1. What Is Data Binding in Blazor?

Data binding connects UI elements (HTML/razor components) with C# properties so that:

- UI updates when data changes
- Data updates when the user interacts with the UI

Blazor supports both **synchronous** and **asynchronous** binding scenarios.

---

2. Synchronous Data Binding

Definition

Synchronous data binding occurs when:

- Data is already available in memory
- No async operations (async/await)
- UI updates immediately on user interaction

This is the **most common** form of binding.

Example 1: Two-Way Synchronous Binding

```
<h3>User Profile</h3>


<input @bind="UserName" />

<p>Hello, @UserName</p>


@code {

   string UserName = "San";

}
```

How It Works

- @bind creates **two-way binding**
- UI and property stay in sync

- Change happens **in the same thread**
- No waiting, no async lifecycle involved

Important Behavior (Interview Favorite)

By default, Blazor uses:

```
@bind:event="onchange"
```

So the value updates **after focus loss** (Tab or click outside).

Immediate Update (Like Angular)

```
<input @bind="UserName" @bind:event="oninput" />
```

Now updates happen **on every keystroke**.

When to Use Synchronous Binding

- Form inputs
- Local calculations
- Toggle switches
- UI state (show/hide, enable/disable)

4. Asynchronous Binding with Lifecycle Methods

Example 2: Load Data from API

```
<h3>Products</h3>


@if (products == null)
{
   <p>Loading...</p>
}
else
```

```
{
    <ul>
        @foreach (var product in products)
        {
            <li>@product</li>
        }
    </ul>
}

@code {
    List<string>? products;

    protected override async Task OnInitializedAsync()
    {
        await Task.Delay(2000); // Simulate API call
        products = new() { "Laptop", "Mobile", "Tablet" };
    }
}
```

What Is Happening

1. Component renders first
2. OnInitializedAsync() runs
3. UI shows **Loading...**
4. Data arrives
5. UI re-renders automatically

---

5. Async Binding Triggered by User Action

```
}
```

Example 3: Button Click with Async Update

```
<button @onclick="LoadData">Load Users</button>


@if (users == null)
{
    <p>No data loaded</p>
}
else
{
    <ul>
        @foreach (var user in users)
        {
            <li>@user</li>
        }
    </ul>
}


@code {
    List<string>? users;


    async Task LoadData()
    {
        await Task.Delay(1500); // API call
        users = new() { "Admin", "Manager", "User" };
    }
}
```

Key Point

- Event handler is async
- UI updates only **after await completes**
- No blocking of UI thread

6. Async Binding with Input Controls (Common Mistake)

❌ Incorrect

```
<input @bind="SearchText" @oninput="SearchAsync" />
```

This causes binding conflicts.

✅ Correct Pattern

```
<input value="@SearchText" @oninput="OnInputChanged" />


@code {
    string SearchText = "";


    async Task OnInputChanged(ChangeEventArgs e)
    {
        SearchText = e.Value?.ToString() ?? "";
        await Task.Delay(500); // API call
    }
}
```

7. Synchronous vs Asynchronous Binding – Comparison

Aspect	Synchronous	Asynchronous
Execution	Immediate	Awaited
Thread	Same thread	Non-blocking
Data Source	Local	API / DB

Lifecycle	Simple	Lifecycle-based
UI Feedback	Instant	Loading states
Usage	Forms, UI state	Real applications