet, to large numbers of users. This means that security must always be at the forefront of your mind when building these applications, because as well as legitimate users, the application will be exposed to malicious third parties. Users may have anonymous access, or they may have a signed-in identity, and you must decide which users can perform what actions. Authentication is the act of establishing a user's identity, while authorization is the process where an already authenticated user is granted access to specific actions or resources. By utilizing authorization, you can prevent users from accessing sensitive material or information and resources

intended for another user or prevent them from performing certain actions. The costs of security breaches can be very high, resulting in loss of data, legal action, and reputational damage. So, in the final section we will look at some specific malicious attacks such as cross-site scripting and SQL injection, and how to defend against them.

Lessons

- Authentication in ASP.NET Core
- Authorization in ASP.NET Core
- Defending from Common Attacks

After completing this module, students will be able to:

- Add basic authentication to your application.
- Configure Microsoft ASP.NET Core Identity.
- Add basic authorization to your application.
- Understand how security exploits work and how to better defend against them.

Module 11: Performance and Communication

Modern web applications need to be able to respond quickly to large numbers of user requests within a small timeframe. Caching allows you to store common requests, avoiding the need to perform the same logic repeatedly. This provides the user with a fast response time and reduces system resources used in conducting the logic for the action. By utilizing various forms of state management, you can build stateful applications on top of stateless web protocols, to give responses tailored to individual user contexts within the same application. Finally, SignalR is an easy-to-use bi-directional communications API that is an abstraction over several different web communications protocols. This allows you to build server-side logic to push content to browser-based web applications in real time.

Lessons

- Implementing a Caching Strategy
- Managing State
- Supporting Two-way Communication

After completing this module, students will be able to:

- Implement caching in a Microsoft ASP.NET Core application.

- Use state management technologies to improve the client experience, by providing a consistent experience for the user.
- Implement two-way communication by using SignalR, allowing the server to notify the client when important events occur.

Module 12: Implementing Web APIs

Most web applications require integration with external systems. Representational State Transfer (REST) services help reduce application overhead and limit the data that is transmitted between client and server systems using open standards. You need to know how to expose a Web API that implements REST services in your ASP.NET application. You also need to know how to call a Web API by using both server-side and client-side code to consume external REST-style Web APIs.

Lessons

- Introducing Web APIs
- Developing a Web API
- Calling a Web API

After completing this module, students will be able to:

- Create services by using ASP.NET Core Web API.
- Call a Web API from server-side code.
- Call a Web API from client-side code.

Module 13: Hosting Multiple Web Applications in IIS Manager (2019)
Hosting multiple web applications on a single server efficiently requires a proper configuration of IIS (Internet Information Services) Manager. This module explores the setup, configurations for managing multiple sites.
Lessons:

- Introduction to IIS Manager (2019)
- Configuring Application Pools and Sites
- Setting Up Multiple Web Applications on a Single Server

Module 14: CRUD Operations in ASP.NET Core
Implementing Create, Read, Update, and Delete (CRUD) operations is fundamental in web applications. This module focuses on how to efficiently develop CRUD functionality using ASP.NET Core with Entity Framework Core.
Lessons:

- Understanding CRUD Operations and MVC Pattern

- Configuring Entity Framework Core for Database Interaction
- Implementing Create, Read, Update, and Delete Functions
- Handling Validation and Error Management in CRUD Operations
- Using APIs for CRUD Operations in ASP.NET Core

Module 15: Creating Progressive Web Applications (PWAs)
Progressive Web Applications (PWAs) enhance the user experience by providing app-like functionality within web browsers. This module covers how to create PWAs in ASP.NET Core.
Lessons:

- Introduction to PWAs and Their Advantages
- Setting Up Service Workers and Manifest Files
- Enhancing Performance and User Experience with PWAs
- Deploying and Testing a PWA