# Blazor

# Overview

For building interactive web UI. Use C# instead of JS.

## Components

Define Flexible UI rendering logic

Handle user events

Nested and reused

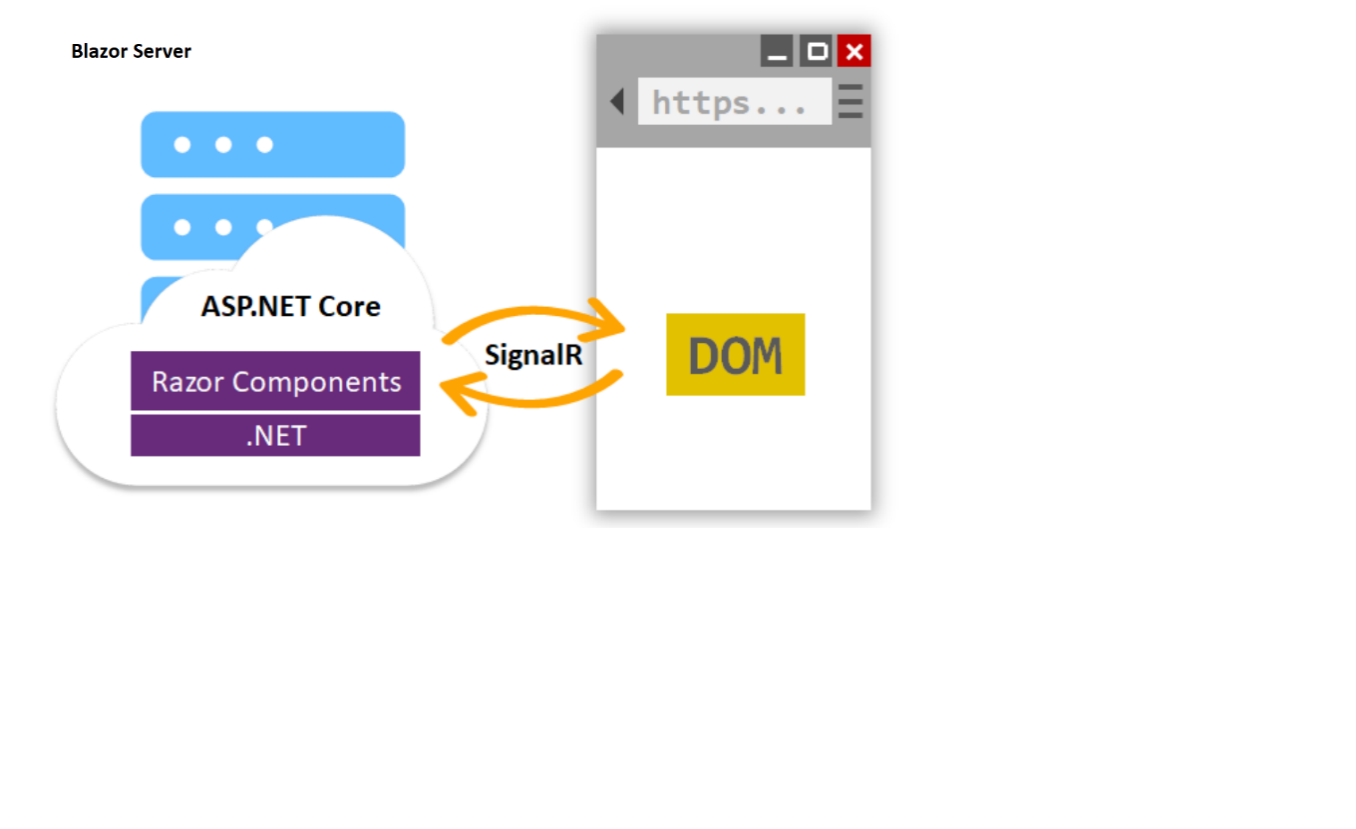
Shared and distributed as Razor class lib or nuget packages

Built as razor markup page in .razor file extension

**Sample component**

<div class="card" style="width:22rem">  
<div class="card-body">  
<h3 class="card-title">@Title</h3>  
<p class="card-text">@ChildContent</p>  
<button @onclick="OnYes">Yes!</button>  
</div>  
</div>  
@code {  
[Parameter]  
public RenderFragment? ChildContent { get; set; }  
[Parameter]  
public string? Title { get; set; }  
private void OnYes()  
{  
Console.WriteLine("Write to the console in C#! 'Yes' button  
selected.");  
}  
}

## Blazor Server



* Hosting Razor components on the server
* UI Updates are handled over signalR connection
* Handles javascript interop
* **Rendering the markup in UI varies** for blazor server app and asp.net core apps using razor views or pages. Both models uses razor language.
* **View to HTML text. Once its done server disposes view or page. For every request the page is re-rendered to HTML again and send to the client**
* **Graph of components is equivalent for DOM in HTML or XML**
* **Component graph includes state held in properties and fields.**
* **Binary form of component graph is sent between server and client**
* **After the connection, the components static prerendered elements are replaced with interactive elements.**
* A component is disposed after the user navigates away from the component

Blazor Webassembly

Run dotnet code in web browser

Bytecode format

Open web standard, so no plugins required

Will access browser functionality using JS interop

Runs in JS sandbox with protection against malicious actions on the client machine

