Microsoft Partner

Microsoft®
.NET

CL_55340

Developing ASP.NET Core Web Applications

www.ked.com.mx

Por favor no imprimas este documento si no es necesario.

About this course.

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

Length.

5 Days.

Audience profile.

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.

Prerequisites.

Before attending this course, students must have:

- Experience with Visual Studio.
- Experience with C# programming, and concepts such as Lambda expressions, LINQ, and anonymous types.
- Experience in using the .NET.
- Experience with HTML, CSS and JavaScript.
- Experience with querying and manipulating data with ADO. NET.
- Knowledge of XML and JSON data structures.

At course completion.

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 applications.
- 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 application that display and edit data and interact with Models and Controllers.
- Connect an ASP.NET Core application to a database using Entity Framework Core.
- Implement a consistent look and feel across an entire web application.
- Write JavaScript code that runs on the client-side and utilizes the jQuery script library to optimize the responsiveness of an web application.
- Add client side packages and configure Task Runners.
- Run unit tests and debugging tools against a web application in Visual Studio 2022.
- Write an application that authenticates and authorizes users to access content securely using Identity.
- Build an 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 web application from a development computer to a web server.

Exam.

None.

Course outline.

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.

- Introducing of Microsoft Web Technologies.
- Getting Started with Razor Pages in ASP.NET Core.
- Introducing ASP.NET Core MVC.

Labs: Exploring ASP.NET Core.

- Exploring a Razor Pages Application.
- Exploring a Web API Application.
- Exploring an Application.

Labs: Installing and Configuring Windows 7.

- Migrating Settings by using Windows Easy Transfer.
- Configuring a Reference Image of Windows 7.
- Configuring a Reference Image.

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 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 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 Views.

Labs: 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.

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

ASP.NET Core is a framework that allows us to build many kinds of applications. In this module 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. This allows the application to become more flexible and to reduce potential points of failure whenever you change a service.

- Configuring Middlewares.
- Configuring Services.

Labs: 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:

- 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.

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 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. The ASP.NET Core MVC framework includes a flexible 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 or after every action in your web application, on every action in a controller, or on other combinations of controller actions.

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

Labs: 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:

- 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.

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

Labs: Developing Views.

- Adding Views to an MVC Application.
- Adding a partial view.
- Adding a view component.

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.

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

Labs: Developing Models.

- Adding a model.
- Working with Forms.
- Add Validation.

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.

- Introduction to Entity Framework Core.
- Working with Entity Framework Core.
- Use Entity Framework Core to connect to Microsoft SQL Server.

Labs: 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:

- 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.

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.

- Using Layouts.
- Using CSS and JavaScript.
- Using JavaScript Libraries.

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

- Applying a layout and link views to it.
- Using CSS.
- Using JavaScript Libraries.

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 the JavaScript Libraries in your web application.

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 are going to 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 interactive client-side web UI with .NET

- Applying Styles and Responsive Design.
- Using Task Runners.
- Looking at ASP.NET Core Blazor.

Labs: Client-Side Development.

- Use gulp to run tasks.
- Styling using Sass.
- Using Bootstrap.

After completing this module, students will be able to:

- 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.

Module 10: Testing and Troubleshooting.

The process of software development inevitably results in coding errors or bugs that result in exceptions, unexpected behaviour, or incorrect results. To improve the quality of your web application and provide a good user experience, you must identify bugs from any source and eliminate them. In traditional software development, testers perform most of the testing at the end of a development project. However, in recent years it has become widely accepted that testing throughout the project life cycle improves code quality and greatly reduces the quantity of bugs in production software. You need to understand how to run tests on individual components to ensure that they function as expected before you assemble them into a complete web application. It is also important that you know how to handle exceptions when they occur and handle them correctly to provide appropriate user feedback, without leaking information about the application structure. Finally, by using logging throughout the application, you can monitor user activities that might lead to unexpected issues and troubleshoot production problems by tracing flows through the application.

- Testing ASP.NET Core Applications.
- Implementing an Exception Handling Strategy.
- Logging ASP.NET Core Applications.

Labs: 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 module, students will be able to:

- 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.

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.

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

Labs: Managing Security.

- Use Identity.
- Add Authorization.
- Avoid the Cross-Site Request Forgery Attack.

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.
- Utilize several different authorization approaches.
- Know how security exploits work and how to better defend against them.

Module 12: 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.

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

Labs: Performance and Communication.

- Implementing a Caching Strategy.
- Managing state.
- 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 13: 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.

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

Labs: 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:

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

Module 14: Hosting and Deployment.

To set up an ASP.NET Core application for a production environment, you will need to perform a number of steps including compilation and preparation of assets, and then deploy it to a dedicated server or service that will host the application. Usually this will need to be accessible from the public Internet.

There are many different technologies that can be used to host your application and you should choose one that is appropriate for your requirements. For example, you could choose to provision a web server on your own hardware and infrastructure, using Microsoft Windows Server and Internet Information Services (IIS), or even use open-source operating systems and web server applications such as NGINX or Apache web server. Alternatively, you could deploy your application to a cloud service such as Microsoft Azure, and pay for and scale the service according to requirements. In this module we will explore these options and in particular the capabilities of Microsoft Azure.

- Hosting and Deploying On-premises.
- Deploying to Microsoft Azure.
- Looking at Microsoft Azure Fundamentals.

Labs: 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:

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