

BLAZOR



Contents

- | | | | |
|----|--|----|-------------------------------|
| 01 | How we develop web applications today? | 06 | Blazor WebAssembly |
| 02 | Blazor | 07 | Server side hosting Model |
| 03 | How can a browser execute C# code? | 08 | Razor project structure |
| 04 | How Create a Blazor App? | 09 | ASP.NET Core razor components |
| 05 | Blazor Hosting Models | | |

How we develop web applications today?

For server-side development, we use programming languages like C#, Java, PHP etc.

For the client-side development we use JavaScript frameworks like Angular, React, Vue etc.

To stay in the business as a developer and remain competitive, it's inevitable we learn both a server-side programming language and a client-side programming language.

But the question is, why should we learn and use 2 different sets of programming languages and frameworks.



Blazor

Blazor is one of the frameworks for the .Net family for web development.

With Blazor we can now build interactive web UIs using C# instead of JavaScript

C# code can be executed both on the server and in the client browser.



How can a browser execute C# code?

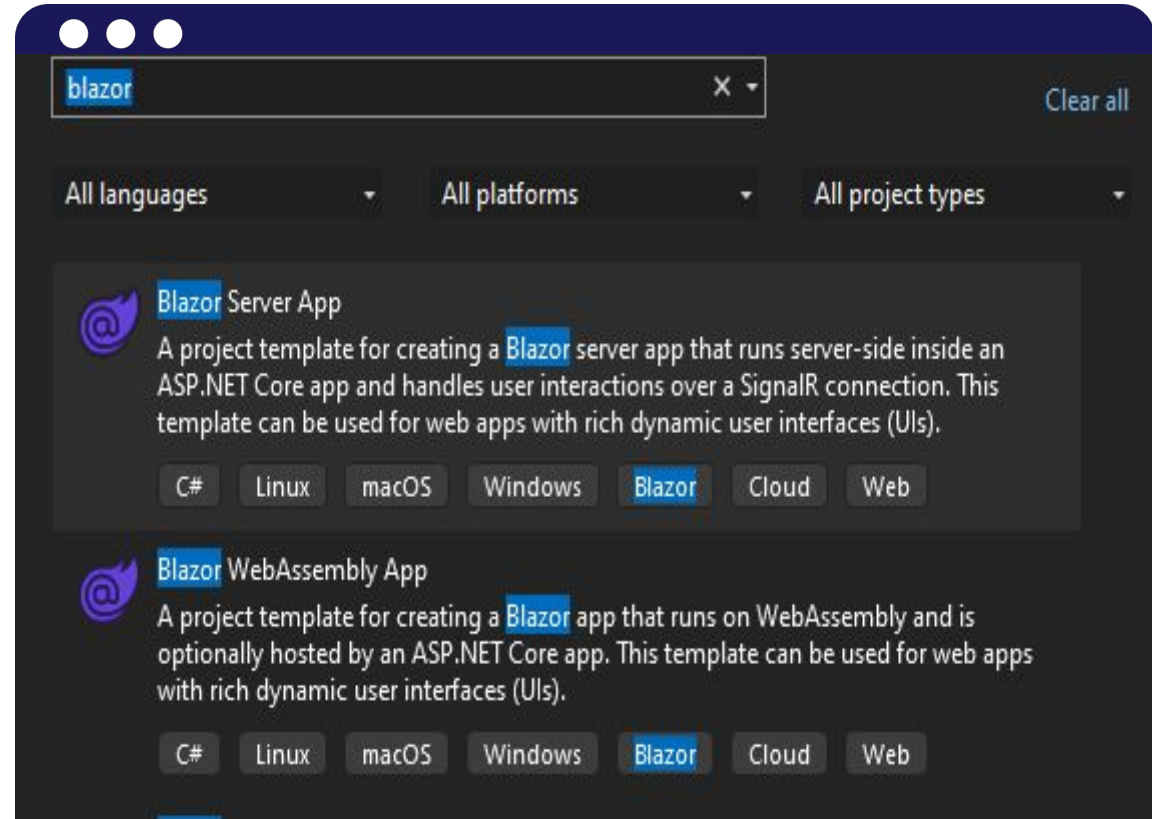
Blazor can run C# code directly in the browser, using WebAssembly.

WebAssembly is based on open web standards. So it is a native part of all modern browsers including mobile browsers.



How Create a Blazor App?

The IDE Used for creating Blazor App is Visual Studio.



Blazor Hosting Models

Blazor offers 2 hosting models:

➔ **Blazor WebAssembly**
(client-side hosting model)

➔ **Blazor Server**
(Server-side Hosting model)

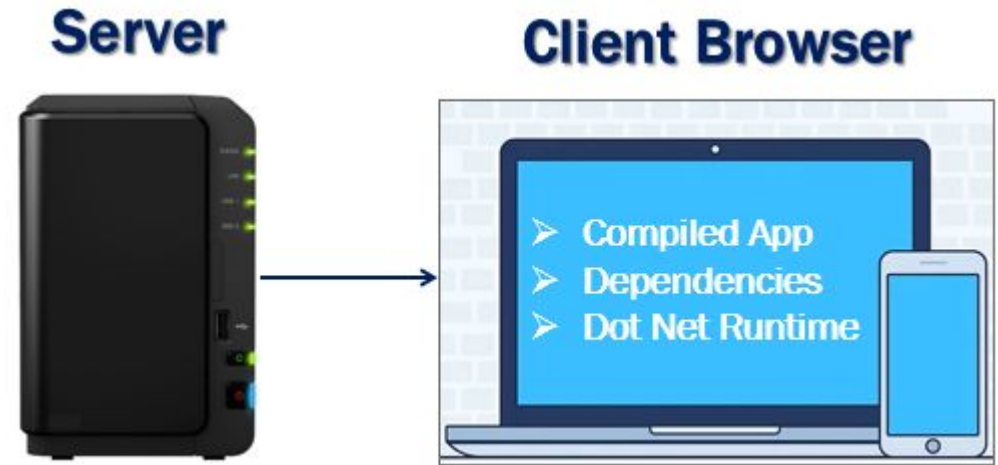


Blazor WebAssembly

Application runs directly in the browser on WebAssembly.

Everything the application needs i.e the compiled application code itself, it's dependencies and the .NET runtime are downloaded to the browser

A Blazor WebAssembly app can run entirely on the client without a connection to the server



Advantages and disadvantages



- ❑ Active server connection is not required.
- ❑ Client Resources and Capabilities is used.
- ❑ Full blown asp.net core web server is not required.
- ❑ Can be hosted on our own webserver, cloud azure or CDN



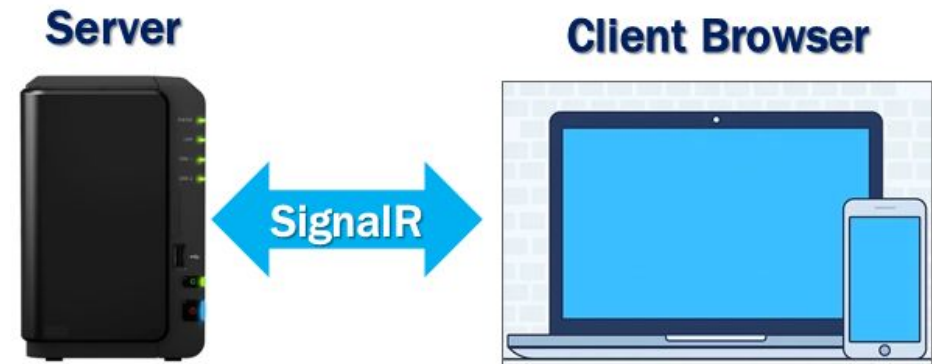
- ❑ The very First request usually takes longer.
- ❑ Restricted to the capabilities of the browser.
- ❑ Capable client hardware and software is required

Server side hosting Model

Application is executed on the server.

Between the client and the server a SignalR connection is established.

When an event occurs on the client such as a button click for example, the information about the event is sent to the server over the SignalR connection. The server handles the event and for the generated HTML a diff (difference) is calculated. The entire HTML is not sent again to the client, it's only the diff that is sent to the client over the SignalR connection. The browser then updates the UI. Since only the diff is applied to update the UI, the application feels faster and more responsive to the user.



Advantages and disadvantages



- Application loads much faster.
- can take full advantages of server capabilities.
- All the client needs, to use the app is browser.
- More secure as the app code is not sent to the client



- Asp.Net core web server is required.
- An active connection to the server is required.
- Higher latency due to the round trip to the server.
- Scalability can be challenge

Blazor project structure

Program.cs: contains the `Main()` method which is the entry point for both the project types

wwwroot: For both the project types, this folder contains the static files like images, stylesheets etc.

App.razor: This is the root component of the application. It uses the built-in Router component and sets up client-side routing.



Pages folder: This folder contains the `_Host` razor page and the routable components that make up the Blazor app. The components have the `.razor` extension.

Shared folder: As the name implies, contains the shared components (`MainLayout` component (`MainLayout.razor`), `NavMenu` component (`NavMenu.razor`))

`_Imports.razor`: contains the common namespaces so we do not have to include them in every razor component.



wwwroot/index.html: This is the root page in a Blazor WebAssembly project and is implemented as an html page.

Startup.cs: it contains the applications's startup logic.
ConfigureServices - Configures the applications DI i.e dependency injection services.
Configure - Configures the app's request processing pipeline.

Pages/_Host.cshtml: This is the root page of the application and is specified by calling `MapFallbackToPage("/_Host")` method. It is implemented as a razor page.



Data folder (Blazor Server):Contains code files related to the sample WeatherForecast service

appsettings.json (Blazor Server):uses this file to store the Configuration settings.



ASP.NET Core razor components

Components are the fundamental building blocks of a Blazor application.

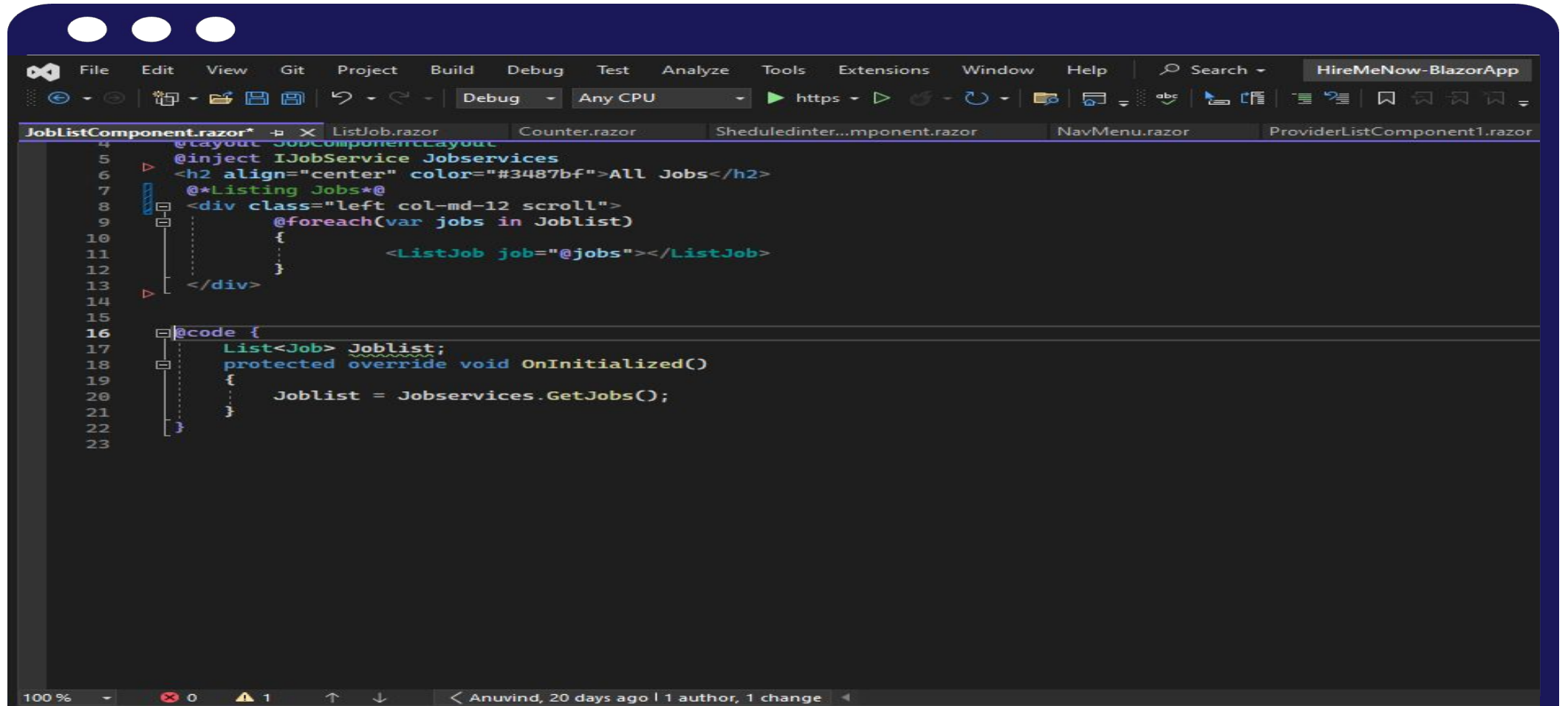
They can be nested, reused, and if implemented properly, can even be shared across multiple projects.

Component files have the extension `.razor`

Component names cannot start with a lowercase character



Razor component example



```
4 @layout JobComponentLayout
5 @inject IJobService Jobservices
6 <h2 align="center" color="#3487bf">All Jobs</h2>
7 @*Listing Jobs*@
8 <div class="left col-md-12 scroll">
9     @foreach(var jobs in Joblist)
10     {
11         <ListJob job="@jobs"></ListJob>
12     }
13 </div>
14
15
16 @code {
17     List<Job> Joblist;
18     protected override void OnInitialized()
19     {
20         Joblist = Jobservices.GetJobs();
21     }
22 }
23
```

Component is combination of two things:

01

**HTML markup which defines the user interface of the component
i.e the look and feel.**

02

C# code which defines the processing logic

Component is rendered by giving path in the @page directive

