Chapter 1

# Introduction to WebAssembly or WASM

**Transpiling**: Takes one language and converts it to another language. Like TypeScript is transpiled into JavaScript.

**WebAssembly**: Takes the parsing and compilation process to the server, before the user even open up their server.

**What is WebAssembly**: it is a binary instruction format for stack-based virtual machines, WASM is designed as a portable target for compilation of high-level languages like C/C++/Rust, enabling deployment on the web for client and server application.

**Stack-Based virtual machine**: A stack-based virtual machine is a type of computer architecture where data and instructions are managed using a stack data structure. Operations are performed on the top elements of the stack, making it a simple and efficient design commonly used in interpreters and virtual machines for programming languages.

**Which applications support WebAssembly**: every modern browser supports WebAssembly without Internet Explorer.

**What is Mono**: is an open source implementation of the .NET CLI specification, meaning that Mono is a platform for running .NET assemblies. You can also use it to build applications for macOS, Linux, Tizen, and others. Mono also allows you to run .NET on Linux (its original purpose) and is written in C++.

**There are two approaches about Mono and .NET WebAssembly**:

1. One is where you take your .NET code and you compile it together with the Mono runtime into one big WASM application. However, this approach takes a lot of time because you need to take several steps to compile everything into WASM, not so practical for day-to-day development.
2. The other approach takes the Mono runtime and compiles it into WASM, and this runs in the browser where it will .NET Intermediate Language just like normal .NET does. The big advantage is that you can simply run .NET assemblies without having to compile them first into WASM.

**Note**: Before Blazor used Mono runtime, then they have created separated runtime for Blazor WebAssembly.

**Interacting Blazor with browsers**: WebAssembly with .NET allows you to run .NET code in the browser. In Blazor you use Razor Pages to dynamically generate HTML code in the client side. With Blazor you don’t need JavaScript to build web apps. For using some browser features you will still need to use JavaScript.

**Blazor Server**: in this model Blazor application is running on the server and the page updates gets serialized on the browser using SignalR.

**Pros and Cons of Blazor Server:**

1. **Smaller downloads**: the browser doesn’t need to download all .NET Blazor application. The browsers jus downloads small JavaScript library which sets up SignalR connection to the server.
2. **Development Process:** Blazor WebAssembly doesn’t support advanced features of .NET debugger, but with Blazor Server you can. So, you can start building your application to the server side and when it is finished you can switch it to the server.
3. **.NET APIs:** if your Blazor application is running on the server you can use all .NET APIs. But it will stop quickly converting it into a client-side application.
4. **Online Only:** running your application on the server means that your application will always be needed to run with a connection with to the server. If the connection drops between the browser and the application the may lose some work and Blazor will try to reconnect to server.
5. **Server Capability:** if your application has thousands of users the server must handle all of them.

**A Tree-Shaking algorithm**: the primary goal of this algorithm is to eliminate unused code from the bundle of project, or this may be called refactoring.

**PWA (Progressive Web Application):** A Progressive Web App (PWA) is a web application that offers a native app-like experience with features such as offline access, push notifications, and the ability to install to a device's home screen. PWAs are responsive, secure, and work across various platforms without the need for app store installation. They enhance web experiences by providing improved performance and user engagement.

**Kestrel**: is a built-in ASP.NET server that is generated with --hosted option. This can run ASP.NET application on Windows, Linux and OSX. This is the best option during development.

**Open the Program.cs:**

1. The configure section is responsible for installing middleware.
2. **The UseDeveloperException method**: this is an error handling middleware which is used in development phase, this method is used to view an error in details.
3. **ASPNETCORE\_ENVIRONMENT:** knows about you that you are in development phase by setting the value of **ASPNETCORE\_ENVIRONMENT** to **DEVELOPMENT**, open **launchSettings.json** file in **Properties** folder you will see this and this is an environment variable.
4. **The** UseBlazorFrameworkFiles **method**: the Blazor bootstrap process requires bunch of files, especially dotnet.wasm. This is served by the Blazor middleware, which is installed by the UseBlazorFrameworkFiles instruction.

**Middleware**: Middleware in Blazor plays a role in request processing, authentication, and overall application flow.

1. **Blazor Server Middleware**: Components or services handling server-side operations, such as authentication and routing.
2. **Blazor WebAssembly Middleware**: Components or services managing client-side operations, like HTTP requests.
3. **Authentication Middleware**: Ensures secure user access.
4. **Routing Middleware**: Manages navigation within the application.
5. **HTTP Middleware**: Handles HTTP requests and responses.

**Blazor Client Project**: if you look into the file Index.html which is located in wwwroot folder there is a div this load the application.

**The Main method**: in Main method builder.RootComponents.Add<App>("#app"); associates with the **App** component.

**The Router**: is responsible for loading the Blazor component depending on the URL address. If the rout is not fount it will display the **Not Found** message.

**A Blazor component is**: A Blazor component is a reusable unit of code in the Blazor framework, combining HTML markup with C# code. Components encapsulate UI and logic, supporting reusability, parameters for data input, event handling, and lifecycle methods. They are crucial for creating dynamic and interactive web applications.

**A Router component**: the router component is responsible for installing router components

**A Layout component**: the layout component is displaying page.

Main Layout is default component for all the components. It is located in MainLayout.razor file.

**in side MainLayout component**: other components are rendered inside MainLayout component. Components are replaced with @Body keyword.

**Bootstrapping**: In the context of a Blazor web application, bootstrapping refers to the process of initializing and starting the application. It involves setting up the necessary configurations, loading components, and preparing the application for execution. In short, it's the procedure that gets your Blazor web app up and running when a user visits the site, typically handled by the Blazor framework automatically.

**Bootstrapping in Blazor WebAssembly**: if you go the web browser`s developer tool and check Network tab it will show you that what are downloaded at the startup of the Blazor WASM, Browser will download all files that are compiled to WebAssembly.

**Examining WebAssembly by the browser**: to examine that files are downloaded by the browser, open developer tool and click Network tab, first of that clear browsers data and refresh the page, you will see that file getting downloaded.

**Files download size**: first download size will be more as can be, but after that download size will be getting decreased, because it will get files from the cache.

**Bootstrapping in Blazor Server**: Blazor Server is not download all runtime files as WebAssembly does, it just downloads less required files. You will see in developer tools>Network tab that all files that are download has size with KBs. And you will see that a WebSocket is open between the server and client.

**An apologize about Null reference types from its inventor**: Tony Hoare invented null reference type in 2009, and he apologized about this mistake, he called this billion-dollar mistake.

**The null-forgiving operator**: this is just an exclamation(!), that is appended to the variable or property name. to set the null flag to false. It means that the compiler skips whether it is null or not.

Chapter 2

# Data Binding

**Razor**: is a markup language that allows you to embed code in a template. Razor can be used to dynamically generate HTML. In ASP.NET MVC Razor generates HTML at the server side, but in Blazor WASM this action is done in the client side. And keeps page updated through user actions. Razor is used to generate a .NET class and everything in @code block is embedded in that class.

**@code** Keywork: include class methods and properties inside of that class. The class name is the razor page name by default.

1. If you want a property to be called from out side you need to add an attribute called [Parameter] to that property.

**One-way binding**: is while data come from component to DOM or data comes from DOM to component. Data from component is like displaying data, and DOM to component is like taking events from user and running code. It means the action must be from one way.

**Attribute binding**: you can bind methods or variable name to HTML attribute by just calling the method or variable name in value of an HTML attribute.

**Conditional Attributes**: if want to change value of an HTML attribute you can do it by typing (@) and adding condition in to parentheses.

**Event handling and data binding**: Blazor allows you to react DOM events, instead of using JavaScript.

**Data binding syntax**: by using @on<event name> you can use DOM`s events. This is just an HTML event just having (@) character at the starting.

**Event arguments**: in regular .NET an event handler method would have two parameters sender and EventArgs, In Blazor it is easy to that just add one parameter that is derived from EventArgs class. For example, we can use the MouseEventArgs instance to see if the Ctrl key is being pressed.

**C# Lambda functions**: you can also use lambda functions to bind an event. For example: @onclick=” () => call method”

**Two-way binding**: getting data and setting data to and from component is two-way binding.

**Two-Data binding syntax**: @bind=” @property of method name” is used to bind data and get data. And then you can operate on that data.

**Binding to other events**: you can bind to an event in while an action happens. The syntax is @bind:{event} as an example: <input type="number" @bind="@increment" @bind:event="oninput" />

**Preventing default actions**: In Blazor, preventing default actions means stopping the default behavior of a browser event. For example, preventing the default form submission behavior can be done by handling the event and calling event.PreventDefault() in the associated C# method. This allows you to handle the event programmatically.

// Events will be discussed later

**Data formatting in Blazor**: Dates can be formatted like this:

<p>

<input @bind="@Today" @bind:format="yyyy-MM-dd" />

</p>

@code {

private DateTime Today {get; set;} = DateTime.Now;

}

DateTime values are the only ones supporting the @bind:format attribute.

**Change Detection**: Blazor runtime updates the DOM when ever it detects change made to your data. It means when you click to a button and it throws an event and executes C# Code and renders UI for you. Blazor is not always capable to detect all changes, so in this case you will have to tell Blazor to make changes in the DOM. Blazor will re-render whenever an event occurs or asynchronous operation happens. Some changes cannot be detected like background thread changes. You need to tell Blazor by calling StateHasChanged method to update page, every Blazor component has this method which is inherited from its base class. When an async method completes, change detection will occur.

**DTOs (Data Transfer Objects)**: These Classes are used to send data from server to client and back from client to server.

Chapter 3

# Components and structure for Blazor application

**What is a Blazor Component**: in simple each razor file in Blazor application is component. A razor file has C# and markup combined in it. Components can be built as children in other components. Every class that is inherited from ComponentBase class is a component. In each Blazor razor file there is also a C# class that is derived from ComponentBase class.