Developing ASP.NET Core MVC Web Applications

Duration: 5 Days **Course Code: M20486**

Overview:

In this 5-day course, the professional web developers will learn to develop advanced ASP.NET Core MVC applications using .NET Core tools and technologies. The focus will be on coding activities that enhance the performance and scalability of the Web site application.

Target Audience:

This course is intended for professional web developers who use Microsoft Visual Studio in an individual-based or team-based, small-sized to large development environment. Candidates for this course are interested in developing advanced web applications and want to manage the rendered HTML comprehensively. They want to create websites that separate the user interface, data access, and application logic.

Objectives:

- After completing this course, students will be able to:
- Describe the Microsoft Web Technologies stack and select an appropriate technology to use to develop any given application.
- Design the architecture and implementation of a web application that will meet a set of functional requirements, user interface requirements, and address business models.
- Configure the pipeline of ASP.NET Core web applications using middleware, and leverage dependency injection across MVC application.
- Add Controllers to an MVC Application to manage user interaction, update models, and select and return Views.
- Develop a web application that uses the ASP.NET Core routing engine to present friendly URLs and a logical navigation hierarchy to users.
- Create Views in an MVC application that display and edit data and interact with Models and Controllers.
- Create MVC Models and write code that implements business logic within Model methods, properties, and events.
- Connect an ASP.NET Core application to a database using Entity Framework Core.
- Implement a consistent look and feel across an entire MVC web application.
- Write JavaScript code that runs on the client-side and utilizes the jQuery script library to optimize the responsiveness of an MVC web application.
- Add client side packages and configure Task Runners.
- Run unit tests and debugging tools against a web application in Visual Studio 2017.
- Write an MVC application that authenticates and authorizes users to access content securely using Identity.
- Build an MVC application that resists malicious attacks.
- Use caching to accelerate responses to user requests.
- Use SignalR to enable two-way communication between client and server.
- Describe what a Web API is and why developers might add a Web API to an application.
- Describe how to package and deploy an ASP.NET Core MVC web application from a development computer to a web server.

Prerequisites:

Before attending this course, students must have:

- Experience with Visual Studio 2017.
- Experience with C# programming, and concepts such as Lambda expressions, LINQ, and anonymous types.

Testing and Certification

- Experience in using the .NET Framework.
- Experience with HTML, CSS and JavaScript.
- Experience with querying and manipulating data with ADO.NET.
- Knowledge of XML and JSON data structures.

Content:

Module 1: Exploring ASP.NET Core MVC

Microsoft ASP.NET Core MVC and the other web technologies of the ASP.NET Core can help you create and host dynamic, powerful, and extensible web applications. ASP.NET Core, of which ASP.NET Core MVC is part, is an open-source, cross-platform framework 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 MVC supports agile, test-driven development cycle. It also allows you to use the latest HTML standard and Front-End frameworks such as Angular, React, and more.

- Overview of Microsoft Web Technologies
- Overview of ASP.NET 4.x
- Introduction to ASP.NET Core MVCLab : Exploring ASP.NET Core MVC
- Exploring a Razor Pages Application
- Exploring a Web API Application
- Exploring an MVC Application

After completing this course, 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.
- Choose between ASP.NET Core and ASP.NET 4.x.
- Describe the role of ASP.NET Core MVC in the web technologies stack, and how to use ASP.NET Core MVC to build web applications.
- Distinguish between MVC models, MVC controllers, and MVC views.
- Run unit tests against the Model–View–Controller (MVC) components, such as model classes and controllers, and locate potential bugs.
- Build a Microsoft ASP.NET Core MVC application that handles exceptions smoothly and robustly.
- Run logging providers that benefit your applications and run them by using a common logging API.

Module 2: Designing ASP.NET Core MVC Web Applications

Microsoft ASP.NET Core MVC 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

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.
- Use existing middleware to set up an ASP.NET Core application.
- Create your own middleware and use it to define custom behavior.
- Understand the basic principles behind Dependency Injection, and how it is used in ASP.NET Core.
- Know how to create a custom service, configure its scope, and inject it to both middleware and ASP.NET Core MVC controllers.
- 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.
- Create an MVC view and add Razor markup to it to display data to users.
- 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.
- Connect an application to a database to access and store data.
- Explain EF Core.
- Work with Entity Framework Core.
- Use EF Core to connect to a database including Microsoft SQL Server.
- Apply a consistent layout to ASP.NET Core MVC applications.
- Add JavaScript code to your web application.
- Use the jQuery library in your web application.
- Use Bootstrap, Sass and Less in a Microsoft ASP.NET Core application.
- Use task runners in an ASP.NET Core application.
- Ensure that a web application displays correctly on devices with different screen sizes.
- Add basic authentication to your application.
- Configure Microsoft ASP.NET Core Identity.
- Add basic authorization to your application.
- Utilize several different authorization approaches.

It is also important that you know how to handle exceptions while they occur. While an application is running, you may encounter unexpected occurrences. It is important that you manage your exceptions correctly and provide a good user feedback while avoiding leaking information about the application structure. Finally, by using logs throughout the application, you can monitor user activities that might lead to unexpected issues and then you can find solutions to bugs, which you normally would be unsure how to reproduce, by following flows which occurred on the production environment and finding additional errors.

- Testing MVC Applications
- Implementing an Exception Handling Strategy
- Logging MVC ApplicationsLab : Testing and troubleshooting
- Testing a Model
- Testing a controller using a fake repository
- Implementing a repository in MVC project
- Add exception handling
- Add logging

After completing this course, 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.
- Choose between ASP.NET Core and ASP.NET 4.x.
- Describe the role of ASP.NET Core MVC in the web technologies stack, and how to use ASP.NET Core MVC to build web applications.
- Distinguish between MVC models, MVC controllers, and MVC views.
- Run unit tests against the Model–View–Controller (MVC) components, such as model classes and controllers, and locate potential bugs.
- Build a Microsoft ASP.NET Core MVC application that handles exceptions smoothly and robustly.
- Run logging providers that benefit your applications and run them by using a common logging API.

Module 11: Managing Security

Since web applications are normally targeted towards users utilizing only a browser to use the application, there are likely to be far more users than in the case of installed applications. However, the open nature of a web application means security must always

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 plan the Model-View-Controller (MVC) web application to meet those needs.

The project plan that you create 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. By writing a detailed and accurate project plan, you can ensure that the powerful features of MVC are used effectively to solve the customer's business problems.

- Planning in the Project Design Phase
- Designing Models, Controllers and ViewsLab : Designing ASP.NET Core MVC Web Applications
- Planning Model Classes
- Planning Controllers
- Planning Views
- Architecting and MVC Web Application

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.
- Use existing middleware to set up an ASP.NET Core application.
- Create your own middleware and use it to define custom behavior.
- Understand the basic principles behind Dependency Injection, and how it is used in ASP.NET Core.
- Know how to create a custom service, configure its scope, and inject it to both middleware and ASP.NET Core MVC controllers.
- 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.
- Create an MVC view and add Razor markup to it to display data to users.
- 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.
- Connect an application to a database to access and store data.
- Explain EF Core.
- Know how security exploits work and how to better defend against them.
- 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.
- Create services by using ASP.NET Core Web API.
- Call a Web API from server-side code and jQuery.
- Host and Deploy an ASP.NET Core MVC application on IIS.
- Host and Deploy an ASP.NET Core MVC application on Microsoft Azure.
- Be able to utilize services offered by Microsoft Azure to improve the capabilities of your web applications.

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. In this module, you will learn how to create the code for models.

- Creating MVC Models
- Working with Forms
- Validate MVC ApplicationLab : Developing Models
- Adding a model
- Working with Forms
- Add Validation

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

be at the forefront of your mind when building them. As part of security, you must decide which users can perform what actions, all while protecting users and your application from malicious third parties with possible access to your application. Authentication is the act of utilizing several parameters to make sure that a user is who they claim to be. By implementing authentication, you can ascertain who a user is and provide them with appropriate content while utilizing your applications.

Authorization is the process where an already authenticated user in the application can be granted access to specific actions or resources. By utilizing authorization, you can prevent users from accessing sensitive material not intended from them or from performing actions which they should not be able to. Finally, at some point in its lifespan, your applications may come under attack by malicious users. These can vary in means and intent, but the cost of being undefended can be great. You may lose potential users who are affected, valuable data could be erroneously changed, and in the worst cases the entire application may become unusable. Solutions to some of the most popular attacks will be reviewed in this module.

- Authentication in ASP.NET Core
- Authorization in ASP.NET Core
- Defending from AttacksLab : Managing Security
- Use Identity
- Add Authorization
- Avoid the Cross-Site Request Forgery Attack

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.
- Use existing middleware to set up an ASP.NET Core application.
- Create your own middleware and use it to define custom behavior.
- Understand the basic principles behind Dependency Injection, and how it is used in ASP.NET Core.
- Know how to create a custom service, configure its scope, and inject it to both middleware and ASP.NET Core MVC controllers.
- Add a controller to a web application that responds to user actions that are specified in the project design.

- Work with Entity Framework Core.
- Use EF Core to connect to a database including Microsoft SQL Server.
- Apply a consistent layout to ASP.NET Core MVC applications.
- Add JavaScript code to your web application.
- Use the jQuery library in your web application.
- Use Bootstrap, Sass and Less in a Microsoft ASP.NET Core application.
- Use task runners in an ASP.NET Core application.
- Ensure that a web application displays correctly on devices with different screen sizes.
- Add basic authentication to your application.
- Configure Microsoft ASP.NET Core Identity.
- Add basic authorization to your application.
- Utilize several different authorization approaches.
- Know how security exploits work and how to better defend against them.
- 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.
- Create services by using ASP.NET Core Web API.
- Call a Web API from server-side code and jQuery.
- Host and Deploy an ASP.NET Core MVC application on IIS.
- Host and Deploy an ASP.NET Core MVC application on Microsoft Azure.
- Be able to utilize services offered by Microsoft Azure to improve the capabilities of your web applications.

Module 3: Configure Middlewares and Services in ASP.NET Core

ASP.NET Core is a framework that allows us to build many different kinds of applications. In this module, 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. A middleware is a segment of code that can be used as part of the request and response pipeline that allows us to handle them according to any relevant parameter. This potentially allows multiple separate requests to be handled in a completely different fashion and receive separate responses.

Services are classes that expose functionality which you can later use throughout different parts of the application, without having to keep track of scope manually in each individual of functional requirements.

- Use existing middleware to set up an ASP.NET Core application.
- Create your own middleware and use it to define custom behavior.
- Understand the basic principles behind Dependency Injection, and how it is used in ASP.NET Core.
- Know how to create a custom service, configure its scope, and inject it to both middleware and ASP.NET Core MVC controllers.
- 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.
- Create an MVC view and add Razor markup to it to display data to users.
- 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.
- Connect an application to a database to access and store data.
- Explain EF Core.
- Work with Entity Framework Core.
- Use EF Core to connect to a database including Microsoft SQL Server.
- Apply a consistent layout to ASP.NET Core MVC applications.
- Add JavaScript code to your web application.
- Use the jQuery library in your web application.
- Use Bootstrap, Sass and Less in a Microsoft ASP.NET Core application.
- Use task runners in an ASP.NET Core application.
- Ensure that a web application displays correctly on devices with different screen sizes.
- Add basic authentication to your application.
- Configure Microsoft ASP.NET Core Identity.
- Add basic authorization to your application.
- Utilize several different authorization approaches.
- Know how security exploits work and how to better defend against them.
- 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.
- Create services by using ASP.NET Core Web API.
- Call a Web API from server-side code

- 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.
- Create an MVC view and add Razor markup to it to display data to users.
- 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.
- Connect an application to a database to access and store data.
- Explain EF Core.
- Work with Entity Framework Core.
- Use EF Core to connect to a database including Microsoft SQL Server.
- Apply a consistent layout to ASP.NET Core MVC applications.
- Add JavaScript code to your web application.
- Use the jQuery library in your web application.
- Use Bootstrap, Sass and Less in a Microsoft ASP.NET Core application.
- Use task runners in an ASP.NET Core application.
- Ensure that a web application displays correctly on devices with different screen sizes.
- Add basic authentication to your application.
- Configure Microsoft ASP.NET Core Identity.
- Add basic authorization to your application.
- Utilize several different authorization approaches.
- Know how security exploits work and how to better defend against them.
- 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.
- Create services by using ASP.NET Core Web API.
- Call a Web API from server-side code and jQuery.
- Host and Deploy an ASP.NET Core MVC application on IIS.
- Host and Deploy an ASP.NET Core MVC application on Microsoft Azure.
- Be able to utilize services offered by Microsoft Azure to improve the capabilities of your web applications.

Module 12: Performance and Communication

Modern web applications require complex interactions with users. Users will often request a lot of data in a small time-frame, while also expecting relevant data as soon as it comes online. This can easily cause a

location and instantiate any dependencies. This is done 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. This allows the application to become more flexible and to reduce potential points of failure whenever you change a service.

- Configuring Middlewares
- Configuring ServicesLab : Configuring Middleware and Services in ASP.NET Core
- Working with Static Files
- Creating custom middleware
- Using dependency injection
- Injecting a service to a controller

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.
- Use existing middleware to set up an ASP.NET Core application.
- Create your own middleware and use it to define custom behavior.
- Understand the basic principles behind Dependency Injection, and how it is used in ASP.NET Core.
- Know how to create a custom service, configure its scope, and inject it to both middleware and ASP.NET Core MVC controllers.
- 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.
- Create an MVC view and add Razor markup to it to display data to users.
- 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.
- Connect an application to a database to access and store data.
- Explain EF Core.
- Work with Entity Framework Core.
- Use EF Core to connect to a database including Microsoft SQL Server.
- Apply a consistent layout to ASP.NET Core MVC applications.
- Add JavaScript code to your web application.

and jQuery.

- Host and Deploy an ASP.NET Core MVC application on IIS.
- Host and Deploy an ASP.NET Core MVC application on Microsoft Azure.
- Be able to utilize services offered by Microsoft Azure to improve the capabilities of your web applications.

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

Web applications often use information and they usually require a data store for that information. By rendering webpages that use data from a data store, you can 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 occasionally 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. A repository is a class that a controller can call to read data from a data store and to write data to a data store. When you write an ASP.NET application you can use the Entity Framework Core (EF Core) and Language Integrated Query (LINQ) technologi

- Introduction to Entity Framework Core
- Working with Entity Framework Core
- Use Entity Framework Core to connect to Microsoft SQL ServerLab : Using Entity Framework Core in ASP.NET Core
- Adding Entity Framework Core
- Use Entity Framework Core to retrieve and store data
- Use Entity Framework Core to connect to Microsoft SQL Server

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.
- Use existing middleware to set up an ASP.NET Core application.
- Create your own middleware and use it to define custom behavior.
- Understand the basic principles behind Dependency Injection, and how it is used in ASP.NET Core.
- Know how to create a custom service, configure its scope, and inject it to both middleware and ASP.NET Core MVC

significant amount of load on an unprepared server, resulting in unnecessarily complex or repeated operations and a heavy load on your server. Fortunately, there are multiple ways to reduce the load. Caching allows you to store commonly repeated requests, preventing the need to perform the same logic repeatedly. By using caching, you can reuse data that has already been loaded and present it to the user. This provides the user with a fast response time and reduces system resources used in conducting the logic for the action. State meanwhile allows achieving a state of consistency between different requests.

By utilizing various forms of state management, you can transform the normally stateless web experience into one that is custom tailored to individual clients, with different users enjoying a separate and relevant experience in the same application. Finally, SignalR is a framework that allows the abstraction of several different communication protocols into an easy to use API, which allows you to easily create a single set of tools on the server and client to facilitate two-way communications. This allows you to focus on the logic you wish to implement while allowing you to not have to cater to specific browsers.

- Implementing a Caching Strategy
- Managing State
- Two-way communicationLab : Performance and Communication
- Implementing a Caching Strategy
- Managing state
- Two-Way communication

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.
- Use existing middleware to set up an ASP.NET Core application.
- Create your own middleware and use it to define custom behavior.
- Understand the basic principles behind Dependency Injection, and how it is used in ASP.NET Core.
- Know how to create a custom service, configure its scope, and inject it to both middleware and ASP.NET Core MVC controllers.
- 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

- Use the jQuery library in your web application.
- Use Bootstrap, Sass and Less in a Microsoft ASP.NET Core application.
- Use task runners in an ASP.NET Core application.
- Ensure that a web application displays correctly on devices with different screen sizes.
- Add basic authentication to your application.
- Configure Microsoft ASP.NET Core Identity.
- Add basic authorization to your application.
- Utilize several different authorization approaches.
- Know how security exploits work and how to better defend against them.
- 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.
- Create services by using ASP.NET Core Web API.
- Call a Web API from server-side code and jQuery.
- Host and Deploy an ASP.NET Core MVC application on IIS.
- Host and Deploy an ASP.NET Core MVC application on Microsoft Azure.
- Be able to utilize services offered by Microsoft Azure to improve the capabilities of your web applications.

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 layer, sometimes referred to as the domain, 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, you will learn how to develop controllers.

Controllers 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. A controller is a class. It contains several methods. These methods 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. URL routing is another very important concept necessary for developing MVC applications. The ASP.NET Core MVC

controllers.
- 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.
- Create an MVC view and add Razor markup to it to display data to users.
- 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.
- Connect an application to a database to access and store data.
- Explain EF Core.
- Work with Entity Framework Core.
- Use EF Core to connect to a database including Microsoft SQL Server.
- Apply a consistent layout to ASP.NET Core MVC applications.
- Add JavaScript code to your web application.
- Use the jQuery library in your web application.
- Use Bootstrap, Sass and Less in a Microsoft ASP.NET Core application.
- Use task runners in an ASP.NET Core application.
- Ensure that a web application displays correctly on devices with different screen sizes.
- Add basic authentication to your application.
- Configure Microsoft ASP.NET Core Identity.
- Add basic authorization to your application.
- Utilize several different authorization approaches.
- Know how security exploits work and how to better defend against them.
- 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.
- Create services by using ASP.NET Core Web API.
- Call a Web API from server-side code and jQuery.
- Host and Deploy an ASP.NET Core MVC application on IIS.
- Host and Deploy an ASP.NET Core MVC application on Microsoft Azure.
- Be able to utilize services offered by Microsoft Azure to improve the capabilities of your web applications.

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

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.
- Create an MVC view and add Razor markup to it to display data to users.
- 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.
- Connect an application to a database to access and store data.
- Explain EF Core.
- Work with Entity Framework Core.
- Use EF Core to connect to a database including Microsoft SQL Server.
- Apply a consistent layout to ASP.NET Core MVC applications.
- Add JavaScript code to your web application.
- Use the jQuery library in your web application.
- Use Bootstrap, Sass and Less in a Microsoft ASP.NET Core application.
- Use task runners in an ASP.NET Core application.
- Ensure that a web application displays correctly on devices with different screen sizes.
- Add basic authentication to your application.
- Configure Microsoft ASP.NET Core Identity.
- Add basic authorization to your application.
- Utilize several different authorization approaches.
- Know how security exploits work and how to better defend against them.
- 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.
- Create services by using ASP.NET Core Web API.
- Call a Web API from server-side code and jQuery.
- Host and Deploy an ASP.NET Core MVC application on IIS.
- Host and Deploy an ASP.NET Core MVC application on Microsoft Azure.
- Be able to utilize services offered by Microsoft Azure to improve the capabilities of your web applications.

Implement two-way communication by using SignalR, allowing the server to notify the client when important events occur.

Module 13: Implementing Web APIs

Most web applications require integration with

framework includes a flexible URL routing system that enables you to define URL mapping rules within your applications. To maximize the reuse of code in controllers, it is important to know how to write action filters. You can use action filters to run code before o

- Writing Controllers and Actions
- Configuring Routes
- Writing Action FiltersLab : Developing Controllers
- Adding controllers and actions to an MVC application
- Configuring routes by using the routing table
- Configuring routes using attributes
- Adding an action filer

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.
- Use existing middleware to set up an ASP.NET Core application.
- Create your own middleware and use it to define custom behavior.
- Understand the basic principles behind Dependency Injection, and how it is used in ASP.NET Core.
- Know how to create a custom service, configure its scope, and inject it to both middleware and ASP.NET Core MVC controllers.
- 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.
- Create an MVC view and add Razor markup to it to display data to users.
- 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.
- Connect an application to a database to access and store data.
- Explain EF Core.
- Work with Entity Framework Core.
- Use EF Core to connect to a database including Microsoft SQL Server.
- Apply a consistent layout to ASP.NET Core MVC applications.
- Add JavaScript code to your web application.
- Use the jQuery library in your web application.
- Use Bootstrap, Sass and Less in a Microsoft

While building web applications, you should apply a consistent look and feel to the application. You should include consistent header and footer sections in all the views. Microsoft ASP.NET Core MVC includes features such as cascading style sheets (CSS) styles and layouts that enhance the appearance and usability of your web application. In ASP.NET Core MVC, you can create interactive HTML elements by using JavaScript. You need to know how to use JavaScript in your web application. To simplify adding JavaScript to your web application, you need to know how to use libraries such as jQuery.

- Using Layouts
- Using CSS and JavaScript
- Using jQueryLab : Using Layouts, CSS and JavaScript in ASP.NET Core
- Applying a layout and link views to it
- Using CSS
- Using JavaScript
- Using jQuery

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.
- Use existing middleware to set up an ASP.NET Core application.
- Create your own middleware and use it to define custom behavior.
- Understand the basic principles behind Dependency Injection, and how it is used in ASP.NET Core.
- Know how to create a custom service, configure its scope, and inject it to both middleware and ASP.NET Core MVC controllers.
- 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.
- Create an MVC view and add Razor markup to it to display data to users.
- 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.
- Connect an application to a database to access and store data.

external systems such as mobile applications. You need to know how to use Web APIs to promote application interaction with external systems. You can use the Web API to implement Representational State Transfer (REST) services in your application. REST services help reduce application overhead and limit the data that is transmitted between client and server systems. You need to know how to call a Web API by using server-side code and jQuery code to effectively implement REST-style Web APIs in your application.

- Introducing Web APIs
- Developing a Web API
- Calling a Web APILab : Implementing Web APIs
- Adding Actions and Call Them Using Microsoft Edge
- Calling a Web API using server-side code
- Calling a Web API using jQuery

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.
- Use existing middleware to set up an ASP.NET Core application.
- Create your own middleware and use it to define custom behavior.
- Understand the basic principles behind Dependency Injection, and how it is used in ASP.NET Core.
- Know how to create a custom service, configure its scope, and inject it to both middleware and ASP.NET Core MVC controllers.
- 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.
- Create an MVC view and add Razor markup to it to display data to users.
- 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.
- Connect an application to a database to access and store data.
- Explain EF Core.
- Work with Entity Framework Core.
- Use EF Core to connect to a database including Microsoft SQL Server.

ASP.NET Core application.
- Use task runners in an ASP.NET Core application.
- Ensure that a web application displays correctly on devices with different screen sizes.
- Add basic authentication to your application.
- Configure Microsoft ASP.NET Core Identity.
- Add basic authorization to your application.
- Utilize several different authorization approaches.
- Know how security exploits work and how to better defend against them.
- 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.
- Create services by using ASP.NET Core Web API.
- Call a Web API from server-side code and jQuery.
- Host and Deploy an ASP.NET Core MVC application on IIS.
- Host and Deploy an ASP.NET Core MVC application on Microsoft Azure.
- Be able to utilize services offered by Microsoft Azure to improve the capabilities of your web applications.

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 view is a combination of HTML markup and C# code that runs on a web server. Therefore, 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.

- Creating Views with Razor Syntax
- Using HTML Helpers and Tag Helpers
- Reusing Code in ViewsLab : Developing Views
- Adding Views to an MVC Application
- Adding a partial view
- Adding a view component
- Explain EF Core.
- Work with Entity Framework Core.
- Use EF Core to connect to a database including Microsoft SQL Server.
- Apply a consistent layout to ASP.NET Core MVC applications.
- Add JavaScript code to your web application.
- Use the jQuery library in your web application.
- Use Bootstrap, Sass and Less in a Microsoft ASP.NET Core application.
- Use task runners in an ASP.NET Core application.
- Ensure that a web application displays correctly on devices with different screen sizes.
- Add basic authentication to your application.
- Configure Microsoft ASP.NET Core Identity.
- Add basic authorization to your application.
- Utilize several different authorization approaches.
- Know how security exploits work and how to better defend against them.
- 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.
- Create services by using ASP.NET Core Web API.
- Call a Web API from server-side code and jQuery.
- Host and Deploy an ASP.NET Core MVC application on IIS.
- Host and Deploy an ASP.NET Core MVC application on Microsoft Azure.
- Be able to utilize services offered by Microsoft Azure to improve the capabilities of your web applications.

Module 9: Client-Side Development

When creating an application, it is important to know how to develop both client-side and server-side of the application. In this module, you are going to learn client-side tools that will allow you to create extensive web applications on any scale. These tools are based on the topics covered in Module 8, "Using Layouts, CSS and JavaScript in ASP.NET Core MVC". In this module, you are going to learn how to use the Bootstrap framework to style your web application. Then you are going to learn how to use Sass and Less, two common Cascading Style Sheets (CSS) preprocessors that add features to CSS, such as variables, nested rules, and functions. These greatly improve the maintainability of complex web applications.
- Apply a consistent layout to ASP.NET Core MVC applications.
- Add JavaScript code to your web application.
- Use the jQuery library in your web application.
- Use Bootstrap, Sass and Less in a Microsoft ASP.NET Core application.
- Use task runners in an ASP.NET Core application.
- Ensure that a web application displays correctly on devices with different screen sizes.
- Add basic authentication to your application.
- Configure Microsoft ASP.NET Core Identity.
- Add basic authorization to your application.
- Utilize several different authorization approaches.
- Know how security exploits work and how to better defend against them.
- 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.
- Create services by using ASP.NET Core Web API.
- Call a Web API from server-side code and jQuery.
- Host and Deploy an ASP.NET Core MVC application on IIS.
- Host and Deploy an ASP.NET Core MVC application on Microsoft Azure.
- Be able to utilize services offered by Microsoft Azure to improve the capabilities of your web applications.

Module 14: Hosting and Deployment

ASP.NET Core MVC applications are designed to provide a service to multiple users simultaneously while only requiring the server to be installed, and the clients to use browsers to access it. This results in highly desirable applications which do not rely on the user installing dedicated software, and ensuring it is accessible by clients on a wide variety of machines. In order to setup an ASP.NET Core application for a production environment, you will need to compile your code and compress it, and then set it up and running on a dedicated server. Hosting involves using a dedicated server to contain the compiled application and serve it to users as a web-based service. There are many different technologies which can be used to host your application and you should choose one that is appropriate for your requirements. Deployment is th

- On-premise hosting and deployment
- Deployment to Microsoft Azure

Next, you will learn how to set up task runners such as Grunt and gulp and how to use them to compile Sass files during the Microsoft Visual Studio build. You will learn how to use the gulp task runner to perform bundling and minification of CSS and JavaScript files and how to set up a watcher task to compile Sass files as you write your code, without the need to rebuild the solution. Finally, you will learn responsive design tools that allow you to customize your web application's display based on the capabilities and specifications of a web browser or a device. You will learn to write CSS media queries, how to use the Bootstrap responsive grid system, and how to apply the CSS flexbox layout to your views.

- Applying Styles
- Using Task Runners
- Responsive designLab : Client-Side Development
- Use gulp to run tasks
- Styling using Sass
- Using Bootstrap

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.
- Use existing middleware to set up an ASP.NET Core application.
- Create your own middleware and use it to define custom behavior.
- Understand the basic principles behind Dependency Injection, and how it is used in ASP.NET Core.
- Know how to create a custom service, configure its scope, and inject it to both middleware and ASP.NET Core MVC controllers.
- 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.
- Create an MVC view and add Razor markup to it to display data to users.
- 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.
- Connect an application to a database to access and store data.
- Microsoft Azure FundamentalsLab : Hosting and Deployment
- Deploying a Web Application to Microsoft Azure
- Upload an Image to Azure Blob Storage

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.
- Use existing middleware to set up an ASP.NET Core application.
- Create your own middleware and use it to define custom behavior.
- Understand the basic principles behind Dependency Injection, and how it is used in ASP.NET Core.
- Know how to create a custom service, configure its scope, and inject it to both middleware and ASP.NET Core MVC controllers.
- 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.
- Create an MVC view and add Razor markup to it to display data to users.
- 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.
- Connect an application to a database to access and store data.
- Explain EF Core.
- Work with Entity Framework Core.
- Use EF Core to connect to a database including Microsoft SQL Server.
- Apply a consistent layout to ASP.NET Core MVC applications.
- Add JavaScript code to your web application.
- Use the jQuery library in your web application.
- Use Bootstrap, Sass and Less in a Microsoft ASP.NET Core application.
- Use task runners in an ASP.NET Core application.
- Ensure that a web application displays correctly on devices with different screen sizes.
- Add basic authentication to your application.
- Configure Microsoft ASP.NET Core Identity.

- Explain EF Core.
- Work with Entity Framework Core.
- Use EF Core to connect to a database including Microsoft SQL Server.
- Apply a consistent layout to ASP.NET Core MVC applications.
- Add JavaScript code to your web application.
- Use the jQuery library in your web application.
- Use Bootstrap, Sass and Less in a Microsoft ASP.NET Core application.
- Use task runners in an ASP.NET Core application.
- Ensure that a web application displays correctly on devices with different screen sizes.
- Add basic authentication to your application.
- Configure Microsoft ASP.NET Core Identity.
- Add basic authorization to your application.
- Utilize several different authorization approaches.
- Know how security exploits work and how to better defend against them.
- 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.
- Create services by using ASP.NET Core Web API.
- Call a Web API from server-side code and jQuery.
- Host and Deploy an ASP.NET Core MVC application on IIS.
- Host and Deploy an ASP.NET Core MVC application on Microsoft Azure.
- Be able to utilize services offered by Microsoft Azure to improve the capabilities of your web applications.

Module 10: Testing and Troubleshooting

Software systems such as web applications are complex and require multiple components, which are often written by different developers, to work together. Incorrect assumptions, inaccurate understanding, coding errors, and many other sources can create bugs that result in exceptions or unexpected behavior. To improve the quality of your web application and create a satisfying user experience, you must identify bugs from any source and eliminate them. Traditionally, testers perform most of the testing at the end of a development project. However, it has recently become widely accepted that testing throughout the project life cycle improves quality and ensures that there are no bugs in

- Add basic authorization to your application.
- Utilize several different authorization approaches.
- Know how security exploits work and how to better defend against them.
- 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.
- Create services by using ASP.NET Core Web API.
- Call a Web API from server-side code and jQuery.
- Host and Deploy an ASP.NET Core MVC application on IIS.
- Host and Deploy an ASP.NET Core MVC application on Microsoft Azure.
- Be able to utilize services offered by Microsoft Azure to improve the capabilities of your web applications.

production software. You need to understand how to run tests on small components of your web application to ensure that they function as expected befor

Additional Information:

This course will be delivered with digital courseware. In order to have the best learning experience you are asked to bring your own second screen to view the courseware. A second screen includes: tablets and laptops.

Further Information:

For More information, or to book your course, please call us on 0800/84.009

info@globalknowledge.be

www.globalknowledge.com/en-be/