Invoking JavaScript Dari .NET Pada Blazor – Part 1

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 .

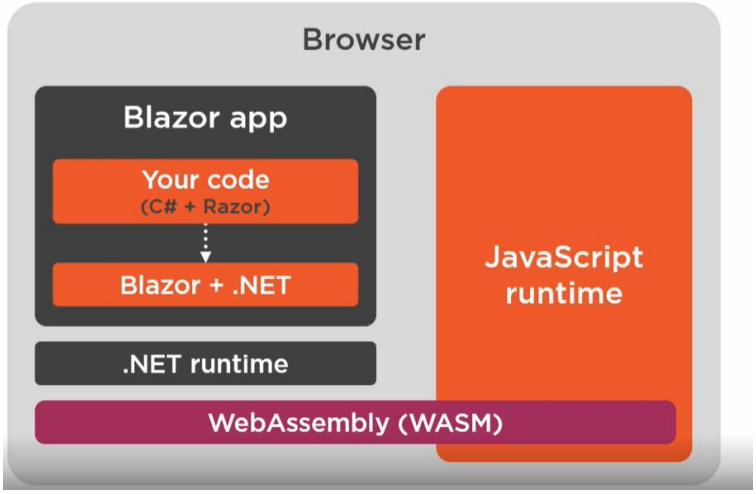
Code .Net berjalan pada WebAssembly, yang artinya kita dapat menjalankan “NET“ didalam browser (Client) tanpa harus menginstall plugin seperti Silverlight, Java mapun Flash.

Pendahuluan

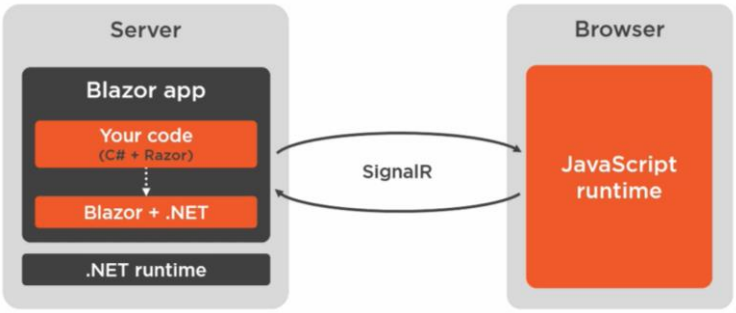
Untuk membuat aplikasi pada Blazor, kita menggunakan C# dan Razor. Razor merupakan kombinasi dari HTML dan C#. Dan output dari blazor aplikasi di eksekusi oleh .Net runtime.

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.



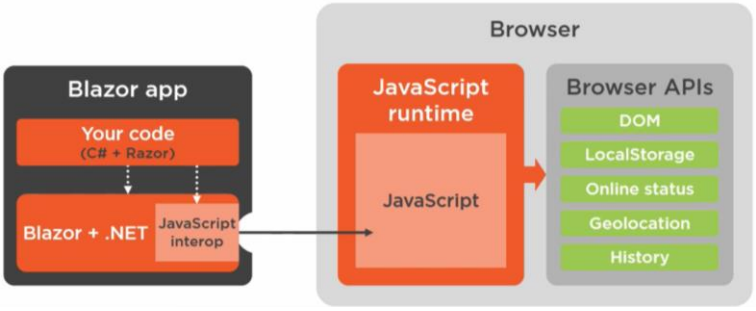
Kita dapat juga menjalankan blazor tanpa menggunakan WebAssembly, yaitu dengan menggunakan Server-Side hosting model. Yang artinya aplikasi blazor tidak dijalankan didalam browser, tetapi oleh server. Untuk melakukan render user interface pada browser, aplikasi berkomunikasi melalui SignalR dengan JavaScript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



Untuk memudahkan dalam memahami artikel ini, kita buat terlebih dahulu sebuah project Blazor App. Pada artikel ini kita akan membahas bagaimana memanggil JavaScript pada Blazor App.

Langkah pertama yang akan kita lakukan adalah, dengan menambah sebuah JavaScript file pada Blazor App.

- Pada "wwwroot", tambahkan sebuah folder dengan nama "scripts".
- Selanjutnya tambahkan sebuah file javascript dengan nama "blazorInterop.js" kedalam folder diatas.
- Lalu tambahkan sebuah Blazor Component dengan nama "InvokeJS.razor"

Untuk latihan yang pertama ini kita akan memanggil fungsi javascript dari Blazor App. Buka kembali file "blazorInterop.js" dan ketikkan sintaks seperti dibawah ini.

```
var blazorInterop = blazorInterop || {};

blazorInterop.showAlert=function(message) {
    alert(message);
};
```

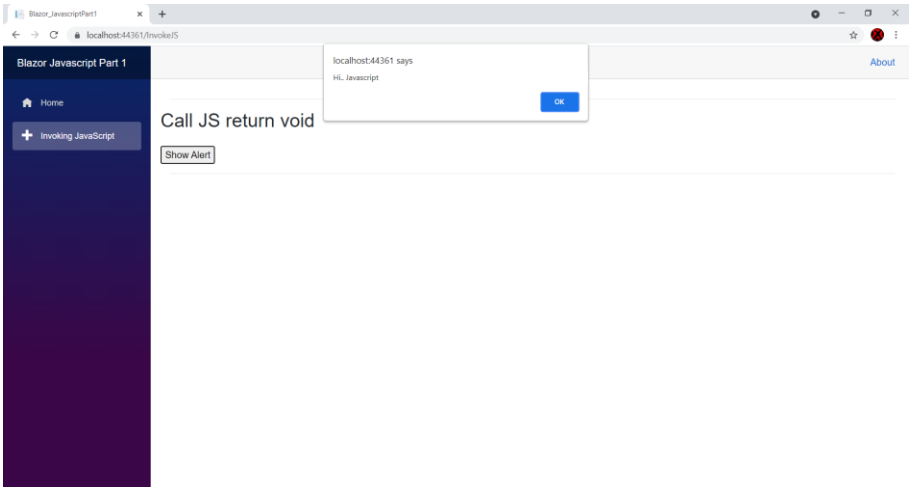
Pada sintaks diatas terdapat sebuah *function global scope* dengan nama "showAlert", dimana fungsinya untuk menampilkan message box pada browser. Untuk nilai/value dari alert-nya akan dikirimkan dari Blazor App. Selanjutnya agar fungsi-fungsi pada "blazorInterop.js" diatas dapat digunakan, buka file "_Host.cshtml" pada folder page. Dan tambahkan sintaks javascript reference, seperti dibawah.

```
<script src="~/scripts/blazorInterop.js"></script>
```

Lalu langkah selanjutnya adalah, bagaimana caranya memanggil sintaks javascript diatas, dengan menggunakan .Net. Sekarang kita kembali ke blazor component "InvokeJS.razor". dan ketikkan sintaks seperti dibawah ini.

```
@inject IJSRuntime JsRuntime
@page "/InvokeJS"

<div>
    <hr />
    <div class="form-group row">
    <h2>
            Call JS return void

    </h2>
    </div>
        <div class="form-group row">
            <button cla="btn btn-secondary" @onclick="ShowAlert">Show Alert </button>
        </div>

    <hr />
</div>
@code{
        private async Task ShowAlert()
        {
                await JsRuntime.InvokeVoidAsync("blazorInterop.showAlert", "Hi.. Javascript");
        }
}
```

Pada baris pertama terdapat sintaks untuk "Inject" IJSRuntime. Yang nantinya digunakan untuk memanggil fungsi pada javascript. Untuk sintaks HTML terdapat sebuah button "Show Alert", dimana pada atribut "onclick" memanggil fungsi "ShowAlert". Sintaks "ShowAlert" yang dipanggil pada button diatas terdapat pada block "code". Pada fungsi tersebut, kita gunakan "JsRuntime" untuk memanggil method "InvokeVoidAsync". Pada parameter pertama adalah nama dari fungsi pada javascript yang akan digunakan, sedangkan yang kedua adalah isi pesan yang akan tampil pada message box/alert. Jalankan program untuk melihat hasilnya. Klik button "Show Alert", pastikan alert/message box muncul seperti gambat dibawah.



Untuk latihan yang kedua, kita akan membuat pesan keluar pada alert berdasarkan text yang terdapat pada textbox.

Tambahkan dua buah control (TextBox dan button) seperti pada sintaks HTML dibawah ini.

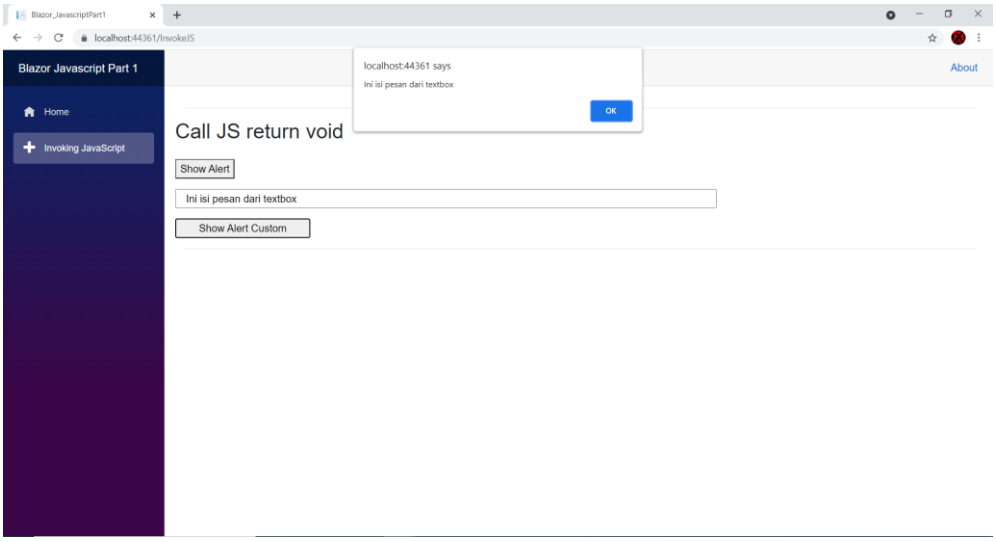
```
<div class="form-group row">
        <input  type="text" placeholder="Ketikkan Isi Pesan"  class="col-sm-8"
                @bind="@CustomMessage"
         @oninput="@((e) => { CustomMessage = (string) e.Value; })"/>

</div>
<div class="form-group row">
        <button cla="btn btn-secondary" @onclick="ShowAlertCustom"
        class="col-sm-2">Show Alert Custom </button>
</div>
```

Pada textbox perhatikan atribut “bind” dan “oninput”, untuk atribut bind valuenya adalah “CustomMessage”, sehingga kita perlu membuat sebuah varible pada “CustomMessage” pada block Code (Sintaks .NET), seperti dibawah. Sedangkan untuk atribute oninput digunakan untuk mengeset nilai dari CustomMessage berdasarkan nilai dari TextBox.

Pada button atribut onclick menggunakan method “ShowAlertCustom”.

```
private string CustomMessage { get; set; }
private async Task ShowAlertCustom()
{
     await JsRuntime.InvokeVoidAsync("blazorInterop.showAlert", CustomMessage);
}
```

Pada dasarnya sintaks “ShowAlertCustom” sama dengan “ShowAlert” yang telah kita buat sebelumnya. Perbedaannya untuk method ini nilai dari pesan-nya dinamis. Sesuai dengan nilai/text yang terdapat pada textbox. Jalankan program untuk melihat hasilnya, pastikan seperti pada gambar dibawah.



Latihan selanjutnya adalah memanggil javascript function yang memiliki return value. Dimana return value nya akan ditampilkan pada halaman web Blazor App. Ikuti langkah-langkah berikut.

Buat sebuah fungsi pada javascript untuk menampilkan input box, seperti dibawah ini.

```
blazorInterop.showPrompt = function (message,defaultValue) {
    return prompt(message, defaultValue);
};
```

Terdapat 2 buah parameter (message,defaultValue), jika parameter “message” bernilai kosong atau empty, maka nilai dari “defaultValue” yang akan digunakan.

Selanjutnya Tambahkan sintak HTML seperti dibawah.

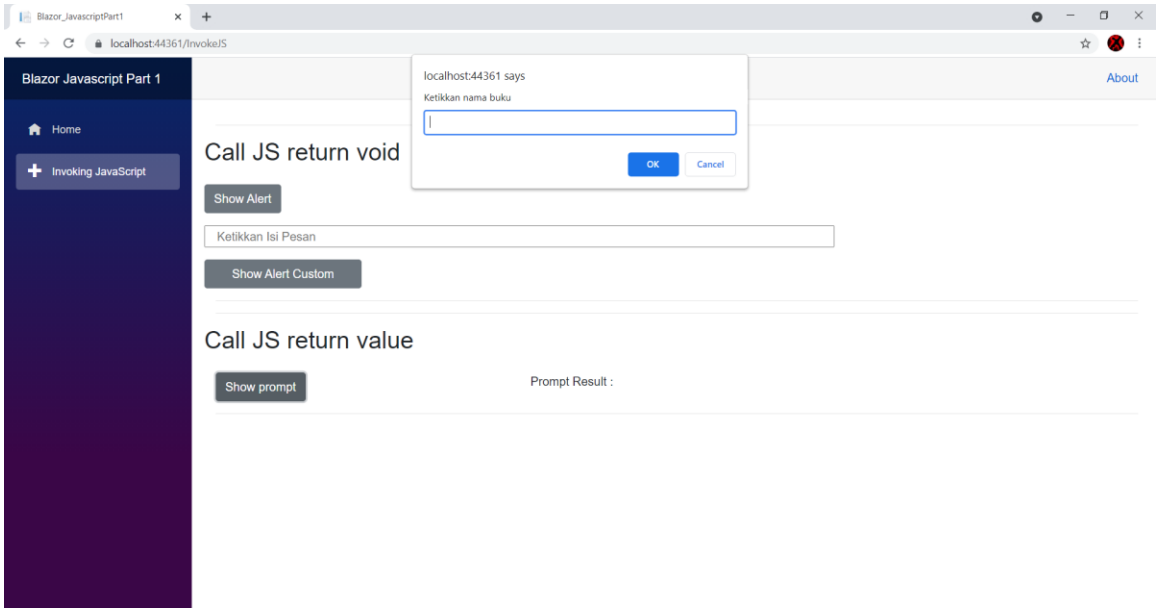
```
<div>
    <hr />
    <div class="form-group row">
        <h2>
            Call JS return value
        </h2>
    </div>

    <div class="form-group row">
        <div class="col-sm-4">
            <button class="btn btn-secondary" @onclick="ShowPrompt">Show prompt </button>
        </div>

        <div class="col-sm-4">Prompt Result :<b> @promptResult</b></div>
    </div>
    <hr />
</div>
```

Pada atribut onclick di button “Show prompt” menggunakan fungsi “ShowPromt” yang akan kita buat setelah ini. Sedangkan untuk menampilkan hasi dari fungsi pada javascript kita menggunakan variable “promptResult”.

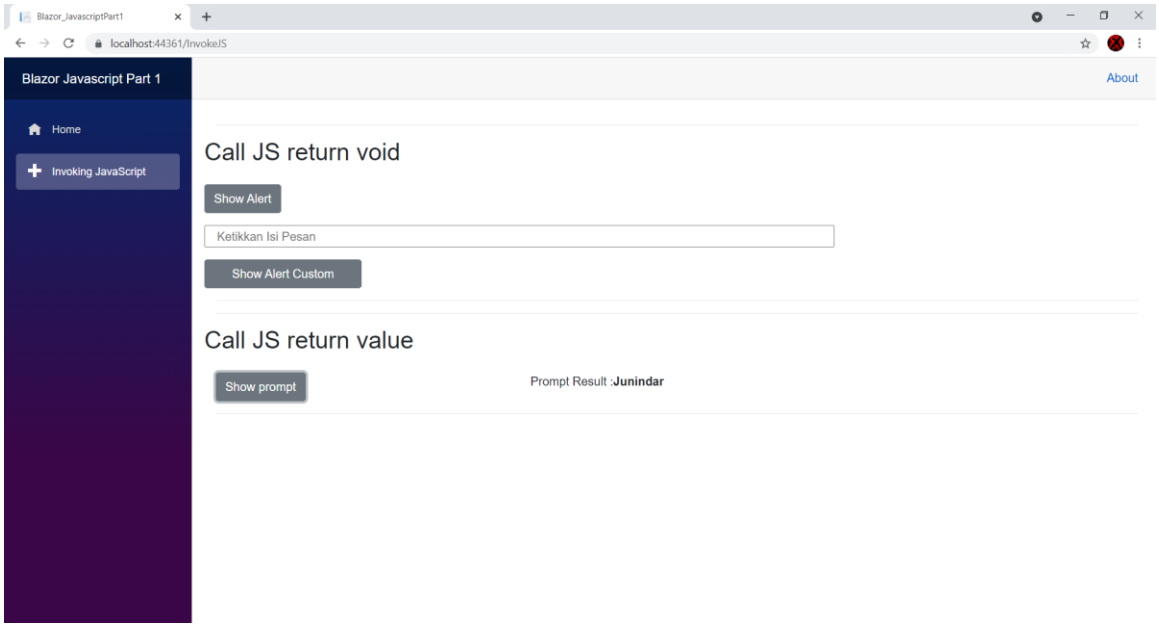
Untuk sintaks C# (.NET) ketikkan seperti dibawah ini.

```
private string promptResult;
private async Task ShowPrompt()
{
   var result = await JsRuntime.InvokeAsync<string>("blazorInterop.showPrompt",
   "Ketikkan nama buku", promptResult ?? "");
    if (result != null)
    {
      promptResult = result;
    }
}
```

"promptResult" digunakan untuk menampung nilai dari fungsi pada javascript yang dipanggil. Selanjutnya untuk fungsi "ShowPrompt", pada JsRuntime kita gunakan method "InvoveAsync<T>, dimana "T" nya adalah string. Pada saat memanggil method tersebut, kita hantarkan 3 buah paramater yaitu nama fungsi pada javascript (`blazorInterop.showPrompt`), Text title pada input box (`Ketikkan nama buku`) dan nilai untuk default value (`promptResult ?? ""`). Untuk default value memiliki kondisi jika nilai "promptResult" adalah kosong maka kita mengirimkan nilai "" (Empty Text). Jalankan program untuk melihat hasilnya. Klik button Show Prompt dan masukkan text pada input box.



Maka text yang kita ketikkan pada input box akan muncul pada halaman/page seperti dibawah ini.



Pada contoh diatas return value dari javascript adalah string, lalu bagaimana jika return valuenya adalah object, dan nilainya ditampung oleh class object pada .NET. Mari kita lanjutkan dengan latihan berikutnya. Tambahkan sebuah javascript function seperti dibawah ini.

```
blazorInterop.createBook = function (judul, penulis) {
    let book =
        {
            judul: judul,
            penulis: penulis,
            penerbit: "Cahaya Buku"

        };
    return book;
};
```

Pada sintaks diatas, terdapat dua buah parameter yang dikirimkan oleh Blazor App (.NET). Dan return value dari fungsi ini adalah sebuah object bertipe array (book). Lalu untuk menampung object tersebut, pada Blazor App kita tambahkan sebuah class dengan nama “Book” seperti dibawah ini.

```
public class Book
{
   public int BookID { get; set; }
   public string Judul { get; set; }
   public string Penulis { get; set; }
   public string Penerbit { get; set; }
   public string Deskripsi { get; set; }
   public bool Status { get; set; }
   public string Gambar { get; set; }

}
```

Lalu sintaks untuk memanggil fungsi javascript diatas adalah seperti dibawah ini.

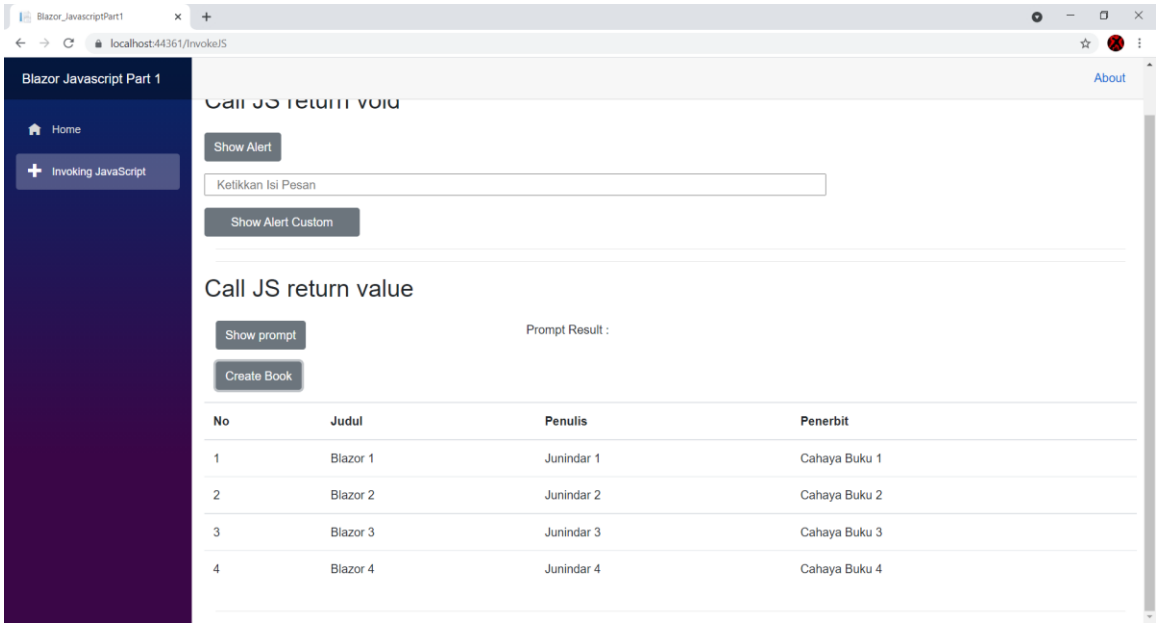
```
List<Book> Books= new List<Book>();
private async Task CreateBook()
{
    var book = await JsRuntime.InvokeAsync<Book>("blazorInterop.createBook",
    "Blazor", "Junindar");
    Books.Add(book);
}
```

Terdapat sebuah variable “Books”, digunakan untuk menampung object-object dari class Book yang didapat dari return value funsgi javascript. Sedangkan sintaks HTML untuk menampilkan data-data dapat dilihat dibawah ini.

Terdapat button untuk memanggil fungsi “CreateBook” yang telah kita buat diatas. Sedangkan untuk menampilkan data-data tersebut, kita gunakan HTML table dengan menggunakan variable “Books”.

```
<div class="form-group row">
        <div class="col-sm-4">
            <button class="btn btn-secondary" @onclick="CreateBook">Create Book </button>
        </div>
</div>
<div class="form-group row">
     <table class="table">
            <thead>
                <tr>

                    <th>No</th>
                    <th>Judul</th>
                    <th>Penulis</th>

                    <th>Penerbit</th>
                </tr>
            </thead>
            <tbody>
         @{
             int iRow = 1;
         }
            @foreach (var book in Books)
            {
                <tr>

                    <td>@iRow</td>
                    <td>@book.Judul @iRow</td>
                    <td>@book.Penulis  @iRow</td>
                    <td>@book.Penerbit  @iRow</td>

                </tr>
                iRow++;
            }

            </tbody>
        </table>
</div>
```

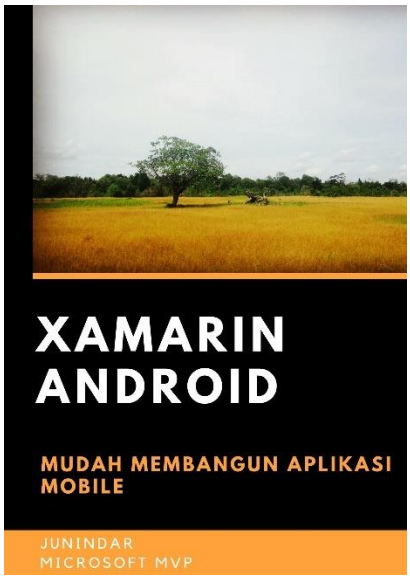
Jalankan program untuk melihat hasilnya. Pastikan pada saat kita klik button Create Book, baris pada table akan bertambah, seperti pada gambar dibawah ini.


Blazor_JavascriptPart1
localhost:44361/InvokeJS
Blazor Javascript Part 1
About
Home
Invoking JavaScript
Show Alert
Ketikkan Isi Pesan
Show Alert Custom
Call JS return value
Show prompt
Prompt Result :
Create Book
No
Judul
Penulis
Penerbit
1
Blazor 1
Junindar 1
Cahaya Buku 1
2
Blazor 2
Junindar 2
Cahaya Buku 2
3
Blazor 3
Junindar 3
Cahaya Buku 3
4
Blazor 4
Junindar 4
Cahaya Buku 4

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/2021/08/invoking-javascript-dari-net-pada.html

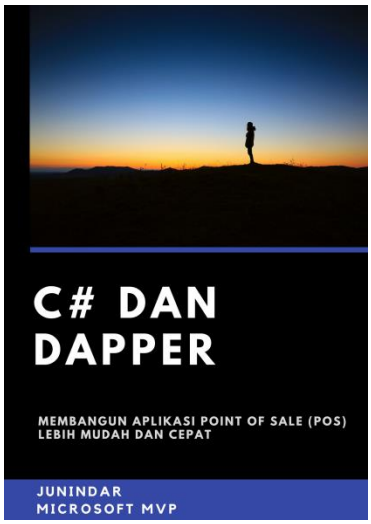
Referensi



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

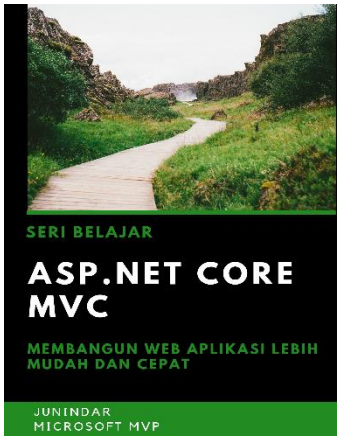
https://play.google.com/store/books/details?id=VSLiDQAAQBAJ

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

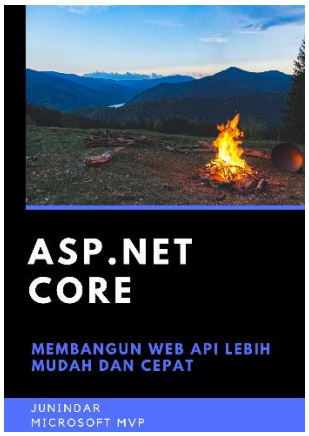


https://play.google.com/store/books/details/Junindar_C_dan_Dapper_Membangun_Aplikasi_POS_Point?id=6TErDwAAQBAJ

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".**