



# Module 1

## Exploring ASP.NET Core



## Module Overview

- Overview of Microsoft Web Technologies
- Overview of ASP.NET 4.x
- Introduction to ASP.NET Core





## Lesson 1: Overview of Microsoft Web Technologies

- Introduction to Microsoft Web Technologies
- Overview of ASP.NET
- Client-Side Web Technologies
- Hosting Technologies



## Introduction to Microsoft Web Technologies

### Develop

- Visual Studio
- Visual Studio Code

### Host

- Kestrel
- IIS
- Nginx
- Microsoft Azure

### Execute

#### Server-Side

- ASP.NET Core

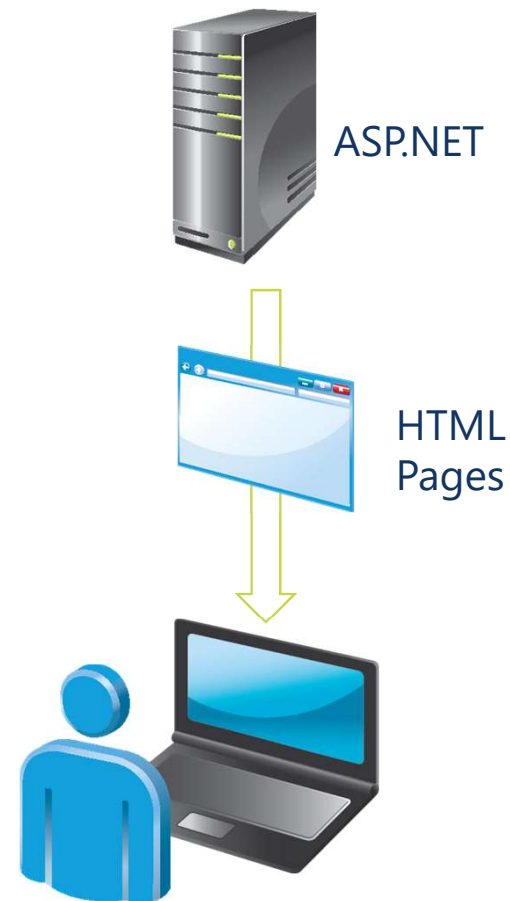
#### Client-Side

- JavaScript
- jQuery
- Angular
- React
- WebAssembly

## Overview of ASP.NET

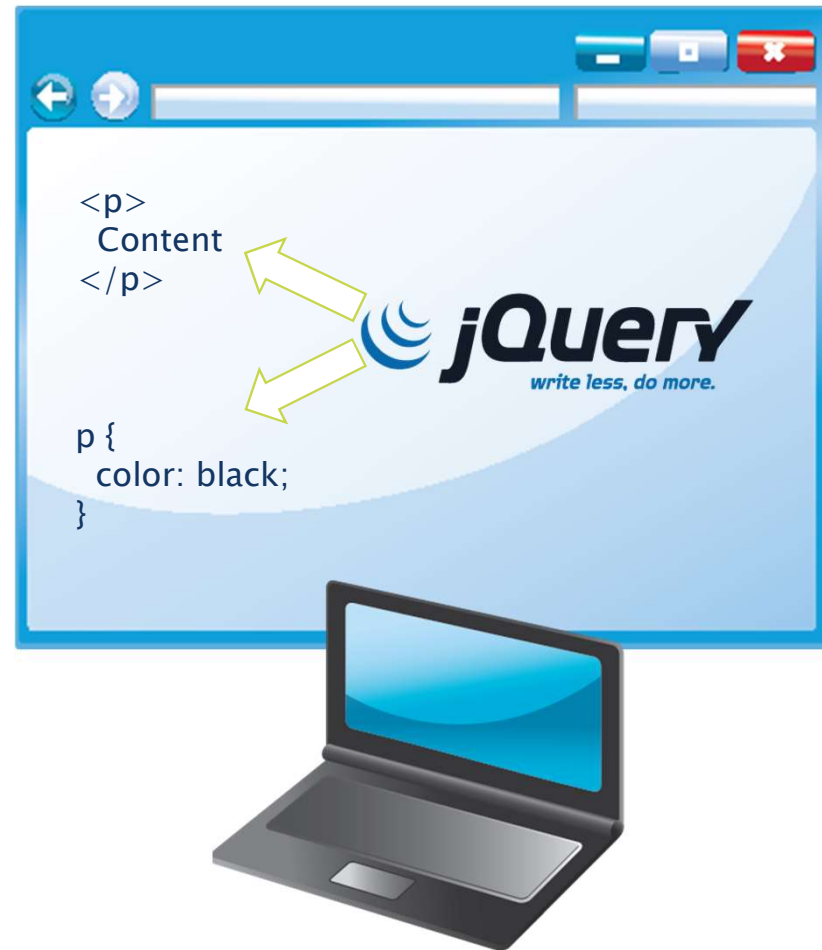
### Programming Models:

- ASP.NET 4.x:
  - Web Pages
  - Web Forms
- ASP.NET 4.x and ASP.NET Core:
  - MVC
  - Web API
- ASP.NET Core:
  - Razor Pages
  - Blazor



## Client-Side Web Technologies

- JavaScript
- jQuery
- AJAX
- Angular
- React
- WebAssembly
- And more



## Hosting Technologies

- Kestrel
  - Self Hosted
- IIS
  - Features
  - Scaling
  - Perimeter Networks
- IIS Express
- Nginx
  - Linux
- Other Web Servers



## Microsoft Azure

- Cloud computing provides scalability, flexibility, security, and reliability
- The Microsoft Azure platform includes:
  - Web Apps
  - Databases (Azure SQL Database, Cosmos DB)
  - Virtual Machines
  - Mobile Apps
  - Media Services







## Lesson 2: Overview of ASP.NET Core

- Server Side Rendered Solutions
- Client Side Rendered Solutions
- Hybrid Solutions






## Benefits and costs of server and client rendered UI

- Three general approaches to building modern web UI with ASP.NET Core:
  - Apps that render UI from the server.
  - Apps that render UI on the client in the browser.
  - Hybrid apps that take advantage of both server and client UI rendering approaches. For example, most of the web UI is rendered on the server, and client rendered components are added as needed.
- There are benefits and drawbacks to consider when rendering UI on the server or on the client.






## Server rendered UI

- HTML and CSS rendered on the server dynamically
    - Minimal client requirements
    - Great for low-end devices and low-bandwidth connections.
    - Allows for a broad range of browser versions at the client.
    - Quick initial page load times.
    - Minimal to no JavaScript to pull to the client.
    - Flexibility of access to protected server resources:
      - Database access.
      - Access to secrets, such as values for API calls to Azure storage.
      - Static site analysis advantages, such as search engine optimization.
  - Drawbacks:
    - Cost of compute and memory concentrated on the server
    - User interactions require a round trip to the server to generate UI updates.
- 



## Client rendered UI

- UI rendered on the client, updating browser DOM as necessary
    - Nearly instant rich interactivity without requiring a round trip to the server.
    - Incremental updates
    - Can be designed to run in a disconnected mode.
    - Reduced server load and cost.
    - Takes advantage of the capabilities of the user's device.
  - Drawbacks:
    - Code for the logic has to be downloaded and executed on the client, adding to the initial load time.
    - Client requirements may exclude users who have low-end devices, older browser versions, or low-bandwidth connections.
- 



## Hybrid

- Components Prerendered on the server
- UI updated incrementally on the client
  - Increased perceived responsiveness
  - Better SEO





## ASP.NET Core web solutions

- ASP.NET Core web UI server rendered solutions
  - ASP.NET Core Razor Pages
  - ASP.NET Core MVC
- ASP.NET Core web UI client rendered solutions
  - Blazor
  - Single Page Applications with Javascript frameworks
- ASP.NET Core web UI hybrid rendered solutions
  - Razor Pages or MVC with Razor Components





## ASP.NET Core Razor Pages

- Page-based model.
- UI and business logic concerns kept separate but within the page.
- Recommended way to create new page-based or form-based apps for developers new to ASP.NET Core.
- Easier starting point than ASP.NET Core MVC.
- Testable and scales to large apps.
- View specific logic and view models can be kept together in their own namespace and directory.
- Groups of related pages can be kept in their own namespace and directory.





## ASP.NET Core MVC


- Model-View-Controller (MVC) architectural pattern.
  - User requests are routed to a controller.
  - Controller works with the model to perform user actions or retrieve results of queries.
  - Controller chooses view to display and provides it with model data.
- Based on a scalable and mature model for building large web apps.
- Clear separation of concerns for maximum flexibility.
- The Model-View-Controller separation of responsibilities ensures that the business model can evolve without being tightly coupled to low-level implementation details.







## Blazor

- Composed of Razor components:
    - Reusable, shareable web UI implemented using C#, HTML, and CSS.
    - Both client and server code written in C#, allowing shared code and libraries.
    - Rendered or prerendered from views and pages.
    - Accelerate app development.
    - Reduce build pipeline complexity.
    - Simplify maintenance.
    - Leverage the existing .NET ecosystem of .NET libraries.
    - Developers understand and work on both client-side and server-side code.
    - UI components from top component vendors.
    - Work with all modern web browsers, including mobile browsers.
    - Open web standards without plug-ins or code transpilation.
- 

## ASP.NET Core Single Page Application with JavaScript

- ASP.NET Core provides project templates for Angular and React, and can be used with other JavaScript frameworks as well.
  - The JavaScript runtime environment is already provided with the browser.
  - Large community and mature ecosystem.
  - Build client-side logic for ASP.NET Core apps using popular JS frameworks, like Angular and React.
- Downsides:
  - More coding languages, frameworks, and tools required.
  - Difficult to share code so some logic may be duplicated.



## Hybrid: ASP.NET Core plus Blazor

- MVC and Blazor are both part of the ASP.NET Core framework
- Designed to be used together
- Razor components can be integrated into Razor Pages and MVC apps in a hosted Blazor WebAssembly or Server solution
- When the page or view is rendered, components can be prerendered at the same time
  - Executes Razor components on the server and renders them into a page or view
  - Improves the perceived load time of the app while setting up interactivity
  - Add islands of interactivity to existing views (pages) with the Component Tag Helper





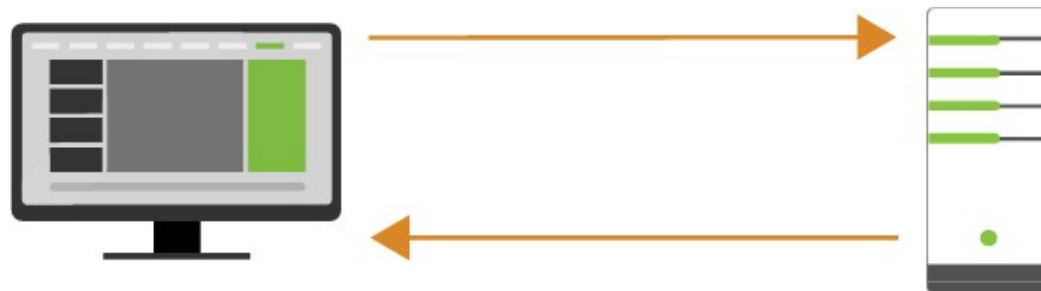
## Shared ASP.NET Features

- Application Startup
- Host
- Middleware
- Dependency Injection
- Configuration
- Logging
- Authentication
- Authorization
- State Management
- Caching



## Overview of Web API

- Helps creating RESTful APIs
- Enables external systems to use your application's business logic
- Accessible to various HTTP clients
- Helps to obtain data in different formats such as JSON and XML
- It supports create, read, update and delete (CRUD) actions
- Ideal for mobile application integration





## Lesson 3: Introduction to ASP.NET Core

- Introduction to ASP.NET Core
- Discussion: Choose between ASP.NET 4.x and ASP.NET Core
- Choose between .NET Core and .NET Framework
- Models, Views, and Controllers
- Demonstration: How to Explore an ASP.NET Core Application





## Razor Pages

- Alternative to the MVC programming model
- Starts with the **@page** directive

```
@page
```

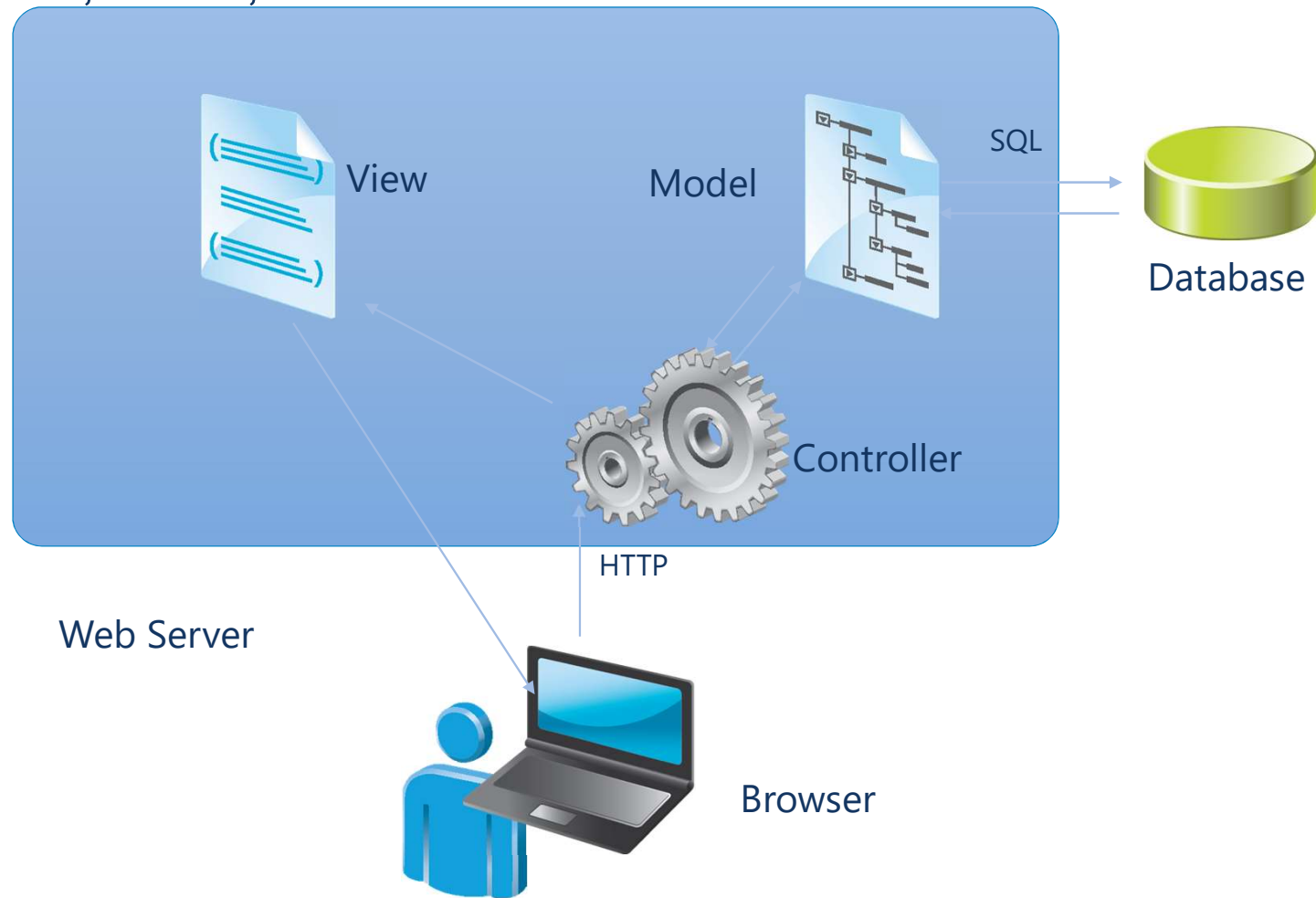
```
@model HomePageModel
```

```
<h1>@Model.Title</h1>
```

```
<h2>@Model.Description</li>
```



## Models, Views, and Controllers







## Demonstration: How to Explore an ASP.NET Core Application

In this demonstration, you will:

1. Examine an ASP.NET Core application that renders the default home page
2. Examine the Page UI code
3. Examine the Page handlers code
4. Examine the Model code
5. Examine the html rendered as a result of models, handlers, and page UI working together





## Lab: Exploring ASP.NET Core

- Exercise 1: Exploring a Razor Pages Application

Estimated Time: 30 minutes





## Lab Scenario

You are working as a junior developer at Adventure Works. You have been asked by a senior developer to investigate the possibility of creating a web-based photo sharing application for your organization's customers, similar to one that the senior developer has seen on the Internet. Such an application will promote a community of cyclists who use Adventure Works equipment, and the community members will be able to share their experiences. This initiative is intended to increase the popularity of Adventure Works Cycles, and thereby to increase sales. You have been asked to begin the planning of the application by examining an existing photo sharing application and evaluating its functionality. You have also been asked to examine programming models available to ASP.NET Core developers.





## Module Review and Takeaways

- Review Questions
- Best Practices
- Common Issues and Troubleshooting Tips



