Dynamic Component Rendering in Blazor

Dynamic component rendering is a core Blazor capability that allows you to decide at runtime which component should be rendered, instead of hard-coding it in Razor markup. This is essential for scenarios such as dashboards, role-based UI, plug-in architectures, and wizard-style workflows.

---

1. What Is Dynamic Component Rendering?

In Blazor, dynamic rendering means:

- **The component type is not known at compile time.**
- **The component is selected based on state, data, or user interaction.**
- **Parameters can also be passed dynamically.**

Blazor provides a built-in mechanism for this via the DynamicComponent component.

---

2. The DynamicComponent Primitive

Syntax

```
<DynamicComponent Type="@componentType" Parameters="@parameters" />
```

Key Properties

Property	Description
Type	Type of the component to render
Parameters	Dictionary of parameter names and values

---

3. Basic Example: Render Components Dynamically

Step 1: Create Components

ComponentA.razor

```
<h3>Component A</h3>
<p>This is Component A</p>
```

ComponentB.razor

```
<h3>Component B</h3>

<p>This is Component B</p>
```

Step 2: Host Component

DynamicHost.razor

```
@using Microsoft.AspNetCore.Components

<button class="btn btn-primary" @onclick="() => LoadComponent(typeof(ComponentA))">
    Load A
</button>

<button class="btn btn-secondary" @onclick="() => LoadComponent(typeof(ComponentB))">
    Load B
</button>

<hr />

<DynamicComponent Type="@CurrentComponent" />

@code {
    private Type? CurrentComponent;

    void LoadComponent(Type componentType)
    {
        CurrentComponent = componentType;
```

```
    }
}
```

Result

- **Clicking Load A renders ComponentA**
- **Clicking Load B renders ComponentB**

---

4. Passing Parameters Dynamically

Child Component

UserCard.razor

```
<h4>User: @Name</h4>

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

Host Component

```
<DynamicComponent Type="@typeof(UserCard)"
                  Parameters="@componentParameters" />

@code {
    Dictionary<string, object> componentParameters =
        new()
        {
            { "Name", "San" }
        };
}
```

Key Rule

- **Dictionary key must match the [Parameter] name exactly**

---

5. Role-Based Dynamic Rendering (Real-World Use Case)

```
<DynamicComponent Type="@DashboardComponent" />


@code {
   Type DashboardComponent => UserRole switch
   {
      "Admin" => typeof(AdminDashboard),
      "User"  => typeof(UserDashboard),
      _       => typeof(AccessDenied)
   };

   string UserRole = "Admin";
}
```

This pattern is frequently used in:

- **Authentication/Authorization flows**
- **Admin vs User dashboards**
- **Feature toggles**

---

6. Dynamic Rendering Using Configuration / Menu

```
<select @onchange="OnMenuChange">
   <option value="">-- Select --</option>
   <option value="Orders">Orders</option>
   <option value="Products">Products</option>
```

```
</select>

<DynamicComponent Type="@SelectedComponent" />

@code {
    Type? SelectedComponent;

    void OnMenuChange(ChangeEventArgs e)
    {
        SelectedComponent = e.Value?.ToString() switch
        {
            "Orders"   => typeof(OrdersComponent),
            "Products" => typeof(ProductsComponent),
            _ => null
        };
    }
}
```

---

7. Dynamic Rendering Without DynamicComponent (Manual Approach)

Useful for small, simple cases, but not scalable

```
@if (mode == "A")
{
    <ComponentA />
}
else if (mode == "B")
{
```

<ComponentB />

}

Limitation

- **Compile-time coupling**
- **Poor maintainability as options grow**

---

10. Common Mistakes

Mistake	Explanation
Passing wrong parameter name	Causes runtime exception
Forgetting Type	Component will not render
Expecting state retention	Components are recreated
Using DynamicComponent for static UI	Overkill

---

11. When Should You Use Dynamic Components?

Use When:

- **UI varies by role / config**
- **Component choice is runtime-driven**
- **Building extensible UI systems**

Avoid When:

- **UI structure is fixed**
- **Simple conditional rendering is sufficient**

Summary

Dynamic Component Rendering in Blazor enables runtime-driven UI composition using DynamicComponent. It is a powerful technique when building flexible, extensible, and role-aware applications, especially in enterprise Blazor Server or WebAssembly projects.

Difference Between **Routing** and **DynamicComponent** in Blazor

Aspect	**Routing**	**DynamicComponent**
Primary purpose	URL-based navigation	Runtime UI composition
Trigger	Browser URL change	Application state / logic
Component selection	Declarative (@page)	Programmatic (Type)
URL changes	Yes	No
Browser history	Yes	No
SEO / Deep linking	Supported	Not supported
Typical usage	Pages / navigation	Widgets / sections