**1.Difference between the Blazor client-side and server-side and also explain web assembly and Signal R connection.**

**Ans -:**

| **Blazor Server Side** | **Blazor Web Assembly (Client Side)** |
| --- | --- |
| * It is a Server side hosting model. | * It is a client side model. |
| * It uses ASP.Net Core application and we can use entity framework to connect with SQL Databases. | * It works on browser and download the Blazor application at client side. It supports all modern browsers. |
| * Create Razor pages as client-side component. | * Create Razor pages as client-side component. |
| * Suitable for Public website as it is SEO friendly because it renders everything over page. | * Blazor Web Assembly loads only the template so all content can’t be crawled by Search Engine. |
| * Once it lost internet connection, then we can’t do anything. We can not even browse the web page. | * In Blazor Web Assembly, we can browse web pages even the connection is lost. In case it depends on any server side functionality such as API then at that point it won’t work. |

**Blazor Web Assembly -:** Blazor is an open source and cross-platform web UI framework for building single-page apps using .NET and C# instead of JavaScript. Blazor is based on a powerful and flexible component model for building rich interactive web UI. We implement Blazor UI components using a combination of .NET code and Razor syntax: an elegant melding of HTML and C#. Blazor components can seamlessly handle UI events, bind to user input, and efficiently render UI updates.

### **SignalR -:**

SignalR is an integral part of Blazor and offers these features:

* It is free, open-source and a first-class citizen of .NET Core
* Sends async messages over persistent connections.
* Connections are two-way.
* Every client has its own connection.
* Azure SignalR Services offers a free tier.
* Example applications are chat applications or the car tracking in the Uber app.

**2. What is the component and component parameter?**

**Ans -:**

**Component -:** A component is a self-contained chunk of user interface (UI), such as a page, dialog, or form. Blazor applications are created using components which are flexible, lightweight, and can be nested, reused, and shared between projects.

* A component is the base element of the Blazor application, i.e., every page is considered as a component in Blazor.
* It uses the combination of Razor, HTML and C# code as a component.

**Parameter -:** In Blazor, we can add parameters to any component which are defined using non-public properties on the component class by decorating that property with attribute.

**3. Explain the Life cycle with example.**

**Ans -:**

## **OnInit & OnInitAsync -:** The synchronous and asynchronous version of the application methods which gets executed when the component gets Initialized.

* OnInitialized is called first, then OnInitializedAsync.
* It is executed when the component is completely loaded.
* You can use this method to load data from services because each control in the UI is loaded after this method.
* It is executed when the component is ready and when it has received the values from the parent in the render tree.

**Example -:**

*Public overridevoid OnInitialized()*

*{*

*}*

**Async method for this**

*Public override async Task OnInitializedAsync()*

*{*

*await...*

*}*

## **OnParametersSet & OnParametersSetAsync**

The synchronous and asynchronous way of setting the parameter when the component receives the parameter from its parent component.

* The OnParametersSet and OnParametersSetAsync methods are called when a component is first initialised.
* After initialisation, OnParametersSet and OnParametersSetAsync are called each time new or updated parameters are receieved from the parent in the render tree.

**Example -:**

*Public overridevoid OnParameterSet()*

*{*

*}*

**Async method for this**

*Public override async Task OnParameterSetAsync()*

*{*

*await...*

*}*

## **OnAfterRender & OnAfterRenderAsync**

The synchronous and asynchronous version of the application methods to perform the additional steps like initializing the other components.

* The OnAfterRender and OnAfterRenderAsync methods are called after each render of the component.
* At the point they are called you can expect that all element and component references are populated.
* It means that if you need to perform an action, such as attaching an event listener, it requires the elements of the component to be rendered in the DOM.
* Another great use for these lifecycle methods is for JavaScript library initialization, which requires DOM elements to be in place to work.

**Example -:**

*Public overridevoid OnAfterRender(bool firstrender)*

*{*

*}*

**Async method for this:**

*Public override async Task OnAfterRenderAsync(bool firstrender)*

*{*

*await...*

*}*

**4. Explain all binding with example.**

**Ans -:**

Data binding is one of the most powerful features of software development technologies. Data binding is the connection bridge between View and the business logic (View Model) of the application. The following are the ways of doing data banding with Blazor.

* One-way Data Binding
* Two-way Data Binding
* Event Binding

**One-way Data Binding -:**

One-way data binding is also known as an interpolation in other frameworks, such as Angular. It is very similar to Razor and also it will be quite straightforward. In one-way binding, we need to pass property or variable name along with @ i.e. @Name (here Name is either the property or variable). In the following example, I have done one-way binding with variable currentTask.

**Example -:**

*@page "/databindingExample"*

*<h3>One-way Data Binding</h3>*

*<p>*

*@currentTask*

*</p>*

*@functions {*

*string currentTask = "Test One-way Data Binding!";*

*}*

**Two-way Data Binding -:**

Blazor now supports two-way binding. This can be achieved by using the "bind" attribute. The current version of Blazor supports the following types for two-way binding.

* string
* int
* DateTime
* Enum
* bool

**Example -:**

*@page "/databindingExample"*

*<h3>Two-way Data Binding</h3>*

*Enter your name: <input type="text" bind=@Name /><br />*

*<br />*

*Have you try the Blazor? <input type="checkbox" bind="IsTry" /><br />*

*<br />*

*<br />*

*<p><b>Summary</b></p>*

*You have entered:@Name <br />*

*Try Blazor: @(IsTry ? "Yes" : "No")*

*@functions {*

*public string Name { get; set; }*

*public bool IsTry { get; set; }*

*}*

**Event Binding -:**

The current version of Blazor, event binding is very limited. Currently, it only supports onclick and on change event. It is under development, so many more event support in the new version. Event binding is done by using function name along with @ i.e. @ButtonClicked (here ButtonClicked is function name).

**Example -:**

*@page "/databindingExample"*

*<h3>Event Binding</h3>*

*<br />*

*<button onclick=@ButtonClicked>Event Binding Example</button>*

*<br />*

*<br />*

*<button onclick=@(() => Console.WriteLine("Inline:Button Clicked"))>Inline Event Binding</button>*

*@functions {*

*void ButtonClicked()*

*{*

*Console.WriteLine("Button Clicked");*

*}*

*}*