Create, Read, Update dan Delete (CRUD) 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 .

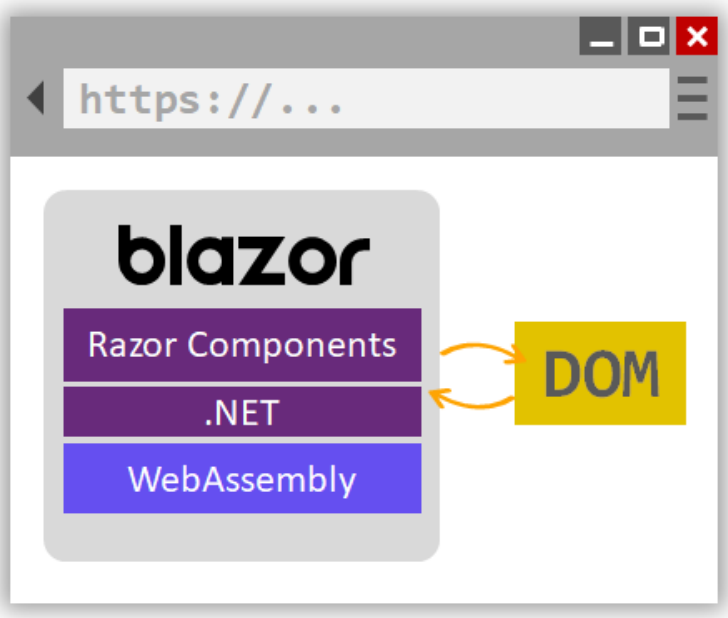
Code .Net berjalan pada WebAssembly, yang artinya kita dapat menjalankan “NET“ didalam browser (Client) tanpa harus menginstall plugin seperti Silverlight, Java mapun Flash.

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 .

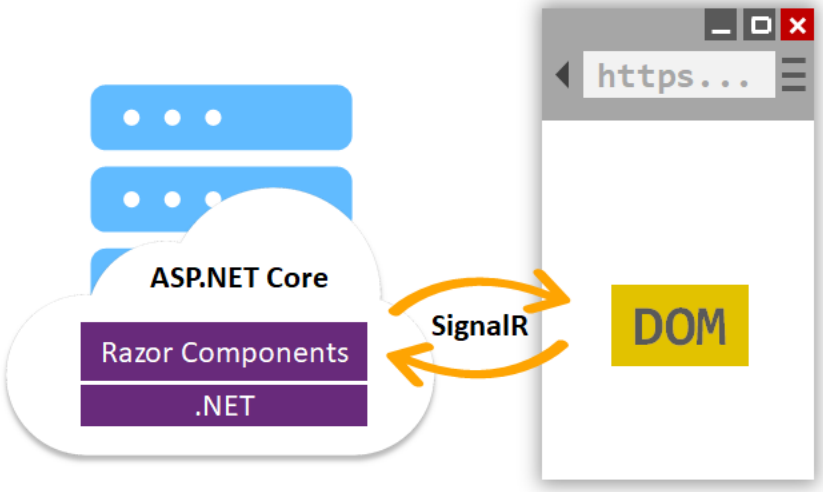
Code .Net berjalan pada Web Assembly, yang artinya kita dapat menjalankan "NET" didalam browser (Client) tanpa harus menginstall plugin seperti Silverlight, Java mapun Flash.

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 artikel sebelumnya http://junindar.blogspot.com/2020/02/pengenalan-blazor.html. Pada artikel ini tidak menjelaskan apa itu blazor dan cara membuat project dengan menggunakan template blazor pada Visual Studio.

Artikel ini akan focus bagaimana melakukan proses CRUD pada blazor dengan menggunakan EF Core. Untuk mengetahui penggunaan EF Core silahkan mengikuti telerbih dahulu pembahasan EF Core pada Ebook ini. http://junindar.blogspot.com/2019/10/aspnet-core-mvc-membangun-aplikasi-web.html

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

Link untuk Project Lampiran ada pada Penutup.

- Tambahkan sebuah folder Entity pada project. Didalam folder ini akan terdapat class Entity Category dan Book, seperti pada latihan sebelumnya.
- Lalu pada folder Data, tambahkan class “PustakaDbContext“ dan ketikkan sintaks seperti dibawah ini.

```
public class PustakaDbContext : DbContext
{
    public PustakaDbContext(DbContextOptions<PustakaDbContext> options) : base(options)
    {

    }
    public DbSet<Category> Categories { get; set; }
    public DbSet<Book> Books { get; set; }

}
```

Pada sintaks diatas dapat dilihat bahwa class "PustakaDbContext" diturunkan dari class "DbContext" dan terdapat property DbSet<TEntity> dari class Book yang telah kita buat pada chapter sebelumnya. Dan pastikan sebelumnya kita import terlebih dahulu Microsoft.EntityFrameworkCore untuk class ini (using Microsoft.EntityFrameworkCore;).

Langkah selanjutnya adalah dengan membuat connection string pada file appsettings.json. Seperti yang telah dijelaskan pada chapter sebelumnya file ini digunakan untuk tempat untuk menyimpan setting pada project yang dibuat. Seperti contoh koneksi ke database, connection string kedatabase dapat kita letakkan di file ini.

```
{
  "Logging": {
    "LogLevel": {
      "Default": "Information",
      "Microsoft": "Warning",
      "Microsoft.Hosting.Lifetime": "Information"
    }
  },
  "AllowedHosts": "*",
  "ConnectionStrings": {
    "pustakaConnection": "data source=localhost;Integrated Security=SSPI;initial catalog=BlazorDB"

  }
}
```

Agar aplikasi dapat membaca connection string pada file diatas, ada beberapa hal yang harus dilakukan seperti berikut. Buka file "Startup.cs" dan tambahkan sintaks berikut.

```
public IConfiguration Configuration { get; }

public Startup(IConfiguration configuration)
{
    Configuration = configuration;
}
```

Selanjutnya pada method "ConfigureServices" kita tambahkan sintaks untuk mendaftarkan EF Core pada ServicesCollection dimana database yang digunakan

adalah SQL Server. Dengan menggunakan AddDbContext kita mengirimkan class "PustakaDbContext" yang telah dibuat sebelumnya.

```
services.AddDbContext<PustakaDbContext>(options =>
       options.UseSqlServer(Configuration.GetConnection
String("pustakaConnection")));
```

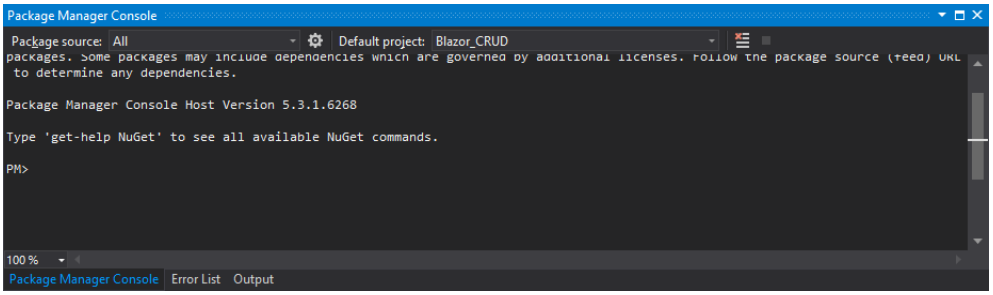
Ada beberapa hal yang perlu diketahui dari langkah-langkah di atas. Pada saat ASP.NET Core dijalankan, secara otomatis akan membaca file appsettings.json dan dikonversi menjadi "Configuration" (Microsoft.Extensions.Configuration.IConfiguration) sehingga informasi-informasi yang ada file tersebut dapat diakses dari aplikasi. Sedangkan "GetConnectionString" adalah method untuk mendapatkan section dari "ConnectionStrings" dengan parameter nama dari connection string.

- Masih pada folder Data, tambahkan dua buah Interface dengan nama masing-masing ICategoryRepository dan IBookRepository.

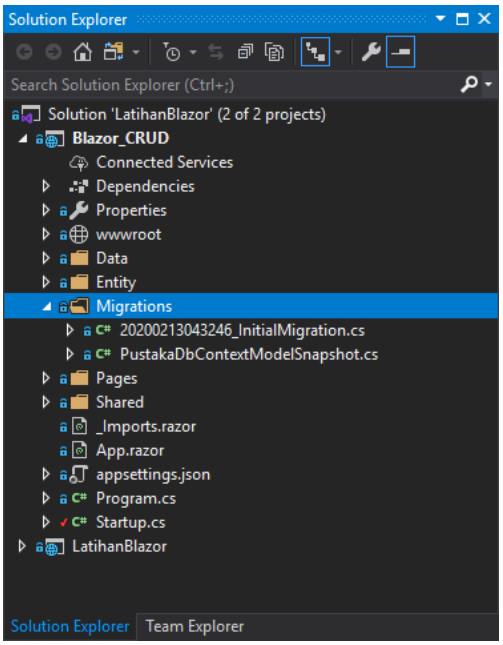
```
public  interface ICategoryRepository
{
      Task<List<Category>> GetAll();
      Task<Category> GetById(int categoryId);
}
public interface IBookRepository
{
        Task Insert(Book book);
        Task Update(Book book);
        Task Delete(int bookId);
        Task<IEnumerable<Book>> GetAllBooks();
        Task<Book> GetBookById(int bookId);
}
```

Interface tidak memiliki implementasi dari sebuah method, untuk membuat implementasi dari method-method pada interface diatas, kita perlu membuat Class untuk masing-masing interface. Sintaks implemetasi dapat dilihat pada project lampiran.

- Setelah selesai dengan langkah-langkah diatas, selanjutnya kita akan membuat database dengan melakukan migrations. Disini kita akan membuat database dengan menggunakan EF Core. Dalam membuat database dengan EF Core akan kita gunakan "Package Manager Console" untuk melakukan migrasi, lalu untuk membuka Package Manager klik View > Other Windows > Package Manager Console.



Sebelum kita melakukan migrasi, pastikan project sudah di Build terlebih dahulu. Untuk melakukan migrasi kita akan menggunakan command pada Package Manager Console. Ketikkan command berikut "add-migration InitialMigration" pada console dan enter. "InitialMigration" adalah nama dari migrasi yang akan kita buat. Setelah menunggu beberapa saat pada project kita akan bertambah sebuah folder dengan nama "Migrations" beserta terdapat beberapa file didalamnya.

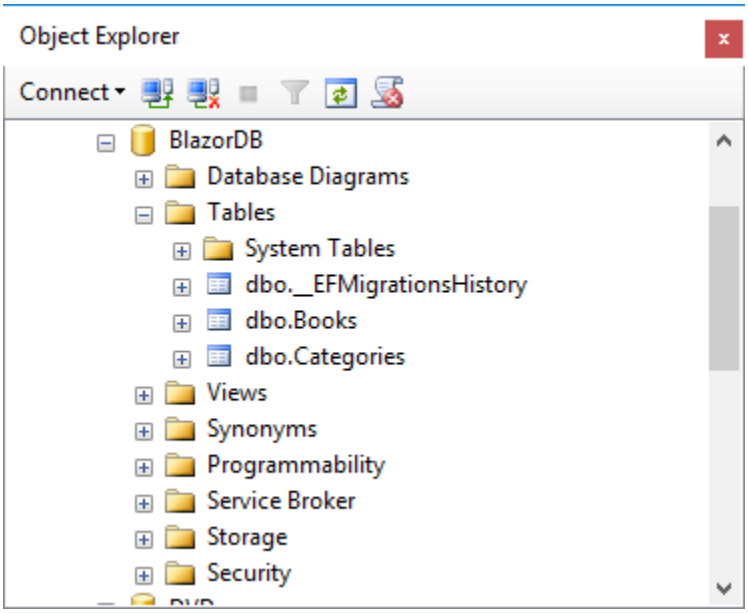


Pada file "_InitialMigration.cs" terdapat dua buah method "Up" dan "Down". Method Up berisi sintaks untuk membuat table pada database, untuk latihan ini hanya terdapat 1 table dengan nama "Books". Sedangkan untuk method Down terdapat sintaks untuk "Drop" table.

Selanjutnya kita akan mengeksekusi sintaks pada file diatas, agar database kita dibuat. Masih menggunakan console, ketikkan command "update-database" lalu enter. Tunggu beberapa saat hingga proses pembuatan database dan table selesai.

Untuk memastikan database telah dibuat, buka SQL Server Management Studio dan lihat apakah database dengan nama “BlazorDB” ada pada daftar database seperti gambar dibawah ini.

Pada gambar dibawah terdapat tiga buah table di dalam database. Yang pertama table “Books” dan “Categories“ yaitu table yang dibuat berdasarkan model pada project. Sedangkan yang kedua adalah table “__EFMigrationsHistory” adalah table history dari proses migrasi pada EF Core. Selanjutnya jalankan program untuk melihat hasilnya. Pastikan tidak ada error pada saat program dijalankan.



Dengan menggunakan EF Core kita dapat mengisi data pada table yang kosong saat aplikasi dijalankan. Sebelumnya buat terlebih dahulu static class dan method. Tambahkan sebuah class dengan nama “DbInitializer” pada folder Data. Untuk detail sintaks dapat dilihat pada project lampiran.

```
public static class DbInitializer
{
  public static void Seed(PustakaDbContext context)
  {
     if (!context.Categories.Any())
     {

          var categories = new List<Category>
          {
               new Category { NamaCategory = "Agama" },
               new Category { NamaCategory = "Bahasa" },

          };

          context.Categories.Add(cat1);
          context.Categories.Add(cat2);


          context.SaveChanges();


     }
  }
}
```

Pada static class diatas, terdapat sebuah static method dengan nama “Seed” dan memiliki sebuah parameter dari “PustakaDbContext” yang telah dibuat sebelumnya. Didalam method ini terdapat sebuah kondisi, jika table Books kosong maka akan diisi datanya. Untuk detail sintaks pada class ini dapat dilihat di project lampiran. Dan bagaimana mengeksekusi method “Seed” ini pada saat aplikasi dijalankan? Buka file Program.cs, pada method “Main” hapus sintaks yang ada di dalam method tersebut. Dan ketikkan sintaks dibawah sebagai penggantinya.

```
var host = CreateWebHostBuilder(args).Build();

using (var scope = host.Services.CreateScope())
{
      var services = scope.ServiceProvider;

      var context = services.GetRequiredService<PustakaDbContext>();
      DbInitializer.Seed(context);
}
host.Run();
```

Pada method Seed yang sebelumnya telah kita buat, disana terdapat sebuah parameter “PustakaDbContext”, sehingga untuk menggunakan method tersebut kita harus mengirimkan “PustakaDbContext” kedalam method Seed (DbInitializer.Seed(context);). Dimana sebelumnya kita gunakan “dependency injection container” untuk mendapatkan “PustakaDbContext”.

- Sampai disini kita telah membuat Database dan fungsi-fungsi CRUD pada EF Core yang nantinya akan digunakan pada Blazor.
- Tambahkan sebuah class dengan nama BookListBase.cs.Class ini merupakan code behind dari Razor component yang digunakan untuk menampilkan daftar buku pada table.

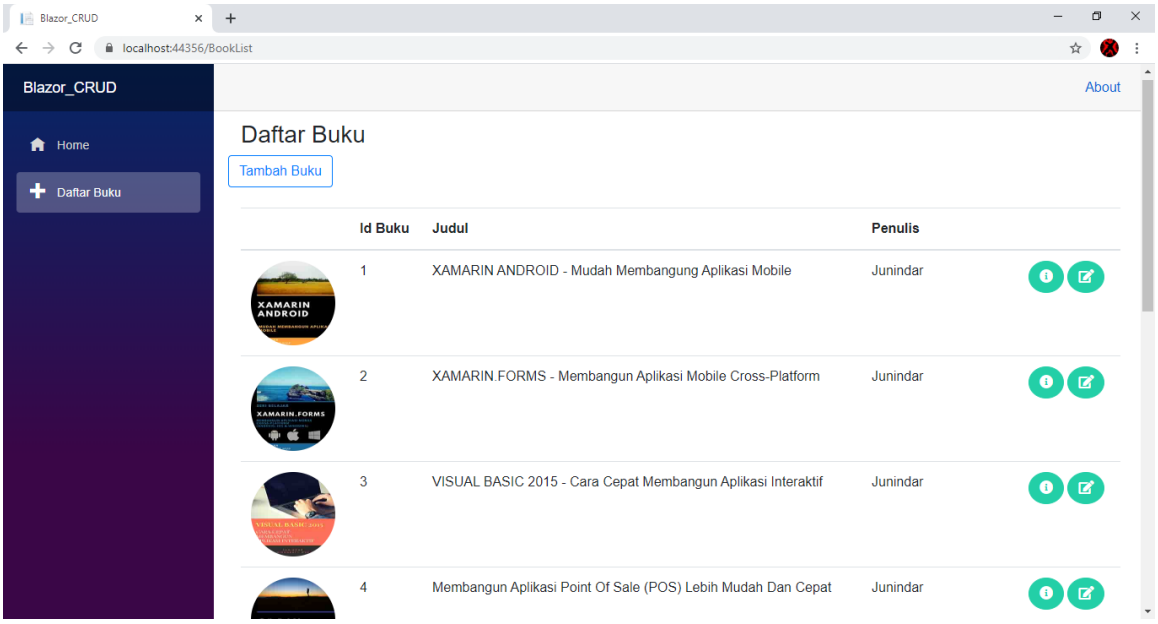
```
public class BookListBase : ComponentBase
{
        [Inject]
        public IBookRepository BookRepository { get; set; }

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

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

Pada class ini terdapat sebuah attribute Inject dari Interface IBookRepository. Dimana agar kita bisa menggunakan method-method dalam interface tersebut. Pada class ini kita akan menggunakan method GetAllBooks yang hasilnya akan ditampung pada property Books.

- Tambahkan sebuah Razor Component dengan nama BookList.razor.

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

Tambahkan sintaks diatas pada file yang baru saja kita buat pada baris paling atas sintaks. Baris pertama digunakan untuk menavigasi ke BookList. Sedangkan pada baris kedua digunakan untuk menghubungkan dengan code behind (BookListBase). Dan hapus code block pada file ini. Pada halaman ini terdapat sebuah link untuk menuju kehalaman membuat data buku baru. Sedangkan pada masing-masing baris buku terdapat dua buah button Detail dan Edit, seperti gambar dibawah.



- Setelah selesai dengan halam daftar buku, selanjutnya kita akan membuat halaman untuk Create, Update dan Delete buku. Tambahkan sebuah class dengan nama BookEditBase.

```
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;
}
```

Sebelum kita lanjutkan pada class ini terlebih dahulu kita buat sebuah interface dan class masing-masing bernama IFileUpload dan FileUpload. Class ini digunakan untuk melakukan proses upload gambar buku ke server. Detail sintaks untuk class ini dapat dilihat pada project lampiran.

Kembali ke class BookEditBase

```
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.

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");`).

```
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");
}
```

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.

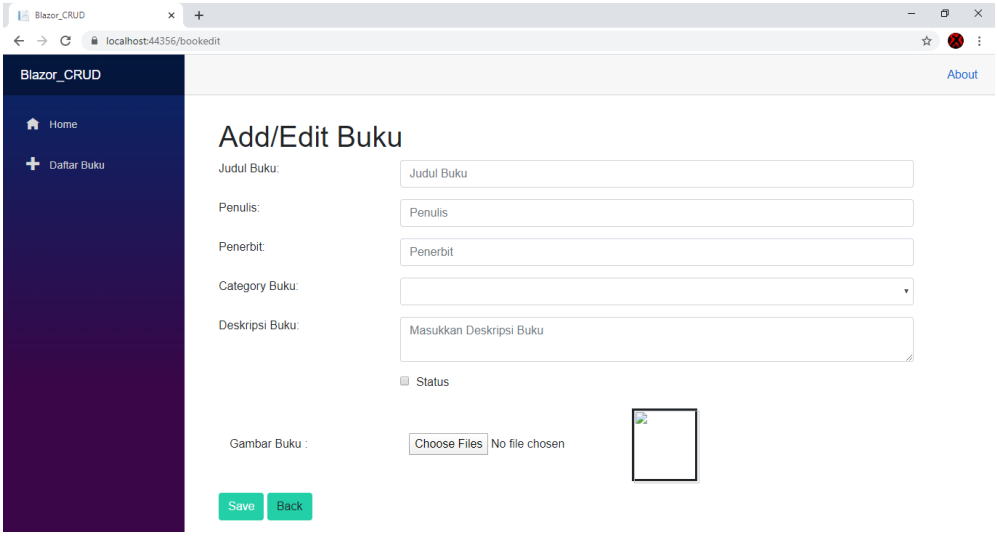
Untuk melakukan “binding” antara property pada model dengan html control seperti InputText, InputSelect dan lain-lain kita gunakan atribut “@bind-Value”, seperti pada contoh InputText Judul dibawah.

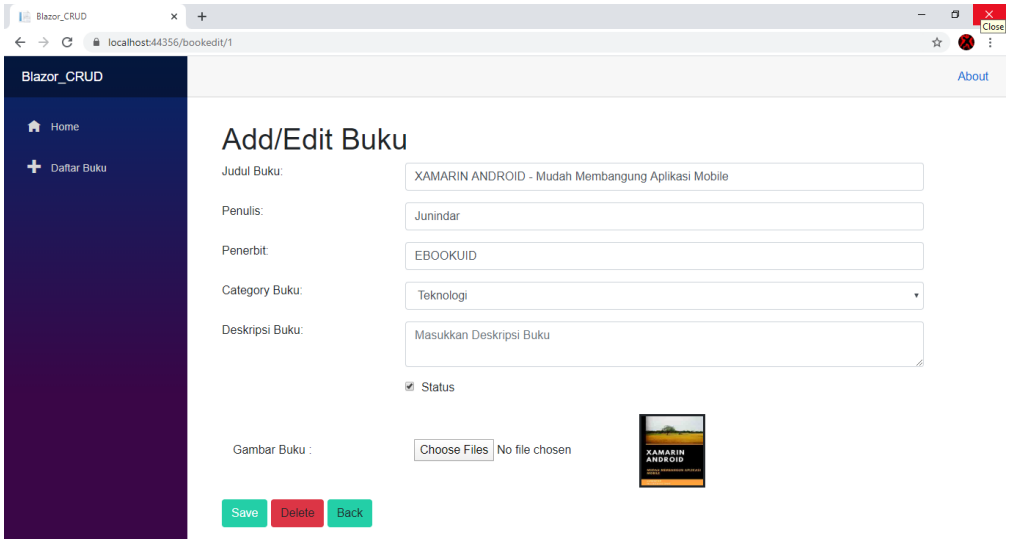
```
<InputText id="Judul" @bind-Value="@Book.Judul"
class="form-control col-sm-8" placeholder="Judul Buku"
required></InputText>
```

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.

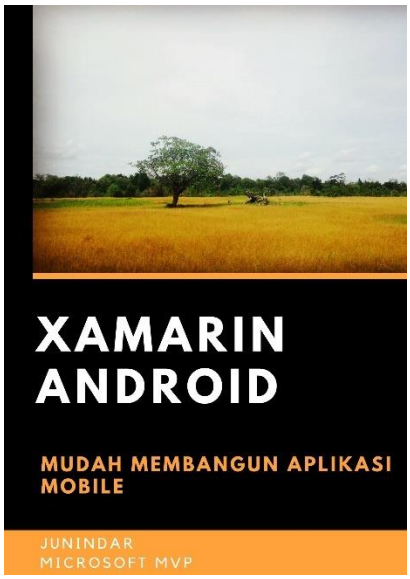




Penutup

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/2020/02/create-read-update-dan-delete-crud-pada.html

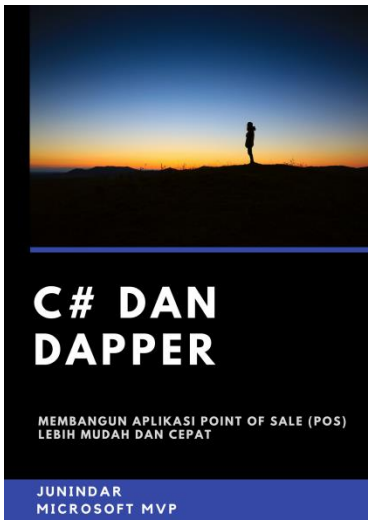
Referensi



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

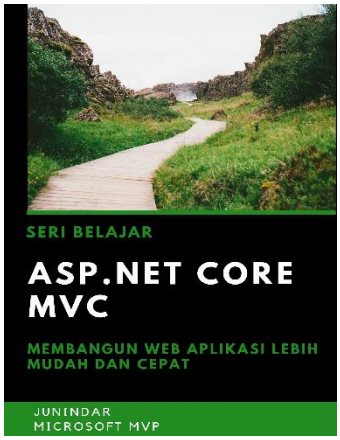
https://play.google.com/store/books/details?id=VSLiDQAAQBAJ

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

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".**