IlmuKomputer.Com
Ikatlah ilmu dengan menuliskannya

Mengakses Web Api Dari 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 .

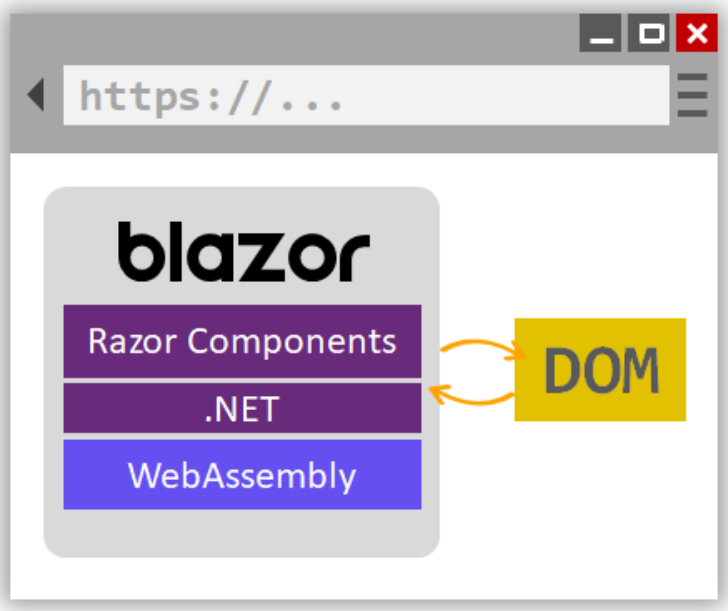
Code .Net berjalan pada WebAssembly, yang artinya kita dapat menjalankan “NET“ didalam browser (Client) tanpa harus menginstall plugin seperti Silverlight, Java mapun Flash.

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 .

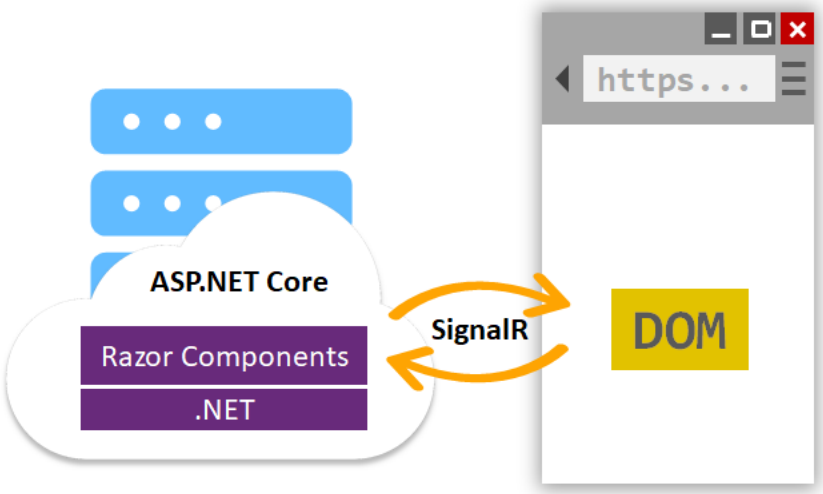
Code .Net berjalan pada Web Assembly, yang artinya kita dapat menjalankan "NET" didalam browser (Client) tanpa harus menginstall plugin seperti Silverlight, Java mapun Flash.

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.



Artikel ini tidak menjelaskan secara detail apa itu Blazor dan bagaimana cara membuat project Blazor dengan Visual Studio 2019, oleh karena itu sangata 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 , http://junindar.blogspot.com/2020/07/membuat-dialog-component-pada-blazor.html dan http://junindar.blogspot.com/2020/08/authentication-dan-authorization-pada.html .

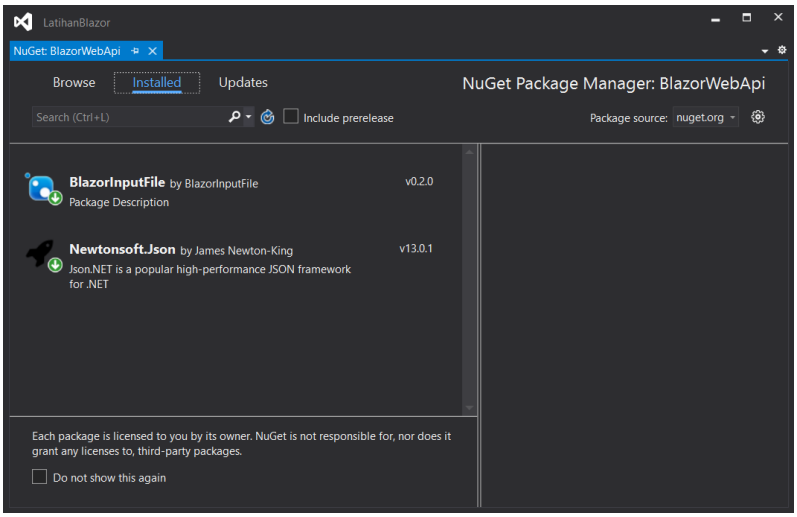
Pada artikel sebelumnya contoh aplikasi yang kita buat berkoneksi langsung dengan database dengan menggunakan Entity Framework. Sedangkan pada artikel ini kita akan membuat bagaimana aplikasi blazor dalam mengakes Web Api. Untuk yang belum memahami dalam membuat Web Api dapat membaca buku "**ASP.NET CORE: Membangun Web API Lebih Mudah dan Cepat**" disini https://play.google.com/store/books/details/Junindar_ASP_NET_CORE?id=COUWEAAAQBAJ .

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.

Setelah berhasil membuat Project Blazor Server App, tambahkan Nuget Package seperti pada gambar dibawah ini.



Selanjutnya buka file "_Imports.razor" dan tambahkan sintaks berikut `@using BlazorInputFile` pada file tersebut.

Sebelum memulai dengan membuat UI terlebih dahulu kita buat Service-service untuk mengakses Web Api. Tambahkan sebuah folder "Models" pada project dan dilanjutkan dengan menambahkan dua buah file class (Category dan Book) seperti dibawah.

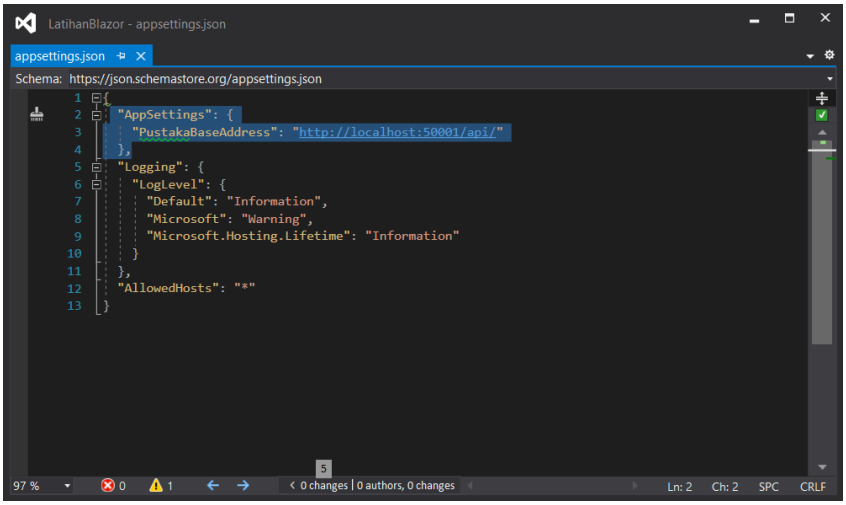
```
public class Category
{
    public int ID { get; set; }
    public string Nama { get; set; }
    public List<Book> Books { get; set; }
}

public class Book
{
   public int ID { 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; }
   public int CategoryID { get; set; }
   public Category Category { get; set; }
}
```

Pastikan nama-nama dari property pada class sama dengan hasil yang terdapat pada web api.

Lalu tambahkan sebuah class dengan nama "AppSettings" seperti dibawah ini.

```
public class AppSettings
{
      public string PustakaBaseAddress { get; set; }
}
```

Class ini digunakan untuk mengakses alamat dari web api (URL). Sedangkan pada latihan ini kita akan meletakkan Url ini didalam file "appsettings.json". Tambahkan informasi dibawah (AppSettings) pada file "appsettings.json".



Dan pada method ConfigureServices (Startup.cs), tambahkan sintaks dibawah ini.

```
var appSettingSection = Configuration.GetSection("AppSettings");
services.Configure<AppSettings>(appSettingSection);
```

Setelah selesai dengan langkah-langkah diatas, kita lanjutkan dengan menambahkan folder "Data" pada project, folder ini merupakan tempat dari file-file service yang ada pada project. Dilanjukan dengan menambahkan dua buah file Interface seperti dibawah.

```
public interface ICategoryRepository
{
        Task<List<Category>> GetAll();
        Task<Category> GetById(int categoryId);
}

public interface IBookRepository
{
        Task Insert(Book book);
        Task Update(Book book);
        Task Delete(int bookId);
        Task<IEnumerable<Book>> GetAllBooks();
        Task<Book> GetBookById(int bookId);
}
```

Seperti kita ketahui interface tidak memiliki implementasi dari sebuah method, oleh karena itu kita akan menambahkan dua buah class lagi untuk mengimplementasikan

method-method pada interface-interface diatas. Untuk mengakses web api kita akan menggunakan HttpClient class.

```
public class CategoryRepository: ICategoryRepository
{
   public HttpClient _httpClient { get; }
   public AppSettings _appSettings { get; }
   public CategoryRepository(HttpClient httpClient, IOptions<AppSettings> appSettings)
   {
            _appSettings = appSettings.Value;
            httpClient.BaseAddress = new Uri(_appSettings.PustakaBaseAddress);
            httpClient.DefaultRequestHeaders.Accept.Clear();
            httpClient.DefaultRequestHeaders.Accept.Add(
                new MediaTypeWithQualityHeaderValue("application/json"));


            _httpClient = httpClient;
    }


}
```

Pada class ini terdapat dua buah property get, yang pertama adalah _httpClient dan yang kedua adalah _appsettings. _httpClient digunakan untuk berkomunikasi dengan web api sedangkan untuk mengakses url dari web api kita gunakan _appsettings yang nilai nya digunakan oleh BaseAddress dalam mengakses web api.

Sintaks dibawah merupakan implementasi dari method GetAll(). Disini dapat kita lihat, method ini melakukan request “GET“ pada end point “categories“. Lalu hasinya akan dikonversi menjadi object List<Class> dengan menggunakan “Newtonsoft.JSON“ library yang telah kita tambahkan sebelumnya.

```
public async Task<List<Category>> GetAll()
{
     var requestMessage = new HttpRequestMessage(HttpMethod.Get, "categories");
     var response = await _httpClient.SendAsync(requestMessage);

     var responseStatusCode = response.StatusCode;

     if (responseStatusCode.ToString() == "OK")
     {
         var responseBody = await response.Content.ReadAsStringAsync();
         return await
         Task.FromResult(JsonConvert.DeserializeObject<List<Category>>(responseBody));
     }

     return null;

}
```

Untuk seluruh sintaks dari service-service dapat dilihat pada project lampiran. Setelah selesai dengan membuat service untuk Category dan Book, tambahkan sintaks dibawah pada method ConfigureServices didalam class Startup.cs.

```
services.AddHttpClient<ICategoryRepository, CategoryRepository>();
services.AddHttpClient<IBookRepository, BookRepository>();
services.AddSingleton<HttpClient>();
```

Langkah selanjutnya adalah membuat UI untuk aplikasi pada project. Tambahkan sebuah class dengan nama BookListBase.cs. Class ini merupakan code behind dari Razor component yang digunakan untuk menampilkan daftar buku hasil dari request pada web api.

```
public class BookListBase : ComponentBase
{
    [Inject]
    public IBookRepository BookRepository { get; set; }

    public IEnumerable<Book> Books { get; set; }

    protected override async Task OnInitializedAsync()
    {
        Books = (await BookRepository.GetAllBooks()).ToList();
    }
}
```

Diatas merupakan sintaks pada class BookListBase yang telah kita buat sebelumnya. Terdapat dua buah properties BookRepository dan Books. Pada BookRepository kita menggunakan atribut Inject.

Disini kita melakukan overide pada method OnInitializeAsync, dengan memanggil method GetAllBooks pada BookRepository. Properti public IEnumerable<Book> Books { get; set; }, digunakan untuk menampung data-data Book dari hasil method GetAllBooks dan nantinya digunakan pada fie BookList.Razor.

Tambahkan sebuah RazorComponent pada folder Pages dengan dengan nama "BookList.razor". dan pada awal baris code, ketikkan sintaks berikut.

```
@page "/BookList"
@inherits BookListBase
```

Sedangkan untuk menampilkan data-data buku pada halaman, ketikkan sintaks seperti dibawah ini.

```
@if (Books == null)
{
    <p><em>Data tidak tersedia</em></p>
}
else
{
 //Detail Sintaks pada lampiran
}
```

Terdapat dua buah kondisi disini, jika Books sama dengan Null maka page akan terdapat sebuah text "Data tidak tersedia", dan jika ada datanya maka data-data tersebut akan ditampilkan pada HTML table.

```
<div class="row">
   <a href="/bookedit" class="btn btn-outline-primary">
      Tambah Buku
   </a>
</div>
    <br />
<table class="table">
        <thead>
        <tr>
            <th></th>
            <th>Id Buku</th>
            <th>Judul</th>
            <th>Penulis</th>
            <th>Penerbit</th>
            <th></th>
        </tr>
        </thead>
        <tbody>
        @foreach (var book in Books)
        {
          //Sintaks details lihat pada project lampiran
        }
        </tbody>
</table>
```

Selain menampilkan data buku, kita juga perlu membuat navigasi untuk menampilkan Detail Buku baik untuk menampilkan data secara read only maupun untuk proses edit dan delete. Dimana ID buku yang digunakan sebagai parameternya. Sebelum kita membuat halaman tersebut pastikan halaman ini dapat berjalan dengan baik terlebih dahulu.

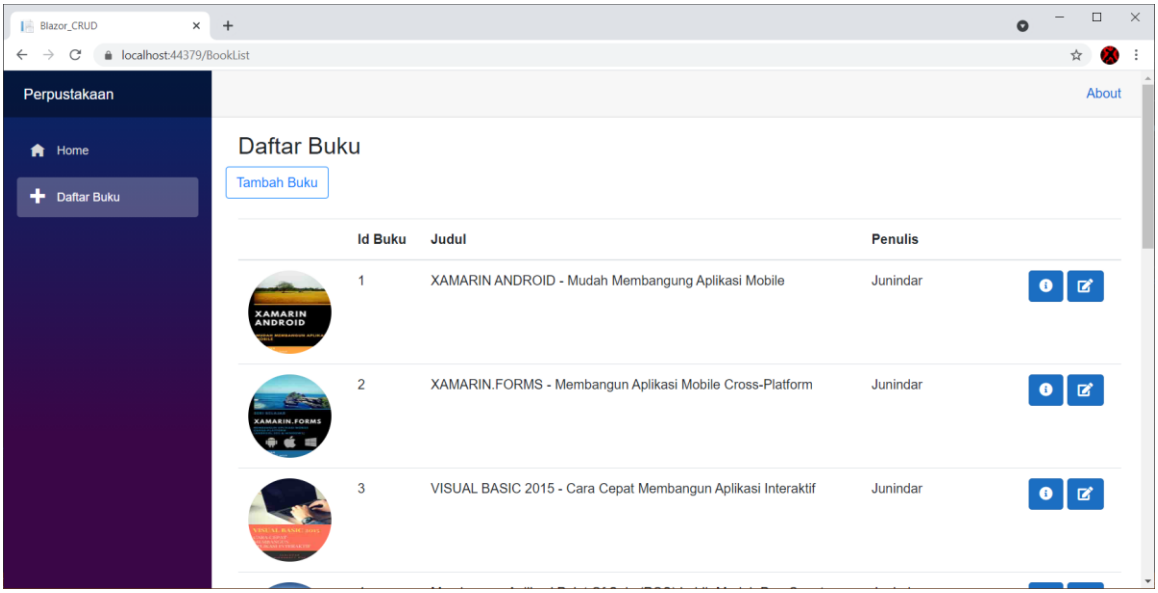
```
<a href="@($"bookdetail/{book.ID}")"
 class="btn btn-primary table-btn">
       <i class="fas fa-info-circle"></i>
</a>

<a href="@($"bookedit/{book.ID}")"
class="btn btn-primary table-btn">
        <i class="fas fa-edit"></i>
</a>
```

Selanjutnya pada folder wwwroot, tambahkan sebuah folder images dan copy file gambar dari project lampiran.

Lalu tambahkan link menu pada file NavMenu.razor seperti pada sintaks dibawah.

```
<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>
```

Dan jalankan project ini pastikan mendapatkan hasil seperti gambar dibawah ini.



Setelah selesai dengan halaman diatas, selanjutnya kita akan membuat untuk halaman BookDetail yang digunakan untuk menampilkan data buku yang dipilih secara detail. Buat sebuah class dengan nama BookDetailBase pada folder Pages.

```
public class BookDetailBase : ComponentBase
{
    [Parameter]
    public string BookId { get; set; }
    public Book Book { get; set; } = new Book();
    [Inject]
    public IBookRepository BookRepository { get; set; }
    [Inject]
    public ICategoryRepository CategoryRepository { get; set; }
    protected string NamaCategory = string.Empty;
    protected override async Task OnInitializedAsync()
    {
       Book = await
       BookRepository.GetBookById(int.Parse(BookId));
       var cat = await
       CategoryRepository.GetById(Book.CategoryID);
       NamaCategory = cat.Nama;
    }
}
```

Pada class ini terdapat sebuah property string “BookId“ dengan menggunakan Attribute Parameter, dimana property ini akan menampung data yang dikirimkan melalui razor komponen. Sedangkan property Book digunakan untuk menampung hasil pencarian berdasarkan property “BookId“ diatas.

Tambahkan sebuah Razor komponen dengan nama BookDetail.

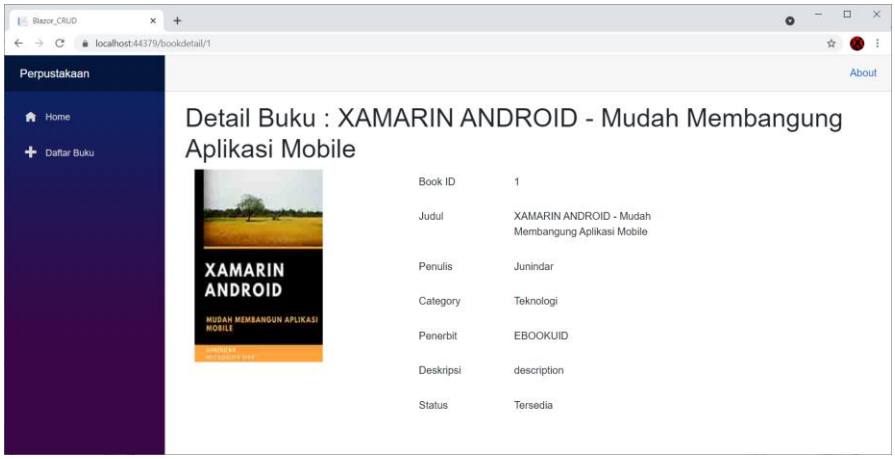
```
@page "/bookdetail/{BookId}"
@inherits BookDetailBase
```

Pada baris pertama dapat kita lihat merupakan navigasi ke bookdetail dengan mengirimkan parameter BookId. Untuk menampilkan hasilnya pada layar dapat dilihat pada sintaks dibawah.

```
<h1 class="page-title">Detail Buku : @Book.Judul</h1>

<div class="col-12 row">
    <div class="col-4">
        <img src="@($"images/{Book.Gambar}")"   width="200"
height="300" />
    </div>
    <div class="col-8 row">
        <div class="col-xs-12 col-sm-8">
            <div class="form-group row">
                <label class="col-sm-4
                col-form-label">Book ID</label>
                <div class="col-sm-8">
                    <label type="text" class="form-control-
                        plaintext">@Book.ID</label>
                </div>
            </div>
              @*Detail sintaks ada lampiran project*@
        </div>
    </div>
</div>
```

Pada halaman ini akan dibagi dua yang pertama untuk gambar buku (`<div class="col-4">`), sedangkan yang kedua (`<div class="col-8 row">`) untuk menampilkan data-data buku.

Sekali lagi jalankan program pada halaman daftar buku, klik button detail. Dan pastikan mendapatkan hasil seperti dibawah ini.



Setelah berhasil menampilkan detail buku pada halaman BookDetail, selanjutnya kita akan membuat halaman yang akan menangani proses Inser, Delete dan Update.

Pada halaman ini, kita akan membuat fungsi untuk meng-upload gambar (cover) dari buku. Oleh karena itu, kita perlu menambahkan package “BlazorInputFile“ pada project. Selanjutnya tambahkan Javascript reference pada file “_Host.html“ untuk “BlazorInputFile“ seperti dibawah ini.

```
<script src="_content/BlazorInputFile/inputfile.js"></script>
```

Dan agar komponen ini dapat digunakan pada project ini, pastikan untuk melakukan import “BlazorInputFile“ pada file “_Import.razor“ (`@using BlazorInputFile`).

Selanjutnya kita akan membuat service untuk melakukan proses upload. Tambahkan sebuah Interface pada folder Data dengan nama “IFileUpload“ dan di-ikuti dengan class implentasinya (FileUpload).

```
public interface IFileUpload
{
    Task UploadAsync(MemoryStream file, string fileName);
}

public class FileUpload : IFileUpload
{
   private IWebHostEnvironment _environment;
   public FileUpload(   IWebHostEnvironment   environment)
   {
      _environment = environment;
   }

   public async Task UploadAsync(MemoryStream file,
   string fileName)
   {
       var uploads =
       Path.Combine(_environment.WebRootPath,
       "images", fileName);

       await using FileStream fs = new
       FileStream(uploads, FileMode.Create,
       FileAccess.Write);
       file.WriteTo(fs);
    }
}
```

Ketikkan sintaks pada interface dan class (FileUpload) seperti diatas. Pada saat pengguna meng-upload gambar, maka gambar tersebut akan di-copy kedalam folder wwwroot/images. Dan terakhir tambahkan service ini kedalam method ConfigureServices pada file Startup.cs. (`services.AddTransient<IFileUpload, FileUpload>();`).

Sampai disini kita telah selesai menambah service untuk melakukan proses Upload file. Selanjutnya kita akan melanjukan dengan membuat halaman BookEdit, ikuti langkah-langkah dibawah ini.

Tambahkan sebuah class dengan nama BookEditBase pada folder Pages.

```
public class BookEditBase: ComponentBase
{

   [Inject]
   public IFileUpload fileUpload { get; set; }

   [Inject]
   public IBookRepository BookRepository { get; set; }

   [Inject]
   public ICategoryRepository CategoryRepository { get; set; }

   [Inject]
   public NavigationManager NavigationManager { get; set; }

   [Parameter]
   public string BookId { get; set; }

   public Book Book { get; set; } = new Book();
   public List<Category> Categories { get; set; } = new
   List<Category>();

   protected string CategoryId = string.Empty;
   protected IFileListEntry file;
   protected MemoryStream fs;
   protected string ImageData = string.Empty;
}
```

Pada sintaks diatas, kita akan menggunakan 3 service yang telah dibuat sebelumnya (IFileUpload, IBookRepository dan ICategoryRepository). Sedangkan untuk melakukan Navigasi pada class ini digunakan NavigationManager.

```
protected override async Task OnInitializedAsync()
{

    Categories = (await CategoryRepository.GetAll()).ToList();
    int.TryParse(BookId, out var bookId);
    if (bookId == 0)
    {
         Book = new Book();
    }
    else
    {
          Book = await BookRepository.
          GetBookById(int.Parse(BookId));
          ImageData = $"images/{Book.Gambar}";
    }
     CategoryId = Book.CategoryID.ToString();
}
```

Didalam method OnInitializedAsync, terdapat beberapa baris code. Yang pertama mengambil data Category yang akan digunakan pada halaman. Pada halaman ini akan terdapat sebuah dropdown yang akan menampilkan data Category. Lalu ada sintaks untuk mengecek apakan property BookId bernilai 0 atau tidak, hal ini dilakukan untuk

mengecek, jika tidak sama dengan "0" maka akan melakukan pencarian data buku berdasarkan BookId.

```
protected async Task HandleValidSubmit()
{
     Book.CategoryID = int.Parse(CategoryId);
     if (Book.BookID == 0)
     {
         await BookRepository.Insert(Book);
     }
     else
     {
          await BookRepository.Update(Book);
     }
     if (file!=null && file.Data.Length>0)
     {
         await fileUpload.UploadAsync(fs,Book.Gambar);
     }

     NavigationManager.NavigateTo("/BookList");
}
```

Langkah selanjutnya dengan membuat method yang digunakan pada button Save, yaitu method dengan nama "`HandleValidSubmit`". Pada method ini digunakan untuk menyimpan data buku, baik itu data baru maupun data yang sudah ada. Untuk mengetahui proses apa yang harus dilakukan, kita gunakan property BookID pada class Book. Jika 0 maka kita gunakan untuk menambahkan data, jika tidak sama dengan 0 maka proses yang dipanggil adalah proses update (perubahan data). Masih pada method "`HandleValidSubmit`" terdapat baris code untuk menyimpan file (gambar) ke server. Setelah proses simpan selesai maka akan kembali kehalaman "BookList" (`NavigationManager.NavigateTo("/BookList");`).

Untuk method FileUpload digunakan untuk meng-konversi IFileListEntry menjadi MemoryStream.

```
protected async Task  FileUpload(IFileListEntry[] files)
{
    file = files.FirstOrDefault();
    if (file != null)
    {
        var ms= new MemoryStream();
        await file.Data.CopyToAsync(ms);
        fs = ms;
        ImageData = $"
        {Convert.ToBase64String(ms.ToArray())}";
        Book.Gambar = file.Name;
    }
}
```

Setelah selesai dengan class diatas, selanjutnya tambahkan sebuah Razor komponen dengan nama BookEdit.razor. Pada artikel ini akan dijelaskan bagian-bagian penting dari sintaks yang ada pada file ini. Untuk lebih detail mengenai sintaks pada file ini dapat dilihat pada project lampiran.

```
@page "/bookedit"
@page "/bookedit/{BookId}"
@inherits BookEditBase
```

Pada baris pertama dan kedua merupakan navigasi kehalaman BookEdit, dimana pada halaman ini bisa dilakukan baik dengan menggunakan parameter maupun tanpa parameter.

```
<EditForm Model="@Book"
          OnValidSubmit="@HandleValidSubmit">

</EditForm>
```

Halaman ini menggunakan komponen EditForm untuk mengelola form seperti validasi atau *form submission event*. Untuk sintaks diatas kita menggunakan dua properties dari EditForm, yaitu Model dan OnValidSubmit.

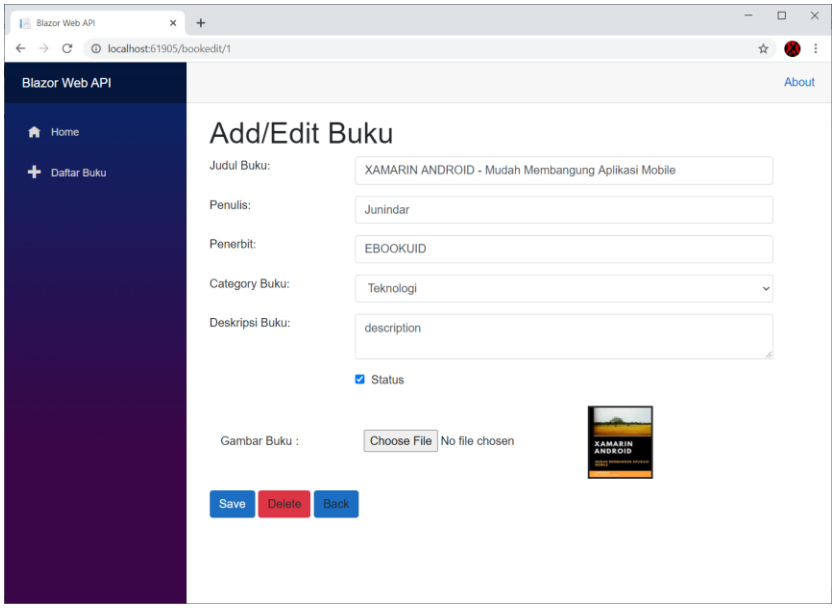
Untuk melakukan “binding” antara property pada model dengan html control seperti InputText, InputSelect dan lain-lain kita gunakan atribut “@bind-Value”, seperti pada contoh InputText Judul dibawah.

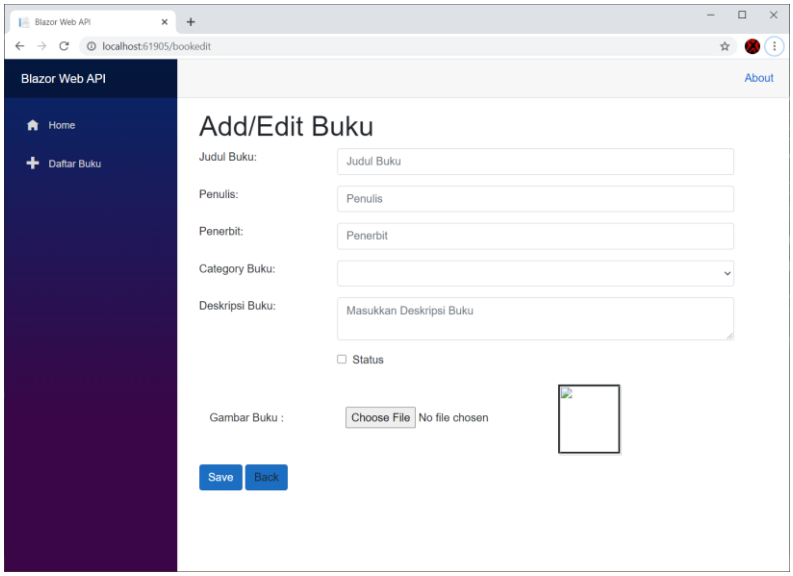
```
<InputText id="Judul" @bind-Value="@Book.Judul"
class="form-control col-sm-8" placeholder="Judul Buku"
required></InputText>
```

Pada Category untuk menampilkan data category pada dropdownlist, kita gunakan cara seperti dibawah.

```
<InputSelect id="category" class="form-control col-sm-8"
@bind-Value="@CategoryId" required>
@foreach (var cat in Categories)
{
      <option value="@cat.CategoryID">
      @cat.NamaCategory</option>
}
</InputSelect>
```

Setelah selesai dengan halaman ini, jalankan program untuk mendapatkan hasil seperti pada gambar dibawah ini. Pastikan proses Insert dan Update dapat berjalan dengan baik





Silahkan lanjutkan latihan ini, dengan membuat proses Delete. Pastikan button delete hanya tampil pada saat proses Edit, sedangkan untuk Add/New hanya menampilkan dua button seperti diatas.

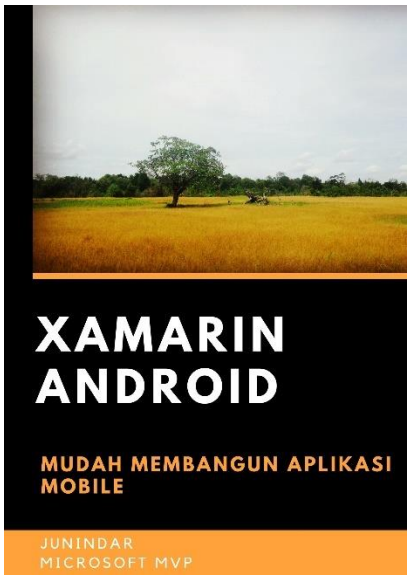
Penutup

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/2021/05/mengakses-web-api-dari-blazor.html

Semoga bermanfaat.

Wassalam.

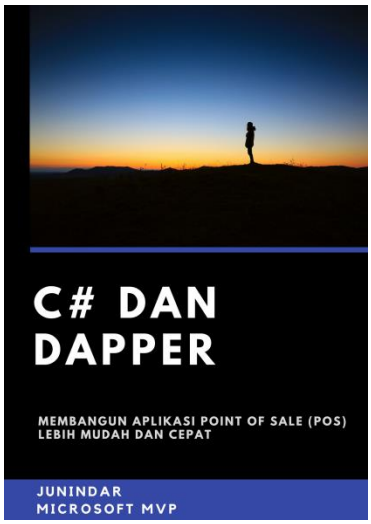
Referensi



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

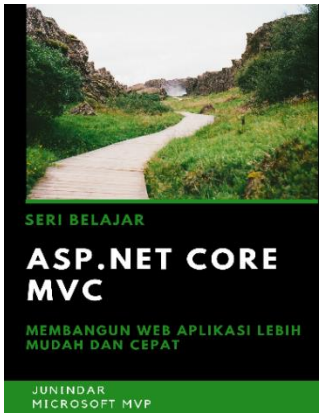
https://play.google.com/store/books/details?id=VSLiDQAAQBAJ

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

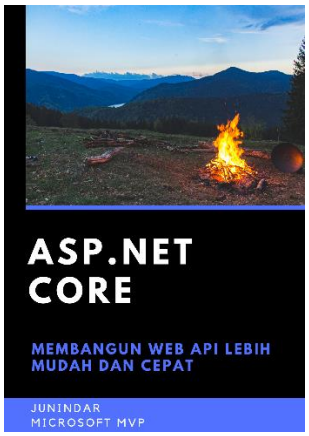


https://play.google.com/store/books/details/Junindar_C_dan_Dapper_Membangun_Aplikasi_POS_Point?id=6TErDwAAQBAJ

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



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

Authentication dan Authorization pada Blazor
Junindar, ST, MCPD, MOS, MCT, MVP .NET

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".**