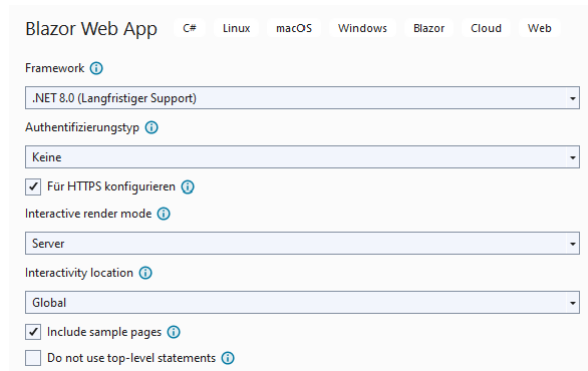


## Blazor Lab 8

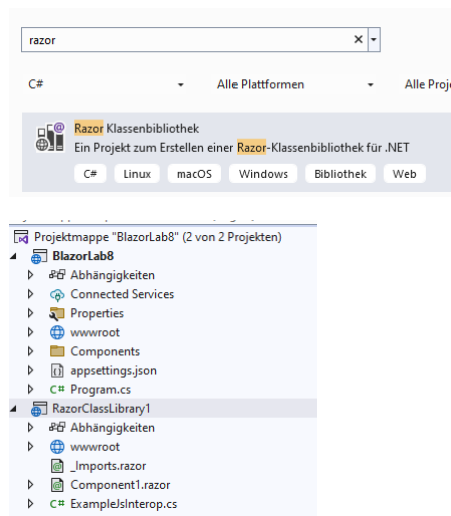
Es wird die Browser Location API genutzt. Das geht nur mit JavaScript und erfordert für den Callback erheblichen Synchronisierungsaufwand. Das Beispiel ist einfach gehalten und darf so in der Praxis nicht verwendet werden

Erstellen sie ein neues **Blazor Web App** Projekt mit Visual Studio und nennen dieses BlazorLab8



Fügen sie zur Solution eine Razor Klassenbibliothek Projekt hinzu. (Rechtsklick Solution/Projektmappe).

Im Projekttypen Razor suchen



Im Blazor Projekt den Verweis auf die Bibliothek hinzufügen- Rechtsklick Projekt – Hinzufügen - Verweis



Im Klassenbibliothek Projekt

Fügen sie eine neue Klasse Location.cs hinzu. Diese dient dazu um die Location Daten zu halten.

```
public class Location
{
    public decimal Latitude { get; set; }
    public decimal Longitude { get; set; }
    public decimal Accuracy { get; set; }
}
```

Erstellen sie im wwwroot Verzeichnis die JavaScript Datei LocationComponent.js. Damit wird die Browser API getCurrentPosition aufgerufen. Die anonyme Callback Methode wird aufgerufen, wenn die Position ermittelt werden konnte und enthält die Position. Um dann wieder in den C# Code zu kommen wird auf dem Dotnet Objekt eine Methode „receiveResponse“ angestossen, samt Parametern.

```
export function GetLocation() {
    if (navigator.geolocation) {
        navigator.geolocation.getCurrentPosition(function (position) {
            DotNet.invokeMethodAsync('RazorClassLibrary1', 'ReceiveResponse',
                position.coords.latitude, position.coords.longitude,
                position.coords.accuracy);
        });
    }
}
```

Erstellen sie nun im Root der Library die Datei LocationComponent.cs. Der nun zu erstellende Code folgt der Logik aus der Projekt Datei ExampleJSInterop.cs. Wenn sie das nicht tippen wollen, können sie Teile daraus kopieren und dann adaptieren.

Grau markiertes muss geändert/ergänzt werden im Vergleich zur Datei ExampleJSInterop

```
public class LocationComponent
{
    private readonly Lazy<Task<IJSObjectReference>> moduleTask;
    public static event Action<Location> OnPosition;

    public LocationComponent(IJSRuntime jsRuntime)
    {
        moduleTask = new(() => jsRuntime.InvokeAsync<IJSObjectReference>(
            "import", "./_content/RazorClassLibrary1/LocationComponent.js").AsTask());
    }
    public async void GetLocationAsync()
    {
        var module = await moduleTask.Value;
        await module.InvokeAsync<Location>("GetLocation");
    }
}
```

Fügen sie weiters in die Datei LocationComponent.cs folgende Funktion hinzu, die von JavaScript, nach erfolgreicher Positionsbestimmung aufgerufen werden soll

```
[JSInvokable]
public static void ReceiveResponse(
    decimal latitude,
    decimal longitude,
    decimal accuracy)
```

```

    {
        OnPosition?.Invoke(new Location
        {
            Latitude = Convert.ToDecimal(latitude),
            Longitude = Convert.ToDecimal(longitude),
            Accuracy = Convert.ToDecimal(accuracy)
        });
    }
}

```

Falls Visual Studio im Code Teile rot unterwellt prüfen sie ob der Namensraum eingefügt ist und ergänzen sie ggf

```
using Microsoft.JSInterop;
```

## Das Blazor Projekt

Folgender Code wird nun im Blazor Projekt editiert. Zunächst ergänzen sie den DI Service Container in der Datei programm.cs

```
Builder.services.AddScoped<LocationComponent>();
```

Editieren Sie die Datei Index.razor aus dem Pages Verzeichnis. In der ersten Zeile fügen Sie per DI die Referenz hinzu

```
@inject LocationComponent LocationService
```

Als nächstes ergänzen sie den HTML Teil und Code Logik in der gleichen Datei

```
<button @onclick="getGPS">wohinich</button>
```

```

Latitude @loc?.Latitude
Longitude @loc?.Longitude
Accuracy @loc?.Accuracy

```

```

@code
{
    Location loc;
    async void getGPS()
    {
        LocationComponent.OnPosition += OnGetPos;
        LocationService.GetLocationAsync();
    }
    void OnGetPos(Location _loc)
    {
        loc = _loc;
        InvokeAsync(StateHasChanged);
    }
}

```