Tutorials Team

About Blazor

- Part of .Net Core 3
- UI with .Net
- Alternative of JS
- Leverage on WebAssembly

Course Outline

- Introduction to Blazor
- Different Hosting Models
- Creating Blazor component
- Use Parameters
- Data Bindings & Events
- Use RenderFragment
- Cascade Parameters
- Routing & URL Helper
- Create & Validate Forms
- Dependency Injection
- JavaScript Interop
- Deploy Application

Prerequirement

- Basic concepts of C#
- Basic ASP.Net development
- Visual Studio

ASP.Net Blazor

For Absolute Beginner

- Software Engineer at Thomson Reuters.
- Ahmedabad, India
- Passion for learning and advocating for new technologies.
- Visit: MNilay.com

Nilay Mehta

What will include in this series?

Where are resource?

How to ask questions?

ASP.Net Blazor

For Absolute Beginner

What is Blazor?

.Net Platform

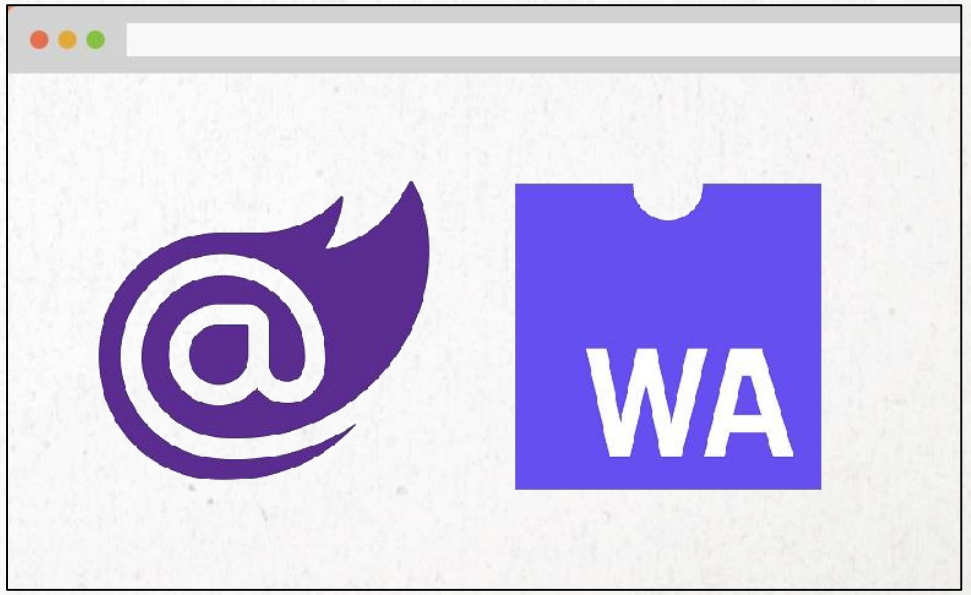
Desktop	Web	Cloud	Mobile	Gaming	IOT	AI
• Windows Form • WPF • UWP	• ASP.Net	• Azure	• Xamarin	• Unity	• ARM	• ML.Net • Big Data

About Blazor

- Part of .Net Core 3
- UI with .Net
- Alternative of JS
- Work in all Modern Browsers
- Leverage on WebAssembly
- Reusable Razor components
- Execute JavaScript

Current Web Application Architecture

After Blazor Web Application Architecture



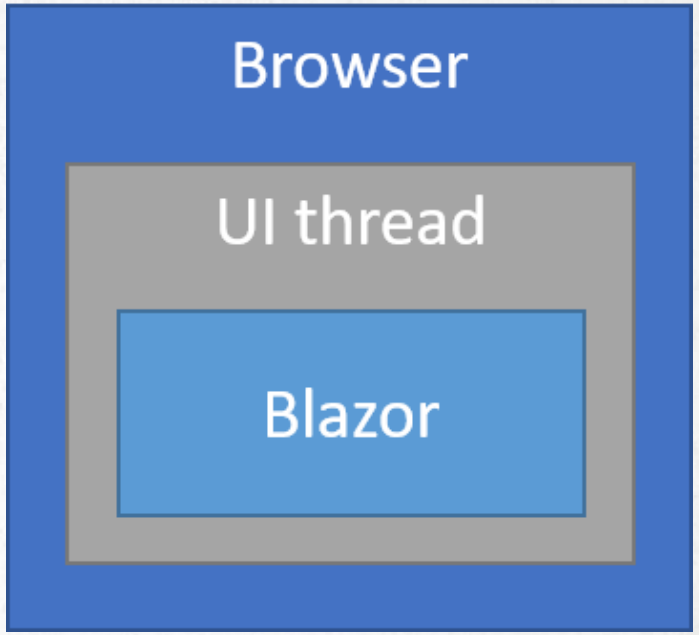
Web Assemblies

- WASM
- Gains in performance
- Open web standard
- Downloaded to the browser
- Written in C, C++, Rust, Java
- Caching

Hosting Models

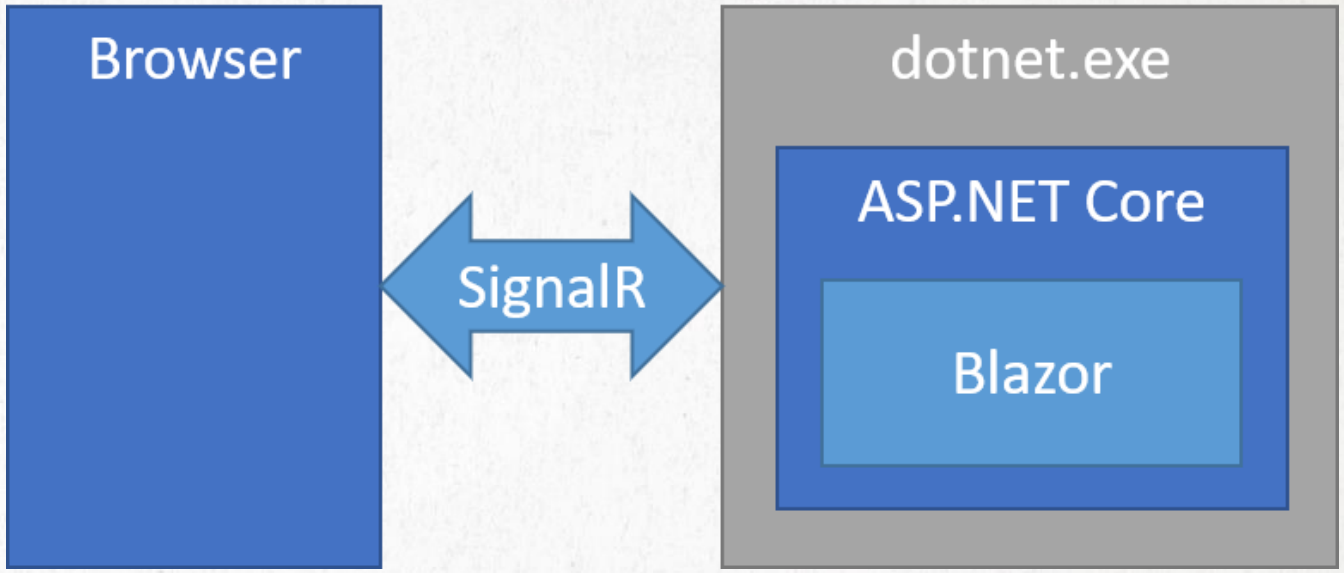
Client-side

- Running in Browser using WebAssembly
- Download .NET runtime in the browser
- Can access static resources
- No .NET server-side dependency
- Not required web server
- Download size



Server-side

- Use SignalR
- Requires server
- No offline access
- Less Download



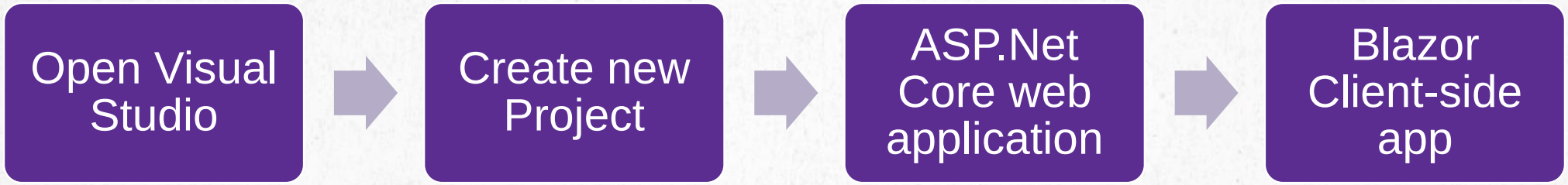
Setting up Environment

Setting up Environment

- Visual Studio Preview
- .Net core SDK 3
- Blazor Extension
- Blazor templates

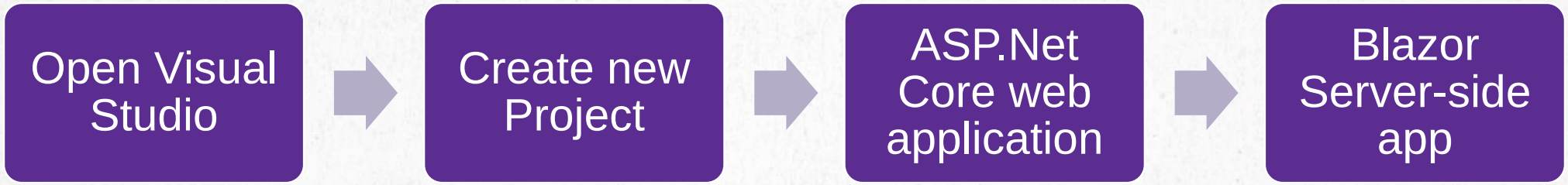
Create Client-side App

Create First Client-side App



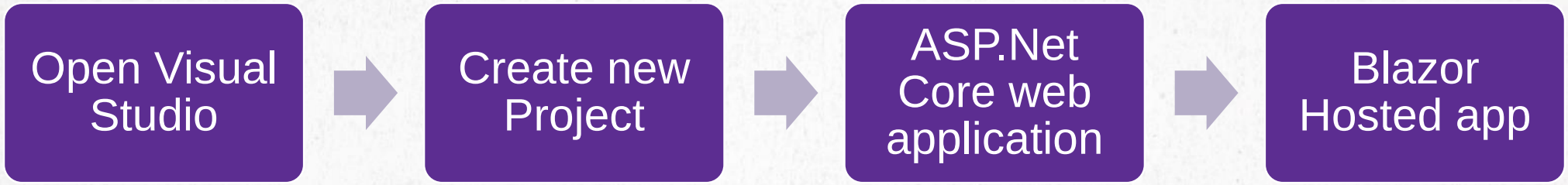
Create Server-side App

Create First Server-side App



Create Hosted App

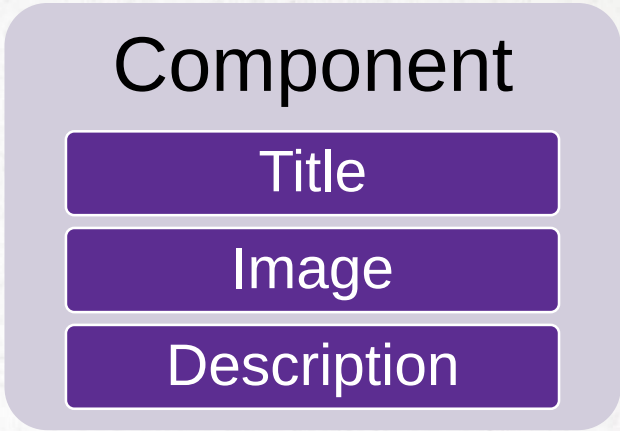
Create First Hosted App



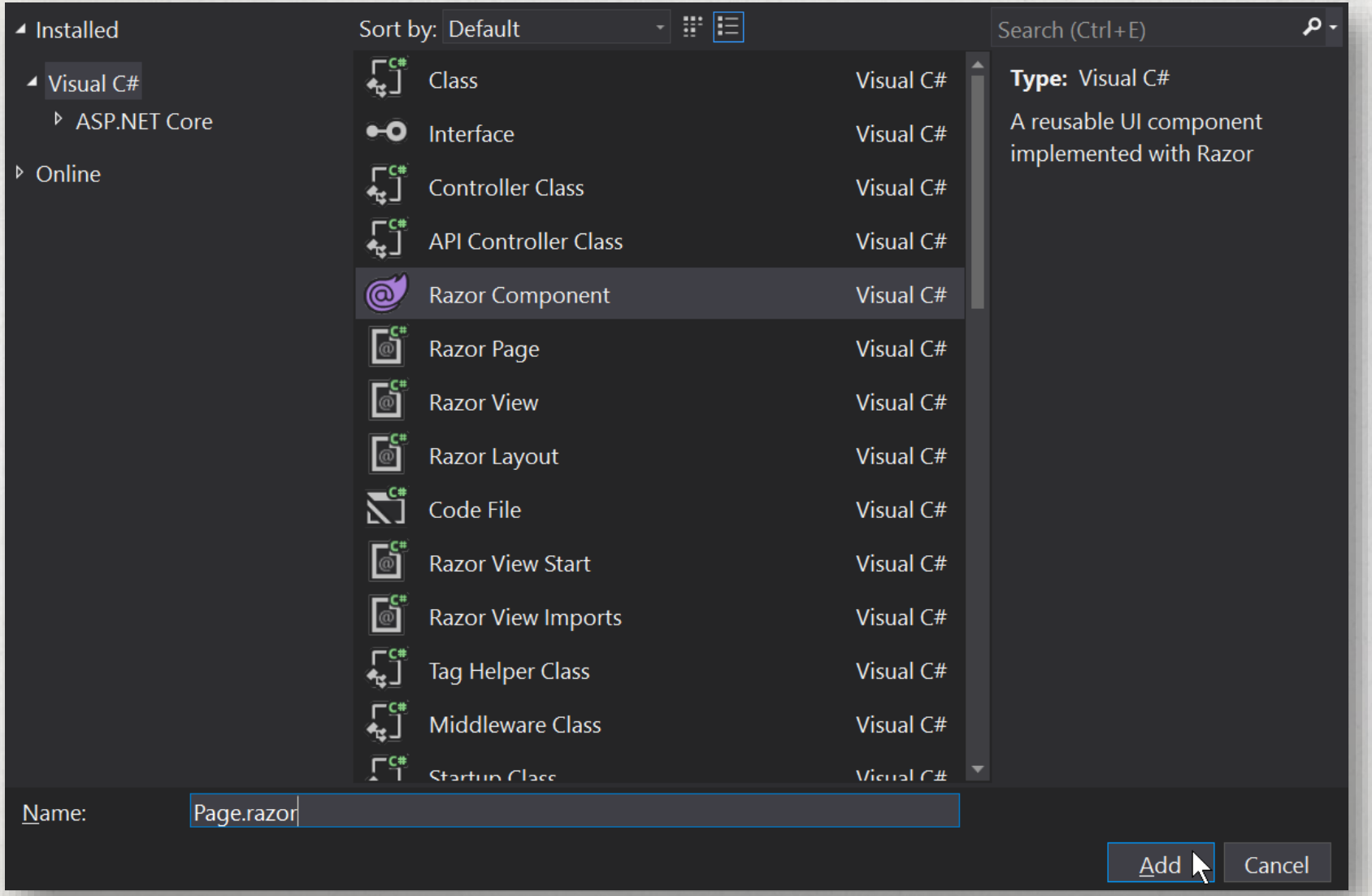
Create Blazor Component

Blazor Component

- Combination of C# and HTML
- HTML like syntax
- Converted to Class
- Inherited from ComponentBase



Create Component



Using Component

```
@page "/Session7/BlazorPage"

<PageTitle></PageTitle>
<PageImage></PageImage>
<PageDescription></PageDescription>

<div class="p-3 mb-2 bg-dark text-white">
    Page end
</div>

<PageTitle></PageTitle>
```

Using C# in Component

Using C# in Component

```
@page "/Session8/CodeComponentDemo"

<h1 class="display-4">@PageTitle</h1>
<img src="@PageImage" width="100%" alt="Blazor" />

@foreach (var description in PageDescriptions)
{
    <p class="text-justify">@description</p>
}

@code {

    private string PageTitle = "Blazor Page";
    private string PageImage = "https://i.imgur.com/8W9UlMJ.png";
    private List<string> PageDescriptions = new List<string> {
        "Lorem ipsum dolor sit amet, consectetur adipiscing elit. Sed pretium ante
        "Nullam vitae vestibulum risus, sed tempor erat. Nunc eget metus lacus. In
        "Pellentesque bibendum auctor diam, eu cursus metus tincidunt vel. Fusce s
        "In sollicitudin sagittis arcu at finibus. Nunc tincidunt, justo non pulvi
        "Morbi luctus imperdiet ante id lobortis. Cras quam nisi, consequat quis t
    };
}
```

Code Behind Component

Code Behind

```
4 references
public class CodeBehindClass : ComponentBase
{
    protected string PageTitle = "Blazor Page";
    protected string PageImage = "https://i.imgur.com/8W9UlMJ.png";…
    protected List<string> PageDescriptions = new List<string>
    {
        "Lorem ipsum dolor sit amet, consectetur adipiscing elit. Sed pret
        "Nullam vitae vestibulum risus, sed tempor erat. Nunc eget metus l
        "Pellentesque bibendum auctor diam, eu cursus metus tincidunt vel.
        "In sollicitudin sagittis arcu at finibus. Nunc tincidunt, justo n
        "Morbi luctus imperdiet ante id lobortis. Cras quam nisi, consequa
    };
}
```

Component

```
@page "/Session9/CodeBehindDemo"
@inherits CodeBehindClass

<h1 class="display-4">@PageTitle</h1>
<img src="https://i.imgur.com/8W9UlMJ.png" width="100%" alt="Blazor" />

@foreach (var description in PageDescriptions)
{
    <p class="text-justify">@description</p>
}
```

Using Parameters

Parameters

- Pass value from Outside
- non-public properties
- [Parameter] attribute

Component with Parameters

```
<div class="card">
    <div class="card-body">
        <blockquote class="blockquote">
            <p class="mb-0">@QuoteText</p>
            <footer class="blockquote-footer">@AutherName <cite title="@Category">@Category</cite
        </blockquote>
    </div>
</div>

@code{

    [Parameter]
    private string QuoteText { get; set; }

    [Parameter]
    private string AutherName { get; set; }

    [Parameter]
    private string Category { get; set; }

}
```

Set Values in Parameters

```
@page "/Session10/ParameterizedDemo"

<h3>ParameterizedDemo</h3>

<CustomBlockquote QuoteText="Put your heart, mind, and soul into even your smallest acts."
                  Category="Inspirational Quotes"
                  AutherName="Swami Sivananda">
</CustomBlockquote>
```

Component Life Cycle

Methods of ComponentBase

Method	Description
BuildRenderTree	It will build HTML which will render to browser
OnInit	It is executed when the component is ready
OnParametersSet	Call when parameters are modified in parent component
OnAfterRender	It will call after rendering of the component
ShouldRender	Specify that UI will be render again after modifying or not
StateHasChanged	It will update UI as per values exist in fields and parameters.

Life Cycle Methods

```
@page "/Session11/LiffeCycle"

<h3>LifeCycle</h3>

@code {

    protected override void OnInit()
    {
        Console.WriteLine("In OnInit");
    }

    protected override void OnParametersSet()
    {
        Console.WriteLine("In OnParametersSet");
    }

    protected override void OnAfterRender()
    {
        Console.WriteLine("In OnAfterRender");
    }
}
```

Data Binding

Data Binding

- Bind components and DOM
- One-way and two-way

Attribute	Description
Bind	Specify field on which binding will be perform on focus lost event
Bind-value	Specify field on which binding will be perform
Bind-value:event	Specify event on which binding changes will reflect
Binding-format	Specify format for DateTime.

One-way binding

```
@page "/Session12/OneWayBinding"

<h3>OneWayBinding</h3>

<div class="form-group">
    <label for="FullName">FullName:</label>
    <input type="text" class="form-control" id="FullName" value="@FullName" />
</div>

<div class="form-group">
    <label for="Age">Age:</label>
    <input type="number" class="form-control" id="Age" value="@Age" />
</div>

<p class="lead">FullName: @FullName<br />Age: @Age</p>

@code {
    private string FullName { get; set; } = "Bill Gates";
    private int Age { get; set; } = 63;
}
```

Two-way binding

```
@page "/Session12/TwoWayBinding"

<h3>TwoWayBinding</h3>

<div class="form-group">
    <label for="FullName">FullName:</label>
    <input type="text" class="form-control" id="FullName" @bind-value="@FullName" @bind-value:event="oninput" />
</div>

<div class="form-group">
    <label for="Age">Age:</label>
    <input type="number" class="form-control" id="Age" @bind="@Age" />
</div>

<div class="form-group">
    <label for="BirthDate">BirthDate:</label>
    <input type="text" class="form-control" id="BirthDate" @bind="@BirthDate" @bind:format="dd-MM-yy" />
</div>

<p class="lead">FullName: @FullName<br />Age: @Age<br />BirthDate: @BirthDate</p>

@code {
    private string FullName { get; set; } = "Bill Gates";
    private int Age { get; set; } = 63;
    private DateTime BirthDate { get; set; } = new DateTime(1955, 10, 28);
}
```

Event Handling

Event Handling

- Events like JavaScript
- onClick, onChange, onLoad, etc
- Set delegate in Attribute
- @on*
- No need to call StateHasChanged()

Event Argument

Event	Class
Clipboard	UIClipboardEventArgs
Drag	UIDragEventArgs
Error	UIErrorEventArgs
Focus	UIFocusEventArgs
<input> change	UIChangeEventArgs
Keyboard	UIKeyboardEventArgs
Mouse	UIMouseEventArgs
Mouse pointer	UIPointerEventArgs
Mouse wheel	UIWheelEventArgs
Progress	UIProgressEventArgs
Touch	UITouchEventArgs

Button's Click

```
@page "/Session13/EventHandleDemo"

<h3>EventHandleDemo</h3>

<button @onclick="@ToggleDisplayText">Toggle Text</button>
<p>@DisplayText</p>

@code {
    private string DisplayText { get; set; } = "Hello!";

    private void ToggleDisplayText(UIMouseEventArgs e)
    {
        DisplayText = (DisplayText.Equals("Hello!"))
            ? "Hello world!"
            : "Hello!";
    }
}
```

Handler in Parameter

Checkbox component

```
<div class="form-check">
    <input class="form-check-input" type="checkbox"
           value="@CheckboxValue" @onchange="@OnCheckChange" id="@CheckboxId" />
    <label class="form-check-label" for="@CheckboxId">
        @CheckboxText
    </label>
</div>

@code {
    [Parameter]
    private string CheckboxId { get; set; }

    [Parameter]
    private string CheckboxText { get; set; }

    [Parameter]
    private string CheckboxValue { get; set; }

    [Parameter]
    private EventCallback<UIChangeEventArgs> OnCheckChange { get; set; }
}
```

Parent component

```
@page   "/Session14/ParentControl"

<h3>ParentControl</h3>

<CheckBoxComponent CheckboxText="1st Checkbox" CheckboxValue="first"
                   CheckboxId="chkFirst" OnCheckChange="@FirstCheckBoxHandler">
</CheckBoxComponent>

<CheckBoxComponent CheckboxText="2st Checkbox" CheckboxValue="second"
                   CheckboxId="chkSecond" OnCheckChange="@SecondCheckBoxHandler">
</CheckBoxComponent>

@code {
    private void FirstCheckBoxHandler(UIChangeEventArgs e)
    {
        Console.WriteLine($"Is first checkbox selected: {e.Value}");
    }

    private void SecondCheckBoxHandler(UIChangeEventArgs e)
    {
        Console.WriteLine($"Second checkbox changed");
    }
}
```

Render Raw HTML

Using MarkupString

```
@page  "/Session15/RawHtmlDemo"

<h3>RawHtmlDemo</h3>

<b>Title: </b>@ItemTitle <br />
<b>Price: </b>@((MarkupString)@ItemPrice)

@code {
    private string ItemTitle { get; set; } = "Ink Pen";
    private string ItemPrice { get; set; } = "<big>$</big> <s>20</s> <b>10</b>";
}
```

Using Child Content

Using Child Content

```
@if (!string.IsNullOrEmpty(Title))
{
    <h3>@Title</h3>
}

<p>@ChildContent</p>

@code {

    [Parameter]
    private string Title { get; set; }

    [Parameter]
    private RenderFragment ChildContent { get; set; }

}
```

Pass Child Content

```
@page "/Session16/ParentPage"

<Paragraph Title="First Para">
    Some text for first paragraph.
</Paragraph>

<Paragraph>
    Second paragraph without title.
</Paragraph>
```

Using RenderFragment

Dropdown component

```
<div class="btn-group">
    <button type="button" class="btn btn-secondary dropdown-toggle" data-toggle="dropdown"
            aria-haspopup="true" aria-expanded="false">
        @ButtonText
    </button>
    <div class="dropdown-menu">
        @MenuItem1

        @if (@MenuItem2 != null)
        {
            <div class="dropdown-divider"></div>
            @MenuItem2
        }
    </div>
</div>

@code {
    [Parameter]
    private string ButtonText { get; set; }

    [Parameter]
    private RenderFragment MenuItem1 { get; set; }

    [Parameter]
    private RenderFragment MenuItem2 { get; set; }
}
```

Parent Content

```
@page "/Session17/ParentPage"

<h3>ParentPage</h3>

<DropdownComponent ButtonText="Manage">
    <MenuItem1>
        <a class="dropdown-item" href="#">Add</a>
        <a class="dropdown-item" href="#">Edit</a>
    </MenuItem1>
    <MenuItem2>
        <a class="dropdown-item" href="#">Delete All</a>
        <a class="dropdown-item" href="#">Export All</a>
    </MenuItem2>
</DropdownComponent>
```

Generic Template

Generic Template

```
@typeparam TList

<nav class="navbar navbar-expand-lg navbar-light bg-light">
    <a class="navbar-brand" href="#">@BrandTitle</a>
    <button class="navbar-toggler" type="button" data-toggle="collapse" data-target="#navbarNavAltMarkup"
            aria-controls="navbarNavAltMarkup" aria-expanded="false" aria-label="Toggle navigation">
        <span class="navbar-toggler-icon"></span>
    </button>
    <div class="collapse navbar-collapse" id="navbarNavAltMarkup">
        <div class="navbar-nav">
            @foreach (var item in NavbarItems)
            {
                @NavbarTemplate(item)
            }
        </div>
    </div>
</nav>

@code {
    [Parameter]
    private string BrandTitle { get; set; }

    [Parameter]
    private List<TList> NavbarItems { get; set; }

    [Parameter]
    private RenderFragment<TList> NavbarTemplate { get; set; }
}
```

Generic Template

```
@page "/Session18/NavBarDemo"

<h3>NavbarDemo</h3>

<NavbarComponent BrandTitle="Blazor Tutorials" NavbarItems="navItems" TList="LinkDetails">
    <NavbarTemplate Context="navItem">
        <a class="nav-item nav-link" href="@navItem.Link">@navItem.Title</a>
    </NavbarTemplate>
</NavbarComponent>

@code {

    private List<LinkDetails> navItems = new List<LinkDetails>
    {
        new LinkDetails {Title="Home", Link = "/Home"},
        new LinkDetails {Title="About Us", Link="/AboutUs"},
        new LinkDetails {Title="Contact Us", Link="/ContactUs"},
        new LinkDetails {Title="Feedback", Link="/Feedback"}
    };

}
```

Using RenderTreeBuilder

Build Razor component

```
@page "/Session19/TreeBuilderDemo"

<h3>TreeBuilderDemo</h3>

@PersonDetailsFragment

<button @onclick="@AddPersonDetails">Add Person</button>

@code {
    private RenderFragment PersonDetailsFragment { get; set; }

    private void AddPersonDetails()
    {
        PersonDetailsFragment = BuildPerson();
    }

    private RenderFragment BuildPerson() => builder =>
    {
        builder.OpenComponent(0, typeof(PersonDetails));
        builder.AddAttribute(1, "Name", "Nilay");
        builder.AddAttribute(2, "Mobile", "9876543210");
        builder.CloseComponent();
    };
}
```

Person Details

```
<p>
    <b>Name:</b> @Name<br />
    <b>Mobile:</b> @Mobile
</p>

@code {

    [Parameter]
    private string Name { get; set; }

    [Parameter]
    private string Mobile { get; set; }

}
```

Build HTML Element

```
[Route("/Session19/PersonDetails")]
0 references
public class PersonDetailsCodeBehind : ComponentBase
{
    [Parameter]
    1 reference
    private string Name { get; set; } = "Nilay";

    [Parameter]
    1 reference
    private string Mobile { get; set; } = "987654";

    61 references
    protected override void BuildRenderTree(RenderTreeBuilder builder)
    {
        base.BuildRenderTree(builder);

        builder.OpenElement(0, "p");
        builder.OpenElement(1, "b");
        builder.AddContent(2, "Name:");
        builder.CloseElement();
        builder.AddContent(3, Name);
        builder.OpenElement(4, "br");
        builder.CloseElement();
        builder.OpenElement(5, "b");
        builder.AddContent(6, "Mobile:");
        builder.CloseElement();
        builder.AddContent(7, Mobile);
        builder.CloseElement();
    }
}
```

ShouldRender & StateHasChanged

ShouldRender

- Flag for Update UI
- Default true

StateHasChanged

- Update UI for current component
- Called after any lifecycle method or events
- Use value of ShouldRender to decide if a UI re-render

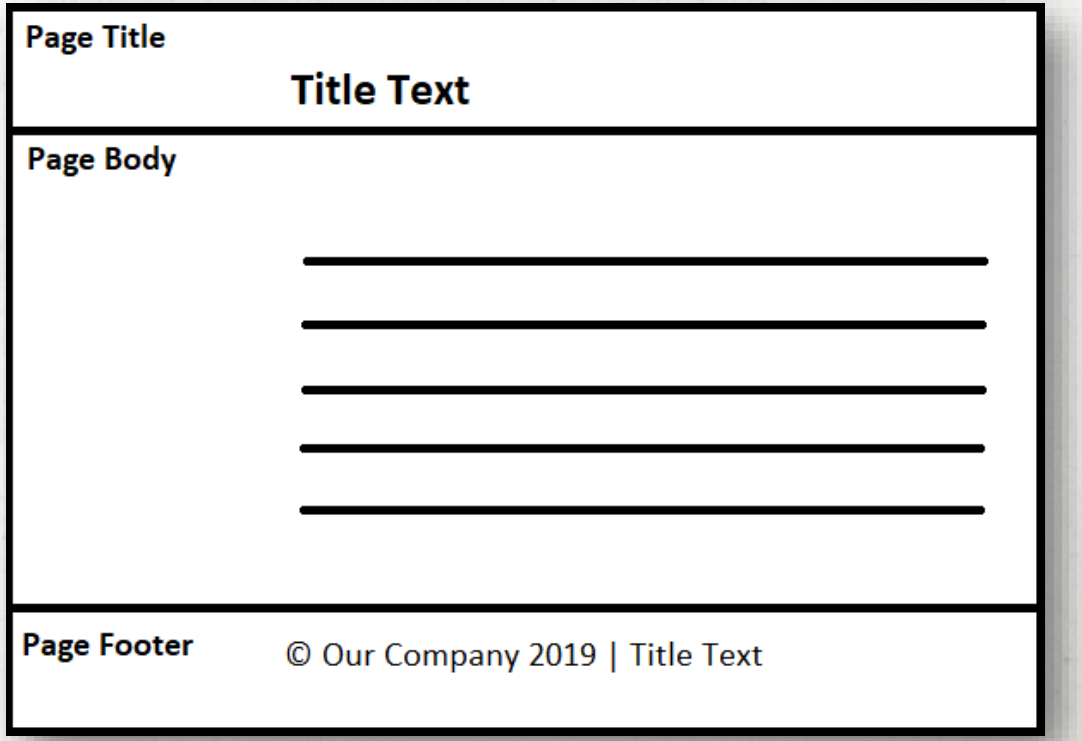
ShouldRender & StateHasChanged

```
<button @onclick="@UpdateCurrentTime">Update Time</button>
<button @onclick="@ToggleRendering">Toggle Rendering</button>
<button @onclick="@StartTimer">Start Timer</button>

<p>@CurrentTime.ToLongTimeString()</p>

@code {
    private bool shouldRenderUIChanges = true;
    private DateTime CurrentTime { get; set; } = DateTime.Now;

    private void UpdateCurrentTime(UIMouseEventArgs e)
    {
        CurrentTime = DateTime.Now;
    }

    private void ToggleRendering(UIMouseEventArgs e)
    {
        shouldRenderUIChanges = !shouldRenderUIChanges;
    }

    protected override bool ShouldRender()
    {
        return shouldRenderUIChanges;
    }

    private void StartTimer(UIMouseEventArgs e)
    {
        var timer = new Timer(new TimerCallback(state =>
        {
            CurrentTime = DateTime.Now;
            this.StateHasChanged();
        }), null, 1000, 1000);
    }
}
```

Using Cascading Values

Cascading Values & Parameters

- Pass a value to all of its descendants
- Pass any value



CasecadingValue

```
@page "/Session21/MainPage"

<CascadingValue Value="@PageTitle">
    <PageTitle></PageTitle>
    <PageBody></PageBody>
    <PageFooter></PageFooter>
</CascadingValue>

@code {
    public string PageTitle { get; set; } = "Hello Page";
}
```

Cascading Parameters

```
<h3>@TitleText</h3>
<hr />

@code{
    [CascadingParameter]
    public string TitleText { get; set; }
}
```

```
<hr />
<small>&copy; Our Company 2019 | @TitleText</small>

@code{
    [CascadingParameter]
    public string TitleText { get; set; }
}
```

Using Multiple Cascading Parameters

Main Page

```
@page "/Session22/MainPage"

<CascadingValue Value="@PageTitle" Name="PageTitle">
    <CascadingValue Value="@PublishedDate">
        <CascadingValue Value="@AuthorName" Name="AuthorName">
            <PageTitle></PageTitle>
            <PageBody></PageBody>
            <PageFooter></PageFooter>
        </CascadingValue>
    </CascadingValue>
</CascadingValue>

@code {
    public string PageTitle { get; set; } = "Hello Page";
    public DateTime PublishedDate { get; set; } = DateTime.Now;
    public string AuthorName { get; set; } = "Nilay";
}
```

Title

```
<h3>@TitleText</h3>
<small>By @AuthorName On @PublishedDate</small>
<hr />

@code{
    [CascadingParameter(Name = "PageTitle")]
    public string TitleText { get; set; }

    [CascadingParameter(Name = "AuthorName")]
    public string AuthorName { get; set; }

    [CascadingParameter]
    public DateTime PublishedDate { get; set; }
}
```

Footer

```
<hr />
<small>&copy; @AuthorName @PublishedDate.Year | @TitleText</small>

@code{
    [CascadingParameter(Name="PageTitle")]
    public string TitleText { get; set; }

    [CascadingParameter(Name="AuthorName")]
    public string AuthorName { get; set; }

    [CascadingParameter]
    public DateTime PublishedDate { get; set; }
}
```

Events in Cascade Value

Set Event

```
@page "/Session23/MainPage"

<CascadingValue Value="@PageTitle" Name="PageTitle">
    <CascadingValue Value="@PublishedDate">
        <CascadingValue Value="@AuthorName" Name="AuthorName">
            <PageTitle OnChangePublishedDate="@UpdatePublishedDate"></PageTitle>
            <PageBody></PageBody>
            <PageFooter></PageFooter>
        </CascadingValue>
    </CascadingValue>
</CascadingValue>

@code {
    public string PageTitle { get; set; } = "Hello Page";
    public DateTime PublishedDate { get; set; } = DateTime.Now;
    public string AuthorName { get; set; } = "Nilay";

    private void UpdatePublishedDate(DateTime updatedDate)
    {
        this.PublishedDate = updatedDate;
        this.StateHasChanged();
    }
}
```

Execute Action

```
<h3>@TitleText</h3>
<small>By @AuthorName On @PublishedDate</small>

<button @onclick="@UpdateValue">Update Value</button>
<hr />

@code{
    [CascadingParameter(Name = "PageTitle")]
    public string TitleText { get; set; }

    [CascadingParameter(Name = "AuthorName")]
    public string AuthorName { get; set; }

    [CascadingParameter]
    public DateTime PublishedDate { get; set; }

    [Parameter]
    Action<DateTime> OnChangePublishedDate { get; set; }

    private void UpdateValue()
    {
        this.OnChangePublishedDate?.Invoke(PublishedDate.AddYears(1));
    }
}
```

Using @key

@key Directive Attribute

- Minimize Rendering complexity
- Does not Re-render component
- Performance cost
- Value can be Model or Unique value
- Does not required compulsory.

Using @key

```
@page "/Session24/DetailsPage"

<h3>DetailsPage</h3>
<button class="btn btn-dark" @onclick="@AddNewPerson">Add new Person</button>

@foreach (Person person in people)
{
    <PersonDetails @key="@person" person="@person"></PersonDetails>
}

@code {
    List<Person> people = new List<Person>
    {
        new Person{ Id=1, FirstName = "Andie", LastName="Block"},
        new Person{ Id=2, FirstName="Allen", LastName="Alexander"},
        new Person{ Id=3, FirstName="James", LastName="Davis"},
        new Person{ Id= 4, FirstName="Stewart", LastName="Lawrence"}
    };

    private void AddNewPerson(UIMouseEventArgs args)
    {
        people.Insert(1, new Person { Id = 5, FirstName = "Mara", LastName = "Gibson" });
    }
}
```

Using @attribute

Using @attribute Directive

```
@attribute [Serializable]
@attribute [Obsolete("This component is Deprecated")]
@*@attribute [System.Runtime.InteropServices.DllImport("some.dll")]*@

<div id="@person.Id" class="card">
    <p><b>Id:</b> @person.Id</p>
    <p><b>FirstName:</b> @person.FirstName</p>
    <p><b>LastName:</b> @person.LastName</p>
</div>

@code {
    [Parameter]
    public Person person { get; set; }
}
```

Master Layout

Create Master Layout

```
@inherits LayoutComponentBase

<nav class="navbar navbar-expand-lg navbar-light bg-light">
    <a class="navbar-brand" href="#">Blazor Demo</a>
    <button class="navbar-toggler" type="button" data-toggle="collapse" data-target="#navbarNavAltMarkup"
            aria-controls="navbarNavAltMarkup" aria-expanded="false" aria-label="Toggle navigation">
        <span class="navbar-toggler-icon"></span>
    </button>
    <div class="collapse navbar-collapse" id="navbarNavAltMarkup">
        <div class="navbar-nav">
            <a class="nav-item nav-link" href="#">Home</a>
            <a class="nav-item nav-link" href="#">About</a>
            <a class="nav-item nav-link" href="#">Contact</a>
        </div>
    </div>
</nav>

@Body

<footer>End of Page</footer>
```

Using Layout

```
@layout MasterPage
@page "/Session26/AboutPage"

<h3>AboutPage</h3>
```

Routing

Page Directive

```
@page "/Session27/HelloPage"
@page "/HelloPage"

<h3>Hello</h3>

@code {

}
```

String Parameter

```
@page "/Session27/DisplayName/{Name}"

<h3>DisplayName</h3>
Your name is @Name

@code {
    [Parameter]
    public string Name { get; set; }
}
```

Integer Parameters

```
@page "/Session27/Sum/{Num1:int}"
@page "/Session27/Sum/{Num1:int}/{Num2:int}"

<h3>Sum</h3>
@Num1 + @Num2 = @Total

@code {
    [Parameter]
    public int Num1 { get; set; }

    [Parameter]
    public int Num2 { get; set; }

    public int Total
    {
        get
        {
            return Num1 + Num2;
        }
    }
}
```

URL Helpers

Using NavLink

```
@inherits LayoutComponentBase
@inject IUriHelper UriHelper

<nav class="navbar navbar-expand-lg navbar-light bg-light">
    <a class="navbar-brand" href="#">Blazor Demo</a>
    <button class="navbar-toggler" type="button" data-toggle="collapse" data-target="#navbarNavAltMarkup"
            aria-controls="navbarNavAltMarkup" aria-expanded="false" aria-label="Toggle navigation">
        <span class="navbar-toggler-icon"></span>
    </button>
    <div class="collapse navbar-collapse" id="navbarNavAltMarkup">
        <div class="navbar-nav">
            <NavLink class="nav-item nav-link" href="/Session28/HomePage">Home</NavLink>
            <NavLink class="nav-item nav-link" href="/Session28/AboutPage">About</NavLink>
            <NavLink class="nav-item nav-link" href="/Session28/ContactPage">Contact</NavLink>
        </div>
    </div>
</nav>

<br />
<button @onclick="@NavigateToHelloPage" class="btn btn-dark">Goto HelloPage</button>
<br />
```

Using UriHelper

```
@code{
    private void NavigateToHelloPage()
    {
        UriHelper.NavigateTo("HelloPage");
    }

    protected override void OnInit()
    {
        UriHelper.OnLocationChanged += OnChangingLocation;
    }

    private void OnChangingLocation(object sender, LocationChangedEventArgs args)
    {
        Console.WriteLine($"Change URL: {args.Location}");
    }
}
```

URI Helper

Member	Description
GetAbsoluteUri()	Gets the current absolute URI
GetBaseUri()	Gets Base URL specified in href of <base>
NavigateTo()	Navigates to the specified URI.
ToAbsoluteUri()	Converts a relative URI into an absolute one
ToBaseRelativePath()	Converts an absolute URI into one relative to the base URI prefix
OnLocationChanged	An event that fires when the navigation location has changed

Using EditForm

Create Form

```
<EditForm Model="@model" OnValidSubmit="@OnValidSubmit">

    <div class="form-group">
        <label for="FullName">Full Name</label>
        <InputText Id="FullName" @bind-Value="@model.FullName" class="form-control"></InputText>
    </div>

    <div class="form-group">
        <label for="DateOfBirth">Date of Birth</label>
        <InputDate Id="DateOfBirth" @bind-Value="@model.DateOfBirth" class="form-control"></InputDate>
    </div>

    <div class="form-group">
        <label for="Age">Age</label>
        <InputNumber Id="Age" @bind-Value="@model.Age" class="form-control"></InputNumber>
    </div>

    <div class="form-group">
        <label for="Country">Country</label>
        <InputSelect Id="Country" @bind-Value="@model.Country" class="form-control">
            <option value="">-- Select --</option>
            <option value="US">US</option>
            <option value="Canada">Canada</option>
            <option value="India">India</option>
        </InputSelect>
    </div>

    <button type="submit" class="btn btn-primary">Submit</button>

</EditForm>
```

Form Validation

Data Annotation

```
public class StudentFormModel
{
    [Required]
    [StringLength(20)]
    6 references
    public string FullName { get; set; }

    [Required]
    6 references
    public DateTime DateOfBirth { get; set; }

    [Required]
    [Range(5, 20)]
    5 references
    public int Age { get; set; }

    [Required]
    6 references
    public string Country { get; set; }

    1 reference
    public override string ToString()
    {
        return $"Hey {FullName}, You are from {Country} and your Date of Birth is {DateOfBirth.ToShortDateString()}";
    }
}
```

Validation Message & Summary

```
<EditForm Model="@model" OnValidSubmit="@OnValidSubmit" OnInvalidSubmit="@OnInvalidSubmit">
    <DataAnnotationsValidator />
    <ValidationSummary />

    <div class="form-group">
        <label for="FullName">Full Name</label>
        <InputText Id="FullName" @bind-Value="@model.FullName" class="form-control"></InputText>
        <ValidationMessage For="@(() => model.FullName)" />
    </div>

    <div class="form-group">
        <label for="DateOfBirth">Date of Birth</label>
        <InputDate Id="DateOfBirth" @bind-Value="@model.DateOfBirth" class="form-control"></InputDate>
        <ValidationMessage For="@(() => model.DateOfBirth)" />
    </div>

    @*..*@

    <button type="submit" class="btn btn-primary">Submit</button>

</EditForm>
```

Using @ref

@Ref usage

```
@page "/Session31/RefDemo"

<h3>RefDemo</h3>

<button class="btn btn-dark" @onclick="@ToggleContent">Toggle Content</button>

<MainContent @ref="@mainContent"></MainContent>

@code {

    MainContent mainContent;

    private void ToggleContent()
    {
        mainContent.ToggleContent();
    }
}
```

Dependency Injection

Dependency Injection

- Same as APS.Net MVC
- Accessing configured services

- Built-in Service DI
  - HttpClient, IJSRuntime, IUriHelper
- Custom Service DI

DI Scope

Lifetime	Description
Singleton	Single instance of a service class shared across many components.
Scoped	Service registration is scoped to the connection. Currently only supported by Server-side Hosting.
Transient	Each component obtains new instance of a service.

- Use @inject directive

@inject <class/interface-name> <instance-name>

Using @inject

```
@page "/Session32/GithubProfileForm"
@inject HttpClient httpClient

<h3>GithubProfileForm</h3>

<EditForm Model="@FormModel" OnValidSubmit="@OnSubmit">
    <div class="form-group">
        <label for="Username">Username</label>
        <InputText Id="Username" @bind-Value="@FormModel.Username" class="form-control"></InputText>
    </div>

    <button type="submit" class="btn btn-primary">Submit</button>
</EditForm>

<GithubProfileDetailsComponent @ref="DetailsComponent"></GithubProfileDetailsComponent>

@code {
    GithubProfileFormModel FormModel = new GithubProfileFormModel();
    GithubProfileDetailsComponent DetailsComponent;

    public async Task OnSubmit()
    {
        string response = await httpClient.GetStringAsync($"https://api.github.com/users/{FormModel.Username}");
        var ProfileDetailsModel = Newtonsoft.Json.JsonConvert.DeserializeObject<GithubProfileDetailsModel>(response);
        DetailsComponent.LoadData(ProfileDetailsModel);
    }
}
```

DI with Custom Service

Custom Service

```
6 references
public interface IGithubService
{
    3 references
    Task<GithubProfileDetailsModel> GetProfileDetails(string username);
}

2 references
public class GithubService : IGithubService
{
    private readonly HttpClient _client;

    0 references
    public GithubService(HttpClient client)
    {
        _client = client;
    }

    3 references
    public async Task<GithubProfileDetailsModel> GetProfileDetails(string username)
    {
        string response = await _client.GetStringAsync($"https://api.github.com/users/{username}");
        return Newtonsoft.Json.JsonConvert.DeserializeObject<GithubProfileDetailsModel>(response);
    }
}
```

Register DI

```
1 reference
public class Startup
{
    0 references
    public void ConfigureServices(IServiceCollection services)
    {
        services.AddSingleton<Pages.Session33.IGithubService,
                              Pages.Session33.GithubService>();
    }

    0 references
    public void Configure(IComponentsApplicationBuilder app)
    {
        app.AddComponent<App>("app");
    }
}
```

Inject Service

```
@page "/Session33/GithubProfileForm"
@inject IGithubService githubService

<h3>GithubProfileForm</h3>

<EditForm Model="@FormModel" OnValidSubmit="@OnSubmit">
    <div class="form-group">
        <label for="Username">Username</label>
        <InputText Id="Username" @bind-Value="@FormModel.Username" class="form-control"></InputText>
    </div>

    <button type="submit" class="btn btn-primary">Submit</button>
</EditForm>

<GithubProfileDetailsComponent @ref="DetailsComponent"></GithubProfileDetailsComponent>

@code {
    GithubProfileFormModel FormModel = new GithubProfileFormModel();
    GithubProfileDetailsComponent DetailsComponent;

    public async Task OnSubmit()
    {
        DetailsComponent.LoadData(await githubService.GetProfileDetails(FormModel.Username));
    }
}
```

Call JavaScript Function

JavaScript Interop

- Execute JavaScript from C#
- IJSRuntime
- Execute function defined under Window scope
- For void pass object as return type.
- InvokeAsync<return-type>(“Identifier-name”, [paras…])
- OnAfterRender

JavaScript Functions

```
<script type="text/javascript">

    window.getBrowserDetails = function () {
        return {
            codeName: navigator.appCodeName,
            browserName: navigator.appName,
            browserVersion: navigator.appVersion,
            cookiesEnabled: navigator.cookieEnabled,
            browserOnline: navigator.onLine,
            language: navigator.language,
            platform: navigator.platform,
            userAgent: navigator.userAgent,
        };
    };

    window.makeSum = function (n1, n2) {
        this.console.log(`Sum of ${n1} and ${n2} is ${(n1 + n2)}`);
    }

</script>
```

Call JavaScript Functions

```razor
@page "/Session34/BrowserDetailsPage"
@inject IJSRuntime jsRuntime

<h3>BrowserDetailsPage</h3>
<div>
    <button class="btn btn-dark" @onclick="@LoadBrowserDetails">Load Browser detials</button>
    <button class="btn btn-dark" @onclick="@MakeSumOf2Numbers">Make Sum of 2 numbers</button>
</div>

@if (browserDetails != null)
{
    <table class="table table-bordered table-striped">...</table>
}

@code {
    private BrowserDetailsModel browserDetails;

    protected async Task LoadBrowserDetails()
    {
        browserDetails = await jsRuntime.InvokeAsync<BrowserDetailsModel>("getBrowserDetails");
    }

    protected async Task MakeSumOf2Numbers()
    {
        await jsRuntime.InvokeAsync<object>("makeSum", 10, 20);
    }
}
```

Call C# from JavaScript

JavaScript Interop

- Execute C# from JavaScript
- DotNet
- invokeMethod or invokeMethodAsync
- [JSInvokable]
- Can call instance methods too

JavaScript Functions

```
window.loadDate = function () {
    var date = DotNet.invokeMethod('BlazorClientSide', 'LoadDate');
    console.log(date);
};

window.loadUserDataAsync = function (instance) {
    instance.invokeMethodAsync('LoadUserDataAsync')
        .then(data => this.console.log(data));
};
```

Class

```
5 references
public class UserData
{
    private List<string> _userData = new List<string>();

    2 references
    public UserData(List<string> userData)
    {
        _userData = userData;
    }

    [JSInvokable]
    0 references
    public async Task<List<string>> LoadUserDataAsync()
    {
        return await Task.FromResult(_userData);
    }
}
```

Component

```
@page "/Session35/JSRuntimeInterop"
@inject IJSRuntime jsRuntime

<h3>JSRuntimeInterop</h3>
<div>
    <button class="btn btn-dark" onclick="window.loadDate()">Load Date</button>
    <button class="btn btn-dark" @onclick="@LoadUserData">Load User Data</button>
</div>

@code {
    [JSInvokable]
    public static DateTime LoadDate()
    {
        return DateTime.Now;
    }

    private async void LoadUserData()
    {
        UserData data = new UserData(new List<string> { "Hello", "World!" });
        await jsRuntime.InvokeAsync<object>("loadUserDataAsync",
                                            DotNetObjectRef.Create(data));
    }
}
```

Debug Application

Decompile DLL

Deploy Application

Upgrade to Preview-9

Update Packages

```
<ItemGroup>
    <PackageReference Include="Microsoft.AspNetCore.Blazor"
            Version="3.0.0-preview7.19365.7" />
    <PackageReference Include="Microsoft.AspNetCore.Blazor.Build"
            Version="3.0.0-preview7.19365.7" PrivateAssets="all" />
    <PackageReference Include="Microsoft.AspNetCore.Blazor.DevServer"
            Version="3.0.0-preview7.19365.7" PrivateAssets="all" />
    <PackageReference Include="Newtonsoft.Json" Version="12.0.2" />
</ItemGroup>
```

```
<ItemGroup>
    <PackageReference Include="Microsoft.AspNetCore.Blazor"
            Version="3.0.0-preview9.19465.2" />
    <PackageReference Include="Microsoft.AspNetCore.Blazor.Build"
            Version="3.0.0-preview9.19465.2" PrivateAssets="all" />
    <PackageReference Include="Microsoft.AspNetCore.Blazor.DevServer"
            Version="3.0.0-preview9.19465.2" PrivateAssets="all" />
    <PackageReference Include="Newtonsoft.Json" Version="12.0.2" />
</ItemGroup>
```

App.razor

```
<Router AppAssembly="typeof(Program).Assembly">
    <NotFoundContent>
        <p>Sorry, there's nothing at this address.</p>
    </NotFoundContent>
</Router>
```

```
<Router AppAssembly="@typeof(Program).Assembly">
    <Found Context="routeData">
        <RouteView RouteData="@routeData" DefaultLayout="@typeof(MainLayout)" />
    </Found>
    <NotFound>
        <LayoutView Layout="@typeof(MainLayout)">
            <p>Sorry, there's nothing at this address.</p>
        </LayoutView>
    </NotFound>
</Router>
```

_Imports.razor

```
@using System.Net.Http
@using Microsoft.AspNetCore.Components.Forms
@using Microsoft.AspNetCore.Components.Layouts
@using Microsoft.AspNetCore.Components.Routing
@using Microsoft.JSInterop
@using BlazorClientSide
@using BlazorClientSide.Shared
```

```
@using System.Net.Http
@using Microsoft.AspNetCore.Components.Forms
@using Microsoft.AspNetCore.Components
@using Microsoft.AspNetCore.Components.Web
@using Microsoft.AspNetCore.Components.Routing
@using Microsoft.JSInterop
@using BlazorClientSide
@using BlazorClientSide.Shared
```

OnInit

```
WeatherForecast[] forecasts;

protected override async Task OnInitAsync()
{
    forecasts = await Http.GetJsonAsync<WeatherForecast[]>("sample-data/weather.json");
}
```

```
WeatherForecast[] forecasts;

protected override async Task OnInitializedAsync()
{
    var response = await Http.GetStringAsync("sample-data/weather.json");
    forecasts = Newtonsoft.Json.JsonConvert.DeserializeObject<WeatherForecast[]>(response);
}
```

Event Args

```
private void FirstCheckBoxHandler (UIChangeEventArgs e) {
    //...
}

private void ToggleDisplayText (UIMouseEventArgs e) {
    //...
}
```

```
private void FirstCheckBoxHandler (ChangeEventArgs e) {
    //...
}

private void ToggleDisplayText (MouseEventArgs e) {
    //...
}
```

UriHelper

```
@inject IUriHelper UriHelper

protected override void OnInit()
{
    UriHelper.OnLocationChanged += OnChangingLocation;
}
```

```
@inject NavigationManager UriHelper

protected override void OnInitialized()
{
    UriHelper.LocationChanged += OnChangingLocation;
}
```

DotNetObjectRef

```
private async void LoadUserData()
{
    UserData data = new UserData(new List<string> { "Hello", "World!" });
    await jsRuntime.InvokeAsync<object>("loadUserDataAsync",
                                        DotNetObjectRef.Create(data));
}
```

```
private async void LoadUserData()
{
    UserData data = new UserData(new List<string> { "Hello", "World!" });
    await jsRuntime.InvokeAsync<object>("loadUserDataAsync",
                                        DotNetObjectReference.Create(data));
}
```

OnAfterRender

```
protected override void OnAfterRender()
{
    Console.WriteLine("In OnAfterRender");
}
```

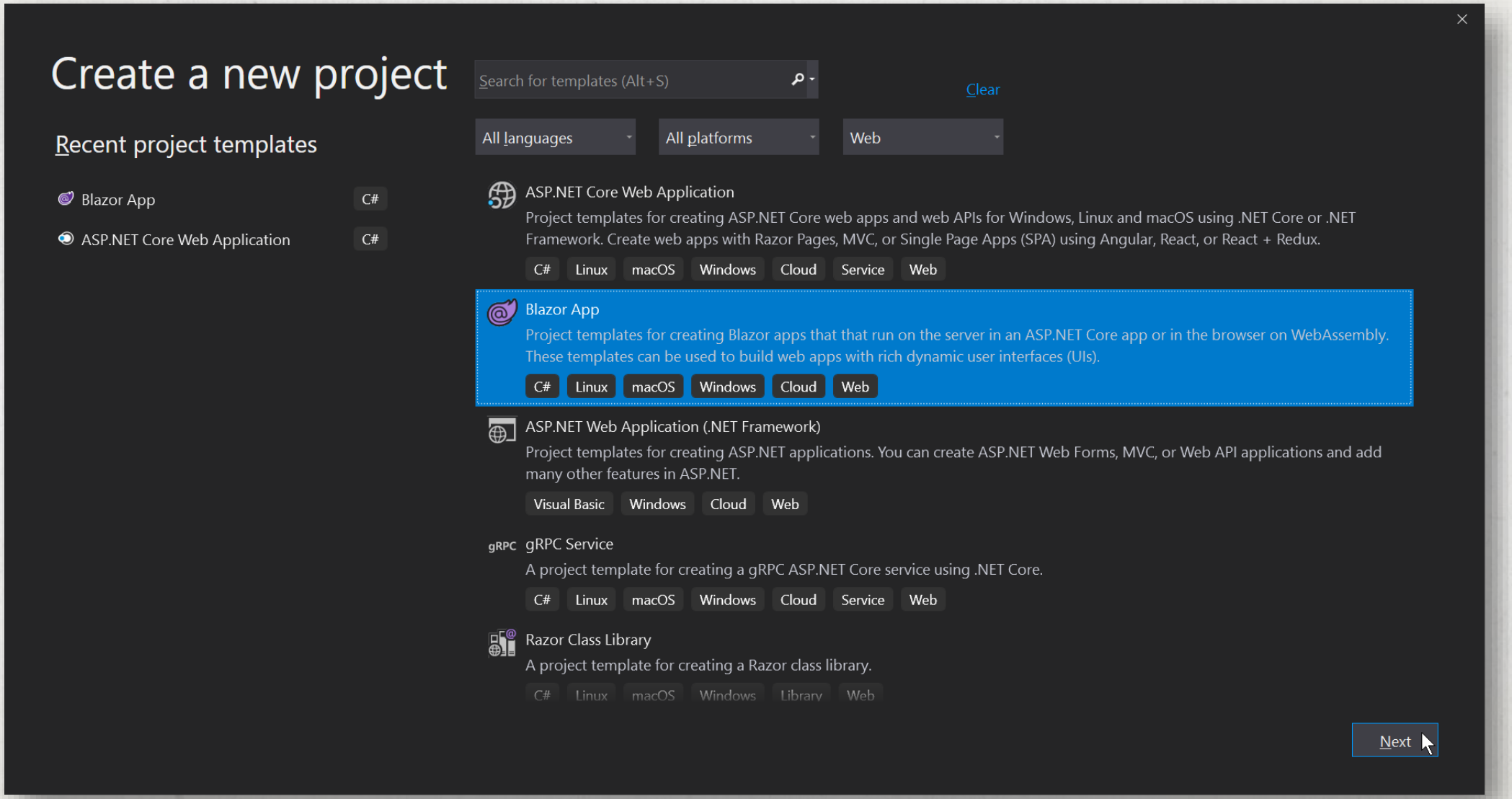
```
protected override void OnAfterRender(bool firstRender)
{
    Console.WriteLine("In OnAfterRender");
}
```

Changes in Preview-9

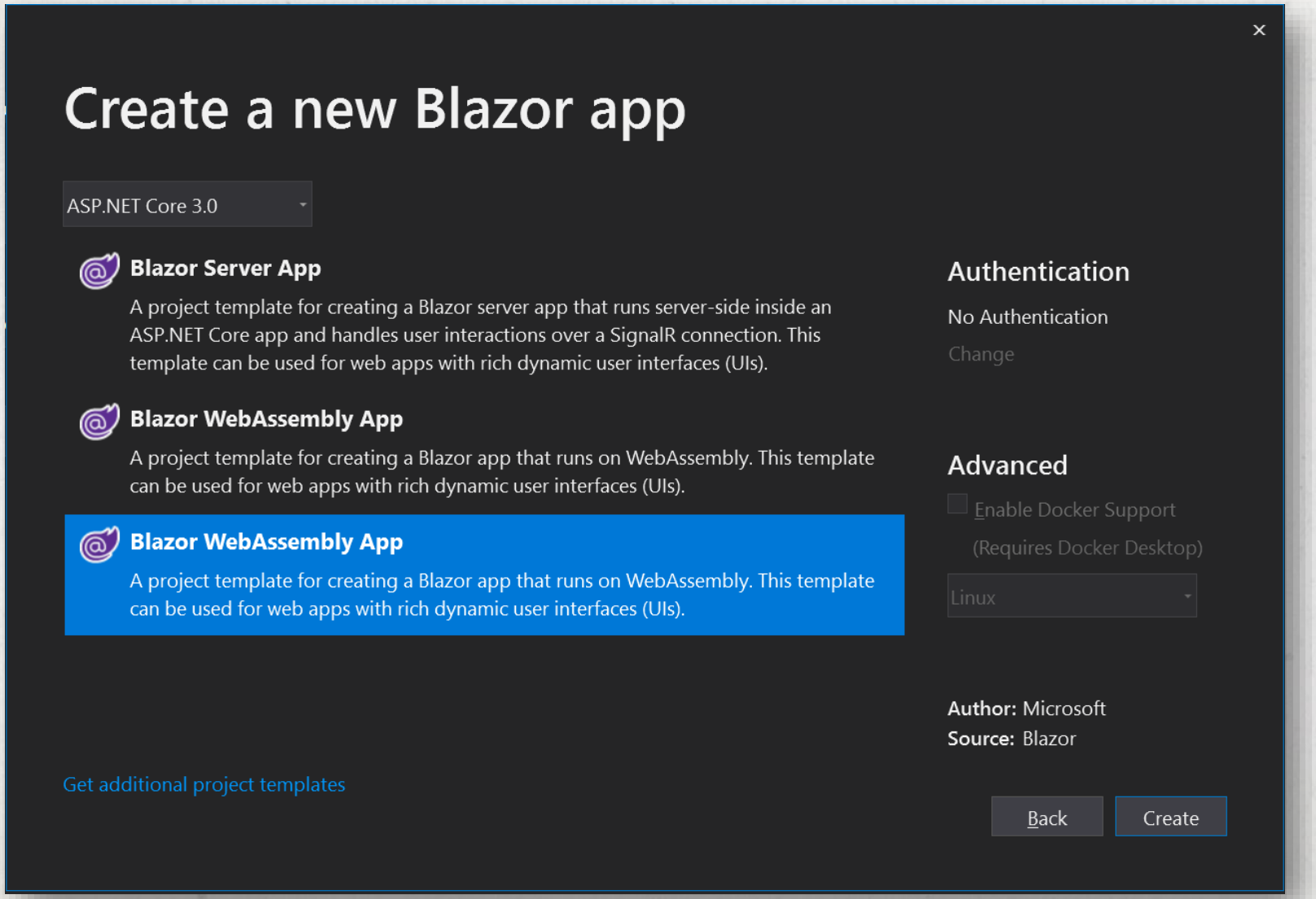
Other Notable changes

- Components in .razor files are now case-sensitive.
- NavLink component updated to handle additional attributes
- Culture aware data binding
- IJSRuntime to return ValueTask<T>
- @Layout in _imports.razor file
- Route to components from multiple assemblies

Template changes



Template changes



Upgrade to Blazor 3.1

Upgrade to 3.1 Preview-1

- Update Templates

  dotnet new -i Microsoft.AspNetCore.Blazor.Templates::3.1.0-preview1.19508.20

- Update NuGet packages

  to 3.1.0-preview1.19506.1

- Update Target

  to netcoreapp3.1

Notable changes

- Partial class support for components
- Pass parameters to top-level components
- Support for shared queues in HttpSysServer
- Breaking changes for SameSite cookies

Partial component

```
public partial class PartialClassDemo
{
    protected string PageTitle = "Blazor Page";
    protected string PageImage = "https://i.imgur.com/8W9UlMJ.png";
    protected List<string> PageDescriptions = new List<string>
    {
        "Lorem ipsum dolor sit amet, consectetur adipiscing elit. Sed pre
        "Nullam vitae vestibulum risus, sed tempor erat. Nunc eget metus
        "Pellentesque bibendum auctor diam, eu cursus metus tincidunt vel
        "In sollicitudin sagittis arcu at finibus. Nunc tincidunt, justo
        "Morbi luctus imperdiet ante id lobortis. Cras quam nisi, consequ
    };
}
```

ASP.Net Blazor

For Absolute Beginner

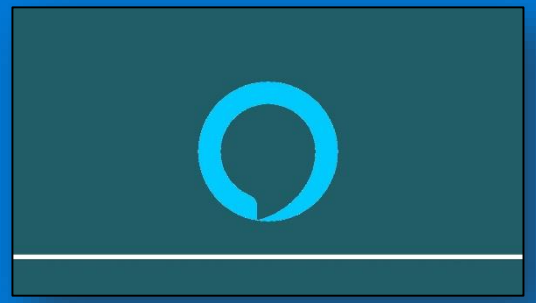
Thanks …

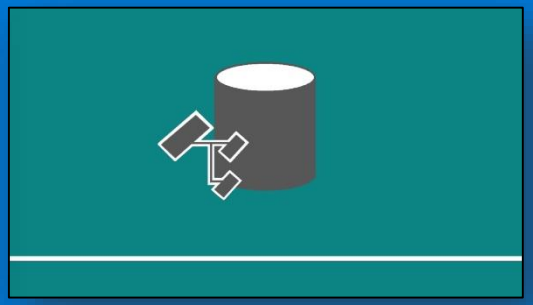
Mail@MNilay.com

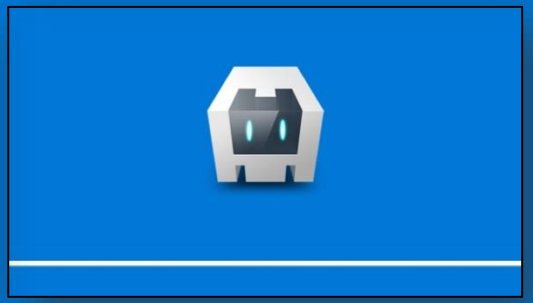
ASP.Net Blazor

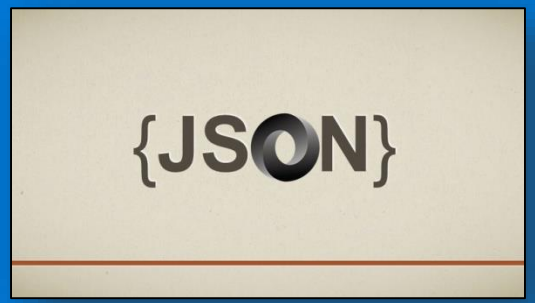
For Absolute Beginner

Tutorials Team









Important Links

YouTube Channel

http://www.youtube.com/c/TutorialsTeam

Official Site

http://www.MNilay.com/TutorialsTeam/

Facebook

https://www.facebook.com/TutorialsTeamOfficial/

My Blog

http://www.MNilay.com/blog/

Offers on other Udemy courses

www.MNilay.com/Offers/