IlmuKomputer.Com
Ikatlah ilmu dengan menuliskannya

JavaScript Interop Pada Aplikasi Blazor – Part 3

Junindar, ST, MCPD, MOS, MCT, MVP

junindar@gmail.com

http://junindar.blogspot.com

Abstrak

Blazor adalah Web Framework yang bersifat Open Source dimana aplikasi Web yang bersifat client-side interactive dapat dikembangkan dengan menggunakan .Net (C#) dan HTML. Pada saat ini C# biasa digunakan untuk melakukan proses back-end dari aplikasi web. Dengan menggunakan fitur baru dari ASP.NET Core yaitu Blazor, kita dapat membangun interactive WEB dengan menggunakan C# dan .NET .

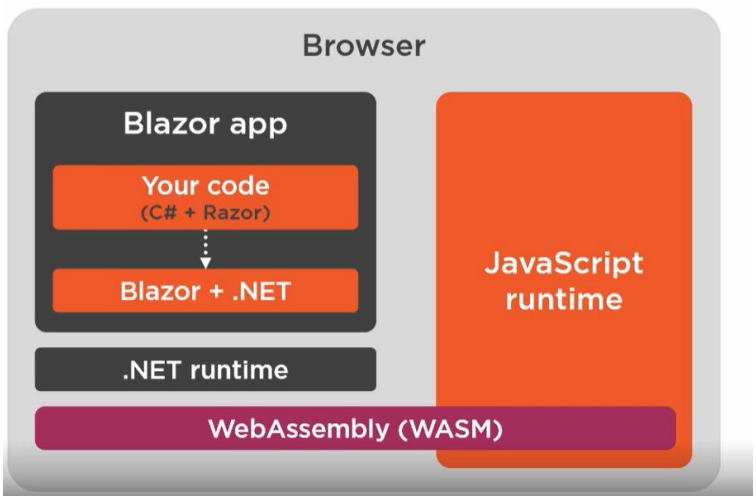
Code .Net berjalan pada WebAssembly, yang artinya kita dapat menjalankan “NET“ didalam browser (Client) tanpa harus menginstall plugin seperti Silverlight, Java mapun Flash.

Pendahuluan

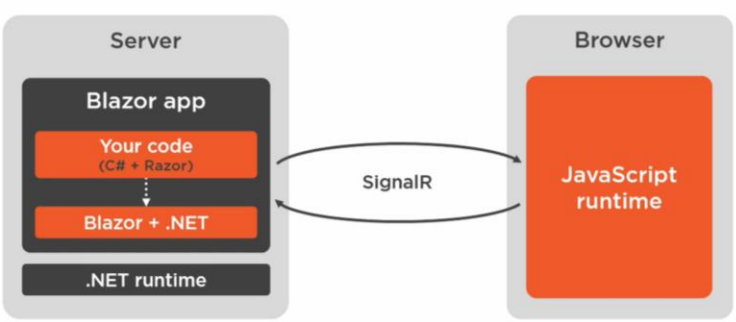
Untuk membuat aplikasi pada Blazor, kita menggunakan C# dan Razor. Razor merupakan kombinasi dari HTML dan C#. Dan output dari blazor aplikasi di eksekusi oleh .Net runtime.

Seperti kita ketahui, terdapat dua model hosting pada aplikasi blazor, yang pertama WebAssembly dan yang kedua adalah Server.

Untuk WebAssembly aplikasi dan .Net runtime berjalan pada sisi client didalam web browser. .Net runtime yang digunakan pada browser berdasarkan WebAssembly atau yang biasa disebut WASM. WASM adalah instruksi berformat binary yang dieksekusi Javascript runtime didalam browser. Jadi ini merupakan cara kerja dari Client Side hosting model pada blazor.

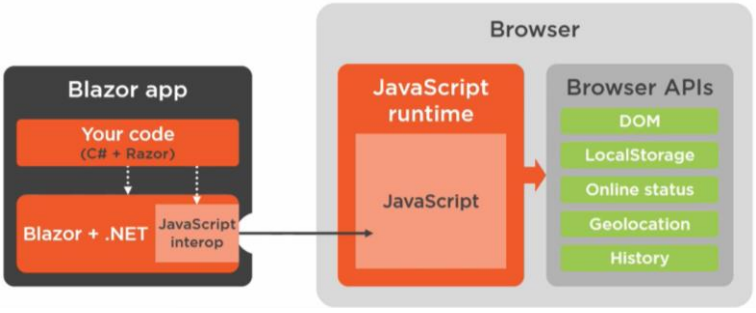


Kita dapat juga menjalankan blazor tanpa menggunakan WebAssembly, yaitu dengan menggunakan Server-Side hosting model. Yang artinya aplikasi blazor tidak dijalankan didalam browser, tetapi oleh server. Untuk melakukan render user interface pada browser, aplikasi berkomunikasi melalui SignalR dengan JavaScript runtime.



Browser juga memiliki browser API yang berbeda-beda, seperti Document Object Model (DOM). Dengan menggunkan DOM kita dapat mengkases dan mengganti elemen HTML pada aplikasi web. Browser API, seperti DOM ini dapat diakses dengan menggunakan JavaScript Runtime. Yang perlu diketahui, tanpa JavaScript Interop kita hanya dapat menggunakan fungsi yang hanya disediakan oleh Blazor Framework dan .Net. Lalu bagaimana jika kita ingin mengakses browser API dari code yang tidak disediakan oleh Blazor Framework? Untuk hal ini kita perlu memanggil code pada JavaScript yang akan mengakses Browser Api.

Blazor mendukung JavaScript Interoperability (JavaScript Interop), dimana kita dapat mengakses code pada JavaScript. Dari sini dapat kita ketahui, kapan kita harus menggunakan JavaScript pada aplikasi Blazor. Dimana jika aplikasi kita menggunakan fungsi-fungsi Browser API seperti DOM, Local Storage, Online Status yang tidak disediakan oleh Blazor Framework.



Pada dua artikel sebelumnya telah kita bahas bagaimana memanggil fungsi pada javascript dari .Net. Lalu bagaimana jika dibalik yaitu memanggil .Net Method dari Javascript, apakah bisa ? Nah, artikel kali ini akan membahas pertanyaan tersebut.

Terdapat dua pembahasan pada artikel ini, yang pertama adalah cara memanggil .Net static method sedangkan yang kedua memanggil .Net Instance method dari JavaScript.

- Memanggil .Net Static Method

  Untuk memanggil static method dari javascript, kita gunakan fungsi DotNet.invokeMethod atau DotNet.invokeMethodAsync. Nilai dari DotNet.invokeMethod adalah hasil dari operasi pada static method. Sedangkan nilai dari DotNet.invokeMethodAsync adalah JavaScript Promise yang merepresentasikan hasil dari operasi.

  Terdapat beberapa paramater yang harus kita kirimkan seperti nama assembly, nama method dan parameter-parameter yang diperlukan, seperti contoh dibawah ini.

  DotNet.invokeMethodAsync('{ASSEMBLY NAME}', '{.NET METHOD ID}', {ARGUMENTS});

  Yang perlu diperhatikan pada .Net static method adalah harus bersifat public dan memiliki atribut [JSInvokable].

  Agar lebih memahami isi artikel, ikuti langkah-langkah dibawah ini.

  Buat sebuah static method seperti dibawah ini, dimana pada method ini retun value nya adalah integer dan memiliki sebuah parameter yang nilainya akan dikirimkan dari javascript.

```
[JSInvokable]
public static int Calculate(int val1)
{
   int iLocalVal = 4;
   return val1 * iLocalVal;
}
```

  Buat sebuah funsgi pada javascript seperti dibawah ini.

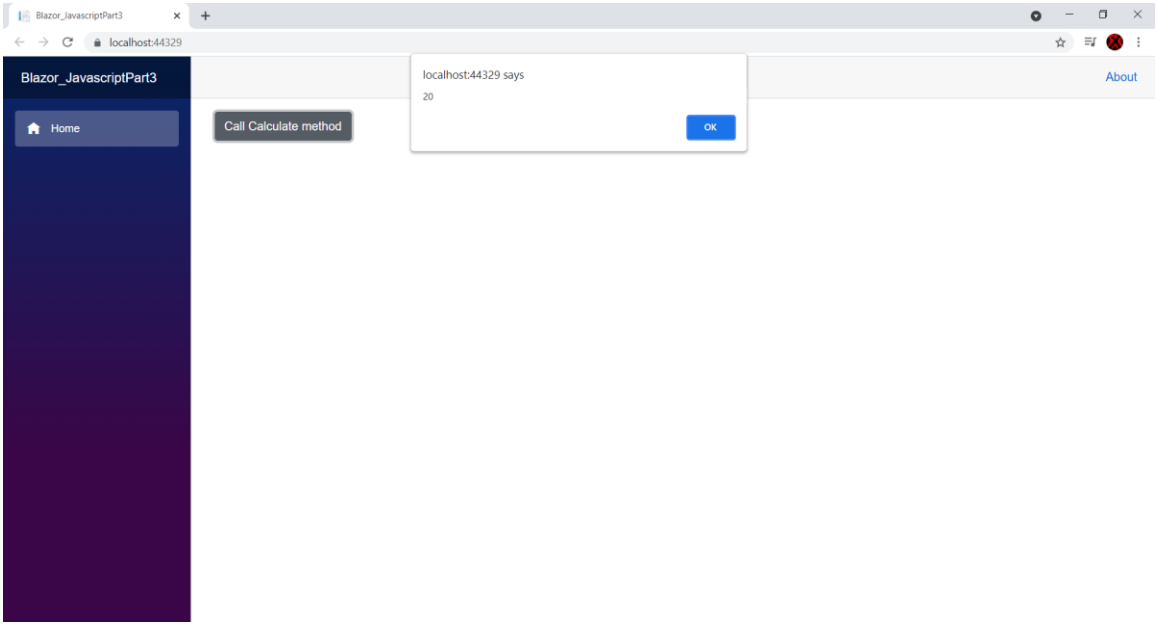
```
blazorInterop.methodCalculate = function () {
    var promise = DotNet.invokeMethodAsync("Blazor_JavascriptPart3",
        "Calculate", 5);
    promise.then(result => alert(result));
};
```

  Blazor_JavascriptPart3 adalah nama assembly atau project. Calculate merupakan nama static method yang dipanggil dan 5 nilai yang akan dikirimkan ke static method. Dan hasil dari operasi pada static method akan ditampilkan pada message box.

Lalu tambahkan sintaks HTML seperti dibawah. Dengan menggunakan event onclick dari button, kita panggil fungsi javascript yang telah dibuat.

```
<h2>Call a static .NET</h2>
<button class="btn btn-secondary" onclick="blazorInterop.methodCalculate()">
    Call Calculate method
</button>
```

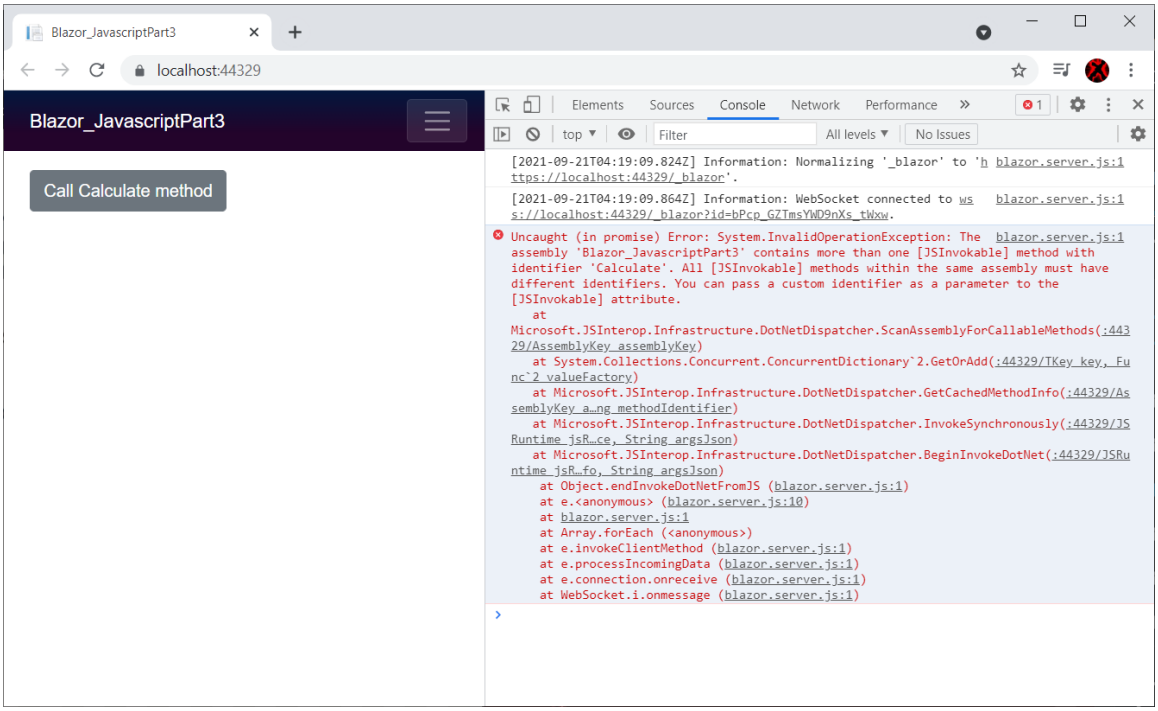
Jalankan program dan pastikan mendapatkan hasil seperti gambar dibawah ini.



Setelah berhasil dengan latihan diatas, kita lanjutkan dengan latihan menggunakan Overload Method. Bagaimana jika kita memiliki static method dengan nama yang sama tetapi memiliki argument yang berbeda atau yang biasa kita kenal “Overload Method”.

```
[JSInvokable]
public static int Calculate(int val1, int val2)
{
    int iLocalVal = 4;
    return val1 * val2* iLocalVal;
}
```

Sebagai contoh pada sintaks diatas ini, sebelumnya kita telah membuat sebuah static method dengan nama “Calculate” dimana paramater-nya berjumlah satu. Sedangkan pada sintaks diatas, memiliki nama yang sama (Calculate) tetapi jumlah paramater-nya adalah dua. Dan coba jalankan program lalu klik button “Call Calculate method”. Pada console di browser akan tedapat error yang menginformasikan bahwa terdapat lebih dari satu method dengan nama “Calculate”.



Untuk mengatasi masalah tersebut, kita perlu membuat custom indentifier pada method yang kedua, seperti pada sintaks dibawah.

```
[JSInvokable("CalculateWithVal2")]
```

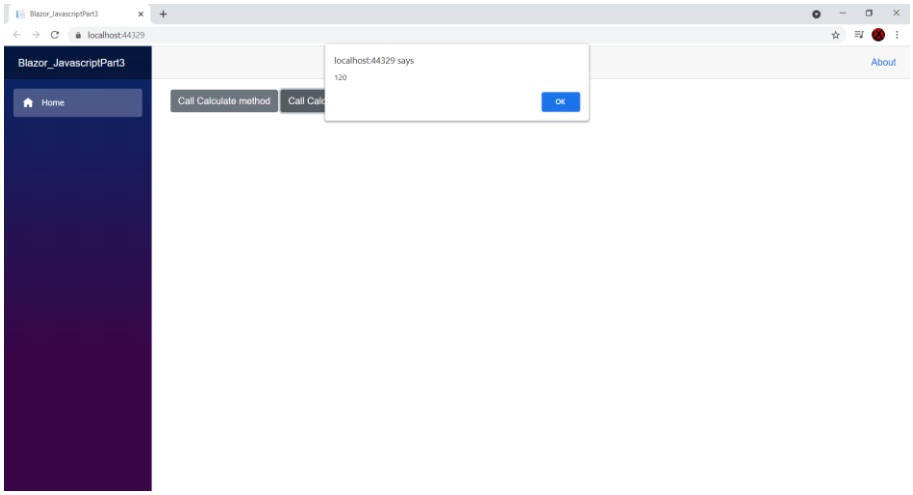
Jika kita membuat custom indentifier seperti diatas, pastikan untuk tidak memanggil nama dari method tersebut, tetapi panggil nama/id dari custom indentifier itu saja seperti pada sintaks dibawah.

```
blazorInterop.methodCalculateCustomIdentifier = function () {
    var promise = DotNet.invokeMethodAsync("Blazor_JavascriptPart3",
        "CalculateWithVal2", 5, 6);
    promise.then(result => alert(result));
};
```

Selanjutnya tambahkan sintaks HTML seperti dibawah ini.

```
<button class="btn btn-secondary"
onclick="blazorInterop.methodCalculateCustomIdentifier()">
    Call Calculate method overload
</button>
```

Jalankan program lalu klik button kedua pada halaman web dan pastikan mendapatkan hasil seperti gambar dibawah.



- Memanggil .Net Instance Method

  Apa itu instance method? Instance method biasa juga dikenal dengan non-static method. Berikut perbedaan static dan non static method.

Static Method	Instance Method (Non-Static)
Menggunakan keyword "static"	Tidak menggunakan keyword "static"
Tidak dapat menggunakan keyword "this"	Dapat menggunakan keyword "this"
Menggunakan memory dari class	Menggunkan memero dari object.

Pada Blazor APP, cara untuk memanggil .Net instance method juga berbeda. Kita tidak bisa secara langsung mengirimkan instance object ke javascript. Kita harus mengirimkan referensi dari object terlebih dahulu. Dan diisini kita menggunakan "DotNetObjectReference" class.

```
private string name;
private string result;

public async Task instanceMethod1()
{
   var  objRef = DotNetObjectReference.Create(this);
   result = await JsRuntime.InvokeAsync<string>("blazorInterop.sayHello1",
   objRef);
}

[JSInvokable]
public string GetHelloMessage()
{
   return $"Hello, {name}!";
}
```

Dapat kita lihat pada sintaks diatas, terdapat sebuah method dengan nama instanceMethod1. Pada method ini kita panggil salah satu fungsi pada javascript (blazorInterop.sayHello1) dengan mengirimkan "DotNetObjectReference" dari component.

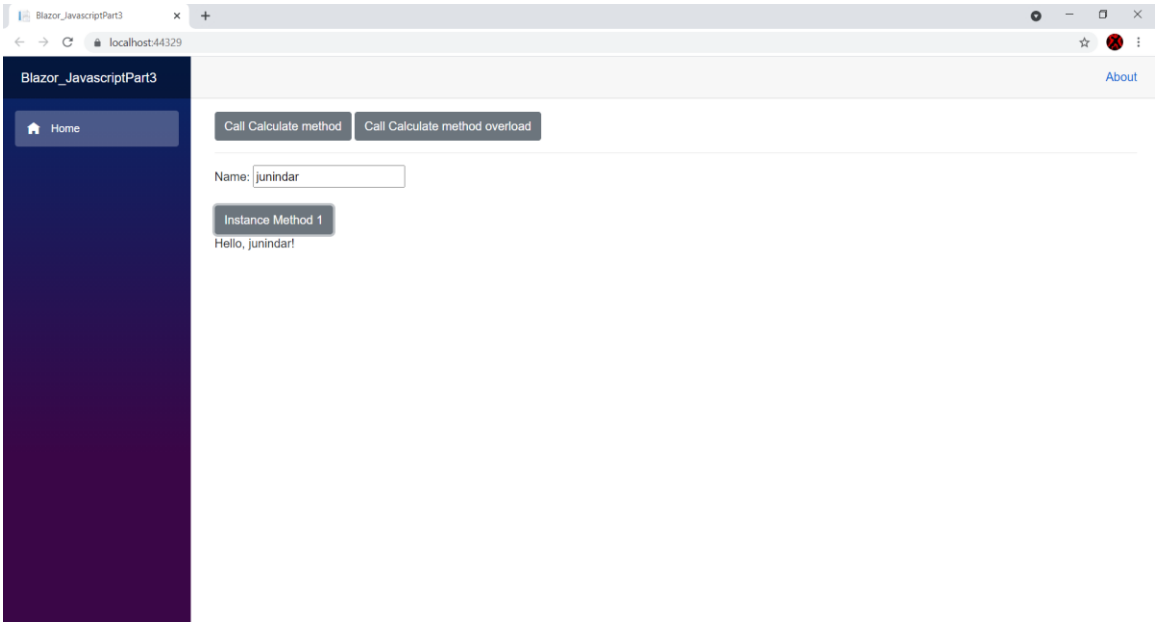
Selanjutnya terdapat sebuah method lagi dengan nama "GetHelloMessage". Nah method ini lah yang nantinya akan dipanggil dari javascript, detail sintaks nya dapat dilihat seperti dibawah.

```
blazorInterop.sayHello1 = (dotNetHelper) => {
    return dotNetHelper.invokeMethodAsync('GetHelloMessage');
};
```

Lalu tambahkan sintaks HTML seperti berikut

```
<p>
    <label>
        Name: <input @bind="name" />
    </label>
</p>
<p>
    <button @onclick="instanceMethod1">
      Instance Method 1
    </button>
</p>
<p>
    @result
</p>
```

Jalankan program untuk melihat hasilnya, pastikan seperti pada gambar dibawah ini. Teks yang kita ketikkan pada textbox akan muncul pada halaman web pada saat button "Instance Method 1" diklik.



Setelah selesai dengan latihan diatas, selanjutnya kita lanjutkan dengan latihan untuk mendapatkan width dan height dari halaman web dengan menggunakan instance method. Pada latihan ini kita akan menggunakan namespace "System.Drawing", pastikan kita import namespace tersebut pada blazor component (@using System.Drawing).

```
private Size _windowSize;

[JSInvokable]
public void SetWindowSize(Size windowSize)
{
     _windowSize = windowSize;
     StateHasChanged();
}

private async Task SendDotNetInstanceToJavaScript()
{
     var dotNetObjectReference = DotNetObjectReference.Create(this);
      await JsRuntime.InvokeVoidAsync("blazorInterop.callDotNetSetWindowSizeMethod",
                dotNetObjectReference);
}
```

Ketikkan sintaks diatas pada code block. Terdapat field dengan tipe "Size" dengan nama "_windowSize". Dan dibawahnya terdapat sebuah instance method dengan mana "SetWindowSize", method ini juga memiliki parameter dengan tipe "Size". Paramater ini yang akan dikirimkan oleh fungsi dari javascript. Untuk melakukan pre-render pada component, kita panggil method "StateHasChanged()". Lalu dilanjutkan dengan membuat async method untuk mengirimkan "DotNetObjectReference" dari component, dengan melakukan invoke fungsi di javascript.

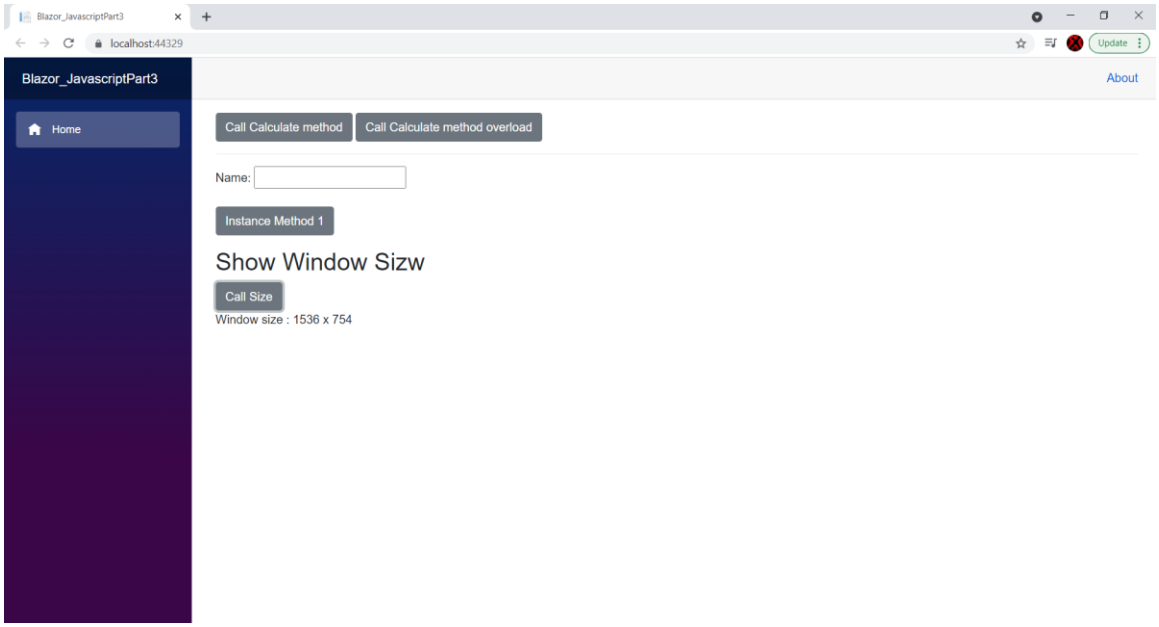
Selanjutnya kita akan membuat fungsi pada javascript seperti dibawah.

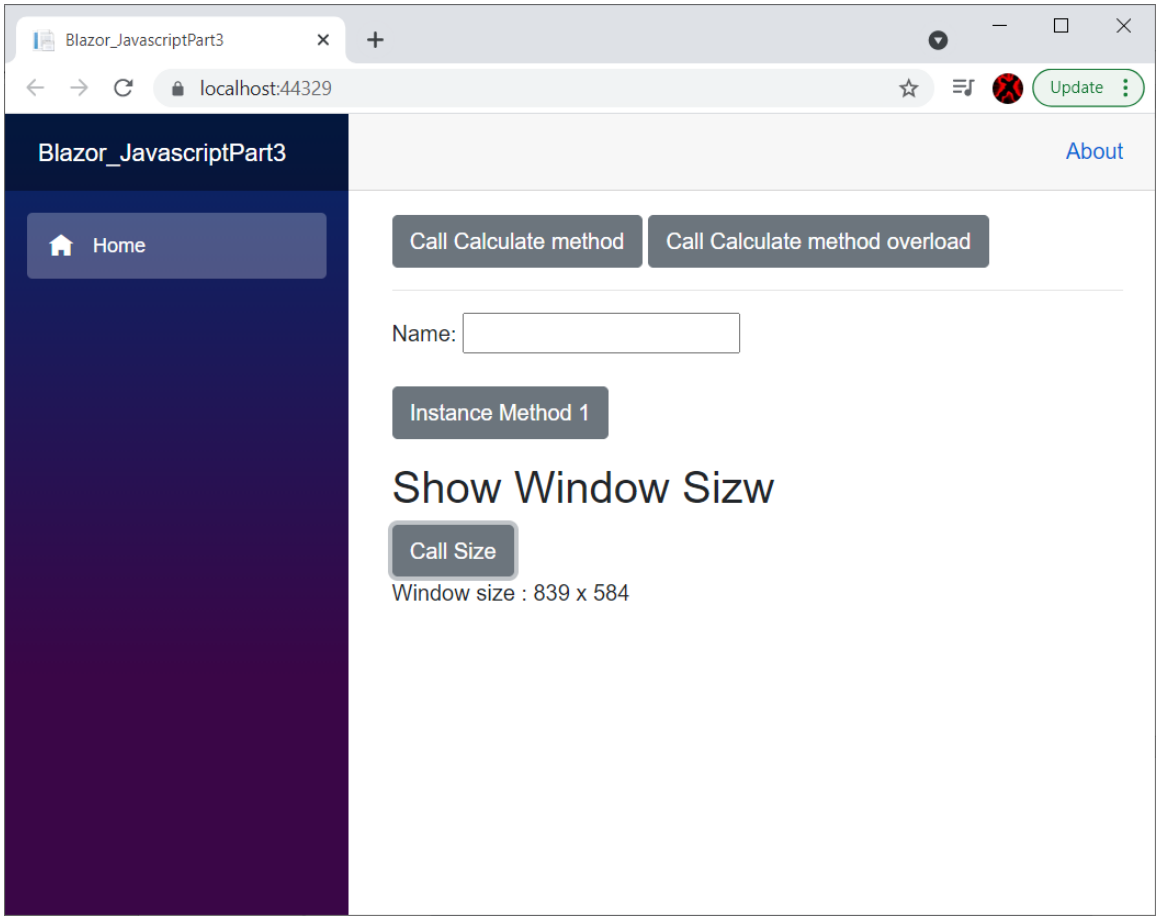
```
blazorInterop.callDotNetSetWindowSizeMethod = function (dotNetObjectRef) {
    dotNetObjectRef.invokeMethodAsync("SetWindowSize",
        {
            width: window.innerWidth,
            height: window.innerHeight
        });
};
```

Dilanjutkan dengan membuat sintaks HTML seperti dibawah ini,

```
<div>
    <h2>Show Window Sizw</h2>
    <button class="btn btn-secondary" @onclick="SendDotNetInstanceToJavaScript">Call Size</button>
    <div>Window size : @_windowSize.Width x @_windowSize.Height</div>
</div>
```

Jalankan program dan klik button "Call Size", dan coba resize halaman web browser lalu klik kembali button "Call Size" pastikan mendapatkan nilai yang berbeda, seperti pada gambar dibawah ini.


Blazor_JavascriptPart3
localhost:44329
Update
Blazor_JavascriptPart3
About
Home
Call Calculate method
Call Calculate method overload
Name:
Instance Method 1
Show Window Sizw
Call Size
Window size : 1536 x 754


Blazor_JavascriptPart3
localhost:44329
Update
Blazor_JavascriptPart3
About
Home
Call Calculate method
Call Calculate method overload
Name:
Instance Method 1
Show Window Sizw
Call Size
Window size : 839 x 584

Untuk latihan terakhir pada artikel ini, kita akan membuat event hadler pada javascript untuk memanggil instance method. Sebagai contoh pada latihan sebelumnya untuk mendapatkan height dan width dari halaman web kita harus menekan button “Call Size“ terlebih dahulu, lalu bagaimana jika untuk mendapatkan nilai-nilai tersebut dilakukan secara otomatis yaitu pada saat browser diubah size-nya. Untuk kita perlu men-register event handler, untuk lebih memahami buat sebuah function javascript seperti dibawah.

```
blazorInterop.registerResizeHandler = function (dotNetObjectRef) {
    function resizeHandler() {
        dotNetObjectRef.invokeMethodAsync("SetWindowSize",
            {
                width: window.innerWidth,
                height: window.innerHeight
            });
    };

    // Set up initial values
    resizeHandler();

    // Register event handler
    window.addEventListener("resize", resizeHandler);
};
```

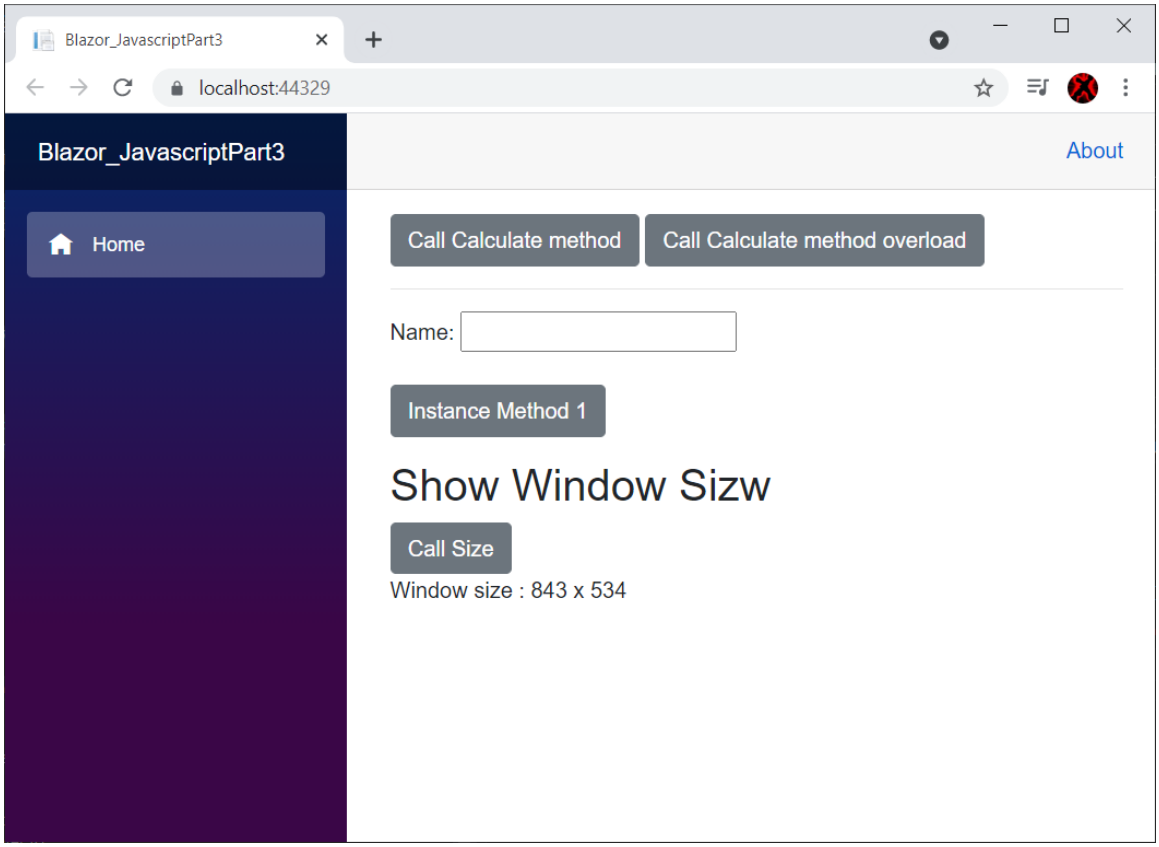
Sebenarnya function diatas sintaksnya hampir sama seperti pada latihan sebelumnya (`callDotNetSetWindowSizeMethod`). Hanya pada function diatas kita tambahkan sebuah function baru dengan nama “resizeHandler“. Lalu kita panggil function “resizeHandler“ untuk mengatur initial value. Selanjutnya untuk memanggil function “resizeHandler“ pada saat setiap kali browser diubah ukuran (size) nya, maka kita gunakan window “addEventListerner“ method.

Setelah selesai dengan javascript, kita kembali ke blazor component dengan menambahkan sintaks dibawah ini.

```
protected override async Task OnAfterRenderAsync(bool firstRender)
{
   if (firstRender)
    {
          var dotNetObjectReference = DotNetObjectReference.Create(this);
          await JsRuntime.InvokeVoidAsync("blazorInterop.registerResizeHandler",
                    dotNetObjectReference);
    }
}
```

Agar kita tidak memanggil javascript function setiap pada saat component di-render, kita hanya perlu sekali saja untuk me-register “resizeHandler“ oleh karena itu kita gunakan parameter “firstRender“.

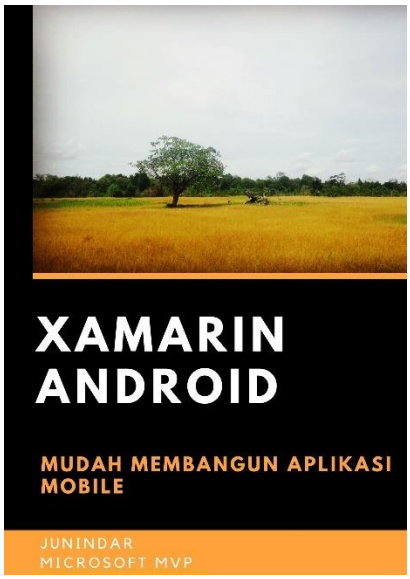
Jalankan program untuk melihat hasilnya, pastikan nilai height dan width berubah setiap kali browser diubah size nya, tanpa perlu melakukan click pada button “Call Size“.


Blazor_JavascriptPart3
localhost:44329
Blazor_JavascriptPart3
About
Home
Call Calculate method
Call Calculate method overload
Name:
Instance Method 1
Show Window Sizw
Call Size
Window size : 843 x 534

Penutup

Sedangkan untuk memudahkan dalam memahami isi artikel, maka penulis juga menyertakan dengan full source code project latihan ini, dan dapat di download disini http://junindar.blogspot.com/2021/10/invoking-javascript-dari-net-pada_16.html

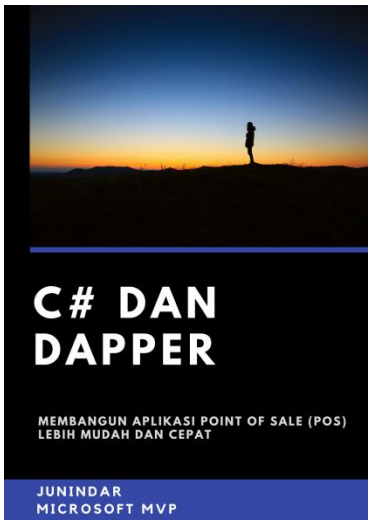
Referensi



https://play.google.com/store/books/details?id=G4tFDgAAQBAJ

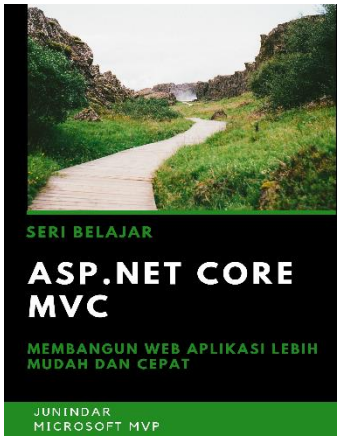
https://play.google.com/store/books/details?id=VSLiDQAAQBAJ

https://play.google.com/store/books/details/Junindar_Xamarin_Forms?id=6Wg-DwAAQBAJ

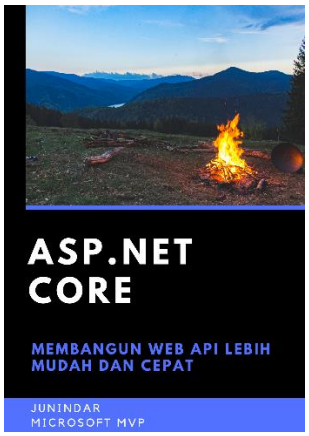


https://play.google.com/store/books/details/Junindar_C_dan_Dapper_Membangun_Aplikasi_POS_Point?id=6TErDwAAQBAJ

https://play.google.com/store/books/details/Junindar_ASP_NET_MVC_Membangun_Aplikasi_Web_Lebih?id=XLlyDwAAQBAJ



https://play.google.com/store/books/details/Junindar_ASP_NET_CORE_MVC?id=xEe5DwAAQBAJ



https://play.google.com/store/books/details/Junindar_ASP_NET_CORE?id=COUWEAAAQBAJ

Biografi Penulis.

Junindar Lahir di Tanjung Pinang, 21 Juni 1982. Menyelesaikan Program S1 pada jurusan Teknik Inscreenatika di Sekolah Tinggi Sains dan Teknologi Indonesia (ST-INTEN-Bandung). Junindar mendapatkan Award Microsoft MVP VB pertanggal 1 oktober 2009 hingga saat ini. Senang mengutak-atik computer yang berkaitan dengan bahasa pemrograman. Keahlian, sedikit mengerti beberapa bahasa pemrograman seperti : VB.Net, C#, SharePoint, ASP.NET, VBA. Reporting: Crystal Report dan Report Builder. Database: MS Access, MY SQL dan SQL Server. Simulation / Modeling Packages: Visio Enterprise, Rational Rose dan Power Designer. Dan senang bermain gitar, karena untuk bisa menjadi pemain gitar dan seorang programmer sama-sama membutuhkan seni. Pada saat ini bekerja di salah satu Perusahaan Consulting dan Project Management di Malaysia sebagai Senior Consultant. Memiliki beberapa sertifikasi dari Microsoft yaitu Microsoft Certified Professional Developer (MCPD – SharePoint 2010), MOS (Microsoft Office Specialist) dan MCT (Microsoft Certified Trainer) Mempunyai moto hidup: "**Jauh lebih baik menjadi Orang Bodoh yang giat belajar, dari pada orang Pintar yang tidak pernah mengimplementasikan ilmunya".**