**Tutorial C# dan ASP.Net**

Sejarah ASP.Net

ASP -> ASP.NET Webform -> ASP.NET MVC -> ASP.NET Core

Untuk membuat aplikasi .net core, kita bisa membuat dengan 2 cara di dalam visual studio.

Yaitu:

1. Razor Pages Application
2. MVC Application

Pre-requisites:

1. Visual Studio Comunity
2. Dot.net core 3.1
3. SQL Server 2019 (developer type)
4. Manajement Studio SQL Server Client

**1. Razor Page**

Razor pages adalah new feature di ASP.NET Core yang bertujuan untuk membuat pembuat aplikasi lebih berfokus ke skenarion yg lebih mudah.

Membuat applikasi baru Razor pages

- Create new project

- Pilih ASP>NET Core Web Application

- Beri nama project kita

- Pilih Web Application

- Klik Create

Bagian di dalam razor pages:

1. Razor Page (UI/VIEW)
2. Page Model (Contains Handlers)

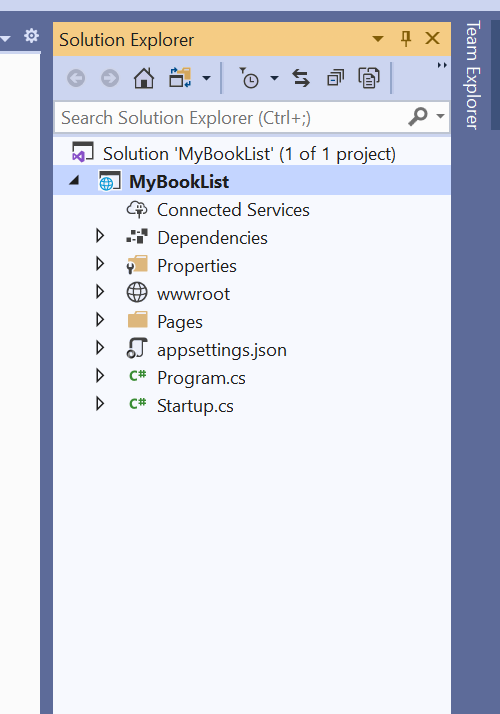
Note:

*Didalam Razor aplikasi Model dan View berada didalam folder Page. Model berada dibawah View.*

*View berektensi .cshtml sedangkan Model akan selalu .cshtml.cs (class)*

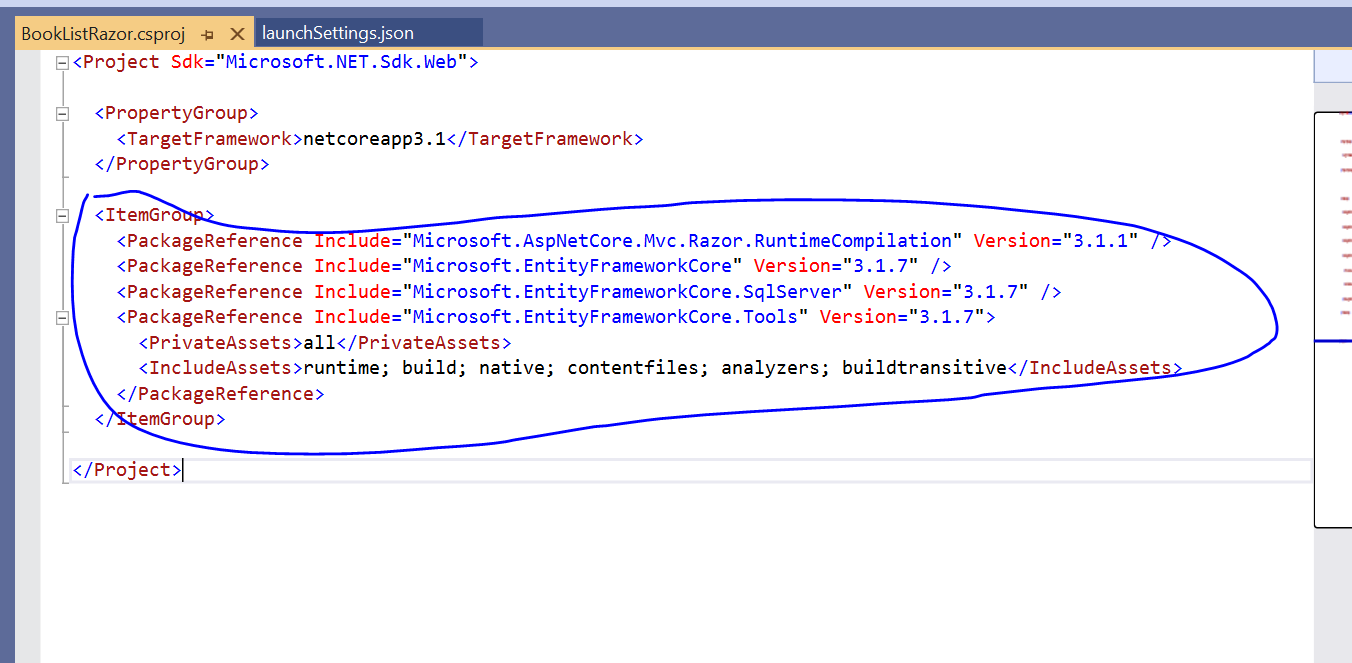
Structur Folder:

Default



Pembahasan:

Jika kita double klik di nama project, maka akan muncul semua nugget package yang sudah kita tambah.



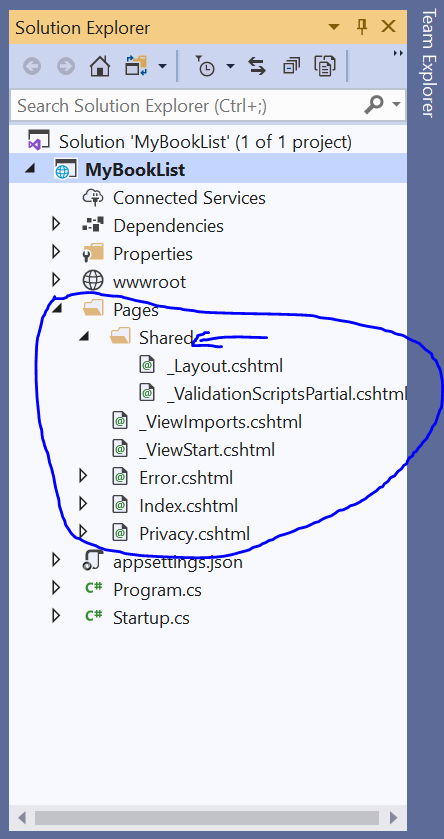
Seperti yang terlihat didalam kolom, itu tidak ditambah manual, itu bertambah otomatis ketika kita menambah nugget package dari tools->nugget package

wwwroot Folder:

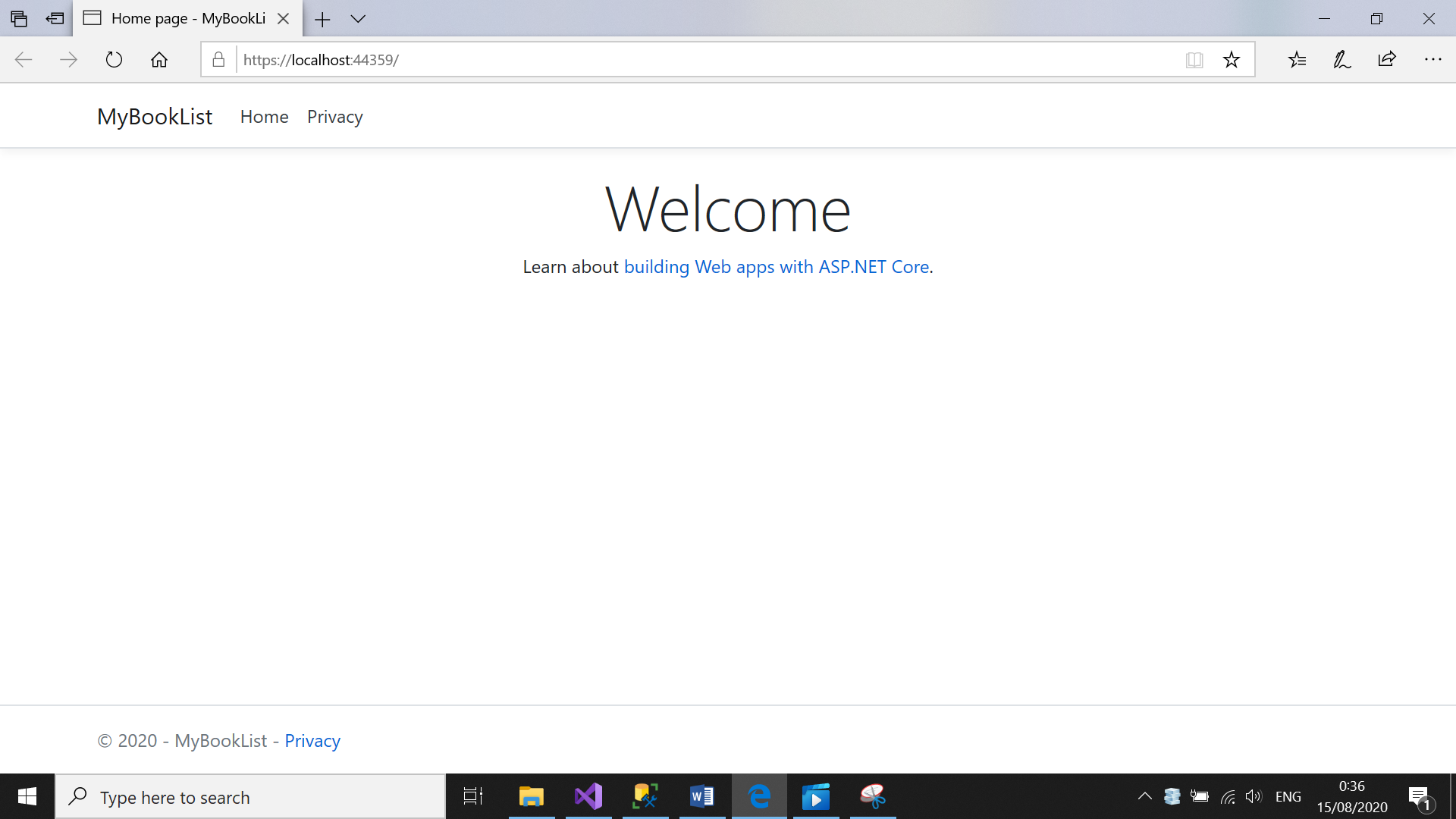
Berisi file static, sperti: Css, Js, Image, dll.

Pages:

Pages merupakan **main** folder atau folder utama di dalam aplikasi razor.



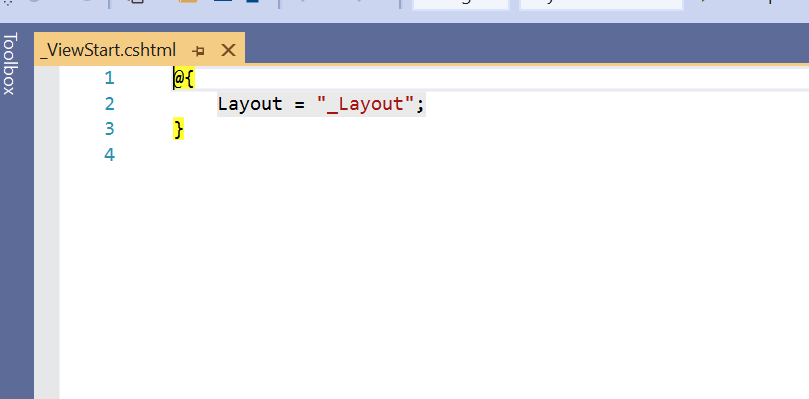
Didalam folder Pages, terdapat folder Shared. Di dalam folder shared, ada file \_Layout.cshtml dan \_ValidationScripsPartial.cshtml. File \_Layout.cshtml inilah yang merupakan file view utama ketika pertama kali aplikasi kita dijalankan seperti berikut atau boleh dikatakan seperti dashboard,



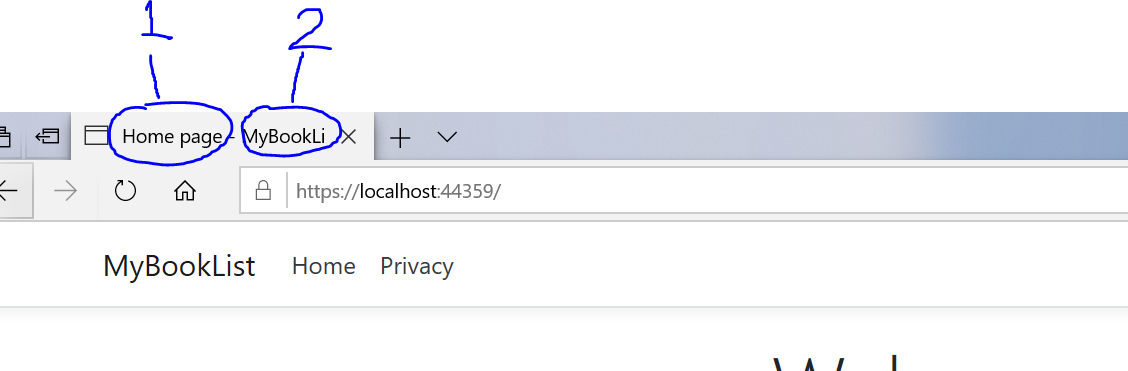
Namun kita bisa mengganti file mana yang kita ingin tampil pertama ketika aplikasi di jalankan. Kita bisa menggantinya dari file \_ViewStart.cshtml yang ada didalam folder pages (di luar folder Shared).

Kemudian di luar folder Shared (masih di dalam folder Pages) ada file \_viewImports.cshtml. Itu berisi code-code untuk menginput helper (tag helper).

Disana juga kita melihat ada file \_ViewStart.cshtml. Disini kita bisa mengganti file mana yang akan tampil pertama ketika aplikasi dijalankan seperti yang sudah dijelaskan di atas.



Kalau kita perhatikan pemberian nama title websitenya:



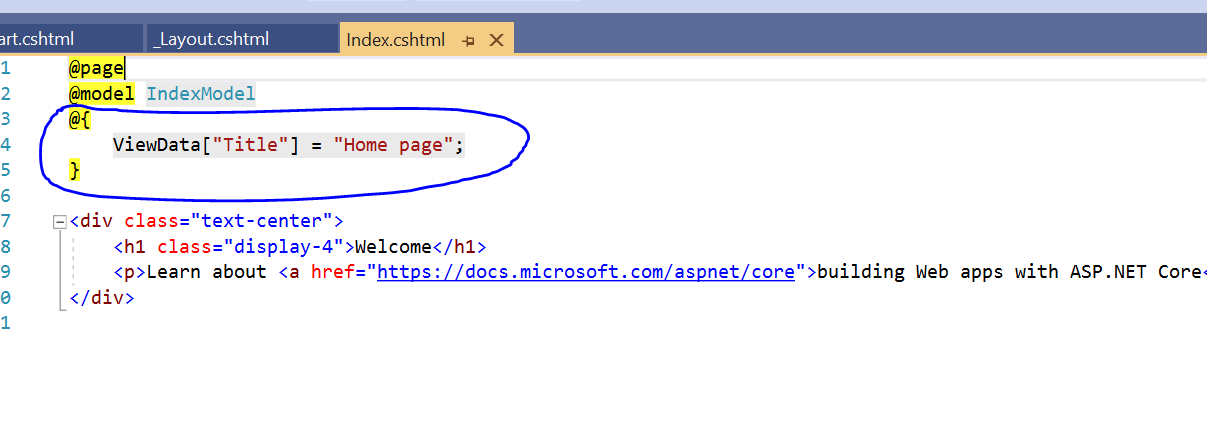
Kalua kita lihat nomor 1 itu, itu dibuat dengan cara interpolasi dari halaman yang berbeda. Perhatikan code berikut ini:

Halaman utama (\_Shared.cshtml)



Halaman sumber interpolasi :

Index.cshtml



**Routing**

Pages: Halaman utama

File default : Index.cshtml

url:

[www.domain.com](http://www.domain.com) -> akan menghasilkan /Pages/index.cshtml

[www.domain.com](http://www.domain.com)/index -> akan menghasilkan /Pages/index.cshtml

[www.domain.com](http://www.domain.com)/account -> akan menghasilkan /Pages/account.cshtml, jika account.cshtml tidak

maka, akan dicarri /Pages/account/index.cshtml (dengan kata lain)

account akan dianggap sebagai folder

**Tag Helpers**

Asp-page : untuk routing, contoh: asp-page=”/Index”

**Appsettings.json**

File konfigurasi utama dalam razor pages adalah appssettings.json. Difile ini kita akan membuat settingan untuk project kita, seperti connections string (koneksi ke database).

**Dependency Injection**

Merupakan sebuah container/ pembungkus library. Contoh real, ketika kita ingin melakukan backpacking, maka kita akan meletakkan perlengkapan kita kedalam tas backpacking kita. Seperti senter, tenda, dll. Tas merupakan container atau dependency injection.

Berikut gambaran dependency injection di dalam aplikasi.

Kita punya 3 halaman, dimana masing-masing 3 halaman ini memiliki fitur logs, send email, dan save data.

Jika dengan cara klasik, maka kita akan membuat object logs, send email, dan save di 3 halaman tersebut dan itu akan membuat kodingan kita banyak. Perhatikan gambar berikut ini:

Page 1

Email

Log

Page 2

Page 3

Save

Jika menggunakan depency injection.

Page 1

Email

Log

Save

Depency Injection

Page 2

Page 3

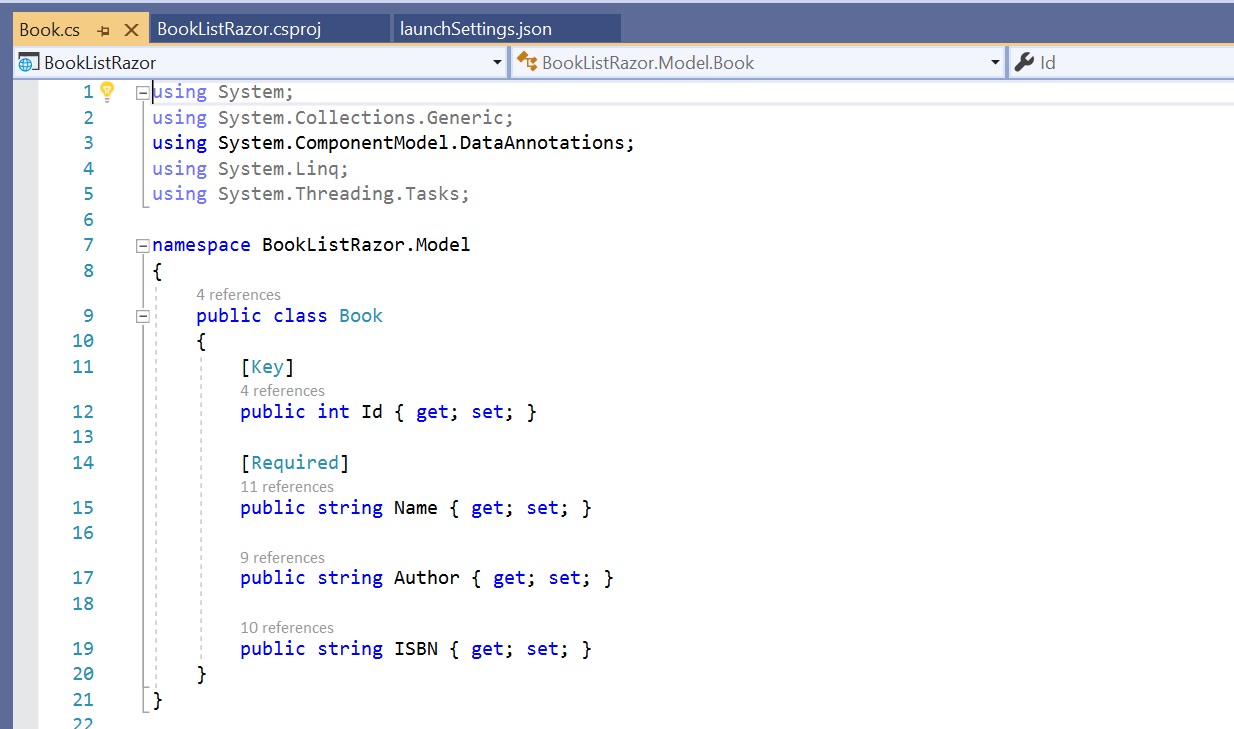
Jika kita perhatikan gambar diatas, semua feturnya (sebelah kiri) pertama di register dulu didalam container dependency injection, dari situ kemudian di berikan ke dalam setiap halaman(pages). Sehingga lebih simple dan tidak perlu create object berkali-kali.

Membuat Model.

Model akan membuat/create Database ke dalam sql server kita. Konsep pembuatan database adalah seperti didalam laravel yaitu menggunakan system migrate. Dengan kata lain, kita tidak langsung membuat database dan table di dalam sql server. Kita akan membuat structur database dari code di C# nya sendiri.

Didalam Razor page sendiri, folder Model belum ada. Dengan demikian kita harus tambah manual. Setelah folder model di create, didalamnya tambahkan sebuah class.

Contoh code model Book.



Jika kita perhatikan, itu merupakan class Book. Didalam class itu kita punya field-field yang akan di create didalam database. Field tersebut adalah:

[Key]

public int Id { get; set; } # key dalam hal ini menyatakan primary key.

[Required]

public string Name { get; set; } # key dalam hal ini menyatakan primary tidak boleh null.

public string Author { get; set; }

public string ISBN { get; set; }

dengan dengan cara diatas kita sudah merancang sebuah table namanya: Book. Table tersebut akan memiliki field: Id (PK), Name (Reuqired), Author type string, IBN type string.

Setelah rancanangan tablenya sudah ok, kita perlu menginstall beberapa **package** dan membuat **connectionString**nya. Package tersebut antara lain:

1. Microsoft.EntityFrameworkCore : Kita menggunakan entity framework untuk mengakses database
2. Microsoft.EntityFrameworkCore.SqlServer
3. Microsoft.EntityFrameworkCore.Tools : ini digunakan untuk proses migration

Setelah packagenya sudah terinstal, sekarang kita akan membuat code untuk **connectionString** nya.

Caranya, buka file appsettings.json, kemudian tambahkan code berikut sebelum Key Logging:

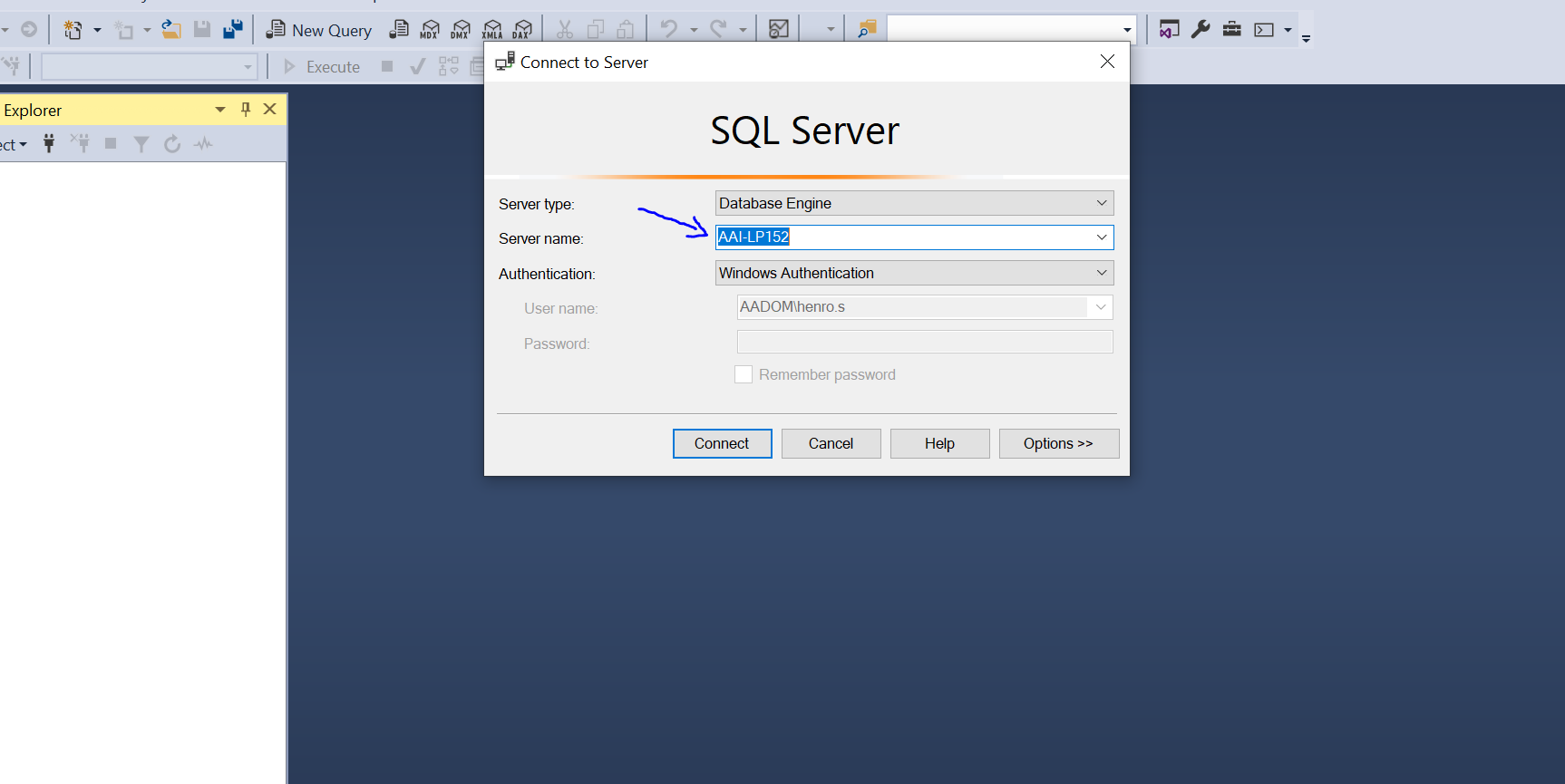
"ConnectionStrings": {

"DefaultConnection": "Server = AAI-LP152; Database=BookListRazor; Trusted\_Connection=True;MultipleActiveResultSets=True"

},

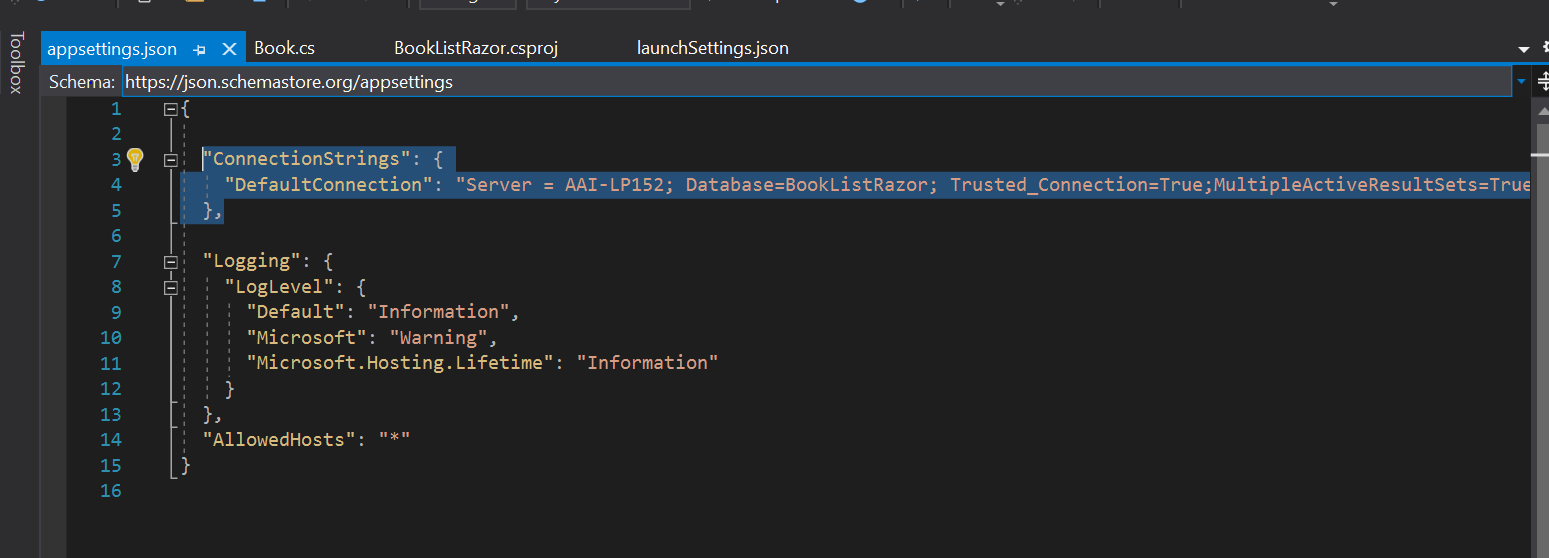
Note:

AAI-LP152 : ini merupakan nama koneksi untuk ke sql server kita. Ini bisa didapat ketika kita membuka sql server manajemen studio (client sql server).



BookListRazor : nama database yang akan tercreate dibatase setelah kita melakukan migration. Dengan kata lain database tersebut belum ada didatabase. Pastikan kita tidak membuat database langsung di sql server.

Sehingga code appsettings.json terlihat seperti ini:



Setelah connectionstring sudah kita buat, sekarang kita perlu mengkonfigurasikannya di file Startup.cs. Namun untuk konfigurasinya kita memerlukan sebuah kelas yang namanya: **ApplicationDbContext.cs**

Berikut code yang ada di dalam file tersebut:

public class ApplicationDbContext : DbContext => **DbContext merupakan class dari entity framework**

{

public ApplicationDbContext(DbContextOptions<ApplicationDbContext> options) : base(options)

{

}

public DbSet<Book> Book { get; set; }

}

Setelah file applicationdbcontext sudah siap, kemudian kita harus menambah class tersebut didalam starup.cs. letakkan code berikut di dalam method **ConfigureServices.**

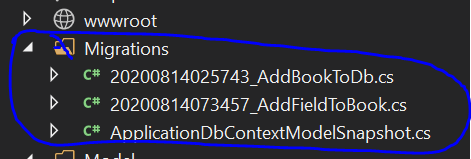
ervices.AddDbContext<ApplicationDbContext>(option => option.UseSqlServer(Configuration.GetConnectionString("DefaultConnection")));

Setelah langkah di atas sudah siap, sekarang kita akan melakukan migration dari terminal visual studio:

Caranya: Tools -> Nuget Package Manager -> Package Manager Console.

Kemudian ketikkan command berikut:

1. Add-migration **AddBookTodB** => Setelah command ini kita jalankan, itu akan membuat sebuah folder namanya Migrations dan isinya file migration. Note: **AddBookTodB** text bebas untuk catatan kita saja, ini sama seperti commit di svn.

****

File didalam folder migrations itu untuk pertama kali seharusnya cuman 1, itu adalah file setelah saya melakukan beberapa migration ke database (seperti menambah table maupun kolom).

1. Setelah filenya tercreate, sekarang kita save commit ke database:

Update-database

Cara membuat library C# adalah membuat function **Class** yang dimana tidak boleh ada function **Main**. Tidak boleh ada function **Main** karena logikanya itu hanya library dimana hanya akan dijalankan ketika dipanggil bukan dijalankan saat program berjalan, seperti layaknya tujuan method **Main** di dalam applikasi.

Cara memanggil function di satu class yang sama.

Jika functionnya di define static, kita bisa langsung : NamaClass.NamaFunction();

Jika functionnya adalah public, maka kita harus membuat dulu object dari class tersebut.

Declare variable:

Cara 1:

**int** angka = 10;

**string** nama = “Dini”**;**

cara 2:

**var** angka = 10

**var** string = “Dini”

Pad acara 1, disana langsung dibuat type data didepan nama variable. Sedangkan pada acara ke 2, menggunakan kata kunci **var**. dengan cara ini C# compiler otomatis ngecek itu berdasarkan value yg di assign ke variable itu.

Convert type data

Byte -> int

Contoh:

**byte** i = 10;

**int** j = i

Bamun, kita tidak bisa dengan cara itu mengkoversi dari **int** ke **byte**. Itu dikarenakan nilai dari **int** lebih besar dari **byte**. Perhatikan perbandingannya berikut:

1 Byte = 8 bits. Batas nilai byte adalah 0 - 255

1 Int = 4 byte = 32 bits. Batas nilai int adalah – 2 jt s/d + 2 jt

1 Ulong = 8 byte = 64 bits.

Jika kita convert dari nilai yg paling besar ke nilai lebih kecil itu C# tidak mengijinkan, karena kemungkinan nilainya akan hilang.

Contoh:

**int** angka = 1000;

**byte** angka1 = angka

maka itu tidak bisa, karena batas nilay byte adalah sampai 255. Itulah sebabnya C# tidak mengijinkan hal tersebut.

Perbedaan fungsi **MAIN** dan **Consctructor** di C#

**Main** = akan dijalankan ketika program kita jalankan.

**Constructor** = akan dijalankan ketika class tersebut institate (new nama\_class)

Try and Catch

static void TryCacthEx()

{

int[] i = { 1, 2, 3 };

try

{

Console.WriteLine(i[4]);

}

catch(Exception e)

{

Console.WriteLine(e.Message);

}

}

Contoh variable type Ref (Reference)

static void ParameterEx(ref int x, ref int y)

{

int temp = x;

x = y;

y = temp;

}

static void Main()

{

int[] args = { 1, 2, 3, 4, 5 };

int i = 1, j = 2;

ParameterEx(ref i, ref j);

Console.WriteLine("{0} {1}", i, j);

}

Note:

Didalam Program C#, semua class yang kita buat harus di masukkkan ke dalam namespace yang sama agar bisa berjalan normal.

namespace Program

{

class Program

{

static void Main()

{

int[] args = { 1, 2, 3, 4, 5 };

int i = 1, j = 2;

ParameterEx(ref i, ref j);

//Console.WriteLine("{0} {1}", i, j);

Test.StaticEx();

Test test = new Test(1, 2);

int result = test.CountEx();

Console.WriteLine(result);

}

}

}

Class yang lain (mungkin di file lain):

namespace Program

{

class Test

{

public int i;

public int j;

public static void StaticEx()

{

Console.WriteLine("this static method");

}

}

}

Kalau tidak seperti itu, maka program kita tidak akan berjalan normal. Sebagai contoh: kita tidak akan bisa memanggil function StaticEx di dalam class Program dari class lain dengan cara normalnya, yaitu: Program.StaticEx();

Namun ada cara lain yaitu dengan menggunakan using nama\_nama\_namespaces. contoh jika class Test tersebut diganti namespacenya menjadi ProgramTest, maka di file class Program harus kita sertakan namespace nya yaitu dengan cara:

using ProgramTest.

Perbedaan static dengan tanpa static di variable.

Perbedaannya antara lain:

1. Hanya akan selalu ada nilai 1 nilai yang sama ketika class tersebut dipanggil.

Contoh:

using System;

namespace StaticVarApplication {

class StaticVar {

public static int num;

public void count() {

num++;

}

public int getNum() {

return num;

}

}

class StaticTester {

static void Main(string[] args) {

StaticVar s1 = new StaticVar();

StaticVar s2 = new StaticVar();

s1.count();

s1.count();

s2.count();

s2.count();

Console.WriteLine("Variable num for s1: {0}", s1.getNum());

Console.WriteLine("Variable num for s2: {0}", s2.getNum());

Console.ReadKey();

}

}

}

Keterangan:

Output dari program di atas adalah:

Variable num for s1: 4

Variable num for s2: 4

Di class : **StaticVarApplication**, kita mempunyai 1 variable dengan modiersnya adalah **static**. Kemudian variable tersebut kita increament (increament akan berjalan tiap kali class tersebut di instantiate). Kemudian di class **StaticTeste**, kita instantiate class tersebut sebanyak 4 kali, tapi dengan nama object yang berbeda. Kita lihat hasilnya adalah 4 dan 4. Namun jika modifiers static tidak di sertakan di variable **num**, maka nilainya adalah 2 dan 2.

Function dimana jumlah parameternya bisa terserah jumlahnya. Ini namanya adalah **Parameter Array**.

Contoh:

static void ParameterArEx(params int[] args)

{

//IEnumerable enumerable = args as IEnumerable;

foreach (int arg in args)

{

Console.WriteLine(arg);

}

}

Contoh cara memanggilnya:

ParameterArEx(1, 2);

ParameterArEx(1, 2, 3);

Kalau kita perhatikan pada saat pemanggilannya jumlah parameternya bisa berubah-ubah. Satu hal lagi yang perlu kita perhatikan adalah type parameter di parameter di functionnya, jika **int,** maka di foreach nya juga harus **int**, begitu juga dengan **string**, maupun tipe data yang lainnya.

**List**

Contoh

Declare:

List<int> jumlah = new List<int>;

Additem:

Jumlah.Add(1);

Note:

2 atau lebih list item bisa dibandingkan dengan operator **==** dan/ atau **!=** .

List<int> jumlah = new List<int>;

List<int> jumlahSecond = new List<int>;

jumlah.Add(1);

jumlahSecond.Add(1);

Bandikan:

Jumlah == jumlahSecond; Hasilnya adalah True, karena memiliki type(int) dan jumlah yang sama (1).