# Module 9

Integrating
Blazor Components
in ASP.NET Core

## Module Overview

- Introduction to Blazor
- Blazor Components
- Blazor Integration and Prerendering

## Lesson 1: Introduction to Blazor

- What Is Blazor?
- Blazor Hosting Models

## What is Blazor?

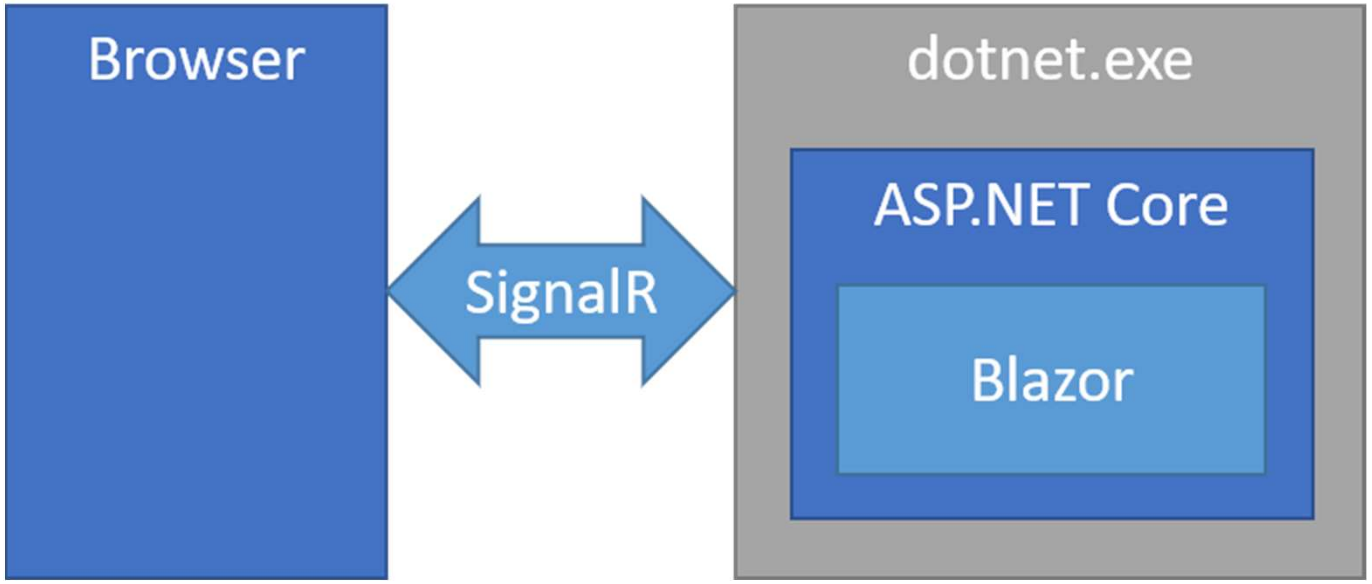
- Feature of ASP.NET
- Framework to  build interactive web UIs using C# instead of JavaScript
- Composed of reusable web UI components implemented using C#, HTML, and CSS
- .NET running on WebAssembly

## Blazor Hosting Model

- Blazor Server
- Blazor WebAssembly

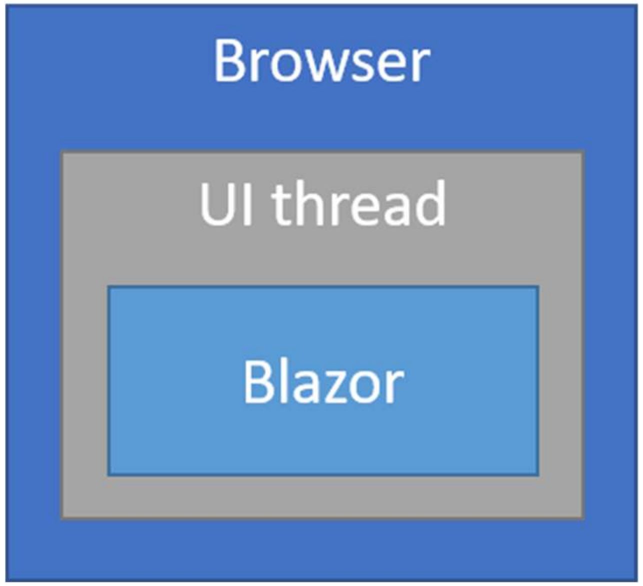
## Blazor Server

- App executed on the server from within an ASP.NET Core app
- UI updates, event handling, and JavaScript calls handled over SignalR connection



## Blazor WebAssembly

- App runs client-side in browser on WebAssembly-based .NET runtime
- The Blazor app, its dependencies, and.NET runtime downloaded to browser
- App executed directly on browser UI thread
- UI updates and event handling occur within same process
- Standalone: app created for deployment without a backend
- Hosted: app is created for deployment with a backend app to serve its files



## Lesson 2: Blazor Components

- Blazor Components Introduction
- Names
- Markup
- Namespaces
- Lifecycle
- Component Parameters
- Event Handling
- Data Binding
- Forms And Validation

## Blazor Components Introduction

- Blazor apps are based on **components**
- A component in Blazor is an element of UI, such as a page, dialog, or data entry form
- Components are .NET C# classes built into .NET assemblies that:
  - Define flexible UI rendering logic
  - Handle user events
  - Can be nested and reused
  - Can be shared and distributed as Razor class libraries or NuGet packages
- The component class is usually written in the form of a Razor markup page with a **`.razor`** file extension
- Components in Blazor are formally referred to as Razor components.
- Unlike Razor Pages and MVC, which are built around a request/response model, components are used specifically for client-side UI logic and composition

## Names

- A component's name **must** start with an uppercase character:
  - `ProductDetail.razor` is valid
  - `productDetail.razor` is invalid
- Common Blazor naming conventions use Pascal case (upper camel case)
  - No spaces
  - No punctuation
  - First letter of each word capitalized, including the first word

## Markup

- UI defined using Razor syntax
- HTML markup and C# rendering logic converted into a component class when compiled
- Members of component class defined in one or more **@code** blocks
  - –Property and field initializers
  - –Parameter values from arguments passed by parent components and route parameters
  - –Methods for user event handling, lifecycle events, and custom component logic

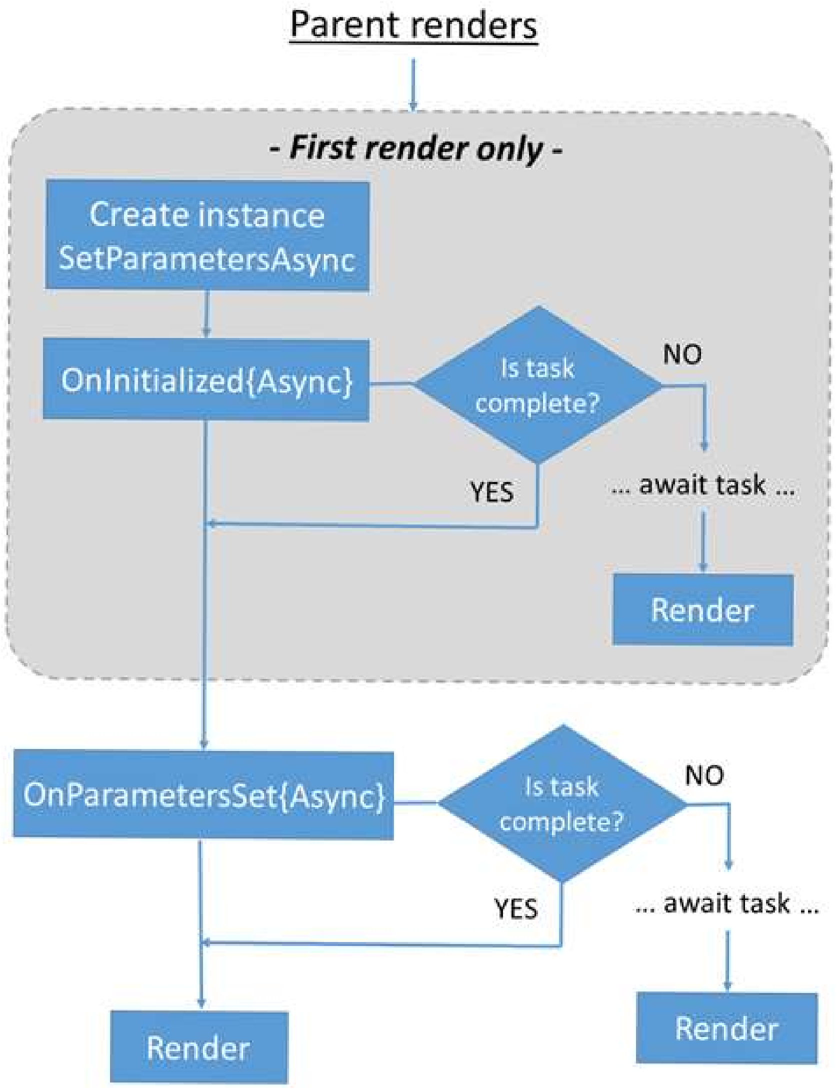
```
<h1 style="font-style:@headingFontStyle">@headingText</h1>
@code {
    private string headingFontStyle = "italic";
    private string headingText = "Put on your new Blazor!";
}
```

## Namespaces

- Derived from the app's root namespace and the component's location (folder) within the app.
- If the app's root namespace is **BlazorSample** and the **Counter** component resides in the **Pages** folder:
  - The **Counter** component's namespace is **BlazorSample.Pages**.
  - The fully qualified type name of the component is **BlazorSample.Pages.Counter**.
- Add an **@using** directive to the parent component or to the app's **_Imports.razor** file.
- Components can also be referenced using their **fully qualified names**, which doesn't require an **@using** directive.

## Lifecycle

- The Razor component processes Razor component lifecycle events in a set of synchronous and asynchronous lifecycle methods.
- The lifecycle methods can be overridden to perform additional operations in components during component initialization and rendering.



## Component Parameters

- Component parameters pass data to components and are defined using public C# properties on the component class with the `[Parameter]` attribute.

```
<div class="card w-25" style="margin-bottom:15px">
   <div class="card-header font-weight-bold">@Title</div>
   <div class="card-body" style="font-style:@Body.Style"> @Body.Text </div>
</div>
@code {
   [Parameter]
   public string Title { get; set; } = "Set By Child";
   [Parameter]
   public PanelBody Body { get; set; } = new() {
      Text = "Set by child.",
      Style = "normal"
   };
}
```

## Event Handling

- Specify delegate event handlers in Razor component markup with **`@on{DOM EVENT}="{DELEGATE}"`**
  - The **`{DOM EVENT}`** placeholder is a Document Object Model (DOM) event (for example, **`click`**)
  - The **`{DELEGATE}`** placeholder is the C# delegate event handler
- For event handling
  - Asynchronous delegate event handlers that return a **`Task`** are supported
  - Delegate event handlers automatically trigger a UI render
  - Exceptions are logged

```
<h1>@currentHeading</h1> <button @onclick="UpdateHeading"> Update heading </button>
@code {
 private string currentHeading = "Initial heading";
 private void UpdateHeading() { currentHeading = $“New Heading"; }
}
```

## Data Binding

Razor components provide data binding features with the **`@bind`** Razor directive attribute with a field, property, or Razor expression value

```
<p> <input @bind="InputValue" /> @InputValue </p>
@code {
   private string? InputValue { get; set; }
}
```

Component parameters permit binding properties of a parent component with
**`@bind-{PROPERTY}`**
where the **`{PROPERTY}`** placeholder is the property to bind

## Forms And Validation

A form is defined using the Blazor framework's **`EditForm`** component

```
<EditForm Model="@exampleModel" OnValidSubmit="@HandleValidSubmit">
    <DataAnnotationsValidator />
    <ValidationSummary />
    <InputText id="name" @bind-Value="exampleModel.Name" />
    <button type="submit">Submit</button>
</EditForm>
@code {
    private ExampleModel exampleModel = new();
    private void HandleValidSubmit() {
        // Process the valid form
    }
}
```

## Lesson 3: Component Integration

- Introduction
- Configuration
- Component Tag Helper

## Introduction

- Razor components can be integrated into Razor Pages and MVC apps in a hosted **Blazor WebAssembly** solution or in a **Blazor Server App**.
- When the page or view is rendered, components can be prerendered at the same time.

## Configuration (1/2)

- Host the Blazor WebAssembly app in an ASP.NET Core app
- In the Blazor WebAssembly client application
  - Delete the `wwwroot/index.html` and `wwwroot.favicon.ico` files
  - Delete the following lines in `Program.cs`:

```
builder.RootComponents.Add<App>("#app");
builder.RootComponents.Add<HeadOutlet>("head::after");
```

## Configuration (2/2)

- In the Hosting ASP.NET Web application Server Side
  - Add the Package `Microsoft.AspNetCore.Components.WebAssembly.Server`
  - Add a Project Reference to the Blazor WebAssembly project.
  - In `program.cs`, between `app.UseHttpsRedirection();` and `app.UseStaticFiles();` add the following:

```
app.UseBlazorFrameworkFiles();
```

  - In the `Pages/Shared/_Layout.cshtml` file, add the following (where `{BLAZOR WEBASSEMBLY NAME}` is the name of the Blazor WebAssembly Project):

```
<base href="~/" />
<link rel="stylesheet" href="~/css/app.css" />
<link rel="stylesheet" href="~/{BLAZOR WEBASSEMBLY NAME}.styles.css"
asp-append-version="true" />
```

## Component Tag Helper

- The **`Component`** Tag Helper supports two render modes for rendering a component from a Blazor WebAssembly app in a page or view:
  - –WebAssembly
  - –WebAssemblyPrerendered
- To make the component interactive, include the Blazor WebAssembly script
- To avoid using the full namespace for the Blazor component with the Component Tag Helper, add an **`@using`** directive for the client project's namespace.

```
@page
@using BlazorHosted.Client.Pages
<component type="typeof(Counter)" render-mode="WebAssemblyPrerendered" />
@section Scripts {
   <script src="_framework/blazor.webassembly.js"></script>
}
```

## Module Review and Takeaways

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