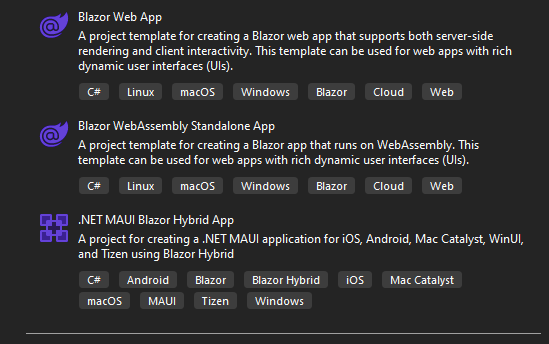
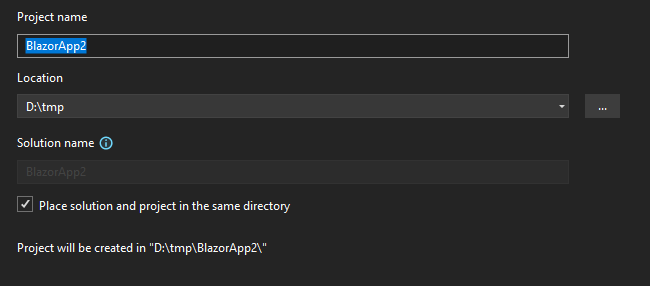
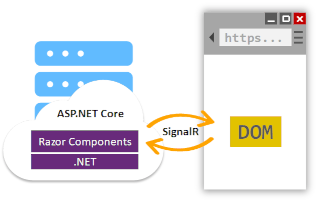
# Blazor 8

1. A whole bunch of great articles about Blazor have been published on the internet not so long ago, see the [Blazor University](https://blazor-university.com/) site. Microsoft has released a brand new version of Blazor. It has become a great WEB programming tool, but some of the statements and recommendations in the aforementioned series of articles are outdated and require corrections. I will describe here what, where and how it has changed.
2. First of all, it should be emphasized that the current version of **Blazor** requires:
3. Windows 11,
4. .NET 8
5. Visual Studio 2022 (Community Edition or any paid version).

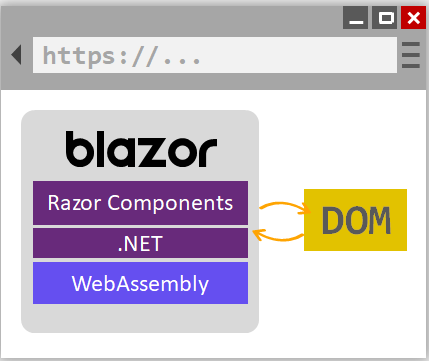
## Project templates

1. Visual Studio offers 3 different templates after creating new application:
2. 
3. **Blazor WebAssembly Standalone** application will create an application very similar to the one you had in your Windows 10 environment.
4. **.NET MAUI Blazor Hybrid App** is a Xamarin enhancement. This template creates applications that work as operating system applications and access the server directly, without a browser.
5. **Blazor Web App** is intended for the development of new WEB applications, and here the new **Blazor** variant works at full power. Visual studio asks for project parameters after selecting **Blazor Web App**. The dialog is the same as in the previous version of Blazor:
6. After setting the project parameters, the main dialog appears:
7. 
8. **Framework dropdown**: there are currently no options. More options will probably appear in this control later.
9. **Authentication type**: here you can choose **none** or **Individual Accounts**. Visual Studio will place the ..\Components\Account directory in the server part project after choosing the second option. The directory contains the files required for name/password authentication. The main menu is also expanded. We will talk about authentication and its configuration later, in a separate section.
10. **Configure for HTTPS**: this checkbox needs to be checked when you use standard authentication. The **SSL certificate** will be used by Blazor even after unchecking this control, as it is required when the page works in Server Side Rendering (SSR) mode. We'll talk about this later when we look at the different rendering modes.
11. **Interactive render mode** allows you to choose one of four options:
12. **None** - in this mode, applications will load extremely fast as there is no work required on the client, and no large WebAssembly assets to download. The server is just sending HTML that the browser then renders. This also means that each request for a new page results in a full page load. This kind of application is an enhancement of Razor pages: you can use Blazor components here, but you will have to use JavaScript for handling events.
13. **Server** - in this mode a page, or component, will be optionally pre-rendered on the server and then made interactive on the client via a **SignalR** connection. Once interactive, all events on the client will be transmitted back to the server over the SignalR connection to be processed on the server. Any updates required to the DOM will then be packaged up and sent to the client over the same SignalR connection where a small Blazor runtime will patch the updates into the DOM.
14. **WebAssembly** - this model is derived from the Blazor WebAssembly hosting model and fully capitalises on client-side capabilities, allowing C# code to run in the user’s browser. Initial data would be downloaded to the client along with the various framework DLLs and WebAssembly runtime. Once on the client, it would be bootstrapped and the page loaded. Any API calls to get data would be made and the UI would be re-rendered as necessary to display any data returned.
15. **Auto (Server and WebAssembly)** - it is main mode of Blazor applications in .NET 8. When setting a page or component to use Auto mode, the initial load of that component will be via server mode making it super fast. But in the background Blazor will download the necessary assets to the client so that on the next load it can be done using WebAssembly mode. This rendering mode will address the biggest pain point for developers when embarking on a new Blazor project, what hosting model should we use? Every component marked with **RenderMode.Auto** will need to execute on both the server and the client. Meaning that there will need to be some form of abstraction in place if the component needs to fetch any data.

### Blazor Server

1. 
2. **ASP.NET Core** apps and **Blazor Server** use the Razor language to describe HTML content for rendering, but they significantly differ in how markup is rendered.
3. When a Razor Page or view is rendered, every line of Razor code emits HTML in text form. After rendering, the server disposes of the page or view instance, including any state that was produced. When another request for the page occurs, the entire page is rerendered to HTML again and sent to the client.
4. Blazor Server produces a graph of components to display similar to an HTML or XML DOM. The component graph includes state held in properties and fields. Blazor evaluates the component graph to produce a binary representation of the markup, which is sent to the client for rendering. After the connection is made between the client and the server, the component's static prerendered elements are replaced with interactive elements. Prerendering content on the server in order to load HTML content on the client quickly makes the app feel more responsive to the client.
5. After the components are interactive on the client, UI updates are triggered by user interaction and app events. When an update occurs, the component graph is rerendered, and a UI diff (difference) is calculated. This diff is the smallest set of DOM edits required to update the UI on the client. The diff is sent to the client in a binary format and applied by the browser.

### Blazor WebAssembly

1. 
2. Running .NET code inside web browsers is made possible by **WebAssembly** (abbreviated wasm). **WebAssembly** is a compact bytecode format optimized for fast download and maximum execution speed. WebAssembly is an open web standard and supported in web browsers without plugins. WebAssembly works in all modern web browsers, including mobile browsers.
3. **WebAssembly** code can access the full functionality of the browser via JavaScript, called **JavaScript interoperability**, often shortened to JavaScript interop or JS interop. .NET code executed via WebAssembly in the browser runs in the browser's JavaScript sandbox with the protections that the sandbox provides against malicious actions on the client machine.
4. When a Blazor WebAssembly app is built and run:
5. C# code files and Razor files are compiled into .NET assemblies.
6. The assemblies and the [.NET runtime](https://learn.microsoft.com/en-us/dotnet/framework/get-started/overview) are downloaded to the browser.
7. Blazor WebAssembly bootstraps the .NET runtime and configures the runtime to load the assemblies for the app. The Blazor WebAssembly runtime uses JavaScript interop to handle DOM manipulation and browser API calls.
8. The size of the published app, its payload size, is a critical performance factor for an app's usability. A large app takes a relatively long time to download to a browser, which diminishes the user experience.
9. For apps that require third-party JavaScript libraries and access to browser APIs, components interoperate with JavaScript. Components are capable of using any library or API that JavaScript is able to use. C# code can call into JavaScript code, and JavaScript code can call into C# code.
10. Blazor implements the .NET Standard, which enables Blazor projects to reference libraries that conform to .NET Standard specifications. .NET Standard is a formal specification of .NET APIs that are common across .NET implementations. .NET Standard class libraries can be shared across different .NET platforms, such as Blazor, .NET Framework, .NET Core, Xamarin, Mono, and Unity.APIs that aren't applicable inside of a web browser (for example, accessing the file system, opening a socket, and threading) throw a **PlatformNotSupportedException**.

### Auto

1. Set these parameters on next screen:
2. **Framework** - .NET 8.x,
3. **Authentication type** – Individual accounts,
4. **Configure for HTTPS** – true,
5. **Interactive render mode** – Auto (Server and WebAssembly),
6. **Interactivity location** – Per page/component,
7. **Include sample pages** – true,
8. **Do not use top-level statements** – false.
9. Visual Studio will create two projects after setting these parameters: server and client. Here I’ll demonstrate how these two projects may be used for processing any database.
10. The ~/Data directory has the ApplicationDbContext.cs file, which contains DB context for the Entity Framework:
    1. public class ApplicationDbContext(DbContextOptions<**ApplicationDbContext**> options)
    2. : IdentityDbContext<**ApplicationUser**>(options)
    3. {
    4. }
    5. Second file in this directory is **ApplicationUser.cs**. This file defines **ApplicationUser** class:
    6. public class ApplicationUser : IdentityUser
    7. {
    8. }
11. In the generated version, this class contains only those fields that are described in the standard class **Microsoft.AspNetCore.Identity**. You can add any number of additional fields.
12. The **appsettings.json** file contains the database connection: see to the "ConnectionStrings" section. The local database is specified in the generated task:
13. "ConnectionStrings": {
14. "DefaultConnection": "Server=(localdb)\\mssqllocaldb;Database=aspnet-BlazorApp4-0c7655a7-1f05-43cd-921a-72502c56eb35;Trusted\_Connection=True;MultipleActiveResultSets=true"
15. },
16. **"Server=…**" is one long string. JSON doesn't allow breaking lines for readability.

#### MS SQL Server

1. It is enough to change the **DefaultConnection** element for connecting to the MS SQL server. Here is connection to server on my computer:
2. "ConnectionStrings": {
3. "DefaultConnection": "Data Source=GedoDell;Initial Catalog=BlazorApp4;Integrated Security=True;Encrypt=False;MultipleActiveResultSets=true"
4. },
5. The connection string can be specified in several different ways. All they are all listed in the [ConnectionStrings](https://www.connectionstrings.com/sql-server/) site.
6. [EPISODE 1 On How to Create Simple CRUD .NET 8.0 Blazor With Auto Render Magic,EF Core, SQL Server. (youtube.com)](https://www.youtube.com/watch?v=xgeoNVsIwug)