overview of Blazor Components

---

1. What Is a Blazor Component?

A **Blazor component** is a **self-contained, reusable UI building block** written using:

- **Razor syntax** (.razor files)
- **C# logic**
- **HTML markup**

In Blazor, **everything is a component**:

- Pages
- Layouts
- Forms
- Navigation menus
- Dialogs
- Reusable UI controls

A component controls:

- **What is rendered**
- **How it behaves**
- **How it responds to user interaction**

---

2. Basic Structure of a Blazor Component

A component typically has three parts:

```
<h3>Hello Blazor</h3>


<p>@Message</p>


<button @onclick="ChangeMessage">Click Me</button>
```

```
@code {
    string Message = "Welcome to Blazor";

    void ChangeMessage()
    {
        Message = "Message Updated!";
    }
}
```

Key Observations

- **HTML markup** defines UI
- @ allows embedding **C# expressions**
- @code block contains component logic
- UI automatically refreshes when state changes

---

3. Component = UI + State + Behavior

A Blazor component encapsulates:

Aspect	Description
UI	HTML + Razor
State	Fields & properties
Behavior	Methods & events
Lifecycle	Initialization → Rendering → Disposal

This is why Blazor is considered a **component-based framework**.

---

4. Types of Blazor Components

4.1 Page Components

- Have routing
- Accessible via URL
- Use @page directive

```
@page "/products"

<h3>Products</h3>
```

4.2 Reusable (Child) Components

- No routing
- Used inside other components
- Accept parameters

```
<ProductCard Title="Laptop" Price="50000" />
```

---

5. Component Hierarchy (Parent → Child)

Blazor applications form a **component tree**.

- Parent components pass data to children
- Children raise events to parents
- State flows **top-down**
- Events flow **bottom-up**

---

6. Component Parameters

Components receive data using [Parameter].

Child Component

```
<h4>@Title</h4>


@code {
    [Parameter]
```

```
    public string Title { get; set; }
}
```

Parent Component

```
<MyComponent Title="Blazor Basics" />
```

Parameters make components **reusable and configurable**.

---

7. Event Handling in Components

Blazor components handle events using C# (not JavaScript).

```
<button @onclick="Save">Save</button>


@code {
    void Save()
    {
        Console.WriteLine("Saved");
    }
}
```

Supported events include:

- @onclick
- @onchange
- @oninput
- @onmouseover
- @onsubmit

---

8. Data Binding in Components

One-way binding

```
<p>@UserName</p>
```

Two-way binding

<input @bind="UserName" />

Blazor uses **event-based rendering**, not continuous change detection like Angular.

---

9. Component Lifecycle

Every component follows a lifecycle.

Common Lifecycle Methods

Method	**Purpose**
OnInitialized()	First-time initialization
OnParametersSet()	When parameters change
OnAfterRender()	After UI render
Dispose()	Cleanup resources

Lifecycle methods are critical for:

- API calls
- SignalR connections
- Timers
- JS interop

---

10. Rendering & State Management

- Blazor uses a **render tree**
- Only **changed DOM parts** are updated
- UI updates occur when:
    - An event fires
    - A parameter changes

This makes Blazor **efficient and performant**.

---

11. Component Reusability & Composition

Components can:

- Contain other components
- Accept RenderFragment (UI as parameters)
- Be placed in shared libraries

Example:

<Modal>

  <h3>Delete Confirmation</h3>

</Modal>

This enables **clean UI composition**, similar to React.
