IlmuKomputer.Com
Ikatlah ilmu dengan menuliskannya

Authentication dan Authorization pada Blazor

Junindar, ST, MCPD, MOS, MCT, MVP

junindar@gmail.com
http://junindar.blogspot.com

Abstrak

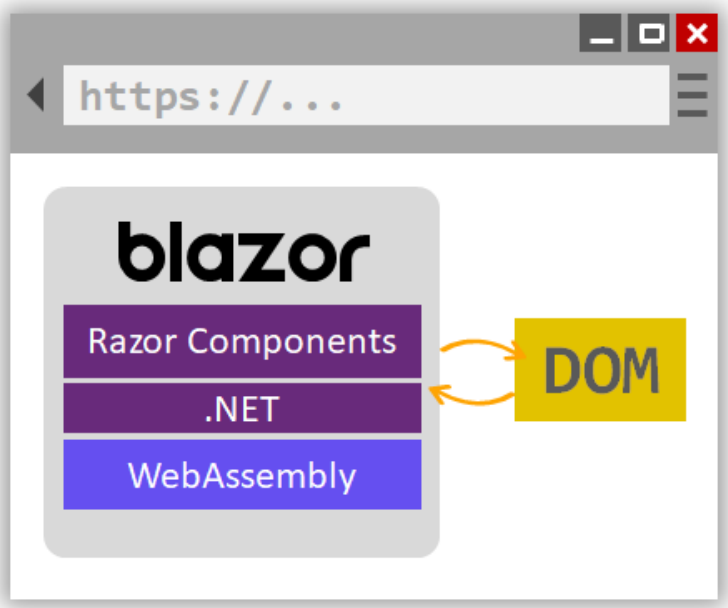
Blazor adalah Web Framework yang bersifat Open Source dimana aplikasi Web yang bersifat client-side interactive dapat dikembangkan dengan menggunakan .Net (C#) dan HTML. Pada saat ini C# biasa digunakan untuk melakukan proses back-end dari aplikasi web. Dengan menggunakan fitur baru dari ASP.NET Core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

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 .

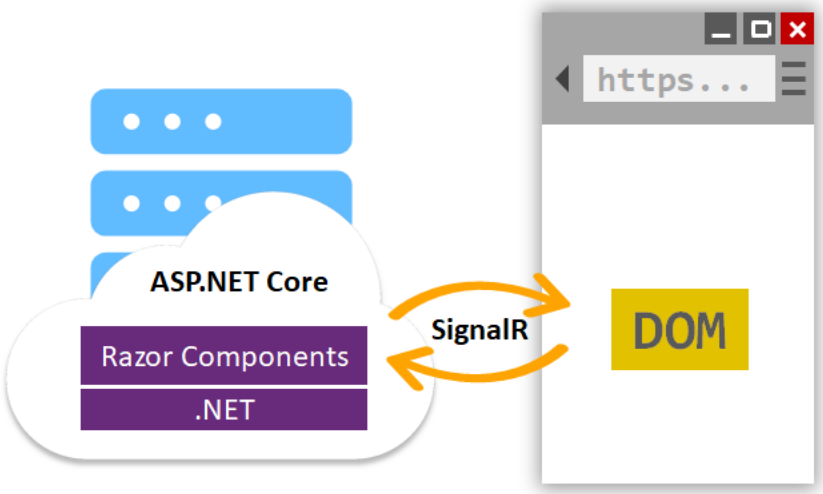
Code .Net berjalan pada Web Assembly, yang artinya kita dapat menjalankan "NET" didalam browser (Client) tanpa harus menginstall plugin seperti Silverlight, Java mapun Flash.

Note : Link untuk Project ini ada pada Bab Penutup



Dengan menggunakan Blazor kita dapat memilih apakah menggunakan WebAssemby (Client Side), seperti yang telah dijelaskan di atas atau Blazor dijalankan diatas server, Pada latihan ini kita akan menggunakan cara kedua yaitu Blazor dijalankan diatas server. Dengan menggunakan cara ini kita memerlukan SignalR untuk menghubungkan antara client (browser) dan server app. Sebagai contoh jika user melakukan proses klik button

pada browser, maka data akan dikirmkan ke Server menggunakan SignalR dan hasilnya akan dikembalikan ke client dengan mengupdate DOM pada client.



Disarankan untuk membaca dan menyeselesaikan latihan pada beberapa artikel sebelumnya http://junindar.blogspot.com/2020/02/pengenalan-blazor.html http://junindar.blogspot.com/2020/02/create-read-update-dan-delete-crud-pada.html dan http://junindar.blogspot.com/2020/07/membuat-dialog-component-pada-blazor.html.

Authentication dan *Authorization* pada sebuah aplikasi adalah termasuk dalam bagian yang sangat penting yang harus diperhatikan oleh para developer.

Pada latihan ini kita akan belajar menggunakan authentication dan authorization pada blazor. Dimana untuk latihan awal kita akan membuat halaman login dan selanjutnya dengan membuat membuat *Authentication* dan *Authorization* pada setiap halaman.

Pada artikel ini tidak menjelaskan apa itu blazor, bagaimana bekerja dengan EF Core membuat fungsi CRUD karena semuanya telah dijelaskan pada artikel sebelumnya. Pastikan anda telah menyelesaikan latihan-latihan pada artikel sebelumnya. Artikel ini akan focus pada authentication dan authorization pada blazor

Untuk memulai latihan pada artikel ini buat terlebih dahulu project Blazor Server side pada Visual Studio 2019.

Note : Link untuk Project Lampiran ada pada Penutup.

Seperti biasa artikel ini akan mengajak pembaca untuk langsung mempratekkan isi artikel dengan latihan-latihan yang telah dibuat, agar dapat memahami isi artikel lebih baik dan benar. Ikuti langkah-langkah dibawah ini dengan baik dan benar.

Sebelumnya kita akan membuat table user terlebih dahulu pada database. Dengan menggunakan EF Core. Pastikan sebelumnya buat class User seperti dibawah terlebih dahulu.

```
public class User
{
        [Key]
        [Required]
        [Display(Name = "User name")]
        public string Username { get; set; }

        [Required]
        [Display(Name = "Nama Pengguna")]
        public string Nama { get; set; }

        [Required]
        [DataType(DataType.Password)]
        [Display(Name = "Password")]
        public string Password { get; set; }

        [Required]
        [Display(Name = "User Role")]
        public string Role { get; set; }

        public bool Status { get; set; }
}
```

Seperti pada latihan sebelumnya, kita akan membuat service untuk Class User diatas yang akan kita gunakan nanti pada saat proses login.

```
public interface IUserRepository
{
         Task<User> CheckLoginAsync(User obj);
}
```

Function ini digunakan untuk mencari data pengguna pada table Users berdasarkan Username dan Password. Sehingga pada saat pengguna menginputkan data pada halama login (Username dan Password) dan setelah button login diklik maka function CheckLoginAsync ini yang akan dipanggil. Sedangkan untuk implementasi dari function tersebut sintaksnya terdapat dibawah ini.

```
public class UserRepository : IUserRepository
{
        private readonly PustakaDbContext _dbContext;

        public UserRepository(PustakaDbContext dbContext)
        {
            _dbContext = dbContext;
        }
        public async Task<User> CheckLoginAsync(User obj)
        {
            var user = await _dbContext.Users.Where(c => c.Username.ToLower()
             == obj.Username.ToLower() &&
            c.Password==obj.Password).FirstOrDefaultAsync();

            return user;
        }
}
```

Lalu tambah sebuah class pada folder Data dengan nama “CustomAuthStateProvider“ dan tambahkan sintaks-sintaks dibawah pada class yang telah kita buat.

```
public class CustomAuthStateProvider : AuthenticationStateProvider
{


}
```

Pada “CustomAuthStateProvider“ menggunakan service “AuthenticationStateProvider“ class yang merupakan bagian dari “Microsoft.AspNetCore.Components.Authorization“ yang digunakan untuk memberikan informasi tentang status authentication pengguna. AuthenticationStateProvider adalah service yang digunakan oleh *AuthorizeView* dan *CascadingAuthenticationState* untuk mendapatkan informasi dari status authentication pengguna.

Berikut method-method yang terdapat pada class “CustomAuthStateProvider“.

```
private ClaimsIdentity UserClaimsIdentity(User user)
{
            var claimsIdentity = new ClaimsIdentity();
            if (user != null)
            {
                var claims = new List<Claim>
                        {
                            new Claim(ClaimTypes.Name, user.Nama),
                            new Claim(ClaimTypes.Email, user.Username),
                            new Claim(ClaimTypes.Role, user.Role),
                        };

                claimsIdentity = new ClaimsIdentity(
                     claims,
                     CookieAuthenticationDefaults.AuthenticationScheme);

            }
            return claimsIdentity;
}
public override Task<AuthenticationState> GetAuthenticationStateAsync()
{
     var identity = new ClaimsIdentity();
     var claimsPrincipal = new ClaimsPrincipal(identity);
     return Task.FromResult(new AuthenticationState(claimsPrincipal));
}
```

Disini kita perlu mengganti sintaks pada method "GetAuthenticationStateAsync", pada method ini akan ditentukan jika user telah diautentikasi atau tidak.

```
public void UserAuthenticated(User user)
{
     var identity = UserClaimsIdentity(user);
     var claimsPrincipal = new ClaimsPrincipal(identity);
     NotifyAuthenticationStateChanged(Task.FromResult(new
                                  AuthenticationState(claimsPrincipal)));
}
public void UserIsLoggedOut()
{
       var identity = new ClaimsIdentity();
       var user = new ClaimsPrincipal(identity);
       NotifyAuthenticationStateChanged(Task.FromResult(new
       AuthenticationState(user)));
}
```

Sedangkan dua method diatas (UserAuthenticated dan UserIsLoggedOut) digunakan untuk mengupdate status otentikasi pada saat user login dan logout. Pada method UserAuthenticated akan memanggil method UserClaimsIdentity untuk mengupdate ClaimsIdentity dengan data user yang telah berhasil login.

Setelah menyelesaikan sintaks-sintaks diatas, pastikan kita register service ini pada ConfigureService yang terdapat didalam Startup class, seperti dibawah.

```
services.AddAuthentication(CookieAuthenticationDefaults.AuthenticationScheme)
                .AddCookie();
services.AddScoped<AuthenticationStateProvider, CustomAuthStateProvider>();
```

Lalu buka file App.Razor dan ganti sintaks pada file tersebut menjadi seperti sintaks dibawah ini.

```
<CascadingAuthenticationState>
    <Router AppAssembly="@typeof(Program).Assembly">
        <Found Context="routeData">
            <AuthorizeRouteView RouteData="@routeData"
              DefaultLayout="@typeof(MainLayout)">
                <NotAuthorized>
                    <h1>Sorry, you're not authorized to view this page.</h1>
                    <p>You may want to try logging in (as someone with the
                     necessary authorization).</p>
                </NotAuthorized>
            </AuthorizeRouteView>
        </Found>
        <NotFound>
            <LayoutView Layout="@typeof(MainLayout)">
                <p>Sorry, there's nothing at this address.</p>
            </LayoutView>
        </NotFound>
    </Router>
</CascadingAuthenticationState>
```

Setelah selesai dengan langkah-langkah diatas kita akan lanjutkan dengan membuat halaman login terlebih dahulu. Tambahkan sebuah razor component dengan nama Login.Razor dilanjutkan dengan menambahkan sebuah class dengan nama LoginBase yang akan menjadi backend dari razor component Login. Class ini akan mengimplementasikan dari IComponent yang merupakan bagian dari "Microsoft.AspNetCore.Components".

```
public class LoginBase: ComponentBase
{
        [Inject]
        public IUserRepository UserRepository { get; set; }

        [Inject]
        private NavigationManager NavigationManager { get; set; }
        [Inject]
        private AuthenticationStateProvider AuthenticationStateProvider { get; set; }

        public User user { get; set; } = new User();

        public string LoginMesssage { get; set; }

        ClaimsPrincipal claimsPrincipal;

        [CascadingParameter]
        private Task<AuthenticationState> authenticationStateTask { get; set; }
}
```

```
protected override async Task OnInitializedAsync()
{
            user = new User();

            claimsPrincipal = (await authenticationStateTask).User;

            if (claimsPrincipal.Identity.IsAuthenticated)
            {
                 ((CustomAuthStateProvider)AuthenticationStateProvider)
                     .UserIsLoggedOut();
            }

}

protected async Task<bool> ValidateUser()
{
             var loginResult = await UserRepository.CheckLoginAsync(user);
             if (loginResult != null)
             {
                 if (loginResult.Status)
                 {
                      ((CustomAuthStateProvider)AuthenticationStateProvider).
                          UserAuthenticated(loginResult);
                      NavigationManager.NavigateTo("/");
                 }
                 else
                 {
                     LoginMesssage = "User Inactivated";
                 }
             }
             else
             {
                 LoginMesssage = "Invalid username or password";
             }
             return await Task.FromResult(true);

}
```

Untuk method OnInitializedAsync digunakan untuk mengecek, apakah user telah login atau tidak. Jika sudah maka status otentikasi akan diganti menjadi logout dengan memanggil method UserIsLoggedOut pada class CustomAuthStateProvider yang telah kita buat sebelumnya. Sedangkan untuk method ValidateUser akan dieksekusi pada saat button login diklik. Jika data user (Username dan Password) ada pada table, maka status otorisasi akan diperbaharui dengan memanggil method UserAuthenticated.

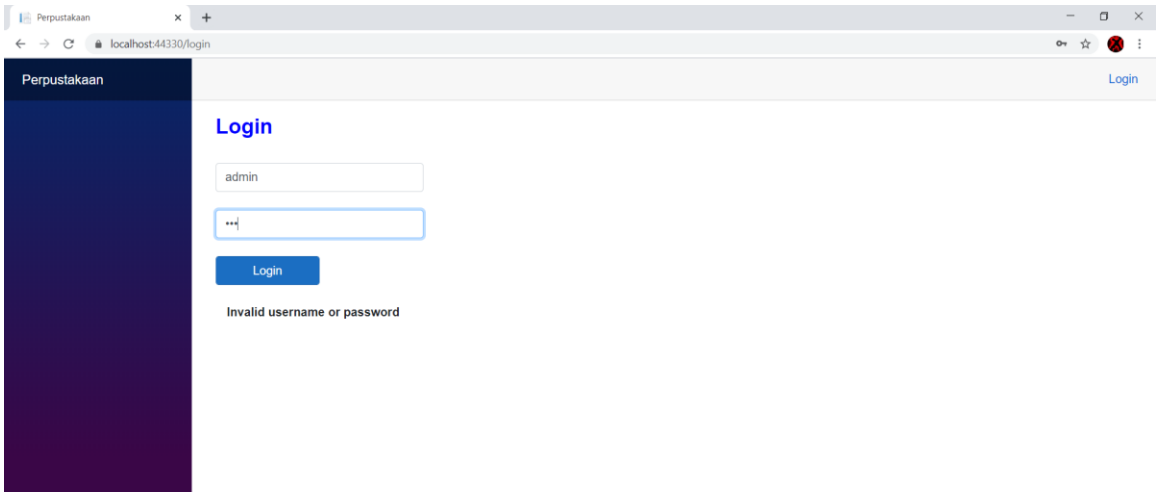
Lalu kita tambahkan sintaks pada Login.Razor, seperti dibawah.

```
@page "/Login"
@inherits BlazorAPP_Login.Pages.LoginBase

<EditForm Model="@user" OnValidSubmit="@ValidateUser">

    @*Detail sintaks pada project lampiran*@

</EditForm>
```

Untuk mencoba halaman login ini, kita perlu menambahkan link pada layout. Bukan file MainLayout.razor dan tambahkan sintak dibawah ini.

```
<AuthorizeView>
            <Authorized>
                <a href="login">Logout</a>
            </Authorized>
            <NotAuthorized>
                <a href="login" >Login</a>
            </NotAuthorized>
</AuthorizeView>
```

Disini kita menggunakan AuthorizedView, dimana jika user belum melakukan login (NotAuthorized) maka link Login yang akan muncul, jika sudah (Authorized) maka link Logout. Lalu jalankan aplikasi dan lakukan proses login untuk melihat hasilnya.

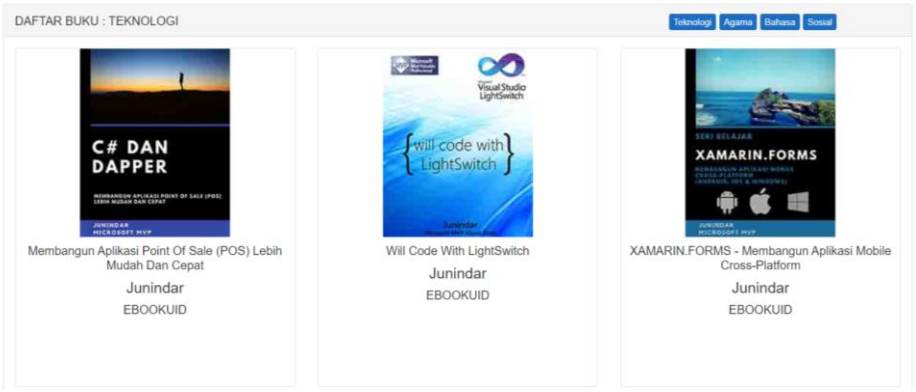


Pastikan proses Login dan Logout berjalan dengan baik. Setelah proses *Authentication* selesai maka akan kita lanjutkan dengan *Authorization*. Pada Latihan ini kita akan menggunakan 3 role yaitu Admin, Staff dan General. Untuk Admin akan memiliki hak akses yang paling tinggi (dapat melakukan semua proses), Staff tidak bisa melakukan proses delete pada data buku. Dan yang terakhir adalah General dimana hanya dapat

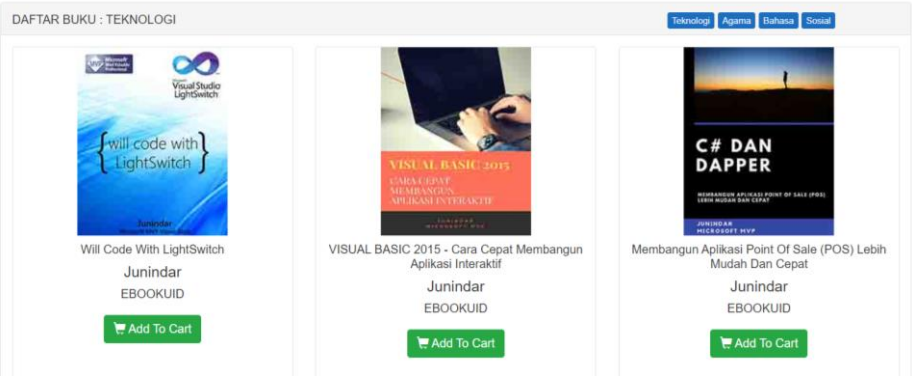
melakukan proses Add Chart. Jadi pada saat user belum melakukan login maka button Add Chart akan hilang, dan hanya akan tampil jika user telah melakukan proses login. Untuk lebih faham mengenai *Authorization* ikuti langkah-langkah dibawah ini.

Buka file CardBook.razor pada folder Components, lalu pada bagian button Add to Chart ganti sintaksnya seperti dibawah ini.

```
<AuthorizeView>
      <Authorized>
                  @{
                    if (book.Status)
                    {
                       <p style="text-align: center">
                             <a class="btn btn-success" role="button"
                             data-bukuId="@book.BookID"
                             data-bukuImage="@book.Gambar"
                             data-bukuJudul="@book.Judul"
                             data-bukuPenulis="@book.Penulis"
                             style="margin-top: 10px; color: white !important">
                             <span class="glyphicon glyphicon-shopping-cart">
                             </span> Add To Cart
                             </a>

                       </p>

                       }
                }
      </Authorized>
</AuthorizeView>
```



Sebelum Login

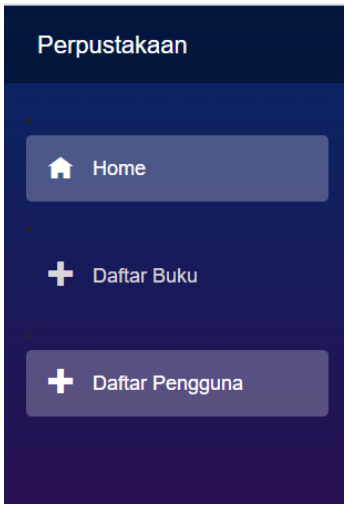


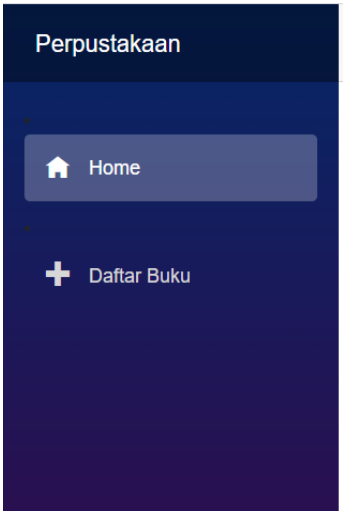
Sesudah Login

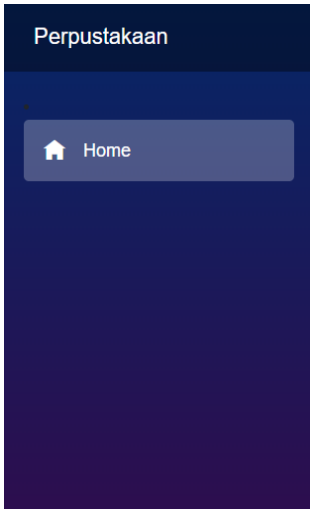
Selanjutnya buka NavMenu.razor pada folder Shared. Pada latihan ini kita akan membuat 3 buah link yaitu Home, Daftar Buku dan Daftar Pengguna. Daftar Pengguna untuk user dengan role Admin, Daftar Buku role Admin dan Staff, sedangkan Home untuk semua role.

```
<AuthorizeView >
    <Authorized>
        <li class="nav-item px-3">
            <NavLink class="nav-link" href="" Match="NavLinkMatch.All">
                <span class="oi oi-home" aria-hidden="true"></span> Home
            </NavLink>
        </li>
    </Authorized>
</AuthorizeView>

<AuthorizeView Roles="Admin, Staff">
    <li class="nav-item px-3">
            <NavLink class="nav-link" href="BookList">
                <span class="oi oi-plus" aria-hidden="true"></span>
                 Daftar Buku
            </NavLink>
        </li>
</AuthorizeView>

<AuthorizeView Roles="Admin">
    <li class="nav-item px-3">
        <NavLink class="nav-link" href="/">
            <span class="oi oi-plus" aria-hidden="true"></span>
                 Daftar Pengguna
        </NavLink>
    </li>
</AuthorizeView>
```

Dapat kita lihat perbedaan sintaks untuk ketiga link diatas, untuk penggunaan spesifik role kita gunakan “AuthorizeView Roles“ dilanjutkan dengan nama role-nya. Jalankan program dan pastikan mendapatkan hasil seperti dibawah ini.



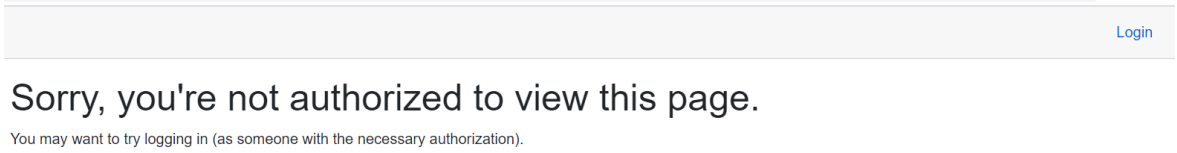




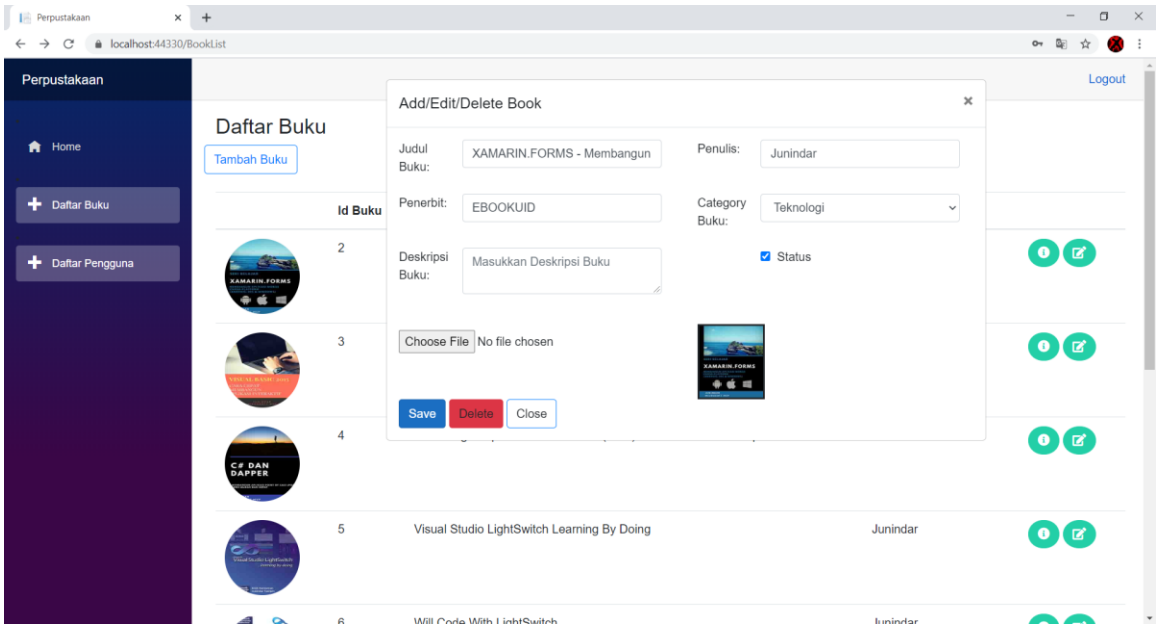
Pada dasarnya proses yang kita lakukan diatas masih dalam level UI. Sehingga jika user langsung mengetikkan alamat pada browser user yang tidak memiliki authorize masih dapat membukanya. Coba login dengan user yang memiliki role yang paling rendah (General), dan ketikkan url berikut “https://localhost:44330/Booklist” pada browser. Sesuai dengan konsep yang telah dibahas diatas seharunya hanya admin dan staff yang bisa membuka halaman ini. Untuk memberikan authorization pada page ketikkan sintaks berikut dibaris ketiga pada BookList.razor.

```
@attribute [Authorize(Roles = "Admin,Staff")]
```

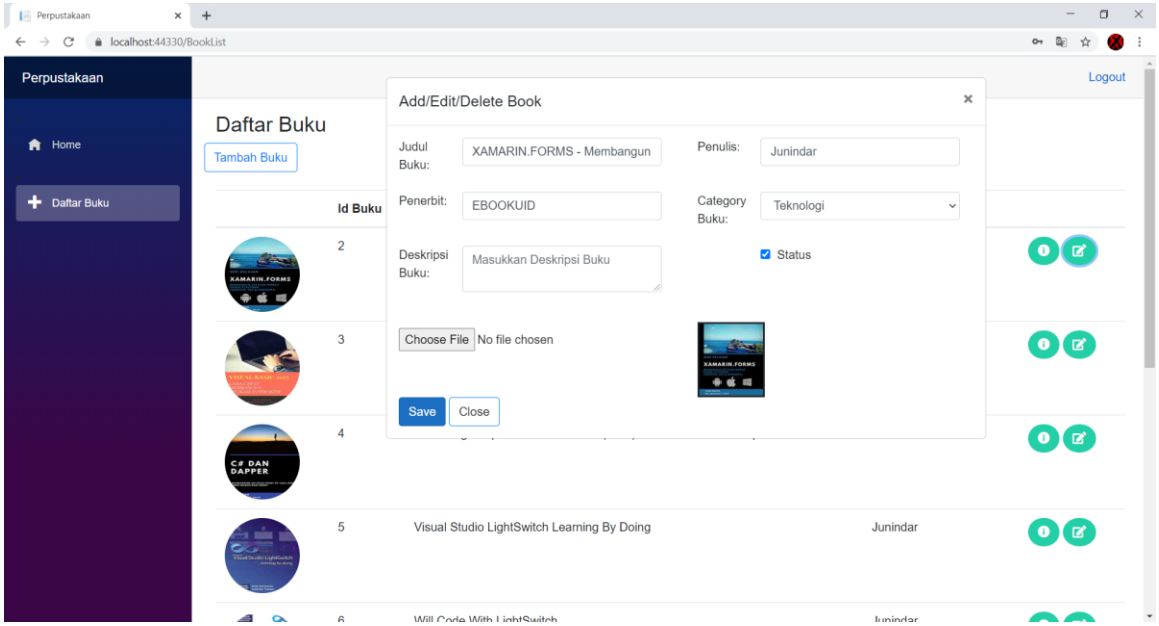
Lalu login kembali dengan menggunakan user dengan role General dan ketikkan url berikut “https://localhost:44330/Booklist” pada browser. Pastikan mendapatkan hasil seperti dibawah.



Sebagai latihan aplikasikan proses otorisasi untuk halaman-halaman lainnya. Seperti BookDetail dan AddEditBook. Untuk halaman AddEditBook pastikan hanya admin yang dapat melakukan proses delete data.



Admin View



Staff View

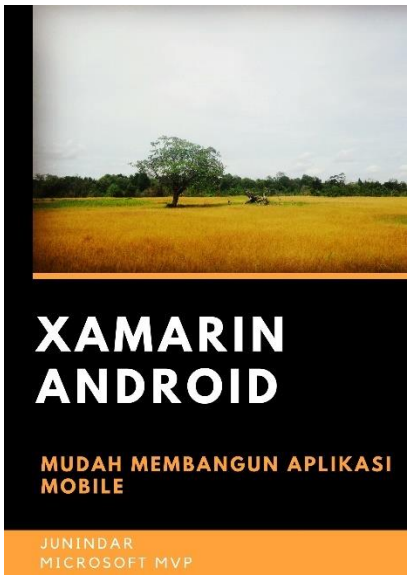
Penutup

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/2020/08/authentication-dan-authorization-pada.html

Semoga bermanfaat.

Wassalam.

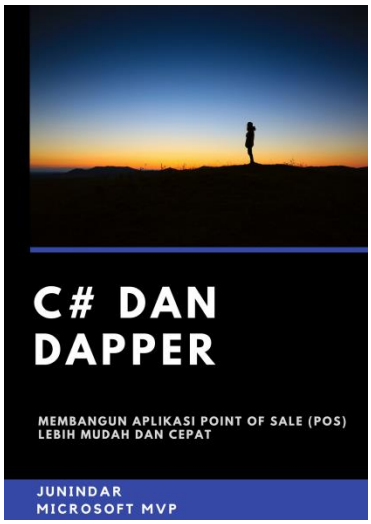
Referensi



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

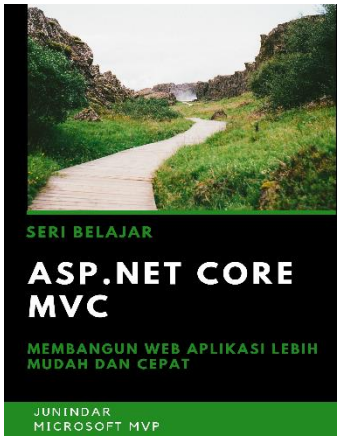
https://play.google.com/store/books/details?id=VSLiDQAAQBAJ

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

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".**