Blazor Server
Introduction

2023

Outline

- Introduction
- Blazor Component
- Hosting models

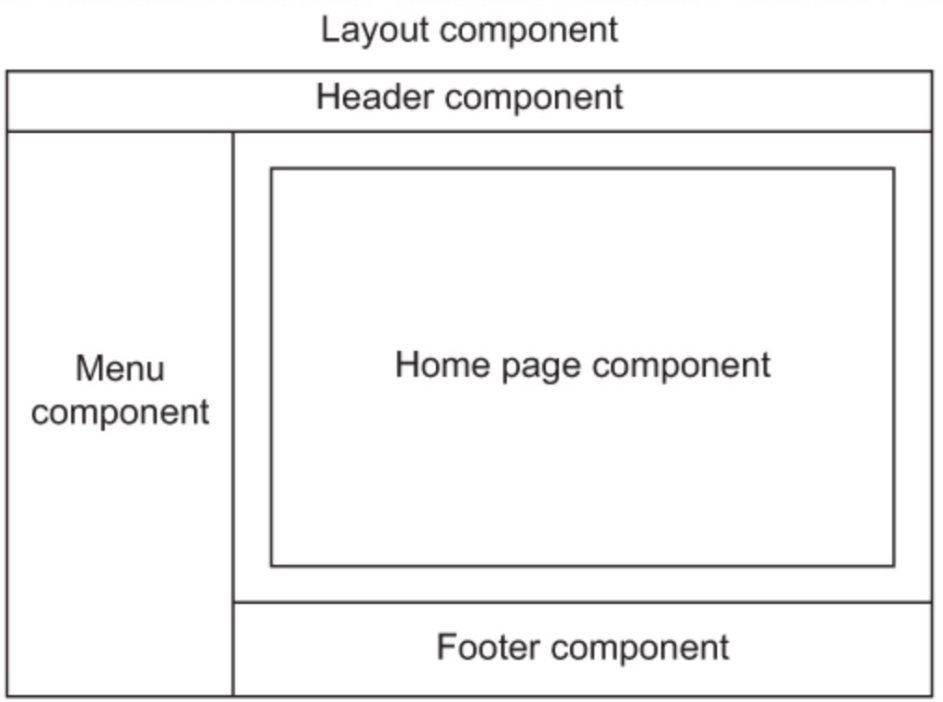
Introduction

Blazor, as with many modern frontend frameworks, uses the concept of components to build the UI.

Everything is a component—pages, parts of a page, layouts.

Think of a component as a building block.

Building blocks can be as big or as small as you decide; however, building an entire UI as a single component isn't a good idea.



Anatomy of a Blazor component

This particular component is known as a routable component, as it has a page directive declared at the top.

```
@page "/counter"                                    ❶

<h1>Counter</h1>                                    ❷

<p>Current count: @currentCount</p>

<button class="btn btn-primary" @onclick="IncrementCount">Click me</button>

@code {                                             ❸
    private int currentCount = 0;

    private void IncrementCount()
    {
        currentCount++;
    }
}
```

Section 1—Used to define directives, add using statements, inject dependencies, or any other general configuration that applies to the whole component.

Section 2—Defines the markup of the component; this is written using the Razor language, a mix of C# and HTML. Here we define the visual elements that make up the component.

Section 3—The code block. This is used to define the logic of the component. It is possible to write any valid C# code into this section. You can define fields, properties, or even entire classes if you wish.

Understanding hosting models

When first getting started with Blazor, you will immediately come across hosting models:

- Blazor WebAssembly
- Blazor Server.

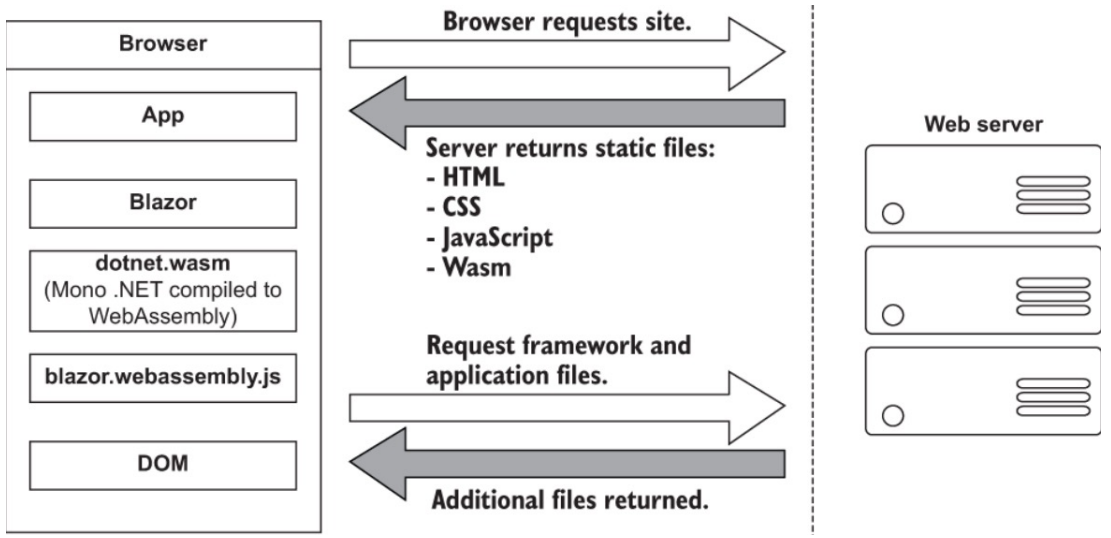
Regardless of which model you choose for your application, the component model is the same, meaning components are written the same way and can be interchanged between either hosting model

Blazor WebAssembly

Blazor WebAssembly allows your application to run entirely inside the client's browser

This part of the framework does three main things:

- Loads and initializes the Blazor application in the browser
- Provides direct DOM manipulation so Blazor can perform UI updates
- Provides APIs for JavaScript interop scenarios

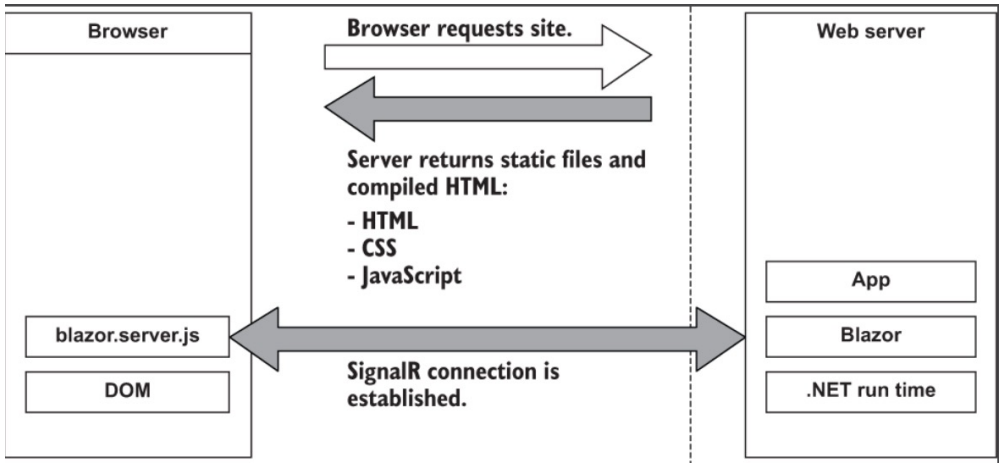


Blazor Server

Unlike Blazor WebAssembly, which behaves more like a desktop application with each user having their own instance, Blazor Server runs one instance of the app that all users connect to;

Once the initial payload is returned to the browser, the files are processed and the DOM is created—then a file called blazor.server.js is executed.

The job of this run time is to establish a SignalR connection back to the Blazor application running on the server.

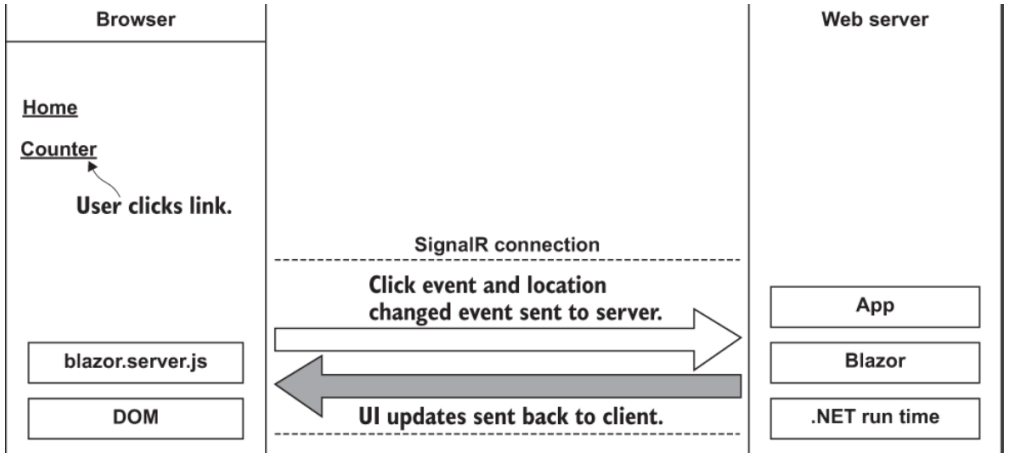


Blazor Server

The user clicks the link in the menu, and the click event is intercepted by Blazor's run time on the cliente

Then processes the event to understand what happened: mouse click event and a navigation event, due to it being a clicked hyperlink.

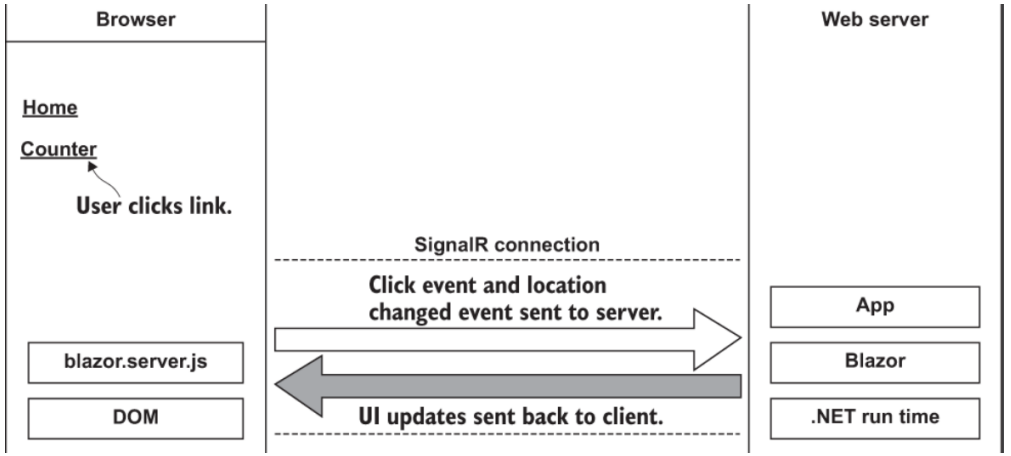
These two events are then bundled up and sent back to the server over the SignalR connection that was established when the application started.



Blazor Server

On the server, the message sent from the client is unpacked and processed.

The Blazor framework then calls any application code necessary.

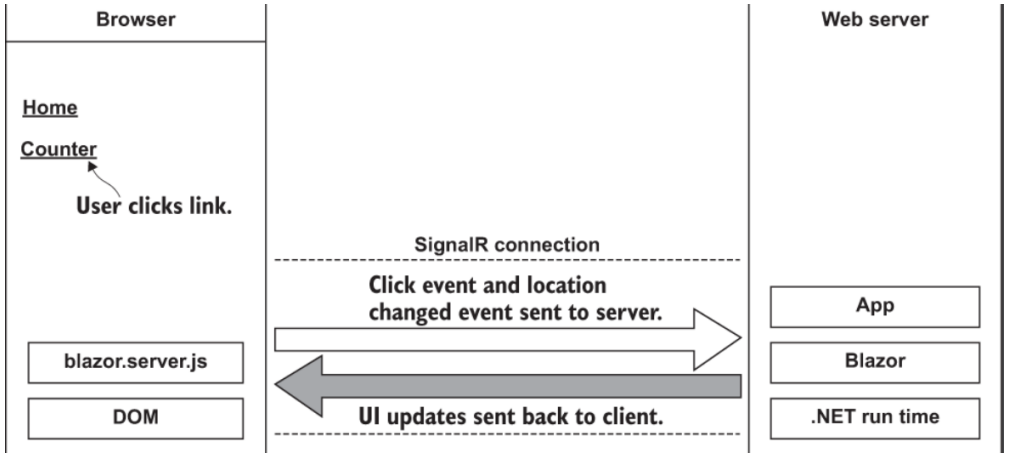


Blazor Server

Blazor will not send back an entirely new page to the client.

It will send back only the minimum number of instructions needed to update the current DOM to match the Counter page

Once back on the client, the changes are unpacked and the required changes are applied to the physical DOM.



Summary

Blazor allows developers to use the power of C# and .NET to create rich interactive UIs without the need for JavaScript.

Blazor is an SPA framework that can run entirely inside the browser via WebAssembly, an open web standard, or on the server utilizing a SignalR connection to link the client's browser with the application.

Summary

Blazor WebAssembly can be used with any existing server technology. However, there are real benefits when using it with an ASP.NET Core backend, as code can easily be shared via a .NET class library.

Applications are written using components. Components allow us to create self-contained pieces of UI that can work alone or in combination with each other.

Blazor Server
Introduction

2023