Blazor Cheat Sheet

Components · Data Binding · Routing · JS Interop

worksheetdownload.com

SETUP

```
dotnet new blazorserver -n MyApp
dotnet new blazorwasm -n MyApp
(standalone)
dotnet run
Blazor Server: real-time SignalR
connection
Blazor WASM: runs .NET in browser via
WebAssembly
dotnet add package
Microsoft.AspNetCore.Components.Web
```

COMPONENTS

```
@* Counter.razor *@
<h1>Count: @count</h1>
<button @onclick="Increment">+</button>
@code {
private int count = 0;
private void Increment() => count++;
}
```

DATA BINDING

```
<input @bind="name" />         //
two-way binding
<input @bind-value="name"
@bind-value:event="oninput" />
<p>@name.ToUpper()</p>         //
one-way from C# to HTML
@bind:format="yyyy-MM-dd"
<select @bind="selectedValue">
@foreach (var o in options) {
<option>@o</option> }
</select>
```

ROUTING

```
@page "/counter"
@page "/counter/{StartCount:int}"
[Parameter] public int StartCount { get;
set; }
<NavLink href="/counter"
Match=NavLinkMatch.All>Counter</NavLink>
@inject NavigationManager Nav
Nav.NavigateTo("/home")
Nav.Uri  // current URL
```

PARAMETERS & CASCADING

```
@* Child.razor *@
[Parameter] public string Title { get;
set; }
[Parameter] public EventCallback<int>
OnChange { get; set; }
[CascadingParameter] public MyTheme
Theme { get; set; }
@* Parent usage *@
<Child Title="Hello"
OnChange="HandleChange" />
<CascadingValue Value="theme"><Child
/></CascadingValue>
```

EVENT HANDLING

```
<button
@onclick="HandleClick">Click</button>
<button @onclick="() =>
count++">Add</button>
<button @onclick="@(e =>
HandleClick(e))">Click</button>
private async Task
HandleClick(MouseEventArgs e) {
await JS.InvokeVoidAsync("alert",
"clicked!");
}
@onfocus @oninput @onkeydown @onsubmit
```

LIFECYCLE METHODS

```
protected override void OnInitialized()
protected override async Task
OnInitializedAsync()
protected override void
OnParametersSet()
protected override async Task
OnParametersSetAsync()
protected override void
OnAfterRender(bool firstRender)
protected override bool ShouldRender()
=> canRender;
public void Dispose() { ... }  //
IDisposable
```

JS INTEROP

```
@inject IJSRuntime JS
await JS.InvokeVoidAsync("console.log",
"Hello from .NET");
var width = await
JS.InvokeAsync<int>("getWindowWidth");
// JS calling .NET:
[JSInvokable]
public static string GetData() => "from
.NET";
// In JS:
DotNet.invokeMethodAsync('MyApp','GetData')
```

FORMS & VALIDATION

```
<EditForm Model="user"
OnValidSubmit="HandleSubmit">
<DataAnnotationsValidator />
<ValidationSummary />
<InputText @bind-Value="user.Name" />
<ValidationMessage For="() => user.Name"
/>
<button type="submit">Save</button>
</EditForm>
[Required][MaxLength(50)] public string
Name { get; set; }
```

COMMON GOTCHAS

Server-side Blazor: network latency on every interaction
Blazor WASM: initial load slow (download .NET runtime)
StateHasChanged() needed for external state changes
Don't use @onclick on non-Blazor elements without @onclick:stopPropagation
IDisposable on components that subscribe to events