



Component Basics

Component Definition

Blazor components are reusable UI elements written in C# and HTML (Razor syntax).

Example:

```
// MyComponent.razor
<h1>Hello, @Name</h1>

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

Components can receive data through **parameters**.

Use the `[Parameter]` attribute to define component parameters.

Lifecycle Methods

<code>OnInitialized</code>	Called when the component is initialized, after parameters are set.
<code>OnParametersSet</code>	Called after the component receives parameters from its parent.
<code>OnAfterRender / OnAfterRenderAsync</code>	Called after the component has been rendered. Use <code>firstRender</code> parameter to run logic only on the initial render.
<code>ShouldRender</code>	Allows you to control when a component should re-render. Return <code>true</code> to re-render, <code>false</code> to skip.
<code>IDisposable.Dispose</code>	Called when the component is being disposed. Use this to unsubscribe from events and release resources.

Event Handling

Blazor uses standard C# event handling.

Example:

```
<button @onclick="HandleClick">Click me</button>

@code {
    private void HandleClick()
    {
        Console.WriteLine("Button clicked!");
    }
}
```

Use `@onclick`, `@onchange`, `@oninput`, etc. to bind events to C# methods.

Data Binding

One-Way Binding

Displaying data from a C# variable in the UI.

Example:

```
<p>Current count: @currentCount</p>

@code {
    private int currentCount = 0;
}
```

Two-Way Binding

Allows updating a C# variable when the UI element changes.

Example:

```
<input type="text" @bind="inputValue" />
<p>You typed: @inputValue</p>

@code {
    private string? inputValue;
}
```

`@bind` attribute provides two-way binding. Use `@bind:event` to specify the event that triggers the update.

Binding with different event

```
<input type="text" @bind="inputValue"
@bind:event="oninput" />
<p>You typed: @inputValue</p>

@code {
    private string? inputValue;
}
```

Routing

Page Directive

Defines the route for a component, making it accessible via a URL.

Example:

```
@page "/counter"

<h1>Counter</h1>
```

Use `@page` directive at the top of the component file to specify the route.

Route Parameters

Pass data to a component through the URL.

Example:

```
@page "/user/{Id:int}"

<h1>User Details</h1>
<p>User ID: @Id</p>

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

Use `{parameterName: dataType}` syntax to define route parameters. The type constraint ensures that the parameter matches the expected type.

Navigation

Programmatically navigate between pages using `NavigationManager`.

Example:

```
@inject NavigationManager NavigationManager

<button @onclick="Navigate">Go to Counter</button>

@code {
    private void Navigate()
    {
        NavigationManager.NavigateTo("/counter");
    }
}
```

Inject `NavigationManager` into your component to use its `NavigateTo` method.

Dependency Injection

Service Registration

Register services in `Program.cs` for use in Blazor components.

Example:

```
builder.Services.AddScoped<IMyService, MyService>();
```

Use `AddScoped`, `AddTransient`, or `AddSingleton` to register services with different lifetimes.

Service Injection

Inject services into components using the `@inject` directive.

Example:

```
@inject IMyService MyService
```

```
<h1>Data from Service</h1>  
<p>@MyService.GetData()</p>
```

Use `@inject` followed by the service type and a variable name to access the service.