



INDONESIA
.NET DEVELOPER COMMUNITY



Seri Belajar

ASP.NET

ASP.NET Core 2 MVC & MS SQL Server
dengan Visual Studio 2017

M Reza Faisal, Erick Kurniawan

Kata Pengantar

Puji dan syukur diucapkan kepada Allah SWT atas selesainya buku sederhana yang berjudul Seri Belajar ASP.NET: ASP.NET Core 2 MVC dan MS SQL Server dengan Visual Studio 2017.

Sebelumnya telah dibuat dua buku Seri Belajar ASP.NET yaitu ASP.NET Core MVC & MySQL dengan Visual Studio Code dan ASP.NET Core MVC & PostgreSQL. Kedua buku memiliki kesamaan dalam penggunaan tool Visual Studio Code sebagai tool pengembangan aplikasi web dengan ASP.NET Core versi 1.1. Kelebihan tool ini adalah bersifat multi platform, tetapi tool ini hanya memiliki fitur-fitur dasar pemrograman.

Sedangkan pada buku ini menggunakan tool pengembangan akan digunakan ASP.NET Core versi 2.1 yang merupakan versi terbaru. Terdapat beberapa perubahan dari sisi konfigurasi dan penyesuaian cara penulisan program. Selain ini buku ini menggunakan tool developmet Visual Studio 2017 yang mempunyai fitur-fitur untuk mempermudah dan mempercepat pengembangan software seperti untuk fitur otomatisasi pembuatan class-class yang diperlukan dan lain-lain.

Harapannya buku ini dapat menjadi panduan bagi web developer untuk membangun aplikasi web multiplatform dengan ASP.NET Core MVC.

Akhir kata, selamat membaca dan semoga buku ini bermanfaat bagi para web developer pemula untuk membuat aplikasi web. Kritik dan saran akan sangat berarti dan dapat ditujukan via email.

Banjarmasin, Januari 2017

Erick Kurniawan

(erick.kurniawan@gmail.com)

M Reza Faisal

(reza.faisal@gmail.com)

Daftar Isi

<i>Kata Pengantar</i>	I
<i>Daftar Isi</i>	II
<i>Daftar Gambar</i>	VI
1 Pendahuluan	12
. NET Core.....	12
ASP.NET Core	13
Web Server	15
Development Tool	15
Visual Studio 2017.....	16
Visual Studio Code	17
Database	18
MS SQL Server 2017.....	18
Bahan Pendukung	19
Buku	19
Source Code	19
2 .NET Core 2 SDK & Runtime	20
Installasi	20
Uji Coba	20
.NET Core Command Line Tool	21
Info & Bantuan	21
Membuat Project	22
Restore	25
Build.....	26
Run	26
Migrasi Project.....	28
3 Visual Studio 2017	29
Installasi	29
Antarmuka	30
Solution Explorer	31
Editor	32
Toolbox	33

Properties34
Output35
Error List35
Solution & Project.....	.36
Solution.....	.36
Project38
Item44
Build & Debug.....	.44
Reference45
NuGet47
4 ASP.NET Core50
ASP.NET Core Project.....	.50
Membuat Solution50
Membuat Project ASP.NET Core50
Cara Kerja ASP.NET Core MVC60
Controller61
View62
Model.....	.64
Catatan.....	.67
5 Entity Framework Core & MS SQL Server69
Pendahuluan69
Aplikasi GuestBook.....	.70
Database First.....	.70
Project70
Database72
Model.....	.75
Controller78
Views82
Code First.....	.89
Project89
Model.....	.90
Database93
Controller & View.....	.97
Kesimpulan98

6 ASP.NET Core Identity	100
Pendahuluan	100
Project	100
Membuat Project	101
Konfigurasi Connection String.....	102
Penjelasan Startup.cs	103
Model.....	106
IdentityUser	106
IdentityRole	107
Database.....	108
Migrasi.....	108
Table.....	110
7 Model-View-Controller	112
Persiapan.....	112
Aplikasi Book Store	112
Template Aplikasi Web	112
Membuat Project	113
Catatan.....	115
Model.....	115
API	116
Tipe Class Model.....	117
Display & Format.....	125
Relasi.....	128
Validasi.....	131
Book Store: Class Model & Atribut.....	134
Class Migrations & Database	142
Controller.....	145
View Bag	147
LINQ	147
Book Store: Komponen Controller.....	150
View	205
Akses File	205
Razor	206
Layout & Antarmuka	206

Sintaks Dasar Razor	215
HTML Helper	224
Tag Helper	235
Book Store: Komponen View	244
8 Implementasi Keamanan	292
Modifikasi Startup.cs	292
Otentikasi	293
Model: UserLoginFormViewModel.cs.....	293
View	293
Controller: HomeController	296
Otorisasi	299
9 Penutup.....	305

Daftar Gambar

Gambar 1. Arsitektur ASP.NET.....	12
Gambar 2. Daftar file installer .NET 2.....	13
Gambar 3. Model-View-Controller.....	14
Gambar 4. Web API/HTTP Service.....	14
Gambar 5. Visual Studio Website.....	15
Gambar 6. Perbandingan Fitur Visual Studio 2017.....	16
Gambar 7. Visual Studio 2017 pada platform Windows.....	16
Gambar 8. Visual Studio 2017 pada platform MacOS.....	17
Gambar 9. Visual Studio Code.....	17
Gambar 10. SQL Server 2017 for Windows.....	18
Gambar 11. Seri Belajar ASP.NET: ASP.NET Core MVC & MySQL. Error! Bookmark not defined.	
Gambar 12. Seri Belajar ASP.NET: ASP.NET Core MVC & PostgreSQL dengan Visual Studio Code..... Error! Bookmark not defined.	
Gambar 13. Seri Belajar ASP.NET: Pengenalan ASP.NET Web API. Error! Bookmark not defined.	
Gambar 14. .NET Core 2 – Versi.....	20
Gambar 15. Daftar file dan folder project ASP.NET Core Empty.....	24
Gambar 16. Daftar file dan folder project ASP.NET Core Web.	25
Gambar 17. Daftar file dan folder project ASP.NET Core Web dengan otentikasi.	25
Gambar 18. dotnet restore.....	26
Gambar 19. dotnet build.....	26
Gambar 20. Tampilan aplikasi web ASP.NET Core Empty.....	27
Gambar 21. Aplikasi ASP.NET Core Web.....	27
Gambar 22. Aplikasi ASP.NET Core Web dengan fitur otentikasi.....	28
Gambar 23. Visual Studio 2017 – Antarmuka installasi.....	29
Gambar 24. Visual Studio 2017 – Start Page.	30
Gambar 25. Visual Studio 2017 – Membuka Project.....	31
Gambar 26. Visual Studio 2017 – Solution Explorer.....	32
Gambar 27. Visual Studio 2017 – Code Editor.....	32
Gambar 28. Visual Studio 2017 – Visual Editor.....	33
Gambar 29. Visual Studio 2017 – Toolbox.....	34
Gambar 30. Visual Studio 2017 – Properties.....	35

Gambar 31. Visual Studio 2017 – Output	35
Gambar 32. Visual Studio 2017 – Error List	35
Gambar 33. Visual Studio 2017 – Blank solution.....	36
Gambar 34. Visual Studio 2017 – Blank Solution	37
Gambar 35. Visual Studio 2017 – Project.....	37
Gambar 36. Visual Studio 2017 – Solution & Project	38
Gambar 37. Visual Studio 2017 – Template project	38
Gambar 38. Visual Studio 2017 – Tipe aplikasi yang dapat dikembangkan dengan bahasa Visual C#.....	39
Gambar 39. Visual Studio 2017 – Daftar template project Web dengan C#.	40
Gambar 40. Visual Studio 2017 – Membuat Blank Solution.	40
Gambar 41. Visual Studio 2017 – Menambah project ke dalam solution.....	41
Gambar 42. Visual Studio 2017 – Menambahkan project Console App.	41
Gambar 43. Visual Studio 2017 – ConsoleApp1.....	42
Gambar 44. Visual Studio 2017 – Menambahkan project web.	42
Gambar 45. Visual Studio 2017 – Template project web ASP.NET 4.6.1.	43
Gambar 46. Visual Studio 2017 – Menambahkan project web application.	43
Gambar 47. Visual Studio 2017 - Add New Item.	44
Gambar 48. Visual Studio 2017 – Output proses build.	45
Gambar 49. Visual Studio 2017 – Output proses debug.....	45
Gambar 50. Visual Studio 2017 - References.....	46
Gambar 51. Visual Studio 2017 - Reference Manager - Assemblies.	46
Gambar 52. Visual Studio 2017 - Refence - Projects.....	47
Gambar 53. Visual Studio 2017 - ConsoleAppNETCore.	48
Gambar 54. Visual Studio 2017 - NuGet Package Manager - Installed.	48
Gambar 55. Visual Studio 2017 - NuGet Package Manager - Browse.	48
Gambar 56. Visual Studio 2017 - NuGet Package Manager - Search & Install.....	49
Gambar 57. Visual Studio 2017 - NuGet pada Solution Explorer.	49
Gambar 58. Solution - ASPNETCoreSQLServer.....	50
Gambar 59. Daftar template project ASP.NET Core.....	51
Gambar 60. Template Project: Empty - Add New Project.	52
Gambar 61. Template Project: Empty - Add New Project.	52
Gambar 62. Template Project: Empty - New ASP.NET Core Web Application.	53
Gambar 63. Template Project: Empty - WebAppEmpty pada Solution Explorer.....	53
Gambar 64. Template Project: Empty - Package Manager Console.....	55

Gambar 65. Template Project: Empty - Diagnostic Tools.....	56
Gambar 66. Template Project: Empty - WebAppEmpty pada web browser.....	56
Gambar 67. Template Project: Web Application - Add New Project.	57
Gambar 68. Template Project: Web Application - New ASP.NET Core Web Application.	57
Gambar 69. Template Project: Web Application - WebAppEmpty pada Solution Explorer. ...	58
Gambar 70. Template Project: Web Application (MVC) - Add New Project.	59
Gambar 71. Template Project: Web Application (MVC) - New ASP.NET Core Web Application.	59

Gambar 72. Template Project: Web Application (MVC) - WebAppEmpty pada Solution Explorer.....	60
Gambar 73. Cara kerja Pattern MVC.....	61
Gambar 74. Controller - Menambah class controller.....	61
Gambar 75. Controller - HelloWorldController.....	61
Gambar 76. View - Add MVC View.	62
Gambar 77. WebAppMVC - Home.	63
Gambar 78. WebAppMVC - HelloWorld.	64
Gambar 79. Antarmuka form pada halaman HelloWorld.....	64
Gambar 80. Model - Add New Item.	65
Gambar 81. Hasil ketika form diisi dan tombol Send diklik.	67
Gambar 82. Catatan 1 tentang ASP.NET Core MVC.	67
Gambar 83. Catatan 2 tentang ASP.NET Core MVC.	68
Gambar 84. SqlServerDbFirst - Add New Project.....	70
Gambar 85. SqlServerDbFirst - New ASP.NET Core Web Application.....	71
Gambar 86. SqlServerDbFirst - Menambah library.....	71
Gambar 87. SqlServerDbFirst - Daftar library yang telah diinstall.....	72
Gambar 88. SqlServerDbFirst - Membuat database SqlServerDbFirst.	72
Gambar 89. SqlServerDbFirst - Membuat tabel Guestbook.....	73
Gambar 90. SqlServerDbFirst - Tombol save table.	74
Gambar 91. SqlServerDbFirst - Memberi nama table.	74
Gambar 92. SqlServerDbFirst – Scaffold-DbContext.	75
Gambar 93. SqlServerDbFirst – Add Scaffold.....	79
Gambar 94. SqlServerDbFirst – Add MVC Controller with views, using Entity Framework. .	79
Gambar 95. SqlServerDbFirst – Index.cshtml.....	84
Gambar 96. SqlServerDbFirst – Detail.cshtml.	85
Gambar 97. SqlServerDbFirst – Create.cshtml.....	86
Gambar 98. SqlServerDbFirst – Edit.cshtml.....	88

Gambar 99. SqlServerDbFirst – Delete.cshtml.....	89
Gambar 100. SqlServerCodeFirst – Daftar library Entity Framework.....	90
Gambar 101. SqlServerCodeFirst - Migration class.....	93
Gambar 102. SqlServerCodeFirst - Migrations	94
Gambar 103. SqlServerCodeFirst - Add Connection.	96
Gambar 104. SqlServerCodeFirst - Server Explorer.....	97
Gambar 105. SqlServerCodeFirst - Add MVC Controller with views, using Entity Framework.	
.....	97
Gambar 106. SqlServerCodeFirst - Controller & View.....	98
Gambar 107. SqlServerIdentity - Membuat project.....	101
Gambar 108. SqlServerIdentity - New ASP.NET Core Web Application.	101
Gambar 109. SqlServerIdentity - Change Authentication.....	102
Gambar 110. SqlServerIdentity - Folder Migrations.	102
Gambar 111. SqlServerIdentity - Antarmuka aplikasi web.	108
Gambar 112. SqlServerIdentity - Form registrasi.	109
Gambar 113. SqlServerIdentity - Migrations.	109
Gambar 114. SqlServerIdentity - Login berhasil.	110
Gambar 115. SqlServerIdentity - Database & table.....	110
Gambar 116. Template admin Gentellela.	113
Gambar 117. Book Store - Daftar buku.	123
Gambar 118. Display atribut model sebagai label pada header tabel.	127
Gambar 119. Display atribut model sebagai label pada form input.....	127
Gambar 120. Relasi pada tabel.	128
Gambar 121. SqlServerBookStore - Migrations class.	144
Gambar 122. SqlServerBookStore - Database SqlServerBookStore dan table-table.....	145
Gambar 123. CategoriesController - Window Add Scaffold.	151
Gambar 124. CategoriesController - Window Add MVC Controller with views, using Entity Framework.	152
Gambar 125. AuthorsController - Window Add Scaffold.	160
Gambar 126. AuthorsController - Window Add MVC Controller with views, using Entity Framework.	161
Gambar 127. BooksController - Window Add MVC Controller with views, using Entity Framework.	169
Gambar 128. BooksController - Daftar buku.	178
Gambar 129. BooksController - Detail.	180
Gambar 130. BooksController - Form menambah buku.	181

Gambar 131. BooksController - Create	182
Gambar 132. BooksController - Edit	185
Gambar 133. BooksController - Delete	190
Gambar 134. File dan folder template gentelella.....	206
Gambar 135. SqlServerBookStore - View - wwwroot.....	207
Gambar 136. Layout aplikasi.	207
Gambar 137. Master layout template gentelella.....	208
Gambar 138. Razor - Pesan kesalahan.	216
Gambar 139. Razor - Penggunaan simbol @@.....	217
Gambar 140. Razor - Ekspresi implisit & eksplisit.....	219
Gambar 141. Razor - Blok kode.	219
Gambar 142. Razor - Blok kode.	220
Gambar 143. Razor - Pengulangan.....	222
Gambar 144. Daftar data tamu.....	223
Gambar 145. Tampilan form sebelum proses validasi.	232
Gambar 146. Tampilan form setelah proses validasi.....	232
Gambar 147. Antarmuka design form.....	233
Gambar 148. Contoh antarmuka implementasi tag helper input.	240
Gambar 149. Contoh antarmuka implementasi tag helper select.....	243
Gambar 150. Contoh antarmuka implementasi tag helper select dengan atribut multiple.	243
Gambar 151. View Categories - Index.cshtml.....	246
Gambar 152. View Categories - Details.cshtml.....	248
Gambar 153. View Categories - Create.cshtml	249
Gambar 154. View Categories - Edit.....	250
Gambar 155. View Categories - Delete.	252
Gambar 156. View Authors - Index.cshtml.....	253
Gambar 157. View Authors - Details.cshtml.....	255
Gambar 158. View Authors - Create.cshtml	256
Gambar 159. View Authors - Edit.	258
Gambar 160. View Authors - Delete.	259
Gambar 161. View Books - Index.cshtml.....	261
Gambar 162. View Books - Details.cshtml.	263
Gambar 163. View Books - Create.cshtml	265
Gambar 164. View Books - Edit.	268
Gambar 165. View Books - Delete.	270

Gambar 166. View Role - Index.cshtml.....	273
Gambar 167. View Role - Detail.cshtml.....	275
Gambar 168. View Role - Create.cshtml	276
Gambar 169. View Role - Edit.....	278
Gambar 170. View Role - Delete.....	280
Gambar 171. View User - Index.cshtml.	282
Gambar 172. View User - Detail.cshtml.....	284
Gambar 173. View User - Create.cshtml.....	286
Gambar 174. View User - Edit.....	288
Gambar 175. View User - Delete.....	290
Gambar 176. Implementasi Keamanan - Form Login.....	296
Gambar 177. Implementasi Keamanan - AccessDenied.cshtml.	296
Gambar 178. Implementasi Keamanan - Role.	299

1

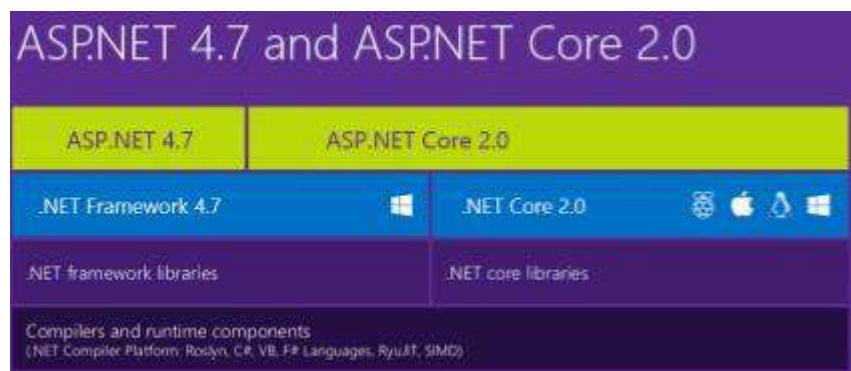
Pendahuluan

.NET Core

.NET Framework adalah software framework yang dikembangkan oleh Microsoft. .NET Framework terdiri atas banyak class library (Framework Class Library) untuk membangun bermacam aplikasi, seperti aplikasi desktop, aplikasi mobile, aplikasi web dan cloud. Sampai saat ini .NET Framework telah mencapai versi 4.7.1. .NET Framework ini hanya dapat digunakan pada platform atau sistem operasi MS Windows. Aplikasi-aplikasi yang dibangun di atas .NET Framework hanya dapat dijalankan jika pada komputer telah terinstall .NET Framework. Artinya aplikasi-aplikasi itu hanya akan jalan pada platform MS Windows.

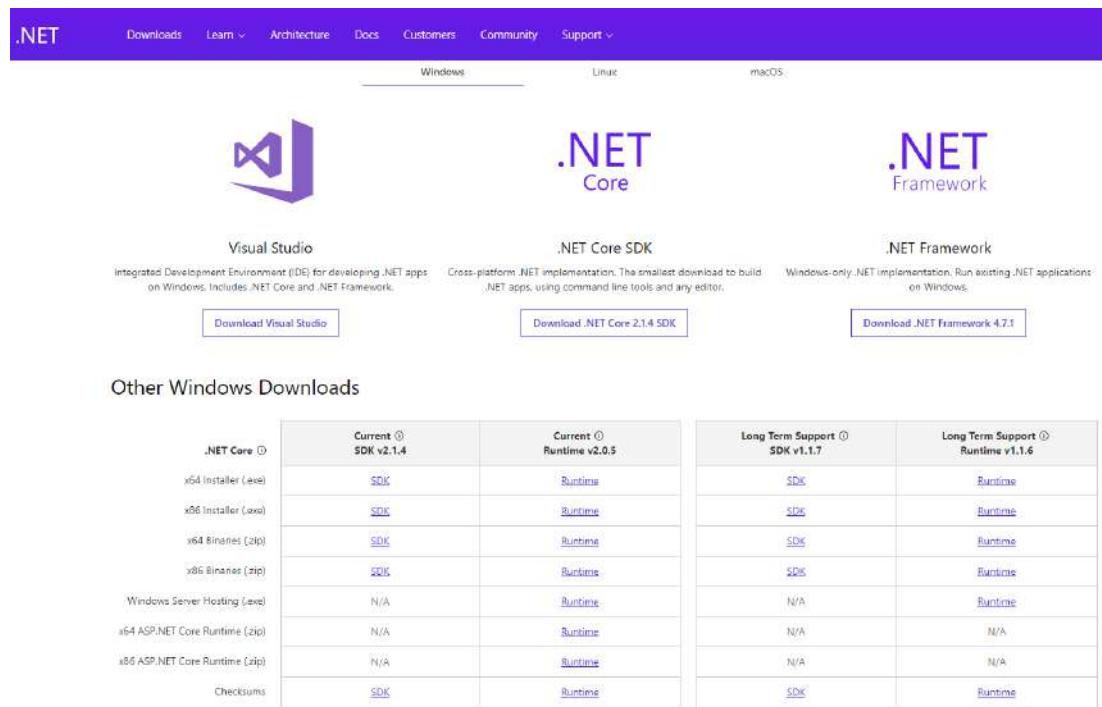
Saat ini Microsoft telah mengembangkan .NET Core, yaitu “.NET Framework” yang bersifat open-source dan multiplatform. Artinya .NET Core dapat dijalankan pada platform Windows, Linux dan Mac OS. Sehingga aplikasi-aplikasi yang dibangun di atas framework ini juga dapat dijalankan di atas banyak platform. Saat ini .NET Core hanya mendukung bahasa pemrograman C# dan F#. Saat buku ini ditulis .NET Core telah mencapai versi 2.1.4.

Gambar di bawah ini adalah arsitektur sederhana dari kedua framework, yaitu .NET Framework dan .NET Core.



Gambar 1. Arsitektur ASP.NET.

Untuk mendapatkan .NET Core dapat diunduh pada link berikut <https://www.microsoft.com/net/download>.



Gambar 2. Daftar file installer .NET 2.

ASP.NET Core

ASP.NET Core merupakan design ulang dari ASP.NET yang telah ada sejak 15 tahun yang lalu. ASP.NET Core adalah framework untuk membangun aplikasi web, IoT app dan backend untuk mobile app. Framework ini bersifat open source dan cross-platform, artinya aplikasi yang dibangun dengan framework ini dapat dijalankan pada sistem operasi Windows, Linux dan Mac OSX. Aplikasi ASP.NET Core dapat dijalankan di atas .NET Core atau .NET framework seperti yang terlihat pada gambar di atas.

Dibandingkan dengan ASP.NET versi sebelumnya, ASP.NET Core mempunyai beberapa perubahan arsitektur. Perubahan ini membuat ASP.NET Core framework menjadi lebih ramping dan modular. Perbedaan lain adalah ASP.NET Core tidak lagi berbasis pada System.Web.dll. ASP.NET Core berbasis kepada package-package di NuGet repository. Hal ini memungkinkan developer untuk melakukan optimasi aplikasi dengan menggunakan package-package NuGet yang diperlukan. Keuntungan hal ini adalah:

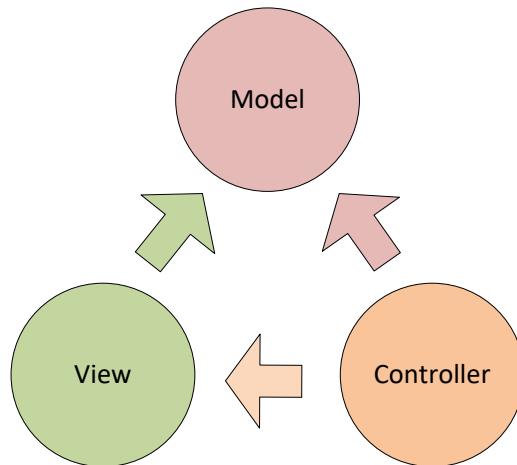
- Aplikasi lebih kecil.
- Aplikasi menjadi lebih aman.
- Mengurangi service.
- Meningkatkan kinerja atau kecepatan.

Tetapi karena ASP.NET Core merupakan framework yang baru saja ditulis, bukan melanjutkan kode sumber framework sebelumnya, maka tidak semua fitur yang telah ditemui pada ASP.NET 4.7 akan ditemui pada framework ini.

Saat ini ASP.NET Core framework dapat digunakan untuk membangun aplikasi web dengan ASP.NET MVC. Selain itu juga dapat digunakan untuk membangun HTTP service dengan menggunakan ASP.NET Web API.

ASP.NET Core MVC

Model-View-Controller (MVC) adalah architectural pattern yang memisahkan aplikasi menjadi tiga komponen utama yaitu Model, View, dan Controller. Dengan menggunakan pattern atau pola ini, request atau permintaan user diarahkan ke komponen Controller, komponen ini bertanggung jawab untuk berkerja dengan komponen Model untuk melakukan aksi dari user dan/atau mengambil hasil dari query. Komponen Controller memilih komponen View untuk ditampilkan kepada user. Komponen View juga menyediakan data Model yang dibutuhkannya.



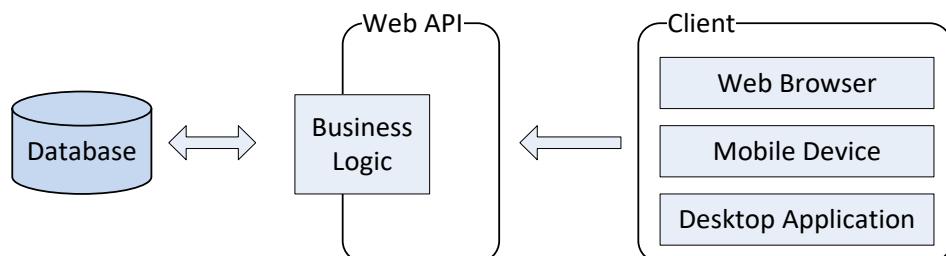
Gambar 3. Model-View-Controller.

ASP.NET Core MVC adalah framework presentasi yang ringan, bersifat open source yang sangat mudah diuji yang dioptimalkan untuk digunakan dengan ASP.NET Core. Framework ini menyediakan cara untuk membangun situs web dinamis yang memungkinkan pemisahan sesuai dengan pola MVC yang telah diterangkan di atas. Framework ini mendukung pengembangan Test-Driven Development (TDD) dan standar web terbaru.

ASP.NET Core Web API

HTTP tidak hanya digunakan untuk menampilkan halaman web saja. HTTP juga mendukung pembangunan API untuk mengekspos layanan dan data. Karena HTTP bersifat sederhana, fleksibel dan dapat diakses dimana saja, maka layanan HTTP dapat menjangkau berbagai client termasuk web browser, perangkat mobile dan aplikasi desktop.

ASP.NET Core Web API adalah framework yang mempermudah web developer untuk membangun layanan HTTP.



Gambar 4. Web API/HTTP Service.

Web Server

Seperti aplikasi web pada umumnya, aplikasi yang dibangun dengan menggunakan framework ASP.NET Core juga memerlukan web server untuk agar bisa diakses dari web browser sebagai web client.

Pada framework ASP.NET sebelumnya digunakan Internet Information Services (IIS) sebagai web server. Tetapi karena IIS hanya dapat berjalan pada platform Windos maka selain IIS juga telah disediakan Kestrel sebagai open-source HTTP server dan cross-platform untuk ASP.NET Core.

Kestrel

Berbeda dengan Apache dan IIS yang didesain untuk banyak kebutuhan umum web server dan juga dapat mendukung banyak bahasa pemrograman dan banyak fitur seperti browsing direktori dan lain-lain. Sedangkan Kestrel didesain hanya untuk hosting ASP.NET Core.

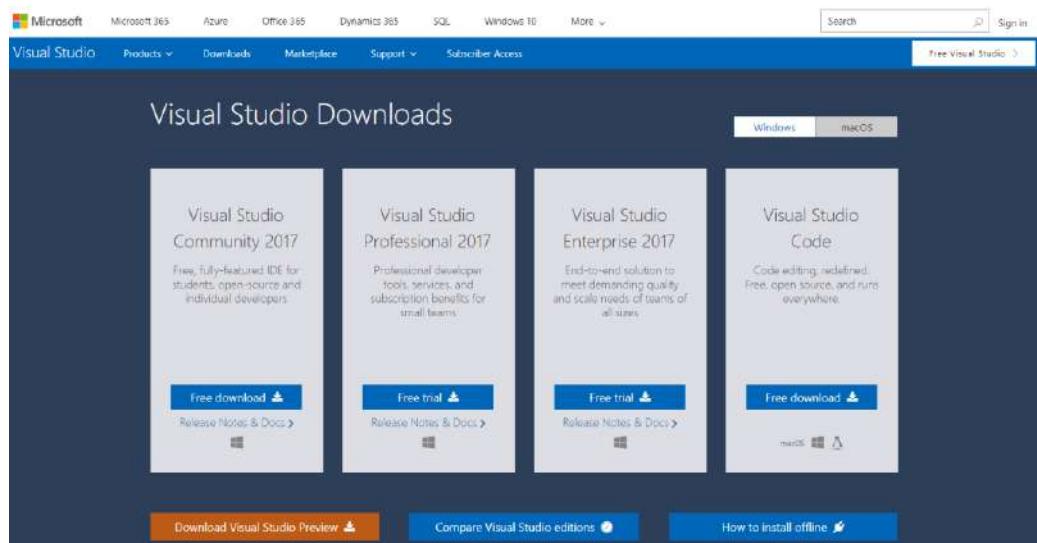
Internet Information Services (IIS)

Telah disebutkan kelebihan IIS yang mendukung kebutuhan umum web server juga dapat digunakan sebagai host ASP.NET Core. Sebagai host ASP.NET Core, IIS tidak berdiri sendiri tetapi perlu dukungan dari Kestrel.

Development Tool

Visual Studio adalah sebuah integrated development environment (IDE) yang dikembangkan oleh Microsoft, yang dapat digunakan untuk mengembangkan aplikasi Android, iOS, Windows, web dan cloud. Tool ini tersedia untuk berbagai platform yaitu Windows, Linux dan MacOS.

Visual Studio dapat diunduh di link berikut ini <https://www.visualstudio.com/downloads/>.



Gambar 5. Visual Studio Website.

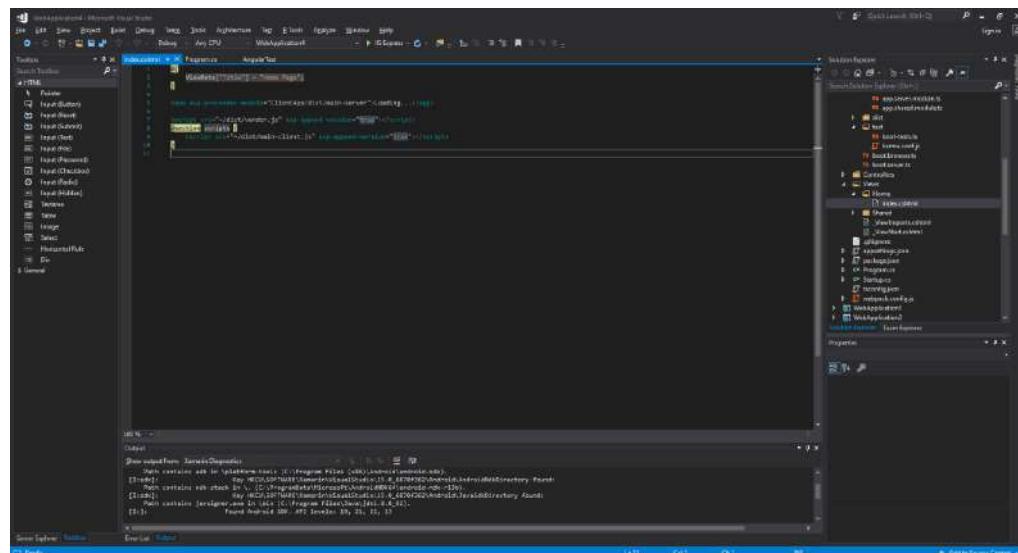
Visual Studio 2017

Visual Studio 2017 tersedia dalam tiga pilihan lisensi yaitu Enterprise dan Professional yang berbayar. Sedangkan Community bersifat gratis. Berikut adalah perbandingan fitur-fitur yang dimiliki oleh masing-masing versi. Visual Studio 2017 hanya tersedia untuk platform Windows dan MacOS.

Supported Features	Visual Studio Community	Visual Studio Professional	Visual Studio Enterprise
① Supported Usage Scenarios	● ● ○	● ● ●	● ● ●
② Development Platform Support	● ● ●	● ● ●	● ● ●
③ Integrated Development Environment	● ● ○	● ● ○	● ● ●
④ Advanced Debugging and Diagnostics	● ● ○○	● ● ○○	● ● ●
⑤ Testing Tools	● ○ ○○	● ○ ○○	● ● ●
⑥ Cross-platform Development	● ● ○○	● ● ○○	● ● ●
⑦ Collaboration Tools and Features	● ● ●	● ● ●	● ● ●

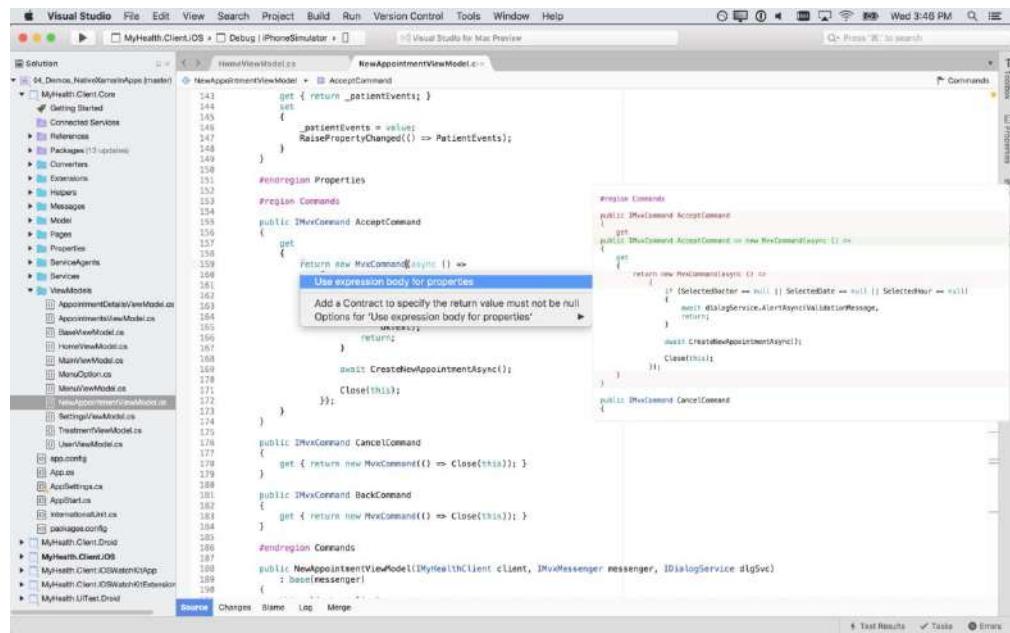
Gambar 6. Perbandingan Fitur Visual Studio 2017.

Berikut ini adalah tampilan Visual Studio 2017 pada platform Windows.



Gambar 7. Visual Studio 2017 pada platform Windows.

Berikut ini adalah tampilan Visual Studio pada platform MacOS.



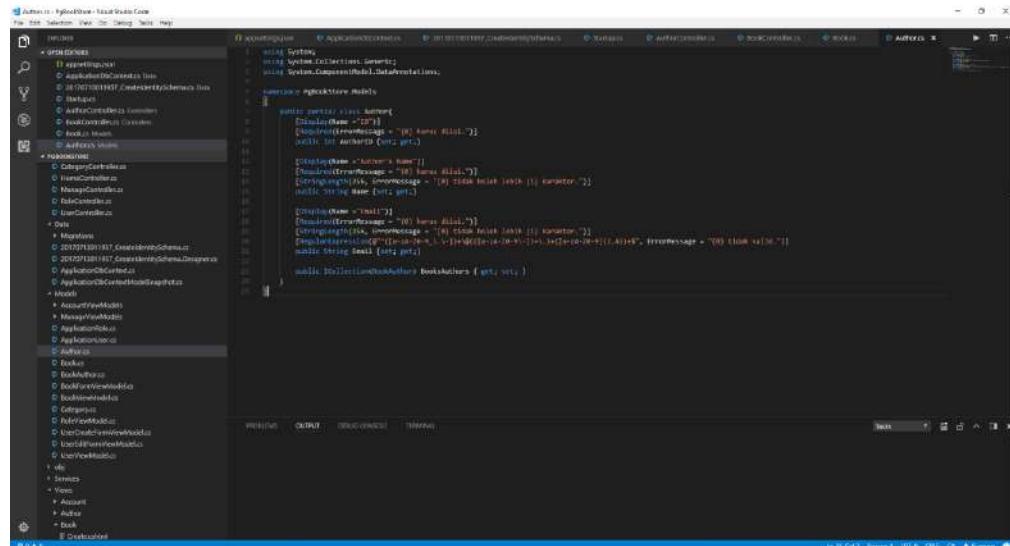
Gambar 8. Visual Studio 2017 pada platform MacOS.

Visual Studio Code

Visual Studio Code adalah versi Visual Studio yang ringan tetapi tetap powerful. Versi ini seperti code editor dengan fitur-fitur tambahan untuk mempermudah penulisan kode program.

Visual Studio Code tersedia pada platform Windows, Linux dan MacOS. Visual Studio Code juga mendukung banyak bahasa pemrograman seperti halnya Visual Studio 2015 ditambah bahasa pemrograman PHP, Node.js dan lain-lain.

Berikut adalah tampilan Visual Studio Code.



Gambar 9. Visual Studio Code.

Database

Database adalah koleksi data yang terorganisir. Database yang sering digunakan adalah relational database yang merupakan koleksi dari skema, tabel, query dan view. Software yang memberikan layanan untuk mengelola database dikenal dengan nama Relational Database Management System (RDBMS) atau Database Server. Saat ini telah banyak tersedia database server, beberapa diantaranya adalah MS SQL Server, MySQL, dan PostgreSQL.

MS SQL Server 2017

MS SQL Server adalah RDBMS yang dikembangkan oleh Microsoft. Versi terbaru dari RDBMS yaitu MS SQL Server 2017 telah tersedia tidak hanya pada platform Windows tetapi juga tersedia platform Linux. Jika ingin memanfaatkan RDBMS ini pada platform MacOS maka dapat memanfaatkan Docker.

MS SQL Server tersedia dalam beberapa edisi yaitu Enterprise, Standard, Developer dan Express Edition. Express Edition dapat digunakan secara gratis dengan beberapa keterbatasan seperti tidak dapat menggunakan kemampuan multi-core dan maksimal ukuran file database yaitu 10GB.

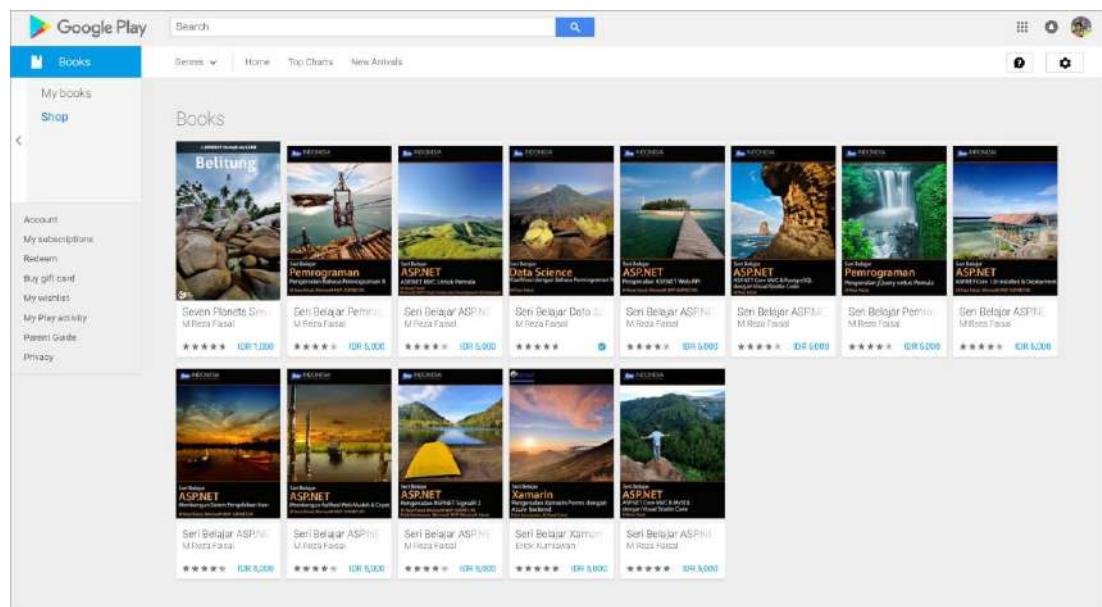
Untuk mencoba MS SQL Server 2017 maka file installer dapat diunduh di sini <https://www.microsoft.com/en-us/sql-server/sql-server-downloads>.



Gambar 10. SQL Server 2017 for Windows.

Bahan Pendukung

Buku



Buku-buku ini dapat diunduh dengan menggunakan link berikut ini:
<https://play.google.com/store/books/author?id=M%20Reza%20Faisal>

Source Code

Source code contoh-contoh yang dibuat pada buku ini dapat diunduh pada repository pada link berikut ini:

1. <https://github.com/rezafaisal/ASPNETCoreSQLServer>.

2

.NET Core 2 SDK & Runtime

Pada bab ini akan dijelaskan langkah-langkah untuk melakukan installasi .NET Core 2 dan ASP.NET Core 2 pada sistem operasi Windows.

Installasi

Saat buku ini ditulis telah tersedia:

- .NET Core SDK v2.1.4.
- .NET Core Runtime v2.0.5.
- ASP.NET Core Runtime v2.0.5

Installer tersebut dapat diunduh pada link berikut
<https://www.microsoft.com/net/download/>.

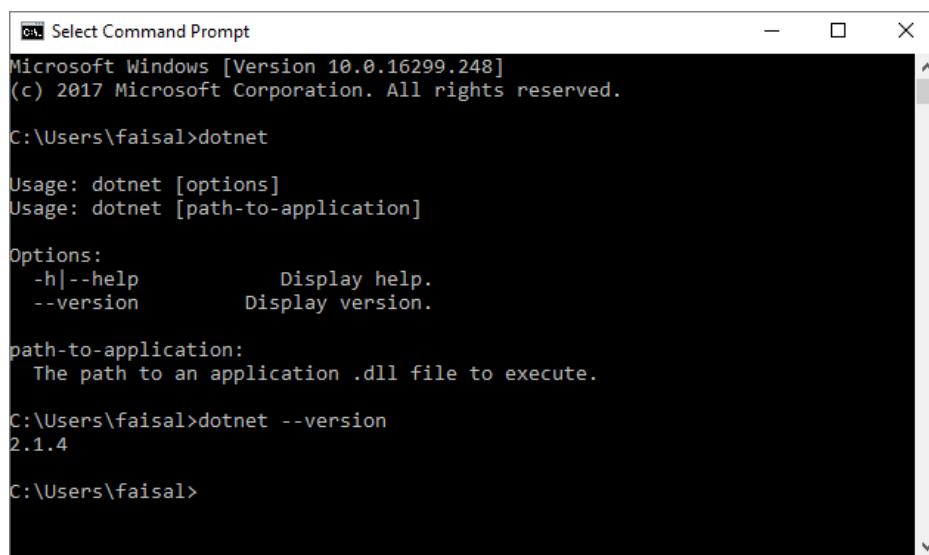
Proses installasi pada platform Windows dan MacOS sangat mudah, hanya dengan cara mengeksekusi file installer kemudian ikuti petunjuk yang diberikan pada window installer.

Uji Coba

Untuk uji coba dapat dilakukan dengan mengetikan perintah berikut ini.

```
dotnet --version
```

Output dari perintah ini adalah sebagai berikut.



```
Microsoft Windows [Version 10.0.16299.248]
(c) 2017 Microsoft Corporation. All rights reserved.

C:\Users\faisal>dotnet
Usage: dotnet [options]
Usage: dotnet [path-to-application]

Options:
  -h|--help           Display help.
  --version          Display version.

path-to-application:
  The path to an application .dll file to execute.

C:\Users\faisal>dotnet --version
2.1.4

C:\Users\faisal>
```

Gambar 11. .NET Core 2 – Versi.

Dari output dapat dilihat versi 2.1.4.

.NET Core Command Line Tool

Pada sub bab sebelumnya telah dapat dilihat penggunaan perintah “dotnet”. Perintah ini adalah perintah utama yang digunakan untuk melakukan hal-hal penting seperti:

1. Membuat project baru.
2. Melakukan migrasi atau upgrade project ke versi yang lebih baru.
3. Melakukan restore library atau package dan tool yang digunakan project.
4. Melakukan build.
5. Melakukan testing.
6. Menjalankan dan mempublish aplikasi.
7. Dan lain-lain.

Pada sub bab ini akan dijelaskan beberapa fungsi penting perintah “dotnet”.

Info & Bantuan

Untuk mengetahui informasi versi .NET Core Command Line Tool yang digunakan dan informasi lainnya dapat digunakan perintah berikut.

```
dotnet --info
```

Hasilnya adalah sebagai berikut.

```
.NET Command Line Tools (2.1.4)

Product Information:
Version:          2.1.4
Commit SHA-1 hash: 5e8add2190

Runtime Environment:
OS Name:        Windows
OS Version:     10.0.16299
OS Platform:    Windows
RID:            win10-x64
Base Path:      C:\Program Files\dotnet\sdk\2.1.4\

Microsoft .NET Core Shared Framework Host

Version : 2.0.5
Build   : 17373eb129b3b05aa18ece963f8795d65ef8ea54
```

Sedangkan untuk mengetahui secara lengkap opsi lengkap yang dapat digunakan pada perintah “dotnet” dapat digunakan perintah berikut.

```
dotnet --help
```

Maka akan dapat dilihat informasi sebagai berikut.

```
.NET Command Line Tools (2.1.4)
Usage: dotnet [runtime-options] [path-to-application]
Usage: dotnet [sdk-options] [command] [arguments] [command-options]

path-to-application:
The path to an application .dll file to execute.

SDK commands:
```

```

new           Initialize .NET projects.
restore       Restore dependencies specified in the .NET project.
run           Compiles and immediately executes a .NET project.
build         Builds a .NET project.
publish       Publishes a .NET project for deployment (including the runtime).
test          Runs unit tests using the test runner specified in the project.
pack          Creates a NuGet package.
migrate       Migrates a project.json based project to a msbuild based project.
clean         Clean build output(s).
sln           Modify solution (SLN) files.
add           Add reference to the project.
remove        Remove reference from the project.
list          List reference in the project.
nuget         Provides additional NuGet commands.
msbuild       Runs Microsoft Build Engine (MSBuild).
vstest        Runs Microsoft Test Execution Command Line Tool.

Common options:
  -v|--verbosity      Set the verbosity level of the command. Allowed values are
q[uiet], m[inimal], n[ormal], d[etailed], and diag[nostic].
  -h|--help           Show help.

Run 'dotnet COMMAND --help' for more information on a command.

sdk-options:
  --version          Display .NET Core SDK version.
  --info             Display .NET Core information.
  -d|--diagnostics  Enable diagnostic output.

runtime-options:
  --additionalprobingpath <path>    Path containing probing policy and assemblies to probe
for.
  --fx-version <version>            Version of the installed Shared Framework to use to
run the application.
  --roll-forward-on-no-candidate-fx Roll forward on no candidate shared framework is
enabled.
  --additional-deps <path>          Path to additonal deps.json file.

```

Membuat Project

Untuk membuat project dengan perintah “dotnet” digunakan opsi “new”. Untuk mengetahui informasi bantuan cara pembuatan project dapat digunakan perintah berikut.

```
dotnet new --help
```

Dan berikut adalah informasi dari perintah di atas.

```

Usage: new [options]

Options:
  -h, --help           Displays help for this command.
  -l, --list            Lists templates containing the specified name. If no name is
specified, lists all templates.
  -n, --name            The name for the output being created. If no name is specified, the
name of the current directory is used.
  -o, --output           Location to place the generated output.
  -i, --install          Installs a source or a template pack.
  -u, --uninstall        Uninstalls a source or a template pack.
  --type                Filters templates based on available types. Predefined values are
"project", "item" or "other".
  --force               Forces content to be generated even if it would change existing
files.
  -lang, --language      Specifies the language of the template to create.

Usage: new [options]

Options:
  -h, --help           Displays help for this command.

```

```

-l, --list      Lists templates containing the specified name. If no name is
specified, lists all templates.
-n, --name      The name for the output being created. If no name is specified, the
name of the current directory is used.
-o, --output    Location to place the generated output.
-i, --install   Installs a source or a template pack.
-u, --uninstall Uninstalls a source or a template pack.
--type        Filters templates based on available types. Predefined values are
"project", "item" or "other".
--force       Forces content to be generated even if it would change existing
files.
--lang, --language Specifies the language of the template to create.

Templates          Short Name     Language     Tags
-----
Console Application      console      [C#], F#, VB
Common/Console
Class library            classlib     [C#], F#, VB
Common/Library
Unit Test Project        mstest      [C#], F#, VB
Test/MSTest
xUnit Test Project       xunit       [C#], F#, VB
Test/xUnit
ASP.NET Core Empty       web         [C#], F#
Web/Empty
ASP.NET Core Web App (Model-View-Controller) mvc         [C#], F#
Web/MVC
ASP.NET Core Web App     razor       [C#]
Web/MVC/Razor Pages
ASP.NET Core with Angular angular     [C#]
Web/MVC/SPA
ASP.NET Core with React.js react      [C#]
Web/MVC/SPA
ASP.NET Core with React.js and Redux reactredux [C#]
Web/MVC/SPA
ASP.NET Core Web API     webapi      [C#], F#
Web/WebAPI
global.json file
Config
NuGet Config             nugetconfig
Config
Web Config               webconfig
Config
Solution File            sln
Solution
Razor Page               page
Web/ASP.NET
MVC ViewImports           viewimports
Web/ASP.NET
MVC ViewStart              viewstart
Web/ASP.NET

Examples:
dotnet new mvc --auth Individual
dotnet new console
dotnet new --help

```

Dari informasi di atas dapat dilihat daftar template project yang dapat dibuat. Dari daftar tersebut terdapat 7 project dan 1 solution. Ada dua pilihan bahasa pemrograman yang dapat dipergunakan yaitu C# dan F#. Hal yang paling penting dari daftar template di atas adalah bagian “Short Name”, karena nilai pada kolom ini yang dipergunakan untuk menentukan tipe project yang akan dibuat nanti.

Untuk melihat seluruh daftar template dapat dilihat perintah berikut ini.

```
dotnet new -all
```

Untuk membuat project digunakan perintah dengan sintaks sebagai berikut.

```
dotnet new [short name] -lang [language] -o [folder name]
```

Sebagai contoh untuk membuat aplikasi console dengan bahasa pemrograman F# digunakan perintah berikut ini.

```
dotnet new console -lang F# -o ConsoleF
```

Jika opsi –lang tidak digunakan, maka akan digunakan nilai default dari opsi ini yaitu bahasa pemrograman C#.

Untuk membuat project aplikasi web tersedia 2 template yang dapat digunakan yaitu:

- ASP.NET Core Empty, template ini digunakan untuk membuat aplikasi web kosong.
- ASP.NET Core Web App, template ini digunakan untuk membuat aplikasi web lengkap.

Untuk membuat aplikasi web dengan menggunakan template ASP.NET Core Empty digunakan perintah perintah berikut ini.

```
dotnet new web -o webempty
```

Hasilnya dapat dilihat di dalam folder webempty dengan isi sebagai berikut.

Name	Date modified	Type	Size
obj	2/24/2018 2:48 PM	File folder	
wwwroot	2/24/2018 2:47 PM	File folder	
Program.cs	2/24/2018 2:47 PM	Visual C# Source f...	1 KB
Startup.cs	2/24/2018 2:47 PM	Visual C# Source f...	2 KB
webempty.csproj	2/24/2018 2:47 PM	Visual C# Project f...	1 KB

Gambar 12. Daftar file dan folder project ASP.NET Core Empty.

Sedangkan untuk membuat project ASP.NET Web App dapat dilakukan dengan dua cara. Cara pertama dengan perintah berikut ini.

```
dotnet new mvc -o webmvc
```

Hasilnya dapat dilihat di dalam folder webmvc dengan isi sebagai berikut.

Name	Date modified	Type	Size
Controllers	2/24/2018 2:50 PM	File folder	
Models	2/24/2018 2:50 PM	File folder	
obj	2/24/2018 2:50 PM	File folder	
Views	2/24/2018 2:50 PM	File folder	
wwwroot	2/24/2018 2:50 PM	File folder	
appsettings.Development.json	2/24/2018 2:50 PM	JSON File	1 KB
appsettings.json	2/24/2018 2:50 PM	JSON File	1 KB
bundleconfig.json	2/24/2018 2:50 PM	JSON File	1 KB
Program.cs	2/24/2018 2:50 PM	Visual C# Source f...	1 KB
Startup.cs	2/24/2018 2:50 PM	Visual C# Source f...	2 KB
webmvc.csproj	2/24/2018 2:50 PM	Visual C# Project f...	1 KB

Gambar 13. Daftar file dan folder project ASP.NET Core Web.

Project di atas menghasilkan aplikasi web ASP.NET Core MVC yang memiliki komponen Controller dan View, tetapi tanpa komponen Model. Aplikasi ini juga belum menggunakan database untuk mengelola user dan fitur otentifikasi.

Untuk membuat aplikasi web dengan fitur otentifikasi dapat digunakan perintah berikut ini.

```
dotnet new mvc --auth Individual -o webmvcauth
```

Hasilnya dapat dilihat pada gambar di bawah ini.

Name	Date modified	Type	Size
Controllers	2/24/2018 2:51 PM	File folder	
Data	2/24/2018 2:51 PM	File folder	
Extensions	2/24/2018 2:51 PM	File folder	
Models	2/24/2018 2:51 PM	File folder	
obj	2/24/2018 2:51 PM	File folder	
Services	2/24/2018 2:51 PM	File folder	
Views	2/24/2018 2:51 PM	File folder	
wwwroot	2/24/2018 2:51 PM	File folder	
app.db	2/24/2018 2:51 PM	Data Base File	104 KB
appsettings.Development.json	2/24/2018 2:51 PM	JSON File	1 KB
appsettings.json	2/24/2018 2:51 PM	JSON File	1 KB
bundleconfig.json	2/24/2018 2:51 PM	JSON File	1 KB
Program.cs	2/24/2018 2:51 PM	Visual C# Source f...	1 KB
Startup.cs	2/24/2018 2:51 PM	Visual C# Source f...	3 KB
webmvcauth.csproj	2/24/2018 2:51 PM	Visual C# Project f...	1 KB

Gambar 14. Daftar file dan folder project ASP.NET Core Web dengan otentifikasi.

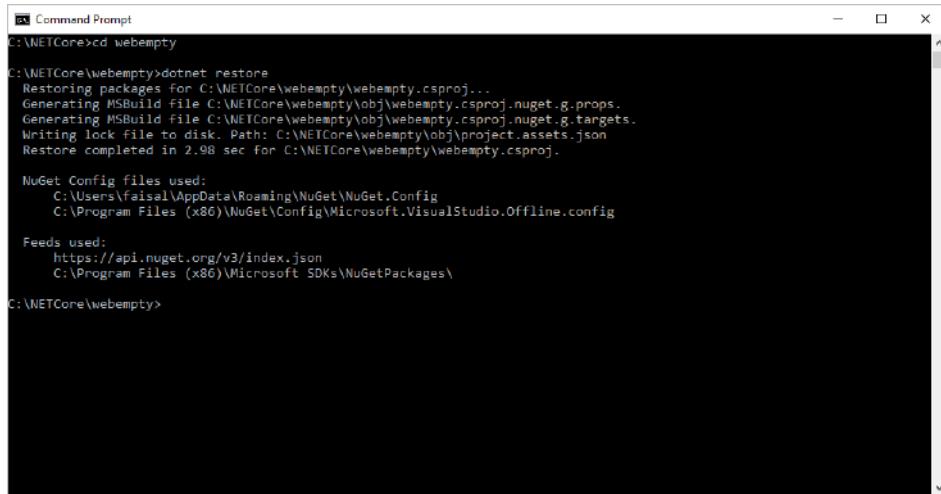
Dari gambar di atas dapat dilihat aplikasi ini memiliki komponen Model, View dan Controller. Selain itu juga dapat dilihat file webmvcauth.db yang merupakan file database SQLite.

Restore

Setelah project dibuat maka langkah selanjutnya adalah melakan proses restore library, package atau tool yang digunakan project. Caranya terlebih dahulu masuk ke folder project yang ingin direstore kemudian jalankan perintah berikut ini.

```
dotnet restore
```

Hasilnya dapat dilihat pada gambar di bawah ini.



```
PS C:\NETCore\webempty>dotnet restore
Restoring packages for C:\NETCore\webempty\webempty.csproj...
Generating MSBuild file C:\NETCore\webempty\obj\webempty.csproj.nuget.g.props.
Generating MSBuild file C:\NETCore\webempty\obj\webempty.csproj.nuget.g.targets.
Writing lock file to disk. Path: C:\NETCore\webempty\obj\project.assets.json
Restore completed in 2.98 sec for C:\NETCore\webempty\webempty.csproj.

NuGet Config files used:
  C:\Users\faisal\AppData\Roaming\NuGet\NuGet.Config
  C:\Program Files (x86)\NuGet\Config\Microsoft.VisualStudio.Offline.config

Feeds used:
  https://api.nuget.org/v3/index.json
  C:\Program Files (x86)\Microsoft SDKs\NuGetPackages\

C:\NETCore\webempty>
```

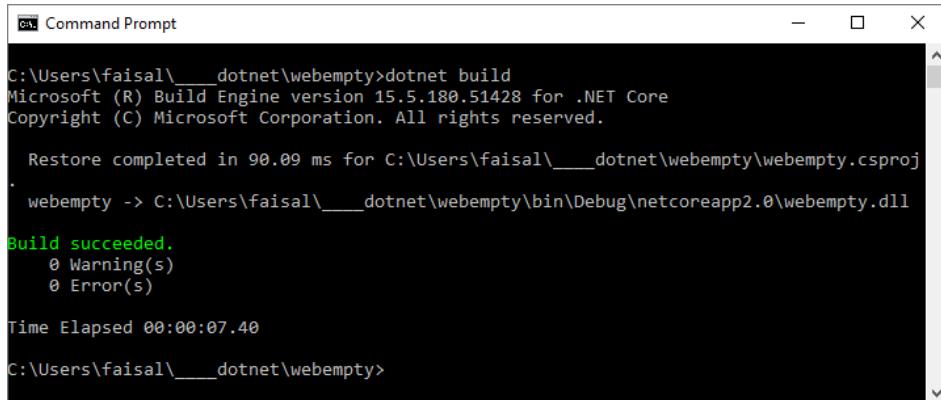
Gambar 15. dotnet restore.

Lakukan proses restore untuk ketiga project yang telah dibuat.

Build

Selanjutnya adalah melakukan proses build atau kompilasi source code yang ada di dalam project. Perintah yang digunakan adalah sebagai berikut.

```
dotnet build
```



```
PS C:\Users\faisal\___dotnet\webempty>dotnet build
Microsoft (R) Build Engine version 15.5.180.51428 for .NET Core
Copyright (C) Microsoft Corporation. All rights reserved.

  Restore completed in 90.09 ms for C:\Users\faisal\___dotnet\webempty\webempty.csproj
.
.
.
  webempty -> C:\Users\faisal\___dotnet\webempty\bin\Debug\netcoreapp2.0\webempty.dll

Build succeeded.
  0 Warning(s)
  0 Error(s)

Time Elapsed 00:00:07.40

C:\Users\faisal\___dotnet\webempty>
```

Gambar 16. dotnet build.

Run

Untuk menjalankan aplikasi digunakan perintah ini.

```
dotnet run
```

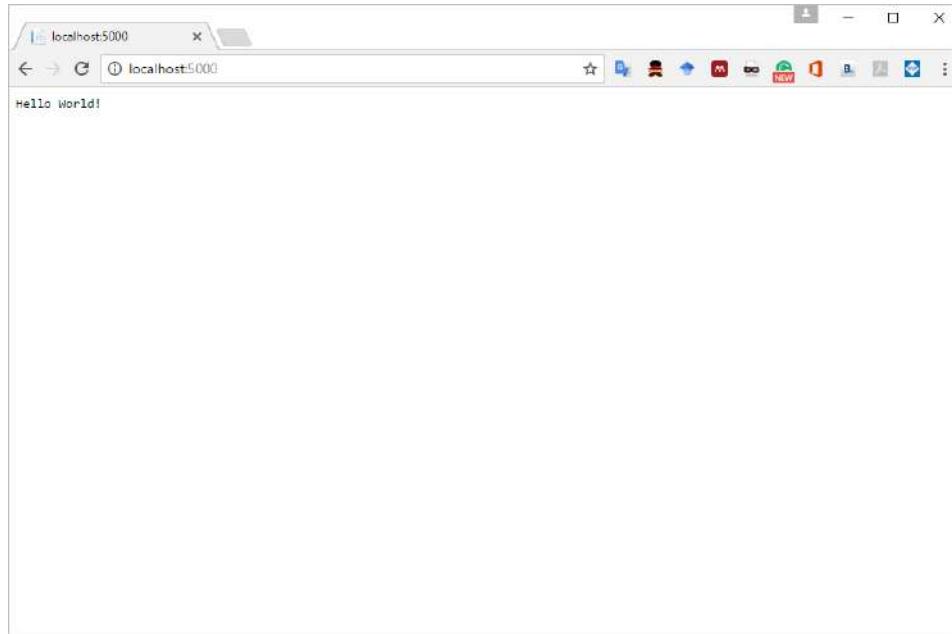
Hasilnya dapat dilihat pada informasi di bawah ini.

```
C:\NETCore\webempty>dotnet run

Hosting environment: Production
Content root path: C:\Users\faisal\___dotnet\webempty
```

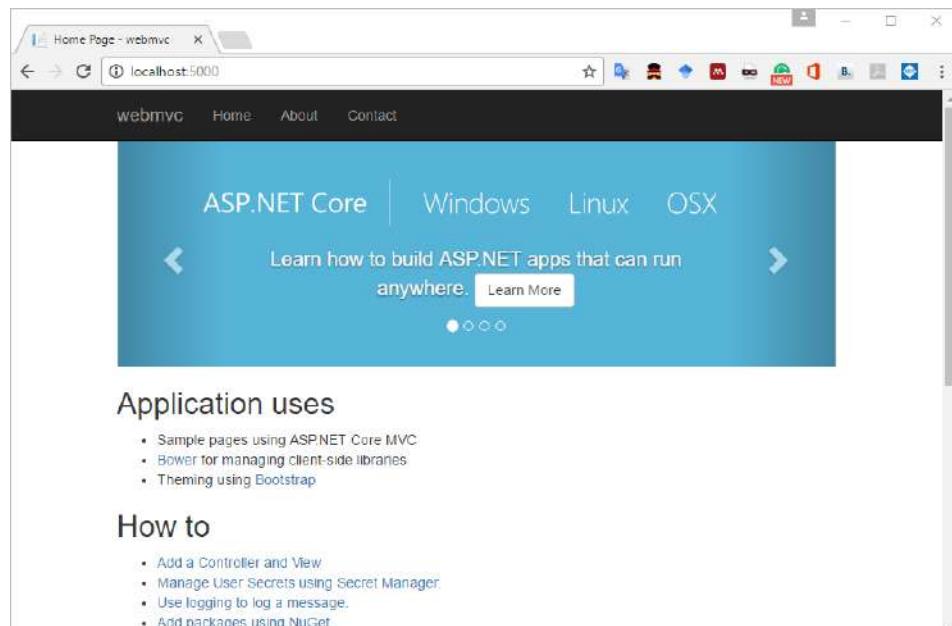
```
Now listening on: http://localhost:5000
Application started. Press Ctrl+C to shut down.
```

Dari informasi di atas dapat dilihat, untuk mengakses aplikasi pada web browser dapat dilakukan dengan mengetikkan alamat <http://localhost:5000> pada address bar web browser. Untuk menghentikan aplikasi dapat dilakukan dengan cara menekan tombol Ctrl+C.



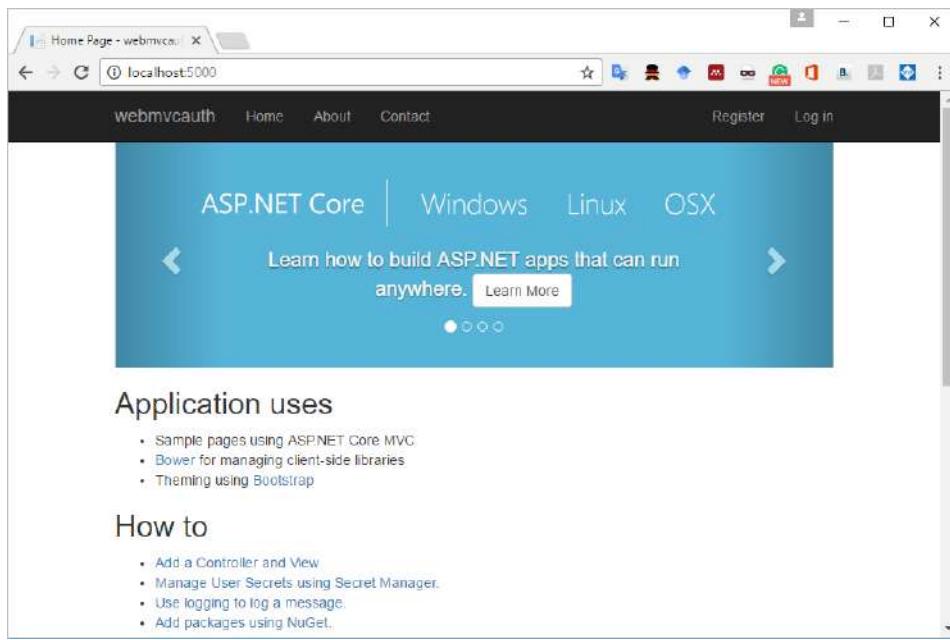
Gambar 17. Tampilan aplikasi web ASP.NET Core Empty.

Sedangkan untuk menjalankan aplikasi web project webmvc, terlebih dahulu masuk ke folder webmvc kemudian ketikkan perintah “dotnet run”. Hasilnya akan dapat dilihat tampilan web seperti berikut.



Gambar 18. Aplikasi ASP.NET Core Web.

Sedangkan untuk menjalankan aplikasi web dari project webmvcauth, maka terlebih dahulu masuk ke folder webmvcauth kemudian ketikan perintah “dotnet run”. Hasilnya dapat dilihat pada gambar di bawah ini.



Gambar 19. Aplikasi ASP.NET Core Web dengan fitur otentifikasi.

Pada gambar di atas dapat dilihat tambahan menu Register dan Login pada kanan atas dari halaman web.

Migrasi Project

Pada era Visual Studio 2015, project ASP.NET dan ASP.NET Core menggunakan file konfigurasi yang disimpan dalam file project.json. Tetapi untuk Visual Studio 2017 dan .NET Core SDK terbaru sudah tidak mengenal file konfigurasi project.json. File konfigurasi digantikan oleh file *.csproj. Sebagai contoh untuk aplikasi web project webmvcauth menggunakan file konfigurasi webmvcauth.csproj.

Untuk project yang masih menggunakan file konfigurasi project.json, maka tidak akan bisa menggunakan “dotnet restore” dari .NET Core SDK terbaru, karena perintah ini tidak menemukan file *.csproj.

Oleh karena itu perlu ada migrasi project yang dapat dilakukan dengan dua cara. Yang pertama adalah dengan cara membuat project tersebut dengan Visual Studio 2017 tapi cara ini hanya dapat dilakukan pada platform MS Windows saja. Cara yang kedua yang dapat dilakukan pada semua platform, yaitu dengan menggunakan perintah berikut.

```
dotnet migrate
```

Perintah ini akan mengubah file project.json menjadi namafolder.csproj. Kemudian menghapus file project.json. Jadi sebaiknya sebelum melakukan migrasi, lakukan backup terlebih dahulu.

3

Visual Studio 2017

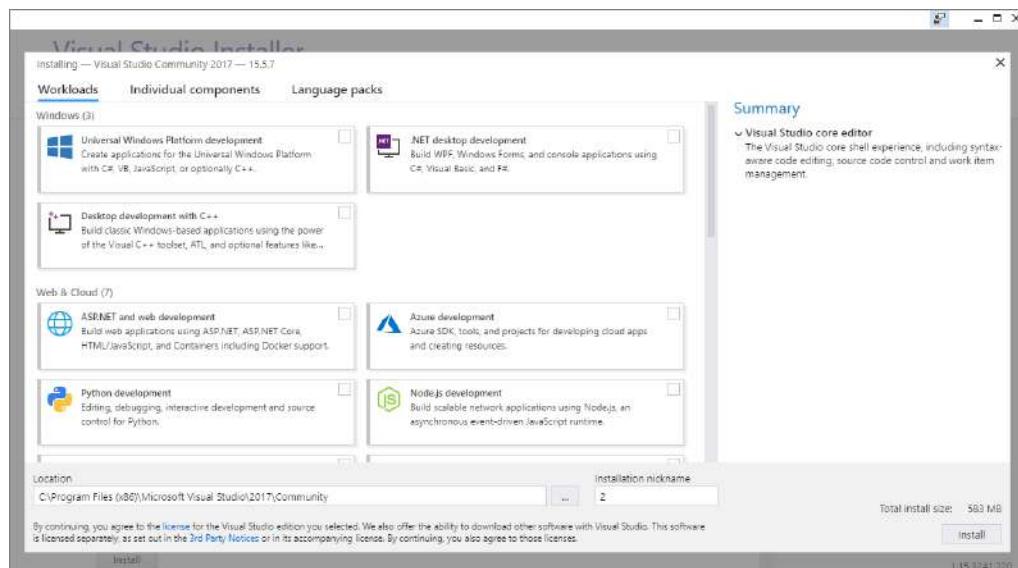
Pada bab ini diberikan panduan penggunaan Visual Studio dimulai dengan membuat solution dan project. Kemudian dilanjutkan proses eksekusi. Panduan dicontohkan dengan menggunakan Visual Studio 2017.

Installasi

Installer Visual Studio 2017 dapat diunduh di <https://www.visualstudio.com/downloads/>. Bagi pembaca yang belum memiliki lisensinya dapat mengunduh Visual Studio Community 2017 yang bersifat gratis.

File installer dengan nama vs_community_647043286.1510626584.exe (nomor di belakang vs_community_ dapat berbeda-beda tergantung waktu file diunduh) berukuran 1MB (1.094KB). Installasi Visual Studio bersifat online, sehingga file installernya berukuran kecil. File-file yang diperlukan akan diunduh secara online saat installasi. Setelah file-file yang diperlukan selesai diunduh, maka dilanjutkan dengan installasi file-file tersebut.

Berikut adalah antarmuka installer.



Gambar 20. Visual Studio 2017 – Antarmuka installasi.

Jika ingin menggunakan Visual Studio untuk membangun aplikasi web saja, maka paket yang perlu dipilih adalah:

- Web & Cloud > ASP.NET and web development.
- Other Toolset > .NET Core cross-platform development.

Untuk kedua paket tersebut berukuran sebesar 2,21GB. Sedangkan jika seluruh paket dipilih maka ukuran file installasi sekitar 20,59GB. Tetapi setelah paket diekstrak dan diinstall maka akan memerlukan ruang hardisk minimal antara 5-10GB.

File yang diunduh saat proses installasi hanya akan disimpan sementara saja, jadi tidak bisa digunakan ulang untuk installasi di mesin lain. Jika ingin mengunduh seluruh file installasi maka dapat dilakukan dengan cara mengetikkan perintah berikut ini pada DOS command prompt.

```
vs_community.exe --layout [lokasi menyimpan file] --lang en-US
```

Jika ingin menyimpan file pada folder D:\VS2017Installer maka dapat diketik seperti kode di bawah ini.

```
vs_community.exe --layout D:\ VS2017Installer --lang en-US
```

Proses ini akan mengunduh seluruh file installer. Ukuran total seluruh file installer sekitar 40-50GB.

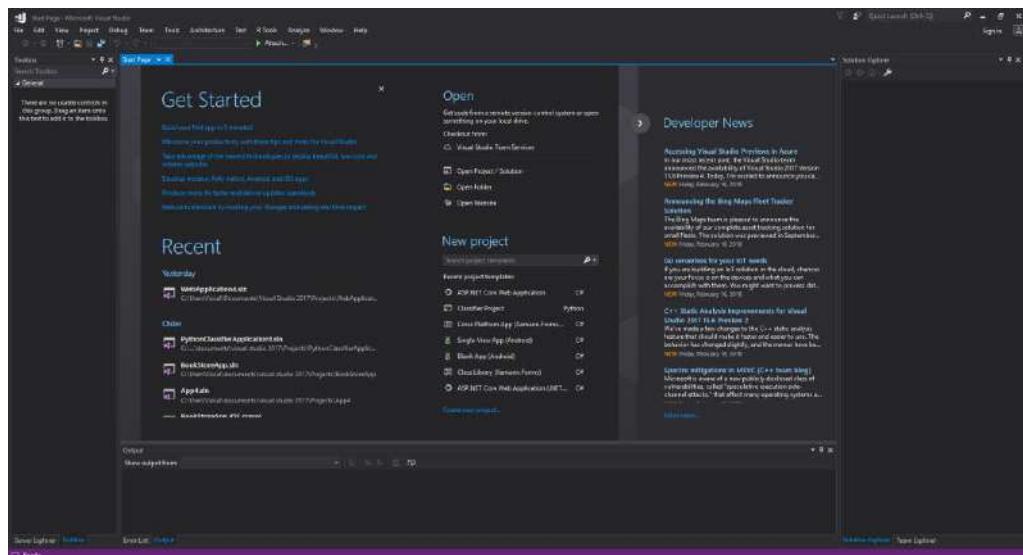
Catatan

Versi Visual Studio 2017 yang digunakan pada ebook ini adalah versi 15.6.0. Sedangkan untuk urutan installasi sebaiknya adalah sebagai berikut:

- Visual Studio 2017 ver 15.6.0.
- .NET Core SDK v2.1.4.
- .NET Core Runtime v2.0.5.
- ASP.NET Core Runtime v2.0.5

Antarmuka

Pada sub bab ini memberikan penjelasan tentang antarmuka Visual Studio 2017. Visual Studio akan menampilkan Start Page ini saat dijalankan.



Gambar 21. Visual Studio 2017 – Start Page.

Pada antarmuka ini, user dapat melakukan aksi seperti:

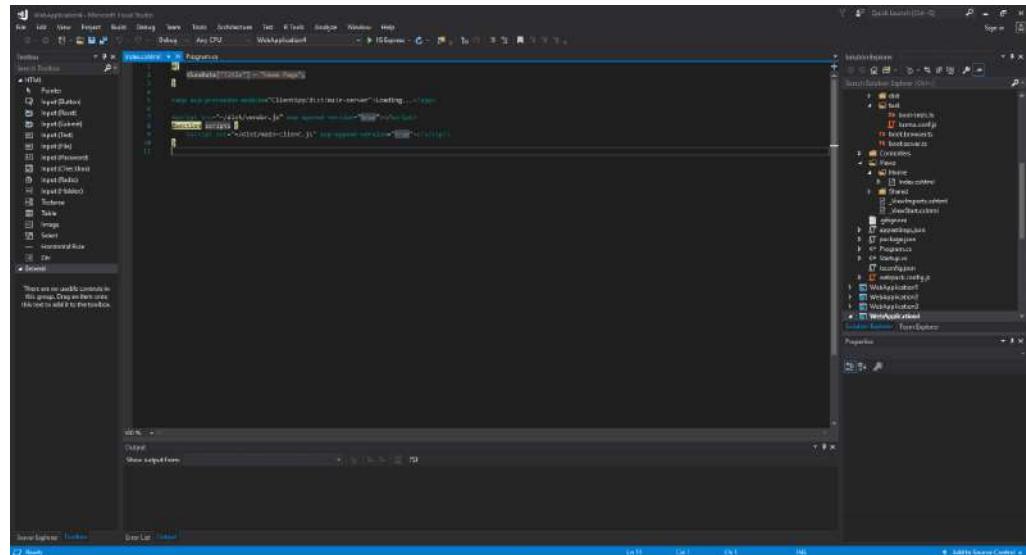
- Recent, membuka project dari daftar history project yang pernah dibuka dengan tool ini.

- Open, membuat project dari media penyimpanan lokal atau remote version control seperti GitHub.
- New Project, membuat project baru.

Seperti aplikasi desktop Windows pada umumnya, terdapat fitur yang umum yang ada yaitu:

- Menu.
- Toolbar.

Setelah salah satu project dibuka maka akan dilihat antarmuka seperti pada gambar di bawah ini.



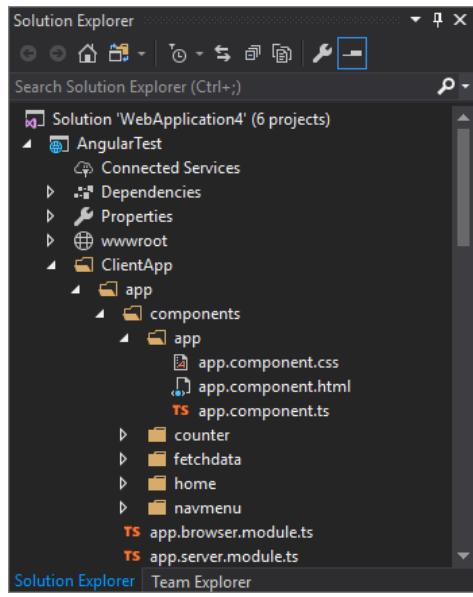
Gambar 22. Visual Studio 2017 – Membuka Project.

Antarmuka di atas terbagi atas beberapa bagian yaitu:

- Solution Explorer.
- Code Editor.
- Toolbox.
- Properties.
- Output.
- Error List.

Solution Explorer

Solution Explorer dapat dilihat pada bagian kanan atas pada Visual Studio 2017.



Gambar 23. Visual Studio 2017 – Solution Explorer.

Solution Explorer menampilkan Solution. Solution dapat berisi satu atau lebih Project. User dapat melihat daftar file yang dimiliki oleh project. User dapat memilih file yang ingin dilihat.

Editor

Editor berfungsi untuk menambah atau memodifikasi item atau kode baik secara visual ataupun kode dari file yang dipilih. Pada gambar di bawah ini contoh Code Editor. Code Editor berfungsi untuk mengedit file text.

```
Index.cshtml ✘ X Program.cs
1  @{
2      ViewData["Title"] = "Home Page";
3  }
4
5  <app asp-prerender-module="ClientApp/dist/main-server">Loading...</app>
6
7  <script src="~/dist/vendor.js" asp-append-version="true"></script>
8  @section scripts {
9      <script src="~/dist/main-client.js" asp-append-version="true"></script>
10 }
11
```

The screenshot shows the Visual Studio 2017 Code Editor with the file 'Index.cshtml' open. The code in the editor is as follows:

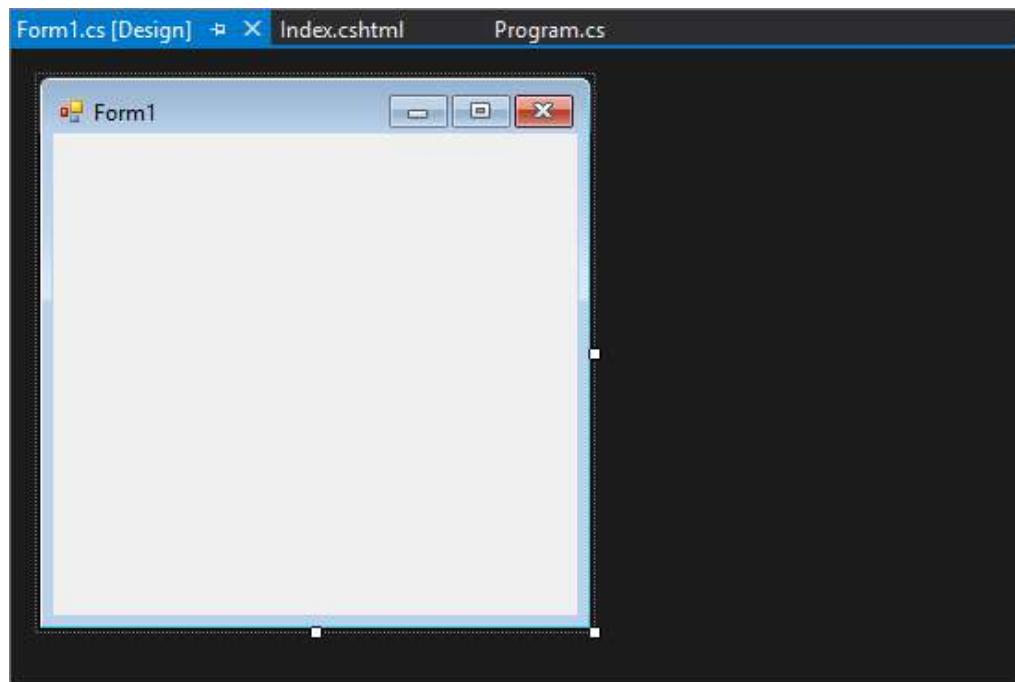
```

Index.cshtml ✘ X Program.cs
1  @{
2      ViewData["Title"] = "Home Page";
3  }
4
5  <app asp-prerender-module="ClientApp/dist/main-server">Loading...</app>
6
7  <script src="~/dist/vendor.js" asp-append-version="true"></script>
8  @section scripts {
9      <script src="~/dist/main-client.js" asp-append-version="true"></script>
10 }
11

```

Gambar 24. Visual Studio 2017 – Code Editor.

Saat file yang dibuka adalah antarmuka dari suatu aplikasi seperti pada contoh di bawah ini, maka Editor akan beralih fungsi sebagai Visual Editor.

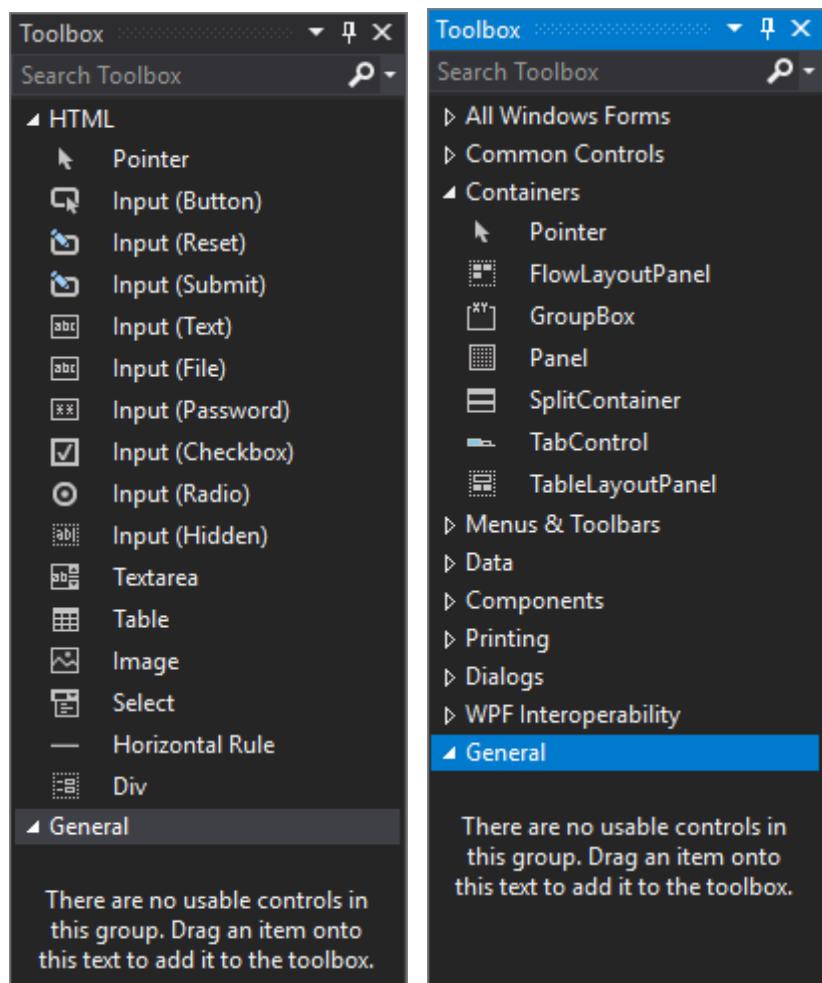


Gambar 25. Visual Studio 2017 – Visual Editor.

Visual Editor memberikan kemampuan pada user untuk melakukan modifikasi antarmuka dengan cara drag-n-drop item atau komponen visual.

Toolbox

Toolbox akan menampilkan item-item yang sesuai dengan file yang sedang aktif.

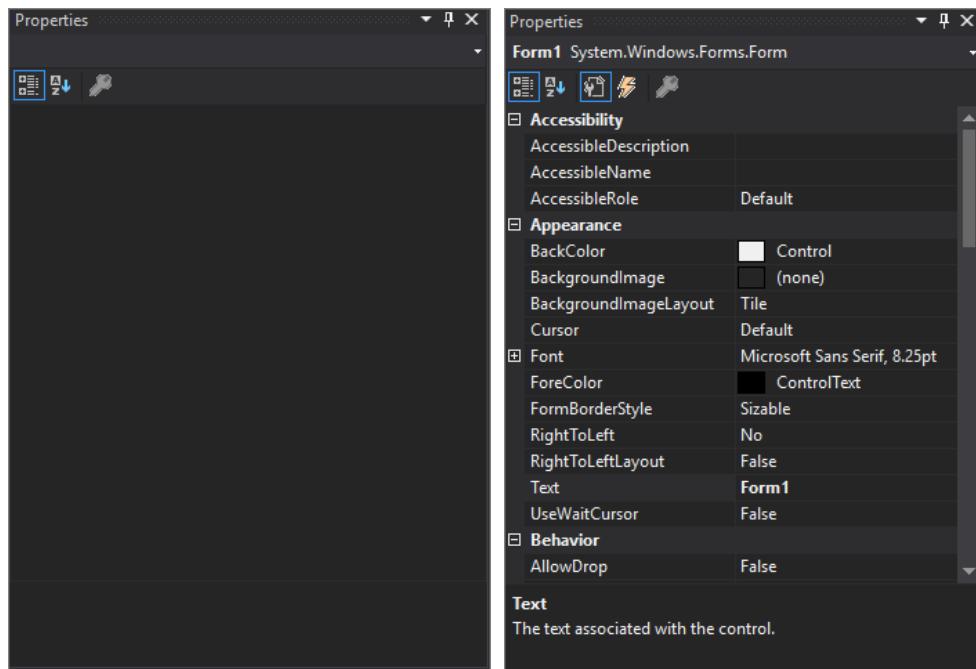


Gambar 26. Visual Studio 2017 – Toolbox.

Sebagai contoh, ketika file HTML dibuka maka akan ditampilkan daftar item-item seperti yang dilihat pada gambar di sebelah kiri. Sedangkan ketika file Window Form dibuka maka Toolbox akan menampilkan daftar item seperti pada gambar di sebelah kanan.

Properties

Bagian Properties menampilkan detail informasi dari item yang dipilih pada Editor. Biasanya informasi yang ditampilkan pada bagian ini berkaitan dengan item yang dipilih saat mode Visual Editor.

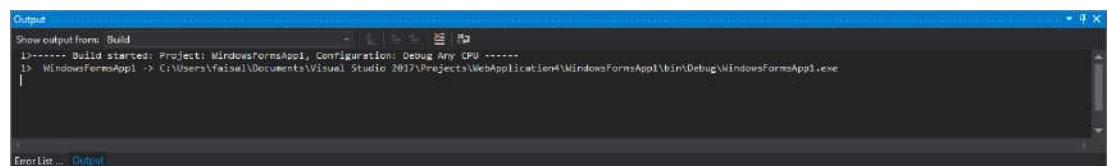


Gambar 27. Visual Studio 2017 – Properties.

Gambar di sebelah kanan tidak menampilkan informasi, hal ini terjadi ketika Editor sedang menampilkan file text. Saat mode Editor Visual, maka dapat dilihat property-property yang dimiliki oleh file yang sedang dibuka.

Output

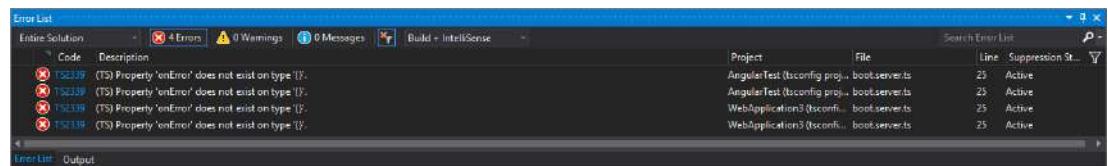
Bagian Output menampilkan keluaran dari proses yang dijalankan pada Visual Studio. Seperti proses build atau kompilasi kode. Selain itu bagian ini juga menampilkan keluaran dari program yang dijalankan, sebagai contoh keluaran dari program berbasis Command Line.



Gambar 28. Visual Studio 2017 – Output.

Error List

Bagian Error List menampilkan error yang terjadi baik dari kode yang ditulis, atau error yang terjadi karena program yang dibuat melakukan kesalahan atau mendapatkan pesan error dari sistem lain. Sebagai ketika program yang dibuat tidak bisa melakukan koneksi ke database.



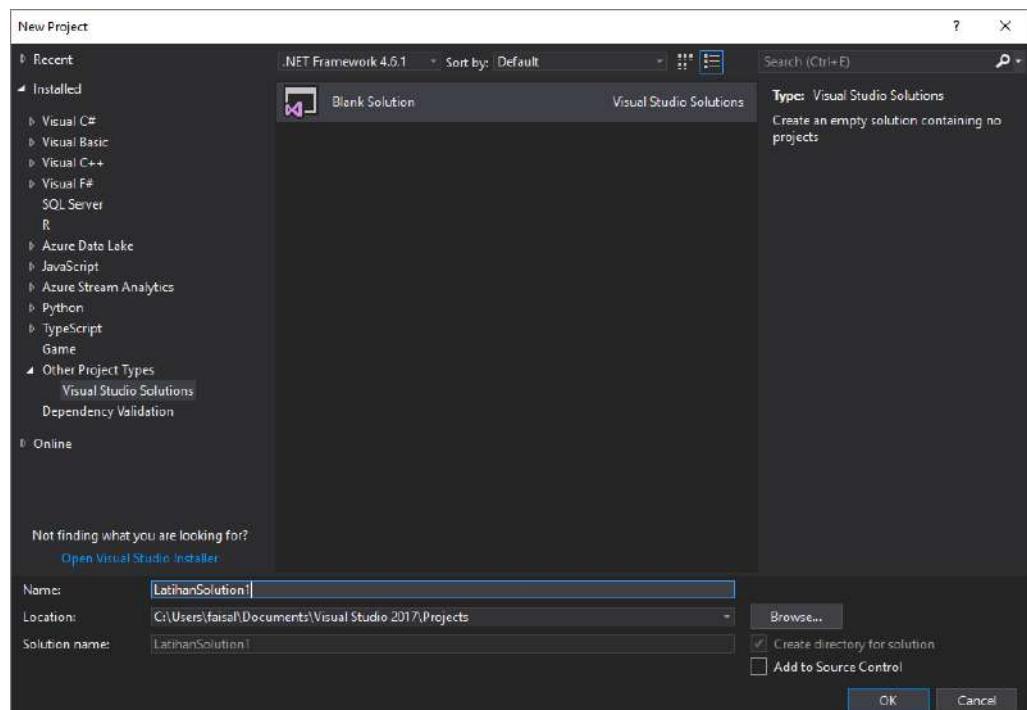
Gambar 29. Visual Studio 2017 – Error List.

Solution & Project

Solution adalah tempat yang dapat berisi banyak project. Setiap solution akan memiliki sebuah file *.sln. Setiap solution akan disimpan ke dalam folder dengan nama yang sama dengan nama solutionnya. Pada Visual Studio, solution dapat dilihat pada bagian Solution Explorer.

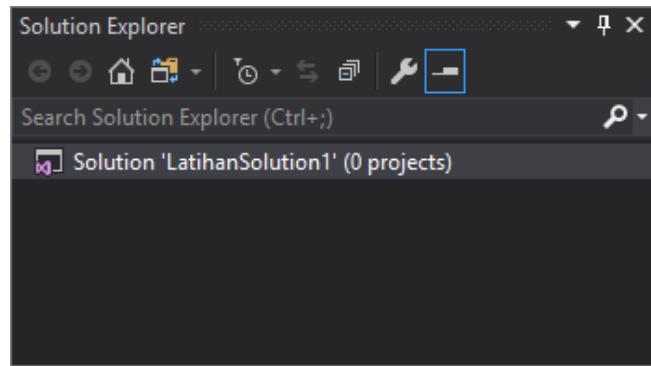
Solution

Pada Visual Studio, solution dapat dibuat dengan dua cara. Yang pertama adalah dengan membuat solution kosong. Langkah pertama adalah dengan membuat project baru dengan cara memilih File > New > Project.



Gambar 30. Visual Studio 2017 – Blank solution.

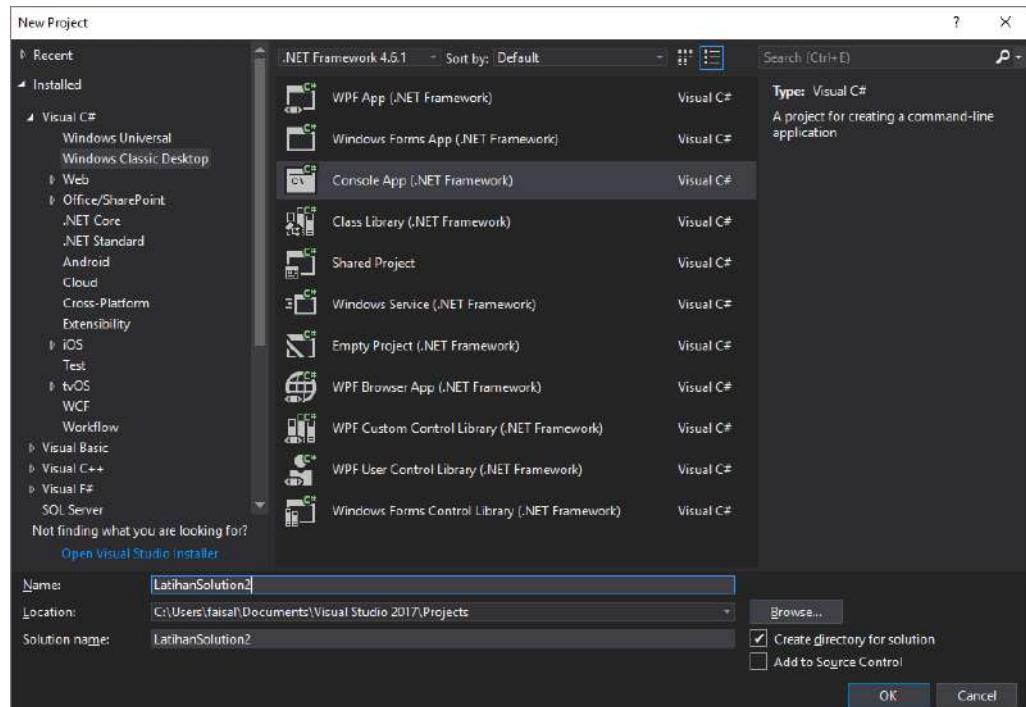
Setelah itu Visual Studio akan menampilkan window New Project seperti pada gambar di atas. Pada daftar Installed pilih Other Project Types > Visual Studio Solutions. Kemudian pilih Blank Solution. Selanjutnya tulis nama solution pada kolom Name. Klik tombol Browse pada kolom Location jika ingin menentukan lokasi penyimpanan. Selanjutnya klik tombol OK.



Gambar 31. Visual Studio 2017 – Blank Solution.

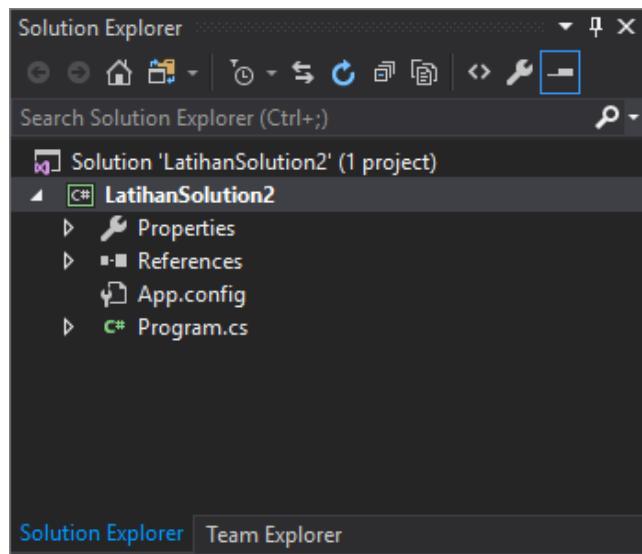
Cara kedua adalah dengan membuat project baru. Untuk mencoba cara ini maka solution di atas harus ditutup terlebih dahulu. Untuk menutup solution pilih File > Close Solution pada menu. Hasilnya dapat dilihat pada bagian Solution Explorer yang kembali kosong.

Untuk membuat project dapat dilakukan dengan cara memilih File > New > Project.



Gambar 32. Visual Studio 2017 – Project.

Sebagai contoh pilih Installed > Visual C# > Windows Classic Desktop > Console App (.NET Framework). Setelah mengisi nama solution klik tombol OK.



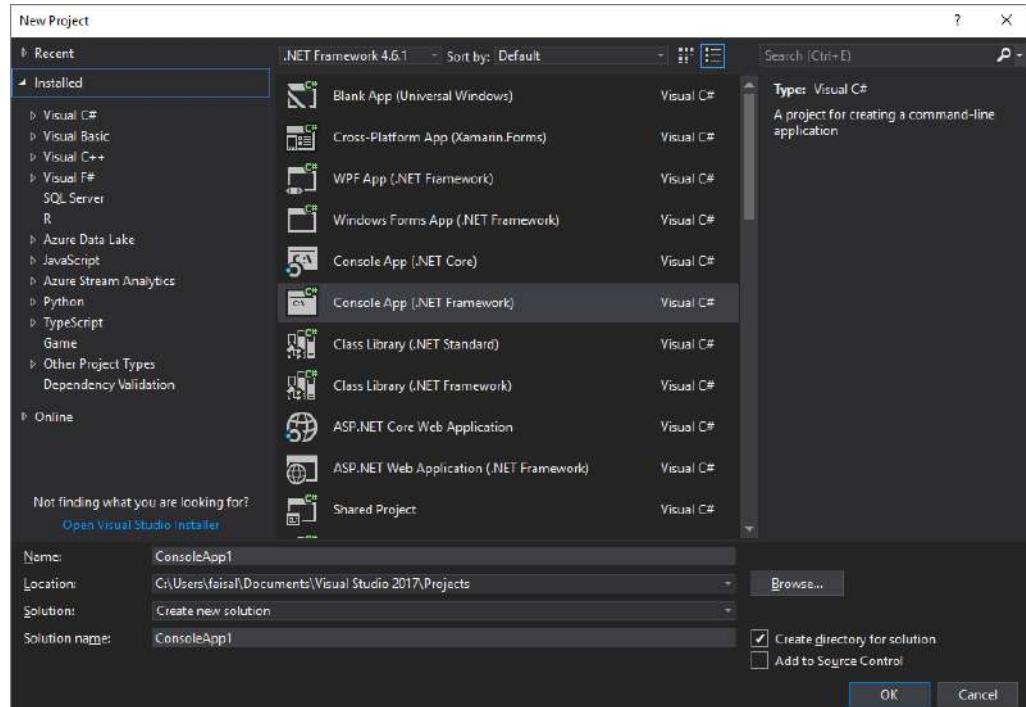
Gambar 33. Visual Studio 2017 – Solution & Project.

Pada gambar di atas dapat dilihat Solution dengan nama LatihanSolution2 yang didalamnya terdapat project dengan nama LatihanSolution2.

Penulis menyarankan untuk menggunakan cara pertama, karena sebaiknya nama solution dibuat berbeda dengan nama project didalamnya.

Project

Pada sub bab sebelumnya telah diperlihatkan cara membuat project dengan Visual Studio. Visual Studio menyediakan template project yang dikelompokkan berdasarkan bahasa pemrograman, teknologi atau platform, dan tipe aplikasi.



Gambar 34. Visual Studio 2017 – Template project.

Pada gambar di atas dapat dilihat beberapa bahasa pemrograman yang dapat dipilih, yaitu:

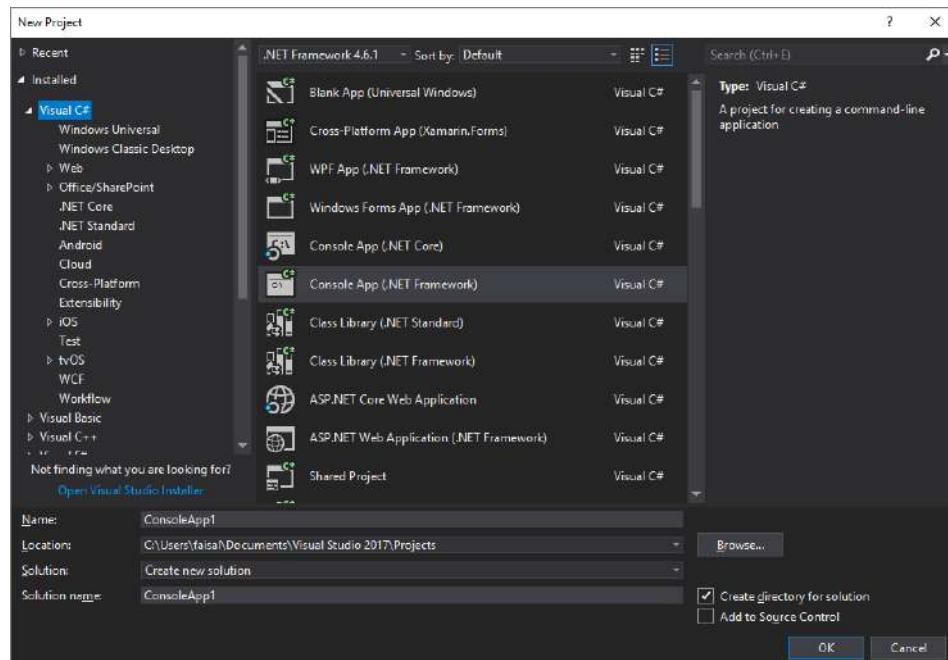
- Visual C#.

- Visual Basic.
- Visual C++.
- Visual F#.
- R.
- JavaScript.
- Python.
- TypeScript.

Selain itu terdapat pilihan platform yang didukung yaitu:

- SQL Server.
- Azure Data Lake.
- Azure Stream Analytics.
- Game.

Jika sebuah bahasa pemrograman dipilih maka dapat dilihat beberapa kelompok tipe aplikasi yang dapat dipilih seperti pada gambar di bawah ini.

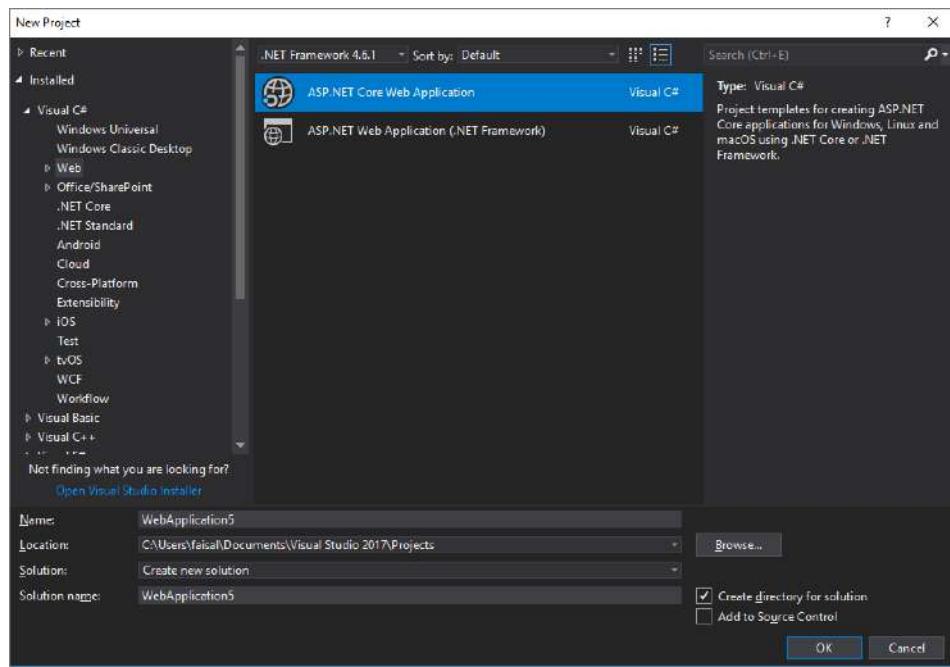


Gambar 35. Visual Studio 2017 – Tipe aplikasi yang dapat dikembangkan dengan bahasa Visual C#.

Dengan bahasa pemrograman Visual C# dapat dibangun aplikasi, yaitu:

- Windows Classic Desktop.
- Web.
- Android.
- iOS.
- Dan lain-lain.

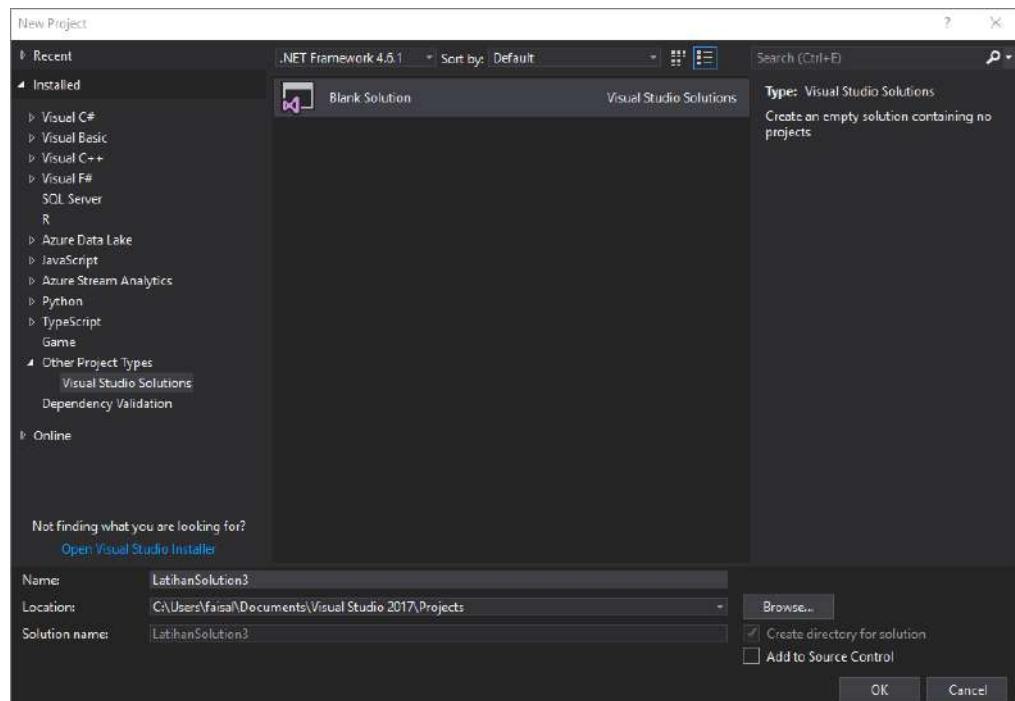
Sebagai contoh jika dipilih tipe aplikasi Web maka dapat dilihat daftar template project seperti pada gambar berikut ini.



Gambar 36. Visual Studio 2017 – Daftar template project Web dengan C#.

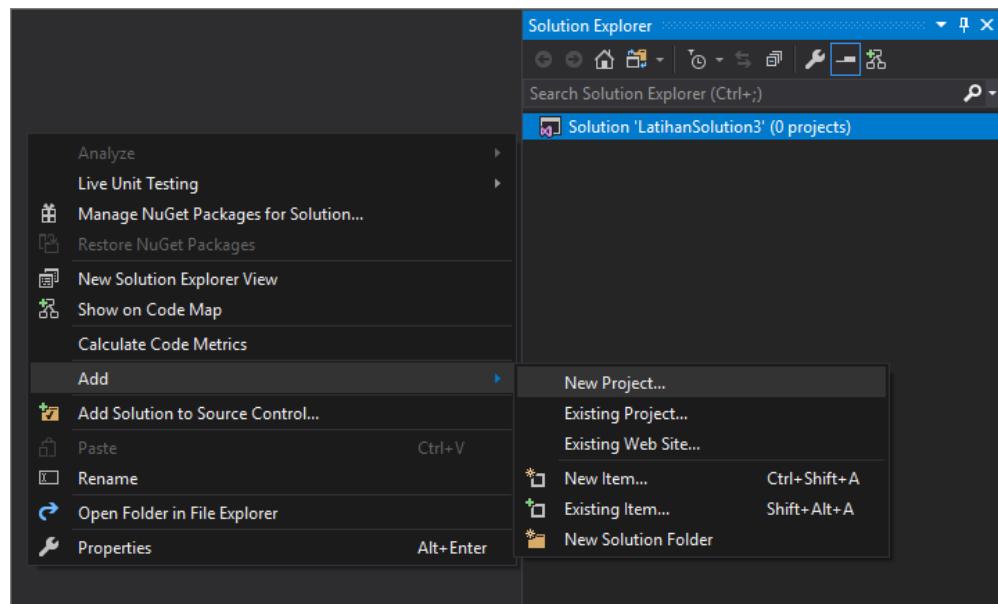
Selanjutnya akan dijelaskan bagaimana membuat solution kosong kemudian menambahkan beberapa project didalamnya.

Pertama adalah memastikan tidak ada solution yang sedang dibuka. Kemudian pilih File > New > Project. Kemudian pada window New Project pilih Installed > Other Project Types > Visual Studio Solution. Pilih Blank Solution kemudian isi kolom Name dengan nama solution yang diinginkan. Diakhiri dengan mengklik tombol OK.



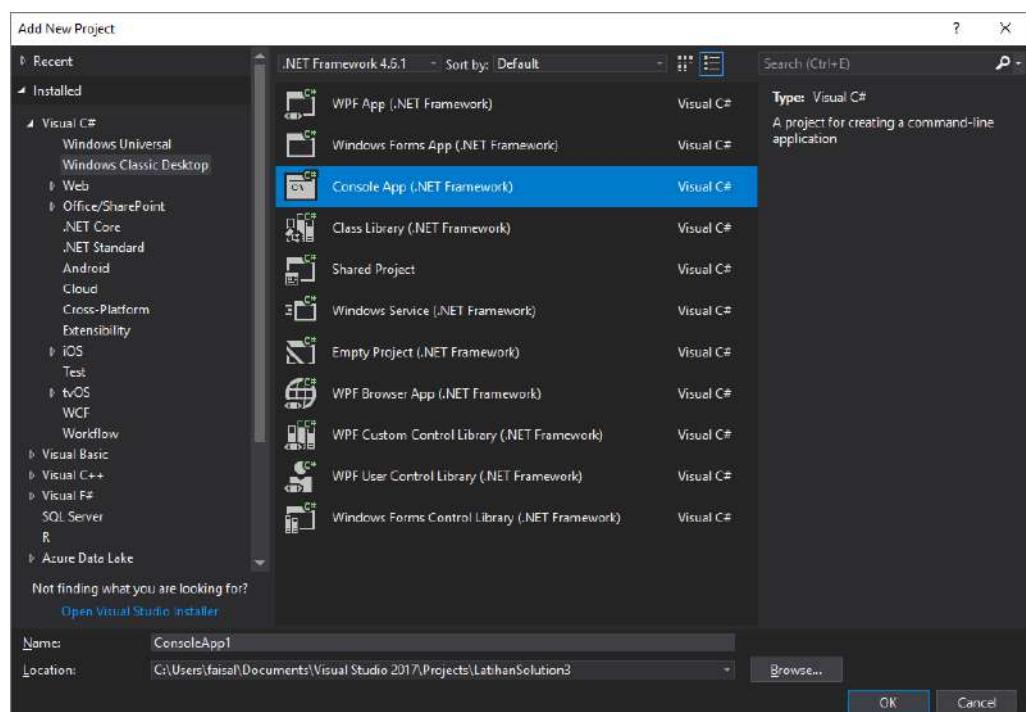
Gambar 37. Visual Studio 2017 – Membuat Blank Solution.

Selanjutnya akan dicontohkan untuk menambahkan dua project ke dalam solution ini. Caranya dengan mengklik kanan pada solution LatihanSolution3 yang terdapat di dalam Solution Explorer.



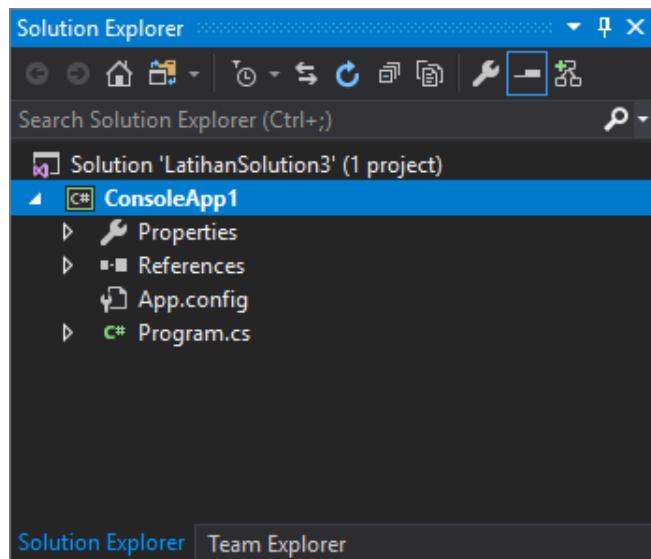
Gambar 38. Visual Studio 2017 – Menambah project ke dalam solution.

Pilih Add > New Project. Untuk contoh pertama menambahkan project console, dengan cara memilih Visual C# > Windows Classic Desktop > Console App (.NET Framework).



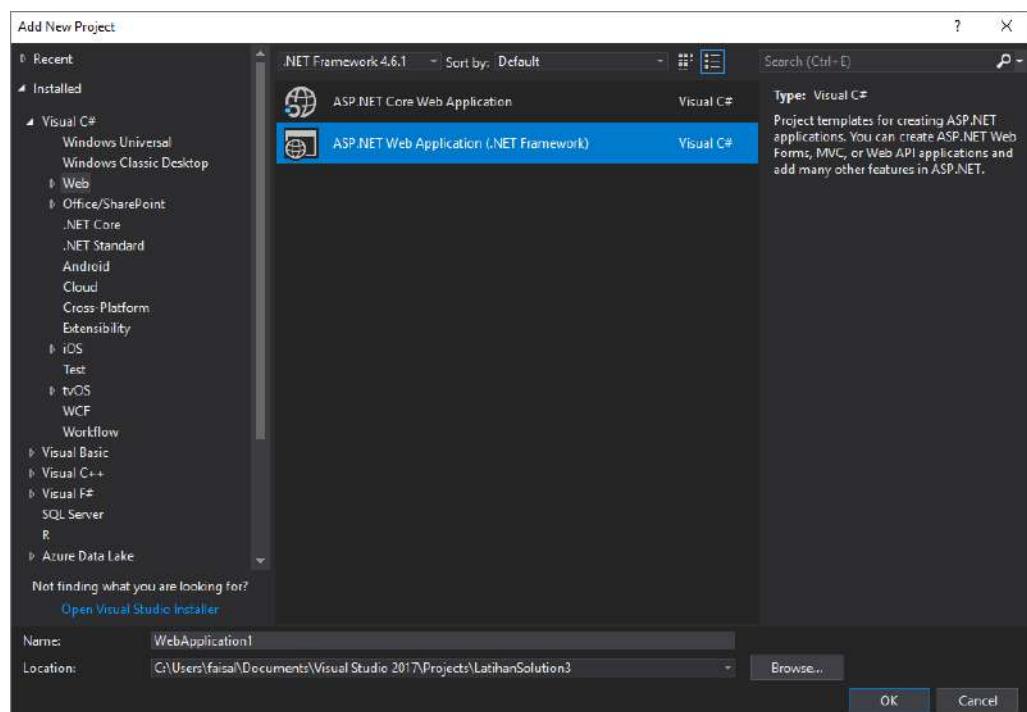
Gambar 39. Visual Studio 2017 – Menambahkan project Console App.

Setelah nama solution diisi dan tombol OK diklik maka dapat dilihat project ini di dalam Solution Explorer.



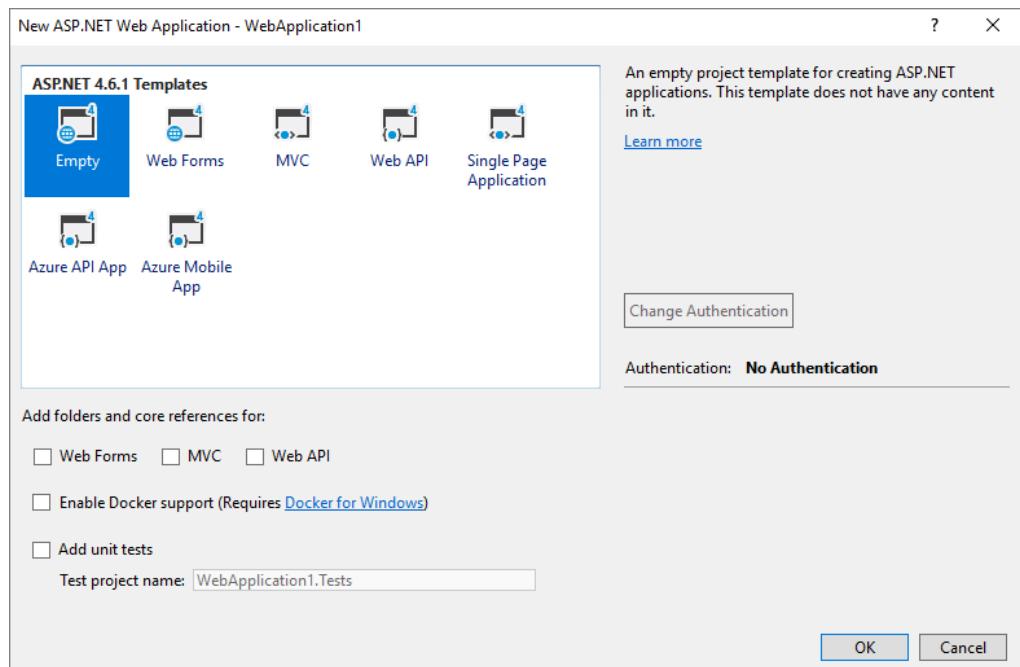
Gambar 40. Visual Studio 2017 – ConsoleApp1.

Project kedua ditambahkan dengan cara yang sama.



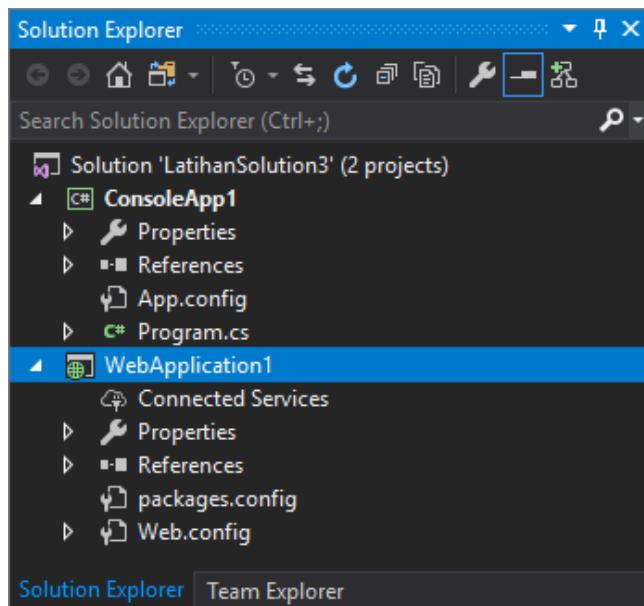
Gambar 41. Visual Studio 2017 – Menambahkan project web.

Setelah tombol OK diklik maka akan ditampilkan window berikut ini. Pada window ini terdapat pilihan template web application yang didukung oleh .NET Framework 4.6.1.



Gambar 42. Visual Studio 2017 – Template project web ASP.NET 4.6.1.

Sebagai contoh dipilih template Empty kemudian klik tombol OK. Hasilnya dapat dilihat pada gambar di bawah ini.



Gambar 43. Visual Studio 2017 – Menambahkan project web application.

Dari penjelasan di atas maka sangat dimungkinkan untuk memiliki sebuah solution yang didalamnya berisi banyak project seperti:

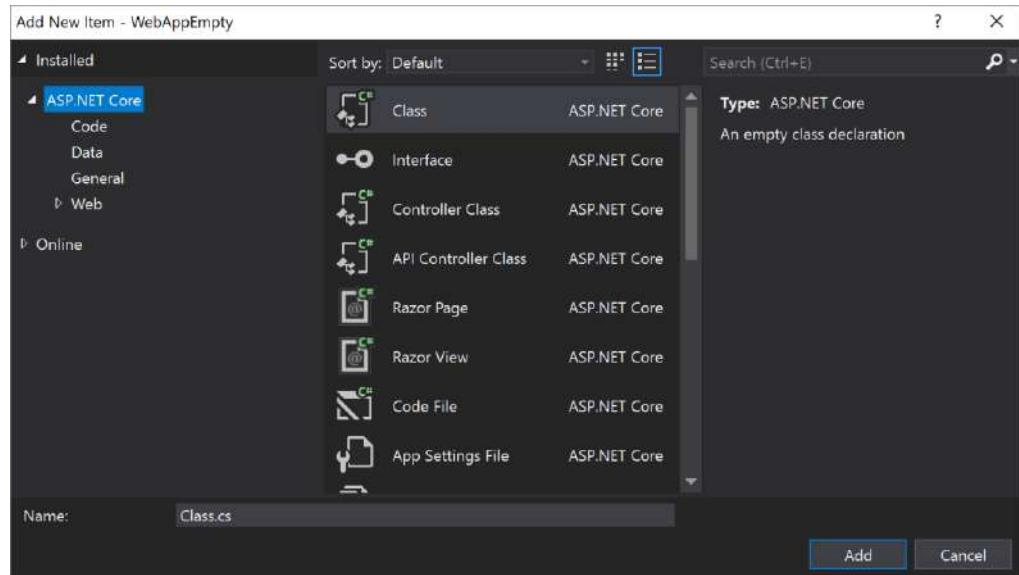
- Project web application.
- Project mobile application.
- Project desktop application.
- Dan lain-lain.

Item

Item adalah sesuatu yang dapat ditambahkan ke dalam project. Item dapat berupa:

- Class.
- Image.
- File HTML.
- Dan lain-lain.

Untuk menambahkan item pada project dapat dilakukan dengan cara klik kanan pada project kemudian pilih Add > New Item.



Gambar 44. Visual Studio 2017 - Add New Item.

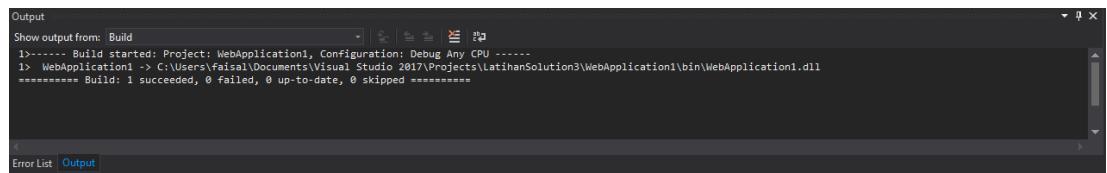
Kemudian pilih template file yang diinginkan, kemudian berikan nama item pada kolom Name. Item yang ditampilkan pada window itu akan sesuai dengan tipe project.

Build & Debug

Pada pembangunan aplikasi desktop atau mobile, file-file kode program akan dikompilasi menjadi file executable dengan format binary. Untuk aplikasi web, file-file kode program akan dideploy ke web server kemudian setelah web server aktif maka aplikasi web tersebut dapat dijalankan dengan menggunakan web browser yang mengakses alamat web server tersebut.

Proses kompilasi dikenal juga dengan istilah Build. Untuk melakukan proses ini dengan Visual Studio, pengguna dapat melakukannya dengan cara klik kanan pada solution yang ada pada Solution Explorer kemudian pilih Build Solution. Dengan cara ini maka proses build akan dilakukan pada seluruh project yang ada di dalam solution. Jika proses build hanya ingin dilakukan pada salah satu project saja, maka lakukan klik kanan pada project yang diinginkan pada Solution Explorer kemudian pilih Build.

Status proses ini dapat dilihat pada bagian Output.

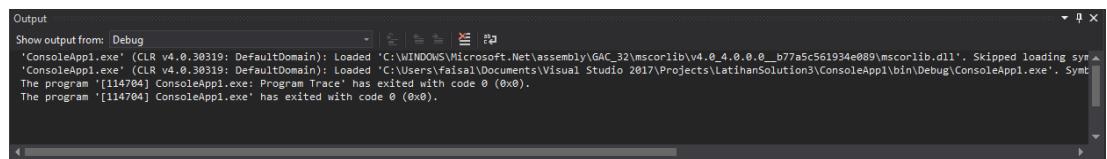


Gambar 45. Visual Studio 2017 – Output proses build.

Proses Debug adalah proses untuk menjalankan StartUp project atau project yang diinginkan. StartUp project biasanya adalah project yang pertama kali dibuat pada solution. Pada Solution Explorer, StartUp project dapat dibedakan dengan nama project dengan cetak tebal seperti yang terlihat pada project ConsoleApp1 pada gambar di atas. Jika proses debug dilakukan dengan cara memilih Debug > Start Debugging maka StartUp project akan dijalankan oleh Visual Studio.

Sedangkan jika ingin melakukan proses debug pada project yang diinginkan saja maka klik kanan pada project yang ada di Solution Explorer kemudian pilih Debug > Start new instance.

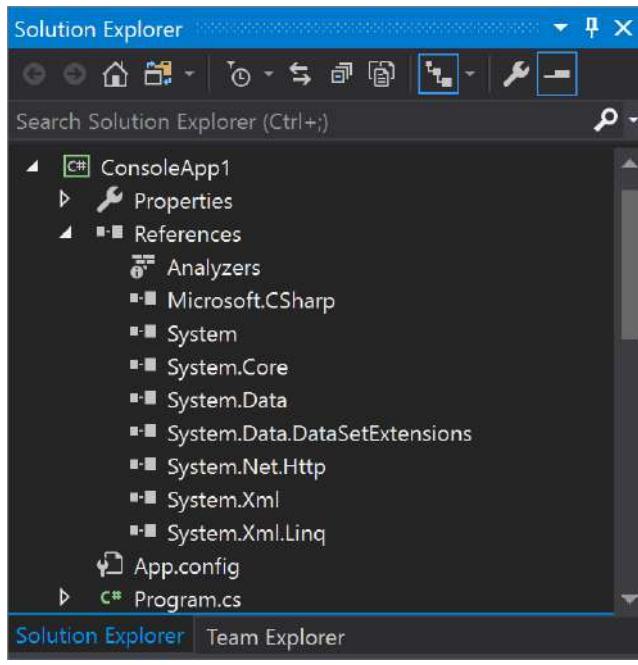
Status proses debug akan dapat dilihat pada bagian Output. Sebagai contoh dilakukan proses debug pada project ConsoleApp1.



Gambar 46. Visual Studio 2017 – Output proses debug.

Reference

Sebuah project memiliki library default. Library default ini akan berbeda-beda antar setiap tipe project. Sebagai contoh project console akan memiliki library yang berbeda jika dibandingkan dengan project mobile. Untuk melihat library apa saja yang dimiliki oleh suatu project, maka dapat dilihat dengan cara membuka References pada project.



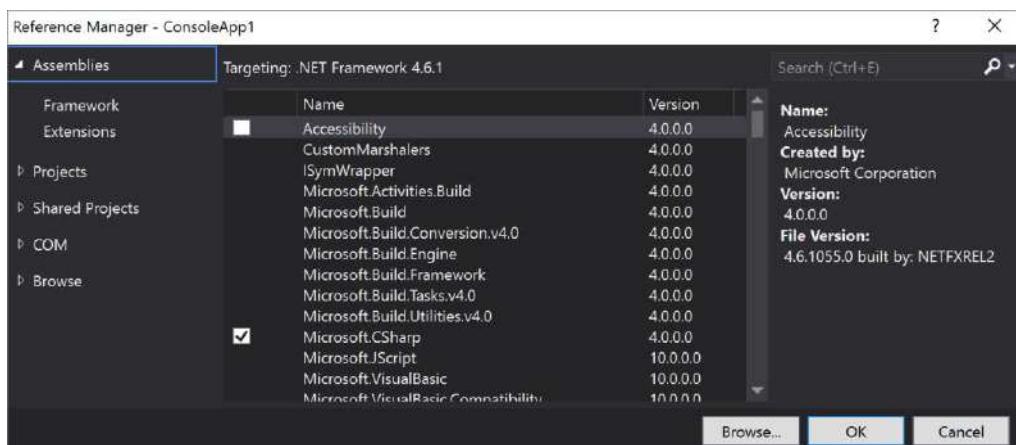
Gambar 47. Visual Studio 2017 - References.

Pada gambar di atas dapat dilihat bahwa project tipe console memiliki library default dengan nama namespace berikut ini:

- Microsoft.Csharp.
- System.
- System.Core.
- System.Data.
- System.Data.DataSetExtensions.
- System.Net.Http.
- System.Xml.
- System.Xml.Linq.

Selain library default tersebut, pengguna dapat menambahkan library lain yang tersedia atau library dari suatu project yang sedang dibangun.

Untuk menambahkan library pada project dapat dilakukan dengan cara klik kanan pada References di project yang diinginkan. Kemudian pilih Add Reference.



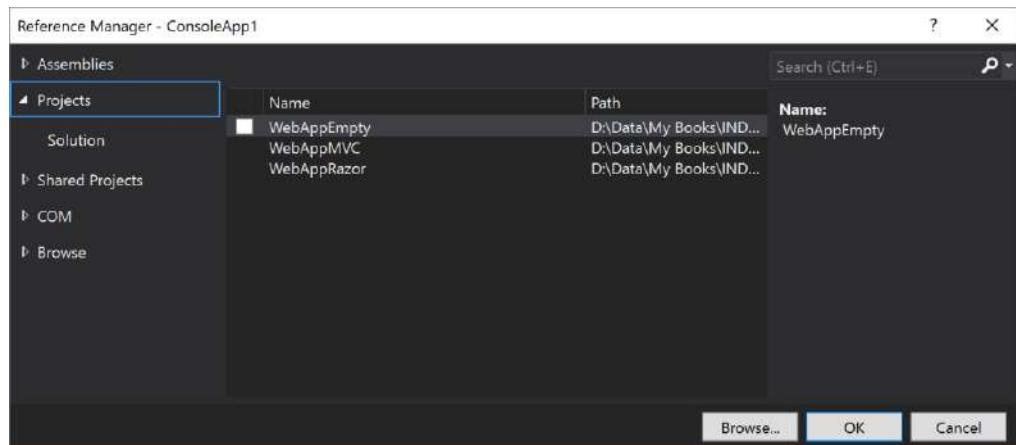
Gambar 48. Visual Studio 2017 - Reference Manager - Assemblies.

Pengguna dapat memilih library dari 3 tipe, yaitu:

- Assembly.
- Project.
- COM.

Pada gambar di atas adalah contoh ketika tab Assemblies aktif. Dapat dilihat daftar assembly yang tersedia dan juga yang telah digunakan pada project. Assembly yang telah digunakan pada project memiliki tanda check seperti gambar di atas.

Jika pada solution sedang dikembangkan project tipe class library, maka project tersebut dapat ditambahkan pada project dengan mengklik tab Projects. Maka dapat dilihat daftar project yang ada pada solution seperti pada gambar di bawah ini (daftar project akan berbeda dengan pembaca, dan mungkin kosong jika pada solution hanya ada 1 project utama saja).



Gambar 49. Visual Studio 2017 - Reference - Projects.

Selain itu library juga bisa ditambahkan dengan cara menambahkan file library *.DLL. Caranya dengan mengklik tombol Browse kemudian pilih file *.DLL yang ingin ditambahkan.

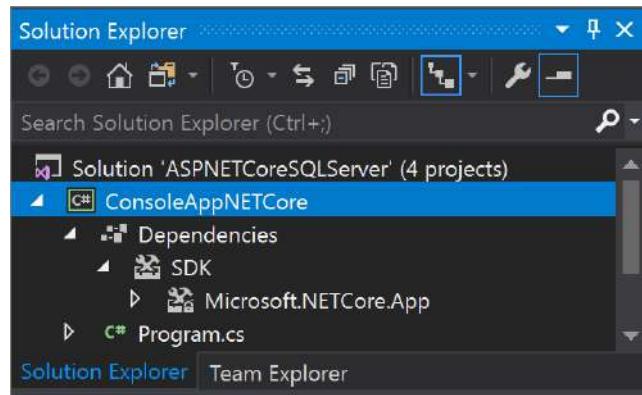
Setelah memilih salah satu assembly atau project maka tersebut maka hasilnya akan dapat dilihat pada daftar bagian References pada project di bagian Solution Explorer.

Catatan

Tidak semua project memiliki Reference seperti contoh project di atas. Project yang memiliki Reference adalah project yang dibangun dengan menggunakan .NET Framework. Sedangkan project yang dibangun dengan menggunakan .NET Core Framework akan menggunakan cara lain untuk menambahkan library.

NuGet

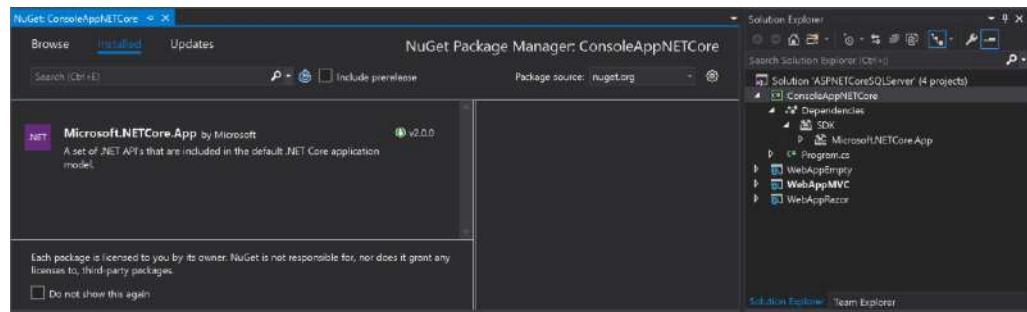
Untuk project yang dibangun dengan .NET Core Framework maka salah satu cara untuk menambahkan library dengan menggunakan NuGet. Sebagai contoh, pada gambar di bawah ini terdapat project dengan nama ConsolAppNETCore yang dibuat dengan menggunakan template project Console App (.NET Core).



Gambar 50. Visual Studio 2017 - ConsoleAppNETCore.

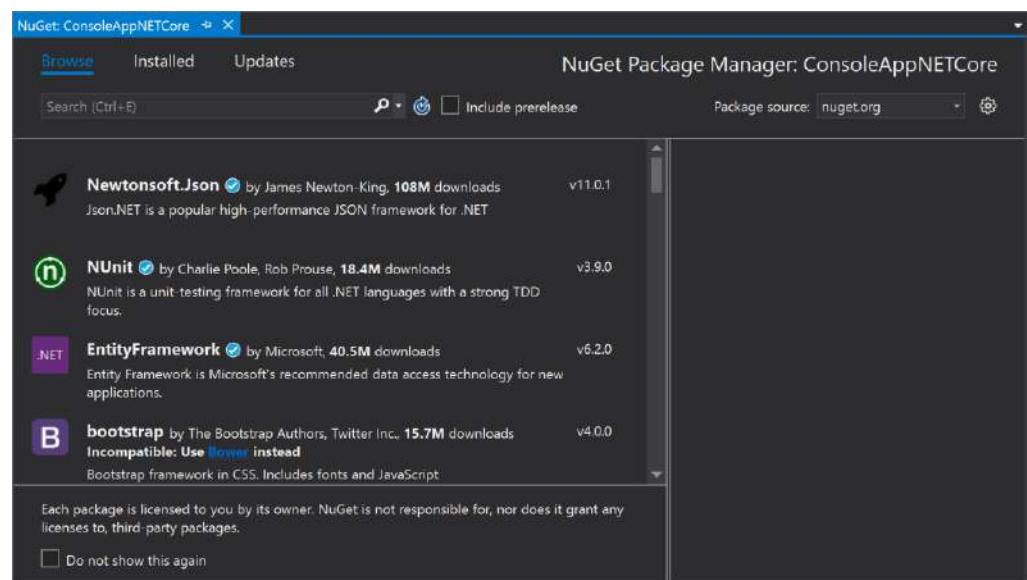
Jika dibandingkan dengan project ConsolApp pada sub bab sebelumnya, terlihat perbedaan struktur project ini. Pada project ini tidak terdapat References. Sebagai gantinya pada project ini terdapat Dependencies yang berisi library-library yang digunakan oleh project ini.

Untuk menambahkan library pada project ini dapat menggunakan NuGet. Caranya klik kanan pada project kemudian pilih Manage NuGet Packages.



Gambar 51. Visual Studio 2017 - NuGet Package Manager - Installed.

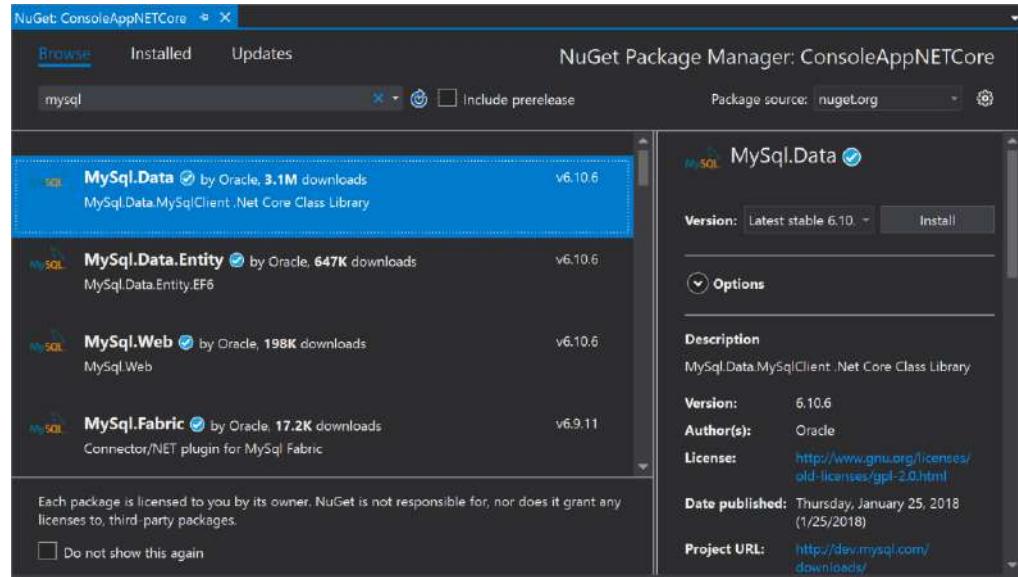
Maka akan ditampilkan daftar library yang telah diinstall. Jika ingin menginstall library maka klik tab Browse. Maka akan ditampilkan library yang tersedia secara online.



Gambar 52. Visual Studio 2017 - NuGet Package Manager - Browse.

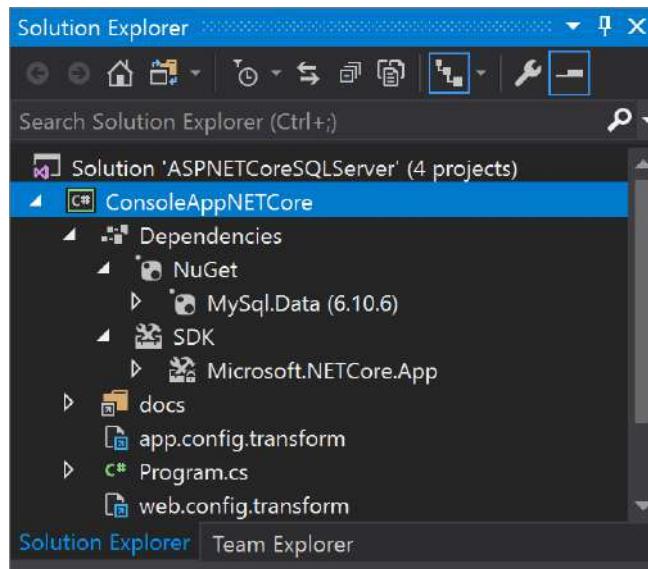
Untuk itu harus dipastikan komputer pengguna terkoneksi dengan internet jika ingin menggunakan fitur ini.

Jika ingin mencari library yang diinginkan, maka bisa mengetikan kata kunci pada kolom Search.



Gambar 53. Visual Studio 2017 - NuGet Package Manager - Search & Install.

Misal library yang diinginkan telah ditemukan adalah MySql.Data, klik library tersebut maka di samping kanan daftar dapat dilihat informasi detail dari library tersebut. Jika ingin menginstallnya maka klik tombol Install. Kemudian ikutin instruksi selanjutnya.



Gambar 54. Visual Studio 2017 - NuGet pada Solution Explorer.

Hasilnya dapat dilihat pada Solution Explorer, pada Dependencies di project terdapat tambahan NuGet > MySQL.Data (6.10.6).

4

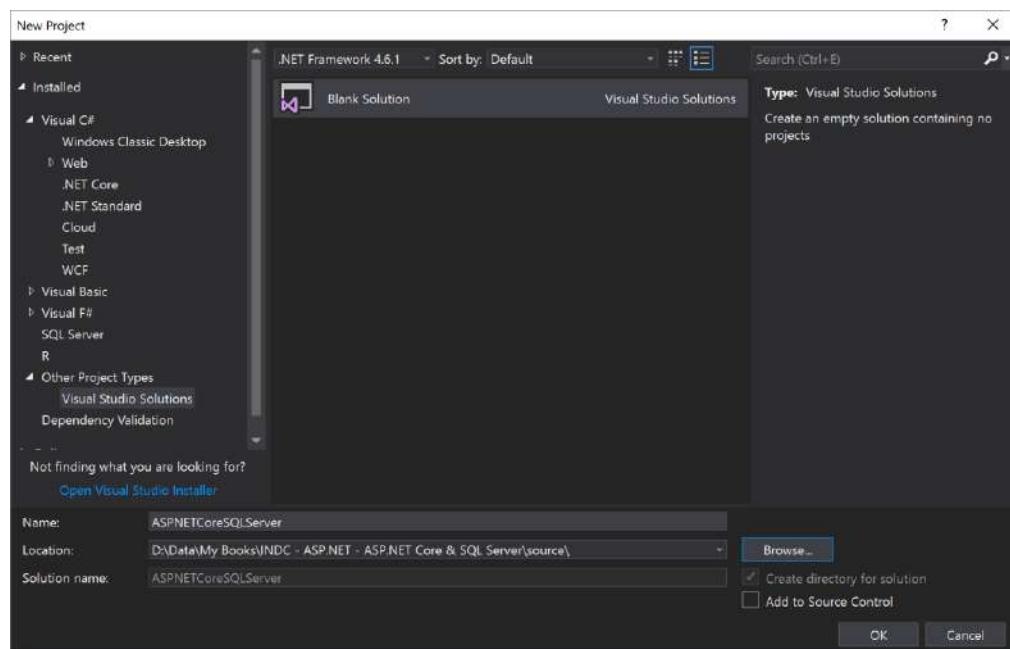
ASP.NET Core

ASP.NET Core Project

Pada sub bab ini akan dijelaskan tentang pembuatan project ASP.NET Core. Visual Studio 2017 memiliki beberapa template project aplikasi web yang bisa dipilih. Pada sub bab ini akan dijelaskan tentang folder dan file utama yang ada pada masing-masing template project tersebut.

Membuat Solution

Seluruh contoh project yang diberikan pada buku ini akan disimpan di dalam sebuah solution. Cara pembuatan solution telah dicontohkan pada bab sebelumnya. Dengan cara yang sama buat solution dengan nama ASPNETCoreSQLServer.



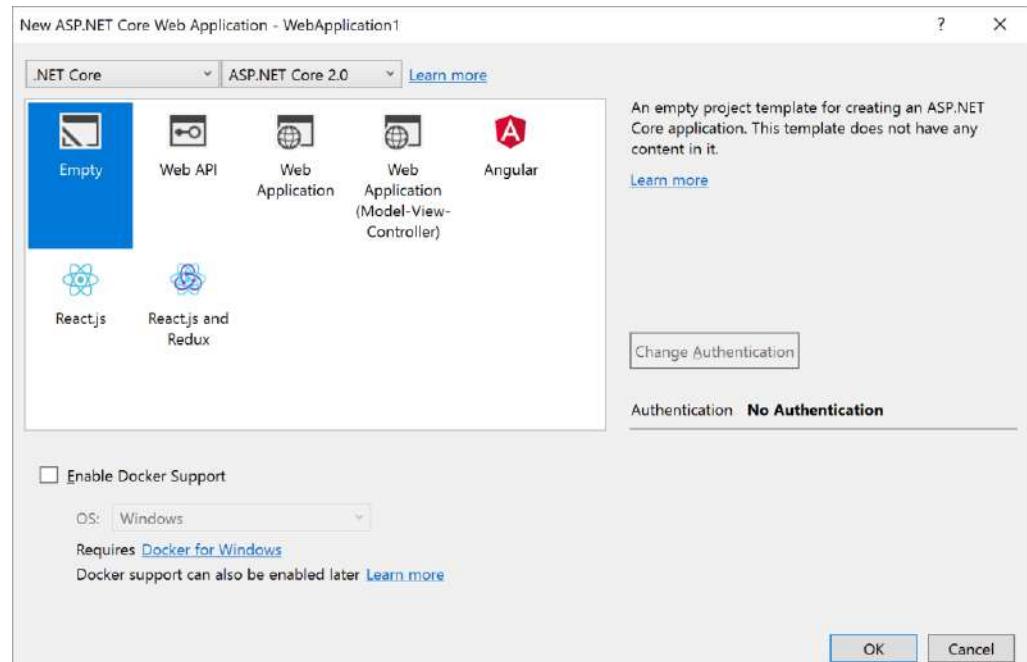
Gambar 55. Solution - ASPNETCoreSQLServer.

Membuat Project ASP.NET Core

Project ASP.NET Core memiliki beberapa template yang bisa digunakan, yaitu:

- Empty.
- Web API.
- Web Application.
- Web Application (Model-View-Controller).
- Angular.
- React.js.

- React.js and Redux.



Gambar 56. Daftar template project ASP.NET Core.

Pada buku ini hanya akan diperlihatkan pemanfaatan 3 template project saja, yaitu:

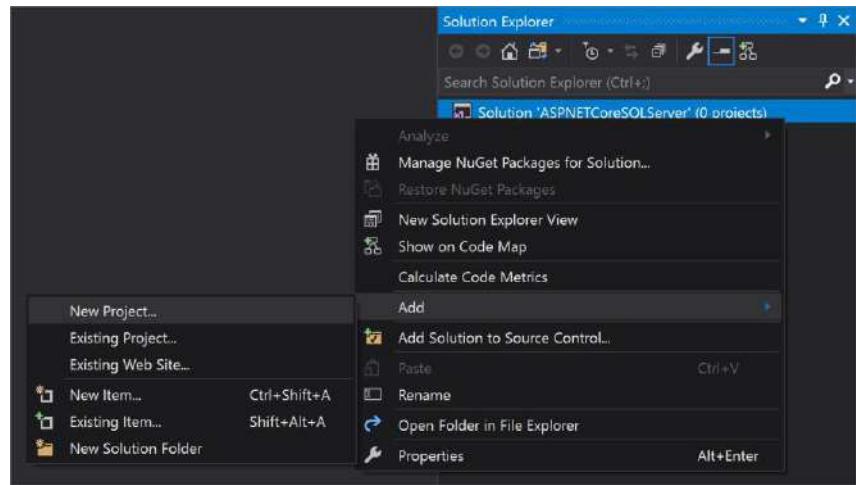
- Empty.
- Web Application.
- Web Application (Model-View-Controller).

Ketiga template tersebut merupakan template project aplikasi web yang berhubungan dengan topik yang dibahas pada buku ini. Sedangkan sisanya akan dibahas pada buku yang berbeda yang informasinya telah disebutkan pada bab 1 sub bab Bahan Pendukung.

Template Project: Empty

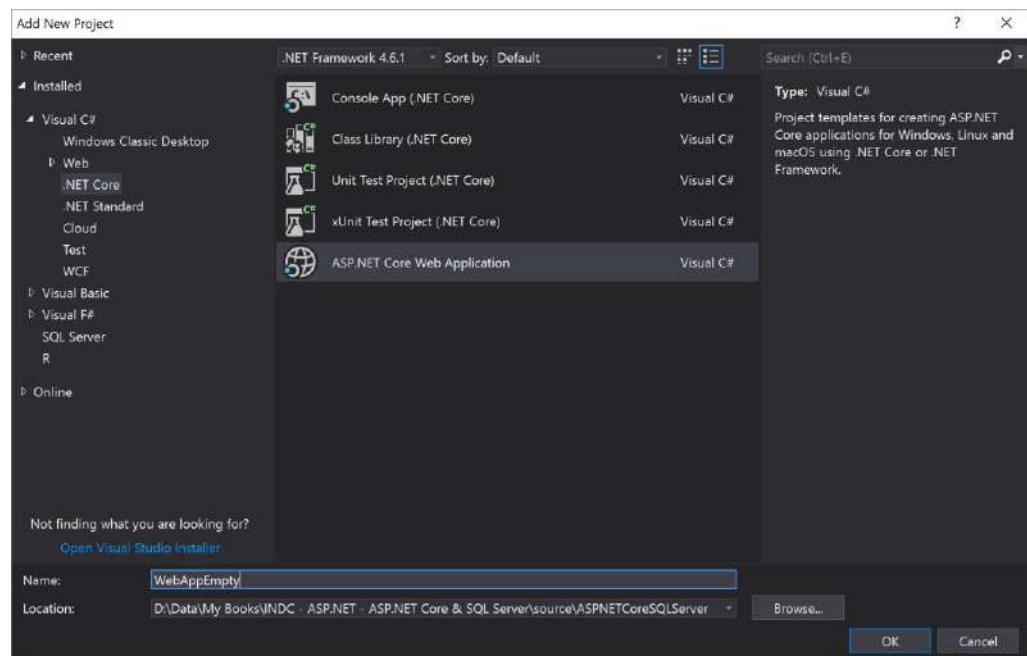
Untuk membuat project dengan template ini dapat dilakukan dengan langkah-langkah berikut ini.

Klik kanan pada Solution ASPNETCoreSQLServer yang sebelumnya telah dibuat di atas. Kemudian pilih Add > New Project.



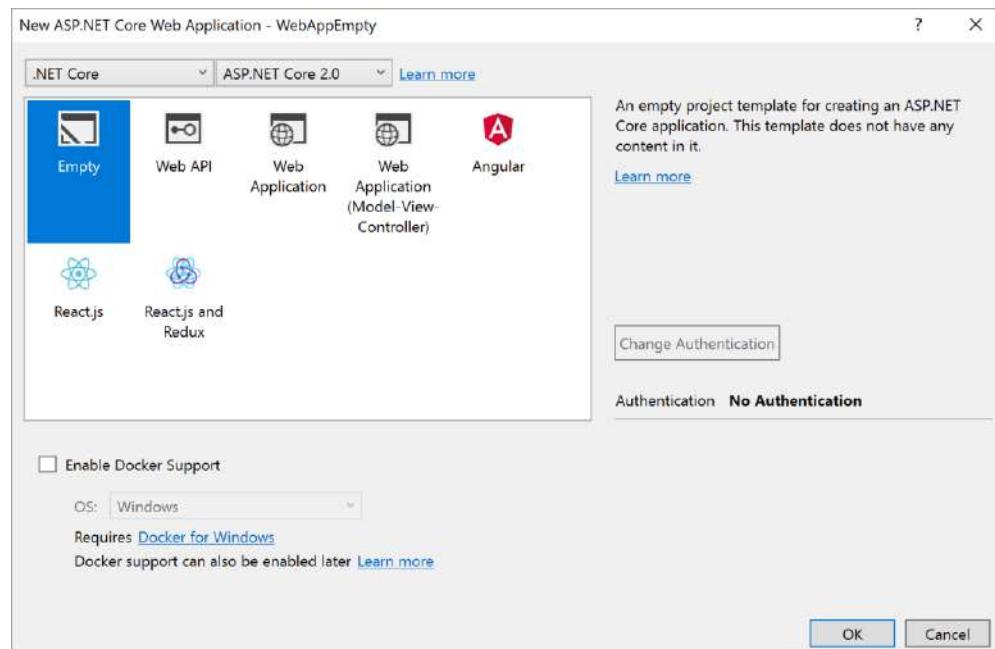
Gambar 57. Template Project: Empty - Add New Project.

Kemudian akan ditampilkan window Add New Project seperti pada gambar di bawah ini.



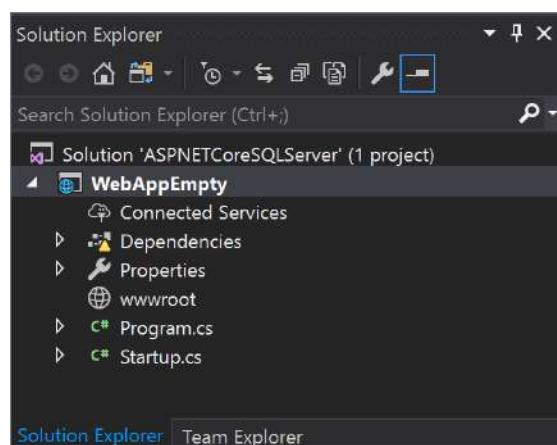
Gambar 58. Template Project: Empty - Add New Project.

Pilih Visual C# > .NET Core > ASP.NET Core Web Application. Kemudian isi kolom Name dengan "WebAppEmpty" sebagai nama project. Kemudian akan ditampilkan window New ASP.NET Core Web Application.



Gambar 59. Template Project: Empty - New ASP.NET Core Web Application.

Kemudian pilih Empty dan klik tombol OK. Hasilnya dapat dilihat pada Solution Explorer.



Gambar 60. Template Project: Empty - WebAppEmpty pada Solution Explorer.

Pada gambar di atas dapat dilihat folder wwwroot dan dua file utama yaitu:

- Startup.cs.
- Program.cs.

File Startup.cs menyimpan class Startup. Class Startup ini berfungsi untuk mengkonfigurasi saluran yang menangani seluruh request ke aplikasi. Setiap aplikasi ASP.NET, baik aplikasi ASP.NET klasik maupun ASP.NET Core, harus memiliki sebuah class Startup. Berikut adalah contoh isi dari file Startup.cs.

```
Startup.cs
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Builder;
using Microsoft.AspNetCore.Hosting;
using Microsoft.AspNetCore.Http;
using Microsoft.Extensions.DependencyInjection;
```

```

namespace WebAppEmpty
{
    public class Startup
    {
        // This method gets called by the runtime. Use this method to add
        // services to the container.
        // For more information on how to configure your application, visit
        https://go.microsoft.com/fwlink/?LinkID=398940
        public void ConfigureServices(IServiceCollection services)
        {

        }

        // This method gets called by the runtime. Use this method to
        // configure the HTTP request pipeline.
        public void Configure(IApplicationBuilder app, IHostingEnvironment
env)
        {
            if (env.IsDevelopment())
            {
                app.UseDeveloperExceptionPage();
            }

            app.Run(async (context) =>
            {
                await context.Response.WriteAsync("Hello World!");
            });
        }
    }
}

```

Pada class Startup terdapat dua method utama yaitu:

- Configure, method ini digunakan untuk menentukan cara aplikasi ASP.NET akan menangani dan menjawab request HTTP. Pada file di atas dapat dilihat jika ada request maka aplikasi akan memberikan jawaban berupa tulisan “Hello World!”. Untuk membuktikan hal ini, pembaca dapat menjalankan aplikasi BelajarASPNETCoreMVC dengan cara yang telah dijelaskan pada bab 3.
- ConfigureServices, method ini digunakan untuk menambahkan service ke container. Method ini dipanggil sebelum method Configure.

File Program.cs umumnya ditemui pada aplikasi standalone baik pada aplikasi console ataupun windows form. File ini tidak akan ditemui pada aplikasi ASP.NET klasik. File ini ada pada aplikasi ASP.NET Core karena aplikasi ini adalah aplikasi standalone console. Berikut ini adalah isi file Program.cs.

Program.cs
<pre> using System; using System.Collections.Generic; using System.IO; using System.Linq; using System.Threading.Tasks; using Microsoft.AspNetCore; using Microsoft.AspNetCore.Hosting; using Microsoft.Extensions.Configuration; using Microsoft.Extensions.Logging; namespace WebAppEmpty { public class Program { public static void Main(string[] args) { } } } </pre>

```

    {
        BuildWebHost(args).Run();
    }

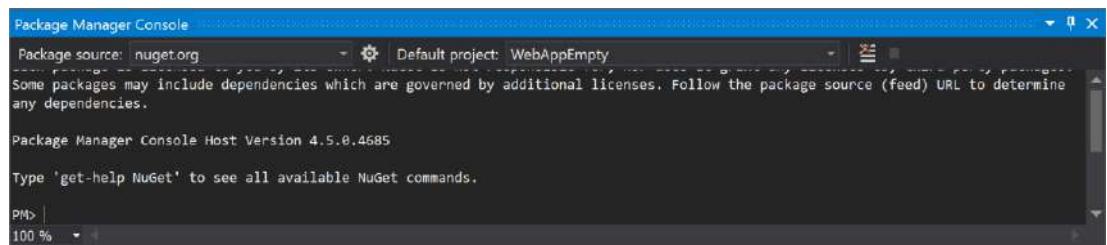
    public static IWebHost BuildWebHost(string[] args) =>
        WebHost.CreateDefaultBuilder(args)
            .UseStartup<Startup>()
            .Build();
}
}

```

Pada aplikasi standalone pasti terdapat file Program.cs yang didalamnya terdapat class Program dengan sebuah method Main. Method Main adalah utama yang akan dipanggil ketika aplikasi standalone dijalankan. Isi method Main pada file di atas berisi kode untuk melakukan proses hosting aplikasi menjalankan web server. Web server yang umumnya digunakan oleh ASP.NET Core adalah Kestrel.

Setelah project dibuat, maka langkah selanjutnya adalah melakukan proses restore yang bertujuan untuk memuat library-library yang sesuai dengan template project yang digunakan. Untuk melakukan proses restore dapat dilakukan dengan menuliskan perintah pada Package Manager Console. Console ini dapat diaktifkan dengan cara memilih Tools > NuGet Package Manager > Package Manager Console pada menu.

Hasilnya akan dapat dilihat pada bagian bawah antarmuka Visual Studio seperti pada gambar di bawah ini.



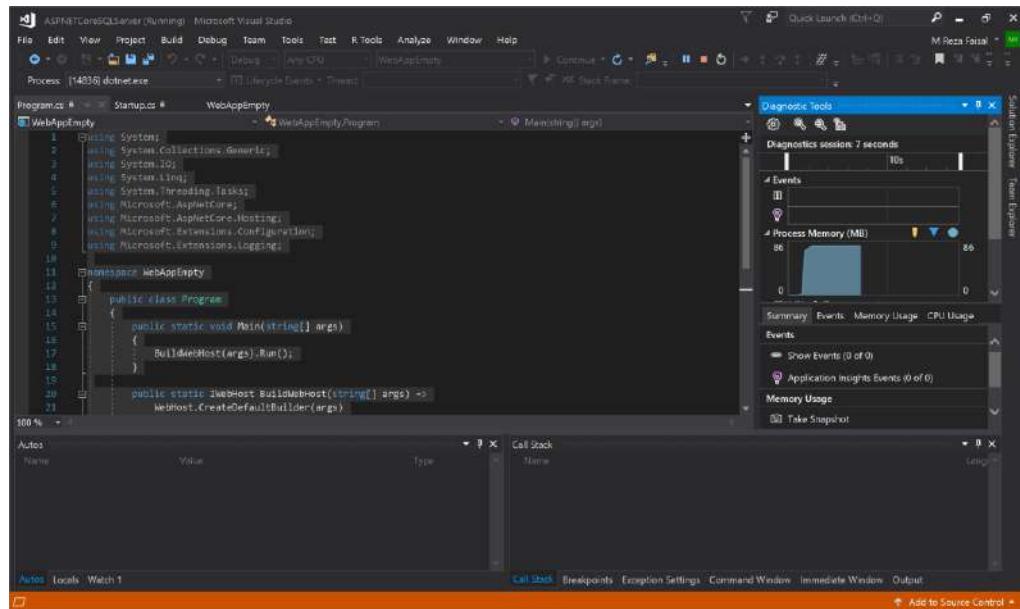
Gambar 61. Template Project: Empty - Package Manager Console.

Untuk proses restore ketikkan gunakan command “dotnet” seperti yang telah diterangkan pada bab sebelumnya. Berikut adalah perintah untuk melakukan proses restore.

```
dotnet restore
```

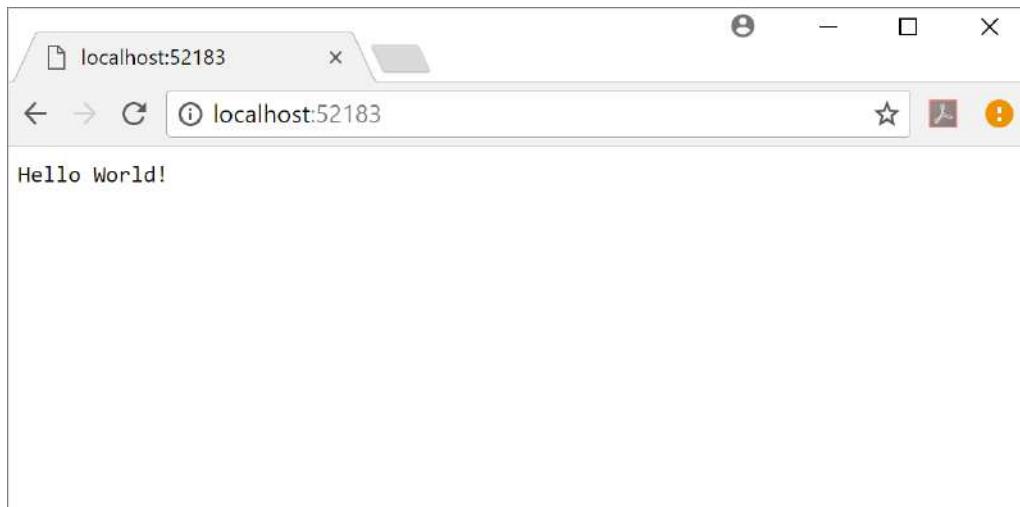
Jika proses berhasil, maka untuk melakukan proses build dan menjalankan project dapat dilakukan dengan cara klik kanan pada project yang ada di Solution Explorer. Kemudian pilih Debug > Start new instance.

Saat proses debug akan terjadi perubahan antarmuka pada Visual Studio 2017 seperti pada gambar berikut.



Gambar 62. Template Project: Empty - Diagnostic Tools.

Dan secara otomatis akan dijalankan web broser yang menampilkan aplikasi web dari project yang dijalankan.

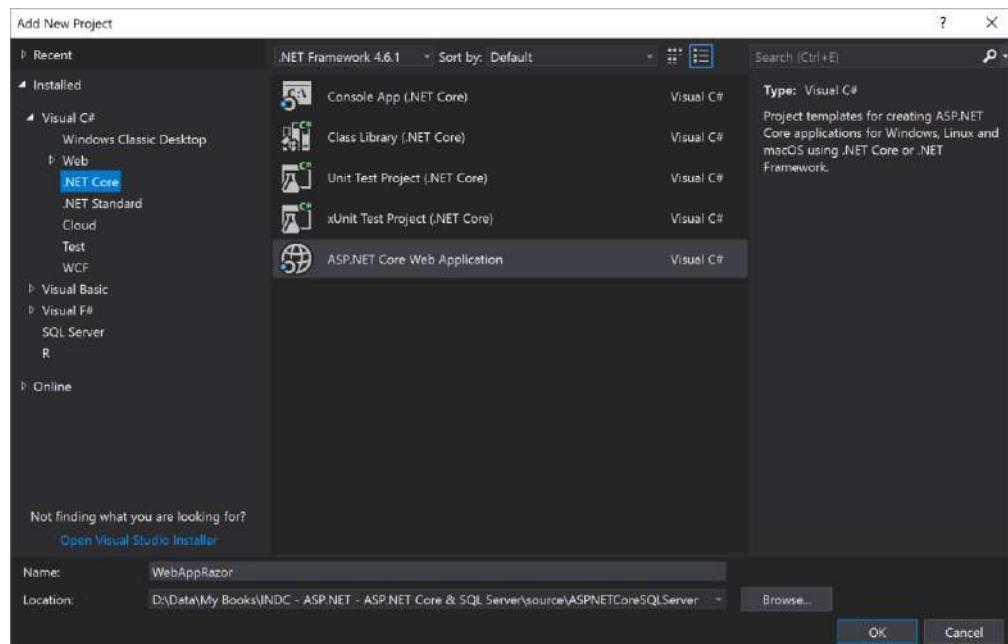


Gambar 63. Template Project: Empty - WebAppEmpty pada web browser.

Template Project: Web Application

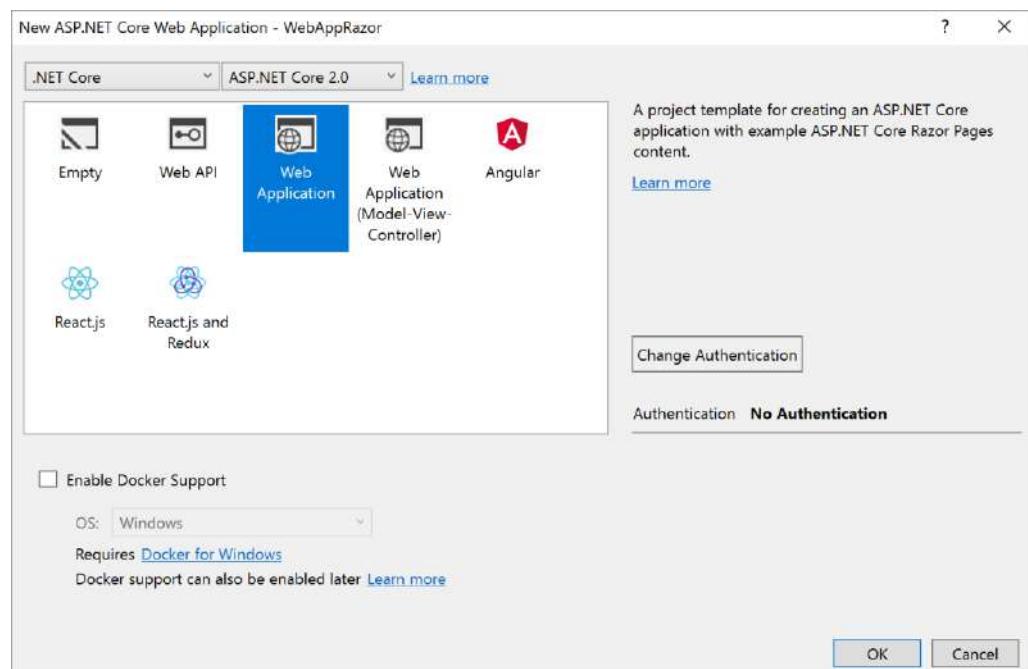
Pada sub bab ini akan dibuat project dengan menggunakan template Web Application. Pembuatan project dilakukan dengan cara yang sama seperti yang telah dijelaskan pada bagian sebelumnya.

Untuk nama project digunakan nama WebAppRazor.



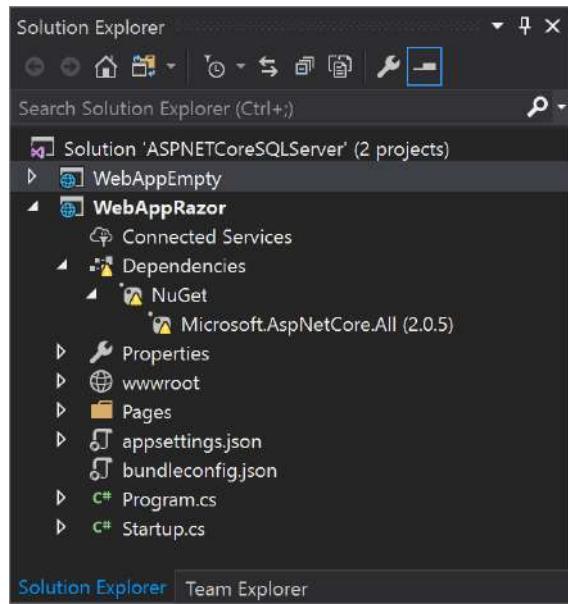
Gambar 64. Template Project: Web Application - Add New Project.

Setelah window New ASP.NET Core Web Application ditampilkan pilih Web Application kemudian klik tombol OK.



Gambar 65. Template Project: Web Application - New ASP.NET Core Web Application.

Selanjutnya dapat dilihat project WebAppRazor pada Solution Explorer.



Gambar 66. Template Project: Web Application - WebAppEmpty pada Solution Explorer.

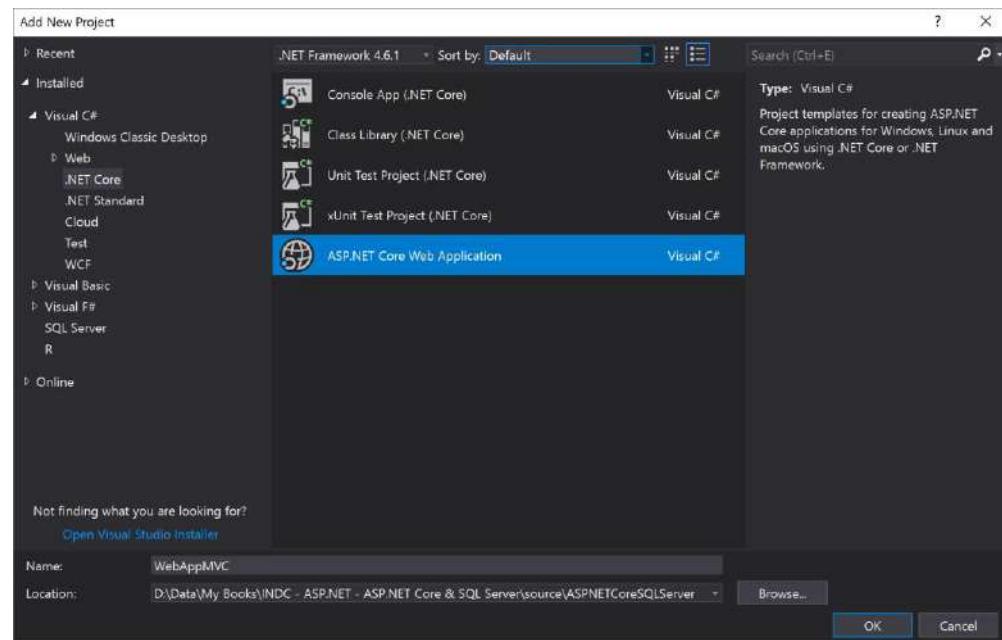
Selanjutnya lakukan kembali proses restore project dengan perintah “dotnet” seperti yang telah diterangkan pada sub bab sebelumnya.

Dari gambar di atas terdapat tambahan sebuah folder dan dua file, yaitu:

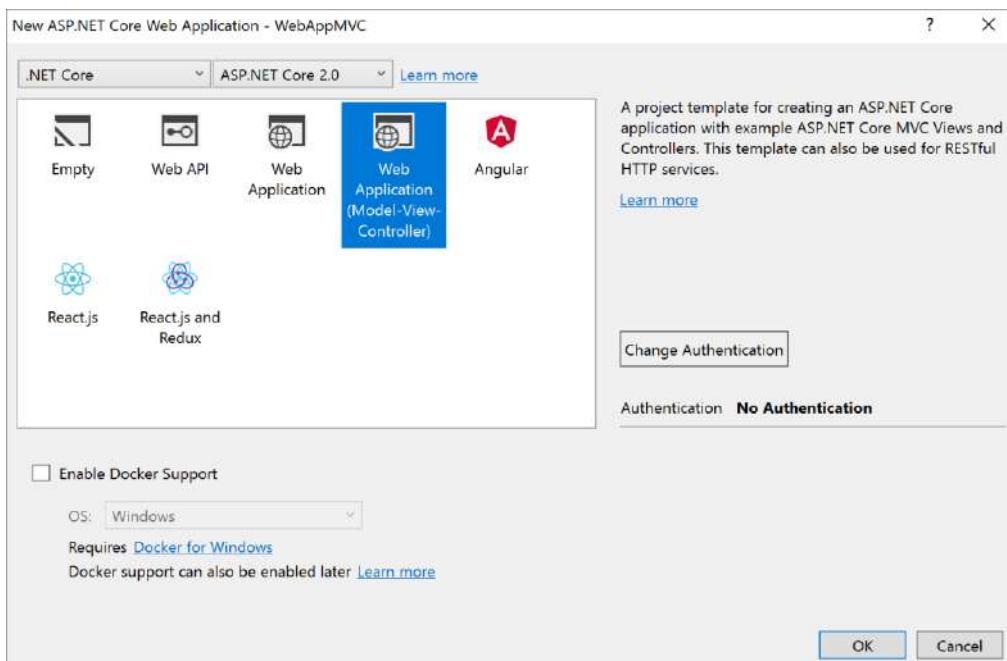
- Folder Pages, folder ini berisi file *.cshtml. File ini adalah file Razor. Pembahasan tentang Razor akan diberikan pada bab selanjutnya.
- File appsetting.json, file ini berfungsi untuk menyimpan setting aplikasi seperti konfigurasi untuk koneksi ke database dan lain-lain.
- File bundleconfig.json, file ini berfungsi untuk menentukan cara untuk melakukan proses bundling dan minification terhadap project.

Template Project: Web Application (Model-View-Controller)

Yang terakhir adalah membuat project dengan menggunakan template Web Application (Model-View-Controller). Nama project yang akan dibuat ini adalah WebAppMVC.

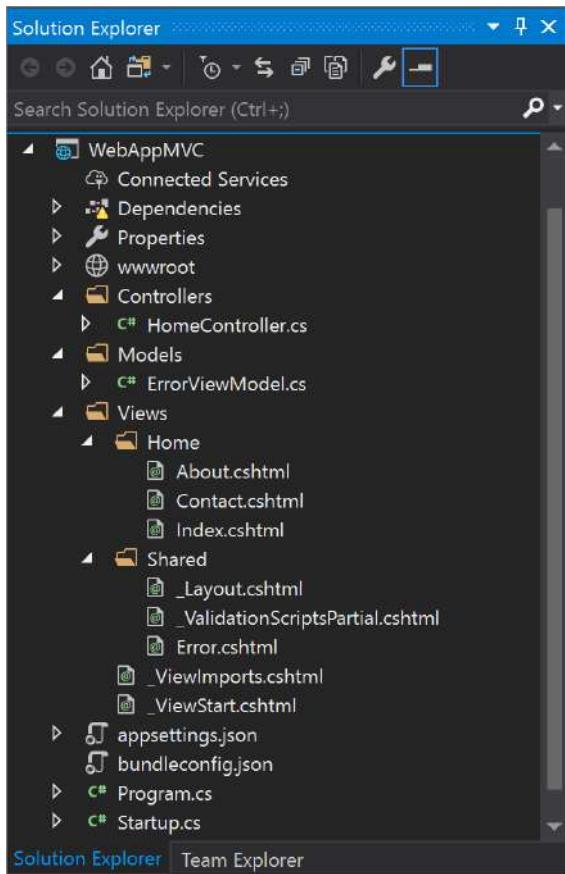


Gambar 67. Template Project: Web Application (MVC) - Add New Project.



Gambar 68. Template Project: Web Application (MVC) - New ASP.NET Core Web Application.

Pada window New ASP.NET Web Application pilih template Web Application (Model-View-Controller).



Gambar 69. Template Project: Web Application (MVC) - WebAppEmpty pada Solution Explorer.

Hasilnya dapat dilihat pada Solution Explorer. Pada gambar di atas dapat dilihat bahwa project WebAppMVC memiliki folder dan file tambahan jika dibandingkan dengan dengan dua project sebelumnya. Folder yang ditambahkan adalah:

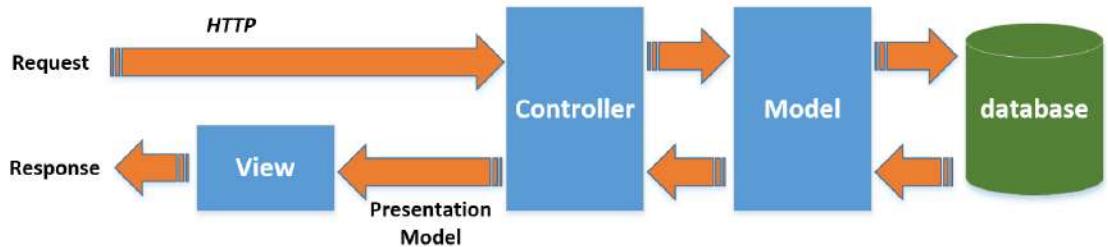
- Model, folder ini menyimpan komponen Model
- View, folder ini menyimpan komponen View.
- Controller, folder ini menyimpan komponen Controller.

Ketiga folder tersebut adalah folder penting pada framework Model-View-Controller. Detail tentang framework ini akan dibahas pada bab selanjutnya.

Cara Kerja ASP.NET Core MVC

Cara kerja ASP.NET Core MVC sama seperti cara kerja ASP.NET MVC versi sebelumnya. Keduanya menggunakan pattern Pattern MVC. Pattern ini termasuk Architectural Pattern. Sedangkan software design pattern yang digunakan pada ASP.NET MVC Core dan ASP.NET MVC adalah Front Controller, artinya kontrol akan bersifat terpusat (center controller) pada sebuah class saja.

Cara kerja MVC dapat dilihat pada gambar berikut ini.

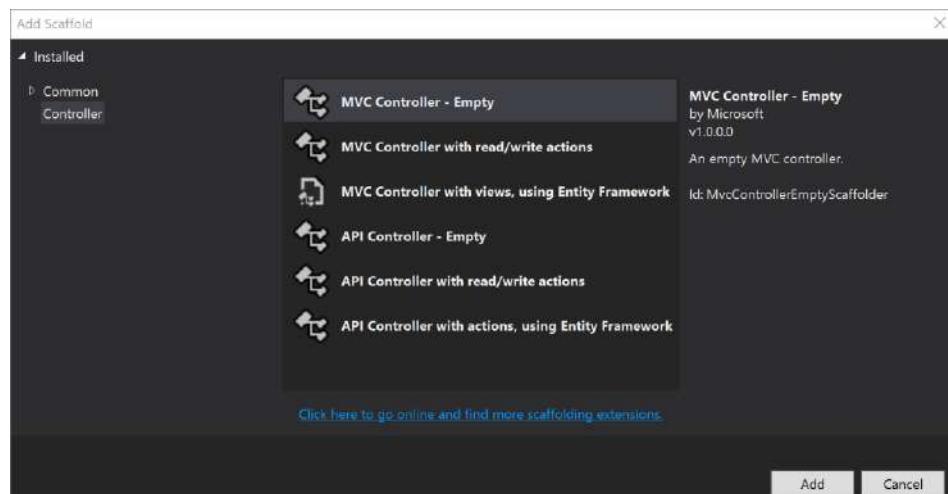


Gambar 70. Cara kerja Pattern MVC.

Untuk menjelaskan cara kerja pattern MVC dan fungsi-fungsi folder ASP.NET Core MVC yang telah disebutkan di atas, maka pada sub bab ini akan diberikan contoh project sederhana. Project ini melanjutnya project WebAppMVC yang telah dibuat sebelumnya.

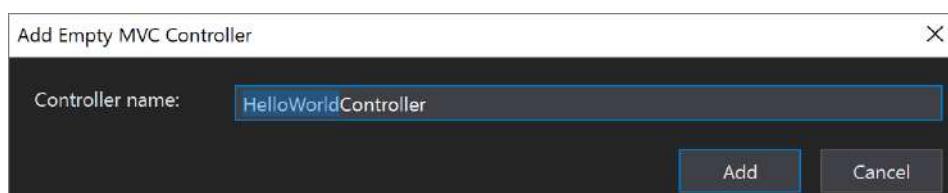
Controller

Langkah pertama adalah membuat komponen controller. Komponen controller disimpan pada folder Controllers. Untuk membuat class controller, klik kanan pada folder Controller di project WebAppMVC, kemudian Add > Controller. Maka akan ditampilkan window Add Scaffold seperti pada gambar di bawah ini.



Gambar 71. Controller - Menambah class controller.

Pilih MVC Controller - Empty, kemudian klik tombol Add. Kemudian pada window Add Empty MVC Controller, berikan nama pada kolom Controller name. Sebagai latihan berikan nama HelloWorldController. Kemudian klik tombol Add.



Gambar 72. Controller - HelloWorldController.

Penamaan nama class Controller harus diakhiri dengan Controller, sebagai contoh:

- HelloWorldController.
- TambahController.
- MahasiswaController.

Dan berikut ini adalah isi dari class HelloWorldController.

```

HelloWorldController.cs
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Mvc;

namespace WebAppMVC.Controllers
{
    public class HelloWorldController : Controller
    {
        public IActionResult Index()
        {
            return View();
        }
    }
}

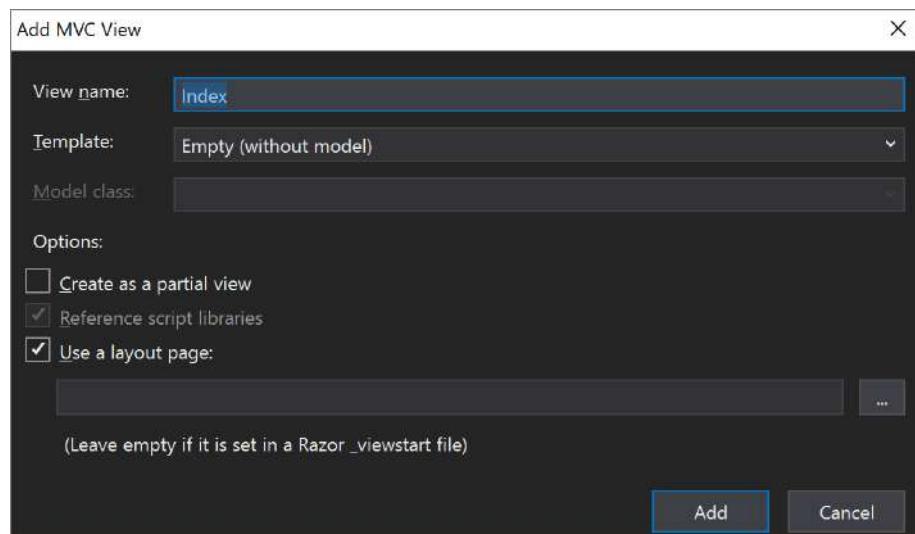
```

Dari kode di atas dapat dilihat bahwa class HelloWorldController merupakan turunan dari class Controller. Hal ini juga akan berlaku untuk semua class controller yang akan dibuat nanti. Di dalam class HelloWorldController dapat dilihat sebuah method Index yang merupakan implementasi dari IActionResult. Di dalam method Index dapat dilihat pemanggilan method View. Method ini berfungsi untuk menampilkan komponen View. Nama file komponen View yang dipanggil oleh method Index ini adalah Index.cshtml

View

Langkah selanjutnya adalah membuat komponen View. Komponen View disimpan di dalam folder Views. Selanjutnya buat folder HelloWorld di dalam folder Views. Nama folder HelloWorld ini disesuaikan dengan nama class controller yaitu HelloWorldController. Jika nama class controller adalah StudentController, maka harus dibuat folder Student

Untuk setiap method action pada class controller harus dibuat sebuah file *.cshtml dengan nama yang sesuai dengan nama method action tersebut. Sebagai contoh jika pada class HelloWorldController dimiliki satu method action dengan nama Index, maka perlu dibuat file Index.cshtml di dalam folder Views\HelloWorld.



Gambar 73. View - Add MVC View.

Untuk membuat file Index.cshtml, klik kanan pada folder Views\HelloWorld kemudian pilih Add > View. Kemudian akan ditampilkan window Add MVC View seperti gambar di atas. Ketik nama file pada kolom View name. Nama file sesuai dengan nama method action yaitu Index. Kemudian klik tombol Add.

Berikut adalah isi dari file Index.cshtml.

```
Index.cshtml
```

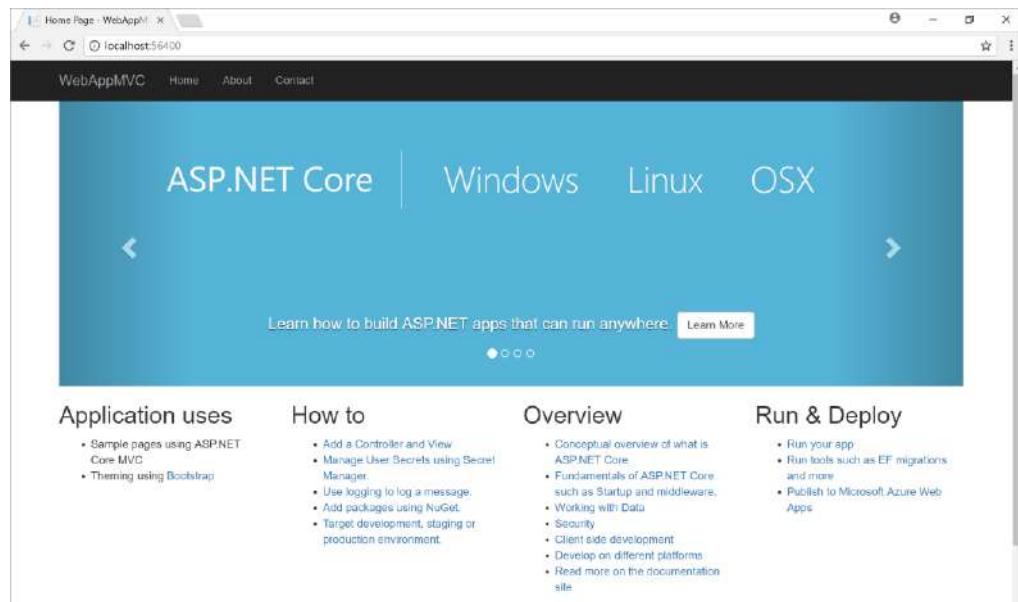
```
@{
    ViewData["Title"] = "Index";
}

<h2>Index</h2>

Hello World!
```

Pada file tersebut dapat ditambahkan "Hello World!". Untuk melihat halaman ini dapat dilakukan dengan menjalankan aplikasi ini, yaitu klik kanan pada project WebAppMVC kemudian pilih Debug > Start new instance.

Berikut adalah halaman yang ditampilkan.



Gambar 74. WebAppMVC - Home.

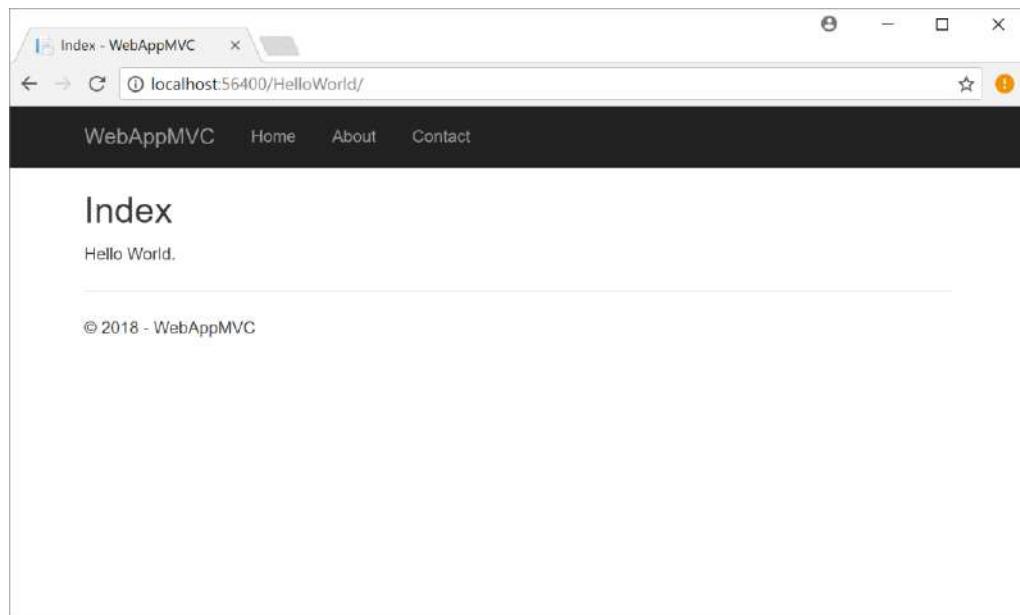
Sedangkan jika ingin mengakses halaman HelloWorld yang telah dibuat, maka dapat diakses dengan menggunakan pola alamat berikut ini.

```
http://[nama host/ip address]:port/HelloWorld
```

Karena alamat halaman utama dari WebAppMVC adalah <http://localhost:56400> maka alamat yang ditulis untuk mengakses halaman HelloWorld adalah sebagai berikut.

```
http://localhost:56400/HelloWorld
```

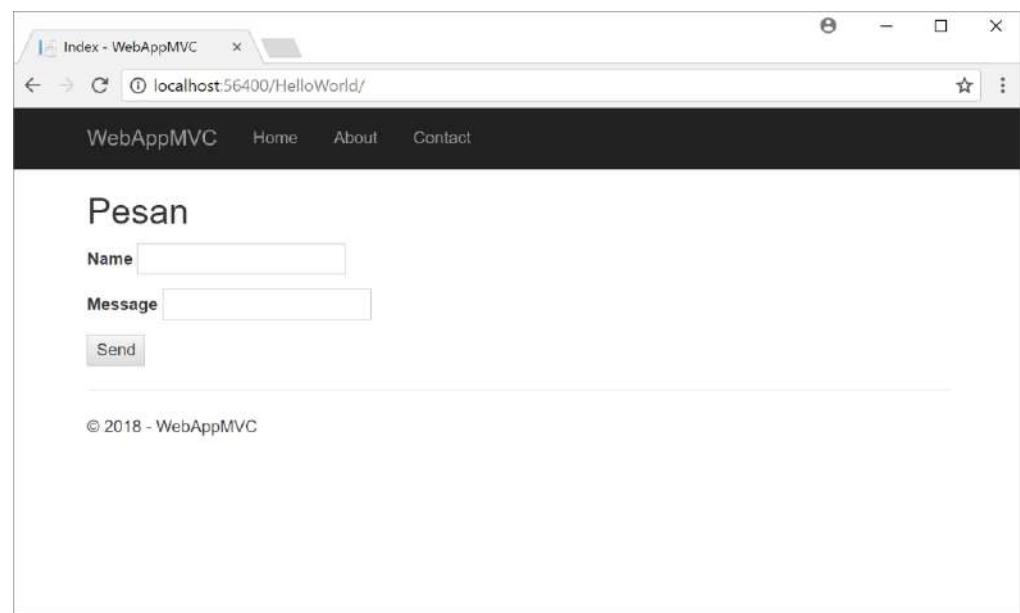
Hasilnya bisa dilihat pada gambar di bawah ini.



Gambar 75. WebAppMVC - HelloWorld.

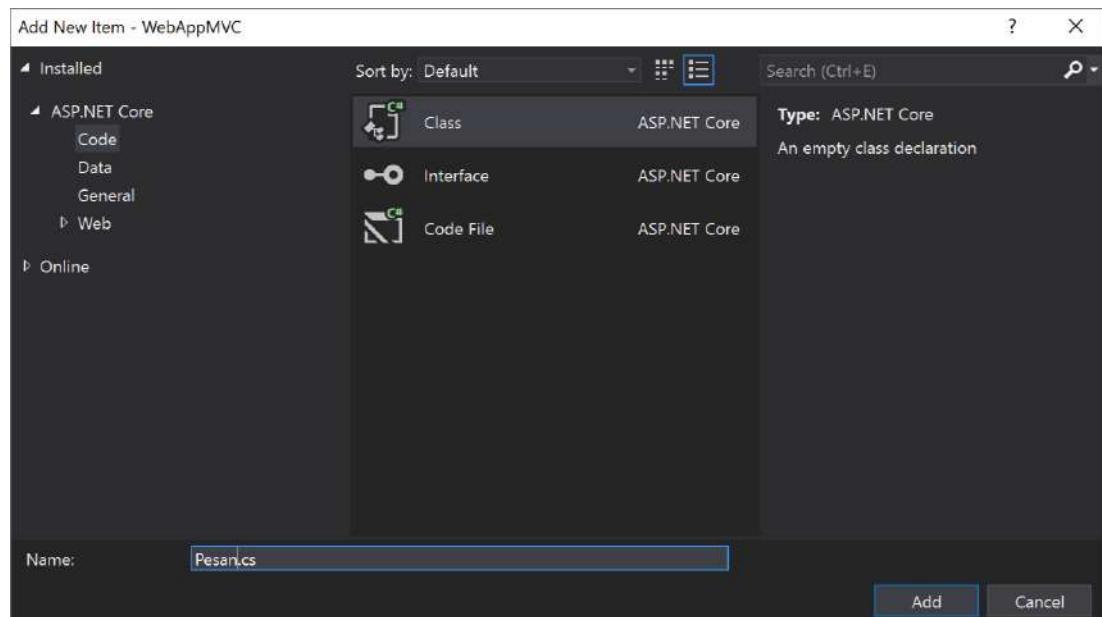
Model

Pada bagian ini akan diperkenalkan tentang komponen model. Penjelasan tentang komponen ini dalam bentuk contoh kasus yaitu dengan membuat halaman aplikasi web sederhana dengan tampilan seperti pada gambar di bawah ini. Fungsi halaman web ini hanya untuk mengirimkan nama, email dan pesan. Kemudian nilai-nilai yang diisi itu akan ditampilkan kembali pada halaman web.



Gambar 76. Antarmuka form pada halaman HelloWorld.

Komponen model akan disimpan di dalam folder Models. Untuk menambahkan komponen ini, klik kanan pada folder Model kemudian pilih Add > Class.



Gambar 77. Model - Add New Item.

Pada window Add New Item, pilih Code > Class. Kemudian berikan nama class pada kolom Name. Nama class yang dibuat adalah Pesan.cs dengan isi sebagai berikut.

```
Pesan.cs
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;

namespace WebAppMVC.Models
{
    public class Pesan
    {
        public String Name { set; get; }
        public String Message { set; get; }

    }
}
```

Selanjutnya akan dimodifikasi kode file Index.cshtml untuk membuat form seperti pada gambar di atas. Berikut ini adalah kode yang digunakan.

```
Index.cshtml
@model WebAppMVC.Models.Pesan

 @{
    ViewData["Title"] = "Index";
}



## Pesan



@using (Html.BeginForm())
    {
        <p>
            @Html.LabelFor(m => m.Name)
            @Html.TextBoxFor(m => m.Name)
        </p>
    }


```

```

<p>
    @Html.LabelFor(m => m.Message)
    @Html.TextBoxFor(m => m.Message)
</p>
<p>
    @ViewBag.Pesan
</p>
<p>
    <input type="submit" value="Send" />
</p>
}
</div>

```

Pada kode di atas dapat dilihat cara pembuatan form dengan menggunakan sintaks Razor. Pada sub bab selanjutnya akan diberikan penjelasan detail tentang sintaks Razor. Hal penting pada kode di atas adalah bagaimana cara menentukan komponen model yang digunakan pada komponen view. Dapat dilihat komponen model ditulis dengan menulis lengkap nama class berserta namespace.

```
@model WebAppMVC.Models.Pesan
```

Setelah langkah di atas perlu dilakukan modifikasi pada class controller untuk menangani aksi ketika tombol Send ditekan. Berikut ini adalah hasil modifikasi pada class controller HelloWorldController.cs.

```

HelloWorldController.cs
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Mvc;

using WebAppMVC.Models;

namespace WebAppMVC.Controllers
{
    public class HelloWorldController : Controller
    {
        [HttpGet]
        public IActionResult Index()
        {
            return View();
        }

        [HttpPost]
        public IActionResult Index(Pesan data)
        {
            ViewBag.Pesan = data.Name + " mengatakan " + data.Message;
            return View();
        }
    }
}

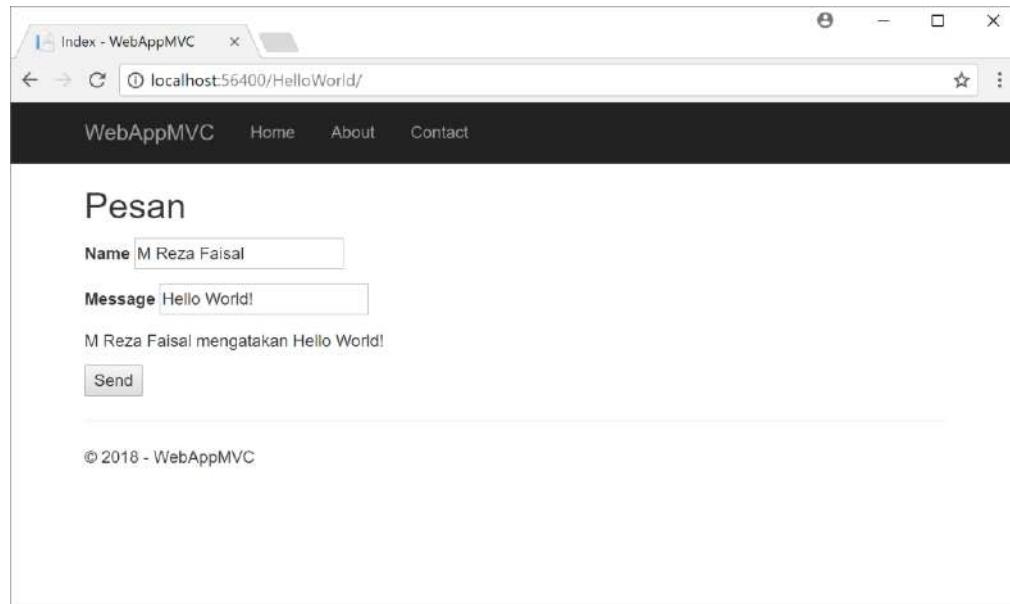
```

Pada komponen controller ini juga harus dipanggil komponen model dengan cara sebagai berikut.

```
using WebAppMVC.Models;
```

Pada file class `HelloWorldController.cs` dapat dilihat terdapat dua method action `Index`. Method action `Index` yang pertama menggunakan method GET, artinya method akan memberikan respon saat diakses dengan method GET. Sedangkan ketika diakses dengan method POST, sebagai contoh saat tombol `Send` diklik, maka method action `Index` method POST yang akan digunakan.

Berikut adalah tampilan halaman setelah tombol `Send` diklik.

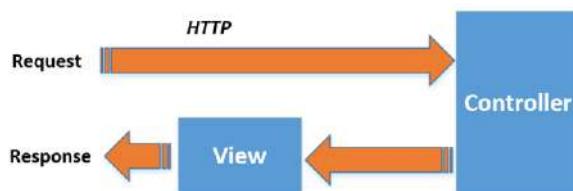


Gambar 78. Hasil ketika form diisi dan tombol `Send` diklik.

Catatan

Berdasarkan dari gambar cara kerja pattern MVC di awal sub bab ini dan penjelasan implementasi pada ASP.NET Core maka ada dua hal penting yang mesti dicatat, yaitu:

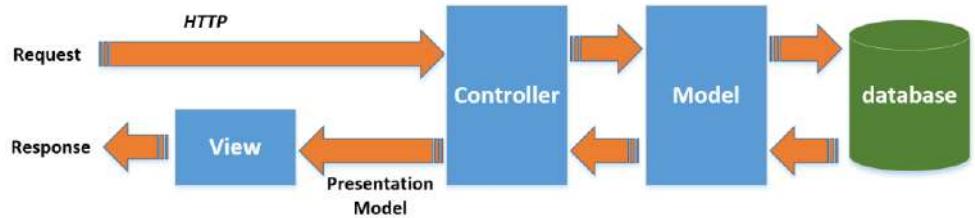
1. Untuk membuat web dengan halaman statik tanpa ada interaksi seperti form atau penggunaan data maka cukup digunakan dua komponen saja yaitu Controller dan View.



Gambar 79. Catatan 1 tentang ASP.NET Core MVC.

Untuk kasus ini maka komponen yang mesti disiapkan terlebih dahulu adalah komponen controller kemudian komponen view.

2. Untuk membuat web dengan yang memiliki fungsi interaksi seperti form atau penggunaan data maka harus digunakan tiga komponen MVC.



Gambar 80. Catatan 2 tentang ASP.NET Core MVC.

Untuk kasus ini maka komponen yang sebaiknya disiapkan atau dibuat terlebih dahulu adalah komponen model, komponen controller kemudian komponen view.

3. Catatan lain adalah hal penting terkait cara proses build dan run aplikasi ASP.NET Core ini. Pastikan jika melakukan modifikasi file selain komponen View, maka harus dilakukan proses build agar dihasilkan file *.dll terbaru kemudian dilakukan proses run kembali, agar aplikasi yang ditampilkan menggunakan hasil file *.dll yang terbaru. Tetapi jika yang dimodifikasi adalah file pada komponen view, seperti file *.cshtml maka untuk hasil modifikasi dapat langsung dilihat dengan cara melakukan proses refresh pada Web Browser saja.

5

Entity Framework Core & MS SQL Server

Pada bab ini akan dijelaskan bagaimana cara melakukan koneksi dan operasi ke database MS SQL Server dari aplikasi web yang dibangun dengan framework ASP.NET Core dan bahasa pemrograman C#.

Pendahuluan

Entity Framework adalah framework untuk mempermudah mengakses database. Framework ini awalnya dibangun sebagai bagian dari .NET framework yang hanya dapat digunakan pada platform Microsoft. Tetapi dengan dikembangkannya .NET Core yang bersifat multiplatform, maka Entity Framework Core juga dapat digunakan pada berbagai platform.

Entity Framework Core atau disingkat EF Core adalah object-relational mapper (OR/M) yang memungkinkan software developer dapat bekerja dengan database dengan object .NET. Hal ini mengurangi kode program untuk mengakses database, karena digantikan oleh class dan method yang telah disediakan oleh framework ini.

EF Core mendukung berbagai macam database, tetapi tergantung ketersediaan provider database. Saat buku ini ditulis telah tersedia provider database sebagai berikut:

1. MS SQL Server.
2. MS SQL Server Compact Edition.
3. SQLite.
4. MySQL, tersedia tiga provider untuk database MySQL yaitu:
 - o MySQL Official., provider
 - o MySQL Pomelo.
 - o MySQL Sapient Guardian.
5. PostgreSQL.
6. Oracle dan lain-lain.

Tetapi tidak semua provider yang disebutkan diatas adalah gratis, ada beberapa provider database yang bersifat berbayar. Untuk mendapatkan update informasi terbaru tentang provider database dapat mengunjungi alamat berikut <https://docs.microsoft.com/en-us/ef/core/providers/>.

EF Core mendukung dua pendekatan dalam mengembangkan aplikasi, yaitu:

1. Database First, pendekatan ini umum dilakukan dimana database dan tabel-tabel di dalamnya telah terlebih dahulu dibuat. Kemudian dibuat class model berdasarkan tabel-tabel di dalam database tersebut
2. Code First, pada pendekatan ini yang dibuat terlebih dahulu adalah class-class model kemudian tabel-tabel pada database akan secara otomatis dibuat saat pertama kali aplikasi web dijalankan.

Pada bab ini akan dijelaskan contoh implementasi EF Core dengan kedua pendekatan tersebut.

Aplikasi GuestBook

Implementasi EF Core dengan kedua pendekatan di atas akan dijelaskan dengan memberikan contoh pembuatan aplikasi yang sama yaitu GuestBook. Tujuannya adalah agar dapat dilihat perbedaan antara pendekatan Database First dan Code First.

Aplikasi ini adalah aplikasi sederhana yang hanya mempunyai fitur-fitur sebagai berikut:

1. Menampilkan daftar buku tamu.
2. Mengisi buku tamu.

Database First

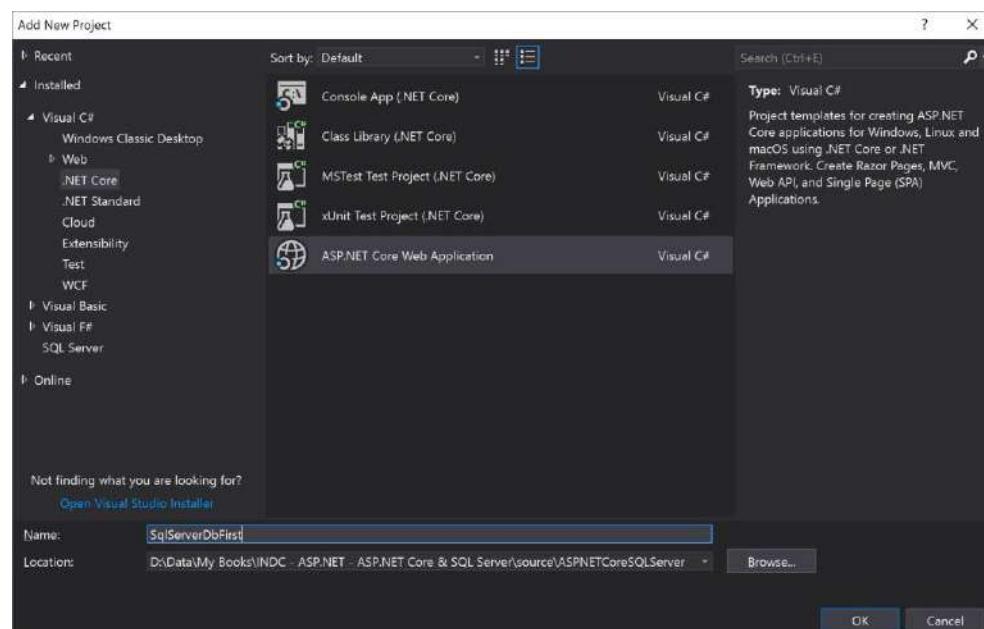
Pada sub bab ini akan diberikan langkah-langkah implementasi Entity Framework Code dengan pendekatan Database First.

Project

Pada sub bab ini akan dilakukan pembuatan project dan setting project.

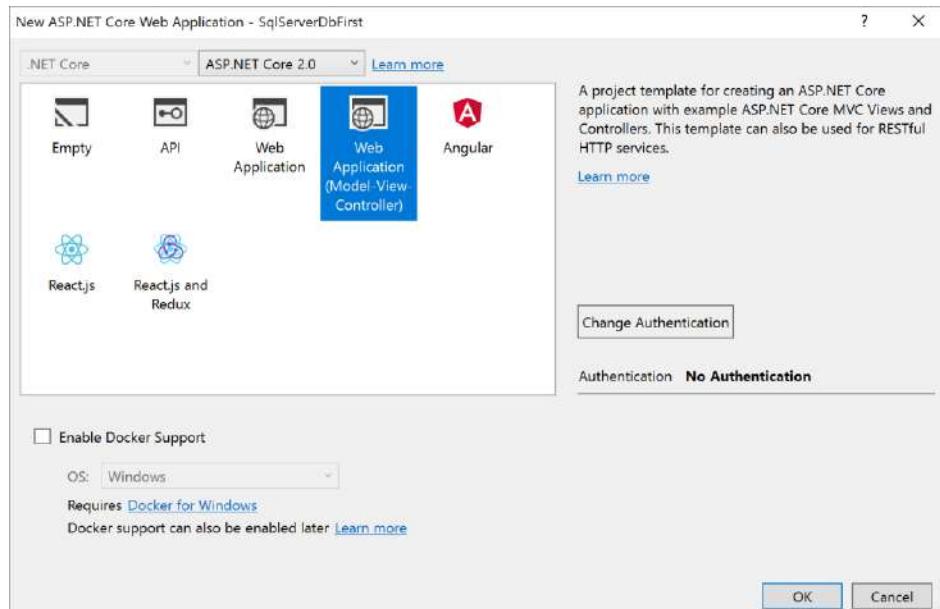
Membuat Project

Langkah pertama adalah membuat project dengan nama SqlServerDbFirst. Untuk membuat project SqlServerDbFirst gunakan template ASP.NET Core Web Application. Dan ikuti langkah-langkah membuat project ini seperti yang dijelaskan di bab sebelumnya.



Gambar 81. SqlServerDbFirst - Add New Project.

Template web yang digunakan adalah Web Application (Model-View-Controller).



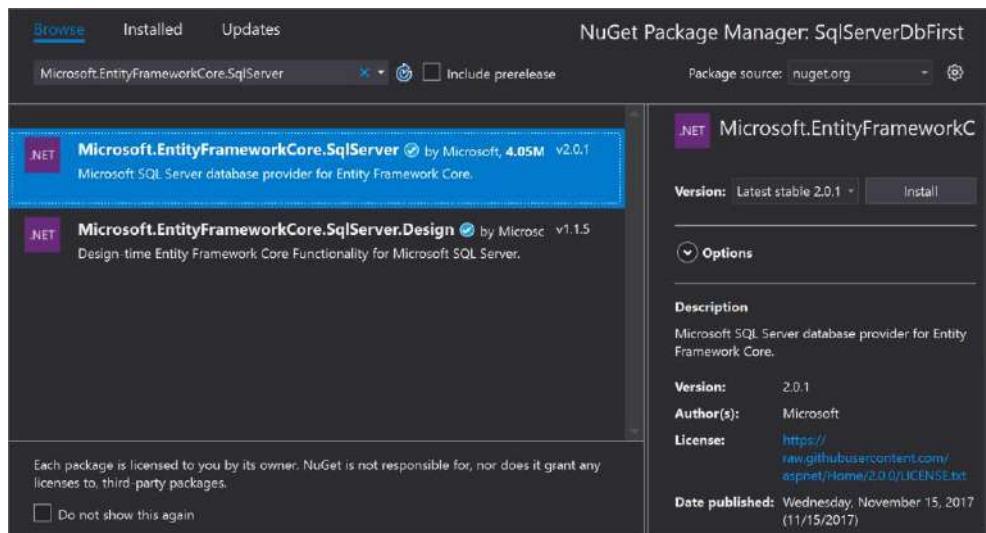
Gambar 82. SqlServerDbFirst - New ASP.NET Core Web Application.

Menyiapkan Library

Pada sub bab ini akan dilakukan persiapan project agar project dapat digunakan untuk melakukan koneksi ke database MS SQL. Library-library yang akan ditambahkan adalah sebagai berikut:

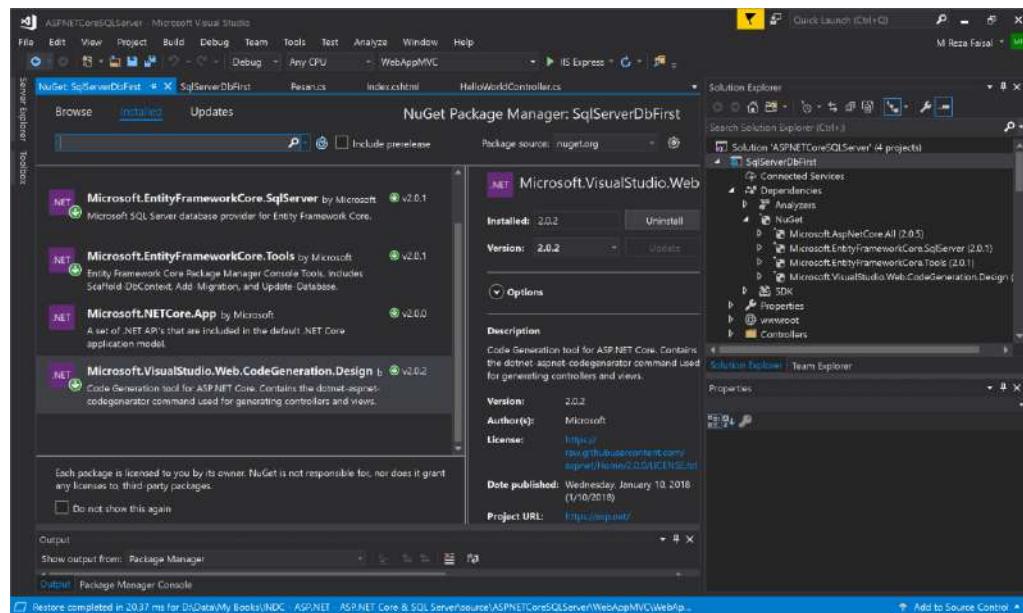
- Microsoft.EntityFrameworkCore.SqlServer, adalah library EF Core untuk melakukan koneksi ke database MS SQL Server.
- Microsoft.EntityFrameworkCore.Tools, adalah tool untuk membuat model dari database.
- Microsoft.VisualStudio.Web.CodeGeneration.Design, adalah tool scaffolding untuk membuat komponen controller dan view.

Untuk menginstall ketiga library ini dapat dapat menggunakan NuGet Package Manager, kemudian cari ketiga library tersebut.



Gambar 83. SqlServerDbFirst - Menambah library.

Untuk memeriksa library yang sudah diinstall, dapat dilihat pada tab Installed yang ada pada NuGet Package Manager atau pada Solution Explorer pada bagian Dependencies > NuGet seperti pada gambar di bawah ini.

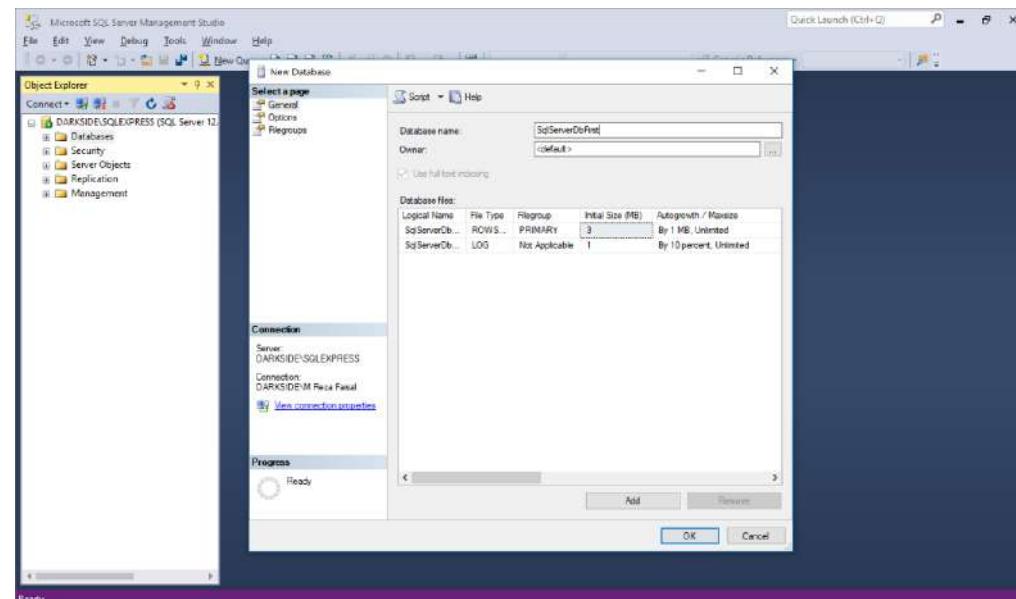


Gambar 84. SqlServerDbFirst - Daftar library yang telah diinstall.

Database

Langkah kedua adalah membuat database pada MS SQL Server dan membuat konfigurasi connecting string.

Untuk membuat database pada MS SQL Server, dapat digunakan Microsoft SQL Server Management Studio yang dapat diunduh di <https://go.microsoft.com/fwlink/?linkid=867670>.



Gambar 85. SqlServerDbFirst - Membuat database SqlServerDbFirst.

Membuat Database

Langkah pertama adalah membuat database dengan Microsoft SQL Server Management Studio. Klik kanan pada Databases kemudian pilih New Database. Kemudian ketikkan nama database pada window New Database seperti pada gambar di atas. Nama database yang digunakan adalah SqlServerDbFirst. Kemudian klik OK.

Langkah kedua adalah membuat tabel. Pada database SqlServerDbFirst, klik kanan pada Tables kemudian pilih Table.

The screenshot shows the 'Table_1' properties dialog in SQL Server Management Studio. The top section displays the table structure:

Column Name	Data Type	Allow Nulls
Id	int	<input type="checkbox"/>
Name	nchar(50)	<input checked="" type="checkbox"/>
Email	nchar(255)	<input checked="" type="checkbox"/>
Message	text	<input checked="" type="checkbox"/>
		<input type="checkbox"/>

The bottom section, 'Column Properties', is expanded for the 'Id' column. It shows the following settings:

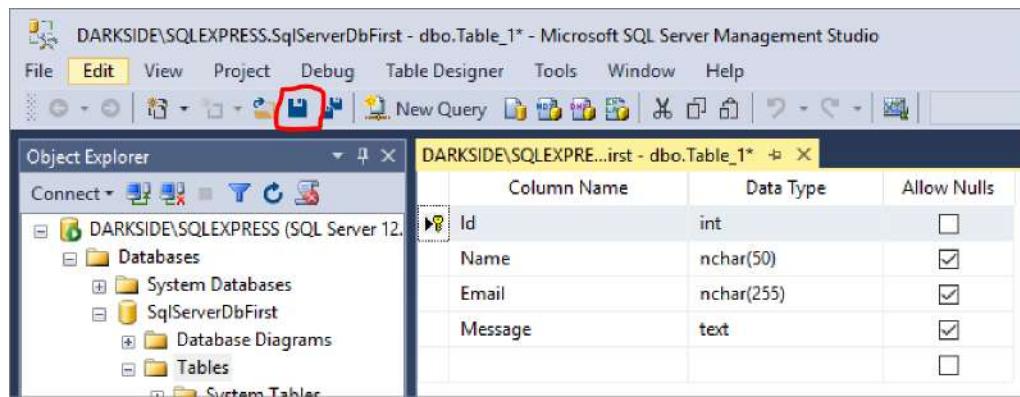
Collation	<database default>
Computed Column Specification	int
Condensed Data Type	int
Description	Yes
Deterministic	No
DTS-published	No
Full-text Specification	No
Has Non-SQL Server Subscriber	No
Identity Specification	Yes
(Is Identity)	Yes
Identity Increment	1
Identity Seed	1
Indexable	Yes

Gambar 86. SqlServerDbFirst - Membuat tabel Guestbook

Tabel memiliki field atau kolom sebagai berikut:

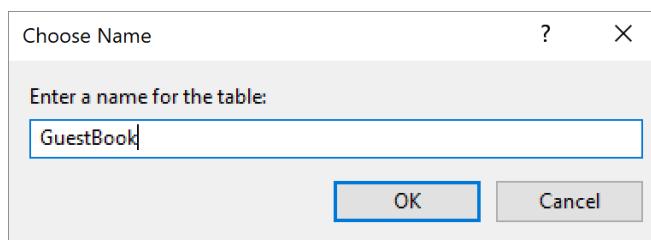
- Id, kolom ini adalah primary key. Tipe data kolom ini adalah integer. Agar nilai dari kolom ini dapat terisi dan nilai yang sesuai dengan urutan data yang masuk maka modifikasi nilai Identity Specification > {Is Identity} menjadi Yes.
- Name.
- Email.
- Message.

Kemudian klik tombol Save Table yang ada pada menu bar di bawah menu.



Gambar 87. SqlServerDbFirst - Tombol save table.

Kemudian berikan nama GuestBook untuk table tersebut.



Gambar 88. SqlServerDbFirst - Memberi nama table.

Koneksi Database

Connection string adalah string yang berisi nilai-nilai yang digunakan untuk melakukan koneksi ke database. Pada project ASP.NET Core MVC, connection string disimpan pada file appsettings.json. Berikut adalah kode connection string.

```
{
  "ConnectionStrings": {
    "Default": "Server=.\SQLEXPRESS; Database=SqlServerDbFirst;User
Id=sa;Password=MRezaFaisal;Trusted_Connection=True"
  }
}
```

Jika pada file appsettings.json telah terdapat konfigurasi lain, maka connection string ditambahkan pada konfigurasi sebelumnya setelah dipisahkan dengan tanda koma. Berikut adalah isi file appsettings.json secara lengkap.

```
appsettings.json
{
  "Logging": {
    "IncludeScopes": false,
    "LogLevel": {
      "Default": "Warning"
    }
  },
  "ConnectionStrings": {
    "Default": "Server=.\SQLEXPRESS;Database=SqlServerDbFirst;User
Id=sa;Password=MRezaFaisal;Trusted_Connection=True"
  }
}
```

Model

Langkah selanjutnya adalah membuat class model dan class data context. Kedua class ini disimpan pada folder Models. Karena telah dibuat table pada database maka pengguna dapat menggunakan perintah Scaffold-DbContext untuk membuat class entity model dan class data context.

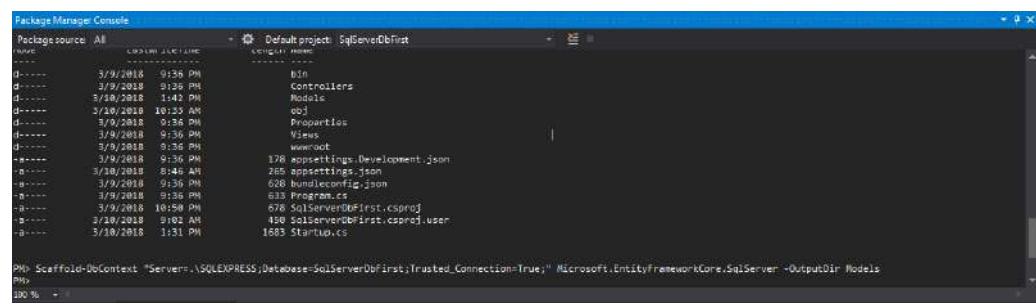
Untuk menjalankan perintah Scaffold-DbContext digunakan Package Manager Console. Untuk mengaktifkan console, pilih Tools > NuGet Package Manager > Package Manager Console.

Kemudian ketikan perintah dengan pola berikut ini.

```
Scaffold-DbContext "Connection String"  
Microsoft.EntityFrameworkCore.SqlServer -OutputDir OUTPUT_FOLDER
```

Jika isi connection string adalah seperti yang ditulis pada sub bab di atas. Kemudian file-file akan disimpan pada folder Models. Maka perintahnya dapat diketik sebagai berikut ini.

```
Scaffold-DbContext "Server=.\SQLEXPRESS;Database=SqlServerDbFirst;User  
Id=sa;Password=MRezaFaisal;Trusted_Connection=True"  
Microsoft.EntityFrameworkCore.SqlServer -OutputDir Models
```



The screenshot shows the Windows Taskbar at the bottom with several pinned icons. The Start button is visible on the far left. The main area of the screen displays the Package Manager Console window. The console output shows the command being run:

```
PM> Scaffold-DbContext "Server=.\SQLEXPRESS;Database=SqlServerDbFirst;Trusted_Connection=True;" Microsoft.EntityFrameworkCore.SqlServer -OutputDir Models
```

Below the command, the console lists numerous files and their details, such as creation date, time, and size, indicating the generation of new files.

Gambar 89. SqlServerDbFirst – Scaffold-DbContext.

Sebelum menjalankan perintah ini harap diperhatikan nilai pada opsi Default project yang akan menentukan ke project mana file yang digenerate akan disimpan.

Hasil dari perintah ini adalah file-file berikut ini.

- GuestBook.cs.
- SqlServerDbFirstContext.cs.

Class Entity Model

Class entity model adalah class yang memiliki property-property sesuai dengan atribut-atribut tabel GuestBook yang berfungsi untuk proses create, retrieve, update dan delete (CRUD).

```
GuestBook.cs  
using System;  
using System.Collections.Generic;  
  
namespace SqlServerDbFirst.Models  
{  
    public partial class GuestBook  
    {  
        public int Id { get; set; }  
        public string Name { get; set; }  
        public string Email { get; set; }  
        public string Message { get; set; }  
    }  
}
```

```
}
```

Class ini dibuat oleh perintah Scaffold-DbContext karena pada database hanya terdapat tabel GuestBook saja. Jika pada database itu terdapat 5 table, maka secara otomatis perintah tersebut akan membuat 5 class entity model untuk masing-masing table. Penjelasan detail tentang class entity model akan dibahas pada Bab Model-View-Controller sub bab Model.

Class Data Context

Pada pendekatan Database First, class ini berfungsi untuk memetakan class entity model dengan tabel pada database. Class ini merupakan class utama untuk implementasi Entity Framework Core pada project aplikasi web. Perbedaan pendekatan antara Database First dan Code First dapat dilihat dari kode class data context ini. Pada sub bab ini dapat dilihat bagaimana class data context pada pendekatan Database First.

Untuk project ini dibuat class data context dengan nama GuestBookDataContext.cs.

```
GuestBookDataContext.cs
using System;
using Microsoft.EntityFrameworkCore;
using Microsoft.EntityFrameworkCore.Metadata;

namespace SqlServerDbFirst.Models
{
    public partial class SqlServerDbFirstContext : DbContext
    {
        public virtual DbSet<GuestBook> GuestBook { get; set; }

        // protected override void OnConfiguring(DbContextOptionsBuilder
        optionsBuilder)
        // {
        //     if (!optionsBuilder.IsConfigured)
        //     {
        // #warning To protect potentially sensitive information in your connection
        // string, you should move it out of source code. See
        // http://go.microsoft.com/fwlink/?LinkId=723263 for guidance on storing
        // connection strings.
        //
        optionsBuilder.UseSqlServer(@"Server=.\SQLEXPRESS;Database=SqlServerDbFirst;
        Trusted_Connection=True;");
        // }
        //

        protected override void OnModelCreating(ModelBuilder modelBuilder)
        {
            modelBuilder.Entity<GuestBook>(entity =>
            {
                entity.Property(e => e.Email).HasColumnType("nchar(255)");
                entity.Property(e => e.Message).HasColumnType("text");
                entity.Property(e => e.Name).HasColumnType("nchar(50)");
            });
        }
    }
}
```

Untuk membuat class ini diperlukan library ini Microsoft.EntityFrameworkCore, library ini adalah untuk implementasi Entity Framework.

Nama class adalah bebas, tetapi class harus merupakan turunan dari class DbContext.

```
public partial class SqlServerDbFirstContext : DbContext
{
}
```

Langkah selanjutnya adalah membuat agar class entity GuestBook dapat digunakan untuk melakukan operasi menambah, membaca, mengedit dan menghapus data. Caranya adalah dengan membuat property GuestBooks dari class data context yang mana tipe propertnya itu dibentuk dari class DbSet.

```
public virtual DbSet<GuestBook> GuestBook { get; set; }
```

Sintaks dari kode di atas adalah sebagai berikut.

```
public virtual DbSet<NAMA_CLASS> NAMA_PROPERTY { get; set; }
```

Untuk pemetaan antara class entity model dengan tabel dapat dilihat pada method OnModelCreating. Pemetaan antara tabel ini biasanya dilakukan jika nama class entity model berbeda dengan nama table yang ada di database. Jika nama keduanya sama maka tidak perlu dilakukan pemetaan.

Untuk contoh kasus nama class entity yang berbeda dengan nama table, misal nama class entity adalah GuestBook dan nama table di database adalah buku_tamu maka perlu dilakukan pemetaan sebagai berikut ini.

```
modelBuilder.Entity<GuestBook>().ToTable("buku_tamu");
```

Sintaks umum kode di atas adalah sebagai berikut.

```
modelBuilder.Entity<GuestBook>().ToTable("NAMA_TABLE");
```

Sedangkan untuk memetakan antara property-property yang dimiliki oleh class entity model dengan atribut-atribut yang dimiliki oleh tabel dengan menggunakan kode berikut ini.

```
modelBuilder.Entity<GuestBook>(entity =>
{
    entity.Property(e => e.Email).HasColumnType("nchar(255)");
    entity.Property(e => e.Message).HasColumnType("text");
    entity.Property(e => e.Name).HasColumnType("nchar(50)");
});
```

Kasus di atas juga dilakukan jika nama property pada class entity model sama dengan nama field pada table.

Jika nama property dan field berbeda maka dapat dilakukan pemetaan dengan sintaks umum sebagai berikut ini.

```
entity.Property(e => e.Message).HasColumnName("NAMA_ATTRIBUT");
```

Kemudian untuk memetakan property yang akan menangani atribut primary key dari tabel digunakan kode berikut ini.

```
entity.Property(e => e.Id).HasColumnName("Id");
```

Sintaks umum kode di atas adalah sebagai berikut.

```
modelBuilder.Entity<NAMA_CLASS>().HasKey(e => new { e.NAMA_PROPERTY });
```

Class Startup.cs

Telah dijelaskan di atas bahwa pada class ini akan dilakukan penentuan provider database yang digunakan, dalam hal ini adalah provider database MS SQL Server. Selain itu juga proses pengambilan nilai connection string untuk digunakan oleh provider database MS SQL Server.

Pada sub bab ini akan dilakukan registrasi class data context menggunakan dependency injection pada class Startup.cs. Untuk itu harus dilakukan modifikasi class SqlServerDbFirstContext.cs dengan cara menghapus method OnConfiguring dan menambahkan kode berikut ini.

```
public SqlServerDbFirstContext(DbContextOptions<SqlServerDbFirstContext>
options) : base(options)
{
}
```

Selanjutnya adalah modifikasi file Startup.cs. Tambahkan kedua namespace ini ke dalam file tersebut.

```
using Microsoft.EntityFrameworkCore;
using SqlServerDbFirst.Models;
```

Kemudian tambahkan kode berikut ini ke dalam method ConfigureServices.

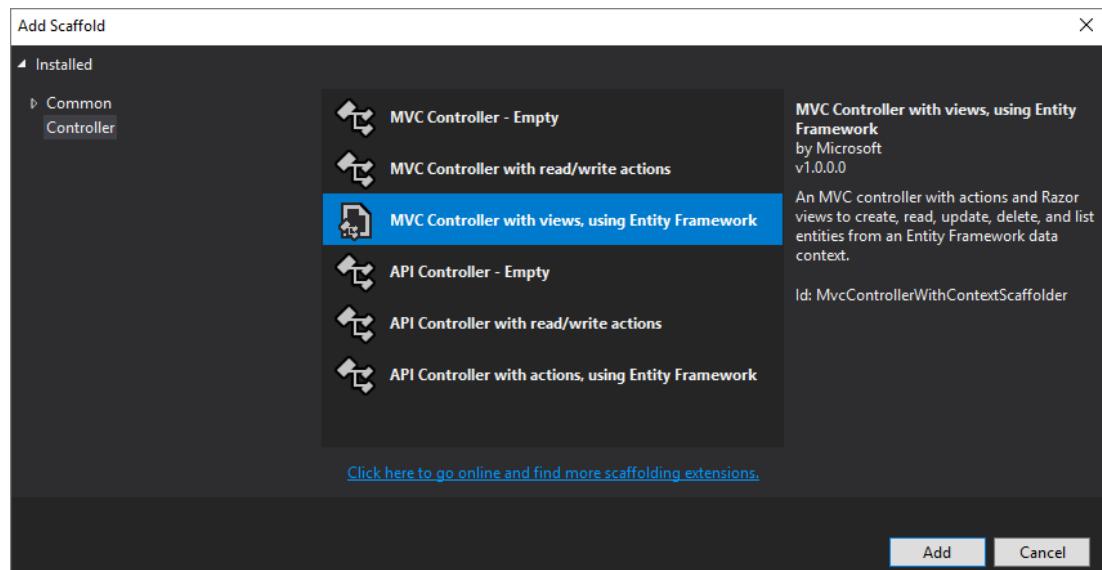
```
services.AddDbContext<SqlServerDbFirstContext>(options =>
options.UseSqlServer(Configuration.GetConnectionString("Default")));
```

Kode di atas adalah cara registrasi class SqlServerDbFirstContext pada class Startup.cs. Kode di atas juga berfungsi untuk membaca konfigurasi connection string pada file appsettings.json.

Controller

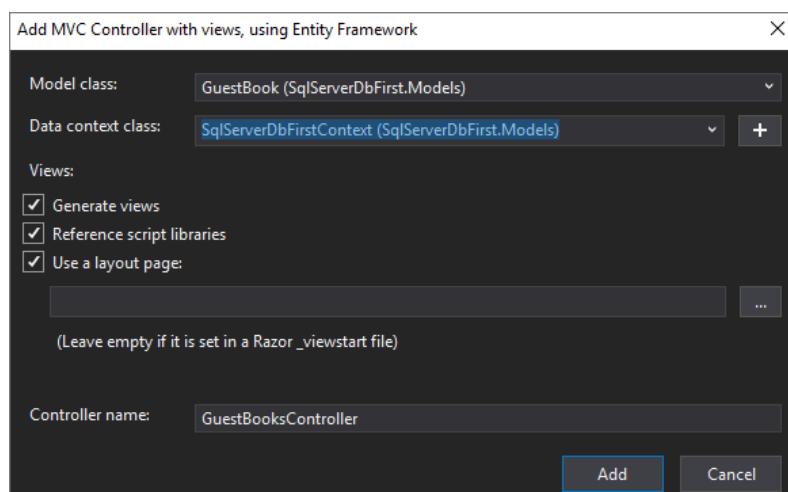
Pada sub bab ini akan dijelaskan cara membuat komponen controller dan view secara otomatis dengan menggunakan fasilitas dari Visual Studio 2017. Pertama klik kanan pada folder Controllers kemudian pilih Add > Controller.

Pada window Add Scaffold pilih MVC Controller with views, using Entity Framework seperti pada gambar di bawah ini.



Gambar 90. SqlServerDbFirst – Add Scaffold.

Klik tombol Add kemudian akan ditampilkan window Add MVC Controller with views, using Entity Framework seperti pada gambar di bawah ini.



Gambar 91. SqlServerDbFirst – Add MVC Controller with views, using Entity Framework.

Pada window di atas, pilih nilai pada opsi Model class dan Data context class. Berikut adalah nilai untuk masing-masing opsi:

- Model class, dipilih nilai GuestBook (SqlServerDbFirst.Models).
- Data context class, dipilih nilai SqlServerDbContext (SqlServerDbFirst.Models).

Class controller yang dibuat ini akan disimpan dengan nama GuestBookController sesuai dengan nilai pada kolom Controller name.

Kemudian klik tombol Add dan hasilnya dapat dilihat pada folder Controllers, terdapat file GuestBooksController.cs. Berikut ini adalah isi dari file ini.

```
GuestBooksController.cs
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Mvc;
```

```

using Microsoft.AspNetCore.Mvc.Rendering;
using Microsoft.EntityFrameworkCore;
using SqlServerDbFirst.Models;

namespace SqlServerDbFirst.Controllers
{
    public class GuestBooksController : Controller
    {
        private readonly SqlServerDbContext _context;

        public GuestBooksController(SqlServerDbContext context)
        {
            _context = context;
        }

        // GET: GuestBooks
        public async Task<IActionResult> Index()
        {
            return View(await _context.GuestBook.ToListAsync());
        }

        // GET: GuestBooks/Details/5
        public async Task<IActionResult> Details(int? id)
        {
            if (id == null)
            {
                return NotFound();
            }

            var guestBook = await _context.GuestBook
                .SingleOrDefaultAsync(m => m.Id == id);
            if (guestBook == null)
            {
                return NotFound();
            }

            return View(guestBook);
        }

        // GET: GuestBooks/Create
        public IActionResult Create()
        {
            return View();
        }

        // POST: GuestBooks/Create
        // To protect from overposting attacks, please enable the specific
        properties you want to bind to, for
        // more details see http://go.microsoft.com/fwlink/?LinkId=317598.
        [HttpPost]
        [ValidateAntiForgeryToken]
        public async Task<IActionResult>
Create([Bind("Id,Name,Email,Message")] GuestBook guestBook)
        {
            if (ModelState.IsValid)
            {
                _context.Add(guestBook);
                await _context.SaveChangesAsync();
                return RedirectToAction(nameof(Index));
            }
            return View(guestBook);
        }
    }
}

```

```

    }

    // GET: GuestBooks/Edit/5
    public async Task<IActionResult> Edit(int? id)
    {
        if (id == null)
        {
            return NotFound();
        }

        var guestBook = await _context.GuestBook.SingleOrDefaultAsync(m
=> m.Id == id);
        if (guestBook == null)
        {
            return NotFound();
        }
        return View(guestBook);
    }

    // POST: GuestBooks/Edit/5
    // To protect from overposting attacks, please enable the specific
properties you want to bind to, for
    // more details see http://go.microsoft.com/fwlink/?LinkId=317598.
    [HttpPost]
    [ValidateAntiForgeryToken]
    public async Task<IActionResult> Edit(int id,
[Bind("Id,Name,Email,Message")] GuestBook guestBook)
    {
        if (id != guestBook.Id)
        {
            return NotFound();
        }

        if (ModelState.IsValid)
        {
            try
            {
                _context.Update(guestBook);
                await _context.SaveChangesAsync();
            }
            catch (DbUpdateConcurrencyException)
            {
                if (!GuestBookExists(guestBook.Id))
                {
                    return NotFound();
                }
                else
                {
                    throw;
                }
            }
            return RedirectToAction(nameof(Index));
        }
        return View(guestBook);
    }

    // GET: GuestBooks/Delete/5
    public async Task<IActionResult> Delete(int? id)
    {
        if (id == null)
        {

```

```

        return NotFound();
    }

    var guestBook = await _context.GuestBook
        .SingleOrDefaultAsync(m => m.Id == id);
    if (guestBook == null)
    {
        return NotFound();
    }

    return View(guestBook);
}

// POST: GuestBooks/Delete/5
[HttpPost, ActionName("Delete")]
[ValidateAntiForgeryToken]
public async Task<IActionResult> DeleteConfirmed(int id)
{
    var guestBook = await _context.GuestBook.SingleOrDefaultAsync(m
=> m.Id == id);
    _context.GuestBook.Remove(guestBook);
    await _context.SaveChangesAsync();
    return RedirectToAction(nameof(Index));
}

private bool GuestBookExists(int id)
{
    return _context.GuestBook.Any(e => e.Id == id);
}
}
}

```

Kemudian pada folder Views akan didapati tambahan folder GuestBooks yang berisi file-file berikut:

- Create.cshtml.
- Delete.cshtml.
- Detail.cshtml.
- Edit.cshtml.
- Index.cshtml.

Detail tentang file ini akan diberikan pada sub bab selanjutnya.

Views

Telah disebutkan di sub bab sebelumnya bahwa telah dibuat 5 file komponen views. Berikut adalah isi masing-masing file tersebut.

Index.cshtml

File ini berfungsi untuk menampilkan daftar data dari table GuestBook.

Index.cshtml
@model IEnumerable<SqlServerDbFirst.Models.GuestBook>
@{
ViewData["Title"] = "Index";

```

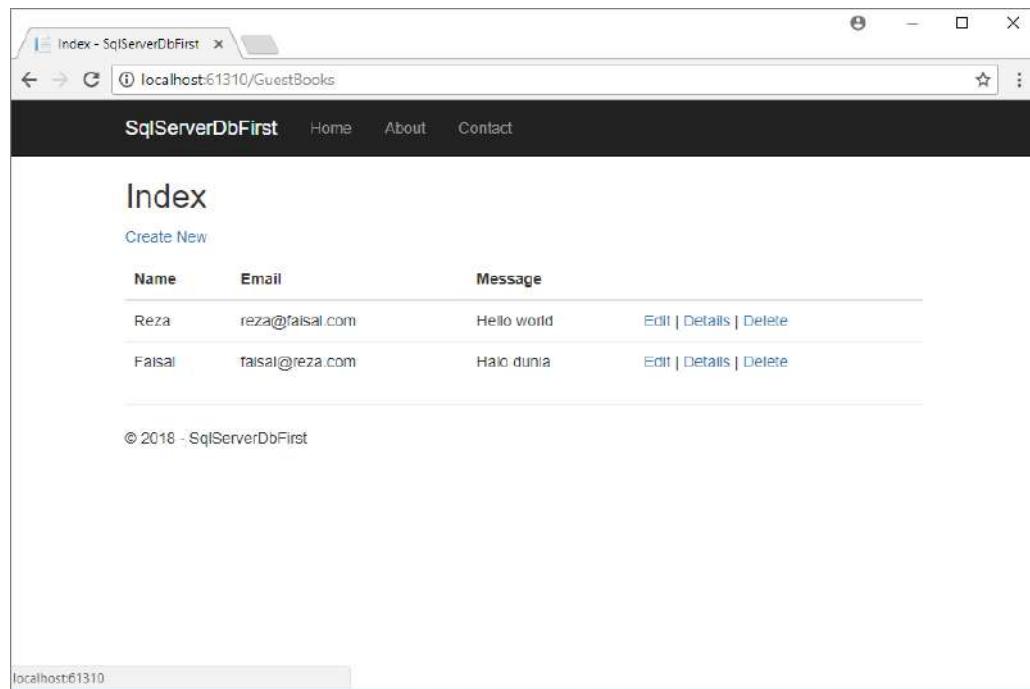
        Layout = "~/Views/Shared/_Layout.cshtml";
    }

<h2>Index</h2>

<p>
    <a asp-action="Create">Create New</a>
</p>
<table class="table">
    <thead>
        <tr>
            <th>
                @Html.DisplayNameFor(model => model.Name)
            </th>
            <th>
                @Html.DisplayNameFor(model => model.Email)
            </th>
            <th>
                @Html.DisplayNameFor(model => model.Message)
            </th>
            <th></th>
        </tr>
    </thead>
    <tbody>
@foreach (var item in Model) {
        <tr>
            <td>
                @Html.DisplayFor(modelItem => item.Name)
            </td>
            <td>
                @Html.DisplayFor(modelItem => item.Email)
            </td>
            <td>
                @Html.DisplayFor(modelItem => item.Message)
            </td>
            <td>
                <a asp-action="Edit" asp-route-id="@item.Id">Edit</a> |
                <a asp-action="Details" asp-route-id="@item.Id">Details</a>
                |
                <a asp-action="Delete" asp-route-id="@item.Id">Delete</a>
            </td>
        </tr>
}
    </tbody>
</table>

```

Berikut adalah tampilan dari view ini.



Gambar 92. SqlServerDbFirst – Index.cshtml.

Detail.cshtml

File ini berfungsi untuk menampilkan daftar data dari table GuestBook.

```
Detail.cshtml
@model SqlServerDbFirst.Models.GuestBook

@{
    ViewData["Title"] = "Details";
    Layout = "~/Views/Shared/_Layout.cshtml";
}



## Details



#### GuestBook



---



@Html.DisplayNameFor(model => model.Name)
:   @Html.DisplayFor(model => model.Name)


@Html.DisplayNameFor(model => model.Email)
:   @Html.DisplayFor(model => model.Email)


@Html.DisplayNameFor(model => model.Message)
:   @Html.DisplayFor(model => model.Message)

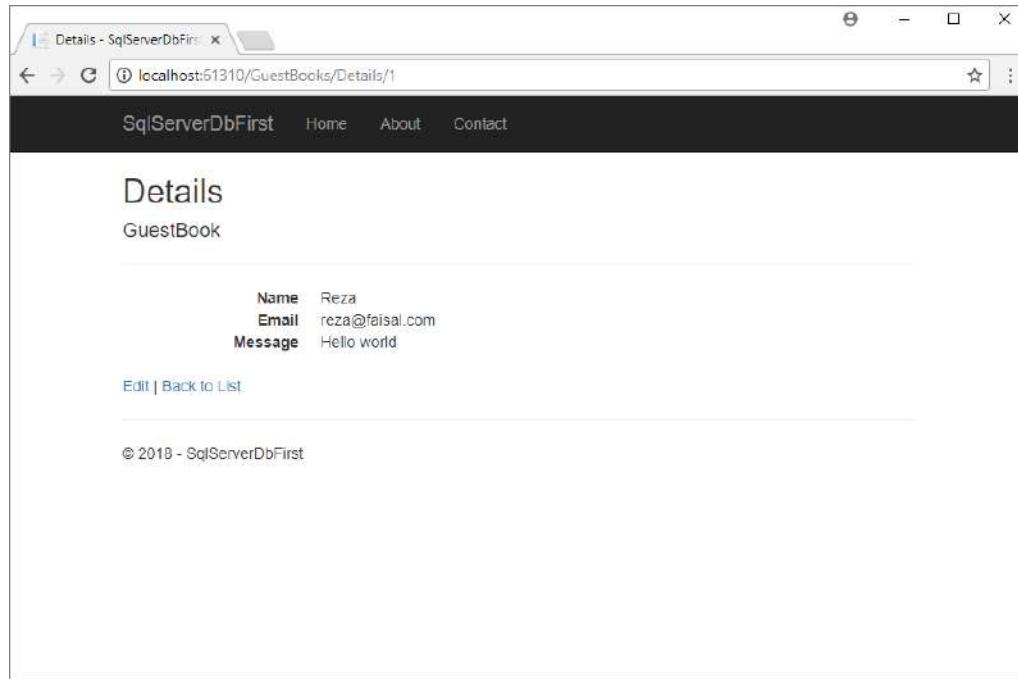

```

```

        </dl>
    </div>
    <div>
        <a asp-action="Edit" asp-route-id="@Model.Id">Edit</a> |
        <a asp-action="Index">Back to List</a>
    </div>

```

Berikut adalah tampilan dari view ini.



Gambar 93. SqlServerDbFirst – Detail.cshtml.

Create.cshtml

File ini berfungsi untuk menampilkan daftar data dari table GuestBook.

```

Create.cshtml
@model SqlServerDbFirst.Models.GuestBook

@{
    ViewData["Title"] = "Create";
    Layout = "~/Views/Shared/_Layout.cshtml";
}

<h2>Create</h2>

<h4>GuestBook</h4>
<hr />
<div class="row">
    <div class="col-md-4">
        <form asp-action="Create">
            <div asp-validation-summary="ModelOnly" class="text-danger"></div>
            <div class="form-group">
                <label asp-for="Name" class="control-label"></label>
                <input asp-for="Name" class="form-control" />
                <span asp-validation-for="Name" class="text-danger"></span>
            </div>
            <div class="form-group">
                <label asp-for="Email" class="control-label"></label>

```

```

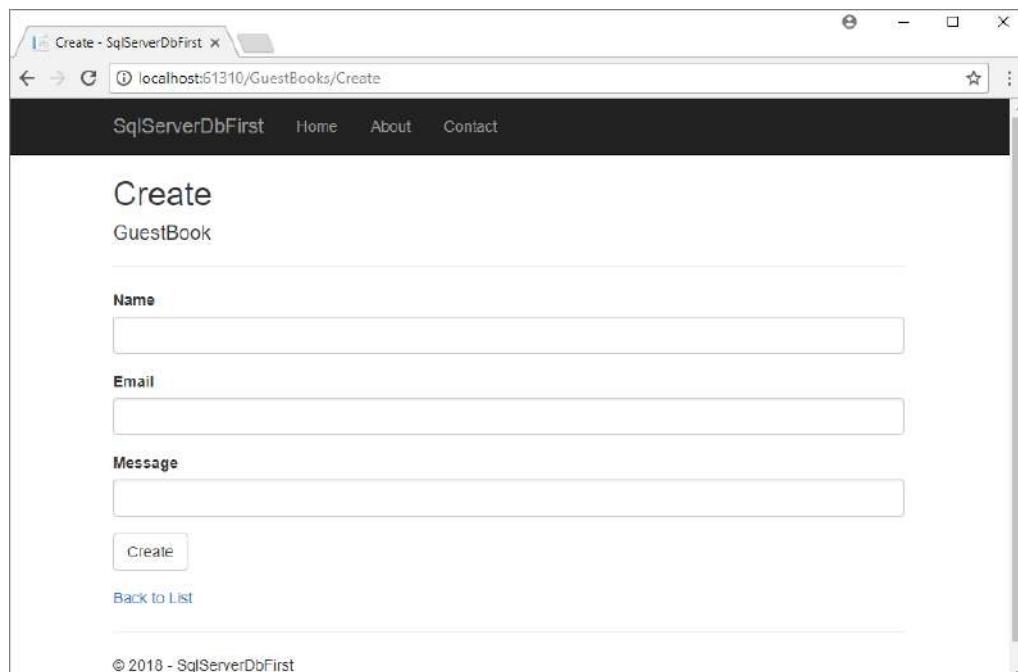
        <input asp-for="Email" class="form-control" />
        <span asp-validation-for="Email" class="text-danger"></span>
    </div>
    <div class="form-group">
        <label asp-for="Message" class="control-label"></label>
        <input asp-for="Message" class="form-control" />
        <span asp-validation-for="Message" class="text-
danger"></span>
    </div>
    <div class="form-group">
        <input type="submit" value="Create" class="btn btn-default" />
    </div>
</div>

<div>
    <a asp-action="Index">Back to List</a>
</div>

@section Scripts {
    @await Html.RenderPartialAsync("_ValidationScriptsPartial")
}

```

Berikut adalah tampilan dari view ini.



Gambar 94. SqlServerDbFirst – Create.cshtml.

Edit.cshtml

File ini berfungsi untuk menampilkan daftar data dari table GuestBook.

```

Edit.cshtml
@model SqlServerDbFirst.Models.GuestBook
@{

```

```

        ViewData["Title"] = "Edit";
        Layout = "~/Views/Shared/_Layout.cshtml";
    }

<h2>Edit</h2>

<h4>GuestBook</h4>
<hr />
<div class="row">
    <div class="col-md-4">
        <form asp-action="Edit">
            <div asp-validation-summary="ModelOnly" class="text-
danger"></div>
                <input type="hidden" asp-for="Id" />
                <div class="form-group">
                    <label asp-for="Name" class="control-label"></label>
                    <input asp-for="Name" class="form-control" />
                    <span asp-validation-for="Name" class="text-danger"></span>
                </div>
                <div class="form-group">
                    <label asp-for="Email" class="control-label"></label>
                    <input asp-for="Email" class="form-control" />
                    <span asp-validation-for="Email" class="text-danger"></span>
                </div>
                <div class="form-group">
                    <label asp-for="Message" class="control-label"></label>
                    <input asp-for="Message" class="form-control" />
                    <span asp-validation-for="Message" class="text-
danger"></span>
                </div>
                <div class="form-group">
                    <input type="submit" value="Save" class="btn btn-default" />
                </div>
            </form>
        </div>
    </div>
    <div>
        <a asp-action="Index">Back to List</a>
    </div>

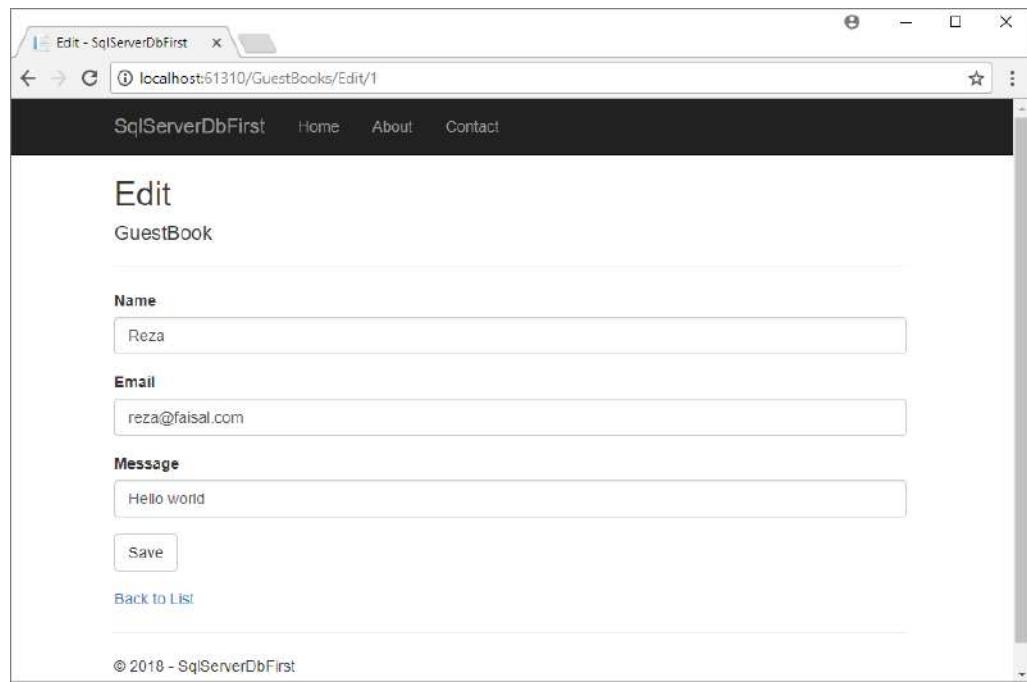
```

```

@section Scripts {
    @await Html.RenderPartialAsync("_ValidationScriptsPartial");
}

```

Berikut adalah tampilan dari view ini.



Gambar 95. SqlServerDbFirst – Edit.cshtml.

Delete.cshtml

File ini berfungsi untuk menampilkan daftar data dari table GuestBook.

```
Delete.cshtml
@model SqlServerDbFirst.Models.GuestBook

@{
    ViewData["Title"] = "Delete";
    Layout = "~/Views/Shared/_Layout.cshtml";
}



## Delete



### Are you sure you want to delete this?



#### GuestBook



---



@Html.DisplayNameFor(model => model.Name)


@Html.DisplayFor(model => model.Name)


@Html.DisplayNameFor(model => model.Email)


@Html.DisplayFor(model => model.Email)


@Html.DisplayNameFor(model => model.Message)


@Html.DisplayFor(model => model.Message)


```

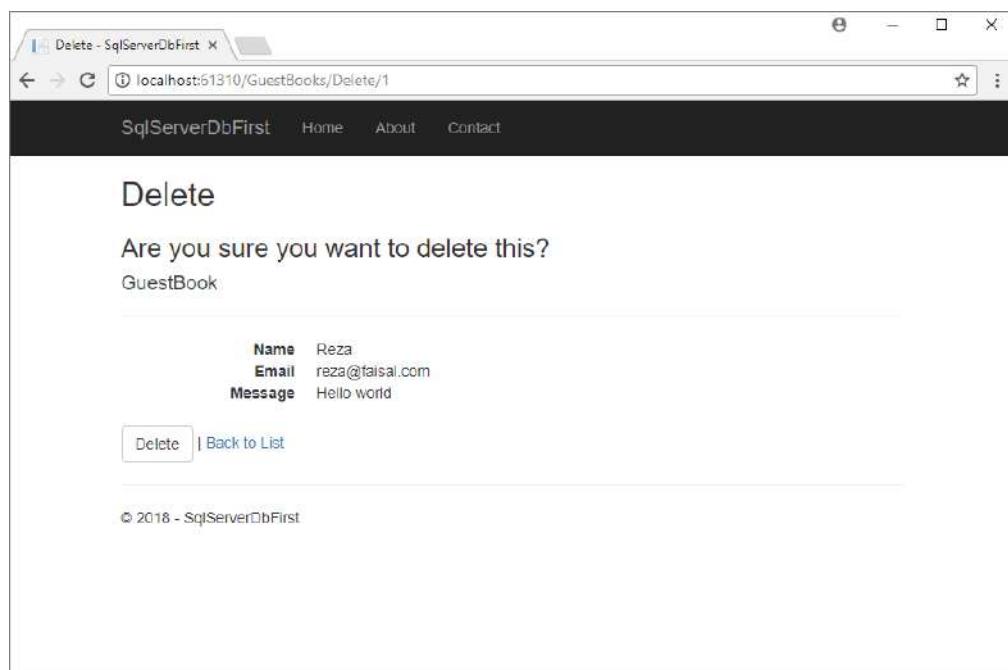
```

        </dd>
    </dl>

    <form asp-action="Delete">
        <input type="hidden" asp-for="Id" />
        <input type="submit" value="Delete" class="btn btn-default" /> |
        <a asp-action="Index">Back to List</a>
    </form>
</div>

```

Berikut adalah tampilan dari view ini.



Gambar 96. SqlServerDbFirst – Delete.cshtml.

Code First

Pada sub bab ini akan diberikan langkah-langkah implementasi Entity Framework Code dengan pendekatan Code First.

Project

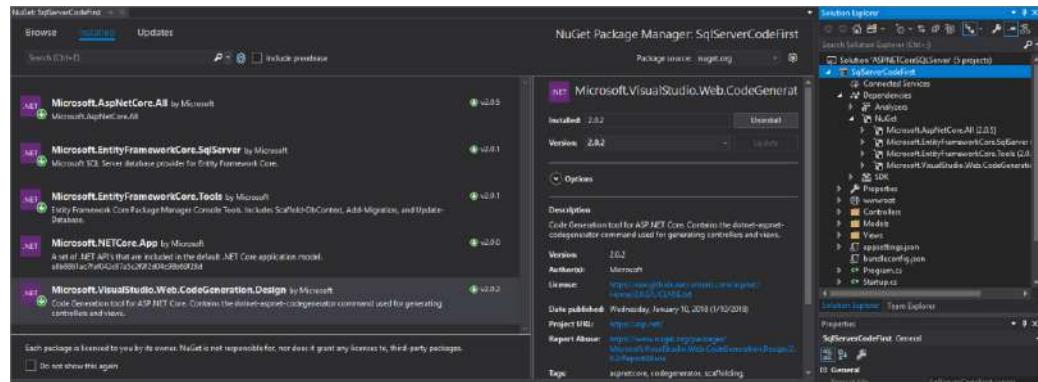
Langkah pertama adalah membuat project dengan nama SqlServerCodeFirst.

Membuat Project

Langkah-langkah pembuatan project SqlServerCodeFirst adalah sama seperti langkah-langkah membuat project SqlServerDbFirst yang telah dibahas pada sub bab Databaset First. Setelah project dibuat, ubah status project ini menjadi Startup Project. Caranya adalah dengan klik kanan pada project SqlServerCodeFirst, kemudian pilih Set as Startup Project.

Menyiapkan Library

Library-library yang mesti diinstall pada project ini sama seperti library-library yang digunakan pada project SqlServerDbFirst. Berikut ini adalah daftar library-library pendukung Entity Framework yang telah diinstall pada project SqlServerCodeFirst.



Gambar 97. SqlServerCodeFirst – Daftar library Entity Framework.

Model

Langkah selanjutnya adalah membuat model yaitu class entity model dan class data context.

Class Entity Model

Berikut ini adalah kode class entity model GuestBook.cs yang harus ditambahkan ke dalam folder Models.

```
GuestBook.cs
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;

namespace SqlServerCodeFirst.Models
{
    public class GuestBook
    {
        public int Id { set; get; }
        public String Name { set; get; }
        public String Email { set; get; }
        public String Message { set; get; }
    }
}
```

Kode ini sama dengan kode yang sebelumnya juga dibuat pada sub bab Database First perbedaannya hanya pada nama namespace yang digunakan yaitu SqlServerDbFirst.Models.

Class Data Context

Langkah selanjutnya adalah membuat class data context, isi class ini berbeda dengan class data context yang dibuat pada pendekatan Database First. Pada class ini tidak perlu dilakukan pemetaan antara class entity model dengan tabel pada database. Hal ini dikarena database dan tabel belum ada saat class ini dibuat.

Berikut ini adalah kode class data context dengan nama SqlServerCodeFirstContext.cs untuk implementasi pendekatan Code First.

```
SqlServerCodeFirstContext.cs
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;

using Microsoft.EntityFrameworkCore;

namespace SqlServerCodeFirst.Models
{
    public class SqlServerCodeFirstContext : DbContext
    {
        public virtual DbSet<GuestBook> GuestBooks { get; set; }

        public SqlServerCodeFirstContext(DbContextOptions<SqlServerCodeFirstContext> options) : base(options)
        {
        }

        protected override void OnModelCreating(ModelBuilder modelBuilder)
        {
            modelBuilder.Entity<GuestBook>().HasKey(e => new { e.Id });
        }
    }
}
```

Untuk membuat class ini diperlukan library Microsoft.EntityFrameworkCore, library ini adalah library untuk implementasi Entity Framework. Nama class adalah bebas, tetapi class harus merupakan turunan dari class DbContext.

```
public class SqlServerCodeFirstContext : DbContext
{
}
```

Pemetaan yang perlu dilakukan pada method OnModelCreating adalah penentuan property mana yang akan dijadikan sebagai primary key dengan kode berikut ini.

```
modelBuilder.Entity<GuestBook>().HasKey(e => new { e.Id });
```

Langkah selanjutnya adalah membuat agar class entity GuestBook dapat digunakan untuk melakukan operasi menambah, membaca, mengedit dan menghapus data. Caranya adalah dengan membuat property GuestBooks dari class data context yang mana tipe property itu dibentuk dari class DbSet.

Berikut ini adalah kode yang digunakan untuk membuat property GuestBooks.

```
public virtual DbSet<GuestBook> GuestBook { get; set; }
```

Class Startup.cs

Isi class ini sama dengan class Startup.cs yang telah dibuat pada pendekatan Database First, perbedaannya hanya pada namespace SqlServerCodeFirst.

```
Startup.cs
```

```
using System;
```

```

using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Builder;
using Microsoft.AspNetCore.Hosting;
using Microsoft.Extensions.Configuration;
using Microsoft.Extensions.DependencyInjection;

using Microsoft.EntityFrameworkCore;
using SqlServerCodeFirst.Models;

namespace SqlServerCodeFirst
{
    public class Startup
    {
        public Startup(IConfiguration configuration)
        {
            Configuration = configuration;
        }

        public IConfiguration Configuration { get; }

        // This method gets called by the runtime. Use this method to add
        services to the container.
        public void ConfigureServices(IServiceCollection services)
        {
            services.AddMvc();
            services.AddDbContext<SqlServerCodeFirstContext>(options =>
options.UseSqlServer(Configuration.GetConnectionString("Default")));
        }

        // This method gets called by the runtime. Use this method to
        configure the HTTP request pipeline.
        public void Configure(IApplicationBuilder app, IHostingEnvironment
env)
        {
            if (env.IsDevelopment())
            {
                app.UseBrowserLink();
                app.UseDeveloperExceptionPage();
            }
            else
            {
                app.UseExceptionHandler("/Home/Error");
            }

            app.UseStaticFiles();

            app.UseMvc(routes =>
            {
                routes.MapRoute(
                    name: "default",
                    template: "{controller=Home}/{action=Index}/{id?}");
            });
        }
    }
}

```

Pada class ini harus ditambahkan namespace berikut ini:

- Microsoft.EntityFrameworkCore.
- SqlServerCodeFirst.Models.

Tujuan penambahan namespace di atas adalah agar class-class yang ada di dalam namespace tersebut dapat diakses pada class Startup.cs ini.

Database

Langkah selanjutnya adalah membuat database dan tabel, tetapi pembuatannya akan dilakukan secara otomatis.

Connection String

Langkah pertama pada proses ini adalah menambahkan connection string pada file appsettings.json sebagai berikut.

```
appsettings.json
{
  "Logging": {
    "IncludeScopes": false,
    "LogLevel": {
      "Default": "Warning"
    }
  },
  "ConnectionStrings": {
    "Default": "Server=.\SQLEXPRESS; Database=SqlServerCodeFirst;UserId=sa;Password=MRezaFaisal;Trusted_Connection=True"
  }
}
```

Migration Class

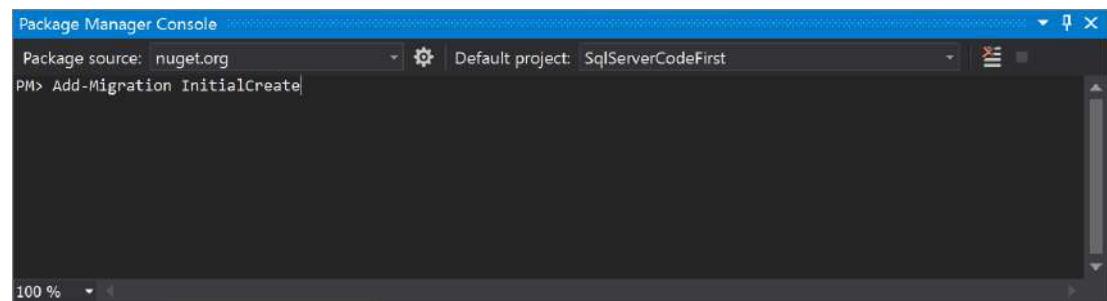
Migration class adalah class yang berisi kode untuk membuat database dan tabel sesuai dengan appsettings.json dan class entity model. Class ini dibuat secara otomatis dengan menggunakan perintah dengan sintaks berikut ini.

```
Add-Migration NAMA_CLASS
```

Jika nama class adalah InitialCreate maka sintaks di atas menjadi sebagai berikut.

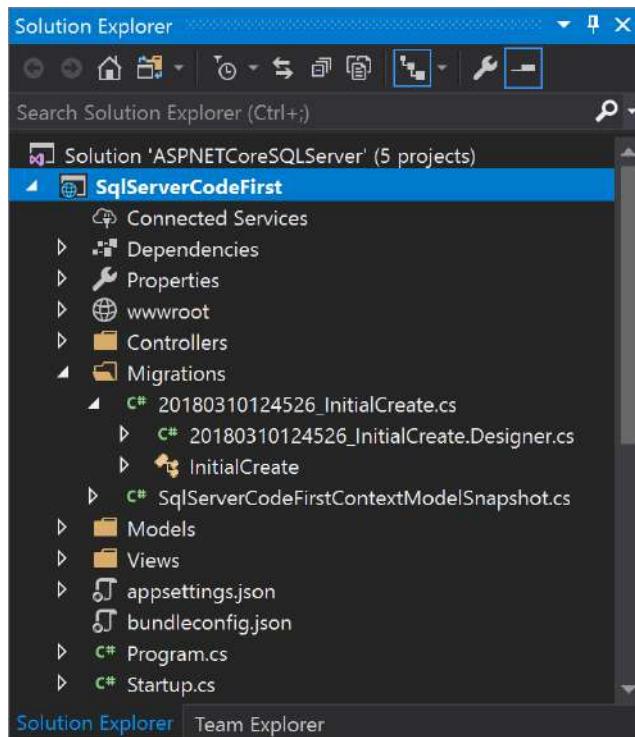
```
Add-Migration InitialCreate
```

Perintah ini dapat diketikkan pada Package Manager Console di Visual Studio. Pastikan pilih SqlServerCodeFirst pada opsi Default project.



Gambar 98. SqlServerCodeFirst - Migration class.

Hasilnya dapat dilihat akan dibuat folder Migrations seperti pada gambar di bawah ini.



Gambar 99. SqlServerCodeFirst - Migrations.

Pada folder Migration terdapat 2 file penting yaitu:

- 20180310124526_InitialCreate.cs.
- SqlServerCodeFirstContextModelSnapshot.cs.

Pada dua file pertama terdapat angka di depan nama file. Angka tersebut menyatakan tanggal dan waktu file dibuat. Berikut ini adalah isi dari ketiga file tersebut.

File InitialCreate.cs adalah file yang dibuat atau digenerate oleh perintah Add-Migration. Class ini berfungsi untuk membuat database yang diperlukan dari awal.

```
20180310124526_InitialCreate.cs
using Microsoft.EntityFrameworkCore.Metadata;
using Microsoft.EntityFrameworkCore.Migrations;
using System;
using System.Collections.Generic;

namespace SqlServerCodeFirst.Migrations
{
    public partial class InitialCreate : Migration
    {
        protected override void Up(MigrationBuilder migrationBuilder)
        {
            migrationBuilder.CreateTable(
                name: "GuestBooks",
                columns: table => new
                {
                    Id = table.Column<int>(nullable: false)
                        .Annotation("SqlServer:ValueGenerationStrategy", "IdentityColumn"),
                    Email = table.Column<string>(nullable: true),
                    Message = table.Column<string>(nullable: true),
                });
        }

        protected override void Down(MigrationBuilder migrationBuilder)
        {
            migrationBuilder.DropTable("GuestBooks");
        }
    }
}
```

```

        Name = table.Column<string>(nullable: true)
    },
    constraints: table =>
    {
        table.PrimaryKey("PK_GuestBooks", x => x.Id);
    });
}

protected override void Down(MigrationBuilder migrationBuilder)
{
    migrationBuilder.DropTable(
        name: "GuestBooks");
}
}
}

```

Pada class di atas terdapat dua method, yaitu:

- Up, method ini untuk membuat table pada database berdasarkan model yang telah dibuat.
- Down, method untuk menghapus table.

Pada proses migrasi membuat snapshot dari skema database yang terbaru.

```

SqlServerCodeFirstContextModelSnapshot.cs
// <auto-generated />
using Microsoft.EntityFrameworkCore;
using Microsoft.EntityFrameworkCore.Infrastructure;
using Microsoft.EntityFrameworkCore.Metadata;
using Microsoft.EntityFrameworkCore.Migrations;
using Microsoft.EntityFrameworkCore.Storage;
using Microsoft.EntityFrameworkCore.Storage.Internal;
using SqlServerCodeFirst.Models;
using System;

namespace SqlServerCodeFirst.Migrations
{
    [DbContext(typeof(SqlServerCodeFirstContext))]
    partial class SqlServerCodeFirstContextModelSnapshot : ModelSnapshot
    {
        protected override void BuildModel(ModelBuilder modelBuilder)
        {
#pragma warning disable 612, 618
            modelBuilder
                .HasAnnotation("ProductVersion", "2.0.1-rtm-125")
                .HasAnnotation("SqlServer:ValueGenerationStrategy", SqlServerValueGenerationStrategy.IdentityColumn);

            modelBuilder.Entity("SqlServerCodeFirst.Models.GuestBook", b =>
            {
                b.Property<int>("Id")
                    .ValueGeneratedOnAdd();

                b.Property<string>("Email");

                b.Property<string>("Message");

                b.Property<string>("Name");

                b.HasKey("Id");

                b.ToTable("GuestBooks");
            });
        }
    }
}

```

```
#pragma warning restore 612, 618
    }
}
```

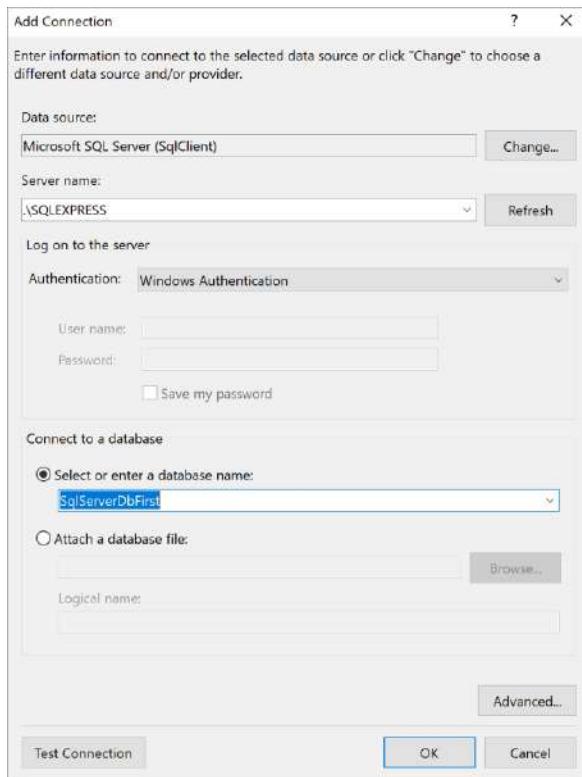
Karena skema database terbaru direpresentasikan oleh kode pada komponen model, maka EF Core tidak harus berinteraksi dengan database pada saat proses migrasi dilakukan. Ketika perintah Add-Migrations dijalankan maka EF Core akan menentukan apa yang diubah dengan membandingkan model dengan file snapshot ini.

Database

Setelah migration class dibuat, selanjutnya adalah membuat database dan tabel sesuai dengan migration class tersebut. Pembuatannya dilakukan secara otomatis dengan cara mengetik perintah ini pada Package Manager Console.

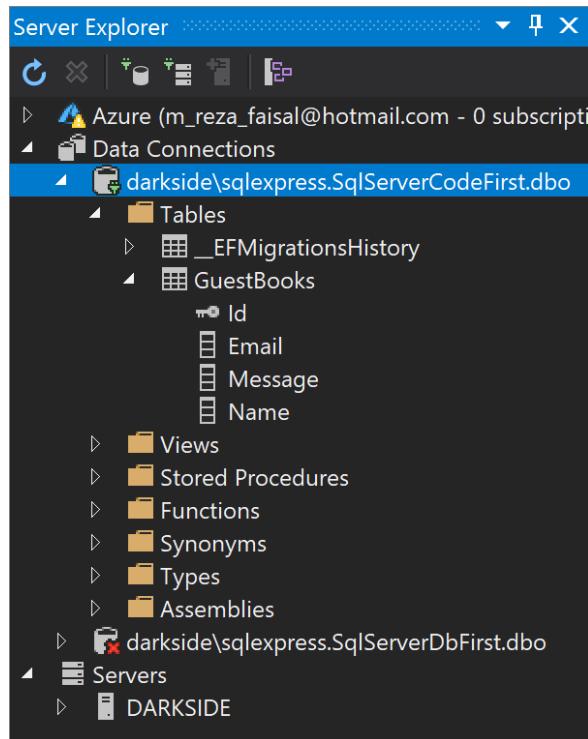
```
Update-Database
```

Hasilnya dapat dilihat database SqlServerCodeFirst dan tabel GuestBooks dibuat secara otomatis. Untuk melihat database dan table ini pada MS SQL server dapat dilakukan melalui Visual Studio. Caranya adalah dengan klik Tools > Connect to Database. Kemudian isikan nilai-nilai pada "Server name" dan "Select or enter database name" pada window Add Connection seperti pada gambar di bawah ini seperti pada gambar di bawah ini.



Gambar 100. SqlServerCodeFirst - Add Connection.

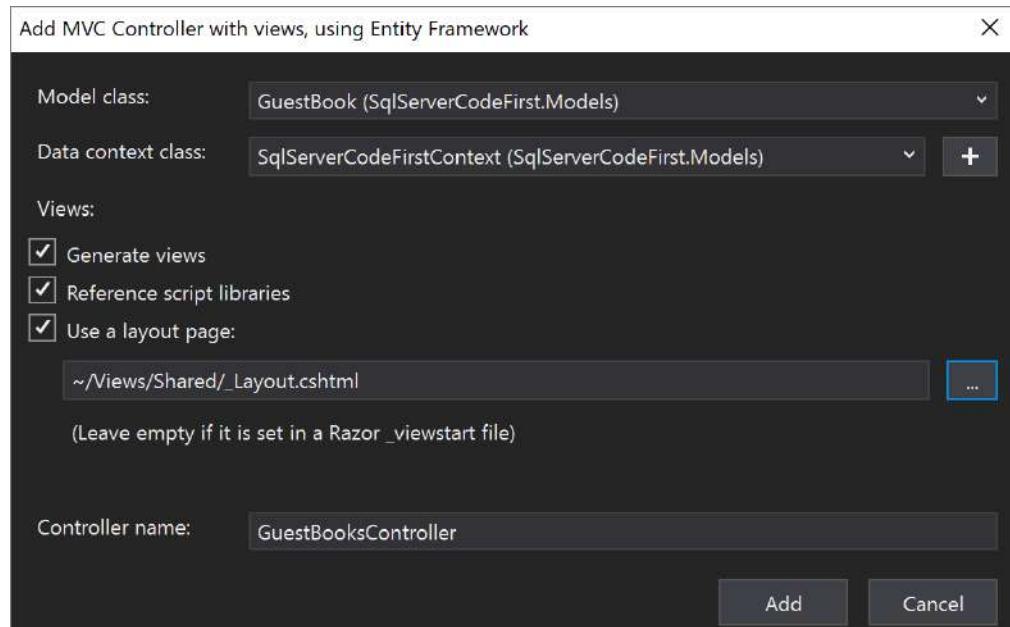
Untuk melihat hasilnya dapat dilihat pada bagian Server Explorer seperti pada gambar di bawah ini.



Gambar 101. SqlServerCodeFirst - Server Explorer.

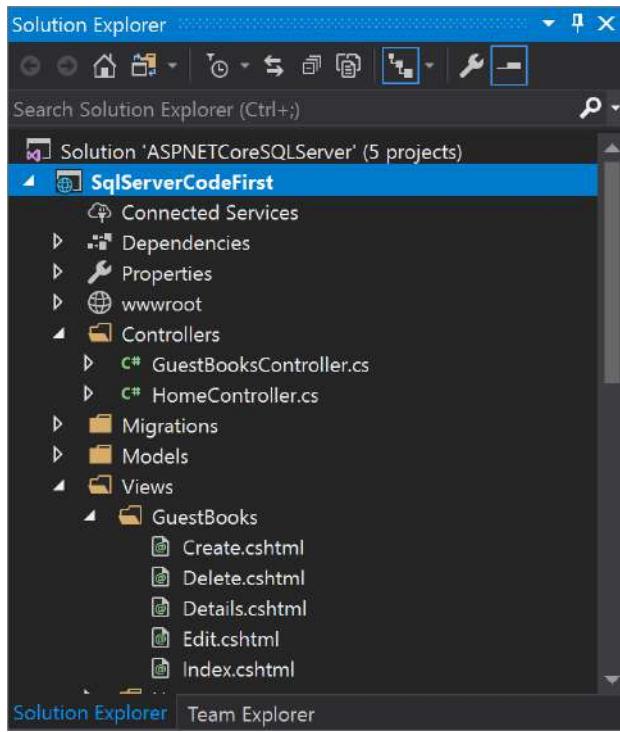
Controller & View

Pada sub bab ini akan dilakukan pembuatan komponen controller dan view dengan cara yang sama dengan pembuatan controller pada sub bab sebelumnya.



Gambar 102. SqlServerCodeFirst - Add MVC Controller with views, using Entity Framework.

Hasilnya dapat dilihat seperti pada gambar di bawah ini.



Gambar 103. SqlServerCodeFirst - Controller & View.

Dapat dilihat pada folder Controllers telah dibuat class GuestBooksController.cs. Kemudian pada folder Views dibuat folder GuestBooks yang berisi file view berikut ini:

- Create.cshtml.
- Delete.cshtml.
- Details.cshtml.
- Edit.cshtml.
- Index.cshtml.

Isi dan antarmuka dari file-file tersebut sama seperti pada project SqlServerDbFirst.

Kesimpulan

Pada bab ini dijelaskan cara kerja ASP.NET Core MVC dan cara-cara untuk melakukan koneksi dan operasi ke database MS SQL Server dari aplikasi web ASP.NET Core MVC dengan menggunakan Entity Framework Core.

Tujuan pembahasan pada bab ini adalah untuk memberikan pengetahuan perbedaan antara pendekatan Database First dan Model First yang secara singkat dapat dirangkum sebagai berikut ini:

1. Pendekatan Database First mempunyai urutan penggerjaan sebagai berikut:
 - o Membuat database dan table.
 - o Membuat connection string untuk koneksi ke database dengan menuliskan nama database yang TELAH dibuat.
 - o Generate komponen model yang terdiri atas class entity model dan data context.
 - o Modifikasi Startup.cs untuk menangani model yang telah dibuat.
 - o Generate komponen controller dan view berdasarkan komponen model yang telah dibuat.

2. Pendekatan Code First mempunyai urutan penggerjaan sebagai berikut:
 - o Membuat komponen model. Karena komponen model adalah class yang terdiri atas kode, maka pendekatan ini disebut sebagai Code First.
 - o Modifikasi Startup.cs untuk menangani model yang telah dibuat.
 - o Membuat connection string untuk koneksi ke database dengan menuliskan nama database yang AKAN dibuat. Agar pembuatan database bisa dilakukan konfigurasi connection string harus menggunakan user yang mempunyai hak akses untuk membuat database. Tapi tentu saja harus hati-hati menggunakan cara ini untuk alasan keamanan.
 - o Membuat class Migration.
 - o Membuat database.
 - o Membuat komponen controller dan view.

Pada sub bab ini belum dijelaskan secara detail tentang tentang kode-kode pada komponen model, view dan controller. Penjelasan detail semua kode yang telah dibuat akan dibahas pada bab selanjutnya.

6

ASP.NET Core Identity

Salah satu cara pengamanan yang umum digunakan adalah dengan melindungi halaman dengan pemeriksaan otentifikasi dengan terlebih dahulu melewati halaman login. Sedangkan otorisasi berfungsi untuk membatasi akses suatu halaman agar hanya dapat diakses oleh user tertentu saja atau user-user dari role tertentu saja.

Pada bab ini akan dijelaskan dan dipraktekan cara melakukan implementasi otentifikasi dan otorisasi dengan ASP.NET Core dan database MS SQL Server.

Pendahuluan

Pada tahun 2005 telah dikenal ASP.NET Membership yang berfungsi untuk mengelola proses otentifikasi dan otorisasi pada aplikasi web. Tetapi secara default, database yang harus digunakan adalah SQL Server. Kemudian dikenal pula ASP.NET Universal Provider yang telah didukung oleh Entity Framework sebagai akses datanya. Tetapi masih memiliki keterbatasan dalam penggunaan database yaitu hanya dapat menggunakan SQL Server saja. Penjelasan lebih lanjut dan contoh implementasi ASP.NET Membership dapat dibaca pada buku berjudul Seri Belajar ASP.NET : Membangun Aplikasi Web Mudah & Cepat dan Seri Belajar ASP.NET : Membangun Sistem Pengelolaan User.

Sekarang ini telah dikenal ASP.NET Identity, yang memungkinkan pengelolaan proses otentifikasi yang lebih bebas. Termasuk kemampuan untuk melakukan otentifikasi atau login dengan memanfaatkan akun pengguna pada social media seperti facebook, twitter atau yang lainnya. ASP.NET Identity dapat digunakan pada semua keluarga ASP.NET Framework seperti ASP.NET MVC, Web Forms, Web Pages, Web API dan SignalR selain itu juga dapat digunakan untuk berbagai platform aplikasi seperti web, phone, store atau aplikasi hybrid.

Kedua library di atas menggunakan skema penyimpanan user dan role berdasarkan aturan yang telah ditentukan oleh library tersebut. Umumnya penyimpanan dilakukan pada database maka penamaan tabel dan field untuk menyimpan user dan role harus mengikuti aturan library tersebut. Umumnya pembuatan tabel-tabel tersebut dilakukan secara otomatis oleh library ini sehingga software developer tidak bisa menambah tabel atau field lain sesuai kebutuhan.

Kelebihan kedua library tersebut adalah kelengkapan fungsi untuk proses otentifikasi dan otorisasi yang dapat langsung digunakan oleh software developer. Sebagai contoh, telah ada method untuk proses SignIn dan SignOut yang langsung dapat digunakan.

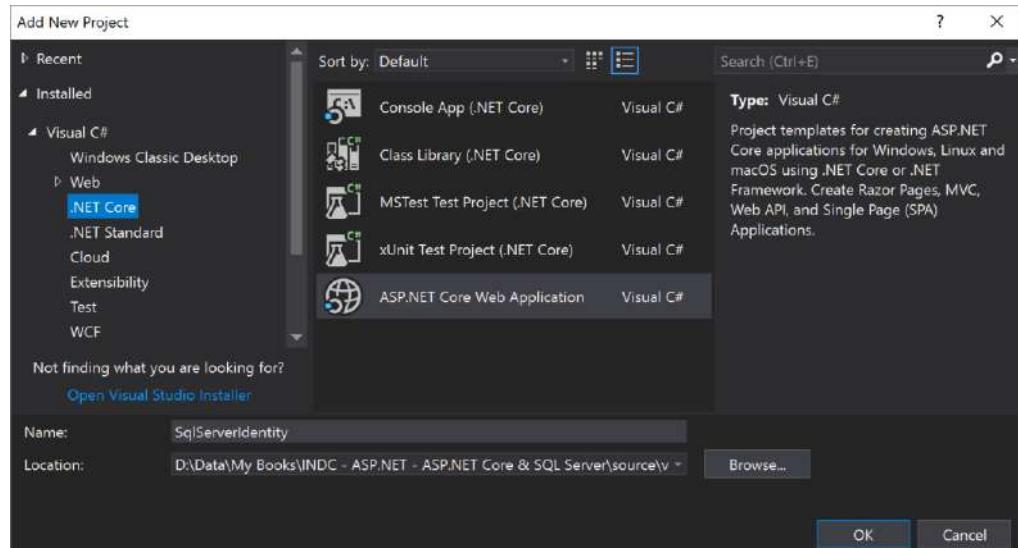
Project

Pada bab ini akan dilakukan pembuatan dan setting project yang mendukung implementasi ASP.NET Core Identity.

Membuat Project

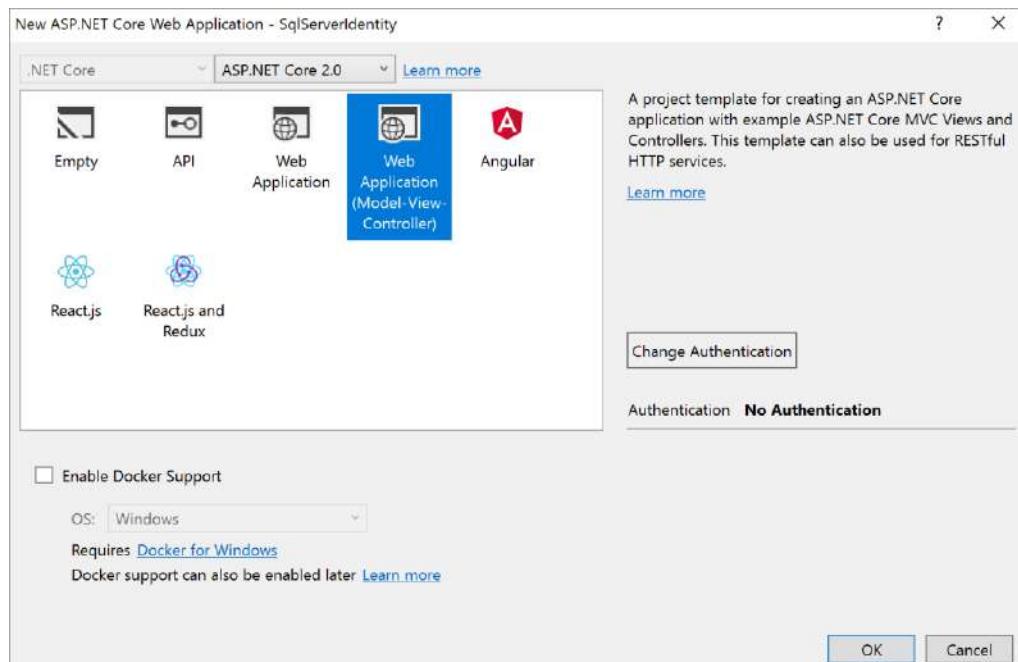
Berikut adalah langkah-langkah untuk membuat project tersebut.

Klik kanan pada solution ASPNETCoreSqlServer pada solution explorer, kemudian pilih Add > New Project.



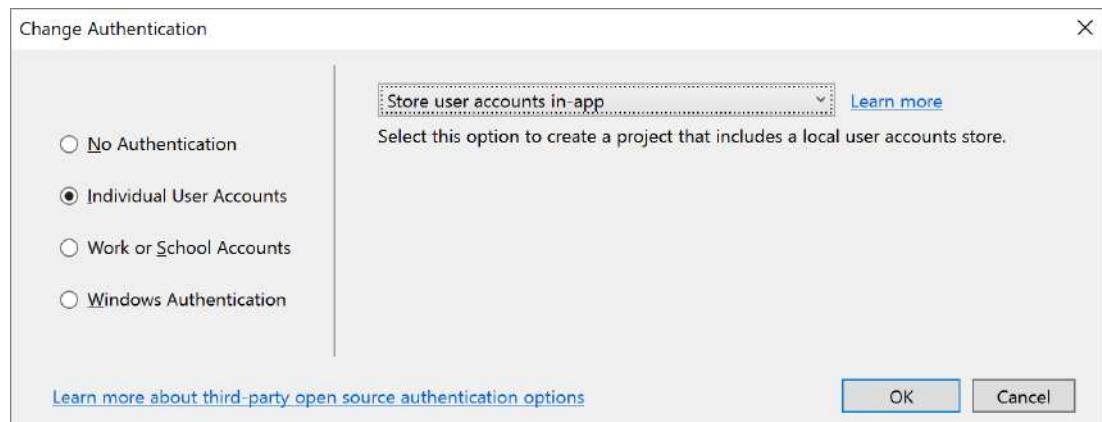
Gambar 104. SqlServerIdentity - Membuat project.

Nama project yang akan dibuat adalah SqlServerIdentity. Akan ditampilkan window New ASP.NET Core Web Application seperti pada gambar di bawah ini.



Gambar 105. SqlServerIdentity - New ASP.NET Core Web Application.

Pada window itu pilih Web Application (Model-View-Controller) kemudian klik tombol Change Authentication. Kemudian ditampilkan window Change Authentication seperti pada gambar di bawah ini.

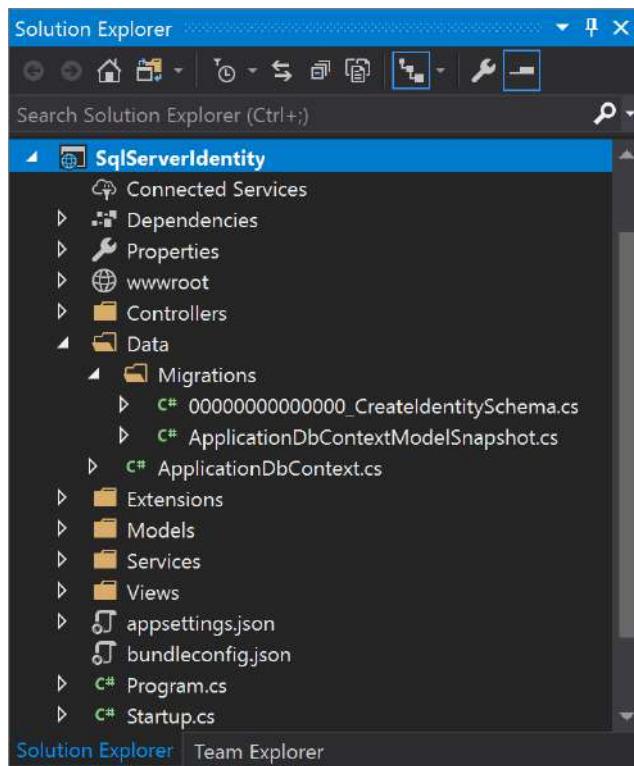


Gambar 106. SqlServerIdentity - Change Authentication.

Pada opsi radiobutton pilih Individual User Accounts. Dan pada opsi dropdownlist pilih Store user accounts in-app. Kemudian klik tombol OK. Dan klik tombol OK lagi pada window New ASP.NET Core Web Application. Kemudian klik kanan pada project SqlServerIdentity kemudian pilih Set as Startup Project.

Sebagai informasi, arti dari "Store user account in-app" adalah database akan dibuat dalam bentuk file yang akan disimpan pada folder di dalam project (umumnya disimpan dalam folder Data).

Sebagai informasi, project ini menggunakan pendekatan Code First. Hal ini dapat dilihat pada folder Migrations pada folder Data di project ini.



Gambar 107. SqlServerIdentity - Folder Migrations.

Konfigurasi Connection String

Berikut adalah isi dari file appsettings.json pada project SqlServerIdentity ini.

```

appsettings.json
{
  "ConnectionStrings": {
    "DefaultConnection": "Server=(localdb)\\mssqllocaldb;Database=aspnet-
SqlServerIdentity-D1748F17-8B1A-468A-8054-
A43FC9402DC3;Trusted_Connection=True;MultipleActiveResultSets=true"
  },
  "Logging": {
    "IncludeScopes": false,
    "LogLevel": {
      "Default": "Warning"
    }
  }
}

```

Pada file appsettings.json di atas dapat dilihat bagaimana cara koneksi database yang akan disimpan dalam bentuk file di dalam project ini. Tetapi karena pembahasan pada buku ini hanya fokus pada penggunaan database MS SQL Server, maka perlu dilakukan penyesuaian konfigurasi connection string. Berikut adalah contoh koneksi ke database MS SQL Server.

```

appsettings.json
{
  "ConnectionStrings": {
    "DefaultConnection": "Server=.\SQLEXPRESS;
Database=SqlServerIdentity;User
Id=sa;Password=MRezaFaisal;Trusted_Connection=True"
  },
  "Logging": {
    "IncludeScopes": false,
    "LogLevel": {
      "Default": "Warning"
    }
  }
}

```

Penjelasan Startup.cs

Jika file Startup.cs pada project sebelumnya (SqlServerDbFirst dan SqlServerCodeFirst) dibandingkan dengan file Startup.cs pada project ini (SqlServerIdentity) maka akan dapat dilihat beberapa perbedaan.

Berikut ini adalah isi file Startup.cs pada project ini.

```

Startup.cs
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Builder;
using Microsoft.AspNetCore.Identity;
using Microsoft.EntityFrameworkCore;
using Microsoft.AspNetCore.Hosting;
using Microsoft.Extensions.Configuration;
using Microsoft.Extensions.DependencyInjection;
using SqlServerIdentity.Data;
using SqlServerIdentity.Models;
using SqlServerIdentity.Services;

namespace SqlServerIdentity
{
  public class Startup

```

```

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

    public IConfiguration Configuration { get; }

    // This method gets called by the runtime. Use this method to add
    // services to the container.
    public void ConfigureServices(IServiceCollection services)
    {
        services.AddDbContext<ApplicationDbContext>(options =>
            options.UseSqlServer(Configuration.GetConnectionString("DefaultConnection")));

        services.AddIdentity<ApplicationUser, IdentityRole>()
            .AddEntityFrameworkStores<ApplicationDbContext>()
            .AddDefaultTokenProviders();

        // Add application services.
        services.AddTransient<IEmailSender, EmailSender>();

        services.AddMvc();
    }

    // This method gets called by the runtime. Use this method to
    // configure the HTTP request pipeline.
    public void Configure(IApplicationBuilder app, IHostingEnvironment env)
    {
        if (env.IsDevelopment())
        {
            app.UseBrowserLink();
            app.UseDeveloperExceptionPage();
            app.UseDatabaseErrorPage();
        }
        else
        {
            app.UseExceptionHandler("/Home/Error");
        }

        app.UseStaticFiles();

        app.UseAuthentication();

        app.UseMvc(routes =>
        {
            routes.MapRoute(
                name: "default",
                template: "{controller=Home}/{action=Index}/{id?}");
        });
    }
}

```

Perbedaan file ini dengan project sebelumnya adalah terletak method ConfigureServices yaitu baris yang dicetak tebal.

```

public void ConfigureServices(IServiceCollection services)
{

```

```

        services.AddDbContext<ApplicationContext>(options =>
            options.UseSqlServer(Configuration.GetConnectionString("DefaultConnection"))
        );

        services.AddIdentity<ApplicationUser, IdentityRole>()
            .AddEntityFrameworkStores<ApplicationContext>()
            .AddDefaultTokenProviders();

        // Add application services.
        services.AddTransient<IEmailSender, EmailSender>();

        services.AddMvc();
    }
}

```

Pada method ini dapat dilihat class data context yang digunakan adalah ApplicationContext. File ini disimpan pada folder Data, berikut adalah isi class ApplicationContext.

ApplicationContext.cs

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Identity.EntityFrameworkCore;
using Microsoft.EntityFrameworkCore;
using SqlServerIdentity.Models;

namespace SqlServerIdentity.Data
{
    public class ApplicationContext : IdentityDbContext<ApplicationUser>
    {
        public ApplicationContext(DbContextOptions<ApplicationContext> options)
            : base(options)
        {
        }

        protected override void OnModelCreating(ModelBuilder builder)
        {
            base.OnModelCreating(builder);
            // Customize the ASP.NET Identity model and override the
            defaults if needed.
            // For example, you can rename the ASP.NET Identity table names
            and more.
            // Add your customizations after calling
            base.OnModelCreating(builder);
        }
    }
}

```

Class data context ini juga berbeda dengan class data context sebelumnya. Pada class data context sebelumnya adalah turunan dari class DbContext. Sedangkan pada class ApplicationContext di atas adalah turunan IdentityDbContext.

Perbedaan file Startup.cs lainnya ada pada baris berikut ini.

services.AddIdentity<ApplicationUser, IdentityRole>() .AddEntityFrameworkStores<ApplicationContext>() .AddDefaultTokenProviders();
--

Kedua perbedaan yang telah disebutkan di atas merupakan cara untuk implementasi ASP.NET Core Identity.

Model

Implementasi ASP.NET Core Identity membutuhkan database dan tabel-tabel untuk menyimpan data user dan role. Pendekatan yang dapat digunakan untuk implementasi adalah Code First, yang artinya diperlukan model untuk mengelola user dan role.

Secara default ASP.NET Core Identity telah memiliki class model tersebut, yaitu:

- `IdentityUser`, class untuk mengelola user.
- `IdentityRole`, class untuk mengelola role.

IdentityUser

Class `IdentityUser` digunakan untuk mengelola user. Class ini memiliki property-property untuk menyimpan nilai yang dimiliki oleh user. Berikut adalah property-property yang dimiliki oleh class ini:

- `Id`.
- `UserName`.
- `NormalizedUserName`.
- `Email`.
- `NormalizedEmail`.
- `EmailConfirmed`.
- `PasswordHash`.
- `SecurityStamp`.
- `ConcurrencyStamp`.
- `PhoneNumber`.
- `PhoneNumberConfirmed`.
- `TwoFactorEnabled`.
- `LockoutEnd`.
- `LockoutEnabled`.
- `AccessFailedCount`.

Selain itu class ini juga memiliki property-property yang berfungsi sebagai relasi dengan model lain, yaitu:

- `Roles`.
- `Claims`.
- `Logins`.

Selain property class ini juga memiliki method-method sebagai berikut:

- `ToString()`.
- `Equals(Object)`.
- `Equals(Object, Object)`.
- `ReferenceEquals(Object, Object)`.
- `GetHashCode()`.
- `GetType()`.
- `MemberwiseClone()`.

Property-property di atas selain property-property yang berfungsi sebagai relasi akan menjadi atribut-atribut pada tabel `AspNetUsers`. Jika ingin menambahkan atribut pada tabel maka cukup menambahkan property. Cara menambahkan property tidak bisa langsung dilakukan pada class ini, tetapi dapat dilakukan dengan cara membuat class baru yang mewarisi class `IdentityUser`. Sebagai contoh dibuat class `ApplicationUser` sebagai berikut.

```

ApplicationUser.cs
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Identity;

namespace SqlServerIdentity.Models
{
    // Add profile data for application users by adding properties to the
    ApplicationUser class
    public class ApplicationUser : IdentityUser
    {
        public string FullName { get; set; }
        public string Address { get; set; }

    }
}

```

Untuk membuat class ApplicationUser mewarisi sifat-sifat class IdentityUser dapat dilihat pada baris yang dicetak tebal. Kemudian dapat dilihat dua property tambahan berikut ini:

- FullName.
- Address.

Agar class ini digunakan oleh ASP.NET Core Identity untuk melakukan migrasi kode dan database maka class ini harus didaftarkan pada method ConfigureServices di class Startup.cs seperti yang telah dibahas pada sub bab sebelumnya.

IdentityRole

Class IdentityRole berfungsi untuk mengelola role. Class ini memiliki property-property untuk menyimpan nilai yang dimiliki oleh role. Berikut adalah property-property yang dimiliki oleh class ini:

- Users.
- Id.
- Name.
- NormalizedName.
- ConcurrencyStamp.

Selain itu class ini juga memiliki property-property yang berfungsi sebagai relasi dengan model lain, yaitu:

- Claims.

Selain property class ini juga memiliki method-method sebagai berikut:

- ToString().
- Equals(Object).
- Equals(Object, Object).
- ReferenceEquals(Object, Object).
- GetHashCode().
- GetType().
- MemberwiseClone().

Property-property pada kelompok pertama akan menjadi atribut-atribut pada tabel AspNetRoles. Seperti halnya penjelasan class IdentityUser, class IdentityRole juga dapat

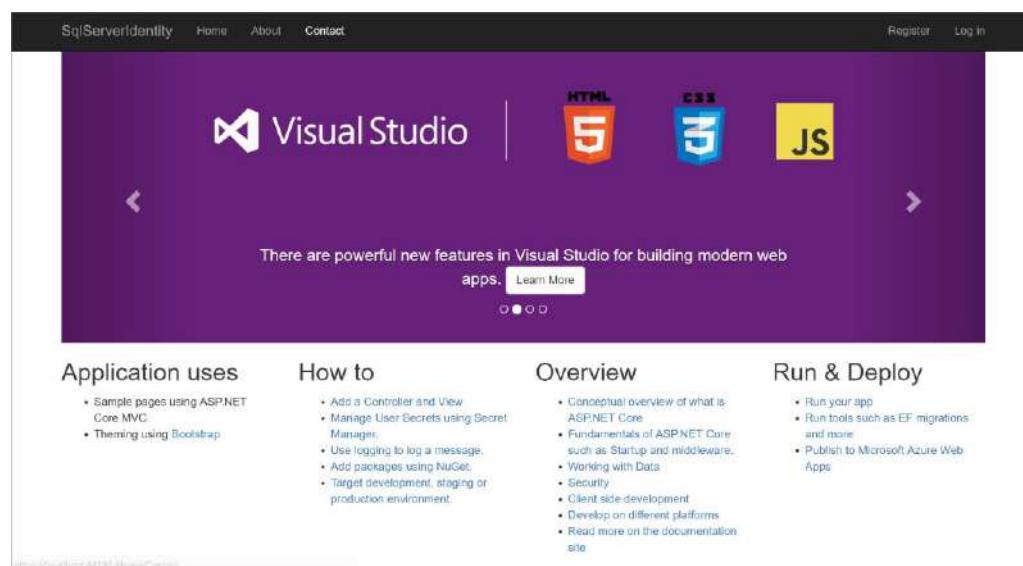
dimodifikasi untuk menambahkan atribut-atribut pada tabel AspNetRoles. Cara modifikasinya adalah dengan membuat class turunan dari class IdentityRoles.

Database

Karena class Migrations telah dibuat secara otomatis saat membuat project, maka selanjutnya adalah update database untuk membuat database beserta table-tablenya. Pada bab sebelumnya telah diperkenalkan cara update database dengan perintah Update-Database. Tetapi pada sub bab ini akan dicontohkan cara update database tanpa cara tersebut.

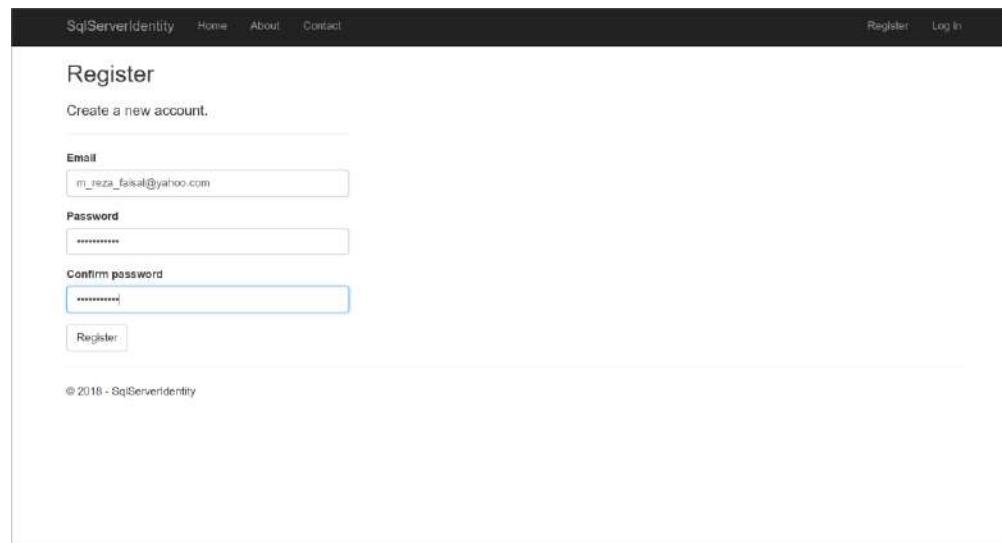
Migrasi

Caranya adalah dengan langsung menjalankan project SqlServerIdentity dengan cara klik kanan pada project kemudian pilih Debug > Start new instance. Berikut adalah antarmuka aplikasi web ini.



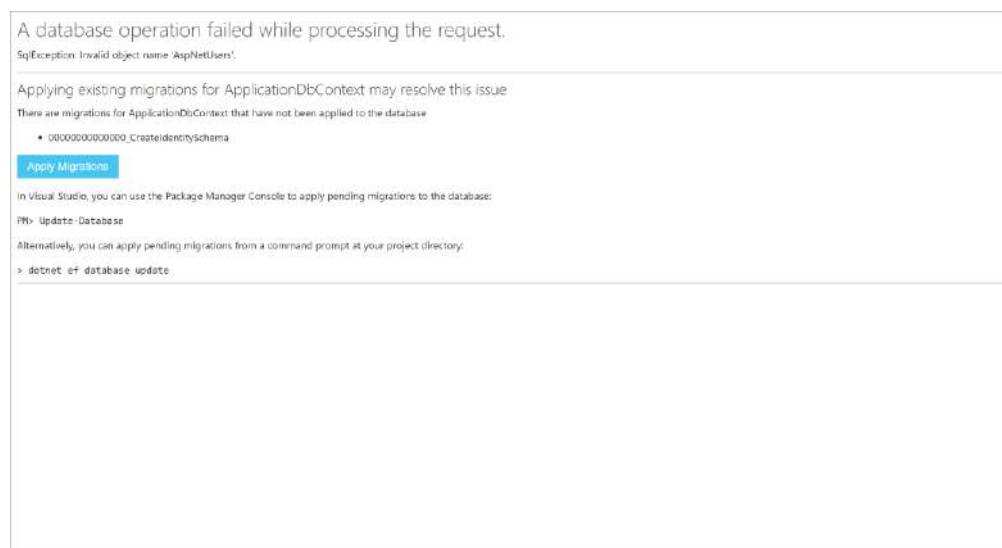
Gambar 108. SqlServerIdentity - Antarmuka aplikasi web.

Kemudian klik link Register pada pojok kanan atas.



Gambar 109. SqlServerIdentity - Form registrasi.

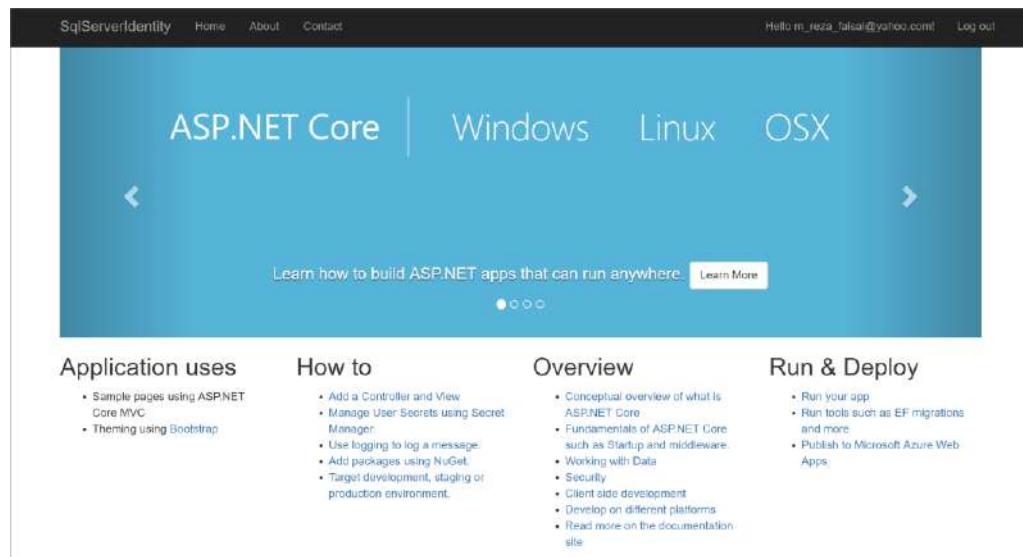
Harap diperhatikan password yang dimasukkan merupakan kombinasi alphanumerik dan karakter spesial. Kemudian klik tombol Register.



Gambar 110. SqlServerIdentity - Migrations.

Aplikasi web akan mencoba melakukan akses ke database, tetapi karena database dan table-table belum dibuat maka aplikasi web akan menawarkan untuk melakukan proses migrasi seperti pada gambar di atas. Klik tombol Apply Migrations, kemudian refresh halaman web ini.

Selanjutnya dapat dilihat proses registrasi telah berhasil dan user telah login.

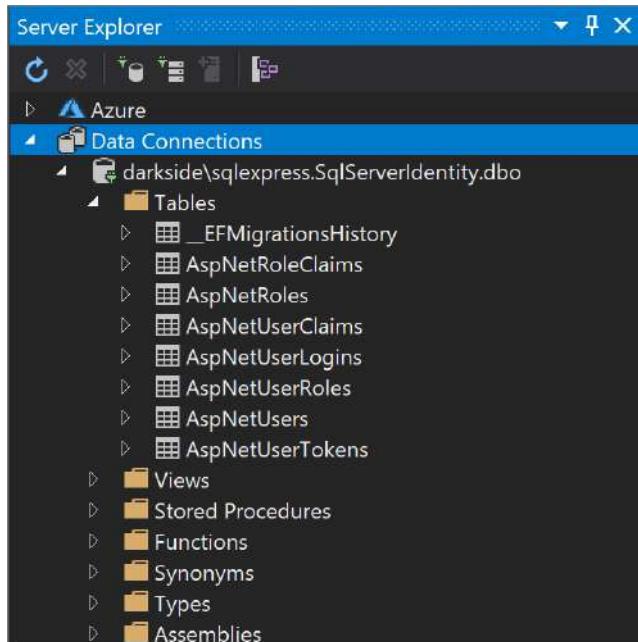


Gambar 111. SqlServerIdentity - Login berhasil.

Jika ingin melihat database dan table yang telah dibuat, dapat dilakukan dengan melakukan koneksi ke database dengan memanfaatkan fitur Connect to Database dari Visual Studio 2017.

Table

Berikut ini adalah table-table yang dibuat pada database SqlServerIdentity.



Gambar 112. SqlServerIdentity - Database & table.

Pada gambar di atas dapat dilihat table-table yang digunakan oleh ASP.NET Core Identity untuk proses otentikasi dan otorisasi. Table-table itu adalah sebagai berikut:

- AspNetRoles dan AspNetRoleClaims berfungsi untuk menyimpan informasi terkait dengan role. Role adalah kelompok atau group yang dimiliki oleh user. sebagai contoh user dapat dikelompokkan dalam group Admin atau User.
- AspNetUsers, AspNetUserLogins, AspNetUserClaims dan AspNetUserTokens berfungsi untuk menyimpan informasi terkait dengan user.

- AspNetUserRoles berfungsi untuk menyimpan relasi antara user dengan role. Pada ASP.NET Identity, seorang user dapat memiliki 1 atau lebih role. Sebagai contoh, seorang user dapat memiliki role sebagai Admin saja. Atau memiliki role sebagai Admin dan User sekaligus.

7

Model-View-Controller

Pada bab ini akan diterangkan detail tentang komponen-komponen pada MVC pada ASP.NET Core MVC dengan cara membuat aplikasi web book store. Aplikasi ini merupakan aplikasi web sederhana tetapi dengan antarmuka seperti aplikasi web yang umum dibangun saat ini. Aplikasi book store ini akan menggunakan pembangunan aplikasi dengan pendekatan Code First dan implementasi ASP.NET Identity.

Persiapan

Aplikasi Book Store

Aplikasi ini akan memiliki fitur-fitur sebagai berikut:

1. Mengelola role.
Fitur ini dapat digunakan untuk menampilkan, menambah, mengedit dan menghapus role.
2. Mengelola user.
Fitur ini digunakan untuk menampilkan, menambah, mengedit dan menghapus user.
3. Mengelola pengarang buku.
Fitur ini dapat digunakan untuk menampilkan, menambah, mengedit dan menghapus data pengarang buku.
4. Mengelola kategori buku.
Fitur ini dapat digunakan untuk menampilkan, menambah, mengedit dan menghapus data kategori buku.
5. Mengelola buku.
Fitur ini dapat digunakan untuk menampilkan, menambah, mengedit dan menghapus data buku.

Template Aplikasi Web

Aplikasi web ini akan dibangun dengan menggunakan template Bootstrap yang tersedia gratis.



Gambar 113. Template admin Gentellela.

Tujuan dari penggunaan template ini agar pembaca nantinya memiliki pengetahuan bagaimana memodifikasi template tersebut agar bisa digunakan dalam membangun aplikasi web dengan ASP.NET Core.

Template admin yang digunakan pada buku ini adalah Gentellela yang menggunakan framework Bootstrap 3. Template ini dapat diunduh dari link berikut <https://github.com/puikinsh/gentelella>. Cara untuk mengubah template ini agar bisa digunakan pada aplikasi web ASP.NET Core akan dijelaskan pada sub bab View.

Membuat Project

Nama project yang akan dibuat adalah SqlServerBookStore. Pembuatan project sama seperti langkah-langkah pembuatan project pada bab ASP.NET Core Identity. Setelah project dibuat klik kanan pada project kemudian Set as Startup Project.

Modifikasi appsettings.json

Modifikasi appsettings.json dilakukan untuk mengubah koneksi ke database MS SQL Server.

```
appsettings.json
{
  "ConnectionStrings": {
    "DefaultConnection": "Server=.\SQLExpress;
Database=SqlServerBookStore;User
Id=sa;Password=MRezaFaisal;Trusted_Connection=True"
  },
  "Logging": {
    "IncludeScopes": false,
    "LogLevel": {
      "Default": "Warning"
    }
  }
}
```

Modifikasi Startup.cs

Modifikasi file Startup.cs dilakukan untuk menunjukkan cara melakukan konfigurasi ASP.NET Core Identity. Berikut ini adalah kode file Startup.cs sebelum dimodifikasi.

Startup.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Builder;
using Microsoft.AspNetCore.Identity;
using Microsoft.EntityFrameworkCore;
using Microsoft.AspNetCore.Hosting;
using Microsoft.Extensions.Configuration;
using Microsoft.Extensions.DependencyInjection;
using SqlServerBookStore.Data;
using SqlServerBookStore.Models;
using SqlServerBookStore.Services;

namespace SqlServerBookStore
{
    public class Startup
    {
        public Startup(IConfiguration configuration)
        {
            Configuration = configuration;
        }

        public IConfiguration Configuration { get; }

        // This method gets called by the runtime. Use this method to add
        services to the container.
        public void ConfigureServices(IServiceCollection services)
        {
            services.AddDbContext<ApplicationContext>(options =>
options.UseSqlServer(Configuration.GetConnectionString("DefaultConnection"))
);

            services.AddIdentity<ApplicationUser, IdentityRole>()
                .AddEntityFrameworkStores<ApplicationContext>()
                .AddDefaultTokenProviders();

            // Add application services.
            services.AddTransient<IEmailSender, EmailSender>();

            services.AddMvc();
        }

        // This method gets called by the runtime. Use this method to
        configure the HTTP request pipeline.
        public void Configure(IApplicationBuilder app, IHostingEnvironment env)
        {
            if (env.IsDevelopment())
            {
                app.UseBrowserLink();
                app.UseDeveloperExceptionPage();
                app.UseDatabaseErrorPage();
            }
            else
        }
    }
}
```

```

    {
        app.UseExceptionHandler("/Home/Error");
    }

    app.UseStaticFiles();

    app.UseAuthentication();

    app.UseMvc(routes =>
    {
        routes.MapRoute(
            name: "default",
            template: "{controller=Home}/{action=Index}/{id?}");
    });
}
}

```

Bagian yang dimodifikasi adalah bagian ini.

```

services.AddIdentity<ApplicationUser, IdentityRole>()
    .AddEntityFrameworkStores<ApplicationContext>()
    .AddDefaultTokenProviders();

```

Bagian tersebut akan diganti menjadi sebagai berikut.

```

services.AddIdentity<ApplicationUser, ApplicationRole>(p => {
    p.Password.RequireDigit = false;
    p.Password.RequireLowercase = false;
    p.Password.RequireUppercase = false;
    p.Password.RequireNonAlphanumeric = false;
    p.Password.RequiredLength = 6;
})
    .AddEntityFrameworkStores<ApplicationContext>()
    .AddDefaultTokenProviders();

```

Perbedaan kode di atas adalah perubahan penggunaan class model IdentityRole menjadi class model ApplicationRole. Class model ApplicationRole adalah modifikasi class IdentityRole yang bertujuan untuk menambahkan property. Pembuatan class model ApplicationRole akan dilakukan pada sub bab Model.

Perbedaan lainnya adalah konfigurasi format password. Pada konfigurasi itu format password dengan panjang minimal 6 karakter, ijin untuk tidak menggunakan angka dan karakter spesial dan ijin untuk tidak menggunakan kombinasi huruf besar dan kecil.

Catatan

Untuk proses modifikasi file class data context dan pembuatan class entity model (ApplicationUser dan ApplicationRole) akan dilakukan di akhir pembahasan Model. Kemudian dilanjutkan proses pembuatan kode migrasi. Kemudian dilanjutkan dengan proses pembuatan database dan tabel secara otomatis.

Model

Pada bab Pengenalan ASP.NET Core MVC sub bab Cara Kerja ASP.NET Core MVC, telah dibuat sebuah class model yaitu class GuestBook. Pada contoh pertama, class model GuestBook hanya digunakan untuk melakukan pertukaran data antar komponen MVC tanpa ada pengambilan dan penyimpanan data dari dan ke database. Kemudian pada bab Entity

Framework Core & MS SQL Server, diberikan contoh dimana class model GuestBook dimodifikasi agar memiliki property-property yang sesuai dengan atribut-atribut pada tabel. Kemudian class model tersebut dapat digunakan untuk membuat tabel dengan bantuan tool pada Entity Framework Core. Selainnya class model digunakan untuk menampung data dari database untuk ditampilkan pada komponen view. Class ini juga digunakan untuk menampung data dari komponen view untuk akhirnya disimpan ke dalam database.

Dari contoh-contoh tersebut, dalam bahasa sederhana maka dapat disimpulkan class model berfungsi sebagai class yang digunakan untuk membuat tabel dan penampung data dalam proses operasi data.

Pada bab ini akan dijelaskan bagaimana membuat model yang mendukung pendekatan Code First dimana class model digunakan untuk melakukan migrasi database dan membuat tabel. Karena pada umumnya tabel yang digunakan pada suatu aplikasi memiliki relasi, maka pada sub bab ini akan diberikan membuat class model yang memiliki relasi sehingga saat proses migrasi database akan dibuat tabel yang juga memiliki relasi.

Kemudian juga akan diberikan fungsi dan manfaat lain yang dimiliki oleh class model yaitu:

- Nilai label yang ditampilkan pada form atau header tabel pada komponen view.
- Validasi nilai yang dimasukkan pada form sebelum diproses dan disimpan ke dalam database.

Sebagaimana pada bab sebelumnya, class-class model yang dibuat pada sub bab ini akan disimpan di dalam folder Models pada project.

API

Pada bab sebelumnya telah diperlihatkan cara membuat komponen model dalam bentuk class model yaitu class GuestBook. Class model GuestBook adalah class model yang sederhana.

Pada bab ini akan dijelaskan cara untuk membuat class model dengan menggunakan konfigurasi yang lebih lengkap untuk membangun komponen model yang mendukung relational database, validasi dan label pada antarmuka komponen view.

Untuk melakukan konfigurasi tersebut dapat digunakan dua library atau API, yaitu:

1. Data annotations.
2. Fluent API.

Data Annotations

Cara konfigurasi dengan menggunakan data annotations dilakukan dengan menambahkan atribut pada class model. Untuk menambahkan atribut pada class model, maka di class tersebut harus menggunakan namespace berikut ini.

```
using System.ComponentModel.DataAnnotations;
using System.ComponentModel.DataAnnotations.Schema;
```

Atribut yang dapat diberikan pada class adalah:

1. Atribut untuk pemetaan antara class model dengan tabel.
2. Atribut untuk pemetaan property class dengan atribut tabel.
3. Atribut untuk menentukan property class sebagai primary key.

Fluent API

Konfigurasi class model dengan Fluent API dilakukan pada file class terpisah, yaitu pada class data context. Cara merupakan bagian dari Entity Framework, sehingga class data context harus menggunakan namespace berikut ini.

```
using Microsoft.EntityFrameworkCore;
```

Fluent API mempunyai fitur konfigurasi yang lebih lengkap, selain memiliki fitur seperti yang dimiliki oleh data annotation, fitur-fitur lainnya adalah:

1. Mendukung computed column atau kolom yang nilainya dihitung atau ditentukan pada database.
2. Mendukung sequence.
3. Mendukung default value atau pemberian nilai default jika nilai suatu kolom tidak diberikan.
4. Mendukung index.
5. Mendukung foreign key constraint yang digunakan untuk relasi antar model.
6. Mendukung lebih dari satu primary key.

Cara konfigurasi dengan menggunakan data annotations dan Fluent API akan dijelaskan dan praktikkan pada sub bab berikutnya.

Tipe Class Model

Penulis membagi class model menjadi dua tipe, yaitu:

1. Class entity model yang selain berfungsi untuk menampilkan data ke komponen view tetapi juga dapat digunakan sebagai object untuk disimpan ke dalam database. Class model ini biasanya dibuat sebagai representasi dari sebuah tabel pada database. Class ini juga dapat digunakan untuk membuat tabel pada database pada pendekatan Code First.
2. Class view model yang berfungsi untuk menampilkan data ke komponen view saja.

Kedua tipe class tersebut memiliki kesamaan yaitu sebagai media yang digunakan untuk pertukaran data antara komponen view dan controller. Tipe class model pertama telah diperkenalkan pada buku ini, yaitu class model GuestBook. Sedangkan untuk class view model akan dijelaskan pada sub bab ini.

Class Entity Model

Class entity model atau class entity adalah sebagai representasi dari tabel yang ada di database, sehingga property-property class ini merupakan representasi dari atribut-atribut dari tabel. Untuk pendekatan Code First, nama class akan menjadi nama tabel dan nama atribut yang digunakan pada class entity ini akan menjadi nama atribut pada tabel tersebut.

Berikut ini adalah 4 class entity model yang digunakan pada aplikasi Book Store yaitu:

- Category.cs.
- Book.cs.
- Author.cs.
- BookAuthor.cs.

Keempat class ini akan disimpan pada folder Models. Berikut adalah isi kode lengkap dari keempat class tersebut. Di bawah ini adalah kode lengkap class entity model Category.

```
Category.cs
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using System.ComponentModel.DataAnnotations;

namespace SqlServerBookStore.Models
{
    public partial class Category
    {
        public int CategoryID { set; get; }
        public String Name { set; get; }
        public ICollection<Book> Books { get; set; }
    }
}
```

Class Category akan dibuat menjadi tabel dengan nama bentuk jamaknya yaitu Categories. Jika ingin menentukan nama tabel sendiri maka perlu ditambahkan pada class data context. Class ini memiliki 3 property yaitu:

- CategoryID, property ini adalah primary key. Dengan menambahkan "ID" di akhir nama suatu property maka secara otomatis menjadikan primary key. Jika tipe data yang digunakan untuk primary key adalah integer, maka secara otomatis nilainya memiliki sifat auto increment.
- Name, property untuk menyimpan nilai nama kategori buku.
- Books, property ini tidak akan menjadi atribut pada tabel Categories. Property ini berfungsi sebagai relasi dengan model Book yang akan dibuat setelah ini. Dengan melihat property ini dapat diketahui jika sebuah object Category dapat digunakan oleh banyak object Book.

Berikut adalah kode lengkap class entity model Book.

```
Book.cs
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using System.ComponentModel.DataAnnotations;
using System.ComponentModel.DataAnnotations.Schema;

namespace SqlServerBookStore.Models
{
    public partial class Book
    {
        [DatabaseGenerated(DatabaseGeneratedOption.None)]
        public int BookID { set; get; }
        public int CategoryID { set; get; }
        public Category Category { get; set; }
        public String Title { set; get; }
        public String Photo { set; get; }
        public DateTime PublishDate { set; get; }
        public double Price { set; get; }
        public int Quantity { set; get; }
        public ICollection<BookAuthor> BooksAuthors { get; set; }
    }
}
```

Class Book akan dibuat menjadi tabel dengan nama bentuk jamaknya yaitu Books. Class ini memiliki property-property sebagai berikut:

- BookID, property ini adalah primary key dari tabel Books. Dapat dilihat di atas property ini terdapat atribut [DatabaseGenerated(DatabaseGeneratedOption.None)], atribut tersebut berfungsi agar nilai tidak menjadi auto increment. Sebagai informasi, DatabaseGeneratedOption memiliki 3 opsi yaitu None, Identity dan Computed. Untuk opsi Identity, nilai akan dibuat secara otomatis ketika data ditambahkan pertama kali. Opsi ini cocok digunakan untuk property primary key. Opsi Computed, nilai akan dibuat secara otomatis ketika data ditambahkan atau diupdate.
- CategoryID, property ini menjadi foreign key yang akan menyimpan nilai id dari tabel Categories.
- Category, property ini tidak akan menjadi atribut pada tabel Book. Property ini berfungsi sebagai relasi dengan model Category. Relasi yang terjadi dengan membuat property CategoryID menjadi foreign key.
- Title, property untuk menyimpan judul buku.
- Photo, property untuk menyimpan nama file foto.
- PublishDate, property untuk menyimpan tanggal publish buku.
- Price, property untuk menyimpan harga buku.
- Quantity, property untuk menyimpan jumlah stok buku.
- BooksAuthors, property ini tidak akan menjadi atribut pada tabel Books. Property ini berfungsi sebagai relasi dengan model BookAuthor yang akan dibuat setelah ini. Dengan melihat property ini dapat diketahui jika sebuah object Book dapat digunakan oleh banyak object BookAuthor.

Berikut adalah kode lengkap class entity model Author.

```
Author.cs
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using System.ComponentModel.DataAnnotations;

namespace SqlServerBookStore.Models
{
    public partial class Author
    {
        public int AuthorID { set; get; }
        public String Name { set; get; }
        public String Email { set; get; }
        public ICollection<BookAuthor> BooksAuthors { get; set; }
    }
}
```

Class Author akan dibuat menjadi tabel dengan nama bentuk jamaknya yaitu Authors. Class ini memiliki property-property sebagai berikut:

- AuthorID, property ini adalah primary key dari tabel Authors.
- Name, property untuk menyimpan nama pengarang.
- Email, property untuk menyimpan email pengarang.
- BooksAuthors, property ini tidak akan menjadi atribut pada tabel Authors. Property ini berfungsi sebagai relasi dengan model BookAuthor. Dengan melihat property ini dapat diketahui jika sebuah object Author dapat digunakan oleh banyak object BookAuthor.

Berikut ini adalah kode lengkap class BookAuthor.

```
BookAuthor.cs
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using System.ComponentModel.DataAnnotations;

namespace SqlServerBookStore.Models
{
    public partial class BookAuthor
    {
        public int BookAuthorID { set; get; }
        public int BookID { set; get; }
        public Book Book { get; set; }
        public int AuthorID { set; get; }
        public Author Author { get; set; }
    }
}
```

Class BookAuthor akan dibuat menjadi tabel BooksAuthors, nama ini sesuai dengan nama property BooksAuthor yang dimiliki oleh class Book dan class Author. Class ini memiliki property-property sebagai berikut:

- BookAuthorID yang menjadi primary key.
- BookID, property ini menjadi foreign key yang akan menyimpan nilai id dari tabel Books.
- Book, property ini tidak akan menjadi atribut pada tabel BooksAuthors. Property ini berfungsi sebagai relasi dengan model Book. Relasi yang terjadi dengan membuat property BookID menjadi foreign key.
- AuthorID, property ini menjadi foreign key yang akan menyimpan nilai id dari tabel Authors.
- Author, property ini tidak akan menjadi atribut pada tabel BooksAuthors. Property ini berfungsi sebagai relasi dengan model Author. Relasi yang terjadi dengan membuat property AuthorID menjadi foreign key.

Dari keterangan di atas dapat dilihat jika model ini akan membuat tabel yang memiliki atribut dengan sebuah primary key dan dua foreign key.

Class entity model yang lain adalah class turunan IdentityUser dan IdentityRole. Nama class turunan tersebut adalah ApplicationUser dan ApplicationRole. Kedua class model ini untuk mengelola data user dan role pada ASP.NET Core Identity. Keduanya telah dijelaskan pada bab ASP.NET Core Identity.

Berikut adalah kode lengkap class ApplicationUser.cs

```
 ApplicationUser.cs
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Identity;

namespace SqlServerBookStore.Models
{
    // Add profile data for application users by adding properties to the
    ApplicationUser class
    public class ApplicationUser : IdentityUser
```

```

{
    public string FullName { get; set; }
    public string Photo { get; set; }

}

```

Dari class di atas dapat dilihat class ApplicationUser akan memiliki property-property yang dimiliki oleh class IdentityUser ditambah dengan property berikut ini:

- FullName, property untuk menyimpan nama lengkap dari user.
- Photo, property untuk menyimpan nama file foto beserta url lokasi penyimpanannya.

Berikut ini adalah kode lengkap class ApplicationRole.

```

ApplicationRole.cs
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Identity;

namespace SqlServerBookStore.Models
{
    public class ApplicationRole : IdentityRole
    {
        public string Description { get; set; }
    }
}

```

Dari kode di atas dapat dilihat bahwa class ApplicationRole memiliki semua property yang dimiliki oleh class IdentityRole ditambah dengan property Description yang berfungsi untuk menyimpan keterangan dari role.

Selanjutnya adalah mendaftarkan class-class di atas ke dalam class data context. Pada project SqlServerBookStore telah dimiliki class data context dengan nama ApplicationDbContext.cs yang berada di dalam folder Data. Penambahan pada class ini merupakan syarat dari class model agar dapat disebut sebagai class entity model.

Berikut adalah isi class ini setelah dimodifikasi.

```

ApplicationDbContext.cs
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Identity.EntityFrameworkCore;
using Microsoft.EntityFrameworkCore;
using SqlServerBookStore.Models;

namespace SqlServerBookStore.Data
{
    public class ApplicationDbContext : IdentityDbContext<ApplicationUser,
    ApplicationRole, string>
    {
        public ApplicationDbContext(DbContextOptions<ApplicationDbContext>
options)
            : base(options)
        {

        }

        public virtual DbSet<Category> Categories { get; set; }
        public virtual DbSet<Author> Authors { get; set; }
    }
}

```

```

public virtual DbSet<Book> Books { get; set; }
public virtual DbSet<BookAuthor> BooksAuthors { get; set; }

protected override void OnModelCreating(ModelBuilder builder)
{
    base.OnModelCreating(builder);
    // Customize the ASP.NET Identity model and override the
    defaults if needed.
    // For example, you can rename the ASP.NET Identity table names
    and more.
    // Add your customizations after calling
    base.OnModelCreating(builder);
}
}
}

```

Berikut adalah keterangan dari kode class di atas. Agar class-class pada folder Models dapat dipanggil pada class ini maka perlu digunakan baris berikut ini.

```
using SqlServerBookStore.Models;
```

Kemudian untuk menambahkan class ApplicationUser dan ApplicationRole agar dapat dikelola oleh class ini dengan media penyimpanan yang sama maka digunakan kode berikut ini.

```
public class ApplicationDbContext : IdentityDbContext<ApplicationUser,
ApplicationRole, string>
```

Pada kode di atas dapat dilihat cara pewarisan class IdentityDbContext dengan parameter masukan berupa class ApplicationUser dan ApplicationRole.

Selanjutnya adalah membuat class entity model Category, Author, Book dan BookAuthor dengan cara sebagai berikut.

```

public virtual DbSet<Category> Categories { get; set; }
public virtual DbSet<Author> Authors { get; set; }
public virtual DbSet<Book> Books { get; set; }
public virtual DbSet<BookAuthor> BooksAuthors { get; set; }

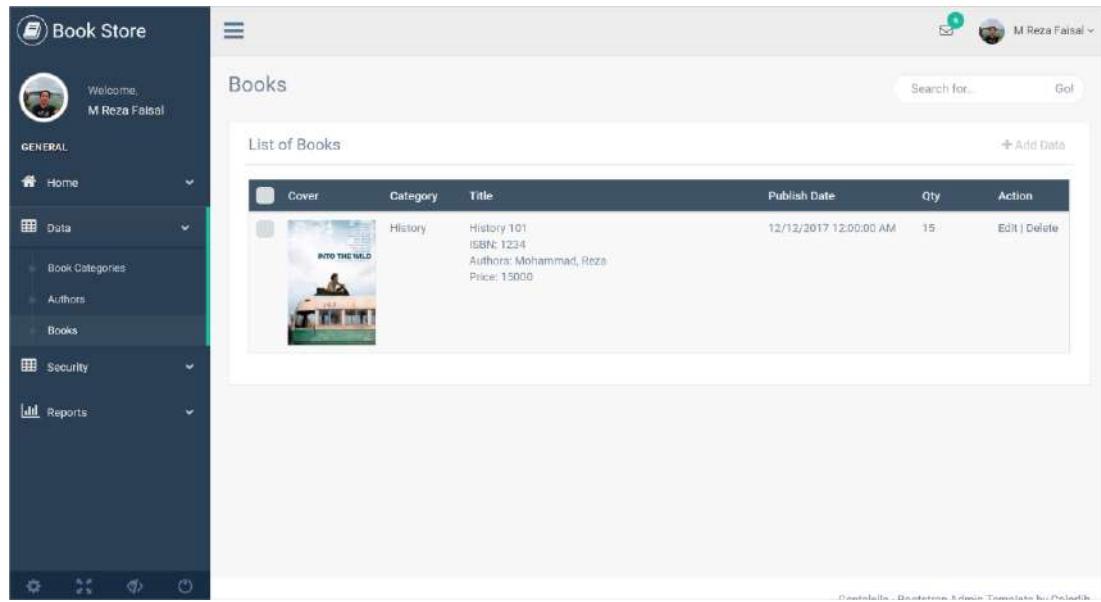
```

Dari kode di atas maka dapat dilihat untuk melakukan operasi database pada tabel Categories, Authors Books dan BooksAuthors dapat dilakukan dengan menggunakan keempat property tersebut.

Class View Model

Class view model dibuat dengan tujuan untuk menampilkan data pada komponen view, baik dalam bentuk tabel atau form untuk input data. Class view model dibuat saat class entity model tidak bisa digunakan untuk menampilkan data atau pada form input yang sesuai dengan keinginan.

Fitur aplikasi web book store yang memerlukan class view model adalah fitur mengelola buku. Pada gambar di bawah ini dapat dilihat antarmuka untuk menampilkan daftar buku pada komponen view. Class view model diperlukan untuk menampilkan daftar buku yang disertai dengan kategori buku dan daftar nama-nama pengarang buku tersebut (jika pengarangnya lebih dari satu). Jika diperhatikan pada table Books ataupun class entity model Book tidak ada field yang menyimpan nama pengarang. Sehingga jika hanya menggunakan class Book, maka tidak akan ada informasi nama pengarang buku yang akan ditampilkan pada komponen view.



Gambar 114. Book Store - Daftar buku.

Table BooksAuthors adalah table yang menyimpan relasi antara nama pengarang dan buku, tetapi isinya hanya BookID dan AuthorID saja. Sehingga class BookAuthor juga tidak bisa digunakan untuk menampilkan informasi seperti yang diinginkan di atas.

Untuk menampilkan informasi tersebut perlu digunakan sebuah model yang dapat digunakan untuk menampung informasi dari 4 table sekaligus yaitu table Books, BooksAuthors, Authors dan Categories.

Berikut ini adalah class view model yang akan digunakan untuk menampilkan data buku.

```
BookViewModel.cs
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;

namespace SqlServerBookStore.Models
{
    public partial class BookViewModel
    {
        public int ISBN { set; get; }
        public String CategoryName { set; get; }
        public String Title { set; get; }
        public String Photo { set; get; }
        public DateTime PublishDate { set; get; }
        public double Price { set; get; }
        public int Quantity { set; get; }
        public string AuthorNames { set; get; }
    }
}
```

Sebagai mana yang telah diketahui bahwa class entity model merupakan representasi sebuah tabel pada database, sehingga harus didaftarkan pada class data context. Sedangkan class view model bukan representasi dari tabel yang ada di database, sehingga class model ini tidak perlu ditambahkan sebagai property pada class data context. Setelah class view model menjadi object maka object ini hanya akan digunakan untuk menampung data yang didapat object dari class entity model.

Berikut adalah kode lengkap bagaimana object dari class BookViewModel menampung data dari object class entity model Author, Book dan Category. Kode ini adalah bagian dari class controller BookController.cs. Pembahasan secara lengkap tentang komponen controller akan dibahas pada sub bab Controller.

```
[HttpGet]
public IActionResult Index()
{
    var bookList = db.Books
        .Include(c => c.Category)
        .Include(ba => ba.BooksAuthors)
        .ThenInclude(a => a.Author)
        .ToList();

    IList<BookViewModel> items = new List<BookViewModel>();
    foreach(Book book in bookList){
        BookViewModel item = new BookViewModel();

        item.ISBN = book.BookID;
        item.Title = book.Title;
        item.Photo = book.Photo;
        item.PublishDate = book.PublishDate;
        item.Price = book.Price;
        item.Quantity = book.Quantity;
        item.CategoryName = book.Category.Name;

        string authorNameList = string.Empty;
        var booksAuthorsList = book.BooksAuthors;
        foreach(BookAuthor booksAuthors in booksAuthorsList){
            var author = booksAuthors.Author;
            authorNameList = authorNameList + author.Name + ", ";
        }
        item.AuthorNames = authorNameList.Substring(0, authorNameList.Length
- 2);

        items.Add(item);
    }

    return View(items);
}
```

Langkah pertama adalah mengambil daftar buku dari database dengan cara sebagai berikut.

```
var bookList = db.Books
    .Include(c => c.Category)
    .Include(ba => ba.BooksAuthors)
    .ThenInclude(a => a.Author)
    .ToList();
```

Langkah berikutnya adalah membuat object collection yang berisi dari object BookViewModel.

```
IList<BookViewModel> items = new List<BookViewModel>();
```

Sekarang object bookList berisi daftar buku. Selanjutnya adalah mengisi object collection di atas dengan object BookViewModel sehingga langkah pertama adalah melakukan pengulangan sejumlah object yang berada di dalam object collection bookList. Kemudian melakukan instansiasi class BookViewModel.

```
foreach(Book book in bookList){
    BookViewModel item = new BookViewModel();
    ...
}
```

```
}
```

Selanjutnya mengisi object item dengan nilai-nilai dari object book dengan cara sebagai berikut.

```
item.ISBN = book.BookID;
item.Title = book.Title;
item.Photo = book.Photo;
item.PublishDate = book.PublishDate;
item.Price = book.Price;
item.Quantity = book.Quantity;
```

Dari contoh di atas dapat dilihat object item sebagai instance class BooksViewModel telah menampung data dari tabel Books. Selanjutnya adalah membaca tabel Categories untuk mengambil nama kategori buku berdasarkan id dari kategori buku tersebut dengan cara berikut ini.

```
item.CategoryName = book.Category.Name;
```

Langkah selanjutnya adalah menyiapkan variable authorNameList untuk menampung nama-nama pengarang. Kemudian membaca tabel books_authors untuk mendapatkan daftar pengarang buku berdasarkan BookID dengan cara berikut ini.

```
string authorNameList = string.Empty;
var booksAuthorsList = book.BooksAuthors;
```

Karena daftar pengarang yang didapat pada tabel BooksAuthors hanya berisi id pengarang saja maka perlu mengambil nama pengarang dengan cara berikut ini, kemudian menyimpan nama-nama tersebut ke dalam variable authorNameList.

```
foreach(BookAuthor booksAuthors in booksAuthorsList){
    var author = booksAuthors.Author;
    authorNameList = authorNameList + author.Name + ", ";
}
item.AuthorNames = authorNameList.Substring(0, authorNameList.Length - 2);

items.Add(item);
```

Dan pada baris terakhir object item disimpan ke dalam object collection items. Maka sekarang sudah didapat data buku yang lengkap, sehingga pada antarmuka dapat dilihat hasil seperti pada gambar di atas.

Dari penjelasan dan contoh di atas dapat dilihat bahwa class view model tidak perlu didaftarkan pada class data context, karena class ini tidak mewakili tabel pada database.

Penjelasan tentang cara melakukan query dan mengambil data dilakukan dengan .NET Language-Integrated Query (LINQ). Pembahasan tentang LINQ akan dibahas lebih lanjut pada sub bab Controller.

Display & Format

Pada sub bab ini akan diperlihatkan cara pemberian atribut pada model untuk keperluan display pada komponen view. Hal ini bermanfaat untuk menampilkan label pada form atau header pada tabel saat menampilkan data yang terkait dengan model tersebut. Jika model tersebut digunakan pada beberapa halaman pada komponen view, maka dengan mengubah atribut pada model tersebut akan mengubah seluruh tampilan label halaman-halaman tersebut.

Untuk melakukan hal ini digunakan class-class dari namespace berikut ini.

```
using System.ComponentModel.DataAnnotations;
using System.ComponentModel.DataAnnotations.Schema;
```

Dengan namespace ini maka cara memberikan atribut menggunakan cara seperti berikut ini.

```
[Display(Name = "LABEL")]
```

Namespace lain yang bisa digunakan adalah sebagai berikut ini.

```
using System.ComponentModel;
```

Dengan namespace ini maka cara pemberian atribut menggunakan cara seperti berikut ini.

```
[DisplayName("LABEL")]
```

Berikut ini adalah contoh penggunaan atribut ini pada class model Author dengan menggunakan data annotation.

Author.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using System.ComponentModel.DataAnnotations;

namespace SqlServerBookStore.Models
{
    public partial class Author
    {
        [Display(Name = "ID")]
        public int AuthorID { set; get; }

        [Display(Name = "Author's Name")]
        public String Name { set; get; }

        [Display(Name = "Email")]
        public String Email { set; get; }

        public ICollection<BookAuthor> BooksAuthors { get; set; }
    }
}
```

ID	Author's Name	Email	Action
1	Mohammad	m@rezafaisal.net	Edit Delete
2	Reza	reza@faisal.net	Edit Delete
3	Faisal	faisal@rezafaisal.net	Edit Delete

Gambar 115. Display atribut model sebagai label pada header tabel.

Dengan menggunakan HTML Helper @Html.DisplayNameFor di halaman web pada komponen view maka atribut pada komponen model ini bisa dipergunakan untuk ditampilkan sebagai label pada header tabel seperti yang terlihat pada gambar di atas. Contoh yang lain adalah penggunaan untuk form input.

Gambar 116. Display atribut model sebagai label pada form input.

Penjelasan tentang HTML Helper akan diberikan pada pada sub bab View.

Selain itu juga atribut DisplayFormat yang digunakan untuk melakukan format nilai property saat ditampilkan pada halaman komponen view. Sintaks dari atribut ini adalah sebagai berikut.

```
[DisplayFormat(DataFormatString = "{FORMAT}")]
```

Berikut ini adalah contoh dari penggunaan atribut ini.

```
[DisplayFormat(DataFormatString = "{0:dd-MMM-yyyy}")]
```

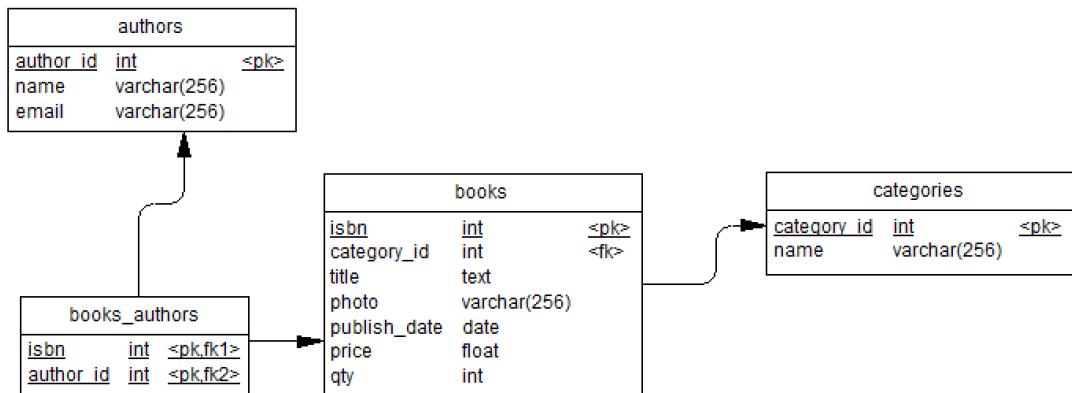
```

public DateTime PublishDate {set; get;}
[DisplayFormat(DataFormatString = "{0:c}")]
public double Price {set; get;}

```

Relasi

Pada database relasional, umumnya terdapat relasi pada antar tabel. Sebagai contoh dapat dilihat pada gambar di bawah ini.



Gambar 117. Relasi pada tabel.

Pada gambar di atas dapat dilihat terdapat 4 tabel. Kemudian pada tabel categories dan books dapat dilihat relasi yang dihubungkan oleh atribut category_id. Dengan adanya relasi tersebut dapat dijelaskan bahwa sebuah record pada tabel categories dapat memiliki lebih dari satu record pada tabel buku, artinya 1 kategori dapat digunakan oleh banyak buku. Relasi ini dapat dikategorikan sebagai relasi one-to-many.

Sedangkan untuk kasus relasi many-to-many dapat dilihat relasi antara tabel books dan authors. Untuk itu diperlukan satu tabel lain yaitu books_authors sehingga dimungkinkan seorang pengarang mengarang banyak buku. Dan sebuah buku ditulis oleh banyak pengarang.

Hal di atas hanya dapat dilakukan jika digunakan pendekatan Database First. Untuk pendekatan Code First maka yang harus dilakukan adalah membuat relasi pada model. Pada sub bab ini akan diberikan cara untuk membuat relasi antar class baik relasi one-to-many atau many-to-many.

Relasi One-to-Many

Berikut ini diberikan contoh dua class yang memiliki relasi one-to-many yaitu class Category dan Book. Untuk menyatakan bahwa class Category dapat memiliki relasi dengan banyak class Book, dapat dilihat pada baris yang dicetak tebal di bawah ini.

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using System.ComponentModel.DataAnnotations;

namespace SqlServerBookStore.Models
{
    public partial class Category
    {
        public int CategoryId { get; set; }
        public string Name { get; set; }
        public virtual ICollection Books { get; set; }
    }
}

```

```

{
    public int CategoryID { set; get; }
    public String Name { set; get; }
    public ICollection<Book> Books { get; set; }
}
}

```

Pada kode yang dicetak tebal di atas tersebut dibuat property Books dengan tipe ICollection<Book>. Artinya property Books dapat menampung banyak object Book. Dengan mengakses property ini maka akan didapat seluruh object Book dari sebuah object Category.

Di bawah ini adalah contoh class Book. Untuk menyatakan class ini memiliki relasi dengan class Category dapat dilihat pada kode yang dicetak tebal.

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using System.ComponentModel.DataAnnotations;
using System.ComponentModel.DataAnnotations.Schema;

namespace SqlServerBookStore.Models
{
    public partial class Book
    {
        [DatabaseGenerated(DatabaseGeneratedOption.None)]
        public int BookID { set; get; }

        [ForeignKey("Category")]
        public int CategoryID { set; get; }

        public Category Category { get; set; }

        public String Title { set; get; }
        . .
    }
}

```

Pada kode di atas dapat dilihat penggunaan atribut ForeignKey untuk menentukan agar property CategoryID menjadi foreign key.

Kemudian dapat dilihat juga property Category dengan tipe dari class Category. Property ini menyatakan bahwa sebuah object Book memiliki relasi kepada sebuah object Category. Property ini juga dapat digunakan untuk mengakses object Category. Sehingga jika terdapat object book1 yang dibuat dari class Book, maka untuk mengetahui nama kategori buku dapat dilakukan dengan cara berikut ini.

```
book1.Category.Name
```

Selanjutnya contoh dan penggunaan relasi one-to-many akan dapat dilihat implementasinya pada sub bab selanjutnya yaitu Book Store: Class Model & Atribut.

Relasi Many-to-Many

Untuk kasus relasi many-to-many dapat dilihat relasi seperti antara tabel books dan authors seperti yang terlihat pada Gambar 86 di atas. Relasi ini membuat sebuah buku dapat ditulis oleh banyak pengarang dan seorang pengarang dapat menulis banyak buku.

Implementasi relasi many-to-many dengan menggunakan class model akan dicontohkan sebagai berikut. Sebagai contoh jika dimiliki class model Author dengan kode berikut ini.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using System.ComponentModel.DataAnnotations;

namespace SqlServerBookStore.Models
{
    public partial class Author
    {
        public int AuthorID { set; get; }
        public String Name { set; get; }
        public String Email { set; get; }

        public ICollection<BookAuthor> BooksAuthors { get; set; }
    }
}
```

Dan class model Book dengan kode sebagai berikut.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using System.ComponentModel.DataAnnotations;
using System.ComponentModel.DataAnnotations.Schema;

namespace SqlServerBookStore.Models
{
    public partial class Book
    {
        [DatabaseGenerated(DatabaseGeneratedOption.None)]
        public int BookID { set; get; }

        [ForeignKey("Category")]
        public int CategoryID { set; get; }
        public Category Category { get; set; }
        public String Title { set; get; }
        public String Photo { set; get; }
        public DateTime PublishDate { set; get; }
        public double Price { set; get; }
        public int Quantity { set; get; }

        public ICollection<BookAuthor> BooksAuthors { get; set; }
    }
}
```

Selanjutnya diperlukan class model tambahan untuk menyimpan relasi many-to-many ini yaitu class BookAuthor berikut ini.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using System.ComponentModel.DataAnnotations;
using System.ComponentModel.DataAnnotations.Schema;

namespace SqlServerBookStore.Models
{
    public partial class BookAuthor
    {
```

```

public int BookAuthorID { set; get; }

[ForeignKey("Book")]
public int BookID { set; get; }

public Book Book { get; set; }

[ForeignKey("Author")]
public int AuthorID { set; get; }

public Author Author { get; set; }

}

```

Untuk membuat relasi class BookAuthor dengan class Book dibuat dapat dilihat pada tiga baris pertama yang dicetak tebal. Sedangkan untuk membuat relasi dengan class Author dapat dilihat pada baris selanjutnya yang dicetak tebal. Dengan kedua relasi ini maka relasi many-to-many antara class Book dan Author telah dibuat.

Validasi

Model juga dapat memiliki atribut yang dapat digunakan untuk validasi pada halaman form. Ada beberapa atribut yang dapat digunakan untuk proses validasi. Berikut adalah atribut-atribut validasi yang dapat digunakan pada model, yaitu:

- Required.
- StringLength.
- DataType.
- MaxLength.
- MinLength.
- Range.
- RegularExpression.
- Compare.

Sintaks penulisan atribut-atribut validasi di atas adalah sebagai berikut.

```
[AttributeName(param, ErrorMessage = "message")]
```

Keterangan:

- AttributeName adalah nama atribut dari daftar di atas.
- param adalah nilai yang diperlukan oleh atribut untuk melakukan validasi. Nilai param ini diperlukan untuk atribut seperti StringLength, untuk menentukan nilai panjang dari string.
- message adalah pesan kesalahan yang akan ditampilkan jika nilai yang dimasukkan tidak memenuhi kondisi validasi yang telah diberikan.

Seperti halnya atribut display yang telah dijelaskan pada sub bab sebelumnya, atribut validasi ini juga ditulis di atas property dari class model. setiap property dapat memiliki atribut validasi lebih dari satu.

```
[Display(Name = "Email")]
[Required(ErrorMessage = "Email harus diisi.")]
[StringLength(256, MinimumLength = 8, ErrorMessage = "Email harus memiliki maksimal 256 dan minimal 8 karakter.")]
```

```
public String Email {set; get;}
```

Sedangkan untuk mempermudah dan mempersingkat penulisan pesan kesalahan pada dapat digunakan placeholder yaitu mempergunakan tanda { }. Berikut adalah nilai yang dapat dipergunakan:

- 0, adalah nama property atau nama display.
- 1, adalah nilai parameter pertama.
- 2, adalah nilai parameter kedua.

Sehingga contoh di atas dapat ditulis dengan cara sebagai berikut ini.

```
[Display(Name = "Email")]
[Required(ErrorMessage = "{0} harus diisi.")]
[DataType(DataType.EmailAddress, ErrorMessage = "{0} tidak valid." )]
[StringLength(256, MinimumLength = 8, ErrorMessage = "{0} harus memiliki maksimal {1} dan minimal {2} karakter.")]
public String Email {set; get;}
```

Required

Atribut Required digunakan untuk memvalidasi property agar harus property harus diisi dengan suatu nilai. Berikut adalah contoh penggunaan atribut ini.

```
[Display(Name = "Author's Name")]
[Required(ErrorMessage = "{0} harus diisi.")]
public String Name {set; get;}
```

Jika nilai property Name di atas tidak diisi saat proses input data pada form, maka akan ditampilkan pesan kesalahan “Author's Name harus diisi.”.

StringLength

Atribut StringLength digunakan untuk memvalidasi property yang bernilai string. Atribut ini akan memvalidasi agar jumlah karakter dari string yang diisikan tidak melebihi dari nilai param. Berikut adalah contoh penggunaan atribut ini.

```
[StringLength(256, ErrorMessage = "{0} tidak boleh lebih {1} karakter.")]
public String Name {set; get;}
```

Pada contoh di atas dapat dilihat nilai param yang diberikan adalah 256, artinya jumlah karakter yang dapat diberikan pada property Name tidak boleh lebih dari 256 karakter. Jika nilai yang diberikan melewati jumlah itu maka akan ditampilkan pesan kesalahan “Author's Name tidak boleh lebih 256 karakter.”.

Selain itu juga dapat diberikan parameter lain untuk menentukan jumlah karakter minimum dengan cara sebagai berikut.

```
[StringLength(256, MinimumLength = 8, ErrorMessage = "{0} harus memiliki maksimal {1} dan minimal {2} karakter.")]
public String Name {set; get;}
```

Jika property class model menggunakan atribut di atas, maka jumlah karakter yang dapat diberikan adalah maksimal 256 dan minimal tidak boleh kurang dari 8 karakter. Dan pesan kesalahan yang akan ditampilkan adalah “Author's Name harus memiliki maksimal 256 dan minimal 8 karakter”.

DataType

Atribut DataType digunakan untuk memvalidasi property agar tipe nilai yang dimasukkan sesuai dengan tipe data yang diberikan. Berikut adalah contoh penggunaan atribut ini.

```
[DataType(DataType.EmailAddress, ErrorMessage="{0} tidak valid." )]  
public String Email {set; get;}
```

Untuk menentukan tipe data digunakan class static DataType. Ada beberapa nilai yang dimiliki oleh class DataType yaitu:

- CreditCard.
- Currency.
- Data.
- DateTime.
- EmailAddress.
- Password.
- PhoneNumber.
- Dan lain-lain.

MaxLength

Atribut ini untuk membatasi jumlah karakter maksimal yang diberikan sebagai nilai property. Berikut adalah contoh penggunaan atribut ini.

```
[MaxLength(256, ErrorMessage ="{0} tidak boleh lebih dari {1} karakter.")]  
public String Email {set; get;}
```

Pesan kesalahan yang akan ditampilkan adalah “Email tidak boleh lebih dari 256 karakter”.

MinLength

Atribut ini untuk membatasi karakter minimal yang diberikan sebagai nilai property. Berikut ini adalah contoh penggunaan atribut ini.

```
[MinLength(8, ErrorMessage = "{0} tidak boleh kurang dari {1} karakter.")]  
public String Email {set; get;}
```

Pesan kesalahan yang akan ditampilkan adalah “Email tidak boleh kurang dari 8 karakter”.

Range

Atribut Range digunakan untuk membatasi nilai minimal dan maksimal nilai angka pada suatu property. Berikut adalah contoh penggunaan atribut ini.

```
[Display(Name = "Harga")]  
[Range(100, 10000, ErrorMessage = "{0} harus diantara Rp. {1} dan Rp. {2}")]  
public decimal Price { get; set; }
```

Pesan kesalahan yang ditampilkan adalah sebagai berikut “Harga harus diantara Rp. 100 dan Rp. 10000”.

Compare

Atribut Compare digunakan untuk membandingkan antara nilai-nilai dari dua property. Sebagai contoh adalah membandingkan nilai property Password dan ConfirmPassword.

```
public string Password { get; set; }  
  
[Compare("Password", ErrorMessage = "{0} dan {1} harus sama.")]  
public string ConfirmPassword { get; set; }
```

Pesan kesalahan yang ditampilkan adalah sebagai berikut “Password dan ConfirmPassword harus sama.”.

RegularExpression

Atribut ini digunakan untuk melakukan validasi nilai property dengan menggunakan pola regular expression. Sebagai contoh untuk melakukan validasi email dapat digunakan contoh sebagai berikut ini.

```
[RegularExpression(@"^([a-zA-Z0-9_\.-])+@[([a-zA-Z0-9\-\-])+\.)([a-zA-Z0-9]{2,4})+$", ErrorMessage = "{0} tidak valid.")]
public string Email { get; set; }
```

Book Store: Class Model & Atribut

Dari penjelasan di atas maka berikut ini adalah kode lengkap class mode yang telah diberikan atribut Display, DisplayFormat dan atribut-atribut untuk validasi.

Category.cs

Class entity model Category ini akan digunakan pada komponen view untuk keperluan menampilkan data, form menambah dan form mengedit data kategori buku. Class ini juga digunakan pada komponen controller untuk proses menambah, mengambil, update dan penghapusan data pada database.

Berikut adalah class entity model Category yang telah diberikan atribut-atribut untuk display dan validasi.

```
Category.cs
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using System.ComponentModel.DataAnnotations;

namespace SqlServerBookStore.Models
{
    public partial class Category
    {
        [Display(Name = "ID")]
        [Required(ErrorMessage = "{0} harus diisi.")]
        public int CategoryID { set; get; }

        [Display(Name = "Book Category Name")]
        [Required(ErrorMessage = "{0} harus diisi.")]
        [StringLength(256, ErrorMessage = "{0} tidak boleh lebih {1} karakter.")]
        public String Name { set; get; }

        public ICollection<Book> Books { get; set; }
    }
}
```

Author.cs

Class entity model Author ini akan digunakan pada komponen view untuk keperluan menampilkan data, menambah dan mengedit data pengarang buku. Class ini juga digunakan pada komponen controller untuk proses menambah, mengambil, update dan penghapusan data pada database.

Kode di bawah ini adalah kode lengkap class entity model Author yang telah diberikan atribut-atribut untuk display dan validasi.

```
Author.cs
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using System.ComponentModel.DataAnnotations;

namespace SqlServerBookStore.Models
{
    public partial class Author
    {
        [Display(Name = "ID")]
        [Required(ErrorMessage = "{0} harus diisi.")]
        public int AuthorID { set; get; }

        [Display(Name = "Author's Name")]
        [Required(ErrorMessage = "{0} harus diisi.")]
        [StringLength(256, ErrorMessage = "{0} tidak boleh lebih {1} karakter.")]
        public String Name { set; get; }

        [Display(Name = "Email")]
        [Required(ErrorMessage = "{0} harus diisi.")]
        [StringLength(256, ErrorMessage = "{0} tidak boleh lebih {1} karakter.")]
        [RegularExpression(@"^([a-zA-Z0-9_\.-])+@(([a-zA-Z0-9\-\-])+\.)+([a-zA-Z0-9]{2,4})+$", ErrorMessage = "{0} tidak valid.")]
        public String Email { set; get; }

        public ICollection<BookAuthor> BooksAuthors { get; set; }
    }
}
```

Book.cs

Class entity model Book ini akan hanya akan digunakan pada komponen controller untuk melakukan proses menambah, mengambil, update dan penghapusan data pada database. Berikut ini adalah kode lengkap class entity model Book yang telah diberikan atribut-atribut untuk display dan validasi.

```
Book.cs
using System;
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations;
using System.ComponentModel.DataAnnotations.Schema;

namespace SqlServerBookStore.Models
{
    public partial class Book{
        [DatabaseGenerated(DatabaseGeneratedOption.None)]
        [Display(Name ="ISBN")]
        [Required(ErrorMessage = "{0} harus diisi.")]
        [RegularExpression("^[0-9]*$", ErrorMessage = "{0} harus angka")]
        [StringLength(13, MinimumLength = 10, ErrorMessage = "{0} tidak boleh lebih {1} dan tidak boleh kurang {2} karakter.")]
        public int BookID {set; get;}

        [ForeignKey("Category")]
    }
}
```

```

[Display(Name ="Category ID")]
[Required(ErrorMessage = "{0} harus diisi.")]
[RegularExpression("^[0-9]*$", ErrorMessage = "{0} harus angka")]
public int CategoryID {set; get;}

public Category Category { get; set; }

[Display(Name ="Title")]
[Required(ErrorMessage = "{0} harus diisi.")]
public String Title {set; get;}

[Display(Name ="Photo")]
public String Photo {set; get;}

[Display(Name ="Publish Date")]
public DateTime PublishDate {set; get;}

[Display(Name ="Price")]
[RegularExpression("^[0-9]*$", ErrorMessage = "{0} harus angka")]
public double Price {set; get;}

[Display(Name ="Quantity")]
[RegularExpression("^[0-9]*$", ErrorMessage = "{0} harus angka")]
public int Quantity {set; get;}

public ICollection<BookAuthor> BooksAuthors { get; set; }
}
}

```

BookAuthor.cs

Class entity model BookAuthor ini akan hanya akan digunakan pada komponen controller untuk melakukan proses menambah, mengambil, update dan penghapusan data pada database. Class ini adalah class penghubung antara class Book dan Author.

Berikut adalah kode lengkap class entity model BookAuthor yang telah diberikan atribut-atribut.

BookAuthor.cs
<pre> using System; using System.Collections.Generic; using System.Linq; using System.Threading.Tasks; using System.ComponentModel.DataAnnotations; using System.ComponentModel.DataAnnotations.Schema; namespace SqlServerBookStore.Models { public partial class BookAuthor { public int BookAuthorID { set; get; } [ForeignKey("Book")] [Display(Name = "ISBN")] [Required(ErrorMessage = "{0} harus diisi.")] [RegularExpression("^[0-9]*\$", ErrorMessage = "{0} harus angka")] [StringLength(13, MinimumLength = 10, ErrorMessage = "{0} tidak boleh lebih {1} dan tidak boleh kurang {2} karakter.")] public int BookID { set; get; } public Book Book { get; set; } } } </pre>

```

    [ForeignKey("Author")]
    [Display(Name = "AuthorID")]
    [Required(ErrorMessage = "{0} harus diisi.")]
    public int AuthorID { set; get; }
    public Author Author { get; set; }
}
}

```

BookViewModel.cs

Class BookViewModel.cs adalah contoh dari implementasi class view model. Pada class ini tidak diberikan atribut untuk validasi, karena class ini hanya digunakan untuk keperluan menampilkan data saja. Class ini hanya akan digunakan pada komponen view Index.cshtml untuk menampilkan data buku pada tabel.

Dan berikut ini adalah kode lengkap class view model BookViewModel.

```

BookViewModel.cs
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using System.ComponentModel.DataAnnotations;

namespace SqlServerBookStore.Models
{
    public partial class BookViewModel
    {
        [Display(Name = "ISBN")]
        public int ISBN { set; get; }

        [Display(Name = "Category")]
        public String CategoryName { set; get; }

        [Display(Name = "Title")]
        public String Title { set; get; }

        [Display(Name = "Photo")]
        public String Photo { set; get; }

        [Display(Name = "Publish Date")]
        [DisplayFormat(DataFormatString = "{0:dd-MMMM-yyyy}")]
        public DateTime PublishDate { set; get; }

        [Display(Name = "Price")]
        [DisplayFormat(DataFormatString = "{0:c}")]
        public double Price { set; get; }

        [Display(Name = "Quantity")]
        public int Quantity { set; get; }

        [Display(Name = "List Author Names")]
        public string AuthorNames { set; get; }
    }
}

```

BookFormViewModel.cs

Class BookFormViewModel.cs adalah contoh implementasi class view model. Class BookFormViewModel.cs ini hanya akan digunakan pada komponen view Create.cshtml dan Edit.cshtml. File komponen view Create.cshtml digunakan sebagai form untuk menambah data buku. Sedangkan file komponen view Edit.cshtml digunakan sebagai form untuk mengedit data buku.

Berikut adalah kode lengkap file class BookFormViewModel.cs.

```
BookFormViewModel.cs
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using System.ComponentModel.DataAnnotations;
using Microsoft.AspNetCore.Http;

namespace SqlServerBookStore.Models
{
    public partial class BookFormViewModel
    {
        [Display(Name = "ISBN")]
        public int ISBN { set; get; }

        [Display(Name = "Category")]
        public int CategoryID { set; get; }

        [Display(Name = "Title")]
        public String Title { set; get; }

        [Display(Name = "Photo")]
        [DataType(DataType.Upload)]
        public IFormFile Photo { set; get; }

        [Display(Name = "Publish Date")]
        [DataType(DataType.Date)]
        public DateTime PublishDate { set; get; }

        [Display(Name = "Price")]
        [DataType(DataType.Currency)]
        public double Price { set; get; }

        [Display(Name = "Quantity")]
        public int Quantity { set; get; }

        [Display(Name = "List of Author Names")]
        public int[] AuthorIDs { set; get; }
    }
}
```

Pada kode di atas dapat dilihat penggunaan interface IFormFile pada property Photo. Interface ini merupakan bagian dari namespace Microsoft.AspNetCore.Http, sehingga jika ingin menggunakan interface ini maka pada awal kode perlu ditambahkan namespace tersebut, seperti yang dapat dilihat pada contoh di atas. Interface IFormFile digunakan untuk keperluan upload file, pada kasus ini ada keperluan untuk mengupload file gambar cover buku.

RoleViewModel.cs

Class RoleViewModel adalah class view model yang digunakan untuk pada komponen view Index.cshtml, Create.cshtml dan Edit.cshtml. Sehingga class tersebut dapat digunakan untuk menampilkan daftar role, menambah dan mengedit data role.

Berikut ini adalah kode lengkap class entity model RoleViewModel yang telah diberikan atribut-atribut untuk display dan validasi.

```
RoleViewModel.cs
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using System.ComponentModel.DataAnnotations;

namespace SqlServerBookStore.Models
{
    public partial class RoleViewModel
    {
        [Display(Name = "Role ID")]
        [Required(ErrorMessage = "{0} harus diisi.")]
        public string RoleID { get; set; }

        [Display(Name = "Role Name")]
        [Required(ErrorMessage = "{0} harus diisi.")]
        public string RoleName { set; get; }

        [Display(Name = "Description")]
        [Required(ErrorMessage = "{0} harus diisi.")]
        public string Description { set; get; }
    }
}
```

UserViewModel.cs

Class UserViewModel adalah class view model yang digunakan pada komponen view Index.cshtml. Class ini berfungsi untuk menampilkan daftar user.

Berikut ini adalah kode lengkap class entity model UserViewModel yang telah diberikan atribut-atribut untuk display. Class ini tidak memerlukan atribut-atribut untuk validasi karena hanya digunakan untuk menampilkan data.

```
UserViewModel.cs
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using System.ComponentModel.DataAnnotations;

namespace SqlServerBookStore.Models
{
    public partial class UserViewModel
    {
        [Display(Name = "Username")]
        public String UserName { set; get; }

        [Display(Name = "Role")]
        public String RoleName { set; get; }

        [Display(Name = "Email")]
    }
}
```

```

        public String Email { set; get; }

        [Display(Name = "Fullname")]
        public String FullName { set; get; }
    }
}

```

UserCreateFormViewModel.cs

Class UserCreateViewModel adalah class view model yang digunakan pada komponen view Create.cshtml. Class ini berfungsi untuk menambah data user.

Berikut ini adalah kode lengkap class entity model UserCreateViewModel yang telah diberikan atribut-atribut untuk display dan validasi.

```

UserCreateFormViewModel.cs
using System;
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations;

namespace SqlServerBookStore.Models
{
    public partial class UserCreateFormViewModel{
        [Display(Name ="Username")]
        [Required(ErrorMessage = "{0} harus diisi.")]
        public String UserName {set; get; }

        [Display(Name ="Role")]
        [Required(ErrorMessage = "{0} harus diisi.")]
        public string[] RoleID {set; get; }

        [Display(Name ="Email")]
        [Required(ErrorMessage = "{0} harus diisi.")]
        [StringLength(256, ErrorMessage = "{0} tidak boleh lebih {1} karakter.")]
        [RegularExpression(@"^([a-zA-Z0-9_\.\-])+@(([a-zA-Z0-9\-])+\.)([a-zA-Z0-9]{2,4})+$", ErrorMessage = "{0} tidak valid.")]
        public String Email {set; get; }

        [Display(Name ="Password")]
        [Required(ErrorMessage = "{0} harus diisi.")]
        [StringLength(256, ErrorMessage = "{0} tidak boleh lebih {1} karakter.")]
        [DataType(DataType.Password)]
        public String Password {set; get; }

        [Display(Name ="Password Confirm")]
        [Required(ErrorMessage = "{0} harus diisi.")]
        [StringLength(256, ErrorMessage = "{0} tidak boleh lebih {1} karakter.")]
        [Compare("Password", ErrorMessage = "{0} dan {1} harus sama.")]
        [DataType(DataType.Password)]
        public String PasswordConfirm {set; get; }

        [Display(Name ="Full Name")]
        [Required(ErrorMessage = "{0} harus diisi.")]
        [StringLength(256, ErrorMessage = "{0} tidak boleh lebih {1} karakter.")]
        public String FullName {set; get;}
    }
}

```

UserEditFormViewModel.cs

Class UserEditViewModel adalah class view model yang digunakan pada komponen view Edit.cshtml. Class ini berfungsi untuk mengedit informasi user yang dipilih.

Berikut ini adalah kode lengkap class entity model UserEditViewModel yang telah diberikan atribut-atribut untuk display dan validasi.

```
UserEditFormViewModel.cs
using System;
using System.ComponentModel.DataAnnotations;

namespace SqlServerBookStore.Models
{
    public partial class UserEditFormViewModel{
        [Display(Name ="Username")]
        [Required(ErrorMessage = "{0} harus diisi.")]
        public String UserName {set; get; }

        [Display(Name ="Role")]
        [Required(ErrorMessage = "{0} harus diisi.")]
        public string[] RoleID {set; get; }

        [Display(Name ="Email")]
        [Required(ErrorMessage = "{0} harus diisi.")]
        [StringLength(256, ErrorMessage = "{0} tidak boleh lebih {1} karakter.")]
        [RegularExpression(@"^([a-zA-Z0-9_\.-])+@[a-zA-Z0-9\.-]+\.[a-zA-Z0-9]{2,4}$", ErrorMessage = "{0} tidak valid.")]
        public String Email {set; get; }

        [Display(Name ="Full Name")]
        [Required(ErrorMessage = "{0} harus diisi.")]
        [StringLength(256, ErrorMessage = "{0} tidak boleh lebih {1} karakter.")]
        public String FullName {set; get;}
    }
}
```

Berbeda dengan class UserCreateFormViewModel, class ini tidak memiliki atribut Password dan PasswordConfirm. Sehingga tidak dimungkinkan untuk mengedit atau mengubah password user yang dipilih pada halaman ini. Hal ini dikarenakan ASP.NET Core Identity hanya memberikan prosedur menganti password dengan cara dilakukan oleh user sendiri, kemudian sistem akan mengirimkan kode ke email user tersebut. Selanjutnya email tersebut dapat digunakan untuk mengubah atau reset password.

ApplicationUser.cs

Class ApplicationUser digunakan untuk manambah atribut pada class IdentityUser. Tidak ada atribut display dan validasi yang ditambahkan pada class ini, karena proses menampilkan daftar user, menambah dan mengedit data digunakan class view model yang telah disebutkan di atas.

Berikut ini adalah kode lengkap dari class ApplicationUser.

```
 ApplicationUser.cs
using System;
using System.Collections.Generic;
using System.Linq;
```

```

using System.Threading.Tasks;
using Microsoft.AspNetCore.Identity;

namespace SqlServerBookStore.Models
{
    // Add profile data for application users by adding properties to the
    ApplicationUser class
    public class ApplicationUser : IdentityUser
    {
        public string FullName { get; set; }
        public string Photo { get; set; }

    }
}

```

ApplicationRole.cs

Class ApplicationRole digunakan untuk menambah atribut pada class IdentityRole. Tidak ada atribut display dan validasi yang ditambahkan pada class ini, karena proses menampilkan daftar user, menambah dan mengedit data digunakan class view model yang telah disebutkan di atas.

Berikut ini adalah kode lengkap dari class ApplicationRole.

```

ApplicationRole.cs
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Identity;

namespace SqlServerBookStore.Models
{
    public class ApplicationRole : IdentityRole
    {
        public string Description { get; set; }
    }
}

```

Class Migrations & Database

Class migrations pada project ini telah dibuat saat project dibuat. Karena pada sub bab ini telah ditambahkan dan dimodifikasi class model maka perlu ada pembaharuan atau update class migrations. Atau membuat class migration yang baru sesuai dengan class model yang telah ada sekarang.

Startup.cs

Sebelum membuat class migrations yang baru, pastikan class Startup.cs telah dimodifikasi seperti berikut ini.

```

Startup.cs
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Builder;

```

```

using Microsoft.AspNetCore.Identity;
using Microsoft.EntityFrameworkCore;
using Microsoft.AspNetCore.Hosting;
using Microsoft.Extensions.Configuration;
using Microsoft.Extensions.DependencyInjection;
using SqlServerBookStore.Data;
using SqlServerBookStore.Models;
using SqlServerBookStore.Services;

namespace SqlServerBookStore
{
    public class Startup
    {
        public Startup(IConfiguration configuration)
        {
            Configuration = configuration;
        }

        public IConfiguration Configuration { get; }

        // This method gets called by the runtime. Use this method to add
        services to the container.
        public void ConfigureServices(IServiceCollection services)
        {
            services.AddDbContext<ApplicationContext>(options =>

options.UseSqlServer(Configuration.GetConnectionString("DefaultConnection"))
);

            services.AddIdentity< ApplicationUser, ApplicationRole>(p => {
                p.Password.RequireDigit = false;
                p.Password.RequireLowercase = false;
                p.Password.RequireUppercase = false;
                p.Password.RequireNonAlphanumeric = false;
                p.Password.RequiredLength = 6;
            })
                .AddEntityFrameworkStores< ApplicationContext >()
                .AddDefaultTokenProviders();

            // Add application services.
            services.AddTransient< IEmailSender, EmailSender >();

            services.AddMvc();
        }

        // This method gets called by the runtime. Use this method to
        configure the HTTP request pipeline.
        public void Configure(IApplicationBuilder app, IHostingEnvironment
env)
        {
            if (env.IsDevelopment())
            {
                app.UseBrowserLink();
                app.UseDeveloperExceptionPage();
                app.UseDatabaseErrorPage();
            }
            else
            {
                app.UseExceptionHandler("/Home/Error");
            }
        }
    }
}

```

```

        app.UseStaticFiles();

        app.UseAuthentication();

        app.UseMvc(routes =>
    {
        routes.MapRoute(
            name: "default",
            template: "{controller=Home}/{action=Index}/{id?}");
    });
}
}

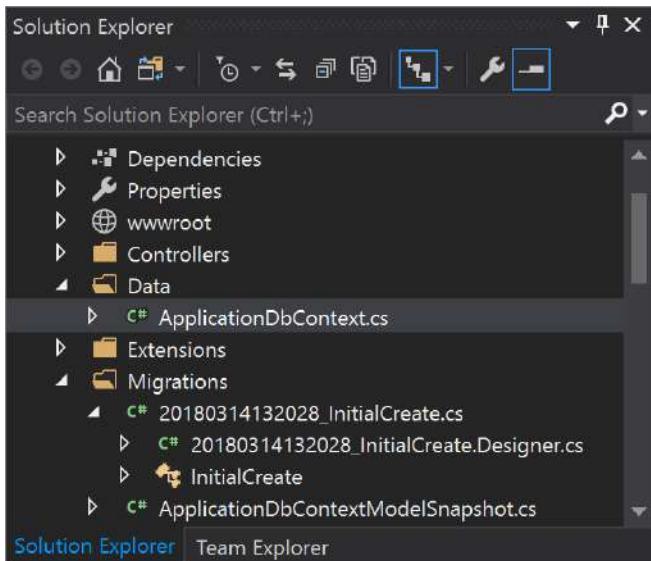
```

Migrations Code

Selanjutnya untuk membuat class migrations yang baru maka terlebih dahulu folder Migrations dan isinya harus dihapus. Selanjutnya buka Package Manager Console dan ketikan perintah berikut ini.

```
Add-Migration InitialCreate
```

Hasilnya dapat dilihat pada folder Migrations seperti pada gambar di bawah ini.



Gambar 118. SqlServerBookStore - Migrations class.

Hasil perintah ini akan dapat dilihat pada folder Migrations berupa file:

- *_InitialCreate.cs
- *_InitialCreate.Designer.cs
- ApplicationDbContextModelSnapshot.cs.

Dimana tanda "*" adalah timestamp saat kedua file tersebut dibuat. Sebagai contoh pada kode yang penulis buat memiliki nama file sebagai berikut:

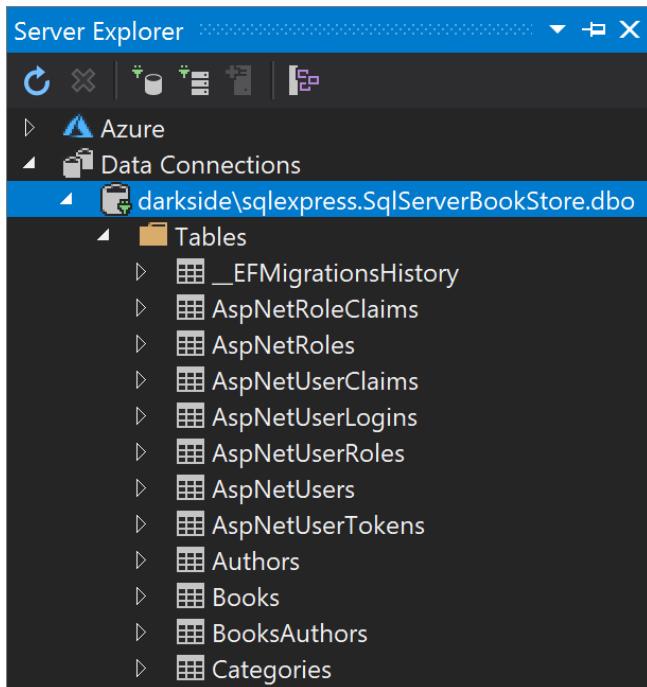
- 20180314132028_InitialCreate.cs.
- 20180314132028_InitialCreate.Designer.cs.

Update Database

Selanjutnya adalah menjalankan perintah ini untuk membuat database dan tabel-tabel sesuai dengan class-class model yang telah dibuat.

```
Update-Database
```

Hasilnya database dan table-table telah dibuat seperti yang terlihat pada gambar di bawah ini.



Gambar 119. SqlServerBookStore - Database SqlServerBookStore dan table-table.

Controller

Controller adalah komponen yang menangani interaksi user, menentukan view yang akan ditampilkan dan menghubungkan dengan model. Sebagai contoh, user dapat berinteraksi dengan controller dengan cara menulis nama controller dan aksi yang diinginkan pada address bar di web browser, kemudian respon dengan memilihkan view yang akan ditampilkan sesuai dengan aksi yang telah dipanggil. Sebelum memilihkan view yg akan ditampilkan, aksi pada controller dapat berisi logika yang berisi baris perintah seperti perintah untuk melakukan pengambilan data ke database kemudian data akan dimasukkan ke dalam collection yang berisi object model, dan kemudian menampilkannya pada view yang diinginkan. Aksi pada controller juga dapat berisi logika untuk menangani proses upload file dan logika-logika lainnya.

Implementasi komponen controller pada ASP.NET Core MVC adalah berupa class controller. Class ini memiliki beberapa aturan, yaitu aturan penamaan dan penulisan program. Untuk penamaan class controller digunakan aturan sebagai berikut.

```
{ControllerName}Controller.cs
```

Jika ControllerName adalah Categories maka nama file yang harus digunakan adalah CategoriesController.cs. Contoh lain, jika nama controller adalah Books maka nama file yang harus digunakan adalah BooksController.cs.

Untuk aturan penulisan program adalah sebagai berikut.

```
using System.Linq;
using Microsoft.AspNetCore.Mvc;

namespace {NameSpace}
{
    public class {ControllerName}Controller : Controller
    {
        .
        .
    }
}
```

Keterangan:

- {NameSpace} adalah name namespace yang biasanya sesuai dengan nama folder tempat menyimpan file class controller ini.
- {ControllerName} adalah nama controller seperti yang dicontohkan pada aturan penamaan di atas.

Selanjutnya di dalam class ini akan berisi method-method seperti umumnya sebuah class. Untuk method yang dapat diakses oleh komponen view maka method tersebut harus mengikuti aturan sebagai method action.

Berikut ini adalah aturan penulisan program method action.

```
[Attribute]
public IActionResult MethodName()
{
    .
    .
    return View();
}
```

Keterangan:

- Sebuah method action menggunakan interface IActionResult yang berfungsi sebagai tipe keluaran dari method tersebut. Pada contoh di atas dapat dilihat bahwa method action mengembalikan keluaran.
- Pada sebuah action dapat diberikan atribut dengan cara penulisan seperti di atas. Salah manfaat atribut adalah dapat memberikan hak akses pada method action.

Atribut yang umum sering digunakan pada method action adalah penggunaan atribut berikut ini:

- `HttpGet`, atribut ini untuk menyatakan bahwa pengiriman data akan dilakukan dengan method GET.
- `HttpPost`, atribut ini untuk menyatakan bahwa pengiriman data akan dilakukan dengan method POST.

Untuk pengembangan aplikasi web Book Store yang terdiri atas 3 fitur maka untuk masing-masing fitur dapat ditangani oleh 3 komponen controller. Oleh karena itu perlu dibuat 3 file class controller yaitu:

- CategoriesController.cs.
- AuthorsController.cs.
- BooksController.cs.

View Bag

ASP.NET Core MVC memiliki sarana yang dapat digunakan untuk melakukan pertukaran data antara controller dan view yaitu ViewBag. Object ini sangat membantu untuk melakukan komunikasi pada sisi server antara controller dan view. View bag dapat menampung nilai primitif seperti bilangan atau string. Selain itu view bag juga dapat menampung object atau kumpulan object yang disimpan pada sebuah collection.

Setelah view bag memiliki nilai selanjutnya nilai tersebut dapat ditampilkan secara langsung pada halaman web sebagai teks atau ditampilkan secara tidak langsung sebagai item pada elemen <select>.

Siklus hidup dari object ini singkat, sehingga valuenya akan bernilai null jika terjadi proses redirect halaman.

Untuk latihan digunakan class controller LatihanController.cs yang telah dibuat pada sub bab sebelumnya. Kemudian pada tambahkan method action Latihan ViewBag, selanjutnya tambahkan file Latihan ViewBag.cshtml pada folder Views/Latihan.

```
[HttpGet]
public IActionResult LatihanViewBag()
{
    return View();
}
```

Contoh pertama penggunaan view bag adalah untuk menampilkan nilai statik yaitu untuk menampilkan nilai variable bertipe integer dan string. Sehingga kode method action di atas menjadi sebagai berikut ini.

```
public IActionResult LatihanViewBag()
{
    ViewBag.VariableInt = 13;
    ViewBag.VariableString = "ASP .NET Core MVC";
    return View();
}
```

Dari contoh di atas maka dapat dilihat sintaks penggunaan object view bag adalah sebagai berikut.

```
ViewBag.NAMA_PROPERTY = NILAI;
```

Untuk menampilkan object view bag pada halaman komponen view dapat dilakukan dengan menggunakan sintaks Razor berikut ini.

```
@ViewBag.NAMA_PROPERTY
```

Sehingga dapat dilihat isi file Latihan ViewBag.cshtml berikut ini.

```
@ ViewBag.VariableInt <br/>
@ ViewBag.VariableString <br/>
```

LINQ

LINQ atau .NET Language-Integrated Query adalah fitur yang dimiliki oleh bahasa pemrograman pada lingkungan .NET. Fitur ini memungkinkan untuk melakukan query pada sintaks di dalam bahasa pemrograman yang digunakan. Query dapat dilakukan pada object bertipe array, enumerable class, dokumen XML dan database.

Pada sub bab ASP.NET Core MVC & MySQL telah diberikan contoh integrasi Entity Framework Core sebagai framework data access pada aplikasi web ASP.NET Core MVC. Pada contoh tersebut juga dapat dilihat implementasi LINQ untuk melakukan query database.

Penggunaan LINQ dapat dilakukan dengan dua cara yaitu dengan cara Extension Method dan penulisan kode query seperti query yang umum digunakan pada SQL. Pada sub bab sebelumnya diberikan contoh penggunaan LINQ dengan menggunakan extension method.

Pada sub bab ini akan diperlakukan contoh dan penjelasan implementasi LINQ untuk mendukung pembangunan aplikasi web book store.

Sumber Data

Seperti yang disebutkan di atas bahwa query dapat dilakukan pada sumber data salah satunya adalah database. Dengan menggunakan Entity Framework sebagai framework data access maka setiap tabel dapat diakses dalam bentuk object DbSet. Pada class BookStoreDataContext.cs dapat dilihat bahwa class ini memiliki 4 property dengan tipe DbSet, yaitu:

- Categories.
- Authors.
- Books.
- BooksAuthors.

Jika dibuat instansiasi object dari class BookStoreDataContext dengan cara berikut ini.

```
BookStoreDataContext db = new BookStoreDataContext();
```

Maka akan dimiliki 4 sumber data yang dapat diakses dengan cara berikut ini.

```
db.Categories  
db.Authors  
db.Books  
db.BooksAuthors
```

Setiap property di atas menyimpan data dari tabel category, author, book dan book_author.

Retrieve & Filter Data

Pada sub bab ini akan diberikan contoh dan penjelasan mengambil dan filter data dengan menggunakan LINQ. Pada contoh sebelumnya yang telah diberikan pada sub bab MySQL Entity Framework Core, dapat dilihat penggunaan LINQ dengan menggunakan extension method. Extension method adalah mempunyai tujuan yang sama seperti method pada umumnya tetapi cara penulisannya yang berbeda.

Method pada umumnya ditulis dengan cara berikut ini.

```
NamaMethod(namaObject)
```

Dari cara di atas dapat dilihat pada method dapat diisi suatu object atau suatu nilai yang akan diproses oleh method. Sedangkan cara penulisan extension method adalah sebagai berikut.

```
namaObject.NamaMethod()
```

Untuk mengambil data digunakan method Select, sehingga untuk mengambil dapat dilakukan dengan cara berikut ini.

```
var items = db.Categories.Select(p => p);
```

Dari contoh di atas, db.Authors dapat dianalogikan sebagai namaObject. Sedangkan method Select(p => p) dapat dianalogikan sebagai method NamaMethod(). Tetapi didalam method Select() dapat dilihat expression yang dikenal dengan nama lambda expression. Sintaks dari lambda expression dapat dilihat di bawah ini.

```
input_parameter => expression_atau_statement
```

Keterangan:

- input_parameter adalah input.
- => adalah operator lambda.
- expression_atau_statement adalah expression atau statement yang menjadi output.

Sehingga p => p dapat diartikan seluruh input p akan menjadi output. Contoh lain adalah x => 2 * x, artinya adalah terdapat input x dan output 2 * x. Sedangkan untuk contoh kasus penggunaan extension method Select() di atas, input p berisi nilai dari db.Authors. Sedangkan yang menjadi output adalah seluruh p. Dan output p akan disimpan ke object items yang berisi kumpulan atau lebih dari satu object Category.

Untuk melakukan filter data digunakan method Where. Berikut adalah contoh melakukan filter dengan method ini.

```
var items = db.BooksAuthors.Where(p => p.ISBN.Equals(book.ISBN));
```

Pada contoh di atas dapat dilihat input p berisi nilai dari db.BooksAuthors, sedangkan yang menjadi output adalah object p yang memiliki nilai property ISBN yang sama dengan nilai book.ISBN. Output p akan disimpan ke dalam object items. Object items akan berisi kumpulan object atau lebih dari satu object BookAuthor.

Untuk mendapatkan sebuah object saja, maka dapat dilakukan filter dengan menggunakan property class yang berperan sebagai primary key. Berikut ini contoh dari solusi kasus tersebut.

```
var item = db.Categories.SingleOrDefault(p=>p.CategoryID.Equals(id));
```

Method SingleOrDefault digunakan untuk mendapatkan sebuah object. Pada kasus di atas sebuah object Category didapat dengan cara melakukan filter berdasarkan property CategoryID.

Ketiga contoh di atas adalah contoh penggunaan extension method. Ketiga contoh di atas dapat ditulis dengan style sintaks SQL dengan sedikit perbedaan. Untuk contoh pertama di atas dapat ditulis menjadi berikut ini.

```
var items = from p in db.Categories select p;
```

Jika pada sintaks SQL dimulai dengan keyword “select” kemudian diikuti “from” untuk menentukan sumber datanya. Tetapi dengan LINQ, hal pertama yang dilakukan adalah memilih sumber data dengan keyword “from” dalam kasus di atas sumber datanya adalah

db.Categories, kemudian baru digunakan keyword “select” untuk memilih property apa saja yang ingin diambil atau ditampilkan.

Sedangkan untuk contoh kedua di atas dapat ditulis sebagai berikut.

```
var items = from p in db.BooksAuthors where p.ISBN.Equals(book.ISBN) select p;
```

Sedangkan untuk contoh ketiga dapat ditulis sebagai berikut ini.

```
var item = (from p in db.Categories where p.CategoryID.Equals(id) select p).FirstOrDefault();
```

Dari penjelasan di atas dapat dilihat bagaimana query menjadi bagian dari bahasa pemrograman pada lingkungan .NET. Masih banyak sintaks query yang dimiliki oleh LINQ seperti layaknya sintaks SQL yang dapat dilihat pada link berikut ini <https://code.msdn.microsoft.com/101-LINQ-Samples-3fb9811b>.

Tambah, Edit & Hapus Data

Untuk operasi tambah, edit dan hapus data merupakan bagian dari framework data access, sehingga tidak ada sintaks LINQ yang dapat digunakan untuk operasi-operasi tersebut. Framework data access yang digunakan untuk pembangunan aplikasi web Book Store adalah Entity Framework Core. Maka berikut ini adalah sintaks dan contoh implementasi operasi tambah, edit dan hapus dengan menggunakan Entity Framework Core.

Untuk operasi tambah data digunakan sintaks berikut ini.

```
db.Add(obj);  
db.SaveChanges();
```

Keterangan:

- db adalah object dari class data context.
- obj adalah object dari class entity model.

Untuk operasi edit data digunakan sintaks berikut ini.

```
db.Update(obj);  
db.SaveChanges();
```

Sedangkan untuk menghapus data digunakan sintaks berikut ini.

```
db.DbSetObject.Remove(obj);  
db.SaveChanges();
```

Keterangan:

- DbSetObject adalah object class DbSet yang didaftarkan sebagai property dari class data context. Dalam kasus ini adalah class BookStoreDataContext.cs.

Sehingga jika ingin menghapus data dari tabel category maka digunakan kode berikut ini.

```
db.Categories.Remove(item);  
db.SaveChanges();
```

Dimana object item adalah object Category yang akan dihapus.

Book Store: Komponen Controller

Pada bab sebelumnya telah diberikan panduan untuk membuat komponen controller sekaligus komponen view. Pada sub bab ini juga akan dibuat kedua komponen tersebut dengan langkah-langkah yang sama. Setelah itu diberikan penjelasan detail mengenai kode

komponen controller yang dibuat. Tetapi akan ada modifikasi terhadap class-class controller yang digenerate itu untuk menyesuaikan dengan antarmuka dan class model yang telah disiapkan. Class tersebut adalah class BooksController.

Selain itu ada juga class controller yang tidak dapat dibuat secara otomatis seperti class controller untuk mengelola role dan user. Hal ini disebabkan data role dan user dikelola dengan menggunakan ASP.NET Core Identity.

Class Controller Categories

Untuk fitur pengelolaan kategori buku dapat digunakan untuk:

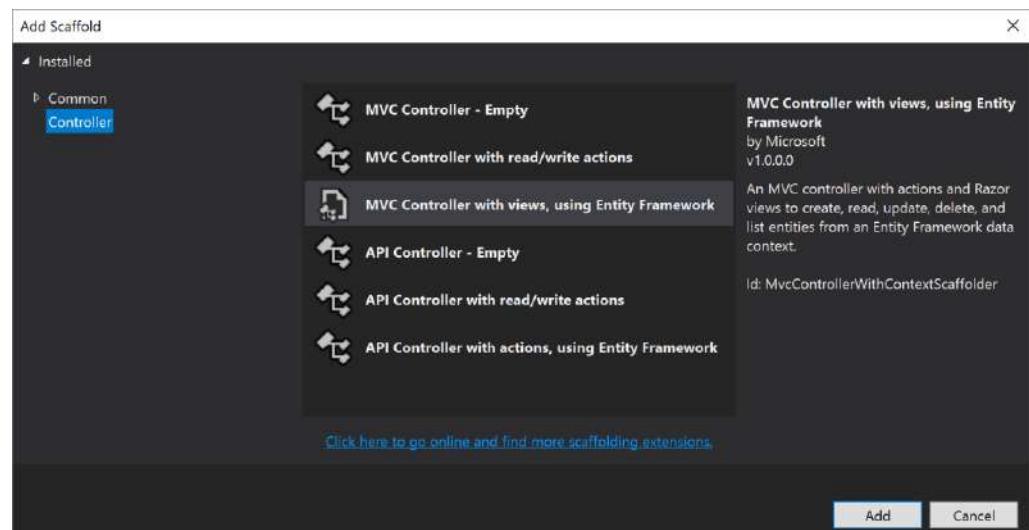
- Menampilkan daftar kategori buku.
- Menampilkan detail kategori buku yang dipilih.
- Menambah data kategori buku.

Untuk menambah data kategori buku terlebih dahulu perlu ditampilkan form input kategori buku. Kemudian pengguna akan mengisi form input tersebut dan setelah tombol submit diklik akan dilakukan proses menyimpanan data yang dimasukkan ke dalam database.

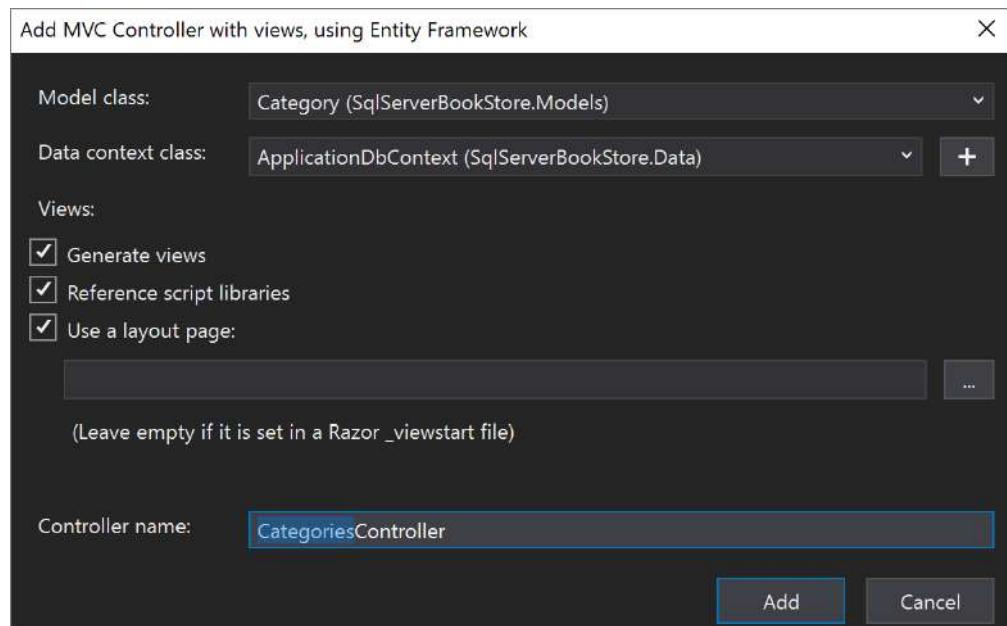
- Mengedit data kategori buku yang dipilih.
- Untuk mengedit data kategori buku terlebih dahulu perlu ditampilkan form input edit kategori buku. Kemudian pengguna akan mengedit data dari form edit. Setelah tombol submit diklik maka dilakukan proses update data ke dalam database.
- Menghapus data kategori buku yang dipilih.

Komponen controller untuk mengelola data Category akan memiliki nama dalam bentuk jamak yaitu Categories. Berikut ini adalah langkah-langkah untuk membuat class controller ini.

Langkah pertama klik kanan pada folder Controllers kemudian pilih Add > Controller. Pada window Add Scaffold pilih MVC Controller with views, using Entity Framework.



Gambar 120. CategoriesController - Window Add Scaffold.



Gambar 121. CategoriesController - Window Add MVC Controller with views, using Entity Framework.

Berikut adalah nilai-nilai yang harus dipilih dan diisi, yaitu:

- Model class, pilih komponen model Category.
- Data context class, pilih ApplicationDbContext.

Setelah tombol Add diklik maka hasilnya dapat dilihat pada file CategoriesController.cs di dalam folder Controllers. Berikut adalah isi dari file tersebut.

```
CategoriesController.cs
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.Mvc.Rendering;
using Microsoft.EntityFrameworkCore;
using SqlServerBookStore.Data;
using SqlServerBookStore.Models;

namespace SqlServerBookStore.Controllers
{
    public class CategoriesController : Controller
    {
        private readonly ApplicationDbContext _context;

        public CategoriesController(ApplicationDbContext context)
        {
            _context = context;
        }

        // GET: Categories
        public async Task<IActionResult> Index()
        {
            return View(await _context.Categories.ToListAsync());
        }

        // GET: Categories/Details/5
        public async Task<IActionResult> Details(int? id)
```

```

{
    if (id == null)
    {
        return NotFound();
    }

    var category = await _context.Categories
        .SingleOrDefaultAsync(m => m.CategoryID == id);
    if (category == null)
    {
        return NotFound();
    }

    return View(category);
}

// GET: Categories/Create
public IActionResult Create()
{
    return View();
}

// POST: Categories/Create
// To protect from overposting attacks, please enable the specific
properties you want to bind to, for
// more details see http://go.microsoft.com/fwlink/?LinkId=317598.
[HttpPost]
[ValidateAntiForgeryToken]
public async Task<IActionResult> Create([Bind("CategoryID,Name")]
Category category)
{
    if (ModelState.IsValid)
    {
        _context.Add(category);
        await _context.SaveChangesAsync();
        return RedirectToAction(nameof(Index));
    }
    return View(category);
}

// GET: Categories/Edit/5
public async Task<IActionResult> Edit(int? id)
{
    if (id == null)
    {
        return NotFound();
    }

    var category = await _context.Categories.SingleOrDefaultAsync(m
=> m.CategoryID == id);
    if (category == null)
    {
        return NotFound();
    }
    return View(category);
}

// POST: Categories/Edit/5
// To protect from overposting attacks, please enable the specific
properties you want to bind to, for
// more details see http://go.microsoft.com/fwlink/?LinkId=317598.

```

```

[HttpPost]
[ValidateAntiForgeryToken]
public async Task<IActionResult> Edit(int id,
[Bind("CategoryID,Name")] Category category)
{
    if (id != category.CategoryID)
    {
        return NotFound();
    }

    if (ModelState.IsValid)
    {
        try
        {
            _context.Update(category);
            await _context.SaveChangesAsync();
        }
        catch (DbUpdateConcurrencyException)
        {
            if (!CategoryExists(category.CategoryID))
            {
                return NotFound();
            }
            else
            {
                throw;
            }
        }
        return RedirectToAction(nameof(Index));
    }
    return View(category);
}

// GET: Categories/Delete/5
public async Task<IActionResult> Delete(int? id)
{
    if (id == null)
    {
        return NotFound();
    }

    var category = await _context.Categories
        .SingleOrDefaultAsync(m => m.CategoryID == id);
    if (category == null)
    {
        return NotFound();
    }

    return View(category);
}

// POST: Categories/Delete/5
[HttpPost, ActionName("Delete")]
[ValidateAntiForgeryToken]
public async Task<IActionResult> DeleteConfirmed(int id)
{
    var category = await _context.Categories.SingleOrDefaultAsync(m => m.CategoryID == id);
    _context.Categories.Remove(category);
    await _context.SaveChangesAsync();
    return RedirectToAction(nameof(Index));
}

```

```

        }

        private bool CategoryExists(int id)
        {
            return _context.Categories.Any(e => e.CategoryID == id);
        }
    }
}

```

Pada kode di atas, akan digunakan dua komponen model yaitu:

- Class data context ApplicationDbContext yang berada pada kelompok namespace SqlServerBookStore.Data. Class data context ini adalah class utama untuk melakukan koneksi dan operasi database.
- Class model entity Category yang berada pada kelompok namespace SqlServerBookStore.Models.

Sehingga pada kode di atas dapat dilihat cara untuk memuat kedua namespace tersebut dengan kode berikut ini.

```

using SqlServerBookStore.Data;
using SqlServerBookStore.Models;

```

Selanjutnya adalah melakukan koneksi ke database dengan memanfaatkan class data context ApplicationDbContext. Caranya adalah dengan membuat object dari class tersebut seperti yang dapat dilihat pada kode berikut ini.

```

private readonly ApplicationDbContext _context;

public CategoriesController(ApplicationDbContext context)
{
    _context = context;
}

```

Dari kode di atas, object _context dapat digunakan untuk melakukan operasi ke database pada method action.

Pada class controller ini terdapat 8 method action, yaitu:

- [HttpGet]Index.
- [HttpGet]Details.
- [HttpGet]Create.
- [HttpPost]Create.
- [HttpGet]Edit.
- [HttpPost]Edit.
- [HttpGet]Delete.
- [HttpPost]Delete.

Method Action [HttpGet]Index

Method ini berfungsi untuk mengambil seluruh data pada database kemudian dikirimkan ke komponen view.

```

// GET: Categories
public async Task<IActionResult> Index()
{
    return View(await _context.Categories.ToListAsync());
}

```

Pada kode di atas dilakukan pengiriman data ke komponen view dengan cara berikut ini.

```
return View(DATA)
```

Isi dari DATA adalah data Category, untuk mengambil data dapat dilakukan dengan cara berikut ini.

```
_context.Categories.ToListAsync()
```

Komponen view dari method action Index ini adalah file Views/Categories/Index.cshtml.

Method Action [HttpGet]Details

Method ini berfungsi untuk mengambil sebuah data yang dipilih kemudian dikirimkan ke komponen view.

```
// GET: Categories/Details/5
public async Task<IActionResult> Details(int? id)
{
    if (id == null)
    {
        return NotFound();
    }

    var category = await _context.Categories
        .SingleOrDefaultAsync(m => m.CategoryID == id);
    if (category == null)
    {
        return NotFound();
    }

    return View(category);
}
```

Pada awal kode di atas dapat dilihat URL untuk menentukan nilai id dari data yang dipilih. Kemudian data dapat diambil dari database dengan cara berikut ini.

```
var category = await _context.Categories
    .SingleOrDefaultAsync(m => m.CategoryID == id);
```

Data akan disimpan pada object category yang selanjutnya akan dikirimkan ke komponen view dengan cara berikut.

```
return View(category);
```

Komponen view dari method action Details ini adalah file Views/Categories/Details.cshtml.

Method Action [HttpGet]Create

Method ini bertujuan untuk menampilkan form untuk menambah data. Pada kode di bawah ini dapat dilihat cara untuk menampilkan komponen view. Belum ada data yang dikirimkan ke komponen view.

```
// GET: Categories/Create
public IActionResult Create()
{
    return View();
}
```

Komponen view dari method action Create ini adalah file Views/Categories/Create.cshtml.

Method Action [HttpPost]Create

Setelah data diisi pada form input yang ditampilkan oleh method action di atas maka proses selanjutnya adalah menyimpan data.

```
[HttpPost]
[ValidateAntiForgeryToken]
public async Task<IActionResult> Create([Bind("CategoryID,Name")] Category category)
{
    if (ModelState.IsValid)
    {
        _context.Add(category);
        await _context.SaveChangesAsync();
        return RedirectToAction(nameof(Index));
    }
    return View(category);
}
```

Untuk mengirimkan data dari form umumnya digunakan method POST, sehingga method ini memiliki atribut [HttpPost]. Dan untuk mengamankan pemalsuan pengiriman data digunakan atribut [ValidateAntiForgeryToken].

Input untuk method ini adalah data yang dikirimkan dari komponen view, hal ini dapat dilihat pada parameter pada method ini.

```
([Bind("CategoryID,Name")] Category category)
```

Selanjutnya data akan ditambahkan dan disimpan dengan cara berikut ini.

```
_context.Add(category);
await _context.SaveChangesAsync();
```

Jika proses penyimpanan berhasil maka akan ditampilkan daftar data dengan cara mengklik komponen view Index.

```
return RedirectToAction(nameof(Index));
```

Jika proses penyimpanan gagal maka pengguna akan tetap berada pada halaman form input dan akan ditampilkan pesan kesalahan.

Method Action [HttpGet]Edit

Method ini bertujuan untuk menampilkan form untuk mengedit data. Pada kode di bawah ini dapat dilihat cara untuk menampilkan komponen view. Data yang akan ditampilkan adalah data yang dipilih untuk diedit.

```
// GET: Categories/Edit/5
public async Task<IActionResult> Edit(int? id)
{
    if (id == null)
    {
        return NotFound();
    }

    var category = await _context.Categories.SingleOrDefaultAsync(m =>
m.CategoryID == id);
```

```

    if (category == null)
    {
        return NotFound();
    }
    return View(category);
}

```

Untuk memilih data digunakan kode berikut ini.

```

var category = await _context.Categories.SingleOrDefault(m =>
m.CategoryID == id);

```

Data yang dipilih disimpan pada objek category. Kemudian untuk mengirimkan data ke komponen view digunakan kode berikut.

```

return View(category);

```

Komponen view dari method action Index ini adalah file Views/Categories/Edit.cshtml.

Method Action [HttpPost]Edit

Method action ini berfungsi untuk menyimpan data yang nilai-nilainya telah dimodifikasi pada form edit.

```

[HttpPost]
[ValidateAntiForgeryToken]
public async Task<IActionResult> Edit(int id, [Bind("CategoryID,Name")]
Category category)
{
    if (id != category.CategoryID)
    {
        return NotFound();
    }

    if (ModelState.IsValid)
    {
        try
        {
            _context.Update(category);
            await _context.SaveChangesAsync();
        }
        catch (DbUpdateConcurrencyException)
        {
            if (!CategoryExists(category.CategoryID))
            {
                return NotFound();
            }
            else
            {
                throw;
            }
        }
        return RedirectToAction(nameof(Index));
    }
    return View(category);
}

```

Cara kerja kode di atas mirip dengan cara kerja method action [HttpPost]Create. Yang membedakan adalah kode berikut ini.

```

_context.Update(category);
await _context.SaveChangesAsync();

```

Kode di atas adalah kode untuk mengupdate data.

Method Action [HttpGet]Delete

Method action ini bertujuan sebagai konfirmasi sebelum data dihapus. Data yang akan dihapus akan ditampilkan terlebih dahulu seperti menampilkan detail data seperti yang dilakukan oleh method action [HttpGet]Details. Berikut adalah kode dari method action ini.

```
// GET: Categories/Delete/5
public async Task<IActionResult> Delete(int? id)
{
    if (id == null)
    {
        return NotFound();
    }

    var category = await _context.Categories
        .SingleOrDefaultAsync(m => m.CategoryID == id);
    if (category == null)
    {
        return NotFound();
    }

    return View(category);
}
```

Komponen view dari method action Index ini adalah file Views/Categories/Delete.cshtml.

Method Action [HttpPost]Delete

Selanjutnya untuk menghapus data digunakan method action ini dengan isi kode sebagai berikut.

```
// POST: Categories/Delete/5
[HttpPost, ActionName("Delete")]
[ValidateAntiForgeryToken]
public async Task<IActionResult> DeleteConfirmed(int id)
{
    var category = await _context.Categories.SingleOrDefaultAsync(m =>
m.CategoryID == id);
    _context.Categories.Remove(category);
    await _context.SaveChangesAsync();
    return RedirectToAction(nameof(Index));
}
```

Nama method di atas adalah DeleteConfirmed, tetapi pada implementasinya akan dipanggil dengan nama Delete. Hal ini dapat dilihat dengan penggunaan atribut berikut.

```
[HttpPost, ActionName("Delete")]
```

Kemudian data yang akan dihapus ditampung terlebih dahulu dengan cara berikut.

```
var category = await _context.Categories.SingleOrDefaultAsync(m =>
m.CategoryID == id);
```

Dan untuk menampus data dari database digunakan kode berikut ini.

```
_context.Categories.Remove(category);
await _context.SaveChangesAsync();
```

Class Controller Authors

Untuk fitur pengelolaan pengarang buku dapat digunakan untuk:

- Menampilkan daftar pengarang buku.
- Menampilkan daftar pengarang buku yang dipilih.
- Menambah data pengarang buku.

Untuk menambah data pengarang buku terlebih dahulu perlu ditampilkan form input pengarang buku. Kemudian pengguna akan mengisi form input tersebut dan setelah tombol submit diklik akan dilakukan proses menyimpanan data yang dimasukkan ke dalam database.

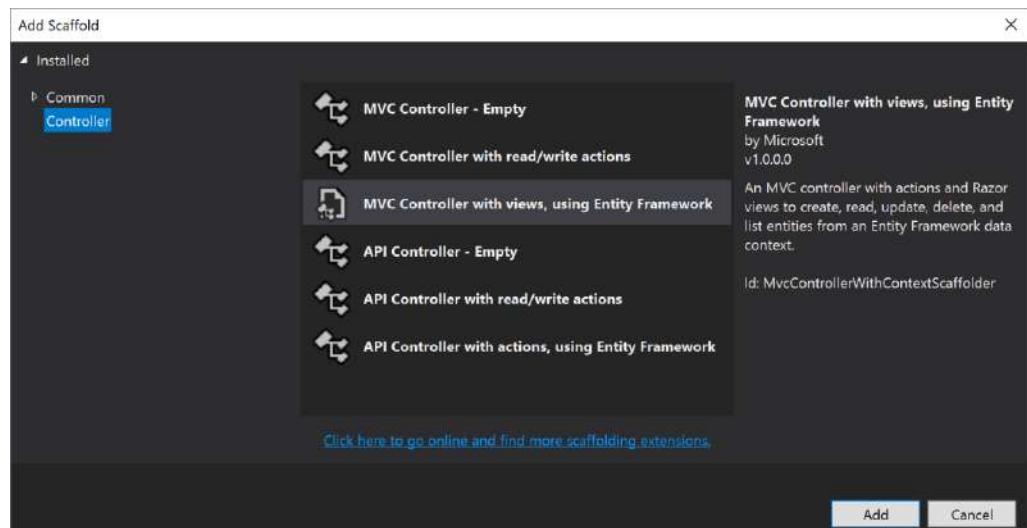
- Mengedit data pengarang buku yang dipilih.

Untuk mengedit data pengarang buku terlebih dahulu perlu ditampilkan form input edit pengarang buku. Kemudian pengguna akan mengedit data dari form edit. Setelah tombol submit diklik maka dilakukan proses update data ke dalam database.

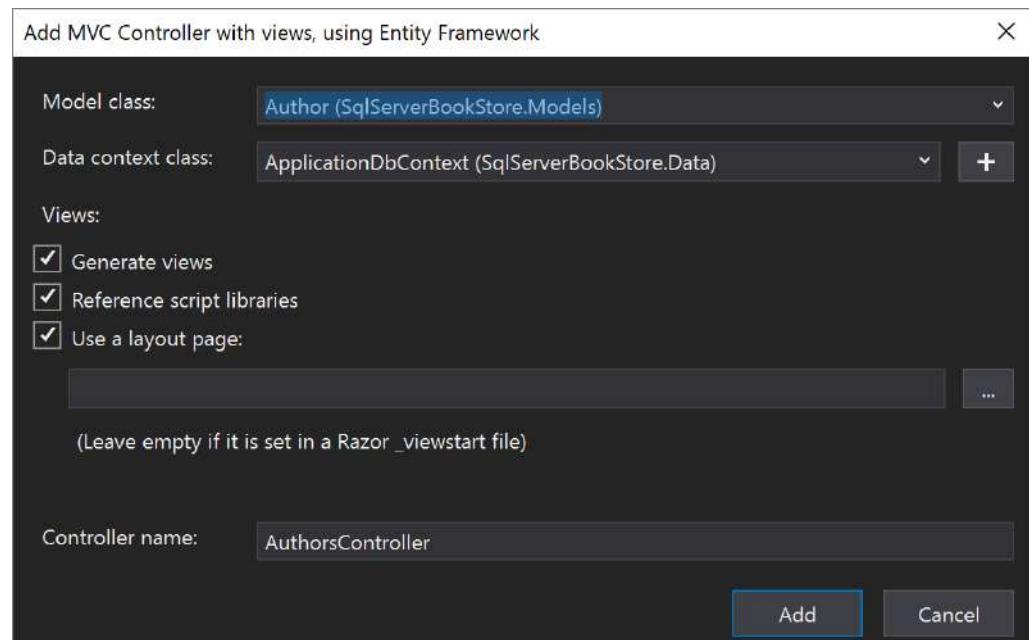
- Menghapus data pengarang buku yang dipilih.

Komponen controller untuk mengelola data Author akan memiliki nama dalam bentuk jamak yaitu Authors. Berikut ini adalah langkah-langkah untuk membuat class controller ini.

Langkah pertama klik kanan pada folder Controllers kemudian pilih Add > Controller. Pada window Add Scaffold pilih MVC Controller with views, using Entity Framework.



Gambar 122. AuthorsController - Window Add Scaffold.



Gambar 123. AuthorsController - Window Add MVC Controller with views, using Entity Framework.

Berikut adalah nilai-nilai yang harus dipilih dan diisi, yaitu:

- Model class, pilih komponen model Author.
- Data context class, pilih ApplicationDbContext.

Setelah tombol Add diklik maka hasilnya dapat dilihat pada file AuthorsController.cs di dalam folder Controllers. Berikut adalah isi dari file tersebut.

```
AuthorsController.cs
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.Mvc.Rendering;
using Microsoft.EntityFrameworkCore;
using SqlServerBookStore.Data;
using SqlServerBookStore.Models;

namespace SqlServerBookStore.Controllers
{
    public class AuthorsController : Controller
    {
        private readonly ApplicationDbContext _context;

        public AuthorsController(ApplicationDbContext context)
        {
            _context = context;
        }

        // GET: Authors
        public async Task<IActionResult> Index()
        {
            return View(await _context.Authors.ToListAsync());
        }

        // GET: Authors/Details/5
        public async Task<IActionResult> Details(int? id)
```

```

{
    if (id == null)
    {
        return NotFound();
    }

    var author = await _context.Authors
        .SingleOrDefaultAsync(m => m.AuthorID == id);
    if (author == null)
    {
        return NotFound();
    }

    return View(author);
}

// GET: Authors/Create
public IActionResult Create()
{
    return View();
}

// POST: Authors/Create
// To protect from overposting attacks, please enable the specific
properties you want to bind to, for
// more details see http://go.microsoft.com/fwlink/?LinkId=317598.
[HttpPost]
[ValidateAntiForgeryToken]
public async Task<IActionResult>
Create([Bind("AuthorID,Name,Email")] Author author)
{
    if (ModelState.IsValid)
    {
        _context.Add(author);
        await _context.SaveChangesAsync();
        return RedirectToAction(nameof(Index));
    }
    return View(author);
}

// GET: Authors/Edit/5
public async Task<IActionResult> Edit(int? id)
{
    if (id == null)
    {
        return NotFound();
    }

    var author = await _context.Authors.SingleOrDefaultAsync(m =>
m.AuthorID == id);
    if (author == null)
    {
        return NotFound();
    }
    return View(author);
}

// POST: Authors/Edit/5
// To protect from overposting attacks, please enable the specific
properties you want to bind to, for
// more details see http://go.microsoft.com/fwlink/?LinkId=317598.

```

```

[HttpPost]
[ValidateAntiForgeryToken]
public async Task<IActionResult> Edit(int id,
[Bind("AuthorID,Name,Email")] Author author)
{
    if (id != author.AuthorID)
    {
        return NotFound();
    }

    if (ModelState.IsValid)
    {
        try
        {
            _context.Update(author);
            await _context.SaveChangesAsync();
        }
        catch (DbUpdateConcurrencyException)
        {
            if (!AuthorExists(author.AuthorID))
            {
                return NotFound();
            }
            else
            {
                throw;
            }
        }
        return RedirectToAction(nameof(Index));
    }
    return View(author);
}

// GET: Authors/Delete/5
public async Task<IActionResult> Delete(int? id)
{
    if (id == null)
    {
        return NotFound();
    }

    var author = await _context.Authors
        .SingleOrDefaultAsync(m => m.AuthorID == id);
    if (author == null)
    {
        return NotFound();
    }

    return View(author);
}

// POST: Authors/Delete/5
[HttpPost, ActionName("Delete")]
[ValidateAntiForgeryToken]
public async Task<IActionResult> DeleteConfirmed(int id)
{
    var author = await _context.Authors.SingleOrDefaultAsync(m =>
m.AuthorID == id);
    _context.Authors.Remove(author);
    await _context.SaveChangesAsync();
    return RedirectToAction(nameof(Index));
}

```

```

        }

        private bool AuthorExists(int id)
        {
            return _context.Authors.Any(e => e.AuthorID == id);
        }
    }
}

```

Pada kode di atas, akan digunakan dua komponen model yaitu:

- Class data context ApplicationDbContext yang berada pada kelompok namespace SqlServerBookStore.Data. Class data context ini adalah class utama untuk melakukan koneksi dan operasi database.
- Class model entity Author yang berada pada kelompok namespace SqlServerBookStore.Models.

Sehingga pada kode di atas dapat dilihat cara untuk memuat kedua namespace tersebut dengan kode berikut ini.

```

using SqlServerBookStore.Data;
using SqlServerBookStore.Models;

```

Selanjutnya adalah melakukan koneksi ke database dengan memanfaatkan class data context ApplicationDbContext. Caranya adalah dengan membuat object dari class tersebut seperti yang dapat dilihat pada kode berikut ini.

```

private readonly ApplicationDbContext _context;

public CategoriesController(ApplicationDbContext context)
{
    _context = context;
}

```

Dari kode di atas, object _context dapat digunakan untuk melakukan operasi ke database pada method action.

Pada class controller ini terdapat 8 method action, yaitu:

- [HttpGet]Index.
- [HttpGet]Details.
- [HttpGet]Create.
- [HttpPost]Create.
- [HttpGet]Edit.
- [HttpPost]Edit.
- [HttpGet]Delete.
- [HttpPost]Delete.

Method Action [HttpGet]Index

Method ini berfungsi untuk mengambil seluruh data pada database kemudian dikirimkan ke komponen view.

```

// GET: Authors
public async Task<IActionResult> Index()
{
    return View(await _context.Authors.ToListAsync());
}

```

Pada kode di atas dilakukan pengiriman data ke komponen view dengan cara berikut ini.

```
return View(DATA)
```

Isi dari DATA adalah data Author, untuk mengambil data dapat dilakukan dengan cara berikut ini.

```
_context.Authors.ToListAsync()
```

Komponen view dari method action Index ini adalah file Views/Authors/Index.cshtml.

Method Action [HttpGet]Details

Method ini berfungsi untuk mengambil sebuah data yang dipilih kemudian dikirimkan ke komponen view.

```
// GET: Authors/Details/5
public async Task<IActionResult> Details(int? id)
{
    if (id == null)
    {
        return NotFound();
    }

    var author = await _context.Authors
        .SingleOrDefaultAsync(m => m.AuthorID == id);
    if (author == null)
    {
        return NotFound();
    }

    return View(author);
}
```

Pada awal kode di atas dapat dilihat URL untuk menentukan nilai id dari data yang dipilih. Kemudian data dapat diambil dari database dengan cara berikut ini.

```
var author = await _context.Authors
    .SingleOrDefaultAsync(m => m.AuthorID == id);
```

Data akan disimpan pada object author yang selanjutnya akan dikirimkan ke komponen view dengan cara berikut.

```
return View(author);
```

Komponen view dari method action Details ini adalah file Views/Authors/Details.cshtml.

Method Action [HttpGet]Create

Method ini bertujuan untuk menampilkan form untuk menambah data. Pada kode di bawah ini dapat dilihat cara untuk menampilkan komponen view. Belum ada data yang dikirimkan ke komponen view.

```
// GET: Authors/Create
public IActionResult Create()
{
    return View();
}
```

Komponen view dari method action Create ini adalah file Views/Authors/Create.cshtml.

Method Action [HttpPost]Create

Setelah data diisi pada form input yang ditampilkan oleh method action di atas maka proses selanjutnya adalah menyimpan data.

```
[HttpPost]
[ValidateAntiForgeryToken]
public async Task<IActionResult> Create([Bind("AuthorID,Name,Email")] Author author)
{
    if (ModelState.IsValid)
    {
        _context.Add(author);
        await _context.SaveChangesAsync();
        return RedirectToAction(nameof(Index));
    }
    return View(author);
}
```

Untuk mengirimkan data dari form umumnya digunakan method POST, sehingga method ini memiliki atribut [HttpPost]. Dan untuk mengamankan pemalsuan pengiriman data digunakan atribut [ValidateAntiForgeryToken].

Input untuk method ini adalah data yang dikirimkan dari komponen view, hal ini dapat dilihat pada parameter pada method ini.

```
[Bind("AuthorID,Name,Email")] Author author
```

Selanjutnya data akan ditambahkan dan disimpan dengan cara berikut ini.

```
_context.Add(author);
await _context.SaveChangesAsync();
```

Jika proses penyimpanan berhasil maka akan ditampilkan daftar data dengan cara mengkases komponen view Index.

```
return RedirectToAction(nameof(Index));
```

Jika proses penyimpanan gagal maka pengguna akan tetap berada pada halaman form input dan akan ditampilkan pesan kesalahan.

Method Action [HttpGet]Edit

Method ini bertujuan untuk menampilkan form untuk mengedit data. Pada kode di bawah ini dapat dilihat cara untuk menampilkan komponen view. Data yang akan ditampilkan adalah data yang dipilih untuk diedit.

```
// GET: Authors/Edit/5
public async Task<IActionResult> Edit(int? id)
{
    if (id == null)
    {
        return NotFound();
    }

    var author = await _context.Authors.SingleOrDefaultAsync(m => m.AuthorID
== id);
    if (author == null)
    {
```

```

        return NotFound();
    }
    return View(author);
}

```

Untuk memilih data digunakan kode berikut ini.

```

var author = await _context.Authors.SingleOrDefault(m => m.AuthorID ==
id);

```

Data yang dipilih disimpan pada objek author. Kemudian untuk mengirimkan data ke komponen view digunakan kode berikut.

```

return View(author);

```

Komponen view dari method action Index ini adalah file Views/Authors/Edit.cshtml.

Method Action [HttpPost]Edit

Method action ini berfungsi untuk menyimpan data yang nilai-nilainya telah dimodifikasi pada form edit.

```

[HttpPost]
[ValidateAntiForgeryToken]
public async Task<IActionResult> Edit(int id, [Bind("AuthorID,Name,Email")]
Author author)
{
    if (id != author.AuthorID)
    {
        return NotFound();
    }

    if (ModelState.IsValid)
    {
        try
        {
            _context.Update(author);
            await _context.SaveChangesAsync();
        }
        catch (DbUpdateConcurrencyException)
        {
            if (!AuthorExists(author.AuthorID))
            {
                return NotFound();
            }
            else
            {
                throw;
            }
        }
        return RedirectToAction(nameof(Index));
    }
    return View(author);
}

```

Cara kerja kode di atas mirip dengan cara kerja method action [HttpPost]Create. Yang membedakan adalah kode berikut ini.

```

_context.Update(author);
await _context.SaveChangesAsync();

```

Kode di atas adalah kode untuk mengupdate data.

Method Action [HttpGet]Delete

Method action ini bertujuan sebagai konfirmasi sebelum data dihapus. Data yang akan dihapus akan ditampilkan terlebih dahulu seperti menampilkan detail data seperti yang dilakukan oleh method action [HttpGet]Details. Berikut adalah kode dari method action ini.

```
// GET: Authors/Delete/5
public async Task<IActionResult> Delete(int? id)
{
    if (id == null)
    {
        return NotFound();
    }

    var author = await _context.Authors
        .SingleOrDefaultAsync(m => m.AuthorID == id);
    if (author == null)
    {
        return NotFound();
    }

    return View(author);
}
```

Komponen view dari method action Index ini adalah file Views/Authors/Delete.cshtml.

Method Action [HttpPost]Delete

Selanjutnya untuk menghapus data digunakan method action ini dengan isi kode sebagai berikut.

```
[HttpPost, ActionName("Delete")]
[ValidateAntiForgeryToken]
public async Task<IActionResult> DeleteConfirmed(int id)
{
    var author = await _context.Authors.SingleOrDefaultAsync(m => m.AuthorID
== id);
    _context.Authors.Remove(author);
    await _context.SaveChangesAsync();
    return RedirectToAction(nameof(Index));
}
```

Nama method di atas adalah DeleteConfirmed, tetapi pada implementasinya akan dipanggil dengan nama Delete. Hal ini dapat dilihat dengan penggunaan atribut berikut.

```
[HttpPost, ActionName("Delete")]
```

Kemudian data yang akan dihapus ditampung terlebih dahulu dengan cara berikut.

```
var author = await _context.Authors.SingleOrDefaultAsync(m => m.AuthorID ==
id);
```

Dan untuk menampus data dari database digunakan kode berikut ini.

```
_context.Authors.Remove(author);
await _context.SaveChangesAsync();
```

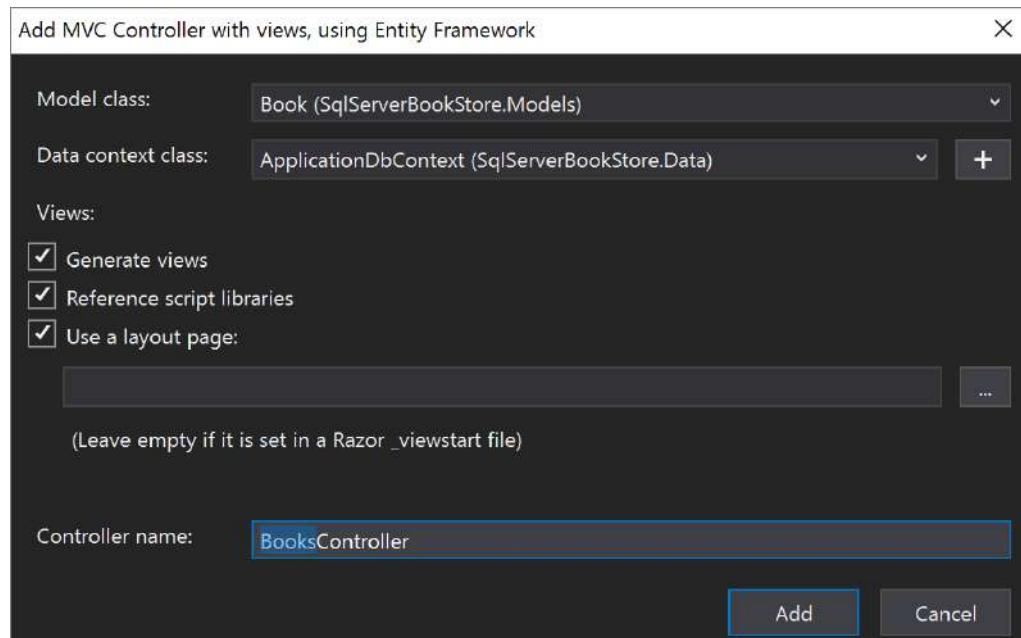
Class Controller Books

Untuk fitur pengelolaan buku dapat digunakan untuk:

- Menampilkan daftar buku.
- Menampilkan detail buku yang dipilih.
- Menambah data buku.
Untuk menambah data buku terlebih dahulu perlu ditampilkan form input buku. Kemudian pengguna akan mengisi form input tersebut dan setelah tombol submit diklik akan dilakukan proses menyimpanan data yang dimasukkan ke dalam database.
- Mengedit data buku yang dipilih.
Untuk mengedit data buku terlebih dahulu perlu ditampilkan form input edit buku. Kemudian pengguna akan mengedit data dari form edit. Setelah tombol submit diklik maka dilakukan proses update data ke dalam database.
- Menghapus data buku yang dipilih.

Maka langkah pertama adalah akan dibuat komponen controller dan view dengan cara berikut ini.

Klik kanan pada folder Controllers, kemudian pilih Add > Controller. Kemudian pada window Add Scaffold, pilih MVC Controller with views, using Entity Framework. Setelah itu ditampilkan window Add MVC Controller with views, using Entity Framework seperti pada gambar di bawah ini.



Gambar 124. BooksController - Window Add MVC Controller with views, using Entity Framework.

Berikut adalah nilai-nilai yang harus dipilih dan diisi, yaitu:

- Model class, pilih komponen model Book.
- Data context class, pilih ApplicationDbContext.

Setelah tombol Add diklik maka hasilnya dapat dilihat pada file BooksController.cs di dalam folder Controllers. Berikut adalah isi dari file tersebut.

```
BooksController.cs
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Mvc;
```

```

using Microsoft.AspNetCore.Mvc.Rendering;
using Microsoft.EntityFrameworkCore;
using SqlServerBookStore.Data;
using SqlServerBookStore.Models;

namespace SqlServerBookStore.Controllers
{
    public class Books1Controller : Controller
    {
        private readonly ApplicationDbContext _context;

        public Books1Controller(ApplicationDbContext context)
        {
            _context = context;
        }

        // GET: Books1
        public async Task<IActionResult> Index()
        {
            var applicationDbContext = _context.Books.Include(b => b.Category);
            return View(await applicationDbContext.ToListAsync());
        }

        // GET: Books1/Details/5
        public async Task<IActionResult> Details(int? id)
        {
            if (id == null)
            {
                return NotFound();
            }

            var book = await _context.Books
                .Include(b => b.Category)
                .SingleOrDefaultAsync(m => m.BookID == id);
            if (book == null)
            {
                return NotFound();
            }

            return View(book);
        }

        // GET: Books1/Create
        public IActionResult Create()
        {
            ViewData["CategoryID"] = new SelectList(_context.Categories, "CategoryID", "Name");
            return View();
        }

        // POST: Books1/Create
        // To protect from overposting attacks, please enable the specific properties you want to bind to, for
        // more details see http://go.microsoft.com/fwlink/?LinkId=317598.
        [HttpPost]
        [ValidateAntiForgeryToken]
        public async Task<IActionResult> Create([Bind("BookID,CategoryID,Title,Photo,PublishDate,Price,Quantity")] Book book)
        {
            if (ModelState.IsValid)

```

```

    {
        _context.Add(book);
        await _context.SaveChangesAsync();
        return RedirectToAction(nameof(Index));
    }
    ViewData["CategoryID"] = new SelectList(_context.Categories, "Ca
toryID", "Name", book.CategoryID);
    return View(book);
}

// GET: Books1/Edit/5
public async Task<IActionResult> Edit(int? id)
{
    if (id == null)
    {
        return NotFound();
    }

    var book = await _context.Books.SingleOrDefaultAsync(m => m.Book
ID == id);
    if (book == null)
    {
        return NotFound();
    }
    ViewData["CategoryID"] = new SelectList(_context.Categories, "Ca
toryID", "Name", book.CategoryID);
    return View(book);
}

// POST: Books1/Edit/5
// To protect from overposting attacks, please enable the specific p
roperties you want to bind to, for
// more details see http://go.microsoft.com/fwlink/?LinkId=317598.
[HttpPost]
[ValidateAntiForgeryToken]
public async Task<IActionResult> Edit(int id, [Bind("BookID,Category
ID,Title,Photo,PublishDate,Price,Quantity")] Book book)
{
    if (id != book.BookID)
    {
        return NotFound();
    }

    if (ModelState.IsValid)
    {
        try
        {
            _context.Update(book);
            await _context.SaveChangesAsync();
        }
        catch (DbUpdateConcurrencyException)
        {
            if (!BookExists(book.BookID))
            {
                return NotFound();
            }
            else
            {
                throw;
            }
        }
    }
}

```

```

        return RedirectToAction(nameof(Index));
    }
    ViewData["CategoryID"] = new SelectList(_context.Categories, "CategoryID", "Name", book.CategoryID);
    return View(book);
}

// GET: Books1/Delete/5
public async Task<IActionResult> Delete(int? id)
{
    if (id == null)
    {
        return NotFound();
    }

    var book = await _context.Books
        .Include(b => b.Category)
        .SingleOrDefaultAsync(m => m.BookID == id);
    if (book == null)
    {
        return NotFound();
    }

    return View(book);
}

// POST: Books1/Delete/5
[HttpPost, ActionName("Delete")]
[ValidateAntiForgeryToken]
public async Task<IActionResult> DeleteConfirmed(int id)
{
    var book = await _context.Books.SingleOrDefault(m => m.Book
ID == id);
    _context.Books.Remove(book);
    await _context.SaveChangesAsync();
    return RedirectToAction(nameof(Index));
}

private bool BookExists(int id)
{
    return _context.Books.Any(e => e.BookID == id);
}
}
}

```

Class controller Books ini tidak hanya akan menggunakan class entity model, tetapi juga akan menggunakan class view model. Class entity model digunakan untuk melakukan operasi dasar database yaitu create, retrieve, update dan delete (CRUD). Sedangkan untuk menampilkan data ke komponen view akan digunakan clas view model. Hal ini perlu dilakukan untuk menyesuaikan dengan antarmuka halaman view yang diinginkan.

Untuk membuat hal seperti itu, maka perlu ada modifikasi kode class controller yang telah dibuat secara otomatis oleh Visual Studio. Berikut adalah kode lengkap BooksController yang telah dimodifikasi.

BooksController.cs
<pre> using System; using System.Collections.Generic; using System.Linq; using System.Threading.Tasks; using Microsoft.AspNetCore.Mvc; </pre>

```

using Microsoft.AspNetCore.Mvc.Rendering;
using Microsoft.EntityFrameworkCore;
using Microsoft.AspNetCore.Hosting;
using System.IO;
using SqlServerBookStore.Data;
using SqlServerBookStore.Models;

namespace SqlServerBookStore.Controllers
{
    public class BooksController : Controller
    {
        private readonly ApplicationDbContext _context;
        private IHostingEnvironment _environment;

        public BooksController(ApplicationDbContext context, IHostingEnvironment environment)
        {
            _context = context;
            _environment = environment;
        }

        // GET: Books
        public IActionResult Index()
        {
            var bookList = _context.Books
                .Include(c => c.Category)
                .Include(ba => ba.BooksAuthors)
                .ThenInclude(a => a.Author)
                .ToList();

            IList<BookViewModel> items = new List<BookViewModel>();
            foreach (Book book in bookList)
            {
                BookViewModel item = new BookViewModel();

                item.ISBN = book.BookID;
                item.Title = book.Title;
                item.Photo = book.Photo;
                item.PublishDate = book.PublishDate;
                item.Price = book.Price;
                item.Quantity = book.Quantity;
                item.CategoryName = book.Category.Name;

                string authorNameList = string.Empty;
                var booksAuthorsList = book.BooksAuthors;
                foreach (BookAuthor booksAuthors in booksAuthorsList)
                {
                    var author = booksAuthors.Author;
                    authorNameList = authorNameList + author.Name + ", ";
                }
                item.AuthorNames = authorNameList.Substring(0, authorNameList.Length - 2);

                items.Add(item);
            }
            return View(items);
        }

        // GET: Books/Details/5
        public IActionResult Details(int? id)
        {

```

```

        var book = _context.Books
            .Include(c => c.Category)
            .Include(ba => ba.BooksAuthors)
            .ThenInclude(a => a.Author).SingleOrDefault(b => b.BookID.Equals(id));

        BookViewModel item = new BookViewModel();
        item.ISBN = book.BookID;
        item.Title = book.Title;
        item.PublishDate = book.PublishDate;
        item.Price = book.Price;
        item.Quantity = book.Quantity;
        item.CategoryName = book.Category.Name;

        string authorNameList = string.Empty;
        var booksAuthorsList = book.BooksAuthors;
        foreach (BookAuthor booksAuthors in booksAuthorsList)
        {
            var author = booksAuthors.Author;
            authorNameList = authorNameList + author.Name + ", ";
        }
        item.AuthorNames = authorNameList.Substring(0, authorNameList.Length - 2);

        return View(item);
    }

    // GET: Books/Create
    public IActionResult Create()
    {
        // ViewData["CategoryID"] = new SelectList(_context.Categories, "CategoryID", "Name");
        ViewBag.Categories = new SelectList(_context.Categories.ToList(), "CategoryID", "Name");
        ViewBag.Authors = new MultiSelectList(_context.Authors.ToList(), "AuthorID", "Name");

        return View();
    }

    [HttpPost]
    [ValidateAntiForgeryToken]
    public IActionResult Create(BookFormViewModel item)
    {
        if (ModelState.IsValid)
        {
            Book book = new Book();
            book.BookID = item.ISBN;
            book.CategoryID = item.CategoryID;
            book.Title = item.Title;
            book.PublishDate = item.PublishDate;
            book.Price = item.Price;
            book.Quantity = item.Quantity;
            _context.Add(book);

            foreach (int authorId in item.AuthorIDs)
            {
                BookAuthor bookAuthor = new BookAuthor();
                bookAuthor.BookID = item.ISBN;
                bookAuthor.AuthorID = authorId;
                _context.Add(bookAuthor);
            }
        }
    }
}

```

```

        }

        _context.SaveChanges();

        if (item.Photo != null)
        {
            var file = item.Photo;
            var uploads = Path.Combine(_environment.WebRootPath, "up
load");
            if (file.Length > 0)
            {
                using (var fileStream = new FileStream(Path.Combine(
uploads, item.ISBN + ".jpg"), FileMode.Create))
                {
                    file.CopyToAsync(fileStream);
                }
            }
        }

        return RedirectToAction("Index");
    }

    return View();
}

[HttpGet]
public IActionResult Edit(int? id)
{
    ViewBag.Categories = new SelectList(_context.Categories.ToList()
, "CategoryID", "Name");
    ViewBag.Authors = new MultiSelectList(_context.Authors.ToList(),
"AuthorID", "Name");

    var book = _context.Books.SingleOrDefault(p => p.BookID.Equals(i
d));

    BookFormViewModel item = new BookFormViewModel();
    item.ISBN = book.BookID;
    item.Title = book.Title;
    item.PublishDate = book.PublishDate;
    item.Price = book.Price;
    item.Quantity = book.Quantity;
    item.CategoryID = book.CategoryID;

    var authorList = _context.BooksAuthors.Where(p => p.BookID.Equal
s(book.BookID)).ToList();
    List<int> authors = new List<int>();
    foreach (BookAuthor bookAuthor in authorList)
    {
        authors.Add(bookAuthor.AuthorID);
    }
    item.AuthorIDs = authors.ToArray();

    return View(item);
}

[HttpPost]
[ValidateAntiForgeryToken]
public IActionResult Edit([Bind("ISBN, CategoryID, Title, Photo, Pub
lishDate, Price, Quantity, AuthorIDs")] BookFormViewModel item)
{
}

```

```

        if (ModelState.IsValid)
        {
            _context.BooksAuthors.RemoveRange(_context.BooksAuthors.Where(p => p.BookID.Equals(item.ISBN)));
            _context.SaveChanges();

            Book book = _context.Books.SingleOrDefault(p => p.BookID.Equals(item.ISBN));
            book.CategoryID = item.CategoryID;
            book.Title = item.Title;
            book.PublishDate = item.PublishDate;
            book.Price = item.Price;
            book.Quantity = item.Quantity;
            _context.Update(book);

            foreach (int authorId in item.AuthorIDs)
            {
                BookAuthor bookAuthor = new BookAuthor();
                bookAuthor.BookID = item.ISBN;
                bookAuthor.AuthorID = authorId;
                _context.Add(bookAuthor);
            }

            _context.SaveChanges();

            if (item.Photo != null)
            {
                var file = item.Photo;
                var uploads = Path.Combine(_environment.WebRootPath, "up
load");
                if (file.Length > 0)
                {
                    using (var fileStream = new FileStream(Path.Combine(
uploads, item.ISBN + ".jpg"), FileMode.Create))
                    {
                        file.CopyToAsync(fileStream);
                    }
                }
            }
        }

        return RedirectToAction("Index");
    }

    return View();
}

// GET: Books/Delete/5
public IActionResult Delete(int? id)
{
    var book = _context.Books
        .Include(c => c.Category)
        .Include(ba => ba.BooksAuthors)
        .ThenInclude(a => a.Author).SingleOrDefault(b => b.BookID.Eq
uals(id));

    BookViewModel item = new BookViewModel();
    item.ISBN = book.BookID;
    item.Title = book.Title;
    item.PublishDate = book.PublishDate;
    item.Price = book.Price;
    item.Quantity = book.Quantity;
}

```

```

        item.CategoryName = book.Category.Name;

        string authorNameList = string.Empty;
        var booksAuthorsList = book.BooksAuthors;
        foreach (BookAuthor booksAuthors in booksAuthorsList)
        {
            var author = booksAuthors.Author;
            authorNameList = authorNameList + author.Name + ", ";
        }
        item.AuthorNames = authorNameList.Substring(0, authorNameList.Length - 2);

        return View(item);
    }

    [HttpPost, ActionName("Delete")]
    [ValidateAntiForgeryToken]
    public IActionResult DeleteConfirmed(int id)
    {
        if (ModelState.IsValid)
        {
            _context.BooksAuthors.RemoveRange(_context.BooksAuthors.Where(p => p.BookID.Equals(id)));
            _context.SaveChanges();

            var book = _context.Books.SingleOrDefault(m => m.BookID == id);
            _context.Books.Remove(book);
            _context.SaveChanges();

            return RedirectToAction("Index");
        }

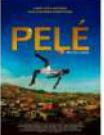
        return View();
    }

    private bool BookExists(int id)
    {
        return _context.Books.Any(e => e.BookID == id);
    }
}

```

Method Action [HttpGet]Index

Berikut ini adalah gambar antarmuka untuk menampilkan data buku yang ingin dibuat.

List of Book						
Cover	Category	Title	Publish Date	Qty	Action	
	Sejarah	Sejarah Jawa ISBN: 1233 Authors: Reza Price: 12	12/12/2012 12:00:00 AM	12	Details Edit Delete	
	Informatika	Bioinformatika ISBN: 2345 Authors: Reza, Faisal Price: 4	4/4/2004 12:00:00 AM	4	Details Edit Delete	

Gambar 125. BooksController - Daftar buku.

Selanjutnya seluruh kode method tersebut akan diganti dengan kode sebagai berikut ini. Langkah pertama yang dilakukan adalah mengambil data buku dari tabel books dengan cara berikut ini.

```
var bookList = _context.Books
    .Include(c => c.Category)
    .Include(ba => ba.BooksAuthors)
    .ThenInclude(a => a.Author)
    .ToList();
```

Cara di atas adalah cara lain untuk mendapatkan seluruh data book. Pada kode di atas dapat dilihat penggunaan kata kunci `Include` yang berfungsi untuk memuat data yang terkait dengan data book yang dipilih. Dari contoh di atas artinya akan dimuat data `BooksAuthors` yang sesuai dengan data `Book`. Kemudian dengan kata kunci `ThenInclude` akan memuat data `Author` yang sesuai dengan data `BooksAuthor` yang telah dimuat sebelumnya.

Hasilnya disimpan pada object `bookList`. Selanjutnya setiap object `Book` akan disimpan pada object `BookViewModel`. Kumpulan object `BookViewModel` akan ditampung di dalam collection bertipe `List` dengan cara seperti berikut.

```
IList<BookViewModel> items = new List<BookViewModel>();
foreach (Book book in bookList)
{
    BookViewModel item = new BookViewModel();

    item.ISBN = book.BookID;
    item.Title = book.Title;
    item.Photo = book.Photo;
    item.PublishDate = book.PublishDate;
    item.Price = book.Price;
    item.Quantity = book.Quantity;
    item.CategoryName = book.Category.Name;
    ...
    items.Add(item);
}
```

Dari contoh di atas object object `items` dipersiapkan untuk menyimpan object `BookViewModel`. Kemudian dengan melakukan pengulangan dilakukan pengambilan dari object `bookList`. Selanjutnya dibuat object `item` yang merupakan instansiasi dari class view model `BookViewModel`. Pada kode di atas dapat dilihat object `item` diisi dengan nilai-nilai seperti ISBN, Title, Photo, PublishDate, Price dan Quantity.

```
item.CategoryName = book.Category.Name;
```

Sedangkan untuk mendapatkan nama kategori buku dapat langsung dilakukan dengan cara di atas. Hal ini bisa dilakukan karena pada model Book dan Category telah memiliki relasi.

Sebuah buku dapat ditulis oleh lebih dari pengarang. Nama-nama pengarang tersebut akan disimpan dalam object authorNameList yang bertipe string yang akan dipisahkan oleh tanda koma (,). Selanjutnya adalah mengambil relasi antara pengarang buku dengan buku dengan cara berikut ini.

```
var booksAuthorsList = book.BooksAuthors;
```

Selanjutnya membaca object booksAuthorList untuk mendapatkan AuthorID yang akan digunakan untuk mendapatkan nama pengarang untuk disimpan ke dalam object authorNameList. Berikut adalah kode yang digunakan untuk keperluan itu.

```
string authorNameList = string.Empty;
var booksAuthorsList = book.BooksAuthors;
foreach (BookAuthor booksAuthors in booksAuthorsList)
{
    var author = booksAuthors.Author;
    authorNameList = authorNameList + author.Name + ", ";
}
item.AuthorNames = authorNameList.Substring(0, authorNameList.Length - 2);
```

Objek items akan menyimpan data dari tabel author dalam bentuk IList. Selanjutnya object items akan dikirim ke komponen view Index.cshtml dengan cara berikut.

```
return View(items);
```

Method Action [HttpGet]Details

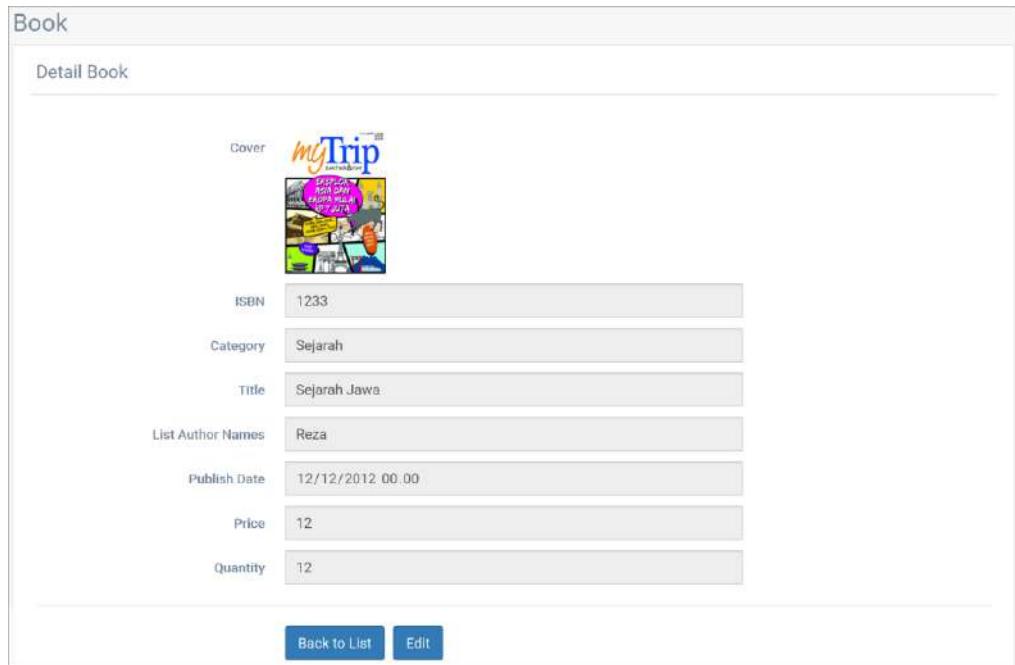
Method ini berfungsi untuk mengambil sebuah data yang dipilih kemudian dikirimkan ke komponen view. Berikut ini adalah kode sebelum dimodifikasi.

```
// GET: Books/Details/5
public async Task<IActionResult> Details(int? id)
{
    if (id == null)
    {
        return NotFound();
    }

    var book = await _context.Books
        .Include(b => b.Category)
        .SingleOrDefaultAsync(m => m.BookID == id);
    if (book == null)
    {
        return NotFound();
    }

    return View(book);
}
```

Modifikasi kode ini diperlukan karena ingin menampilkan detail data buku seperti berikut ini.



Gambar 126. BooksController - Detail.

Berikut adalah kode method action Details setelah dimodifikasi.

```
public IActionResult Details(int? id)
{
    var book = _context.Books
        .Include(c => c.Category)
        .Include(ba => ba.BooksAuthors)
        .ThenInclude(a => a.Author).SingleOrDefault(b => b.BookID.Equals(id));

    BookViewModel item = new BookViewModel();
    item.ISBN = book.BookID;
    item.Title = book.Title;
    item.PublishDate = book.PublishDate;
    item.Price = book.Price;
    item.Quantity = book.Quantity;
    item.CategoryName = book.Category.Name;

    string authorNameList = string.Empty;
    var booksAuthorsList = book.BooksAuthors;
    foreach (BookAuthor booksAuthors in booksAuthorsList)
    {
        var author = booksAuthors.Author;
        authorNameList = authorNameList + author.Name + ", ";
    }
    item.AuthorNames = authorNameList.Substring(0, authorNameList.Length - 2);
}

return View(item);
}
```

Isi kode di atas memiliki kemiripan dengan method action Index. Perbedaan paling mendasar adalah cara mengambil data buku berdasarkan id, seperti yang dapat dilihat pada cetak tebal.

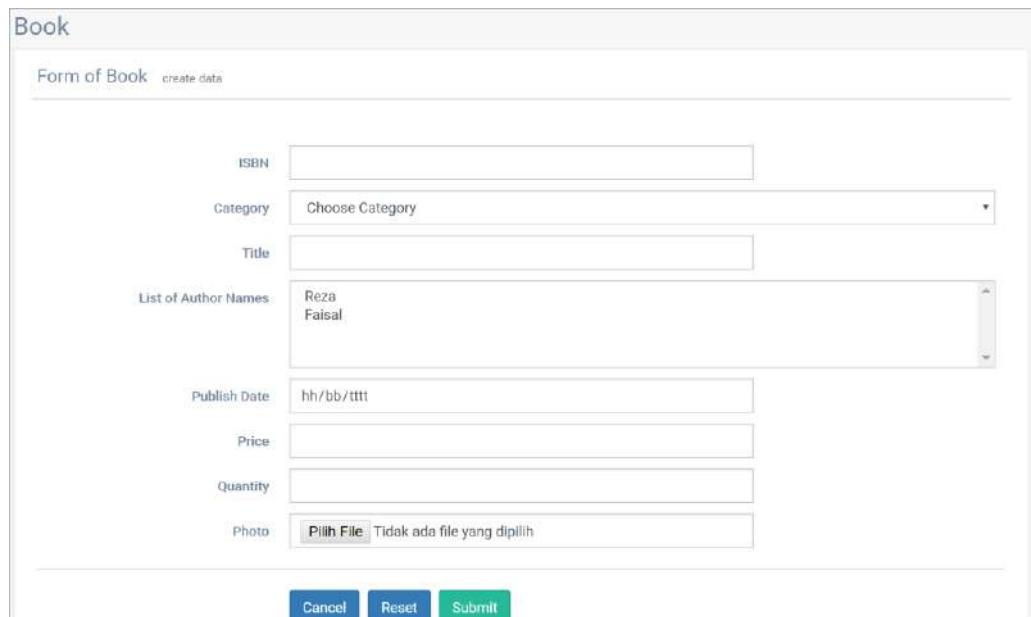
```
var book = _context.Books
    .Include(c => c.Category)
    .Include(ba => ba.BooksAuthors)
    .ThenInclude(a => a.Author).SingleOrDefault(b => b.BookID.Equals(id));
```

Method Action [HttpGet]Create

Berikut ini adalah kode method action Create.

```
// GET: Books/Create
public IActionResult Create()
{
    ViewData["CategoryID"] = new SelectList(_context.Categories,
"CategoryID", "Name");
    return View();
}
```

Method action ini bertujuan untuk menampilkan form untuk menambah data buku. File komponen view yang digunakan sebagai form ini adalah Create.cshtml. Berikut adalah tampilan form untuk menambah data buku.



Gambar 127. BooksController - Form menambah buku.

Berikut adalah isi kode method action Create yang telah dimodifikasi.

```
public IActionResult Create()
{
    ViewBag.Categories = new SelectList(_context.Categories.ToList(),
"CategoryID", "Name");
    ViewBag.Authors = new MultiSelectList(_context.Authors.ToList(),
"AuthorID", "Name");

    return View();
}
```

Pada input Category dan List of Author Names dapat dilihat ditampilkan data dari tabel Categories dan Authors. Sehingga pada method action ini dapat dilihat kode pengambilan data dari kedua tabel tersebut dengan menggunakan kode di atas.

Object ViewBag.Categories menyimpan data tabel categories. Dan object ViewBag.Authors menyimpan data dari tabel authors. Selanjutnya dapat dilihat bagaimana kedua object ini ditampilkan halaman komponen view Create.cshtml.

Method Action [HttpPost]Create

Berikut adalah isi method action Create sebelum dimodifikasi.

```
[HttpPost]
[ValidateAntiForgeryToken]
public async Task<IActionResult> Create([Bind("BookID,CategoryID,Title,Photo
,PublishDate,Price,Quantity")] Book book)
{
    if (ModelState.IsValid)
    {
        _context.Add(book);
        await _context.SaveChangesAsync();
        return RedirectToAction(nameof(Index));
    }
    ViewData["CategoryID"] = new SelectList(_context.Categories, "CategoryID"
    , "Name", book.CategoryID);
    return View(book);
}
```

Kode di atas perlu dimodifikasi untuk membuat form tambah data buku seperti pada gambar berikut ini.

Gambar 128. BooksController - Create.

Dan berikut ini adalah isi method Create yang telah dimodifikasi.

```
[HttpPost]
[ValidateAntiForgeryToken]
public IActionResult Create(BookFormViewModel item)
{
    if (ModelState.IsValid)
    {
        Book book = new Book();
        book.BookID = item.ISBN;
        book.CategoryID = item.CategoryID;
        book.Title = item.Title;
        book.PublishDate = item.PublishDate;
        book.Price = item.Price;
        book.Quantity = item.Quantity;
        _context.Add(book);

        foreach (int authorId in item.AuthorIDs)
        {
            BookAuthor bookAuthor = new BookAuthor();
```

```

        bookAuthor.BookID = item.ISBN;
        bookAuthor.AuthorID = authorId;
        _context.Add(bookAuthor);
    }

    _context.SaveChanges();

    if (item.Photo != null)
    {
        var file = item.Photo;
        var uploads = Path.Combine(_environment.WebRootPath, "upload");
        if (file.Length > 0)
        {
            using (var fileStream = new FileStream(Path.Combine(uploads,
item.ISBN + ".jpg"), FileMode.Create))
            {
                file.CopyToAsync(fileStream);
            }
        }
    }

    return RedirectToAction("Index");
}

return View();
}

```

Method action ini bertujuan untuk menyimpan nilai-nilai dari object BookFormViewModel yang property-propertinya telah diisi melalui form pada file Create.cshtml. Agar method ini dapat menerima kiriman object file Create.cshtml maka dapat dilihat parameter input item seperti pada kode berikut ini.

```

public IActionResult Create(BookFormViewModel item)
{
    ...
}

```

Selanjutnya untuk memvalidasidi item sebelum disimpan dapat dilakukan dengan cara berikut.

```

if(ModelState.IsValid){
    ...
}

```

Karena object BookFormViewModel bukan object dari class entity model, maka model ini tidak dapat langsung digunakan untuk menyimpan data ke database.

Langkah pertama adalah menyimpan beberapa nilai-nilai dari object BookFormViewModel ke dalam object Book dengan cara berikut ini.

```

Book book = new Book();
book.BookID = item.ISBN;
book.CategoryID = item.CategoryID;
book.Title = item.Title;
book.PublishDate = item.PublishDate;
book.Price = item.Price;
book.Quantity = item.Quantity;
_context.Add(book);

```

Dari kode di atas dapat dilihat sebagian nilai dari object BookFormViewModel telah dimasukkan ke dalam object entity model Book sehingga bisa disimpan ke dalam database. Selanjutnya menyimpan nilai property AuthorIDs dari object BookFormViewModel yang berisi daftar pengarang buku yang dipilih pada form dengan cara berikut ini.

```

foreach (int authorId in item.AuthorIDs)
{
    BookAuthor bookAuthor = new BookAuthor();
    bookAuthor.BookID = item.ISBN;
    bookAuthor.AuthorID = authorId;
    _context.Add(bookAuthor);
}

```

Dari kode di atas dapat dilihat nilai id buku dan id pengarang disimpan di dalam object entity model BookAuthor. Setelah seluruh nilai-nilai dari object BookFormViewModel dipindahkan ke object-object entity model Book dan BookAuthor maka selanjutnya dapat dilakukan penyimpanan data dengan cara berikut.

```

_context.SaveChanges();

```

Langkah selanjutnya adalah menangani proses upload file dan menyimpan file tersebut ke folder upload pada folder wwwroot. Berikut adalah kode yang digunakan untuk menangani hal tersebut.

```

if (item.Photo != null)
{
    var file = item.Photo;
    var uploads = Path.Combine(_environment.WebRootPath, "upload");
    if (file.Length > 0)
    {
        using (var fileStream = new FileStream(Path.Combine(uploads,
item.ISBN + ".jpg"), FileMode.Create))
        {
            file.CopyToAsync(fileStream);
        }
    }
}

```

Setelah data dan file yang diupload sukses disimpan, selanjutnya akan kembali ditampilkan daftar pengarang buku dengan menggunakan baris berikut ini.

```

return RedirectToAction("Index");

```

Method Action [HttpGet]Edit

Berikut ini adalah isi kode method Edit sebelum dimodifikasi.

```

public async Task<IActionResult> Edit(int? id)
{
    if (id == null)
    {
        return NotFound();
    }

    var book = await _context.Books.SingleOrDefaultAsync(m => m.BookID == id);
    if (book == null)
    {
        return NotFound();
    }
    ViewData["CategoryID"] = new SelectList(_context.Categories, "CategoryID",
", "Name", book.CategoryID);
    return View(book);
}

```

Berikut ini adalah antarmuka form edit data yang akan dibuat.

The screenshot shows a web-based form titled "Book". The form is titled "Form of Book edit data". It contains the following fields:

- ISBN: 1233
- Category: Sejarah
- Title: Sejarah Jawa
- List of Author Names: Reza, Faisal
- Publish Date: 12/12/2012
- Price: 12
- Quantity: 12
- Photo: Pilih File (No file selected)

At the bottom of the form are two buttons: "Cancel" and "Update".

Gambar 129. BooksController - Edit.

Untuk keperluan itu maka method action Edit akan dimodifikasi seperti berikut ini.

```
[HttpGet]
public IActionResult Edit(int? id)
{
    ViewBag.Categories = new SelectList(_context.Categories.ToList(), "CategoryID", "Name");
    ViewBag.Authors = new MultiSelectList(_context.Authors.ToList(), "AuthorID", "Name");

    var book = _context.Books.SingleOrDefault(p => p.BookID.Equals(id));

    BookFormViewModel item = new BookFormViewModel();
    item.ISBN = book.BookID;
    item.Title = book.Title;
    item.PublishDate = book.PublishDate;
    item.Price = book.Price;
    item.Quantity = book.Quantity;
    item.CategoryID = book.CategoryID;

    var authorList = _context.BooksAuthors.Where(p => p.BookID.Equals(book.BookID)).ToList();
    List<int> authors = new List<int>();
    foreach (BookAuthor bookAuthor in authorList)
    {
        authors.Add(bookAuthor.AuthorID);
    }
    item.AuthorIDs = authors.ToArray();

    return View(item);
}
```

Method action ini bertujuan untuk menampilkan form untuk mengedit data buku yang dipilih. Untuk mendapatkan data buku yang akan diedit maka pada method ini memiliki parameter input untuk menerima kiriman id pengarang buku dari komponen view Index.cshtml.

```
public IActionResult Edit(int? id)
{
    . . .
```

```
}
```

Method action ini akan menampilkan komponen view Edit.cshtml. Pada form ini juga terdapat kode untuk menampung data dari tabel categories dan authors untuk ditampilkan dengan cara berikut ini.

```
ViewBag.Categories = new SelectList(_context.Categories.ToList(),
"CategoryID", "Name");

ViewBag.Authors = new MultiSelectList(_context.Authors.ToList(), "AuthorID",
"Name");
```

Kemudian dengan memanfaatkan parameter input id tersebut maka dapat dilakukan pengambilan data dengan menggunakan cara berikut ini.

```
var book = _context.Books.SingleOrDefault(p => p.BookID.Equals(id));
```

Karena model yang digunakan pada halaman Edit.cshtml adalah BookFormViewModel, maka nilai-nilai property dari object book harus disalin ke property-property pada object item yang merupakan instansiasi class BookFormViewModel. Berikut adalah kode yang digunakan.

```
BookFormViewModel item = new BookFormViewModel();
item.ISBN = book.BookID;
item.Title = book.Title;
item.PublishDate = book.PublishDate;
item.Price = book.Price;
item.Quantity = book.Quantity;
item.CategoryID = book.CategoryID;
```

Selanjutnya untuk nilai property AuthorIDs diisi dengan menggunakan kode berikut ini.

```
var authorList = _context.BooksAuthors.Where(p =>
p.BookID.Equals(book.BookID)).ToList();

List<int> authors = new List<int>();
foreach (BookAuthor bookAuthor in authorList)
{
    authors.Add(bookAuthor.AuthorID);
}

item.AuthorIDs = authors.ToArray();
```

Data buku yang dipilih ditampung oleh object item, kemudian object tersebut dikirim ke komponen view Edit.cshtml dengan cara seperti berikut.

```
return View(item);
```

Method Action [HttpPost]Edit

Berikut ini adalah kode method Edit sebelum dimodifikasi.

```
[HttpPost]
[ValidateAntiForgeryToken]
public async Task<IActionResult> Edit(int id,
[Bind("BookID,CategoryID,Title,Photo,PublishDate,Price,Quantity")] Book
book)
{
    if (id != book.BookID)
    {
        return NotFound();
    }
}
```

```

    if (ModelState.IsValid)
    {
        try
        {
            _context.Update(book);
            await _context.SaveChangesAsync();
        }
        catch (DbUpdateConcurrencyException)
        {
            if (!BookExists(book.BookID))
            {
                return NotFound();
            }
            else
            {
                throw;
            }
        }
        return RedirectToAction(nameof(Index));
    }
    ViewData["CategoryID"] = new SelectList(_context.Categories,
    "CategoryID", "Name", book.CategoryID);
    return View(book);
}

```

Dan berikut adalah kode method ini setelah dimodifikasi.

```

[HttpPost]
[ValidateAntiForgeryToken]
public IActionResult Edit([Bind("ISBN, CategoryID, Title, Photo,
PublishDate, Price, Quantity, AuthorIDs")] BookFormViewModel item)
{
    if (ModelState.IsValid)
    {
        _context.BooksAuthors.RemoveRange(_context.BooksAuthors.Where(p =>
p.BookID.Equals(item.ISBN)));
        _context.SaveChanges();

        Book book = _context.Books.SingleOrDefault(p =>
p.BookID.Equals(item.ISBN));
        book.CategoryID = item.CategoryID;
        book.Title = item.Title;
        book.PublishDate = item.PublishDate;
        book.Price = item.Price;
        book.Quantity = item.Quantity;
        _context.Update(book);

        foreach (int authorId in item.AuthorIDs)
        {
            BookAuthor bookAuthor = new BookAuthor();
            bookAuthor.BookID = item.ISBN;
            bookAuthor.AuthorID = authorId;
            _context.Add(bookAuthor);
        }

        _context.SaveChanges();

        if (item.Photo != null)
        {
            var file = item.Photo;
            var uploads = Path.Combine(_environment.WebRootPath, "upload");
            if (file.Length > 0)

```

```

        {
            using (var fileStream = new FileStream(Path.Combine(uploads,
item.ISBN + ".jpg"), FileMode.Create))
            {
                file.CopyToAsync(fileStream);
            }
        }

        return RedirectToAction("Index");
    }

    return View();
}

```

Method action ini bertujuan untuk menyimpan data buku telah diubah nilai-nilainya melalui form pada file Edit.cshtml. Agar method ini dapat menerima kiriman object file Edit.cshtml maka dapat dilihat parameter input item seperti pada kode berikut ini.

```

public IActionResult Edit([Bind("ISBN, CategoryID, Title, Photo,
PublishDate, Price, Quantity, AuthorIDs")] BookFormViewModel item)
{
    . . .
}

```

Pada parameter input di atas dapat dilihat object item tipe BookFormViewModel. Berbeda jika dibandingkan dengan parameter input pada method action [HttpPost] Create, method action ini memiliki tambahan atribut Bind yang berisi nama property-property dari class entity model. Tujuan penggunaan atribut Bind pada object item adalah untuk mengisi nilai-nilai property ke object item tersebut.

Selanjutnya adalah memvalidasi object sebelum disimpan dengan kode berikut.

```

if (ModelState.IsValid)
{
    . . .
}

```

Langkah selanjutnya adalah menghapus record pada tabel books_authors yang memiliki nilai ISBN yang sama dengan buku yang dipilih untuk diedit ini. berikut adalah kode yang digunakan.

```

_context.BooksAuthors.RemoveRange(_context.BooksAuthors.Where(p =>
p.BookID.Equals(item.ISBN)));

_context.SaveChanges();

```

Langkah selanjutnya adalah memilih object buku yang diedit untuk ditampung pada object entity model.

```

Book book = _context.Books.SingleOrDefault(p => p.BookID.Equals(item.ISBN));

```

Selanjutnya adalah mengubah nilai-nilai property object book dan menyimpan ke database dengan cara berikut ini.

```

book.CategoryID = item.CategoryID;
book.Title = item.Title;
book.PublishDate = item.PublishDate;
book.Price = item.Price;
book.Quantity = item.Quantity;
_context.Update(book);

```

Kemudian menyimpan data pengarang buku yang dipilih dengan cara berikut ini.

```

foreach (int authorId in item.AuthorIDs)
{
    BookAuthor bookAuthor = new BookAuthor();
    bookAuthor.BookID = item.ISBN;
    bookAuthor.AuthorID = authorId;
    _context.Add(bookAuthor);
}

_context.SaveChanges();

```

Kemudian dilakukan pemeriksaan jika ada file cover buku yang diupload. Jika ada file yang diupload maka akan dilakukan proses penyimpanan file tersebut pada folder wwwroot/upload.

```

if (item.Photo != null)
{
    var file = item.Photo;
    var uploads = Path.Combine(_environment.WebRootPath, "upload");
    if (file.Length > 0)
    {
        using (var fileStream = new FileStream(Path.Combine(uploads,
item.ISBN + ".jpg"), FileMode.Create))
        {
            file.CopyToAsync(fileStream);
        }
    }
}

```

Setelah data sukses disimpan, selanjutnya akan kembali ditampilkan daftar buku dengan menggunakan baris berikut ini.

```

return RedirectToAction("Index");

```

Method Action [HttpGet]Delete

Berikut ini adalah kode method action Delete.

```

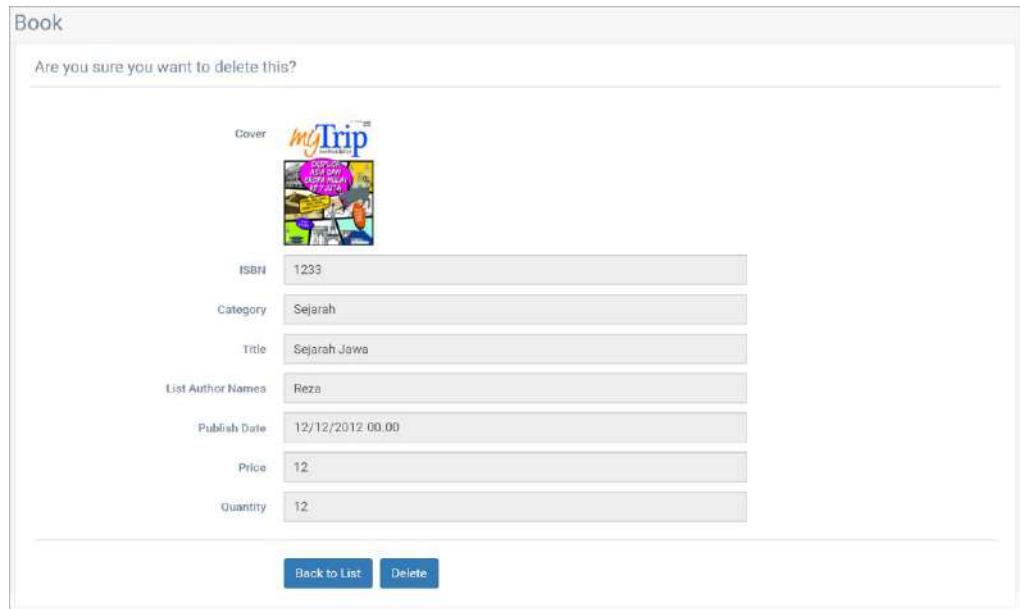
// GET: Books/Delete/5
public async Task<IActionResult> Delete(int? id)
{
    if (id == null)
    {
        return NotFound();
    }

    var book = await _context.Books
        .Include(b => b.Category)
        .SingleOrDefaultAsync(m => m.BookID == id);
    if (book == null)
    {
        return NotFound();
    }

    return View(book);
}

```

Method ini bertujuan untuk menampilkan halaman konfirmasi data yang akan dihapus. Antarmuka halaman konfirmasi yang akan dibuat adalah sebagai berikut.



Gambar 130. BooksController - Delete.

Untuk menyesuaikan dengan antarmuka itu, maka method ini akan dimodifikasi seperti berikut.

```
public IActionResult Delete(int? id)
{
    var book = _context.Books
        .Include(c => c.Category)
        .Include(ba => ba.BooksAuthors)
        .ThenInclude(a => a.Author).SingleOrDefault(b => b.BookID.Equals(id));

    BookViewModel item = new BookViewModel();
    item.ISBN = book.BookID;
    item.Title = book.Title;
    item.PublishDate = book.PublishDate;
    item.Price = book.Price;
    item.Quantity = book.Quantity;
    item.CategoryName = book.Category.Name;

    string authorNameList = string.Empty;
    var booksAuthorsList = book.BooksAuthors;
    foreach (BookAuthor booksAuthors in booksAuthorsList)
    {
        var author = booksAuthors.Author;
        authorNameList = authorNameList + author.Name + ", ";
    }
    item.AuthorNames = authorNameList.Substring(0, authorNameList.Length - 2);
}

return View(item);
}
```

Isi method action ini sama dengan isi method action Detail.

Method Action [HttpPost]Delete

Berikut ini adalah method action Delete sebelum dimodifikasi.

```
// POST: Books/Delete/5
[HttpPost, ActionName("Delete")]
```

```

[ValidateAntiForgeryToken]
public async Task<IActionResult> DeleteConfirmed(int id)
{
    var book = await _context.Books.SingleOrDefaultAsync(m => m.BookID == id);
    _context.Books.Remove(book);
    await _context.SaveChangesAsync();
    return RedirectToAction(nameof(Index));
}

```

Dan kode berikut ini adalah isi method action Delete setelah dimodifikasi.

```

[HttpPost, ActionName("Delete")]
[ValidateAntiForgeryToken]
public IActionResult DeleteConfirmed(int id)
{
    if (ModelState.IsValid)
    {
        _context.BooksAuthors.RemoveRange(_context.BooksAuthors.Where(p => p.BookID.Equals(id)));
        _context.SaveChanges();

        var book = _context.Books.SingleOrDefault(m => m.BookID == id);
        _context.Books.Remove(book);
        _context.SaveChanges();

        return RedirectToAction("Index");
    }

    return View();
}

```

Method ini berfungsi untuk menghapus data buku yang dipilih. Agar method ini dapat menerima id buku untuk dihapus maka diperlukan parameter input seperti berikut.

```

public IActionResult Delete(int id)
{
    . . .
}

```

Kemudian dilakukan validasi sebelum melakukan proses penghapusan data.

```

if(ModelState.IsValid)
{
    . . .
}

```

Data yang akan dihapus pertama kali adalah data pada BooksAuthors, yang menyimpan relasi antara data buku dengan pengarang. Berikut adalah kode yang digunakan untuk menghapus data berdasarkan id dari buku.

```

_context.BooksAuthors.RemoveRange(_context.BooksAuthors.Where(p => p.BookID.Equals(id)));

_context.SaveChanges();

```

Selanjutnya adalah mencari objek buku yang akan dihapus dengan kode berikut ini.

```

var book = _context.Books.SingleOrDefault(m => m.BookID == id);

```

Objek book akan berisi data buku yang akan dihapus. Selanjutnya untuk menghapus buku dari tabel Books digunakan kode berikut ini.

```

_context.Books.Remove(book);

```

```
_context.SaveChanges();
```

Setelah data sukses dihapus, selanjutnya akan kembali ditampilkan daftar buku dengan menggunakan baris berikut ini.

```
return RedirectToAction("Index");
```

Class Controller Role

Fitur pengelolaan role dapat digunakan untuk:

- Menampilkan daftar role.
- Menampilkan detail role.
- Menambah data role.

Untuk menambah data role terlebih dahulu perlu ditampilkan form input role. Kemudian pengguna akan mengisi form input tersebut dan setelah tombol submit diklik akan dilakukan proses menyimpanan data yang dimasukkan ke dalam database.

- Mengedit data role yang dipilih.
Untuk mengedit data role terlebih dahulu perlu ditampilkan form input edit role. Kemudian pengguna akan mengedit data dari form edit. Setelah tombol submit diklik maka dilakukan proses update data ke dalam database.
- Menghapus data role yang dipilih.

Pembuatan class controller Role tidak dilakukan dengan memanfaatkan fitur Visual Studio, seperti yang telah dilakukan pada class controller Authors dan Categories. Hal ini disebabkan pengelolaan data Role tidak menggunakan Entity Framework tetapi memanfaatkan ASP.NET Core Identity.

Berikut ini adalah kode lengkap dari class RoleController.cs.

```
RoleController.cs
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.Identity;
using SqlServerBookStore.Models;
using SqlServerBookStore.Data;

namespace SqlServerBookStore.Controllers
{
    public class RoleController : Controller
    {
        private readonly RoleManager<ApplicationRole> db;

        public RoleController(RoleManager<ApplicationRole> roleManager)
        {
            this.db = roleManager;
        }

        [HttpGet]
        public IActionResult Index()
        {

            var items = new List<RoleViewModel>();
```

```

        items = db.Roles.Select(r => new RoleViewModel
    {
        RoleID = r.Id,
        RoleName = r.Name,
        Description = r.Description
    }).ToList();

    return View(items);
}

[HttpGet]
public async Task<IActionResult> Detail(string id)
{
    RoleViewModel item = new RoleViewModel();
    ApplicationRole role = await db.FindByIdAsync(id);
    if (role != null)
    {
        item.RoleID = role.Id;
        item.RoleName = role.Name;
        item.Description = role.Description;
    }
    return View(item);
}

[HttpGet]
public IActionResult Create()
{
    return View();
}

[HttpPost]
public async Task<IActionResult> Create(RoleViewModel item)
{
    if (ModelState.IsValid)
    {
        ApplicationRole role = new ApplicationRole();
        role.Id = item.RoleID;
        role.Name = item.RoleName;
        role.Description = item.Description;

        var result = await db.CreateAsync(role);

        return RedirectToAction("Index");
    }

    return View();
}

[HttpGet]
public async Task<IActionResult> Edit(string id)
{
    RoleViewModel item = new RoleViewModel();
    ApplicationRole role = await db.FindByIdAsync(id);
    if (role != null)
    {
        item.RoleID = role.Id;
        item.RoleName = role.Name;
        item.Description = role.Description;
    }
    return View(item);
}

```

```

[HttpPost]
[ValidateAntiForgeryToken]
public async Task<IActionResult> Edit([Bind("RoleID,RoleName,Description")]
RoleViewModel item)
{
    if (ModelState.IsValid)
    {
        ApplicationRole role = await db.FindByIdAsync(item.RoleID);
        if (role != null)
        {
            role.Id = item.RoleID;
            role.Name = item.RoleName;
            role.Description = item.Description;
            var result = await db.UpdateAsync(role);
        }
        return RedirectToAction("Index");
    }

    return View();
}

[HttpGet]
public async Task<IActionResult> Delete(string id)
{
    RoleViewModel item = new RoleViewModel();
    ApplicationRole role = await db.FindByIdAsync(id);
    if (role != null)
    {
        item.RoleID = role.Id;
        item.RoleName = role.Name;
        item.Description = role.Description;
    }
    return View(item);
}

[HttpPost, ActionName("Delete")]
[ValidateAntiForgeryToken]
public async Task<IActionResult> DeleteConfirmed(string id)
{
    if (ModelState.IsValid)
    {
        ApplicationRole role = await db.FindByIdAsync(id);
        var result = await db.DeleteAsync(role);

        return RedirectToAction("Index");
    }

    return View();
}
}

```

Method Action [HttpGet] Index

Method action ini berfungsi untuk menampilkan daftar role. Data role yang disimpan di database akan ditampung di dalam object items yang merupakan instansiasi dari class RoleViewModel.

```
var items = new List<RoleViewModel>();
```

Selanjutnya untuk mengambil data dari database dapat dilakukan dengan menggunakan kode berikut ini.

```
var items = new List<RoleViewModel>();
items = db.Roles.Select(r => new RoleViewModel
{
    RoleID = r.Id,
    RoleName = r.Name,
    Description = r.Description
}).ToList();
```

Cara di atas ini sedikit berbeda dengan cara pengambilan data yang telah dilakukan pada class controller CategoriesController, AuthorsController atau BooksController. Pada cara di atas, setelah data role diambil dengan cara seperti pada baris pertama, selanjutnya memilih property-property yang dimiliki data role untuk disimpan ke property-property object items.

Dan selanjutnya object items dikirim ke komponen view untuk ditampilkan.

```
return View(items);
```

Method Action [HttpGet] Detail

Method action ini berfungsi untuk menampilkan detail data role yang dipilih.

```
RoleViewModel item = new RoleViewModel();
ApplicationRole role = await db.FindByIdAsync(id);
if (role != null)
{
    item.RoleID = role.Id;
    item.RoleName = role.Name;
    item.Description = role.Description;
}
```

Untuk mendapatkan objek role yang dipilih dapat dilihat pada baris yang dicetak tebal.

Method Action [HttpGet] Create

Method action ini bertujuan untuk menampilkan form untuk menambah data role. File komponen view yang digunakan sebagai form ini adalah Create.cshtml.

Method Action [HttpPost] Create

Method action ini berfungsi untuk menyimpan data yang dikirimkan dari komponen view Create.cshtml. Untuk menyimpan data role ke database digunakan object dari class ApplicationRole dan berikut adalah cara untuk mengisi object tersebut dengan nilai dari komponen view.

```
ApplicationRole role = new ApplicationRole();
role.Id = item.RoleID;
role.Name = item.RoleName;
role.Description = item.Description;
```

Selanjutnya untuk menyimpan object tersebut ke dalam database digunakan perintah berikut.

```
var result = await db.CreateAsync(role);
```

Method Action [HttpGet] Edit

Method action ini bertujuan untuk menampilkan form untuk mengedit data role yang dipilih. Untuk mendapatkan data role yang akan diedit maka pada method ini memiliki parameter input untuk menerima kiriman id pengarang buku dari komponen view Index.cshtml.

```
public async Task<IActionResult> Edit(string id)
{
    . . .
}
```

Dengan memanfaatkan parameter input id tersebut maka dapat dilakukan pengambilan data dengan menggunakan cara berikut ini.

```
ApplicationRole role = await db.FindByIdAsync(id);
```

Kemudian data yang ada pada object role akan disimpan kembali ke dalam object item yang merupakan instansiasi class RoleViewModel.

```
RoleViewModel item = new RoleViewModel();

if (role != null)
{
    item.RoleID = role.Id;
    item.RoleName = role.Name;
    item.Description = role.Description;
}
```

Data role yang dipilih ditampung oleh object item, kemudian object tersebut dikirim ke komponen view Edit.cshtml dengan cara seperti berikut.

```
return View(item);
```

Method Action [HttpPost] Edit

Method action ini berfungsi untuk menyimpan data role yang dikirimkan dari komponen view Edit.cshtml. Agar method ini dapat menerima kiriman object file Edit.cshtml maka dapat dilihat parameter input item seperti pada kode berikut ini.

```
public async Task<IActionResult> Edit([Bind("RoleID,RoleName,Description")]
RoleViewModel item)
{
    . . .
}
```

Selanjutnya data yang akan diedit mesti dicari terlebih dahulu di dalam database dan ditampung ke dalam object role berikut ini.

```
ApplicationRole role = await db.FindByIdAsync(item.RoleID);
```

Jika data yang dicari ditemukan atau ketika object role memiliki nilai maka selanjutnya dilakukan penyimpanan data dengan cara seperti berikut ini.

```
if (role != null)
{
    role.Id = item.RoleID;
    role.Name = item.RoleName;
    role.Description = item.Description;
    var result = await db.UpdateAsync(role);
}
```

Selanjutnya jika proses penyimpanan berhasil maka akan ditampilkan komponen view Index.html.

Method Action [HttpGet] Delete

Method action ini untuk menampilkan detail data yang akan dihapus. Cara menampilkannya seperti yang telah dilakukan oleh method action Detail.

```
RoleViewModel item = new RoleViewModel();
ApplicationRole role = await db.FindByIdAsync(id);
if (role != null)
{
    item.RoleID = role.Id;
    item.RoleName = role.Name;
    item.Description = role.Description;
}
```

Method Action [HttpPost] Delete

Method ini digunakan untuk menghapus objek role yang telah dipilih.

```
[HttpPost, ActionName("Delete")]
[ValidateAntiForgeryToken]
public async Task<IActionResult> DeleteConfirmed(string id)
{
    .
    .
}
```

Nama method di atas adalah DeleteConfirmed, tetapi dengan memberikan atribut ActionName("Delete") maka untuk memanggil method ini dengan cara nama Delete. Selanjutnya untuk menghapus objek role yang dipilih digunakan kode di bawah ini.

```
ApplicationRole role = await db.FindByIdAsync(id);
var result = await db.DeleteAsync(role);
```

Class Controller User

Fitur pengelolaan user dapat digunakan untuk:

- Menampilkan daftar user.
- Menampilkan detail user yang dipilih.
- Menambah data user.

Untuk menambah data user terlebih dahulu perlu ditampilkan form input user. Kemudian pengguna akan mengisi form input tersebut dan setelah tombol submit diklik akan dilakukan proses menyimpanan data yang dimasukkan ke dalam database.

- Mengedit data user yang dipilih.
Untuk mengedit data user terlebih dahulu perlu ditampilkan form input edit user. Kemudian pengguna akan mengedit data dari form edit. Setelah tombol submit diklik maka dilakukan proses update data ke dalam database.
- Menghapus data user yang dipilih.

Pembuatan class controller User tidak dilakukan dengan memanfaatkan fitur Visual Studio, seperti yang telah dilakukan pada class controller Authors dan Categories. Hal ini disebabkan pengelolaan data User tidak menggunakan Entity Framework tetapi memanfaatkan ASP.NET Core Identity.

Berikut ini adalah kode lengkap dari class UserController.cs.

```
UserController.cs
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.Mvc.Rendering;
using Microsoft.AspNetCore.Identity;
using SqlServerBookStore.Models;
using SqlServerBookStore.Data;

namespace SqlServerBookStore.Controllers
{
    public class UserController : Controller
    {
        private readonly UserManager< ApplicationUser > userManager;
        private readonly RoleManager< ApplicationRole > roleManager;

        public UserController(UserManager< ApplicationUser > userManager, Role
Manager< ApplicationRole > roleManager)
        {
            this.userManager = userManager;
            this.roleManager = roleManager;
        }

        [HttpGet]
        public async Task< IActionResult > Index()
        {

            var items = new List< UserViewModel >();
            var userList = userManager.Users.ToList();
            foreach (var user in userList)
            {
                var item = new UserViewModel();
                item.UserName = user.UserName;
                item.Email = user.Email;
                item.FullName = user.FullName;

                string roleNameList = string.Empty;
                var roleList = await userManager.GetRolesAsync(user);
                foreach (var role in roleList)
                {
                    roleNameList = roleNameList + role + ", ";
                }
                item.RoleName = roleNameList.Substring(0, roleNameList.Length - 2);

                items.Add(item);
            }
            return View(items);
        }

        [HttpGet]
        public async Task< IActionResult > Detail(string id)
        {
            var user = await userManager.FindByNameAsync(id);

            UserViewModel item = new UserViewModel();
            item.UserName = user.UserName;
```

```

        item.Email = user.Email;
        item.FullName = user.FullName;

        string roleNameList = string.Empty;
        var roleList = await userManager.GetRolesAsync(user);
        foreach (var role in roleList)
        {
            roleNameList = roleNameList + role + ", ";
        }
        item.RoleName = roleNameList.Substring(0, roleNameList.Length - 2);

        return View(item);
    }

    [HttpGet]
    public IActionResult Create()
    {
        ViewBag.Roles = new MultiSelectList(roleManager.Roles.ToList(),
"Id", "Name");
        return View();
    }

    [HttpPost]
    public async Task<IActionResult> Create(UserCreateFormViewModel item)
    {
        ViewBag.Roles = new MultiSelectList(roleManager.Roles.ToList(),
"Id", "Name");

        if (ModelState.IsValid)
        {
            ApplicationUser user = new ApplicationUser();
            user.FullName = item.FullName;
            user.UserName = item.UserName;
            user.Email = item.Email;
            IdentityResult result = await userManager.CreateAsync(user,
item.Password);

            if (result.Succeeded)
            {
                result = await userManager.AddToRolesAsync(user, item.Ro
leID);

                if (result.Succeeded)
                {
                    return RedirectToAction("Index");
                }
            }
        }

        return View();
    }

    [HttpGet]
    public async Task<IActionResult> Edit(string id)
    {
        ViewBag.Roles = new MultiSelectList(roleManager.Roles.ToList(),
"Id", "Name");
        var user = await userManager.FindByNameAsync(id);
    }
}

```

```

        UserEditFormViewModel item = new UserEditFormViewModel();
        item.UserName = user.UserName;
        item.Email = user.Email;
        item.FullName = user.FullName;

        var roles = await userManager.GetRolesAsync(user);
        item.RoleID = roles.ToArray();

        return View(item);
    }

    [HttpPost]
    [ValidateAntiForgeryToken]
    public async Task<IActionResult> Edit([Bind("UserName, RoleID, Email
, Password, PasswordConfirm, FullName")] UserEditFormViewModel item)
    {
        ViewBag.Roles = new MultiSelectList(roleManager.Roles.ToList(),
"Id", "Name");

        if (ModelState.IsValid)
        {
            ApplicationUser user = await userManager.FindByNameAsync(item.UserName);
            user.Email = item.Email;
            user.FullName = item.FullName;
            var existingRoles = await userManager.GetRolesAsync(user);
            IdentityResult result = await userManager.UpdateAsync(user);

            if (result.Succeeded)
            {
                result = await userManager.RemoveFromRolesAsync(user, existingRoles);

                if (result.Succeeded)
                {
                    result = await userManager.AddToRolesAsync(user, item.RoleID);
                    if (result.Succeeded)
                    {
                        return RedirectToAction("Index");
                    }
                }
            }
        }
        return View();
    }

    [HttpGet]
    public async Task<IActionResult> Delete(string id)
    {
        var user = await userManager.FindByNameAsync(id);

        UserViewModel item = new UserViewModel();
        item.UserName = user.UserName;
        item.Email = user.Email;
        item.FullName = user.FullName;

        string roleNameList = string.Empty;
        var roleList = await userManager.GetRolesAsync(user);
        foreach (var role in roleList)

```

```

        {
            roleNameList = roleNameList + role + ", ";
        }
        item.RoleName = roleNameList.Substring(0, roleNameList.Length - 2);

        return View(item);
    }

    [HttpPost, ActionName("Delete")]
    [ValidateAntiForgeryToken]
    public async Task<IActionResult> DeleteConfirmed(string id)
    {
        if (ModelState.IsValid)
        {
            var user = await userManager.FindByNameAsync(id);
            var result = await userManager.DeleteAsync(user);
            return RedirectToAction("Index");
        }

        return View();
    }
}

```

Pada kode di atas dapat dilihat penambahan dua property pada class UserController yaitu:

1. userManager, adalah object hasil instansiasi class UserManager yang berfungsi untuk mengakses data user.
2. roleManager, adalah object hasil instansiasi class RoleManager yang berfungsi untuk mengakses data role.

Class UserManager dan RoleManager adalah bagian dari ASP.NET Core Identity.

Keduanya akan mempermudah untuk mengelola data user dan role. Selain itu pada class ini juga ditambahkan constructor UserController untuk melakukan inisialisasi nilai pada property userManager dan roleManager.

Method Action [HttpGet] Index

Method action ini bertujuan untuk mengirimkan data user untuk ditampilkan pada halaman view Index.cshtml. Data user akan berisi informasi user dan juga nama role yang dimiliki oleh user tersebut. Untuk menampung data tersebut akan digunakan class UserViewModel.

```
var items = new List<UserViewModel>();
```

Selanjutnya untuk mengambil data user dari database digunakan kode berikut ini.

```
var userList = userManager.Users.ToList();
```

Selanjutnya adalah melakukan pengulangan untuk memasukan data dari userList ke dalam object item yang merupakan instansiasi dari class UserViewModel.

```

foreach (var user in userList)
{
    var item = new UserViewModel();
    item.UserName = user.UserName;
    item.Email = user.Email;
    item.FullName = user.FullName;

    string roleNameList = string.Empty;

```

```

var roleList = await userManager.GetRolesAsync(user);
foreach (var role in roleList)
{
    roleNameList = roleNameList + role + ", ";
}
item.RoleName = roleNameList.Substring(0, roleNameList.Length - 2);

items.Add(item);
}

```

Pada pengulangan di atas juga dapat dilihat bagaimana cara mengambil daftar role dari user yang kemudian dimasukkan sebagai nilai property RoleName. Setiap user dapat memiliki sebuah role atau lebih dari satu.

Setelah itu data yang telah ditampung pada object items ditampilkan ke halaman view Index.cshtml.

```

return View(items);

```

Method Action [HttpGet] Detail

Method ini berfungsi untuk menampilkan detail data user yang dipilih. Berikut ini adalah kode untuk mengambil objek data user berdasarkan username.

```

var user = await userManager.FindByNameAsync(id);

```

Setelah objek user berisi data yang dipilih, selanjutnya digunakan kode berikut untuk mengisi objek UserViewModel.

```

UserViewModel item = new UserViewModel();
item.UserName = user.UserName;
item.Email = user.Email;
item.FullName = user.FullName;

string roleNameList = string.Empty;
var roleList = await userManager.GetRolesAsync(user);
foreach (var role in roleList)
{
    roleNameList = roleNameList + role + ", ";
}
item.RoleName = roleNameList.Substring(0, roleNameList.Length - 2);

```

Kemudian data akan dikirimkan ke komponen view.

```

return View(item);

```

Method Action [HttpGet] Create

Method action ini berfungsi untuk menampilkan halaman Create.cshtml yang berfungsi sebagai form untuk input data user. Karena pada form input data user diperlukan daftar role yang dapat dipilih maka perlu dipersiapkan data role dengan cara berikut ini.

```

ViewBag.Roles = new MultiSelectList(roleManager.Roles.ToList(), "Id",
"Name");

```

Method Action [HttpPost] Create

Method action ini berfungsi untuk menyimpan data user yang telah diisikan pada form input pada halaman Create.cshtml. Data user dari halaman Create.cshtml dikirimkan via object item yang merupakan instan dari class UserCreateFormViewModel. Selanjutnya nilai-nilai pada object item ini akan dimasukkan pada object user yang merupakan instansiasi dari class model ApplicationUser. Berikut adalah kode yang digunakan.

```
 ApplicationUser user = new ApplicationUser();
user.FullName = item.FullName;
user.UserName = item.UserName;
user.Email = item.Email;

IdentityResult result = await userManager.CreateAsync(user, item.Password);
```

Kemudian pada baris terakhir pada kode di atas adalah cara untuk menyimpan data user ke dalam database. Method CreateAsync berfungsi untuk membuat user dengan parameter input nama user dan password.

Jika proses pembuatan user berhasil maka dilanjutkan dengan penambahan role untuk user tersebut dengan cara berikut ini.

```
result = await userManager.AddToRolesAsync(user, item.RoleID);
```

Method Action [HttpGet] Edit

Method action ini berfungsi untuk menampilkan form edit data user yang telah dipilih pada halaman Index.cshtml. Untuk mengambil nilai id user yang dipilih dapat dilakukan dengan menjadikannya sebagai parameter input pada method action Edit seperti pada kode di bawah ini.

```
public async Task<IActionResult> Edit(string id)
{
    . . .
}
```

Untuk mengisi form edit dengan data user yang dipilih maka terlebih dahulu perlu dilakukan pengambilan data user yang dipilih dengan cara berikut ini.

```
var user = await userManager.FindByNameAsync(id);
```

Selanjutnya memindahkan nilai property-property pada object user di atas ke object item yang merupakan instansiasi dari class view model UserEditFormViewModel.

```
UserEditFormViewModel item = new UserEditFormViewModel();
item.UserName = user.UserName;
item.Email = user.Email;
item.FullName = user.FullName;
```

Sedangkan untuk mengambil nilai role dari user tersebut digunakan cara sebagai berikut.

```
var roles = await userManager.GetRolesAsync(user);
item.RoleID = roles.ToArray();
```

Dan pada baris kedua dapat dilihat data role disimpan ke dalam property RoleID.

Method Action [HttpPost] Edit

Method action ini berfungsi untuk menyimpan data user yang dikirimkan dari form edit Edit.cshtml.

```

public async Task<IActionResult> Edit([Bind("UserName, RoleID, Email,
Password, PasswordConfirm, FullName")] UserEditFormViewModel item)
{
    . . .
}

```

Pada method action di atas dapat dilihat terdapat parameter input item yang object yang dibentuk dari class UserEditFormViewModel. Dengan method Bind yang terlihat pada kode di atas maka object item ini telah diisi dengan nilai-nilai yang dimasukkan pada form Edit.cshtml.

Selanjutnya data user yang akan diedit diambil dari database dengan cara berikut ini.

```
 ApplicationUser user = await userManager.FindByNameAsync(item.UserName);
```

Kemudian mengisi nilai-nilai baru ke dalam property-property pada object user dan kemudian menyimpan ke dalam database.

```

user.Email = item.Email;
user.FullName = item.FullName;

IdentityResult result = await userManager.UpdateAsync(user);

```

Selanjutnya adalah memperbarui data role dari user tersebut dengan cara menghapus terlebih dahulu data role yang lama dari user tersebut. Kemudian memasukkan data role baru pada user tersebut.

```

var existingRoles = await userManager.GetRolesAsync(user);

if (result.Succeeded)
{
    result = await userManager.RemoveFromRolesAsync(user, existingRoles);

    if (result.Succeeded)
    {
        result = await userManager.AddToRolesAsync(user, item.RoleID);
        if (result.Succeeded)
        {
            return RedirectToAction("Index");
        }
    }
}

```

Method Action [HttpGet] Delete

Method action ini berfungsi untuk menampilkan halaman konfirmasi data yang akan dihapus. Cara untuk menampilkan data ini seperti yang telah dilakukan pada method action Detail.

```

var user = await userManager.FindByNameAsync(id);

UserViewModel item = new UserViewModel();
item.UserName = user.UserName;
item.Email = user.Email;
item.FullName = user.FullName;

string roleNameList = string.Empty;
var roleList = await userManager.GetRolesAsync(user);
foreach (var role in roleList)
{
    roleNameList = roleNameList + role + ", ";
}

```

```
    }
    item.RoleName = roleNameList.Substring(0, roleNameList.Length - 2);

    return View(item);
```

Method Action [HttpPost] Delete

Method action ini bertujuan untuk menghapus data user yang dipilih dari halaman Index.cshtml. Untuk mendapatkan id user yang akan dihapus maka perlu ditambahkan parameter input seperti berikut.

```
public async Task<IActionResult> Delete(string id)
{
    . . .
}
```

Selanjutnya data user yang akan dihapus diambil dari database dengan cara berikut ini.

```
var user = await userManager.FindByNameAsync(id);
```

Dan untuk menghapus data user tersebut digunakan kode berikut ini.

```
var result = await userManager.DeleteAsync(user);
```

Dan selanjutnya halaman dikembalikan ke halaman Index.cshtml dengan cara mengakses method action Index seperti berikut ini.

```
return RedirectToAction("Index");
```

View

Pada bab sebelumnya telah dijelaskan fungsi komponen view untuk menampilkan halaman web sebagai antarmuka agar user dapat berinteraksi dengan aplikasi. Pada halaman web ini dapat dimanfaatkan untuk menampilkan data dalam bentuk tabel, menampilkan form input dan lain-lain.

Akses File

Pada aplikasi web, sudah umum untuk menggunakan style CSS dan kode JavaScript untuk membantu membuat antarmuka. Biasanya kode style CSS dan JavaScript tersebut dapat disimpan pada sebuah file. Dan untuk mengakses file tersebut dari halaman web, biasanya digunakan kode seperti baris berikut ini.

```
<link href="css/bootstrap-responsive.min.css" rel="stylesheet">
<script src="js/jquery-1.9.1.min.js"></script>
```

Pada atribut href atau src dapat dilihat bagaimana cara menentukan lokasi file yang akan diakses. Pada ASP.NET Core, file-file JavaScript, CSS atau gambar dikelompokkan sebagai file static. File-file ini disimpan dalam folder wwwroot. Jika file script JavaScript disimpan di dalam folder wwwroot/js dan file CSS disimpan di dalam folder wwwroot/css, maka cara mengakses file-file tersebut dipermudah dengan cara sebagai berikut ini.

```
<link href="~/css/bootstrap-responsive.min.css" rel="stylesheet">
<script src="~/js/jquery-1.9.1.min.js"></script>
```

Razor

Komponen view adalah halaman web yang berisi kode HTML, CSS dan juga Javascript. Pada framework ASP.NET Core MVC dapat digunakan Razor. Razor adalah sintaks markup untuk melekatkan kode server side pada halaman web di komponen view. Sintaks Razor terdiri atas Razor markup, C# dan HTML. File yang menggunakan sintaks Razor harus disimpan ke dalam file .cshtml.

Layout & Antarmuka

Pada sub bab Model dapat dilihat antarmuka dari aplikasi Book Store. Layout dan antarmuka fitur pengelolaan kategori buku dan pengelolaan pengarang buku memiliki keseragaman, yaitu memiliki header yang sama, menu yang sama dan footer yang sama. Hal ini Razor mendukung master layout yang dapat membuat layout dan antarmuka komponen view seragam.

Persiapan

Setelah mengunduh source code template dari link <https://github.com/puikinsh/genteella>, maka langkah selanjutnya adalah extract file genteella-master.zip. Berikut adalah file dan folder yang template ini.

Name	Date modified	Type	Size
build	2/25/2017 5:25 PM	File folder	
documentation	2/25/2017 5:25 PM	File folder	
production	2/25/2017 5:25 PM	File folder	
src	2/25/2017 5:25 PM	File folder	
vendors	2/25/2017 5:27 PM	File folder	
.bowerrc	2/25/2017 5:25 PM	BOWERRC File	1 KB
.gitignore	2/25/2017 5:25 PM	Text Document	1 KB
bower.json	2/25/2017 5:25 PM	JSON File	3 KB
changelog.md	2/25/2017 5:25 PM	MD File	1 KB
gulpfile.js	2/25/2017 5:25 PM	JavaScript File	2 KB
LICENSE.txt	2/25/2017 5:25 PM	Text Document	2 KB
package.json	2/25/2017 5:25 PM	JSON File	1 KB
README.md	2/25/2017 5:25 PM	MD File	7 KB

Gambar 131. File dan folder template genteella.

Langkah pertama adalah menghapus folder-folder berikut ini yang ada pada folder wwwroot:

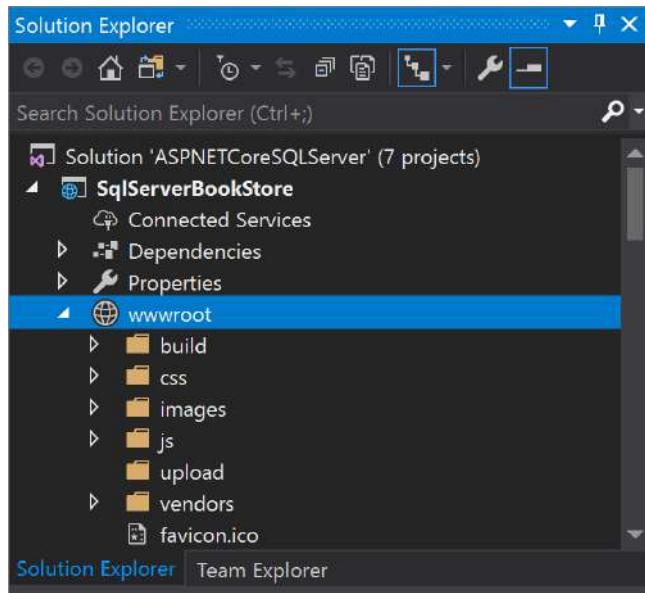
- css.
- images.
- js.
- lib.

Selanjutnya salin folder-folder berikut ke dalam folder wwwroot pada project SqlServerBookStore:

- build.
- vendors.

Selanjutnya di dalam folder production, salin folder-folder berikut ini ke dalam folder wwwroot:

- css.
- images.
- js.

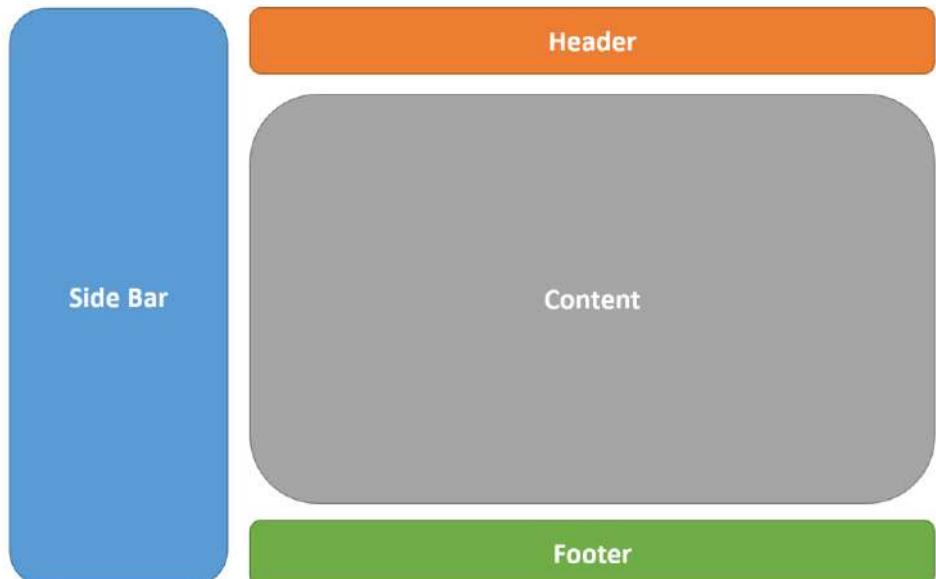


Gambar 132. SqlServerBookStore - View - wwwroot.

Hasilnya dapat dilihat pada Solution Explorer seperti gambar di atas.

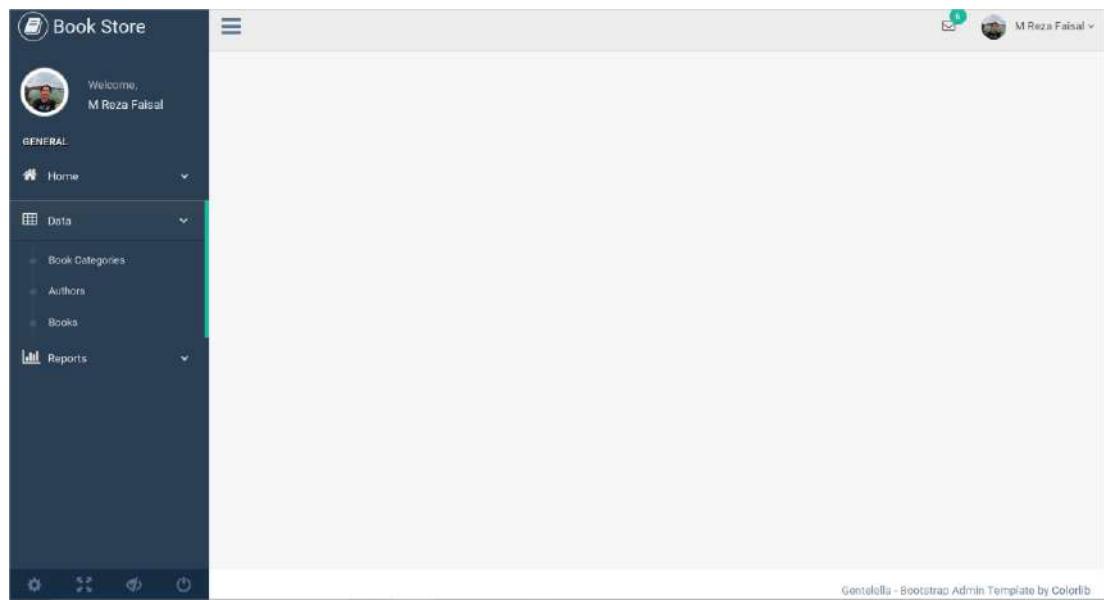
Layout

Langkah selanjutnya adalah membuat layout. Layout aplikasi Book Store memiliki bagian atau area sebagai berikut.



Gambar 133. Layout aplikasi.

Dengan menggunakan template gentelella maka dapat dibuat master layout seperti gambar berikut ini.



Gambar 134. Master layout template gentelella.

Berikut adalah langkah-langkah yang dilakukan untuk membuat layout pada aplikasi web ASP.NET Core MVC.

File layout terletak pada folder Views\Shared. File layout yang telah ada di project SqlServerBookStore adalah _Layout.cshtml. Untuk menggunakan file ini sebagai layout maka file ini akan dimodifikasi dengan kode baru.

Untuk memodifikasi _Layout.cshtml dapat dilakukan dengan menyalin isi salah satu file yang terdapat pada folder production pada source code gentelella-master. Tetapi yang disalin hanya bagian-bagian ini saja:

- Side bar.
- Header.
- Footer.

Sedangkan bagian content akan berisi kode berikut ini.

```
<!-- page content -->
<div class="right_col" role="main">
@RenderBody()
</div>
<!-- /page content -->
```

Kode @RenderBody() adalah expression Razor yang berfungsi untuk memanggil content ada komponen view yang dipanggil pada method action di class controller.

Berikut adalah isi lengkap file _Layout.cshtml yang telah dimodifikasi.

```
_Layout.cshtml
<!DOCTYPE html>
<html lang="en">
<head>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
    <!-- Meta, title, CSS, favicons, etc. -->
    <meta charset="utf-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1">
```

```

<title>ASP.NET Core MVC: Book Store</title>

    <!-- Bootstrap -->
    <link href="~/vendors/bootstrap/dist/css/bootstrap.min.css" rel="stylesheet">
    <!-- Font Awesome -->
    <link href="~/vendors/font-awesome/css/font-awesome.min.css" rel="stylesheet">
    <!-- NProgress -->
    <link href="~/vendors/nprogress/nprogress.css" rel="stylesheet">
    <!-- iCheck -->
    <link href="~/vendors/iCheck/skins/flat/green.css" rel="stylesheet">

    <!-- bootstrap-progressbar -->
    <link href="~/vendors/bootstrap-progressbar/css/bootstrap-progressbar-3.3.4.min.css" rel="stylesheet">
    <!-- JQVMap -->
    <link href="~/vendors/jqvmap/dist/jqvmap.min.css" rel="stylesheet" />
    <!-- bootstrap-daterangepicker -->
    <link href="~/vendors/bootstrap-daterangepicker/daterangepicker.css" rel="stylesheet">

    <!-- Custom Theme Style -->
    <link href="~/build/css/custom.min.css" rel="stylesheet">
</head>

<body class="nav-md">
    <div class="container body">
        <div class="main_container">
            <div class="col-md-3 left_col">
                <div class="left_col scroll-view">
                    <div class="navbar nav_title" style="border: 0;">
                        <a href="index.html" class="site_title"><i class="fa fa-book"></i> <span>Book Store</span></a>
                    </div>

                    <div class="clearfix"></div>

                    <!-- menu profile quick info -->
                    <div class="profile clearfix">
                        <div class="profile_pic">
                            
                        </div>
                        <div class="profile_info">
                            <span>Welcome,</span>
                            <h2>M Reza Faisal</h2>
                        </div>
                    </div>
                    <!-- /menu profile quick info -->

                    <br />

                    <!-- sidebar menu -->
                    <div id="sidebar-menu" class="main_menu_side hidden-print main_menu">
                        <div class="menu_section">
                            <h3>General</h3>
                            <ul class="nav side-menu">
                                <li>

```

```

        <a><i class="fa fa-
home"></i> Home <span class="fa fa-chevron-down"></span></a>
            <ul class="nav child_menu">
                <li><a asp-controller="Home" asp-
action="Index">Dashboard</a></li>
                    </ul>
                </li>
                <li>
                    <a><i class="fa fa-
table"></i> Data <span class="fa fa-chevron-down"></span></a>
                        <ul class="nav child_menu">
                            <li><a asp-
controller="Category" asp-action="Index">Book Categories</a></li>
                                <li><a asp-controller="Author" asp-
action="Index">Authors</a></li>
                                    <li><a asp-controller="Book" asp-
action="Index">Books</a></li>
                                        </ul>
                                    </li>
                                    <li>
                                        <a><i class="fa fa-
table"></i> Security <span class="fa fa-chevron-down"></span></a>
                                            <ul class="nav child_menu">
                                                <li><a asp-controller="Role" asp-
action="Index">Roles</a></li>
                                                    <li><a asp-controller="User" asp-
action="Index">Users</a></li>
                                                        </ul>
                                                    </li>
                                                    <li>
                                                        <a><i class="fa fa-bar-chart-
o"></i> Reports <span class="fa fa-chevron-down"></span></a>
                                                            <ul class="nav child_menu">
                                                                <li><a href="#">Book by Category Cha
rt</a></li>
                                                                <li><a href="#">Book Selling</a></li
>
                                                            </ul>
                                                        </li>
                                                        <li>
                                                            </ul>
                                                        </li>
                                                    </div>
                                                </div>
                                                <!-- /sidebar menu -->
                                                <!-- /menu footer buttons -->
                                                <div class="sidebar-footer hidden-small">
                                                    <a data-toggle="tooltip" data-
placement="top" title="Settings">
                                                        <span class="glyphicon glyphicon-cog" aria-
hidden="true"></span>
                                                    </a>
                                                    <a data-toggle="tooltip" data-
placement="top" title="FullScreen">
                                                        <span class="glyphicon glyphicon-
fullscreen" aria-hidden="true"></span>
                                                    </a>
                                                    <a data-toggle="tooltip" data-
placement="top" title="Lock">
                                                        <span class="glyphicon glyphicon-eye-
close" aria-hidden="true"></span>

```

```

                </a>
                <a data-toggle="tooltip" data-
placement="top" title="Logout" asp-controller="Home" asp-action="Logout">
                    <span class="glyphicon glyphicon-off" aria-
hidden="true"></span>
                </a>
            </div>
            <!-- /menu footer buttons -->
        </div>
    </div>

    <!-- top navigation -->
    <div class="top_nav">
        <div class="nav_menu">
            <nav>
                <div class="nav_toggle">
                    <a id="menu_toggle"><i class="fa fa-
bars"></i></a>
                </div>

                <ul class="nav navbar-nav navbar-right">
                    <li class="">
                        <a href="javascript:;" class="user-
profile dropdown-toggle" data-toggle="dropdown" aria-expanded="false">
                            <img src "~/images/reza.jpg" alt=""> M Re
za Faisal
                            <span class=" fa fa-angle-down"></span>
                        </a>
                        <ul class="dropdown-menu dropdown-
usermenu pull-right">
                            <li><a href="javascript:;"> Profile</a><
/li>
                            <li>
                                <a href="javascript:;">
                                    <span class="badge bg-red pull-
right">50%</span>
                                    <span>Settings</span>
                                </a>
                            </li>
                            <li><a href="javascript:;">Help</a></li>
                            <li><a asp-controller="Home" asp-
action="Logout"><i class="fa fa-sign-out pull-right"></i> Log Out</a></li>
                        </ul>
                    </li>

                    <li role="presentation" class="dropdown">
                        <a href="javascript:;" class="dropdown-
toggle info-number" data-toggle="dropdown" aria-expanded="false">
                            <i class="fa fa-envelope-o"></i>
                            <span class="badge bg-green">6</span>
                        </a>
                        <ul id="menu1" class="dropdown-menu list-
unstyled msg_list" role="menu">
                            <li>
                                <a>
                                    <span class="image"><img src "~/
images/img.jpg" alt="Profile Image" /></span>
                                    <span>
                                        <span>John Smith</span>
                                        <span class="time">3 mins ag
o</span>

```

```

        </span>
        <span class="message">
            Film festivals used to be do
-or-die moments for movie makers. They were where...
        </span>
        </a>
    </li>
    <li>
        <a>
            <span class="image"></span>
            <span>
                <span>John Smith</span>
                <span class="time">3 mins ag
o</span>
            </span>
            <span class="message">
                Film festivals used to be do
-or-die moments for movie makers. They were where...
            </span>
            </a>
        </li>
        <li>
            <a>
                <span class="image"></span>
                <span>
                    <span>John Smith</span>
                    <span class="time">3 mins ag
o</span>
                </span>
                <span class="message">
                    Film festivals used to be do
-or-die moments for movie makers. They were where...
                </span>
                </a>
            </li>
            <li>
                <a>
                    <span class="image"></span>
                    <span>
                        <span>John Smith</span>
                        <span class="time">3 mins ag
o</span>
                    </span>
                    <span class="message">
                        Film festivals used to be do
-or-die moments for movie makers. They were where...
                    </span>
                    </a>
                </li>
                <li>
                    <div class="text-center">
                        <a>
                            <strong>See All Alerts</stro
ng>
                        <i class="fa fa-angle-
right"></i>
                    </a>
                </div>

```

```

                </li>
            </ul>
        </li>
    </ul>
</nav>
</div>
<!-- /top navigation --&gt;
&lt;!--
-- --&gt;
<!-- page content --&gt;
&lt;div class="right_col" role="main"&gt;
    @RenderBody()
&lt;/div&gt;
<!-- /page content --&gt;
&lt;!--
-- --&gt;
<!-- footer content --&gt;
&lt;footer&gt;
    &lt;div class="pull-right"&gt;
        Gentelella - Bootstrap Admin Template by &lt;a href="https://colorlib.com"&gt;Colorlib&lt;/a&gt;
    &lt;/div&gt;
    &lt;div class="clearfix"&gt;&lt;/div&gt;
&lt;/footer&gt;
<!-- /footer content --&gt;
&lt;/div&gt;
&lt;/div&gt;

<!-- jQuery --&gt;
&lt;script src="~/vendors/jquery/dist/jquery.min.js"&gt;&lt;/script&gt;
<!-- Bootstrap --&gt;
&lt;script src="~/vendors/bootstrap/dist/js/bootstrap.min.js"&gt;&lt;/script&gt;
<!-- FastClick --&gt;
&lt;script src="~/vendors/fastclick/lib/fastclick.js"&gt;&lt;/script&gt;
<!-- NProgress --&gt;
&lt;script src="~/vendors/nprogress/nprogress.js"&gt;&lt;/script&gt;
<!-- Chart.js --&gt;
&lt;script src="~/vendors/Chart.js/dist/Chart.min.js"&gt;&lt;/script&gt;
<!-- gauge.js --&gt;
&lt;script src="~/vendors/gauge.js/dist/gauge.min.js"&gt;&lt;/script&gt;
<!-- bootstrap-progressbar --&gt;
&lt;script src="~/vendors/bootstrap-progressbar/bootstrap-progressbar.min.js"&gt;&lt;/script&gt;
<!-- iCheck --&gt;
&lt;script src="~/vendors/iCheck/checkbox.min.js"&gt;&lt;/script&gt;
<!-- Skycons --&gt;
&lt;script src="~/vendors/skycons/skycons.js"&gt;&lt;/script&gt;
<!-- Flot --&gt;
&lt;script src="~/vendors/Flot/jquery.flot.js"&gt;&lt;/script&gt;
&lt;script src="~/vendors/Flot/jquery.flot.pie.js"&gt;&lt;/script&gt;
&lt;script src="~/vendors/Flot/jquery.flot.time.js"&gt;&lt;/script&gt;
&lt;script src="~/vendors/Flot/jquery.flot.stack.js"&gt;&lt;/script&gt;
&lt;script src="~/vendors/Flot/jquery.flot.resize.js"&gt;&lt;/script&gt;
<!-- Flot plugins --&gt;
&lt;script src="~/vendors/flot.orderbars/js/jquery.flot.orderBars.js"&gt;&lt;/script&gt;
&lt;script src="~/vendors/flot-spline/js/jquery.flot.spline.min.js"&gt;&lt;/script&gt;
&lt;script src="~/vendors/flot.curvedlines/curvedLines.js"&gt;&lt;/script&gt;
<!-- DateJS --&gt;
</pre>

```

```

<script src="~/vendors/DateJS/build/date.js"></script>
<!-- JQVMap -->
<script src="~/vendors/jqvmap/dist/jquery.vmap.js"></script>
<script src="~/vendors/jqvmap/dist/maps/jquery.vmap.world.js"></script>
<script src="~/vendors/jqvmap/examples/js/jquery.vmap.sampledata.js"></script>
<!-- bootstrap-daterangepicker -->
<script src="~/vendors/moment/min/moment.min.js"></script>
<script src="~/vendors/bootstrap-daterangepicker/daterangepicker.js"></script>

<!-- Custom Theme Scripts -->
<script src="~/build/js/custom.min.js"></script>

</body>
</html>

```

Source code _Layout.cshtml ini juga dapat dilihat pada link GitHub yang telah disebutkan pada bab Pendahuluan sub bab Bahan Pendukung.

Content

Seperti yang telah disebutkan di atas, content akan berisi file view yang dipanggil oleh method action. Sebuah file view ini dapat berdiri sendiri tanpa layout atau dapat juga menggunakan layout.

Jika file view ingin berdiri sendiri tanpa menggunakan layout, maka di awal file tersebut dapat ditambahkan baris berikut ini.

```

@{
    Layout = null;
}

```

Jika sebuah file view ingin menggunakan master layout maka di awal file tersebut dapat menggunakan baris berikut ini.

```

@{
    Layout = "~/Views/Shared/_Layout.cshtml";
}

```

Jika project aplikasi web memiliki banyak file view, misal ada dimiliki 10 controller dan disetiap controller memiliki 4 method action maka akan dimiliki 40 file view. Jika seluruh file view menggunakan file _Layout.cshtml sebagai layout maka pada 40 file view tersebut harus ditambahkan baris di atas.

Untuk otomatisasi agar seluruh file-file view menggunakan file layout yang sama dapat dilakukan dengan cara berikut ini. Langkah pertama adalah membuat file _ViewStart.cshtml yang disimpan pada folder Views. File ini adalah file yang akan dibaca oleh seluruh file view.

Kemudian pada file _ViewStart.cshtml diisi dengan kode di bawah ini.

```

_ViewStart.cshtml
@{
    Layout = "~/Views/Shared/_Layout.cshtml";
}

```

Selain file _ViewStart.cshtml juga dikenal file _ViewImports.cshtml. File _ViewImports.cshtml berfungsi untuk menyimpan directive seperti @using atau @addTagHelper agar bisa digunakan oleh seluruh file view. Untuk keperluan aplikasi web Book Store ini maka perlu dibuat file _ViewImports.cshtml yang disimpan pada folder Views. Isi dari file ini adalah sebagai berikut.

```

@using Microsoft.AspNetCore.Identity

```

```
@using SqlServerBookStore
@using SqlServerBookStore.Models
@using SqlServerBookStore.Models.AccountViewModels
@using SqlServerBookStore.Models.ManageViewModels
@addTagHelper *, Microsoft.AspNetCore.Mvc.TagHelpers
```

Baris pertama berfungsi untuk mendeklarasikan namespace SqlServerBookStore.Models agar seluruh model yang terdapat pada namespace tersebut dapat digunakan pada seluruh file view. Baris kedua adalah deklarasi yang berfungsi agar seluruh file view dapat menggunakan Tag Helper. Tag Helper akan dijelaskan secara detail pada sub bab selanjutnya.

Sintaks Dasar Razor

Pada bahasa pemrograman web biasanya digunakan penanda yang membedakan antara baris HTML biasa dan baris sintaks Razor. Baris yang memiliki penanda yang akan diparsing oleh runtime pada web server sesuai dengan perintah pada baris tersebut dan kemudian akan dirender menjadi kode HTML. Pada Razor, tanda yang digunakan adalah karakter @. Sintaks pada pemrograman web mempunyai beberapa jenis seperti blok kode dan ekspresi.

Persiapan

Agar pembaca dapat mencoba langsung penjelasan dan contoh-contoh pada sub bab ini maka terlebih dulu dapat melakukan persiapan untuk membuat komponen control dan komponen view sebagai berikut.

Untuk latihan akan dibuat controller baru yaitu Latihan dengan menambahkan file LatihanController.cs pada folder Controllers.

```
LatihanController.cs
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Mvc;

namespace SqlServerBookStore.Controllers
{
    public class LatihanController : Controller
    {
        public IActionResult Index()
        {
            return View();
        }
    }
}
```

Langkah selanjutnya adalah membuat komponen view dengan cara terlebih dahulu membuat folder Latihan pada folder Views. Kemudian menambahkan file Index.cshtml pada folder Views\Latihan.

```
Index.cshtml
@{
    Layout = null;
}

<!DOCTYPE html>

<html>
<head>
```

```

<meta name="viewport" content="width=device-width" />
<title>Latihan</title>
</head>
<body>
<h1>Latihan</h1>

</body>
</html>

```

Untuk mengakses halaman ini dapat dilakukan dengan memanggil controller Latihan dengan cara berikut ini.

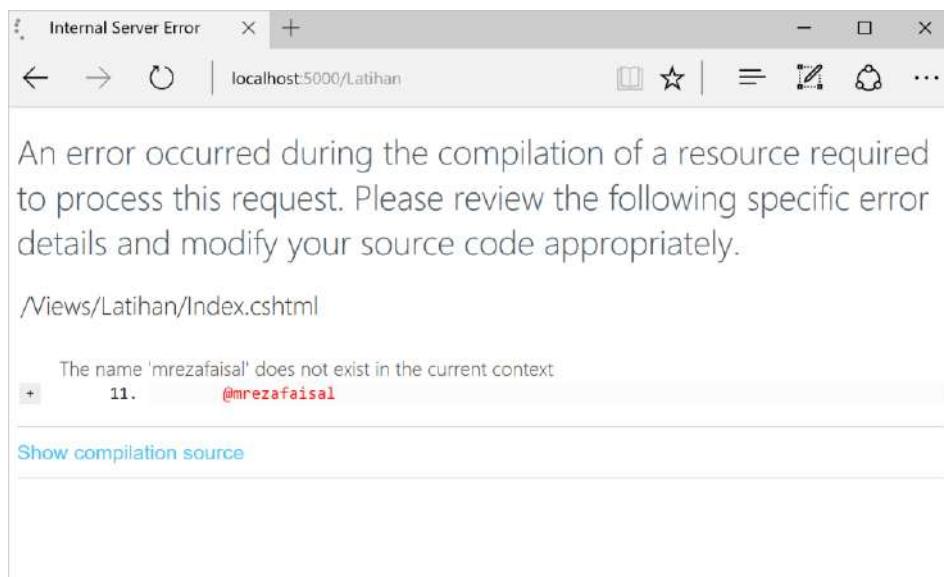
<https://localhost:44364/Latihan>

Selanjutnya pembaca dapat menggunakan halaman ini untuk mempraktikkan contoh-contoh yang akan diberikan pada sub bab ini.

Simbol @

Simbol @ adalah penanda awal dari yang digunakan oleh Razor untuk melakukan transisi dari HTML ke kode C#. Kemudian Razor akan mengevaluasi ekspresi C# dan menulis outputnya dalam bentuk HTML. Jika simbol @ diikuti oleh keyword Razor maka akan dilakukan transisi menjadi markup spesifik, selain itu akan dilakukan transisi menjadi kode C#.

Karena penggunaan simbol @ merupakan penanda awal sintaks Razor, maka penulisan simbol ini pada halaman file *.cshtml yang tidak mengikuti kaidah penulisan sintaks Razor akan membuat terjadinya kesalahan pemrograman. Sebagai contoh, jika ingin menulis nama user Twitter @mrezafaisal pada halaman ini, maka dipastikan Razor akan menganggap “mrezafaisal” di belakang simbol @ sebagai keyword Razor atau ekspresi C#. Tetapi karena “merezafaisal” tidak ditemukan didalam keyword Razor atau C# maka akan dapat dilihat pesan kesalahan seperti gambar di bawah ini.

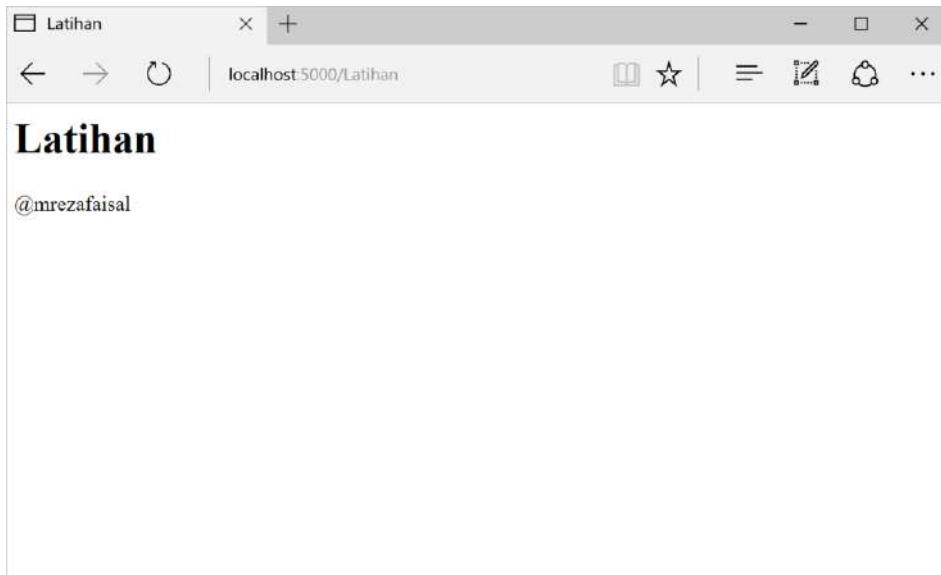


Gambar 135. Razor - Pesan kesalahan.

Untuk menghindari kesalahan terjadi maka perlu ditambahkan simbol @ sehingga menjadi sebagai berikut.

```
@@mrezafaisal
```

Hasilnya dapat dilihat pada gambar di bawah ini.



Gambar 136. Razor - Penggunaan simbol @@.

Untuk menggunakan simbol @ pada email, maka tidak diperlukan tambahan simbol @ lagi seperti contoh di bawah ini.

```
<a href="mailto:Support@contoso.com">Support@contoso.com</a>
```

Reserved Keyword

Reserverd keyword terbagi atas dua yaitu:

- Razor keyword.
- C# Razor keyword

Berikut adalah Razor keyword yang dapat digunakan pada ASP.NET Core, yaitu:

- functions.
- inherits.
- model.
- section.

Contoh penulisan keyword ini adalah sebagai berikut.

```
@functions{
    string HelloWorld()
    {
        return "Hello World";
}
```

Sedangkan C# Razor keyword adalah sebagai berikut:

- case.
- do.
- default.
- for.

- foreach.
- if.
- lock.
- switch.
- try.
- using.
- while.

Ekspresi

Ekspresi atau expression adalah baris yang berfungsi untuk menampilkan nilai dari suatu variable atau output dari suatu fungsi. Ekspresi dapat dibedakan menjadi beberapa tipe, yaitu:

- Ekspresi implisit (implicit expression).

Ekspresi ini diawali dengan simbol @ dan diikuti oleh kode C#. Dengan cara dapat digunakan untuk mengeksekusi method generic yang dimiliki suatu class. Berikut adalah contoh ekspresi implisit.

```
<h2>Ekspresi implisit</h2>
<p>@DateTime.Now</p>
<p>@DateTime.IsLeapYear(2016)</p>
<p>@DateTime.IsLeapYear(2017)</p>
<p>@HelloWorld()</p>
```

- Ekspresi eksplisit (explicit expression).

Ekspresi eksplisit diawali dengan simbol @ diikuti oleh tanda kurung (). Berikut adalah contoh implementasi ekspresi eksplisit.

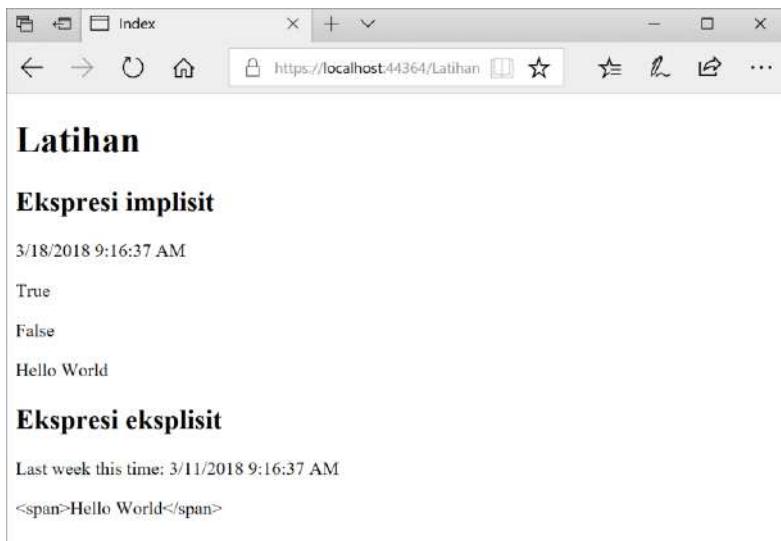
```
<h2>Ekspresi eksplisit </h2>
<p>Last week this time: @(DateTime.Now - TimeSpan.FromDays(7))</p>
```

- Ekspresi Encoding.

Ekspresi encoding adalah ekspresi C# untuk melakukan encoding kode HTML agar bisa ditampilkan pada halaman web. berikut adalah contoh implementasi ekspresi encoding ini.

```
@("<span>Hello World</span>")
```

Hasil ketiga contoh ekspresi di atas dapat dilihat pada gambar di bawah ini.



Gambar 137. Razor - Ekspresi implisit & eksplisit.

Blok Kode

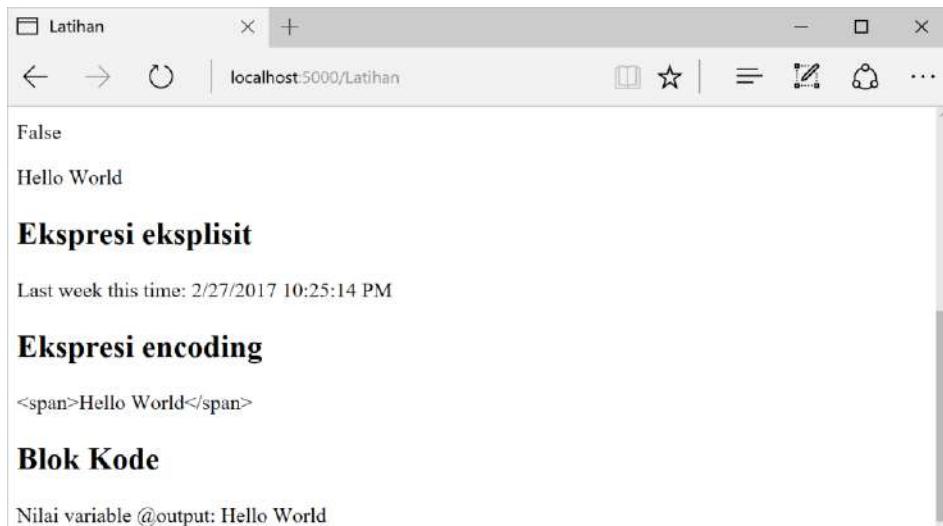
Blok kode Razor diawali dengan simbol @ kemudian diikuti dengan tanda {}. Berbeda dengan ekspresi, kode yang berada di dalam blok kode ini tidak dirender. Berikut adalah contoh implementasi blok kode.

```
@{
    var output = "Hello World";
}
```

Jika kode di atas ditulis pada Index.cshtml, maka dapat dilihat tidak ada output yang dapat dilihat pada web browser. Jika ingin menampilkan nilai variable output agar dapat dilihat pada web browser maka perlu digunakan ekspresi sebagai berikut.

```
<p>Nilai variable @@output: @output</p>
```

Hasilnya dapat dilihat pada gambar di bawah ini.



Gambar 138. Razor - Blok kode.

Penggunaan blok kode dapat dilakukan dengan beberapa cara, yaitu:

- Implicit transitions.

Di dalam blok kode selain dapat berisi kode C# juga dapat berisi kode HTML. Kode HTML yang berada di dalam blok kode akan dirender kembali menjadi kode HTML. Berikut adalah contoh implementasi cara ini.

```
@{
    var nama = "M Reza Faisal";
    <p>Hello @nama</p>
}
```

- Explicit delimited transition.

Cara ini dapat digunakan untuk mendefinisikan bagian yang harus dirender sebagai kode HTML.

```
@for (var i = 0; i < 5; i++)
{
    <p>Bilangan: @i</p>
}
```

Jika bagian di dalam blok tidak menggunakan tag HTML seperti contoh berikut ini, maka "Bilangan" dianggap sebagai kode C# sehingga akan terjadi kesalahan, karena kata "Bilangan" bukan keyword C#.

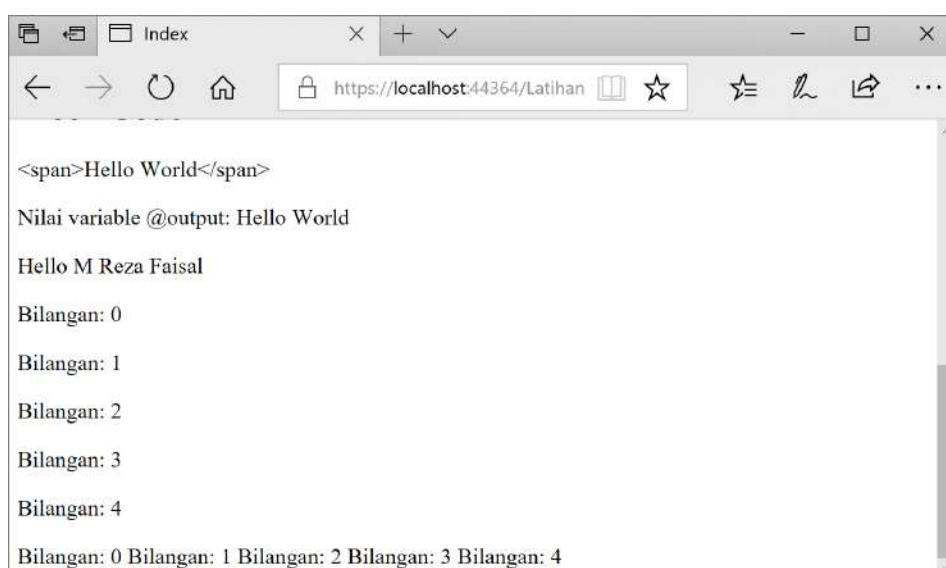
```
@for (var i = 0; i < 5; i++)
{
    Bilangan: @i
}
```

- Explicit line transition dengan simbol @:

Penggunaan simbol @: dapat digunakan menggantikan tag HTML yang dicontohkan pada cara di atas. Berikut adalah implementasi cara ini.

```
@for (var i = 0; i < 5; i++)
{
    @:Bilangan: @i
}
```

Hasil dari ketiga cara di atas dapat dilihat pada gambar di bawah ini.



Gambar 139. Razor - Blok kode.

Percabangan

Untuk melakukan pemeriksaan kondisi atau percabangan digunakan keyword berikut ini:

- @if, else if, else.
- @switch.

Berikut adalah contoh penulisan @if, else if, else.

```
@{var value = 5;}

@if (value % 2 == 0)
{
    <p>Bilangan genap</p>
}
else if (value >= 11)
{
    <p>Bilangan lebih besar dari 11.</p>
}
else
{
    <p>Bilangan lebih kecil dari 11 dan ganjil.</p>
}
```

Sedangkan untuk keyword @switch dapat dilihat contoh penggunaannya di bawah ini.

```
@switch (value)
{
    case 1:
        <p>Nilai adalah 1!</p>
        break;
    case 10:
        <p>Nilai adalah 10!</p>
        break;
    default:
        <p>Nilai bukan 1 atau 10.</p>
        break;
}
```

Pengulangan

Untuk pengulangan dapat digunakan keyword @for, @foreach, @while dan @do while. Berikut adalah contoh penggunaan keyword untuk pengulangan. Pada contoh di bawah ini dapat dilihat terdapat array names yang berisi 3 item.

```
<h2>Pengulangan</h2>
@{
    string[] names = new string[3];
    names[0] = "wati";
    names[1] = "budi";
    names[2] = "iwan";
}
<h3>@@for</h3>
@for (int i = 0; i < names.Length; i++)
{
    <text>@names[i] &nbsp;</text>
}

<h3>@@foreach</h3>
@foreach (var name in names)
{
    <text>@name &nbsp;</text>
}

<h3>@@while</h3>
```

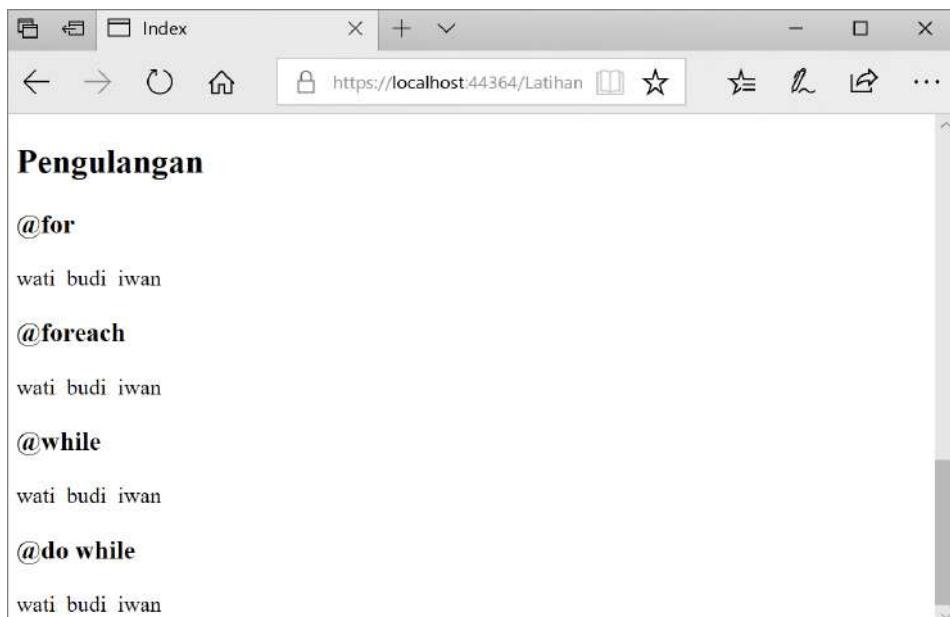
```

@{ var j = 0; }
@while (j < names.Length)
{
    <text>@names[j] &nbsp;</text>
    j++;
}

<h3>@@do while</h3>
@{ var k = 0; }
@do
{
    <text>@names[k] &nbsp;</text>
    k++;
} while (k < names.Length);

```

Hasilnya dapat dilihat pada gambar di bawah ini.



Gambar 140. Razor - Pengulangan.

Directive

Pada Razor terdapat beberapa directive penting yang dapat digunakan dengan fungsinya masing-masing.

Pada C#, keyword using digunakan untuk memastikan object siap digunakan. Pada Razor, hal seperti itu juga dapat dilakukan dengan menggunakan directive @using. Cara penggunaan directive @using akan dapat dilihat dalam penggunaan HTML Helper. Sebagai contoh digunakan untuk merender tag form seperti contoh di bawah ini.

```

@using (Html.BeginForm())
{
    <div>
        email:
        <input type="email" id="Email" name="Email" value="" />
        <button type="submit"> Register </button>
    </div>
}

```

Directive penting lainnya adalah @model yang terlihat pada halaman-halaman view pada project MyGuestBook atau EFCoreGuestBook. Directive ini berfungsi untuk menentukan

model yang digunakan pada suatu halaman view. Directive ini hanya dapat digunakan sekali pada halaman view. Contoh penggunaan directive ini dapat dilihat pada aplikasi MyGuestBook dan EFCoreGuestBook. Sebagai contoh pada project EFCoreGuestBook dapat dilihat pada file Create.cshtml pada folder Views/Home.

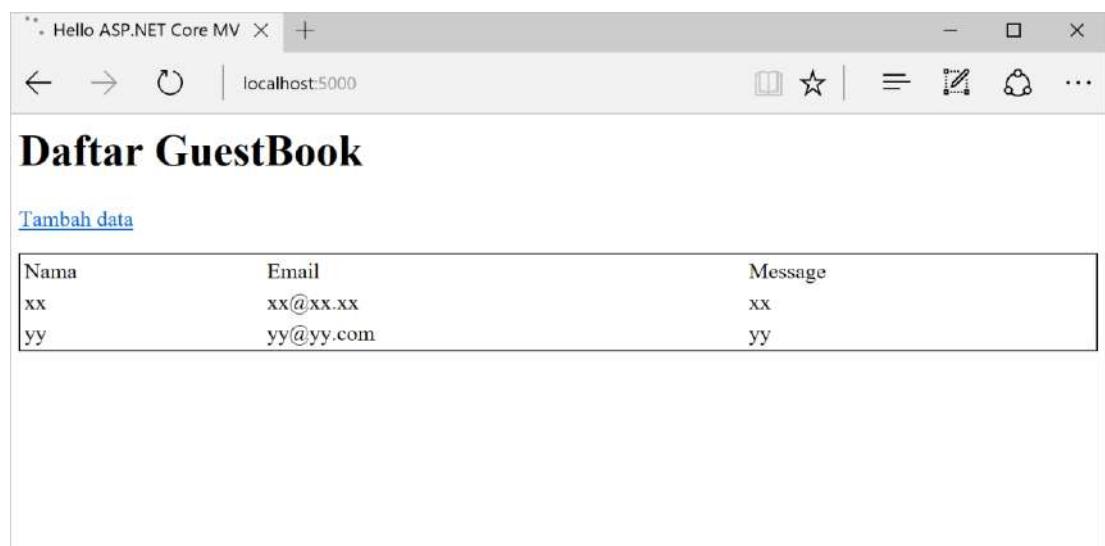
```
...  
@model EFCoreGuestBook.Models.GuestBook  
...
```

Contoh di atas digunakan untuk mengakses sebuah object dari model tersebut saja. Cara di atas biasanya digunakan pada halaman view untuk menampilkan form input dari model tersebut. Atau menampilkan detail informasi dari model tersebut.

Contoh yang lain juga dapat dilihat pada halaman Index.cshtml, dimana directive @model ditulis sebagai berikut.

```
@model IList<EFCoreGuestBook.Models.GuestBook>
```

Dari contoh di atas dapat dilihat directive @model juga dapat mengelola object collection suatu class model. Cara ini biasanya digunakan untuk menampilkan data pada tabel seperti pada gambar berikut.



The screenshot shows a web browser window titled "Hello ASP.NET Core MV". The address bar displays "localhost:5000". The main content area has a title "Daftar GuestBook" and a link "Tambah data". Below is a table with three columns: Nama, Email, and Message. The data rows are:

Nama	Email	Message
xx	xx@xx.xx	xx
yy	yy@yy.com	yy

Gambar 141. Daftar data tamu.

Exception Handling

Untuk penanganan kesalahan atau exception handling, seperti pada C#, maka dapat digunakan cara yang sama yaitu menggunakan statement @try, catch, finally. Berikut adalah contoh exception handling pada Razor.

```
@try
{
    throw new InvalidOperationException("Terjadi kesalahan.");
}
catch (Exception ex)
{
    <p>Pesan kesalahan: @ex.Message</p>
}
finally
{
    <p>Silakan lakukan aksi sekali lagi.</p>
}
```

Komentar

Untuk membuat komentar atau agar suatu baris tidak dieksekusi atau dinon-aktifkan dapat dilakukan dengan menggunakan cara penulisan sintaks Razor seperti berikut ini.

```
@{  
    /*  
     *  
     komentar 1  
     komentar 2  
     komentar 3  
     */  
  
    // komentar  
}
```

Cara di atas dilakukan jika baris berada di dalam blok kode Razor. Jika ingin melakukan menon-aktifkan blok kode Razor dapat digunakan cara sebagai berikut.

```
@*  
{@  
    /*  
     *  
     komentar 1  
     komentar 2  
     komentar 3  
     */  
  
    // komentar  
}  
*@
```

HTML Helper

HTML Helper adalah method yang output menghasilkan tag-tag HTML seperti tag untuk membuat link, label, form dan komponen-komponennya seperti input, textarea, select dan lain-lain. Berberapa HTML Helper telah digunakan pada aplikasi web SqlServerDbFirst dan SqlServerCodeFirst.

Berikut adalah daftar lengkap dari HTML Helper:

- Html.BeginForm.
- Html.EndForm.
- Html.Label.
- Html.TextBox.
- Html.TextArea.
- Html.Password.
- Html.DropDownList.
- Html.CheckBox.
- Html.RadioButton.
- Html.ListBox.
- Html.Hidden.

Jika pada halaman view terdapat form atau table yang digunakan untuk menampilkan data atau mengirimkan data berdasarkan suatu model, maka dapat digunakan HTML Helper di bawah ini:

- Html.LabelFor.
- Html.TextBoxFor.

- Html.TextAreaFor.
- Html.DropDownListFor.
- Html.CheckBoxFor.
- Html.RadioButtonFor.
- Html.ListBoxFor.
- Html.HiddenFor.

Link

Untuk membuat hyperlink dapat digunakan HTML Helper dengan sintaks berikut ini.

```
@Html.ActionLink(text, action, controller, value, htmlAttribute)
```

Keterangan:

- text, teks yang akan dilihat pada hyperlink.
- action, nama method action yang digunakan.
- controller, nama controller yang digunakan.
- value, nilai yang digunakan sebagai parameter pada method action.
- htmlAttribute, nilai yang dapat dimasukkan sebagai atribut pada hyperlink.

Berikut adalah contoh penggunaannya.

```
@Html.ActionLink("Dashboard", "Index", "Home", new { id = "123"}, new { @class = "style1"})
```

Hasilnya menjadi seperti berikut ini.

```
<a class="style1" href="/Home/Index/123">Dashboard</a>
```

Helper lain yang dapat digunakan untuk membuat hyperlink adalah method dengan sintaks berikut ini.

```
@Url.Action(action, controller, value)
```

Berikut adalah contoh penggunaan method di atas.

```
@Url.Action("Index", "Home", new {id = "123"})
```

Hasilnya adalah string sebagai berikut ini.

```
/Home/Index/123
```

Dari output di atas dapat dilihat jika method @Url.Action hanya akan bernilai string dengan format di atas. Sehingga method ini tidak dapat berdiri sendiri tetapi harus digunakan pada tag HTML seperti contoh berikut ini.

```
<a href='@Url.Action("Index", "Home", new {id = "123"})'>Dashboard</a>

<a href='@Url.Action("Index", "Home", new {id = "123"})'>
    
</a>
```

Label & Display

Method HTML Helper berikut ini digunakan untuk menampilkan label dengan sintaks sebagai berikut.

```
@Html.Label(expression, text, htmlAttribute)
```

Berikut adalah contoh penggunaan method di atas.

```
@Html.Label("LblName", "Author Name", new { @class = "style1" })
```

Kode di atas akan dirender menjadi tag HTML berikut ini.

```
<label class="style1" for="LblName">Author Name</label>
```

Selain itu juga dapat digunakan method dengan sintaks berikut ini.

```
@Html.LabelFor(expression)
```

Method ini biasanya digunakan untuk menampilkan atribut [Display(Name = value)] yang dimiliki oleh property dari class model. Sebagai contoh untuk class model Category maka digunakan kode di bawah.

```
@model SqlServerBookStore.Models.Category  
  
@Html.LabelFor(m => m.CategoryID)  
@Html.LabelFor(m => m.Name)
```

Hasil kode di atas akan dirender tag HTML sebagai berikut.

```
<label for="CategoryID">ID</label>  
<label for="Name">Book Category Name</label>
```

Untuk atribut for berisi dengan nama property yang menjadi nilai expression method ini, sebagai contoh pada baris terakhir digunakan “m => m.Name” sebagai nilai expression, dan dapat dilihat nilai atribut for adalah Name. Untuk atribut Name, method ini akan menampilkan “Book Category Name” yang merupakan nilai dari atribut [Display(Name = “Book Category Name”)] dari property ini. Jika property class model tidak memiliki atribut ini, maka yang ditampilkan adalah nama property itu sendiri.

Sedangkan untuk kasus menampilkan data pada tabel digunakan method dengan sintaks seperti berikut ini.

```
@Html.DisplayFor(expression)
```

Contoh kasus ini dapat dilihat pada file Index.cshtml pada folder yang sama dengan file Create.cshtml di atas.

```
@model IEnumerable<SqlServerDbFirst.Models.GuestBook>  
.  
. .  
@foreach (var item in Model) {  
    . .  
        @Html.DisplayFor(modelItem => item.Name)  
    . .  
}
```

Method tersebut tidak akan merender tag HTML tetapi hanya nilai dari property yang dipanggil yaitu nilai property Name dari class model GuestBook.

Form

Untuk membuat form, langkah pertama yang dilakukan adalah dengan cara menggunakan method dengan sintaks berikut ini.

```
@using (Html.BeginForm("action", "controller"))
{
    // komponen input
}
```

Keterangan:

- action, nama method action.
- controller, nama class controller.

Berikut adalah contoh penggunaan method di atas.

```
@using (Html.BeginForm("Index", "Home"))
{
}
```

Hasilnya adalah kode HTML di bawah ini.

```
<form action="/Home/Index" method="post">
</form>
```

Pada contoh di atas dapat dilihat cara menentukan nilai atribut action pada suatu form. Method di atas juga dapat ditulis singkat seperti berikut.

```
@using (Html.BeginForm())
{}
```

Jika kode di atas berada pada file yang dipanggil oleh method action Create dari controller Home seperti yang telah dicontohkan pada aplikasi EFCoreGuestBook di sub bab sebelumnya. Jika tidak ditentukan nilai parameter action dan controller seperti contoh sebelumnya, maka nilai parameter ini akan disesuaikan dengan method action dan controller yang menggunakan file view tersebut. Artinya method di atas akan dirender menjadi tag HTML berikut.

```
<form action="/Home/Create" method="post">
</form>
```

Setelah tag form dibuat dengan method di atas, maka komponen-komponen input dapat diletakkan didalamnya.

Method-method HTML Helper yang dapat digunakan di dalam @Html.BeginForm() dapat dibagi menjadi dua tipe, yaitu:

1. Standard, tipe HTML Helper ini digunakan tanpa ada hubungan dengan class model. Berikut adalah daftar HTML Helper tipe ini, yaitu:
 - @Html.TextBox
 - @Html.TextArea
 - @Html.Password
 - @Html.Hidden
 - @Html.CheckBox
 - @Html.RadioButton
 - @Html.DropDownList
 - @Html.ListBox
 - @Html.TextArea
2. Strongly Typed, tipe ini digunakan bersama dengan class model. Berikut
 - @Html.TextBoxFor

- @Html.PasswordFor
- @Html.HiddenFor
- @Html.CheckBoxFor
- @Html.RadioButtonFor
- @Html.DropDownListFor
- @Html.ListBoxFor
- @Html.TextAreaFor

Input Teks

Dari HTML Helper di atas beberapa diantaranya dapat digunakan sebagai input text pada form. Input text tipe standar dapat digunakan dengan sintaks berikut ini.

```
@Html.TextBox(expression, value, htmlAttribute)
```

Berikut adalah contoh penggunaannya.

```
@Html.TextBox("Name", "", new { @class = "style1" })
```

Hasilnya akan dirender tag HTML berikut ini.

```
<input class="style1" id="Name" name="Name" type="text" value="" />
```

Sedangkan untuk tipe strongly typed input teks dapat digunakan dengan sintaks sebagai berikut ini.

```
@Html.TextBoxFor(expression, htmlAttribute)
```

Berikut adalah contoh penggunaan method di atas dengan menggunakan class model Category.

```
@model SqlServerBookStore.Models.Category

@using (Html.BeginForm())
{
    @Html.LabelFor(m => m.Name)
    @Html.TextBoxFor(m => m.Name, new {@class = "style1"})
    <br/>
    <button type="submit" class="btn btn-success">Submit</button>
}
```

Untuk baris yang menggunakan method @Html.TextBoxFor akan dirender menjadi tag HTML berikut ini.

```
<input class="input-validation-error style1"
      data-val="true"
      data-val-length="Book Category Name tidak boleh lebih dari 256 karakter."
      data-val-length-max="256"
      data-val-required="Book Category Name harus diisi."
      id="Name"
      name="Name"
      type="text"
      value="" />
```

Hasil render di atas selain berisi informasi sesuai dengan nilai parameter yang dimasukkan pada method @Html.TextBoxFor juga berasal dari informasi dari atribut-atribut dari property Name pada class model Category.

Method lain yang dapat digunakan adalah @Html.TextArea dengan sintaks sebagai berikut ini.

```
@Html.TextArea(expression, value, htmlAttribute)
```

Sedangkan sintaks text area tipe strongly typed adalah sebagai berikut.

```
@Html.TextAreaFor(expression, value, htmlAttribute)
```

Dan berikut ini adalah contoh penggunaan method ini.

```
@model SqlServerBookStore.Models.Category  
  
@using (Html.BeginForm())  
{  
    @Html.LabelFor(m => m.Name)<br/>  
    @Html.TextAreaFor(m => m.Name, new {@class = "style1"})  
    <br/>  
    <button type="submit" class="btn btn-success">Submit</button>  
}
```

Input Satu Pilihan

Dropdownlist dan radio button adalah komponen untuk input form satu pilihan, artinya user hanya dapat memilih sebuah nilai dari pilihan yang tersedia. Untuk HTML Helper dropdown list dapat menggunakan contoh berikut ini.

```
@Html.DropDownList("DropDownListCategory",  
new SelectList(new List<Object> {  
    new { value = "1", text = "Computer" },  
    new { value = "2", text = "History" }  
},  
"value", "text", ""), "Choose Book Category")
```

Contoh di atas akan dirender menjadi tag HTML berikut ini.

```
<select id="DropDownListCategory" name="DropDownListCategory">  
    <option value="">Choose Book Category</option>  
    <option value="1">Computer</option>  
    <option selected="selected" value="2">History</option>  
</select>
```

Sedangkan kode di bawah ini adalah contoh untuk method @Html.DropDownListFor.

```
@model SqlServerBookStore.Models.Book  
  
@using (Html.BeginForm())  
{  
    @Html.LabelFor(m => m.CategoryID)  
    @Html.DropDownListFor(m => m.CategoryID,  
    new SelectList(new List<Object> {  
        new { value = "1", text = "Computer" },  
        new { value = "2", text = "History" }  
    }, "value", "text", ""), "Choose Book Category")  
  
    <button type="submit" class="btn btn-success">Submit</button>  
}
```

Method lain yang dapat digunakan @Html.RadioButton dengan contoh sebagai berikut.

```
@Html.RadioButton("RadioButtonCategory", "1") <span>Computer</span>
```

```
@Html.RadioButton("RadioButtonCategory", "2") <span>History</span>
```

Kode di atas akan dirender menjadi tag HTML sebagai berikut.

```
<input id="RadioButtonCategory" name="RadioButtonCategory" type="radio" value="1" />
<span>Computer</span>

<input id="RadioButtonCategory" name="RadioButtonCategory" type="radio" value="2" />
<span>History</span>
```

Sedangkan untuk method @Html.RadioButtonFor digunakan contoh kode berikut ini.

```
@model SqlServerBookStore.Models.Book

@using (Html.BeginForm())
{
    @Html.LabelFor(m => m.CategoryID)
    @Html.RadioButtonFor(m => m.CategoryID, "1") <span>Computer</span>
    @Html.RadioButtonFor(m => m.CategoryID, "2") <span>History</span>
    <br/>
    <button type="submit" class="btn btn-success">Submit</button>
}
```

Hasil dari method @Html.RadioButtonFor akan menghasilkan tag HTML seperti berikut ini.

```
<input data-val="true"
       data-val-regex="Category ID harus angka" data-val-regex-pattern="^[0-9]*$"
       data-val-required="Category ID harus diisi."
       id="CategoryID" name="CategoryID" type="radio" value="1" />

<span>Computer</span>
```

Input Banyak Pilihan

Salah satu contoh input yang dapat dipilih lebih dari 1 nilai adalah checkbox dan list box. Berikut ini adalah penggunaan method @Html.CheckBox.

```
@Html.CheckBox("CheckBoxAuthor1") <span>Mohammad</span>
@Html.CheckBox("CheckBoxAuthor2") <span>Reza</span>
@Html.CheckBox("CheckBoxAuthor3") <span>Faisal</span>
```

Hasilnya berupa tag HTML berikut ini.

```
<input name="CheckBoxAuthor1" type="checkbox" value="false" />
<input name="CheckBoxAuthor2" type="checkbox" value="false" />
<input name="CheckBoxAuthor3" type="checkbox" value="false" />
```

Untuk membuat list box digunakan contoh berikut ini.

```
@Html.ListBox("ListBoxAuthor", new SelectList(new List<Object> {
    new { value = "1", text = "Mohammad" },
    new { value = "2", text = "Reza" },
    new { value = "3", text = "Faisal" }
}, "value", "text", ""))
```

Tag HTML yang dihasilkan adalah sebagai berikut.

```
<select id="ListBoxAuthor" multiple="multiple" name="ListBoxAuthor">
<option value="1">Mohammad</option>
<option value="2">Reza</option>
<option value="3">Faisal</option>
</select>
```

Tombol

Untuk membuat tombol submit data tidak tersedia HTML Helper, sehingga cukup digunakan tag HTML untuk membuat tombol seperti contoh berikut ini.

```
<button type="submit">Simpan</button>
<button type="reset">Batal</button>
```

Validasi

Validasi adalah proses untuk memeriksa kebenaran dari nilai yang dimasukkan pada komponen input pada form. Implementasi validasi pada form dilakukan dengan menambahkan @Html.ValidationSummary(true) di dalam blok @Html.BeginForm seperti contoh di bawah ini.

```
@model SqlServerBookStore.Models.Category

@using (Html.BeginForm())
{
    @Html.ValidationSummary(true)

    @Html.LabelFor(m => m.Name)
    @Html.TextBoxFor(m => m.Name, new {@class = "style1"})
    @Html.ValidationMessageFor(m => m.Name)
    <br/><br/>
    <button type="submit" class="btn btn-success">Submit</button>
}
```

Jika ingin melakukan validasi input textbox di atas maka cukup dengan menambahkan method @Html.ValidationMessageFor setelah method @Html.TextBoxFor. Dapat dilihat pada kedua method tersebut menggunakan expression yang sama yaitu “m => m.Name”.

Berikut ini adalah hasil render menjadi tag HTML dari kode di atas.

```
<form action="/Latihan/Template" method="post">

    <label for="Name">Book Category Name</label><br/>

    <input class="style1"
        data-val="true"
        data-val-length="Book Category Name tidak boleh lebih dari 256 karakter."
        data-val-length-max="256"
        data-val-required="Book Category Name harus diisi."
        id="Name" name="Name" type="text" value="" />

    <span class="field-validation-valid" data-valmsg-for="Name" data-valmsg-replace="true"></span>

    <br/><br/>

    <button type="submit" class="btn btn-success">Submit</button>
</form>
```

Berikut adalah tampilan dari contoh kode di atas.

Gambar 142. Tampilan form sebelum proses validasi.

Jika input textbox tidak diisi dan tombol submit diklik maka akan ditampilkan pesan kesalahan berikut ini.

Gambar 143. Tampilan form setelah proses validasi.

Dan berikut ini adalah perubahan tag HTML setelah proses validasi.

```

<form action="/Latihan/Template" method="post">

    <label for="Name">Book Category Name</label><br/>

    <input class="input-validation-error style1"
        data-val="true"
        data-val-length="Book Category Name tidak boleh lebih 256 karakter."
        data-val-length-max="256"
        data-val-required="Book Category Name harus diisi."
        id="Name" name="Name" type="text" value="" />

    <span class="field-validation-error" data-valmsg-for="Name" data-valmsg-replace="true">Book Category Name harus diisi.</span>

    <br/><br/>

    <button type="submit" class="btn btn-success">Submit</button>

```

```
</form>
```

Proses validasi di atas adalah proses validasi server side, artinya perubahan atau penampilan pesan kesalahan dilakukan oleh pemrograman server side. Proses validasi server side ini dilakukan oleh method action pada class controller. Penjelasan proses validasi ini akan dijelaskan pada sub bab Controller.

Catatan

Ada dua hal yang perlu diperhatikan jika menggunakan HTML Helper pada komponen view. Yang pertama adalah jika telah dibuat design tampilan form dengan antarmuka seperti berikut ini.

The image shows a user interface for a form. It consists of several input fields and a set of buttons at the bottom. The fields are labeled: 'First Name *' (with a required asterisk), 'Last Name *' (with a required asterisk), 'Middle Name / Initial' (without a required asterisk), 'Gender' (with radio buttons for 'Male' and 'Female' where 'Female' is selected), and 'Date Of Birth *' (with a required asterisk). Below the fields are three buttons: 'Cancel' (blue), 'Reset' (blue), and 'Submit' (green).

Gambar 144. Antarmuka design form.

Dengan kode HTML sebagai berikut.

```
<form id="demo-form2" class="form-horizontal form-label-left">
  <div class="form-group">
    <label class="control-label col-md-3 col-sm-3 col-xs-12" for="first-name">First Name
      <span class="required">*</span>
    </label>
    <div class="col-md-6 col-sm-6 col-xs-12">
      <input type="text" id="first-name" required="required" class="form-control col-md-7 col-xs-12">
    </div>
  </div>
  <div class="form-group">
    <label class="control-label col-md-3 col-sm-3 col-xs-12" for="last-name">Last Name
      <span class="required">*</span>
    </label>
    <div class="col-md-6 col-sm-6 col-xs-12">
      <input type="text" id="last-name" name="last-name" required="required" class="form-control col-md-7 col-xs-12">
    </div>
  </div>
  <div class="form-group">
    <label for="middle-name" class="control-label col-md-3 col-sm-3 col-xs-12">
```

```

        Middle Name / Initial
    </label>
    <div class="col-md-6 col-sm-6 col-xs-12">
        <input id="middle-name" class="form-control col-md-7 col-xs-12"
               type="text" name="middle-name">
    </div>
</div>
<div class="form-group">
    <label class="control-label col-md-3 col-sm-3 col-xs-12">Gender
    </label>
    <div class="col-md-6 col-sm-6 col-xs-12">
        <div id="gender" class="btn-group" data-toggle="buttons">
            <label class="btn btn-default"
                  data-toggle-class="btn-primary"
                  data-toggle-passive-class="btn-default">
                <input type="radio" name="gender" value="male">
                Male
            </label>
            <label class="btn btn-primary"
                  data-toggle-class="btn-primary"
                  data-toggle-passive-class="btn-default">
                <input type="radio" name="gender" value="female">
                Female
            </label>
        </div>
    </div>
</div>
<div class="form-group">
    <label class="control-label col-md-3 col-sm-3 col-xs-12">
        Date Of Birth
        <span class="required">*</span>
    </label>
    <div class="col-md-6 col-sm-6 col-xs-12">
        <input id="birthday"
               class="date-picker form-control col-md-7 col-xs-12"
               required="required" type="text">
    </div>
</div>
<div class="ln_solid"></div>
<div class="form-group">
    <div class="col-md-6 col-sm-6 col-xs-12 col-md-offset-3">
        <button class="btn btn-primary" type="button">Cancel</button>
        <button class="btn btn-primary" type="reset">Reset</button>
        <button type="submit" class="btn btn-success">Submit</button>
    </div>
</div>
</form>

```

Jika hasil design di atas akan digunakan sebagai halaman view maka tag <form> dan tag <input> di atas harus diubah menjadi method HTML Helper yang telah dipelajari sebelumnya. Dari kasus ini dapat dilihat bahwa hasil design tidak dapat langsung digunakan sebagai halaman view.

Hal yang kedua adalah ketika halaman view atau master layout yang telah menggunakan HTML Helper perlu diubah designnya oleh web designer. Maka web design tidak bisa mengubah atau memperbaiki design jika tidak memiliki pengetahuan tentang sintaks Razor. Selain itu sintaks Razor dan method HTML Helper akan membuat antarmuka saat dilihat pada tool web design yang digunakan oleh web designer.

Dari kedua hal tersebut maka perlu ada cara alternatif untuk membuat halaman view. Salah satu caranya adalah dengan menggunakan Tag Helper.

Tag Helper

Seperti disebutkan di atas keberadaan tag helper dapat menjadi cara alternatif untuk membuat halaman view. Untuk aplikasi SqlServerDbFirst dan SqlServerCodeFirst telah menggunakan tag helper.

Secara umum fungsi HTML helper dan tag helper adalah sama, yaitu digunakan untuk membuat link, label, display, form dan lain-lain. Namun dengan cara yang berbeda. Berikut ini adalah penjelasan cara menggunakan tag helper dan contoh-contohnya terkait dengan pembangunan aplikasi EFCoreBookStore.

Persiapan

Untuk menggunakan tag helper pada halaman view, maka harus ditambahkan baris berikut ini pada file _ViewImports.cshtml.

```
@addTagHelper *, Microsoft.AspNetCore.Mvc.TagHelpers
```

Link

Berikut ini adalah tag HTML yang digunakan untuk membuat link.

```
<a href="/Category/Create" class="style1">  
    Add Data  
</a>
```

Atau untuk link image digunakan tag HTML seperti berikut.

```
<a href="/Category/Create" class="style1">  
      
</a>
```

Dengan menggunakan tag helper maka kedua tag HTML di atas dapat ditulis dengan sintaks berikut ini.

```
<a asp-controller="controller_name" asp-action="action_name" >text</a>
```

Dari sintaks di atas dapat dilihat bahwa tag helper tetap menggunakan tag HTML sebagai dasar. Yang membedakan dengan tag HTML biasa adalah pada atribut-atribut yang hanya dimiliki oleh tag helper yaitu:

- asp-controller, atribut yang bernilai nama controller.
- asp-action, atribut yang bernilai nama action.

Sehingga kedua tag HTML di atas dapat dengan mudah diubah menjadi tag helper dengan cara di bawah ini.

```
<a asp-controller="Category" asp-action="Create" class="style1">Test</a>  
  
<a href="/Category/Create" class="style1">  
      
</a>
```

Dengan menggunakan tag helper, elemen HTML dapat memiliki gabungan atribut-atribut HTML dan atribut-atribut tag helper.

Label

Sintaks Tag Helper untuk label adalah sebagai berikut.

```
<label asp-for="property_name"></label>
```

Keterangan:

- property_name, nama property dari class model yang akan ditampilkan oleh elemen label.

Berikut adalah contoh penggunaan tag helper label.

```
@model SqlServerDbFirst.Models.GuestBook  
<label asp-for="Name" class="control-label"></label>
```

Image

Untuk membuat elemen menjadi tag helper dapat dilakukan dengan sintaks berikut ini.

```
<img src("~/URL" asp-append-version="true" />
```

Berikut ini adalah contoh penggunaan tag helper ini yang nanti dapat dilihat pada halaman Book/Index.cshtml yang dapat dilihat pada sub bab selanjutnya yaitu sub bab Book Store: Komponen View.

```
<img src "~/upload/@(item.ISBN).jpg" asp-append-version="true" />
```

Tag helper tersebut akan dirender menjadi sebagai berikut.

```
<img src "/upload/12345.jpg?v=L16QIS29B9y7C-nPIzdhZpiMMhx2nuarhCj-QvWUnsU" />
```

Tag helper dengan atribut asp-append-version cocok digunakan untuk menampilkan data yang melibatkan gambar, karena tanpa tambahan atribut tersebut gambar akan diambil dari cache karena memiliki nama file yang sama. Tetapi dengan penambahan nilai versi di belakang nama file maka dianggap nama file berbeda, sehingga file gambar diambil dari kembali dari server.

Form

Untuk membuat elemen <form> menjadi tag helper dengan menggunakan sintaks berikut ini.

```
<form asp-controller="controller_name"  
      asp-action="action_name"  
      asp-anti-forgery="false/true"  
      asp-route-returnurl="url">  
  . . .  
</form>
```

Keterangan:

- controller_name, nama class controller.
- action_name, nama method action.
- asp-anti-forgery, opsi penggunaan anti forgery.
- url, adalah url redirect untuk kembali.

Atribut-atribut di atas tidak seluruhnya harus digunakan. Implementasinya dapat dilihat pada contoh di bawah ini.

```
<form asp-controller="Category" asp-action="Create">
</form>
```

Hasilnya render adalah sebagai berikut.

```
<form action="/Category/Create" method="post">
</form>
```

Input

Elemen <input> pada tag HTML dapat digunakan untuk beberapa tipe input, yaitu:

- Text untuk input teks.
- Radio button, tipe ini digunakan untuk memilih sebuah nilai dari beberapa pilihan pilihan nilai yang tersedia.
- Check box bentuk ini digunakan digunakan untuk memilih lebih dari satu nilai dari beberapa pilihan nilai yang tersedia.
- File, tipe ini digunakan untuk memilih file yang akan diupload.

Tag helper <input> memiliki kemampuan untuk menentukan tipe input secara otomatis berdasarkan tipe data dari property pada class model. Sebagai contoh dimiliki class model berikut ini.

```
ContohModel.cs
using System;
using System.ComponentModel.DataAnnotations;

namespace SqlServerBookStore.Models
{
    public partial class ContohModel{
        [Display(Name ="Contoh Text")]
        public String ContohText{set; get; }

        [Display(Name ="Contoh Date Time")]
        public DateTime ContohDateTime{set; get; }

        [Display(Name ="Contoh Number")]
        public Double ContohNumber{set; get; }

        [Display(Name ="Contoh Boolean")]
        public Boolean ContohBoolean{set; get; }
    }
}
```

Kemudian berikut ini adalah contoh penggunaan tag helper input.

```
@model SqlServerBookStore.Models.ContohModel

<form asp-controller="Latihan" asp-action="Template">
    <label asp-for="ContohText"></label>:
    <input asp-for="ContohText" />
    <br />

    <label asp-for="ContohDateTime"></label>:
```

```

<input asp-for="ContohDateTime" />
<br />

<label asp-for="ContohNumber"></label>:
<input asp-for="ContohNumber" />
<br />

<label asp-for="ContohBoolean"></label>:
<input asp-for="ContohBoolean" />
<br />

<button type="submit" class="btn btn-success">Submit</button>
</form>

```

Pada tag helper input digunakan atribut asp-for untuk menentukan property class model yang ditangani oleh tag helper. Dari contoh di atas dapat dilihat tidak ada penentuan nilai untuk atribut "type" pada tag <input> seperti umumnya ditemui pada tag HTML. Hasil dari render HTML contoh tag helper di atas adalah sebagai berikut.

```

<form action="/Latihan/Template" method="post">
<label for="ContohText">Contoh Text</label>
<input type="text" id="ContohText" name="ContohText" value="" />
<br/>

<label for="ContohDateTime">Contoh Date Time</label>:
<input type="datetime" data-val="true"
       data-val-required="The Contoh Date Time field is required."
       id="ContohDateTime" name="ContohDateTime" value="" />
<br/>

<label for="ContohNumber">Contoh Number</label>:
<input type="text" data-val="true"
       data-val-number="The field Contoh Number must be a number."
       data-val-required="The Contoh Number field is required."
       id="ContohNumber" name="ContohNumber" value="" />
<br/>

<label for="ContohBoolean">Contoh Boolean</label>:
<input data-val="true"
       data-val-required="The Contoh Boolean field is required."
       id="ContohBoolean" name="ContohBoolean" type="checkbox"
       value="true" />
<br/>

<button type="submit" class="btn btn-success">Submit</button>
</form>

```

Dari hasil render di atas dapat dilihat secara otomatis telah ditambahkan atribut-atribut pada setiap elemen input sesuai dengan tipe datanya.

Tag helper input juga dapat membaca atribut pada setiap property pada class model untuk menentukan tipe input yang akan digunakan. Berikut ini adalah contoh model yang menggunakan atribut untuk menentukan tipe data untuk property.

ContohAtributModel.cs
<pre> using System; using System.Collections.Generic; using System.Linq; using System.Threading.Tasks; using System.ComponentModel.DataAnnotations; </pre>

```

namespace SqlServerBookStore.Models
{
    public class ContohAtributModel
    {
        [Display(Name = "Contoh Email")]
        [EmailAddressAttribute]
        public String ContohEmail { set; get; }

        [Display(Name = "Contoh URL")]
        [UrlAttribute]
        public String ContohUrl { set; get; }

        [Display(Name = "Contoh Phone")]
        [PhoneAttribute]
        public String ContohPhone { set; get; }

        [Display(Name = "Contoh Password")]
        [DataType(DataType.Password)]
        public String ContohPassword { set; get; }

        [Display(Name = "Contoh Date")]
        [DataType(DataType.Date)]
        public DateTime ContohDate { set; get; }

        [Display(Name = "Contoh Time")]
        [DataType(DataType.Time)]
        public DateTime ContohTime { set; get; }
    }
}

```

Dan berikut ini adalah contoh penggunaan tag helper input untuk class model di atas.

```

@model SqlServerBookStore.Models.ContohAtributModel

<form asp-controller="Latihan" asp-action="Template">
    <label asp-for="ContohEmail"></label>:
    <input asp-for="ContohEmail" />
    <br />

    <label asp-for="ContohUrl"></label>:
    <input asp-for="ContohUrl" />
    <br />

    <label asp-for="ContohPhone"></label>:
    <input asp-for="ContohPhone" />
    <br />

    <label asp-for="ContohPassword"></label>:
    <input asp-for="ContohPassword" />
    <br />

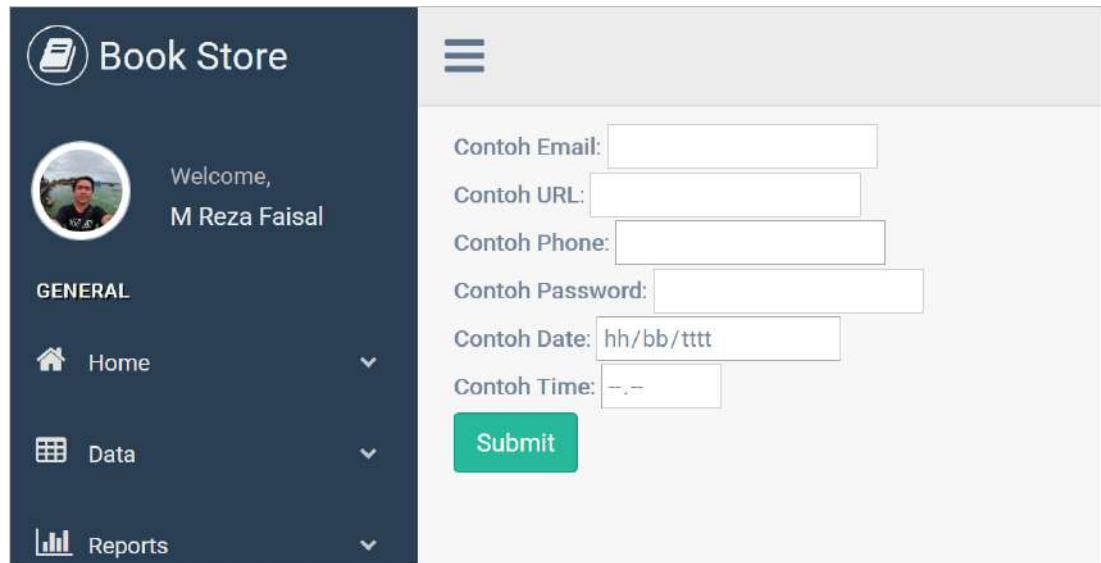
    <label asp-for="ContohDate"></label>:
    <input asp-for="ContohDate" />
    <br />

    <label asp-for="ContohTime"></label>:
    <input asp-for="ContohTime" />
    <br />

    <button type="submit" class="btn btn-success">Submit</button>
</form>

```

Kode di atas akan menghasilkan antarmuka seperti pada gambar di bawah ini.



Gambar 145. Contoh antarmuka implementasi tag helper input.

Dan berikut ini adalah hasil render HTML dari contoh tag helper di atas.

```
<form action="/Latihan/Template" method="post">
    <label for="ContohEmail">Contoh Email</label>
    <input type="email" data-val="true"
        data-val-email="The Contoh Email field is not a valid e-
        mail address."
        id="ContohEmail" name="ContohEmail" value="" />
    <br/>

    <label for="ContohUrl">Contoh URL</label>
    <input type="url" data-val="true"
        data-val-url="The Contoh URL field is not a valid fully-
        qualified http, https, or ftp URL."
        id="ContohUrl" name="ContohUrl" value="" />
    <br/>

    <label for="ContohPhone">Contoh Phone</label>
    <input type="tel" data-val="true"
        data-val-
        phone="The Contoh Phone field is not a valid phone number."
        id="ContohPhone" name="ContohPhone" value="" />
    <br/>

    <label for="ContohPassword">Contoh Password</label>
    <input type="password" id="ContohPassword" name="ContohPassword" />
    <br/>

    <label for="ContohDate">Contoh Date</label>
    <input type="date" data-val="true"
        data-val-required="The Contoh Date field is required."
        id="ContohDate" name="ContohDate" value="" />
    <br/>

    <label for="ContohTime">Contoh Time</label>
    <input type="time" data-val="true"
        data-val-required="The Contoh Time field is required."
        id="ContohTime" name="ContohTime" value="" />
    <br/>
```

```
<button type="submit" class="btn btn-success">Submit</button>
</form>
```

Dari contoh di atas dapat dilihat bagaimana secara otomatis tag helper input memberikan atribut HTML sesuai dengan atribut property class model.

Walau pada contoh di atas diperlihatkan cara kerja otomatis tag helper input untuk menentukan nilai atribut type, tetapi developer tetap dapat memberikan nilai atribut type secara manual. Misalnya dapat dilihat pada contoh di bawah ini.

```
<label asp-for="ContohPassword"></label>
<input asp-for="ContohPassword" type="password" />

<label asp-for="ContohText"></label>
<input asp-for="ContohText" type="radio" value="true"/>
```

TextArea

Berikut ini adalah sintaks penggunaan tag helper textarea.

```
<textarea asp-for="property_name"></textarea>
```

Berikut adalah contoh penggunaan tag helper di atas.

```
@model SqlServerBookStore.Models.ContohModel

<form asp-controller="Latihan" asp-action="Template">
    <label asp-for="ContohText"></label><br />
    <textarea asp-for="ContohText" rows="3"></textarea>
    <br />

    <button type="submit" class="btn btn-success">Submit</button>
</form>
```

Tag helper textarea di atas akan dirender menjadi tag HTML berikut ini.

```
<textarea id="ContohText" name="ContohText">
```

Select

Berikut adalah sintaks tag helper select.

```
<select asp-for="property_name" asp-items="data">
</select>
```

Keterangan:

- property_name, nama property dari class model yang dikelola oleh tag helper select.
- data, object collection yang berisi data untuk ditampilkan pada elemen select.

Berikut adalah contoh penggunaan tag helper select.

```
@model SqlServerBookStore.Models.ContohModel

<form asp-controller="Latihan" asp-action="Template">
    <label asp-for="ContohText"></label><br />
    <select asp-for="ContohText"
        asp-
        items="@new SelectList(ViewBag.Authors, "AuthorID", "Name"))"></select>
    <br />

    <button type="submit" class="btn btn-success">Submit</button>
```

```
</form>
```

Pada contoh di atas dapat dilihat pada atribut asp-items diisi dengan data dari ViewBag.Authors. Nilai pada object ViewBag.Authors diisi dari komponen controller dengan kode berikut ini.

```
LatihanController.cs
```

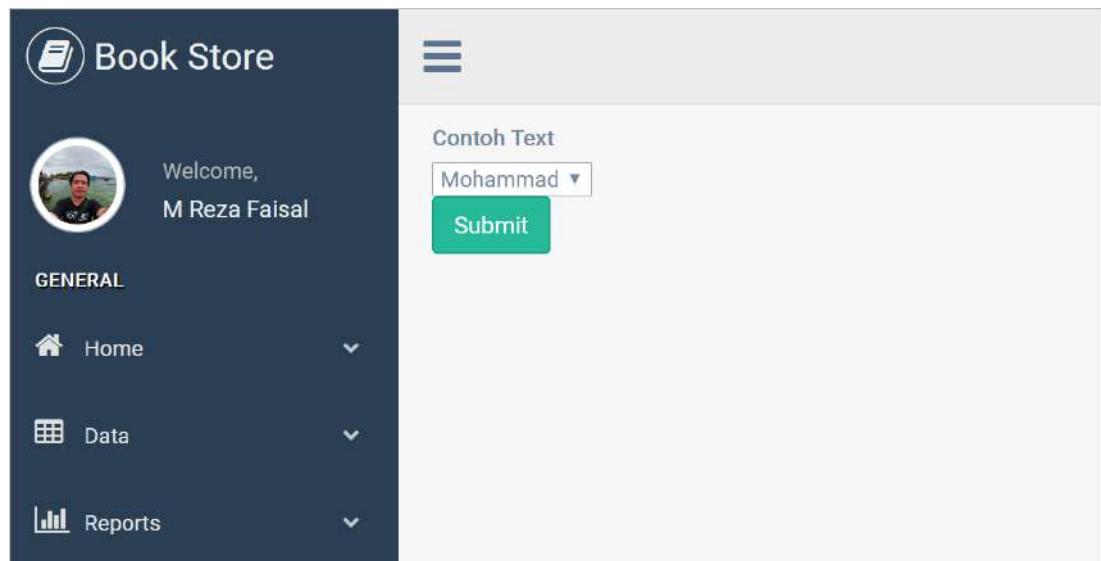
```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Mvc;
using SqlServerBookStore.Models;
using SqlServerBookStore.Data;

namespace SqlServerBookStore.Controllers
{
    public class LatihanController : Controller
    {
        private readonly ApplicationDbContext _context;

        public LatihanController(ApplicationDbContext context)
        {
            _context = context;
        }
        public IActionResult Index()
        {
            return View();
        }

        public IActionResult Template()
        {
            ViewBag.Authors = _context.Authors.ToList();
            return View();
        }
    }
}
```

Hasilnya dapat dilihat pada gambar di bawah ini.



Gambar 146. Contoh antarmuka implementasi tag helper select.

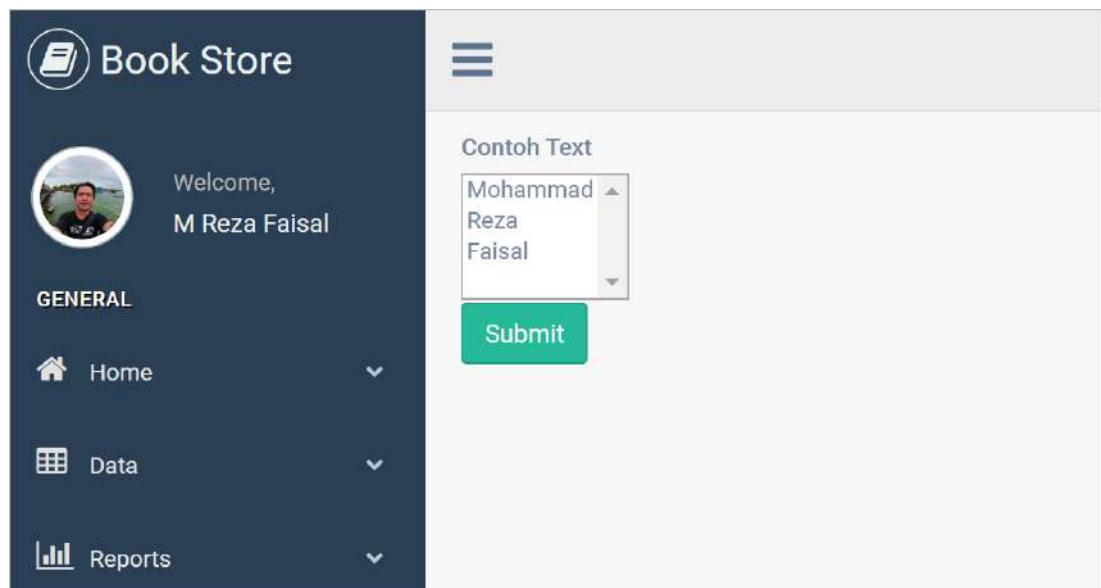
Tag helper select di atas dirender menjadi tag HTML berikut ini.

```
<select id="ContohText" name="ContohText">
    <option value="1">Mohammad</option>
    <option value="2">Reza</option>
    <option value="3">Faisal</option>
</select>
```

Tag <select> di atas hanya dapat digunakan untuk memilih satu nilai dari daftar nilai yang disediakan. Jika ingin menggunakan tag <select> agar dapat digunakan untuk memilih lebih dari satu nilai maka dapat digunakan atribut multiple seperti contoh berikut ini.

```
<select asp-for="ContohText"
       asp-items="@new SelectList(ViewBag.Authors, "AuthorID", "Name")"
       multiple="multiple">
</select>
```

Hasilnya dapat dilihat pada gambar di bawah ini.



Gambar 147. Contoh antarmuka implementasi tag helper select dengan atribut multiple.

Tombol

Untuk membuat tombol submit data dengan tag helper cukup digunakan tag HTML berikut ini.

```
<button type="submit">Submit</button>
<button type="reset">Reset</button>
```

Validasi

Untuk proses validasi, langkah pertama adalah menambahkan baris berikut ini di dalam tag <form></form>.

```
<div asp-validation-summary="All"></div>
```

Pada baris di atas digunakan atribut asp-validation-summary. Nilai yang dapat diberikan kepada atribut tersebut adalah sebagai berikut:

- All, jika ingin menampilkan pesan validasi pada bagian tag helper “asp-validation-summary” di atas dan di tag helper “asp-validation-for” di bawah ini.
- ModelOnly, hanya akan menampilkan pesan validasi di bagian tag helper “asp-validation-for” di bawah ini.
- None, tidak akan melakukan proses validasi.

Tag dapat digunakan untuk menampilkan pesan validasi. Berikut ini adalah sintaks yang digunakan untuk validasi.

```
<span asp-validation-for="property_name"></span>
```

Berikut adalah contoh penggunaan tag helper validasi ini.

```
@model SqlServerBookStore.Models.Category

<form asp-controller="Category" asp-action="Create">
    <div asp-validation-summary="All"></div>

    <label asp-for="Name"></label>
    <input asp-for="Name">
    <span asp-validation-for="Name"></span>
</form>
```

Book Store: Komponen View

Dari penjelasan di atas maka pada sub bab ini akan dilakukan pembuatan file komponen view untuk setiap fitur dari aplikasi book store. Berikut adalah daftar file komponen view untuk setiap fitur:

1. Mengelola kategori buku.
 - Index.cshtml yang akan menampilkan tabel daftar kategori buku. Pada halaman ini juga terdapat fitur untuk menghapus data kategori buku. Kategori buku hanya bisa dihapus jika belum ada buku yang menggunakan kategori tersebut.
 - Detail.cshtml berfungsi untuk menampilkan data kategori buku yang dipilih pada halaman daftar kategori buku.
 - Create.cshtml sebagai form input untuk menambah data kategori buku.
 - Edit.cshtml untuk form mengedit data kategori buku.

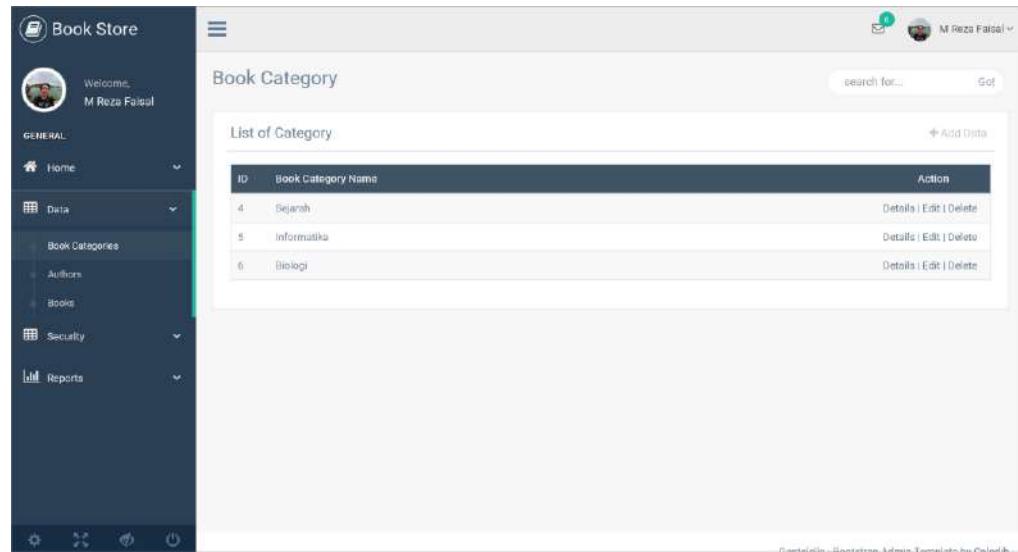
- Delete.cshtml berfungsi untuk menampilkan detail kategori buku yang akan dihapus.
- 2. Mengelola pengarang buku, fitur ini akan memiliki 2 file yaitu:
 - Index.cshtml yang akan menampilkan tabel daftar pengarang buku. Pada halaman ini juga terdapat fitur untuk menghapus data pengarang buku. Pengarang buku hanya bisa dihapus jika belum ada buku yang terkait dengan data pengarang tersebut.
 - Detail.cshtml berfungsi untuk menampilkan data pengarang buku yang dipilih pada halaman daftar pengarang buku.
 - Create.cshtml sebagai form input untuk menambah data pengarang buku.
 - Edit.cshtml untuk form mengedit data pengarang buku.
 - Delete.cshtml berfungsi untuk menampilkan detail pengarang buku yang akan dihapus.
- 3. Mengelola buku
 - Index.cshtml yang akan menampilkan tabel daftar buku. Pada halaman ini juga terdapat fitur untuk menghapus data buku.
 - Detail.cshtml berfungsi untuk menampilkan data buku yang dipilih pada halaman daftar buku.
 - Create.cshtml sebagai form input untuk menambah data buku.
 - Edit.cshtml untuk form mengedit data buku.
 - Delete.cshtml berfungsi untuk menampilkan detail buku yang akan dihapus.
- 4. Mengelola role
 - Index.cshtml berfungsi untuk menampilkan daftar role. Pada halaman ini juga terdapat fitur untuk menghapus role.
 - Detail.cshtml berfungsi untuk menampilkan data role yang dipilih pada halaman daftar role.
 - Create.cshtml sebagai form input untuk menambah role.
 - Edit.cshtml untuk form mengedit role yang dipilih.
 - Delete.cshtml berfungsi untuk menampilkan detail role yang akan dihapus.
- 5. Mengelola user
 - Index.cshtml berfungsi untuk menampilkan daftar user. pada halaman ini juga memiliki fitur untuk menghapus user yang dipilih.
 - Detail.cshtml berfungsi untuk menampilkan data user yang dipilih pada halaman daftar user.
 - Create.cshtml sebagai form input untuk menambah user.
 - Edit.cshtml untuk form mengedit user yang dipilih.
 - Delete.cshtml berfungsi untuk menampilkan detail user yang akan dihapus.

View Categories

File-file berikut ini menangani antarmuka untuk class CategoriesController. Semua file di bawah ini disimpan pada folder Views/Categories.

Index.cshtml

Berikut adalah antarmuka untuk menampilkan daftar kategori buku.



Gambar 148. View Categories - Index.cshtml.

Berikut adalah kode yang digunakan untuk membuat antarmuka tersebut.

```
Index.cshtml
@model IEnumerable<SqlServerBookStore.Models.Category>



### Book Category



Go!



## List of Category



- <a href="#" asp-controller="Categories" asp-action="Create"><i class="fa fa-plus"></i> Add Data</a>



| ID | Book Category Name | Action                                                                  |
|----|--------------------|-------------------------------------------------------------------------|
| 4  | Besarah            | <a href="#">Details</a>   <a href="#">Edit</a>   <a href="#">Delete</a> |
| 5  | Informatika        | <a href="#">Details</a>   <a href="#">Edit</a>   <a href="#">Delete</a> |
| 6  | Biologi            | <a href="#">Details</a>   <a href="#">Edit</a>   <a href="#">Delete</a> |


```

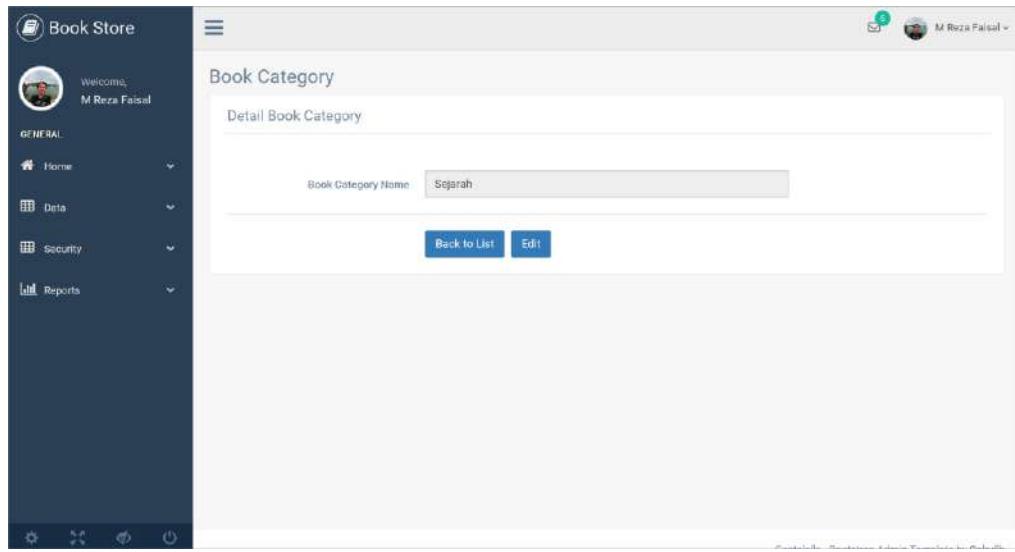
```


| @Html.DisplayNameFor(model => model.FirstOrDefault().CategoryID) | @Html.DisplayNameFor(model => model.FirstOrDefault().Name) | <span class="nobr">Action</span> |
|------------------------------------------------------------------|------------------------------------------------------------|----------------------------------|
|------------------------------------------------------------------|------------------------------------------------------------|----------------------------------|


```

Details.cshtml

Berikut adalah antarmuka untuk menampilkan data kategori buku yang dipilih pada halaman daftar kategori buku.



Gambar 149. View Categories - Details.cshtml.

Berikut adalah kode untuk membuat antarmuka di atas.

```
Details.cshtml
@model SqlServerBookStore.Models.Category



<
    <div class="page-title">
        <div class="title_left">
            <h3>Book Category</h3>
        </div>

        <div class="clearfix"></div>

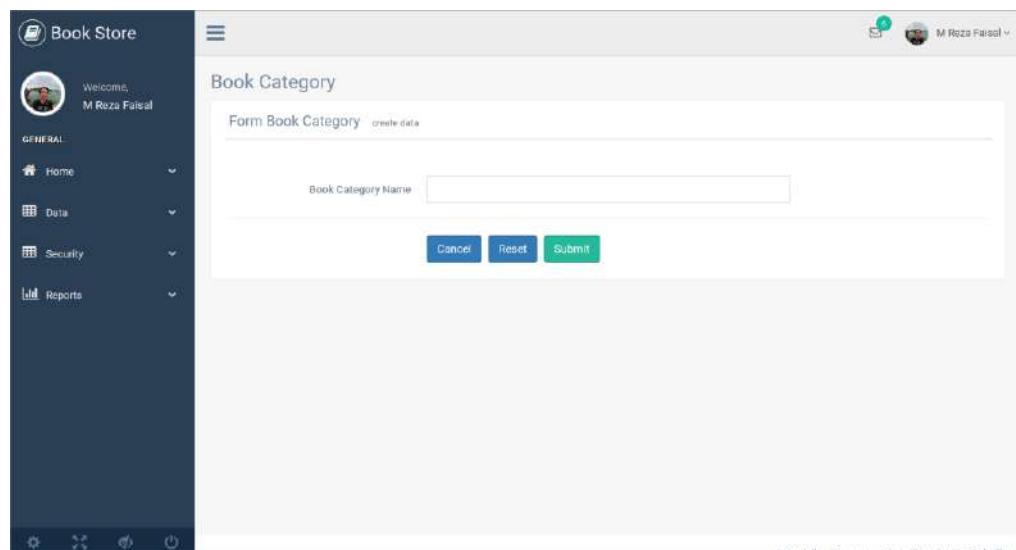
        <div class="row">
            <div class="col-md-12 col-sm-12 col-xs-12">
                <div class="x_panel">
                    <div class="x_title">
                        <h2>Detail Book Category</h2>
                        <div class="clearfix"></div>
                    </div>
                    <div class="x_content">
                        <br />
                        <form asp-controller="Categories" asp-action="Create" data-parsley-validate class="form-horizontal form-label-left">
                            <div asp-validation-summary="All"></div>
                            <div class="form-group">
                                <label asp-for="Name" class="control-label col-md-3 col-sm-3 col-xs-12"></label>
                                <div class="col-md-6 col-sm-6 col-xs-12">
                                    <input asp-for="Name" readonly="" class="form-control col-md-7 col-xs-12" />
                                </div>
                            </div>
                            <div class="ln_solid"></div>
                            <div class="form-group">
                                <div class="col-md-6 col-sm-6 col-xs-12 col-md-offset-3">
                                    <button class="btn btn-primary" type="button" onclick="@Url.Action("Index", "Categories")">Back to List</button>
                                </div>
                            </div>
                        </form>
                    </div>
                </div>
            </div>
        </div>
    </div>


```

```
                                <button class="btn btn-primary" type="button" onclick="location.href='@Url.Action("Edit", "Categories", new { id = @Model.CategoryID})'">Edit</button>
                            </div>
                        </div>
                    </form>
                </div>
            </div>
        </div>
    </div>
</div>
</div>
```

Create.cshtml

Berikut adalah antarmuka untuk input data kategori buku.



Gambar 150. View Categories - Create.cshtml

Berikut adalah kode untuk membuat antarmuka di atas.

```
 Create.cshtml
 @model SqlServerBookStore.Models.Category

<div class="">
    <div class="page-title">
        <div class="title_left">
            <h3>Book Category</h3>
        </div>

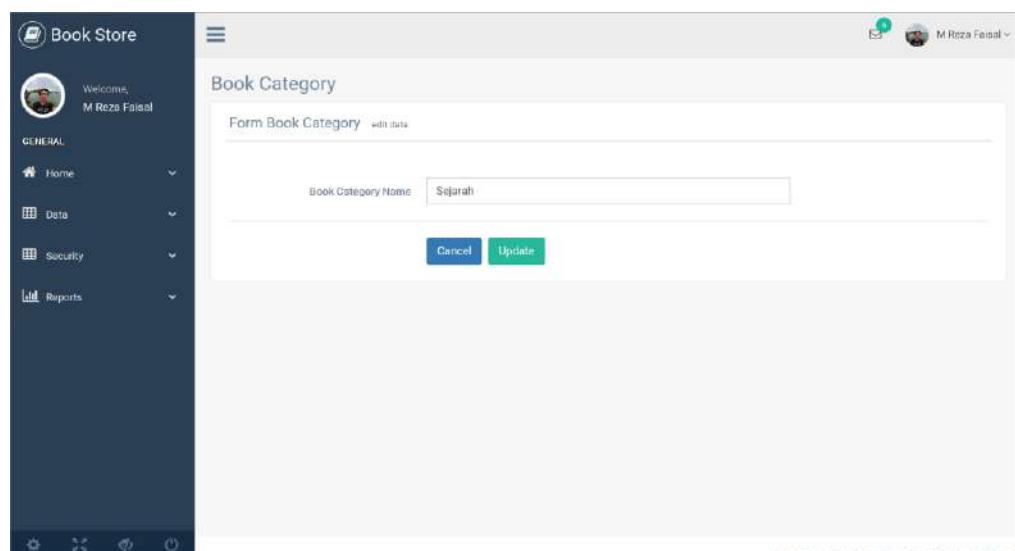
        <div class="clearfix"></div>

        <div class="row">
            <div class="col-md-12 col-sm-12 col-xs-12">
                <div class="x_panel">
                    <div class="x_title">
                        <h2>Form Book Category <small>create data</small></h2>
                    </div>
                    <div class="clearfix"></div>
                </div>
                <div class="x_content">
                    <br />
```

```
<form asp-controller="Categories" asp-
action="Create" data-parsley-validate class="form-horizontal form-label-
left">
    <div asp-validation-summary="All"></div>
    <div class="form-group">
        <label asp-for="Name" class="control-
label col-md-3 col-sm-3 col-xs-12"></label>
        <div class="col-md-6 col-sm-6 col-xs-12">
            <input asp-
for="Name" required="" class="form-control col-md-7 col-xs-12">
            <span asp-validation-for="Name"></span>
        </div>
    </div>
    <div class="ln_solid"></div>
    <div class="form-group">
        <div class="col-md-6 col-sm-6 col-xs-12 col-
md-offset-3">
            <button class="btn btn-
primary" type="button" onclick="@Url.Action("Index", "Categor-
ies")">Cancel</button>
            <button class="btn btn-
primary" type="reset" onclick='document.forms[0].reset();return false;'>Res-
et</button>
            <button type="submit" class="btn btn-
success">Submit</button>
        </div>
    </div>
    </form>
</div>
</div>
</div>
</div>
</div>
```

Edit.cshtml

Berikut ini adalah antarmuka untuk mengedit data kategori buku yang dipilih.



Gambar 151. View Categories - Edit.

Berikut adalah kode untuk membuat antarmuka di atas.

```

Edit.cshtml
@model SqlServerBookStore.Models.Category

<div class="">
    <div class="page-title">
        <div class="title_left">
            <h3>Book Category</h3>
        </div>

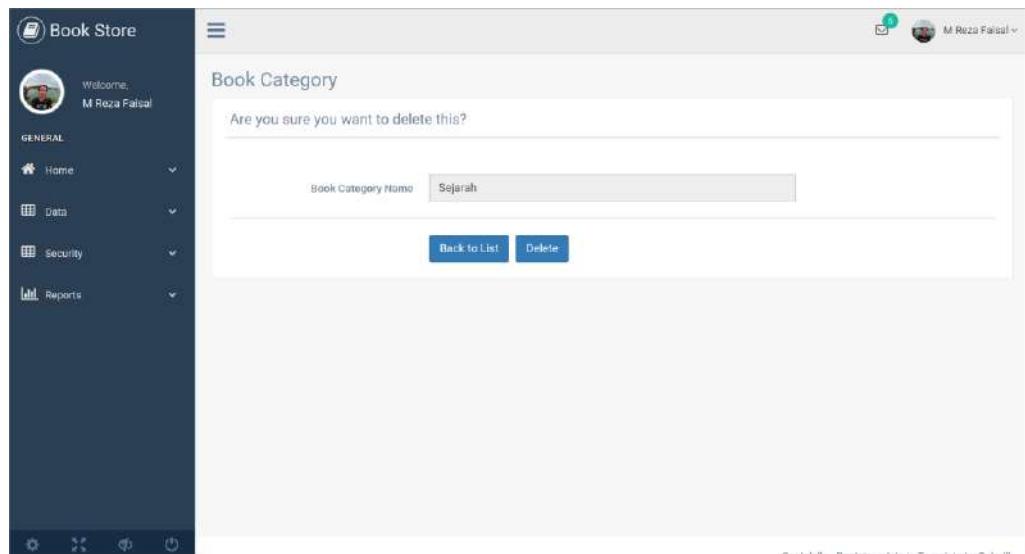
        <div class="clearfix"></div>

        <div class="row">
            <div class="col-md-12 col-sm-12 col-xs-12">
                <div class="x_panel">
                    <div class="x_title">
                        <h2>Form Book Category <small>edit data</small></h2>
                        <div class="clearfix"></div>
                    </div>
                    <div class="x_content">
                        <br />
                        <form asp-controller="Categories" asp-
action="Edit" data-parsley-validate class="form-horizontal form-label-left">
                            <div asp-validation-summary="All"></div>
                            <input type="hidden" asp-for="CategoryID" />
                            <div class="form-group">
                                <label asp-for="Name" class="control-
label col-md-3 col-sm-3 col-xs-12"></label>
                                <div class="col-md-6 col-sm-6 col-xs-12">
                                    <input asp-
for="Name" required="required" class="form-control col-md-7 col-xs-12">
                                    <span asp-validation-for="Name"></span>
                                </div>
                            </div>
                            <div class="ln_solid"></div>
                            <div class="form-group">
                                <div class="col-md-6 col-sm-6 col-xs-12 col-
md-offset-3">
                                    <button class="btn btn-
primary" type="button" onclick="location.href='@Url.Action("Index", "Categor-
ies')'">Cancel</button>
                                    <button type="submit" class="btn btn-
success">Update</button>
                                </div>
                            </div>
                        </div>
                    </div>
                </div>
            </div>
        </div>
    </div>
</div>

```

Delete.cshtml

Berikut adalah antarmuka untuk menampilkan halaman konfirmasi untuk menghapus data kategori buku yang telah dipilih.



Gambar 152. View Categories - Delete.

Berikut adalah kode untuk membuat antarmuka di atas.

```
>Delete.cshtml
@model SqlServerBookStore.Models.Category



<
    <div class="page-title">
        <div class="title_left">
            <h3>Book Category</h3>
        </div>

        <div class="clearfix"></div>

        <div class="row">
            <div class="col-md-12 col-sm-12 col-xs-12">
                <div class="x_panel">
                    <div class="x_title">
                        <h2>Are you sure you want to delete this?</h2>
                        <div class="clearfix"></div>
                    </div>
                    <div class="x_content">
                        <br />
                        <form asp-controller="Categories" asp-
action="Delete" data-parsley-validate class="form-horizontal form-label-
left">
                            <div asp-validation-summary="All"></div>
                            <input type="hidden" asp-for="CategoryID" />
                            <div class="form-group">
                                <label asp-for="Name" class="control-
label col-md-3 col-sm-3 col-xs-12"></label>
                                <div class="col-md-6 col-sm-6 col-xs-12">
                                    <input asp-
for="Name" readonly="" class="form-control col-md-7 col-xs-12">
                                    </div>
                                </div>
                                <div class="ln_solid"></div>
                                <div class="form-group">
                                    <div class="col-md-6 col-sm-6 col-xs-12 col-
md-offset-3">
                                        <button class="btn btn-
primary" type="button" onclick="@Url.Action("Index", "Categori
es")">Back to List</button>
                                    </div>
                                </div>
                            </div>
                        </form>
                    </div>
                </div>
            </div>
        </div>
    </div>


```

```

                <button class="btn btn-primary" type="submit">Delete</button>
            </div>
        </div>
    </form>
</div>
</div>
</div>
</div>
</div>

```

View Authors

File-file berikut ini menangani antarmuka untuk class AuthorsController. Semua file di bawah ini disimpan pada folder Views/Authors.

Index.cshtml

Berikut adalah antarmuka untuk menampilkan daftar pengarang buku.

ID	Author's Name	Email	Action
1	Reza	reza@faisal.com	Details Edit Delete
2	Faisal	faisal@reza.com	Details Edit Delete

Gambar 153. View Authors - Index.cshtml.

Berikut adalah kode yang digunakan untuk membuat antarmuka tersebut.

```

Index.cshtml
@model IEnumerable<SqlServerBookStore.Models.Author>

<div class="">
    <div class="page-title">
        <div class="title_left">
            <h3>Author</h3>
        </div>
        <div class="title_right">
            <div class="col-md-5 col-sm-5 col-xs-12 form-group pull-right top_search">
                <div class="input-group">
                    <input type="text" class="form-control" placeholder="search for...">
                    <span class="input-group-btn">

```

```

        <button class="btn btn-default" type="button">Go!</button>
            </span>
        </div>
    </div>
</div>

<div class="clearfix"></div>

<div class="row">
    <div class="col-md-12 col-sm-12 col-xs-12">
        <div class="x_panel">
            <div class="x_title">
                <h2>List of Author</h2>
                <ul class="nav navbar-right panel_toolbox">
                    <li><a asp-controller="Authors" asp-action="Create"><i class="fa fa-plus"></i> Add Data</a></li>
                </ul>
                <div class="clearfix"></div>
            </div>

            <div class="x_content">
                <div class="table-responsive">
                    <table class="table table-striped jambo_table bulk_action">
                        <thead>
                            <tr class="headings">
                                <th class="column-title" style="width: 5%">@Html.DisplayNameFor(model => model.AuthorID)</th>
                                <th class="column-title" style="width: 40%">@Html.DisplayNameFor(model => model.Name)</th>
                                <th class="column-title" style="width: 40%">@Html.DisplayNameFor(model => model.Email)</th>
                                <th class="column-title no-link last" style="width: 15%; text-align:center"><span class="nobr">Action</span></th>
                            </tr>
                        </thead>

                        <tbody>
                            @{
                                var odd = false;
                            }
                            @foreach (var item in Model)
                            {
                                <tr class="@odd ? "odd": "even" pointer">
                                    <td class=" ">@item.AuthorID</td>
                                    <td class=" ">@item.Name</td>
                                    <td class=" ">@item.Email</td>
                                    <td class=" last">
                                        <a asp-controller="Authors" asp-action="Details" asp-route-id="@item.AuthorID">Details</a> |
                                        <a asp-controller="Authors" asp-action="Edit" asp-route-id="@item.AuthorID">Edit</a> |
                                        <a asp-controller="Authors" asp-action="Delete" asp-route-id="@item.AuthorID">Delete</a>
                                    </td>
                                </tr>
                                odd = !odd;
                            }
                        </tbody>
                    </table>
    </div>

```

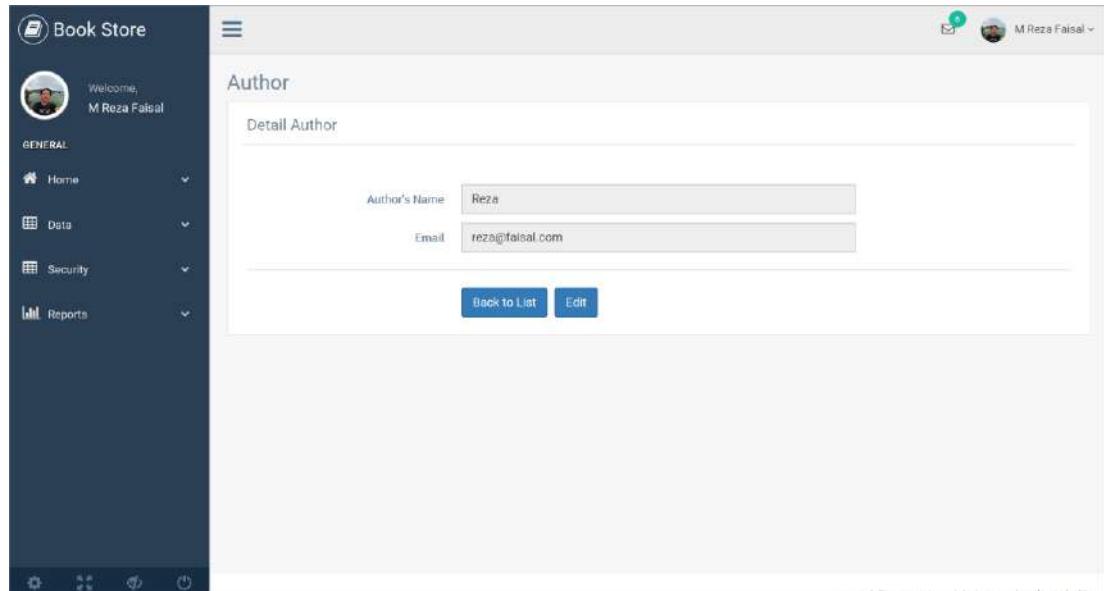
```

        </div>
    </div>
</div>
</div>
</div>

```

Details.cshtml

Berikut adalah antarmuka untuk menampilkan data pengarang buku yang dipilih pada halaman daftar pengarang buku.



Gambar 154. View Authors - Details.cshtml.

Berikut adalah kode untuk membuat antarmuka di atas.

```

Details.cshtml
@model SqlServerBookStore.Models.Author

<div class="">
    <div class="page-title">
        <div class="title_left">
            <h3>Author</h3>
        </div>

        <div class="clearfix"></div>

        <div class="row">
            <div class="col-md-12 col-sm-12 col-xs-12">
                <div class="x_panel">
                    <div class="x_title">
                        <h2>Detail Author</h2>
                        <div class="clearfix"></div>
                    </div>
                    <div class="x_content">
                        <br />
                        <form asp-controller="Authors" asp-
action="Create" data-parsley-validate class="form-horizontal form-label-
left">
                            <div asp-validation-summary="All"></div>
                            <div class="form-group">

```

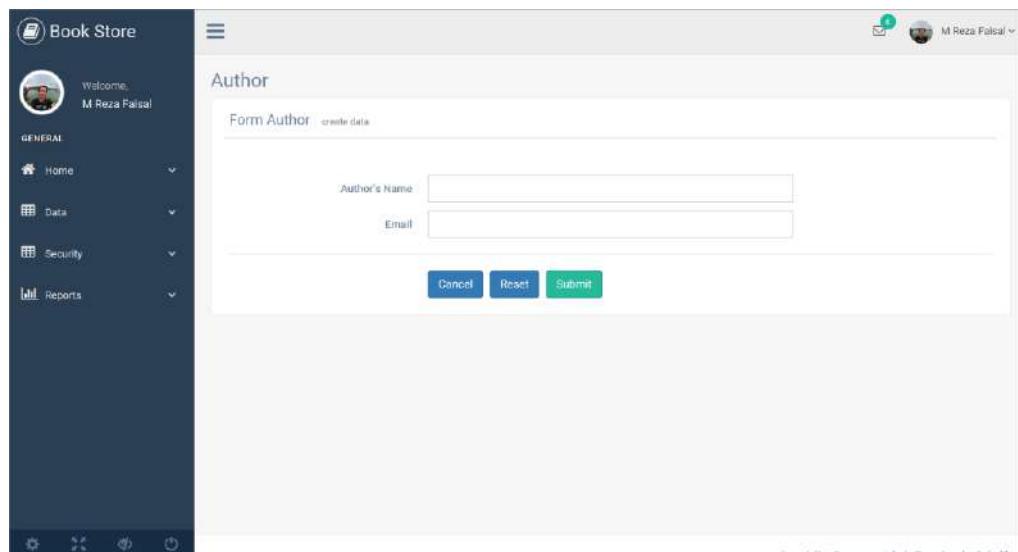
```

        <label asp-for="Name" class="control-
label col-md-3 col-sm-3 col-xs-12"></label>
            <div class="col-md-6 col-sm-6 col-xs-12">
                <input asp-
for="Name" readonly="readonly" class="form-control col-md-7 col-xs-12">
            </div>
        </div>
        <div class="form-group">
            <label asp-for="Email" class="control-
label col-md-3 col-sm-3 col-xs-12"></label>
                <div class="col-md-6 col-sm-6 col-xs-12">
                    <input asp-
for="Email" readonly="readonly" class="form-control col-md-7 col-xs-12">
                </div>
            </div>
            <div class="ln_solid"></div>
            <div class="form-group">
                <div class="col-md-6 col-sm-6 col-xs-12 col-
md-offset-3">
                    <button class="btn btn-
primary" type="button" onclick="location.href='@Url.Action("Index", "Authors
")'">Back to List</button>
                    <button class="btn btn-
primary" type="button" onclick="location.href='@Url.Action("Edit", "Authors
", new { id = @Model.AuthorID})'">Edit</button>
                </div>
            </div>
        </div>
    </div>
</div>
</div>

```

Create.cshtml

Berikut adalah antarmuka untuk input data pengarang buku.



Gambar 155. View Authors - Create.cshtml

Berikut adalah kode untuk membuat antarmuka di atas.

Create.cshtml

```
@model SqlServerBookStore.Models.Author

<div class="">
    <div class="page-title">
        <div class="title_left">
            <h3>Author</h3>
        </div>

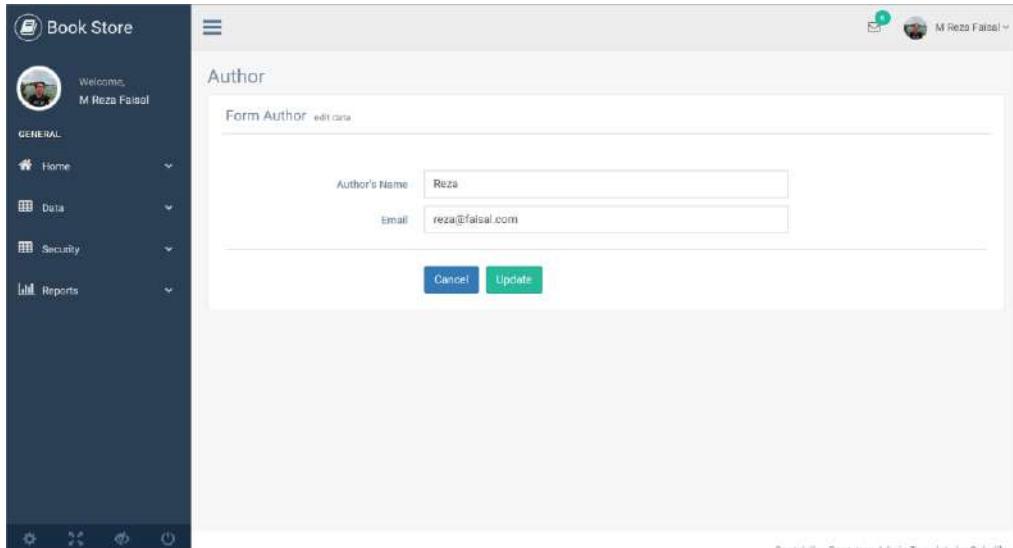
        <div class="clearfix"></div>

        <div class="row">
            <div class="col-md-12 col-sm-12 col-xs-12">
                <div class="x_panel">
                    <div class="x_title">
                        <h2>Form Author <small>create data</small></h2>
                        <div class="clearfix"></div>
                    </div>
                    <div class="x_content">
                        <br />
                        <form asp-
controller="Authors" method="POST" enctype="multipart/form-data" asp-
action="Create" data-parsley-validate class="form-horizontal form-label-
left">
                            <div asp-validation-summary="All"></div>
                            <div class="form-group">
                                <label asp-for="Name" class="control-
label col-md-3 col-sm-3 col-xs-12"></label>
                                <div class="col-md-6 col-sm-6 col-xs-12">
                                    <input asp-
for="Name" required="required" class="form-control col-md-7 col-xs-12">
                                    <span asp-validation-for="Name"></span>
                                </div>
                            </div>
                            <div class="form-group">
                                <label asp-for="Email" class="control-
label col-md-3 col-sm-3 col-xs-12"></label>
                                <div class="col-md-6 col-sm-6 col-xs-12">
                                    <input asp-
for="Email" required="required" class="form-control col-md-7 col-xs-12">
                                    <span asp-validation-for="Email"></span>
                                </div>
                            </div>
                            <div class="ln_solid"></div>
                            <div class="form-group">
                                <div class="col-md-6 col-sm-6 col-xs-12 col-
md-offset-3">
                                    <button class="btn btn-
primary" type="button" onclick="location.href='@Url.Action("Index", "Authors")'">Cancel</button>
                                    <button class="btn btn-
primary" type="reset" onclick='document.forms[0].reset();return false;'>Rese
t</button>
                                    <button type="submit" class="btn btn-
success">Submit</button>
                                </div>
                            </div>
                        </form>
                    </div>
                </div>
            </div>
        </div>
    </div>
```

```
</div>
</div>
</div>
```

Edit.cshtml

Berikut ini adalah antarmuka untuk mengedit data pengarang buku yang dipilih.



Gambar 156. View Authors - Edit.

Berikut adalah kode untuk membuat antarmuka di atas.

```
>Edit.cshtml
@model SqlServerBookStore.Models.Author



### Author



## Form Authoredit data


```

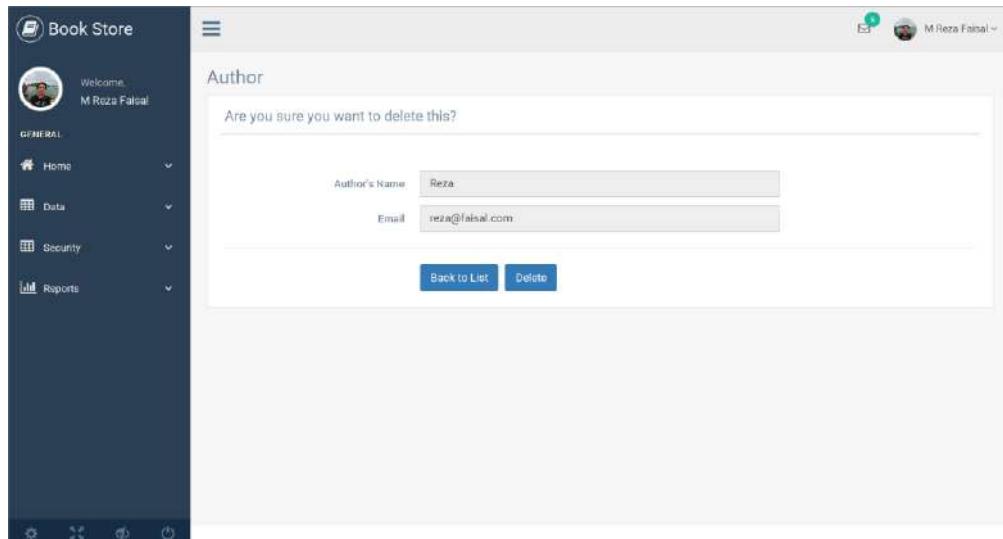
```

        <input asp-
for="Name" required="required" class="form-control col-md-7 col-xs-12">
            <span asp-validation-for="Name"></span>
        </div>
    </div>
    <div class="form-group">
        <label asp-for="Email" class="control-
label col-md-3 col-sm-3 col-xs-12"></label>
        <div class="col-md-6 col-sm-6 col-xs-12">
            <input asp-
for="Email" required="required" class="form-control col-md-7 col-xs-12">
                <span asp-validation-for="Email"></span>
            </div>
        </div>
        <div class="ln_solid"></div>
        <div class="form-group">
            <div class="col-md-6 col-sm-6 col-xs-12 col-
md-offset-3">
                <button class="btn btn-
primary" type="button" onclick="location.href='@Url.Action("Index", "Authors
")'">Cancel</button>
                <button type="submit" class="btn btn-
success">Update</button>
            </div>
        </div>
    </div>
</div>
</div>
</div>
</div>
</div>
</div>

```

Delete.cshtml

Berikut adalah antarmuka untuk menampilkan halaman konfirmasi untuk menghapus data pengarang buku yang telah dipilih.



Gambar 157. View Authors - Delete.

Berikut adalah kode untuk membuat antarmuka di atas.

```
Delete.cshtml
@model SqlServerBookStore.Models.Author



<div class="page-title">
        <div class="title_left">
            <h3>Author</h3>
        </div>

        <div class="clearfix"></div>

        <div class="row">
            <div class="col-md-12 col-sm-12 col-xs-12">
                <div class="x_panel">
                    <div class="x_title">
                        <h2>Are you sure you want to delete this?</h2>
                        <div class="clearfix"></div>
                    </div>
                    <div class="x_content">
                        <br />
                        <form asp-controller="Authors" asp-
action="Delete" data-parsley-validate class="form-horizontal form-label-
left">
                            <div asp-validation-summary="All"></div>
                            <input type="hidden" asp-for="AuthorID" />
                            <div class="form-group">
                                <label asp-for="Name" class="control-
label col-md-3 col-sm-3 col-xs-12"></label>
                                <div class="col-md-6 col-sm-6 col-xs-12">
                                    <input asp-
for="Name" readonly="readonly" class="form-control col-md-7 col-xs-12">
                                </div>
                            </div>
                            <div class="form-group">
                                <label asp-for="Email" class="control-
label col-md-3 col-sm-3 col-xs-12"></label>
                                <div class="col-md-6 col-sm-6 col-xs-12">
                                    <input asp-
for="Email" readonly="readonly" class="form-control col-md-7 col-xs-12">
                                </div>
                            </div>
                            <div class="ln_solid"></div>
                            <div class="form-group">
                                <div class="col-md-6 col-sm-6 col-xs-12 col-
md-offset-3">
                                    <button class="btn btn-
primary" type="button" onclick="location.href='@Url.Action("Index", "Authors")'">Back to List</button>
                                    <button class="btn btn-
primary" type="submit">Delete</button>
                                </div>
                            </div>
                        </form>
                    </div>
                </div>
            </div>
        </div>
    </div>


```

View Books

File-file berikut ini menangani antarmuka untuk class BooksController. Semua file di bawah ini disimpan pada folder Views/Books.

Index.cshtml

Berikut adalah antarmuka untuk menampilkan daftar buku.

Cover	Category	Title	Publish Date	Qty	Action
	Sejarah	Sejarah Jawa ISBN: 1233 Authors: Reza, Faisal Price: 12	12/12/2012 12:00:00 AM	12	Details Edit Delete
	Informatika	Bioinformatika ISBN: 2545 Authors: Reza, Faisal Price: 4	4/4/2004 12:00:00 AM	4	Details Edit Delete

Gambar 158. View Books - Index.cshtml.

Berikut adalah kode yang digunakan untuk membuat antarmuka tersebut.

```
Index.cshtml
@model IList<SqlServerBookStore.Models.BookViewModel>



261



www.perpustakaanebook.com


```

```

        <h2>List of Book</h2>
        <ul class="nav navbar-right panel_toolbox">
            <li><a asp-controller="Books" asp-
action="Create"><i class="fa fa-plus"></i> Add Data</a></li>
        </ul>
        <div class="clearfix"></div>
    </div>

    <div class="x_content">
        <div class="table-responsive">
            <table class="table table-
striped jambo_table bulk_action">
                <thead>
                    <tr class="headings">
                        <th class="column-
title" style="width: 10%">Cover</th>
                        <th class="column-
title" style="width: 10%">Category</th>
                        <th class="column-
title" style="width: 35%">Title</th>
                        <th class="column-
title" style="width: 20%">Publish Date</th>
                        <th class="column-
title" style="width: 10%">Qty</th>
                        <th class="column-title no-
link last" style="width: 15%"><span class="nobr">Action</span></th>
                    </tr>
                </thead>

                <tbody>
                    @{
                        var odd = false;
}
@foreach (var item in Model)
{
    <tr class="@((odd ? "odd": "even")) pointe
r">
        <td class=" " ></td>
        <td class=" " >@item.CategoryName</td>
        <td class=" " >
            @item.Title<br />
            ISBN: @item.ISBN<br />
            Authors: @item.AuthorNames<br />
            Price: @item.Price<br />
        </td>
        <td class=" " >@item.PublishDate</td>
        <td class=" " >@item.Quantity</td>
        <td class=" last">
            <a asp-controller="Books" asp-
action="Details" asp-route-id="@item.ISBN">Details</a> |
            <a asp-controller="Books" asp-
action="Edit" asp-route-id="@item.ISBN">Edit</a> |
            <a asp-controller="Books" asp-
action="Delete" asp-route-id="@item.ISBN">Delete</a>
        </td>
    </tr>
    odd = !odd;
}
    </tbody>
</table>

```

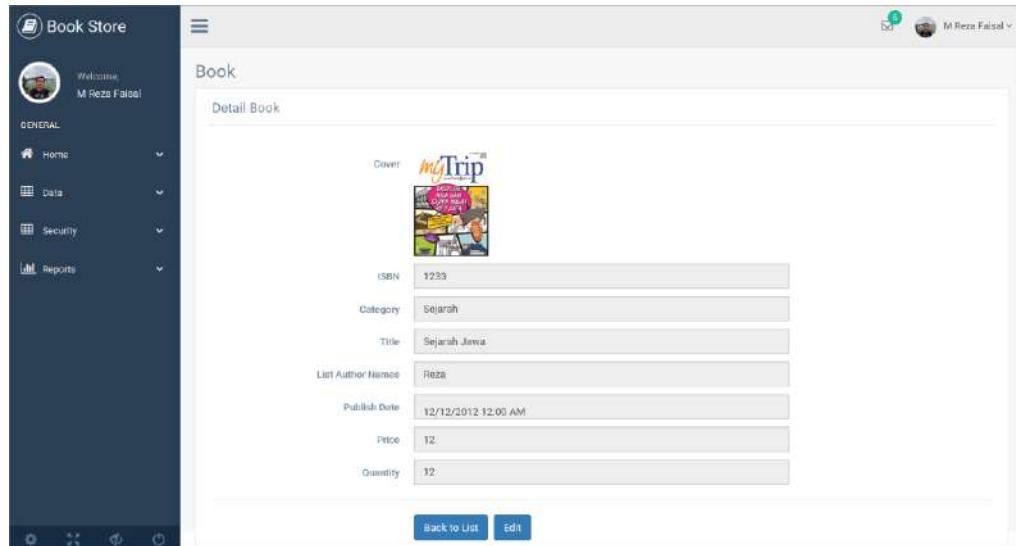
```

        </div>
    </div>
    </div>
</div>
</div>

```

Details.cshtml

Berikut adalah antarmuka untuk menampilkan data buku yang dipilih pada halaman daftar buku.



Gambar 159. View Books - Details.cshtml.

Berikut adalah kode untuk membuat antarmuka di atas.

```

Details.cshtml
@model SqlServerBookStore.Models.BookViewModel

<div class="">
    <div class="page-title">
        <div class="title_left">
            <h3>Book</h3>
        </div>

        <div class="clearfix"></div>

        <div class="row">
            <div class="col-md-12 col-sm-12 col-xs-12">
                <div class="x_panel">
                    <div class="x_title">
                        <h2>Detail Book</h2>
                        <div class="clearfix"></div>
                    </div>
                    <div class="x_content">
                        <br />
                        <form data-parsley-validate class="form-horizontal form-label-left">
                            <div class="form-group">
                                <label class="control-label col-md-3 col-sm-3 col-xs-12">Cover</label>
                                <div class="col-md-6 col-sm-6 col-xs-12">

```

```

        
    </div>
</div>
<div class="form-group">
    <label asp-for="ISBN" class="control-
label col-md-3 col-sm-3 col-xs-12"></label>
    <div class="col-md-6 col-sm-6 col-xs-12">
        <input asp-
for="ISBN" readonly="readonly" class="form-control col-md-7 col-xs-12">
    </div>
</div>
<div class="form-group">
    <label asp-
for="CategoryName" class="control-label col-md-3 col-sm-3 col-xs-
12"></label>
    <div class="col-md-6 col-sm-6 col-xs-12">
        <input asp-
for="CategoryName" readonly="readonly" class="form-control col-md-7 col-xs-
12">
    </div>
</div>
<div class="form-group">
    <label asp-for="Title" class="control-
label col-md-3 col-sm-3 col-xs-12"></label>
    <div class="col-md-6 col-sm-6 col-xs-12">
        <input asp-
for="Title" readonly="readonly" class="form-control col-md-7 col-xs-12">
    </div>
</div>
<div class="form-group">
    <label asp-for="AuthorNames" class="control-
label col-md-3 col-sm-3 col-xs-12"></label>
    <div class="col-md-6 col-sm-6 col-xs-12">
        <input asp-
for="AuthorNames" readonly="readonly" class="form-control col-md-7 col-xs-
12">
    </div>
</div>
<div class="form-group">
    <label asp-for="PublishDate" class="control-
label col-md-3 col-sm-3 col-xs-12"></label>
    <div class="col-md-6 col-sm-6 col-xs-12">
        <input asp-
for="PublishDate" readonly="readonly" class="form-control col-md-7 col-xs-
12">
    </div>
</div>
<div class="form-group">
    <label asp-for="Price" class="control-
label col-md-3 col-sm-3 col-xs-12"></label>
    <div class="col-md-6 col-sm-6 col-xs-12">
        <input asp-
for="Price" readonly="readonly" class="form-control col-md-7 col-xs-12">
    </div>
</div>
<div class="form-group">
    <label asp-for="Quantity" class="control-
label col-md-3 col-sm-3 col-xs-12"></label>
    <div class="col-md-6 col-sm-6 col-xs-12">

```

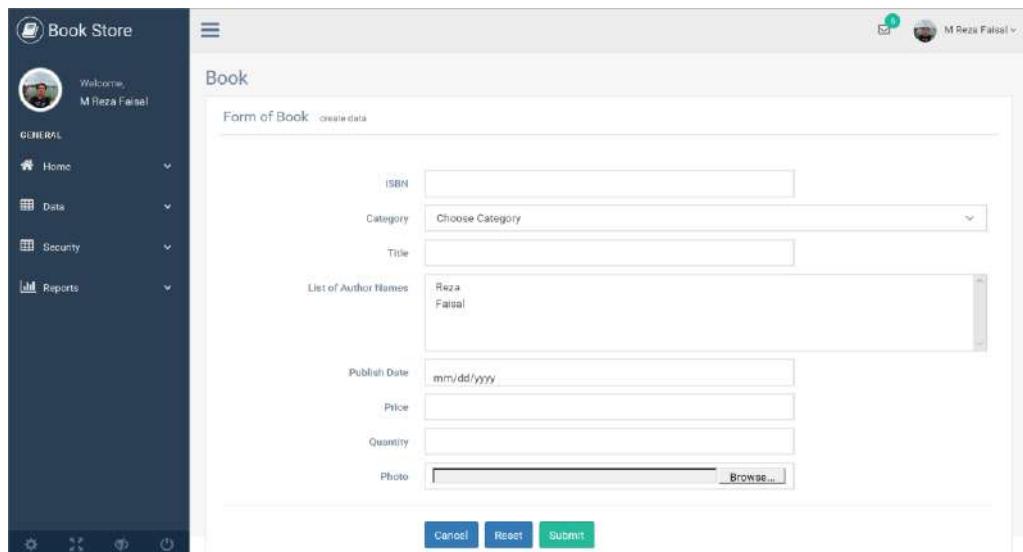
```

                <input asp-
for="Quantity" readonly="readonly" class="form-control col-md-7 col-xs-12">
            </div>
        </div>
        <div class="ln_solid"></div>
        <div class="form-group">
            <div class="col-md-6 col-sm-6 col-xs-12 col-
md-offset-3">
                <button class="btn btn-
primary" type="button" onclick="location.href='@Url.Action("Index", "Books")'">Back to List</button>
                <button class="btn btn-
primary" type="button" onclick="location.href='@Url.Action("Edit", "Books", new { id = @Model.ISBN})'">Edit</button>
            </div>
        </div>
    </div>
</div>
</div>
</div>

```

Create.cshtml

Berikut adalah antarmuka untuk input data buku.



Gambar 160. View Books - Create.cshtml

Berikut adalah kode untuk membuat antarmuka di atas.

```

Create.cshtml
@model SqlServerBookStore.Models.BookFormViewModel

<div class="">
    <div class="page-title">
        <div class="title_left">
            <h3>Book</h3>
        </div>
        <div class="title_right">

```

```

<div class="row">
    <div class="col-md-12 col-sm-12 col-xs-12">
        <div class="x_panel">
            <div class="x_title">
                <h2>Form of Book <small>create data</small></h2>
                <div class="clearfix"></div>
            </div>
            <div class="x_content">
                <br />
                <form method="POST" enctype="multipart/form-data" asp-controller="Books" asp-action="Create" data-parsley-validate class="form-horizontal form-label-left">
                    <div asp-validation-summary="All"></div>
                    <div class="form-group">
                        <label asp-for="ISBN" class="control-label col-md-3 col-sm-3 col-xs-12"></label>
                        <div class="col-md-6 col-sm-6 col-xs-12">
                            <input asp-for="ISBN" required="" class="form-control col-md-7 col-xs-12" type="text" />
                            <span asp-validation-for="ISBN" class="text-danger"></span>
                        </div>
                    </div>
                    <div class="form-group">
                        <label asp-for="CategoryID" class="control-label col-md-3 col-sm-3 col-xs-12"></label>
                        <div class="col-md-9 col-sm-9 col-xs-12">
                            <select asp-for="CategoryID" asp-items="@ViewBag.Categories" class="form-control">
                                <option>Choose Category</option>
                                </select>
                                <span asp-validation-for="CategoryID" class="text-danger"></span>
                        </div>
                    </div>
                    <div class="form-group">
                        <label asp-for="Title" class="control-label col-md-3 col-sm-3 col-xs-12"></label>
                        <div class="col-md-6 col-sm-6 col-xs-12">
                            <input asp-for="Title" required="" class="form-control col-md-7 col-xs-12" type="text" />
                            <span asp-validation-for="Title" class="text-danger"></span>
                        </div>
                    </div>
                    <div class="form-group">
                        <label asp-for="AuthorIDs" class="control-label col-md-3 col-sm-3 col-xs-12"></label>
                        <div class="col-md-9 col-sm-9 col-xs-12">
                            <select asp-for="AuthorIDs" asp-items="@ViewBag.Authors" class="select2_multiple form-control" multiple="multiple"></select>
                            <span asp-validation-for="AuthorIDs" class="text-danger"></span>
                        </div>
                    </div>
                    <div class="form-group">
                        <label asp-for="PublishDate" class="control-label col-md-3 col-sm-3 col-xs-12"></label>
                        <div class="col-md-6 col-sm-6 col-xs-12">
                            <input asp-for="PublishDate" required="" class="form-control col-md-7 col-xs-12" type="text" />
                            <span asp-validation-for="PublishDate" class="text-danger"></span>
                        </div>
                    </div>
                </form>
            </div>
        </div>
    </div>
</div>

```

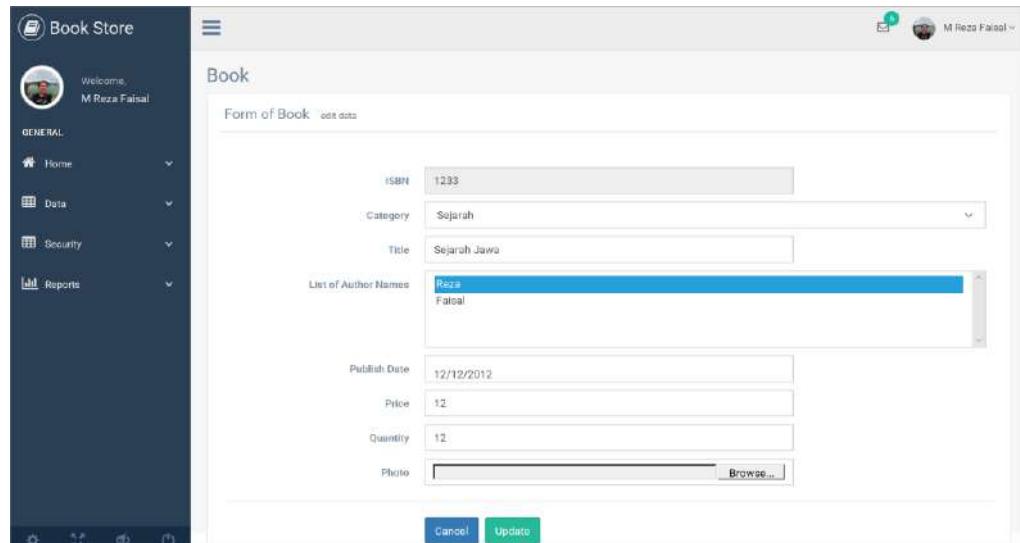
```

        <span asp-validation-
for="PublishDate"></span>
        </div>
    </div>
    <div class="form-group">
        <label asp-for="Price" class="control-
label col-md-3 col-sm-3 col-xs-12"></label>
        <div class="col-md-6 col-sm-6 col-xs-12">
            <input asp-
for="Price" required="required" class="form-control col-md-7 col-xs-12">
            <span asp-validation-for="Price"></span>
        </div>
    </div>
    <div class="form-group">
        <label asp-for="Quantity" class="control-
label col-md-3 col-sm-3 col-xs-12"></label>
        <div class="col-md-6 col-sm-6 col-xs-12">
            <input asp-
for="Quantity" required="required" class="form-control col-md-7 col-xs-12">
            <span asp-validation-
for="Quantity"></span>
        </div>
    </div>
    <div class="form-group">
        <label asp-for="Photo" class="control-
label col-md-3 col-sm-3 col-xs-12"></label>
        <div class="col-md-6 col-sm-6 col-xs-12">
            <input asp-
for="Photo" type="file" class="form-control col-md-7 col-xs-12">
        </div>
    </div>
    <div class="ln_solid"></div>
    <div class="form-group">
        <div class="col-md-6 col-sm-6 col-xs-12 col-
md-offset-3">
            <button class="btn btn-
primary" type="button" onclick="@Url.Action("Index", "Books")
">Cancel</button>
            <button class="btn btn-
primary" type="reset" onclick='document.forms[0].reset();return false;'>Res-
et</button>
            <button type="submit" class="btn btn-
success">Submit</button>
        </div>
    </div>
    </form>
    </div>
    </div>
    </div>
    </div>
    </div>

```

Edit.cshtml

Berikut ini adalah antarmuka untuk mengedit data buku yang dipilih.



Gambar 161. View Books - Edit.

Berikut adalah kode untuk membuat antarmuka di atas.

```

Edit.cshtml
@model SqlServerBookStore.Models.BookFormViewModel

<div class="">
    <div class="page-title">
        <div class="title_left">
            <h3>Book</h3>
        </div>

        <div class="clearfix"></div>

        <div class="row">
            <div class="col-md-12 col-sm-12 col-xs-12">
                <div class="x_panel">
                    <div class="x_title">
                        <h2>Form of Book <small>edit data</small></h2>
                        <div class="clearfix"></div>
                    </div>
                    <div class="x_content">
                        <br />
                        <form method="POST" enctype="multipart/form-data" asp-controller="Books" asp-action="Edit" data-parsley-validate class="form-horizontal form-label-left">
                            <div asp-validation-summary="All"></div>
                            <input type="hidden" asp-for="ISBN" />
                            <div class="form-group">
                                <label asp-for="ISBN" class="control-label col-md-3 col-sm-3 col-xs-12"></label>
                                <div class="col-md-6 col-sm-6 col-xs-12">
                                    <input asp-for="ISBN" readonly="readonly" required="required" class="form-control col-md-7 col-xs-12">
                                    <span asp-validation-for="ISBN"></span>
                                </div>
                            </div>
                            <div class="form-group">
                                <label asp-for="CategoryID" class="control-label col-md-3 col-sm-3 col-xs-12"></label>
                                <div class="col-md-9 col-sm-9 col-xs-12">

```

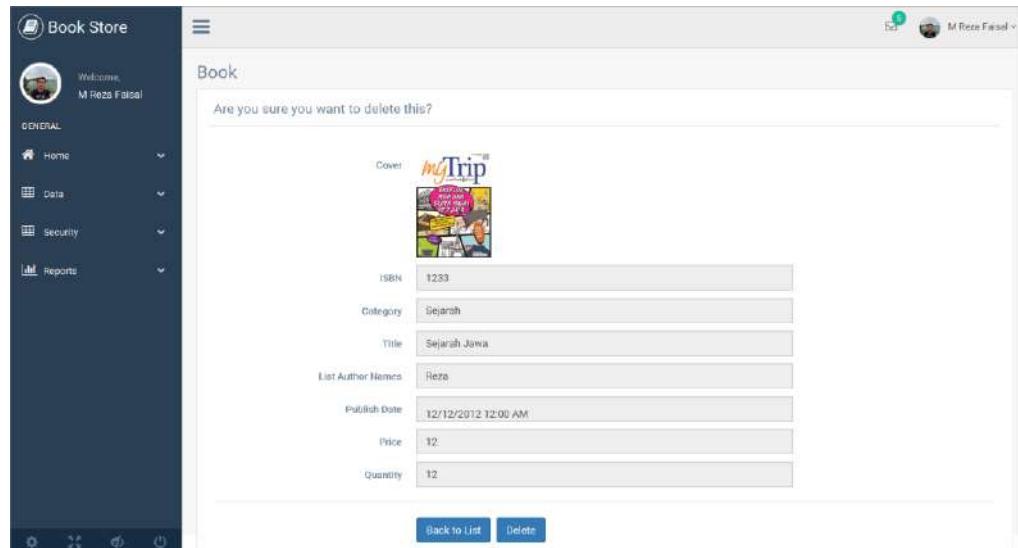
```

        <select asp-for="CategoryID" asp-
items="@ViewBag.Categories" class="form-control">
            <option>Choose Category</option>
        </select>
        <span asp-validation-
for="CategoryID"></span>
    </div>
</div>
<div class="form-group">
    <label asp-for="Title" class="control-
label col-md-3 col-sm-3 col-xs-12"></label>
        <div class="col-md-6 col-sm-6 col-xs-12">
            <input asp-
for="Title" required="required" class="form-control col-md-7 col-xs-12">
            <span asp-validation-for="Title"></span>
        </div>
    </div>
    <div class="form-group">
        <label asp-for="AuthorIDs" class="control-
label col-md-3 col-sm-3 col-xs-12"></label>
        <div class="col-md-9 col-sm-9 col-xs-12">
            <select asp-for="AuthorIDs" asp-
items="@ViewBag.Authors" class="select2_multiple form-
control" multiple="multiple"></select>
            <span asp-validation-
for="AuthorIDs"></span>
        </div>
    </div>
    <div class="form-group">
        <label asp-for="PublishDate" class="control-
label col-md-3 col-sm-3 col-xs-12"></label>
        <div class="col-md-6 col-sm-6 col-xs-12">
            <input asp-
for="PublishDate" required="required" class="form-control col-md-7 col-xs-
12">
            <span asp-validation-
for="PublishDate"></span>
        </div>
    </div>
    <div class="form-group">
        <label asp-for="Price" class="control-
label col-md-3 col-sm-3 col-xs-12"></label>
        <div class="col-md-6 col-sm-6 col-xs-12">
            <input asp-
for="Price" required="required" class="form-control col-md-7 col-xs-12">
            <span asp-validation-for="Price"></span>
        </div>
    </div>
    <div class="form-group">
        <label asp-for="Quantity" class="control-
label col-md-3 col-sm-3 col-xs-12"></label>
        <div class="col-md-6 col-sm-6 col-xs-12">
            <input asp-
for="Quantity" required="required" class="form-control col-md-7 col-xs-12">
            <span asp-validation-
for="Quantity"></span>
        </div>
    </div>
    <div class="form-group">
        <label asp-for="Photo" class="control-
label col-md-3 col-sm-3 col-xs-12"></label>

```

Delete.cshtml

Berikut adalah antarmuka untuk menampilkan halaman konfirmasi untuk menghapus data buku yang telah dipilih.



Gambar 162. View Books - Delete.

Berikut adalah kode untuk membuat antarmuka di atas.

```
    <div class="">
        <div class="page-title">
            <div class="title_left">
                <h3>Book</h3>
            </div>
        </div>
        <div class="clearfix"></div>
```

```

<div class="row">
    <div class="col-md-12 col-sm-12 col-xs-12">
        <div class="x_panel">
            <div class="x_title">
                <h2>Are you sure you want to delete this?</h2>
                <div class="clearfix"></div>
            </div>
            <div class="x_content">
                <br />
                <form asp-controller="Books" asp-
action="Delete" data-parsley-validate class="form-horizontal form-label-
left">
                    <input type="hidden" asp-for="ISBN" />
                    <div class="form-group">
                        <label class="control-label col-md-3 col-sm-
3 col-xs-12">Cover</label>
                        <div class="col-md-6 col-sm-6 col-xs-12">
                            
                        </div>
                    </div>
                    <div class="form-group">
                        <label asp-for="ISBN" class="control-
label col-md-3 col-sm-3 col-xs-12"></label>
                        <div class="col-md-6 col-sm-6 col-xs-12">
                            <input asp-
for="ISBN" readonly="readonly" class="form-control col-md-7 col-xs-12" />
                        </div>
                    </div>
                    <div class="form-group">
                        <label asp-
for="CategoryName" class="control-label col-md-3 col-sm-3 col-xs-
12"></label>
                        <div class="col-md-6 col-sm-6 col-xs-12">
                            <input asp-
for="CategoryName" readonly="readonly" class="form-control col-md-7 col-xs-
12" />
                        </div>
                    </div>
                    <div class="form-group">
                        <label asp-for="Title" class="control-
label col-md-3 col-sm-3 col-xs-12"></label>
                        <div class="col-md-6 col-sm-6 col-xs-12">
                            <input asp-
for="Title" readonly="readonly" class="form-control col-md-7 col-xs-12" />
                        </div>
                    </div>
                    <div class="form-group">
                        <label asp-for="AuthorNames" class="control-
label col-md-3 col-sm-3 col-xs-12"></label>
                        <div class="col-md-6 col-sm-6 col-xs-12">
                            <input asp-
for="AuthorNames" readonly="readonly" class="form-control col-md-7 col-xs-
12" />
                        </div>
                    </div>
                    <div class="form-group">
                        <label asp-for="PublishDate" class="control-
label col-md-3 col-sm-3 col-xs-12"></label>
                        <div class="col-md-6 col-sm-6 col-xs-12">

```

```

        <input asp-
for="PublishDate" readonly="readonly" class="form-control col-md-7 col-xs-
12">
            </div>
        </div>
        <div class="form-group">
            <label asp-for="Price" class="control-
label col-md-3 col-sm-3 col-xs-12"></label>
            <div class="col-md-6 col-sm-6 col-xs-12">
                <input asp-
for="Price" readonly="readonly" class="form-control col-md-7 col-xs-12">
            </div>
        </div>
        <div class="form-group">
            <label asp-for="Quantity" class="control-
label col-md-3 col-sm-3 col-xs-12"></label>
            <div class="col-md-6 col-sm-6 col-xs-12">
                <input asp-
for="Quantity" readonly="readonly" class="form-control col-md-7 col-xs-12">
            </div>
        </div>
        <div class="ln_solid"></div>
        <div class="form-group">
            <div class="col-md-6 col-sm-6 col-xs-12 col-
md-offset-3">
                <button class="btn btn-
primary" type="button" onclick="location.href='@Url.Action("Index", "Books")'">Back to List</button>
                <button class="btn btn-
primary" type="submit">Delete</button>
            </div>
        </div>
    </div>
</div>
</div>

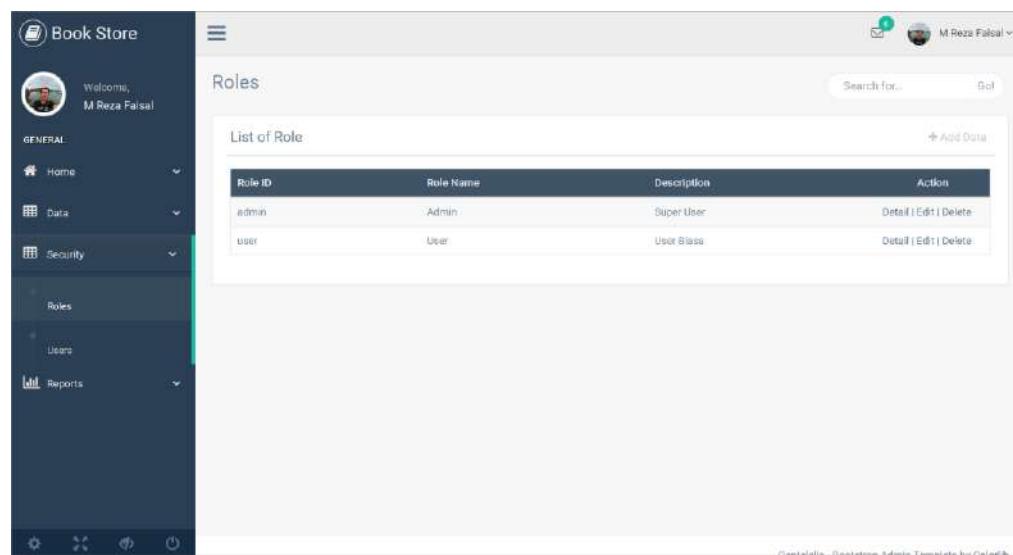
```

View Role

File-file berikut ini menangani antarmuka untuk class RoleController. Semua file di bawah ini disimpan pada folder Views/Role.

Index.cshtml

Berikut adalah antarmuka untuk menampilkan daftar role.



Gambar 163. View Role - Index.cshtml.

Berikut adalah kode yang digunakan untuk membuat antarmuka tersebut.

```
Index.cshtml
@model IList<SqlServerBookStore.Models.RoleViewModel>



### Roles



Go!


```

```

        <h2>List of Role</h2>
        <ul class="nav navbar-right panel_toolbox">
            <li><a asp-controller="Role" asp-
action="Create"><i class="fa fa-plus"></i> Add Data</a></li>
        </ul>
        <div class="clearfix"></div>
    </div>

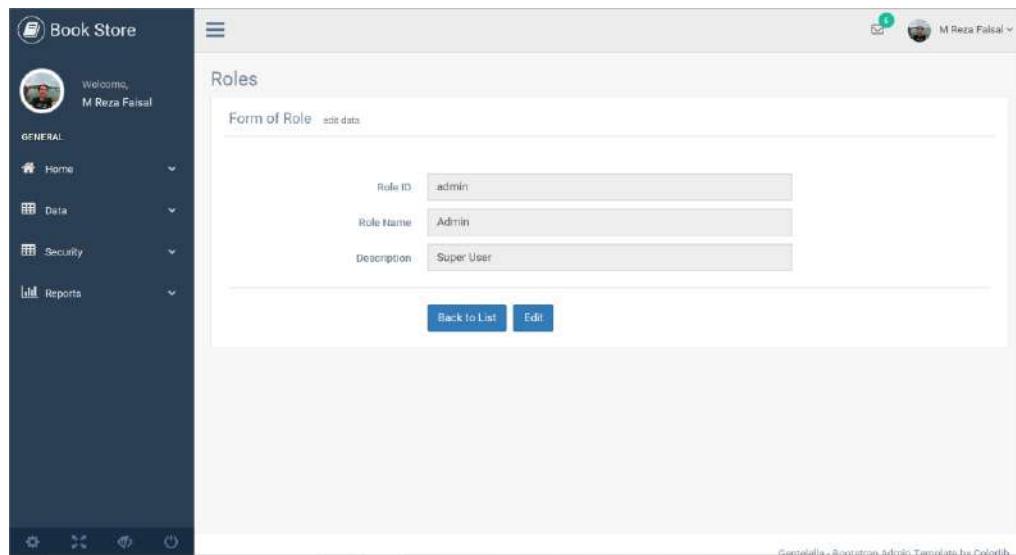
    <div class="x_content">
        <div class="table-responsive">
            <table class="table table-
striped jambo_table bulk_action">
                <thead>
                    <tr class="headings">
                        <th class="column-
title" style="width: 25%">@Html.DisplayNameFor(model => model.FirstOrDefault
().RoleID)</th>
                        <th class="column-
title" style="width: 30%">@Html.DisplayNameFor(model => model.FirstOrDefault
().RoleName)</th>
                        <th class="column-
title" style="width: 30%">@Html.DisplayNameFor(model => model.FirstOrDefault
().Description)</th>
                        <th class="column-title no-
link last" style="width: 15%;text-
align:center"><span class="nobr">Action</span></th>
                    </tr>
                </thead>

                <tbody>
                    @{
                        var odd = false;
}
                    @foreach (var item in Model)
{
                        <tr class="@odd ? "odd": "even"") pointe
r">
                            <td class=" " >@item.RoleID</td>
                            <td class=" " >@item.RoleName</td>
                            <td class=" " >@item.Description</td>
                            <td class=" last">
                                <a asp-controller="Role" asp-
action="Detail" asp-route-id="@item.RoleID">Detail</a> |
                                <a asp-controller="Role" asp-
action="Edit" asp-route-id="@item.RoleID">Edit</a> |
                                <a asp-controller="Role" asp-
action="Delete" asp-route-id="@item.RoleID">Delete</a>
                            </td>
                        </tr>
                        odd = !odd;
}
                </tbody>
            </table>
        </div>
    </div>
</div>
</div>

```

Detail.cshtml

Berikut adalah antarmuka untuk menampilkan data role yang dipilih pada halaman daftar role.



Gambar 164. View Role - Detail.cshtml.

Berikut adalah kode untuk membuat antarmuka di atas.

```
Detail.cshtml
@model SqlServerBookStore.Models.RoleViewModel

<div class="">
    <div class="page-title">
        <div class="title_left">
            <h3>Roles</h3>
        </div>

        <div class="clearfix"></div>

        <div class="row">
            <div class="col-md-12 col-sm-12 col-xs-12">
                <div class="x_panel">
                    <div class="x_title">
                        <h2>Form of Role <small>edit data</small></h2>
                        <div class="clearfix"></div>
                    </div>
                    <div class="x_content">
                        <br />
                        <form data-parsley-validate class="form-horizontal form-label-left">
                            <div asp-validation-summary="All"></div>
                            <div class="form-group">
                                <label asp-for="RoleID" class="control-label col-md-3 col-sm-3 col-xs-12"></label>
                                <div class="col-md-6 col-sm-6 col-xs-12">
                                    <input asp-for="RoleID" readonly="readonly" class="form-control col-md-7 col-xs-12" />
                                </div>
                            </div>
                            <div class="form-group">
                                <label asp-for="RoleName" class="control-label col-md-3 col-sm-3 col-xs-12"></label>
                                <div class="col-md-6 col-sm-6 col-xs-12">

```

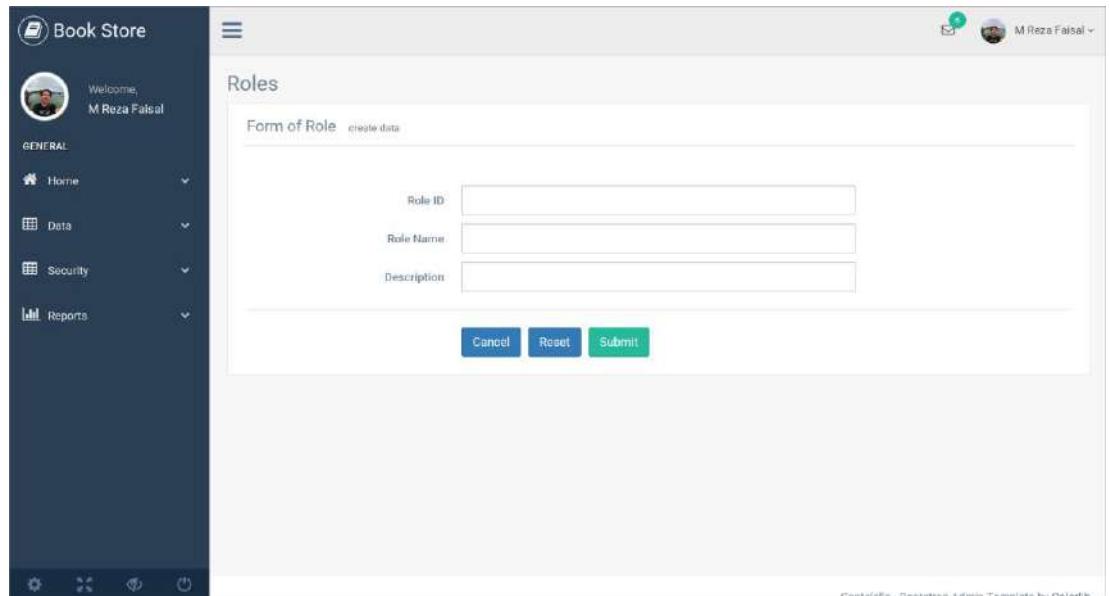
```

                <input asp-
for="RoleName" readonly="readonly" class="form-control col-md-7 col-xs-12">
            </div>
        </div>
        <div class="form-group">
            <label asp-for="Description" class="control-
label col-md-3 col-sm-3 col-xs-12"></label>
            <div class="col-md-6 col-sm-6 col-xs-12">
                <input asp-
for="Description" readonly="readonly" class="form-control col-md-7 col-xs-
12">
                    </div>
                </div>
                <div class="ln_solid"></div>
                <div class="form-group">
                    <div class="col-md-6 col-sm-6 col-xs-12 col-
md-offset-3">
                        <button class="btn btn-
primary" type="button" onclick="location.href='@Url.Action("Index", "Role")'>Back to List</button>
                        <button class="btn btn-
primary" type="button" onclick="location.href='@Url.Action("Edit", "Role", n
ew { id = @Model.RoleID})'">Edit</button>
                    </div>
                </div>
            </div>
        </div>
    </div>
</div>
</div>
</div>
</div>
</div>

```

Create.cshtml

Berikut adalah antarmuka untuk input data role.



Gambar 165. View Role - Create.cshtml

Berikut adalah kode untuk membuat antarmuka di atas.

```
Create.cshtml
@model SqlServerBookStore.Models.RoleViewModel

<div class="">
    <div class="page-title">
        <div class="title_left">
            <h3>Roles</h3>
        </div>

        <div class="clearfix"></div>

        <div class="row">
            <div class="col-md-12 col-sm-12 col-xs-12">
                <div class="x_panel">
                    <div class="x_title">
                        <h2>Form of Role <small>create data</small> </h2>
                        <div class="clearfix"></div>
                    </div>
                    <div class="x_content">
                        <br />
                        <form asp-controller="Role" asp-
action="Create" data-parsley-validate class="form-horizontal form-label-left">
                            <div asp-validation-summary="All"></div>
                            <div class="form-group">
                                <label asp-for="RoleID" class="control-
label col-md-3 col-sm-3 col-xs-12"></label>
                                <div class="col-md-6 col-sm-6 col-xs-12">
                                    <input asp-
for="RoleID" required="required" class="form-control col-md-7 col-xs-12">
                                    <span asp-validation-
for="RoleID"></span>
                                </div>
                            </div>
                            <div class="form-group">
                                <label asp-for="RoleName" class="control-
label col-md-3 col-sm-3 col-xs-12"></label>
                                <div class="col-md-6 col-sm-6 col-xs-12">
                                    <input asp-
for="RoleName" required="required" class="form-control col-md-7 col-xs-12">
                                    <span asp-validation-
for="RoleName"></span>
                                </div>
                            </div>
                            <div class="form-group">
                                <label asp-for="Description" class="control-
label col-md-3 col-sm-3 col-xs-12"></label>
                                <div class="col-md-6 col-sm-6 col-xs-12">
                                    <input asp-
for="Description" required="required" class="form-control col-md-7 col-xs-
12">
                                    <span asp-validation-
for="Description"></span>
                                </div>
                            </div>
                            <div class="ln_solid"></div>
                            <div class="form-group">
                                <div class="col-md-6 col-sm-6 col-xs-12 col-
md-offset-3">
```

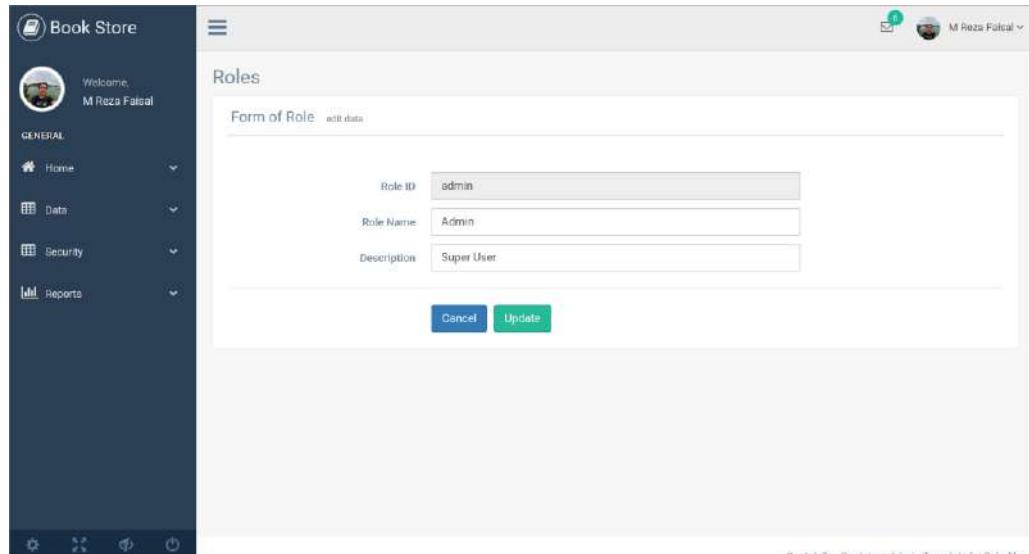
```

        <button class="btn btn-primary" type="button" onclick="location.href='@Url.Action("Index", "Role")'">Cancel</button>
        <button class="btn btn-primary" type="reset" onclick='document.forms[0].reset();return false;'>Reset</button>
        <button type="submit" class="btn btn-success">Submit</button>
    </div>
    </div>
</form>
</div>
</div>
</div>
</div>
</div>
</div>

```

Edit.cshtml

Berikut ini adalah antarmuka untuk mengedit data role yang dipilih.



Gambar 166. View Role - Edit.

Berikut adalah kode untuk membuat antarmuka di atas.

```

Edit.cshtml
@model SqlServerBookStore.Models.RoleViewModel

<div class="">
    <div class="page-title">
        <div class="title_left">
            <h3>Roles</h3>
        </div>
        <div class="clearfix"></div>
        <div class="row">
            <div class="col-md-12 col-sm-12 col-xs-12">
                <div class="x_panel">
                    <div class="x_title">
                        <h2>Form of Role <small>edit data</small></h2>
                        <div class="clearfix"></div>

```

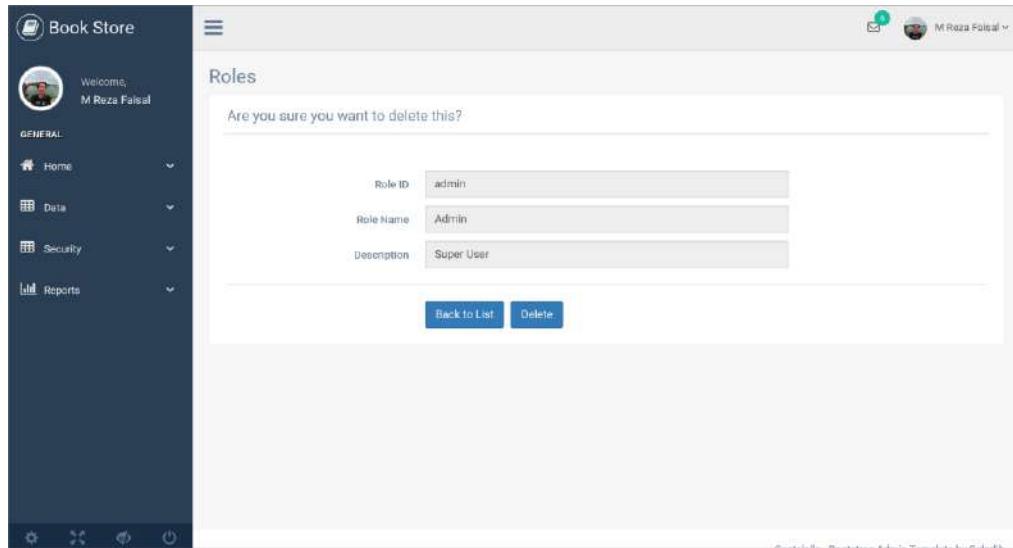
```

        </div>
        <div class="x_content">
            <br />
            <form asp-controller="Role" asp-action="Edit" data-
parsley-validate class="form-horizontal form-label-left">
                <div asp-validation-summary="All"></div>
                <div class="form-group">
                    <label asp-for="RoleID" class="control-
label col-md-3 col-sm-3 col-xs-12"></label>
                        <div class="col-md-6 col-sm-6 col-xs-12">
                            <input asp-
for="RoleID" required="required" readonly="readonly" class="form-
control col-md-7 col-xs-12">
                                <span asp-validation-
for="RoleID"></span>
                        </div>
                </div>
                <div class="form-group">
                    <label asp-for="RoleName" class="control-
label col-md-3 col-sm-3 col-xs-12"></label>
                        <div class="col-md-6 col-sm-6 col-xs-12">
                            <input asp-
for="RoleName" required="required" class="form-control col-md-7 col-xs-12">
                                <span asp-validation-
for="RoleName"></span>
                        </div>
                </div>
                <div class="form-group">
                    <label asp-for="Description" class="control-
label col-md-3 col-sm-3 col-xs-12"></label>
                        <div class="col-md-6 col-sm-6 col-xs-12">
                            <input asp-
for="Description" required="required" class="form-control col-md-7 col-xs-
12">
                                <span asp-validation-
for="Description"></span>
                        </div>
                </div>
                <div class="ln_solid"></div>
                <div class="form-group">
                    <div class="col-md-6 col-sm-6 col-xs-12 col-
md-offset-3">
                        <button class="btn btn-
primary" type="button" onclick="location.href='@Url.Action("Index", "Role")'>Cancel</button>
                        <button type="submit" class="btn btn-
success">Update</button>
                    </div>
                </div>
            </form>
        </div>
    </div>
</div>

```

Delete.cshtml

Berikut adalah antarmuka untuk menampilkan halaman konfirmasi untuk menghapus data role yang telah dipilih.



Gambar 167. View Role - Delete.

Berikut adalah kode untuk membuat antarmuka di atas.

```
>Delete.cshtml
@model SqlServerBookStore.Models.RoleViewModel

<div class="">
    <div class="page-title">
        <div class="title_left">
            <h3>Roles</h3>
        </div>

        <div class="clearfix"></div>

        <div class="row">
            <div class="col-md-12 col-sm-12 col-xs-12">
                <div class="x_panel">
                    <div class="x_title">
                        <h2>Are you sure you want to delete this?</h2>
                        <div class="clearfix"></div>
                    </div>
                    <div class="x_content">
                        <br />
                        <form asp-controller="Role" asp-
action="Delete" data-parsley-validate class="form-horizontal form-label-left">
                            <div asp-validation-summary="All"></div>
                            <input type="hidden" asp-for="RoleID" />
                            <div class="form-group">
                                <label asp-for="RoleID" class="control-
label col-md-3 col-sm-3 col-xs-12"></label>
                                <div class="col-md-6 col-sm-6 col-xs-12">
                                    <input asp-
for="RoleID" readonly="readonly" class="form-control col-md-7 col-xs-12">
                                </div>
                            </div>
                            <div class="form-group">

```

```

        <label asp-for="RoleName" class="control-
label col-md-3 col-sm-3 col-xs-12"></label>
            <div class="col-md-6 col-sm-6 col-xs-12">
                <input asp-
for="RoleName" readonly="readonly" class="form-control col-md-7 col-xs-12">
                    </div>
            </div>
            <div class="form-group">
                <label asp-for="Description" class="control-
label col-md-3 col-sm-3 col-xs-12"></label>
                    <div class="col-md-6 col-sm-6 col-xs-12">
                        <input asp-
for="Description" readonly="readonly" class="form-control col-md-7 col-xs-
12">
                            </div>
                    </div>
                    <div class="ln_solid"></div>
                    <div class="form-group">
                        <div class="col-md-6 col-sm-6 col-xs-12 col-
md-offset-3">
                            <button class="btn btn-
primary" type="button" onclick="location.href='@Url.Action("Index", "Role")'>Back to List</button>
                            <button class="btn btn-
primary" type="submit">Delete</button>
                        </div>
                    </div>
                </div>
            </div>
        </div>
    </div>
</div>

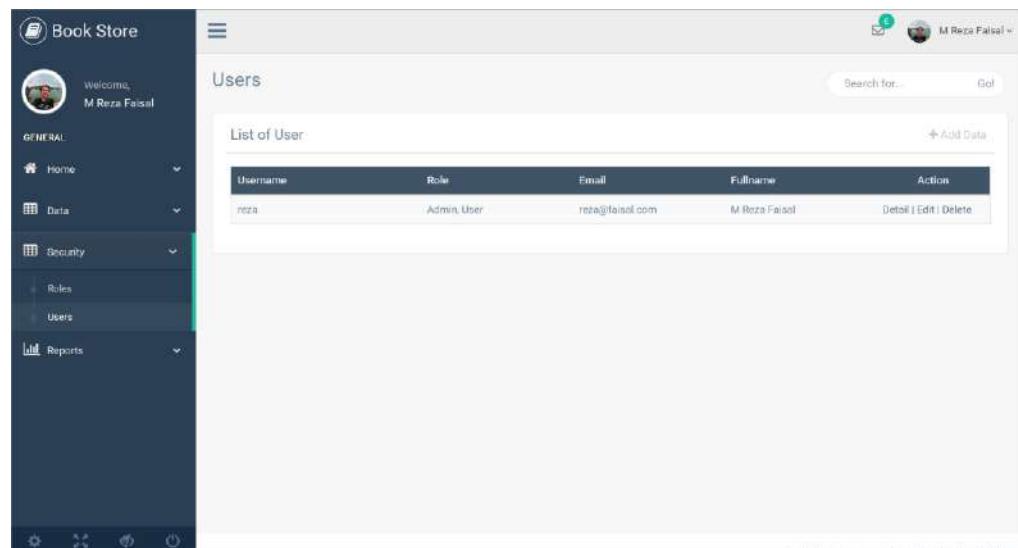
```

View User

File-file berikut ini menangani antarmuka untuk class UserController. Semua file di bawah ini disimpan pada folder Views/User.

Index.cshtml

Berikut adalah antarmuka untuk menampilkan daftar user.



Gambar 168. View User - Index.cshtml.

Berikut adalah kode yang digunakan untuk membuat antarmuka tersebut.

```
Index.cshtml
@model IList<SqlServerBookStore.Models.UserViewModel>



<
    <div class="page-title">
        <div class="title_left">
            <h3>Users</h3>
        </div>

        <div class="title_right">
            <div class="col-md-5 col-sm-5 col-xs-12 form-group pull-right top_search">
                <div class="input-group">
                    <input type="text" class="form-control" placeholder="Search for...">
                    <span class="input-group-btn">
                        <button class="btn btn-default" type="button">Go!</button>
                    </span>
                </div>
            </div>
        </div>
    </div>

    <div class="clearfix"></div>

    <div class="row">
        <div class="col-md-12 col-sm-12 col-xs-12">
            <div class="x_panel">
                <div class="x_title">


```

```

        <h2>List of User</h2>
        <ul class="nav navbar-right panel_toolbox">
            <li><a asp-controller="User" asp-
action="Create"><i class="fa fa-plus"></i> Add Data</a></li>
        </ul>
        <div class="clearfix"></div>
    </div>

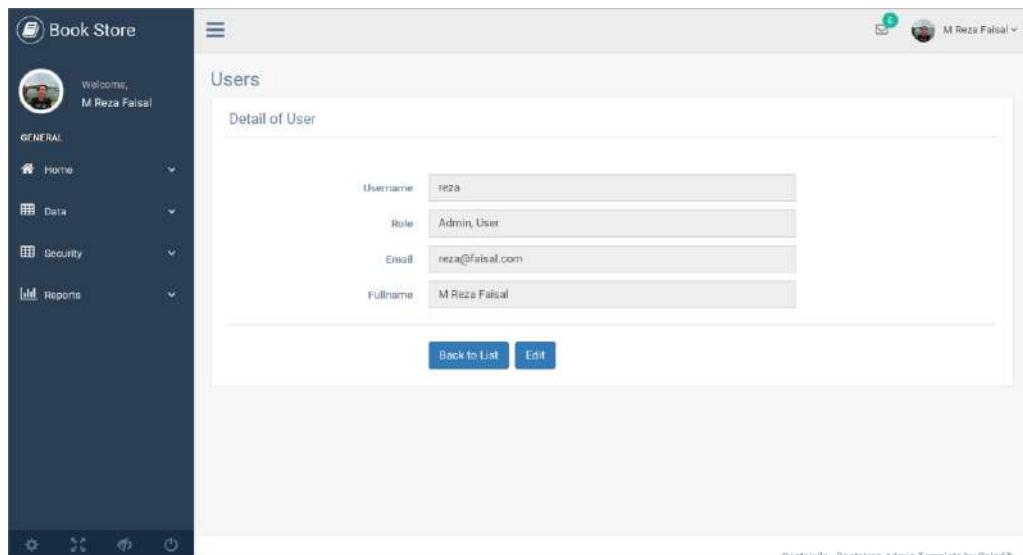
    <div class="x_content">
        <div class="table-responsive">
            <table class="table table-
striped jambo_table bulk_action">
                <thead>
                    <tr class="headings">
                        <th class="column-
title" style="width: 25%">@Html.DisplayNameFor(model => model.FirstOrDefault
().UserName)</th>
                        <th class="column-
title" style="width: 20%">@Html.DisplayNameFor(model => model.FirstOrDefault
().RoleName)</th>
                        <th class="column-
title" style="width: 20%">@Html.DisplayNameFor(model => model.FirstOrDefault
().Email)</th>
                        <th class="column-
title" style="width: 20%">@Html.DisplayNameFor(model => model.FirstOrDefault
().FullName)</th>
                        <th class="column-title no-
link last" style="width: 15%;text-
align:center"><span class="nobr">Action</span></th>
                </tr>
            </thead>

            <tbody>
                @{
                    var odd = false;
                }
                @foreach (var item in Model)
                {
                    <tr class="@{odd ? "odd": "even"} pointe
r">
                        <td class=" " >@item.UserName</td>
                        <td class=" " >@item.RoleName</td>
                        <td class=" " >@item.Email</td>
                        <td class=" " >@item.FullName</td>
                        <td class=" last">
                            <a asp-controller="User" asp-
action="Detail" asp-route-id="@item.UserName">Detail</a> |
                            <a asp-controller="User" asp-
action="Edit" asp-route-id="@item.UserName">Edit</a> |
                            <a asp-controller="User" asp-
action="Delete" asp-route-id="@item.UserName">Delete</a>
                        </td>
                    </tr>
                    odd = !odd;
                }
            </tbody>
        </table>
    </div>
</div>
</div>

```

Detail.cshtml

Berikut adalah antarmuka untuk menampilkan data user yang dipilih pada halaman daftar user.



Gambar 169. View User - Detail.cshtml.

Berikut adalah kode untuk membuat antarmuka di atas.

```
Detail.cshtml
@model SqlServerBookStore.Models.UserViewModel

<div class="">
    <div class="page-title">
        <div class="title_left">
            <h3>Users</h3>
        </div>
        <div class="clearfix"></div>
    </div>
    <div class="row">
        <div class="col-md-12 col-sm-12 col-xs-12">
            <div class="x_panel">
                <div class="x_title">
                    <h2>Detail of User</h2>
                    <div class="clearfix"></div>
                </div>
                <div class="x_content">
                    <br />
                    <form asp-controller="User" asp-action="Edit" data-parsley-validate class="form-horizontal form-label-left">
                        <div asp-validation-summary="All"></div>
                        <div class="form-group">
                            <label asp-for="UserName" class="control-label col-md-3 col-sm-3 col-xs-12"></label>
                            <div class="col-md-6 col-sm-6 col-xs-12">
                                <input asp-for="UserName" readonly="readonly" class="form-control col-md-7 col-xs-12" />
                            </div>
                        </div>
                        <div class="form-group">
                            <label asp-for="RoleName" class="control-label col-md-3 col-sm-3 col-xs-12"></label>
                            <div class="col-md-6 col-sm-6 col-xs-12">
                                <input asp-for="RoleName" class="form-control col-md-7 col-xs-12" />
                            </div>
                        </div>
                    </form>
                </div>
            </div>
        </div>
    </div>
</div>
```

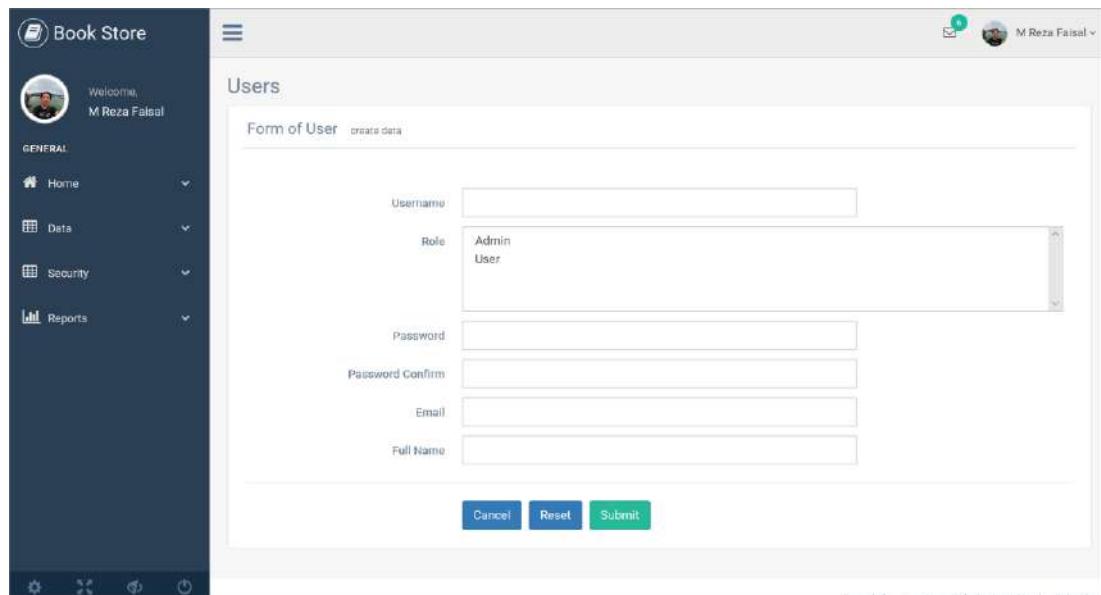
```

                <input asp-
for="RoleName" readonly="readonly" class="form-control col-md-7 col-xs-12">
            </div>
        </div>
        <div class="form-group">
            <label asp-for="Email" class="control-
label col-md-3 col-sm-3 col-xs-12"></label>
            <div class="col-md-6 col-sm-6 col-xs-12">
                <input asp-
for="Email" readonly="readonly" class="form-control col-md-7 col-xs-12">
                </div>
            </div>
            <div class="form-group">
                <label asp-for="FullName" class="control-
label col-md-3 col-sm-3 col-xs-12"></label>
                <div class="col-md-6 col-sm-6 col-xs-12">
                    <input asp-
for="FullName" readonly="readonly" class="form-control col-md-7 col-xs-12">
                    </div>
                </div>
                <div class="ln_solid"></div>
                <div class="form-group">
                    <div class="col-md-6 col-sm-6 col-xs-12 col-
md-offset-3">
                        <button class="btn btn-
primary" type="button" onclick="location.href=@Url.Action("Index", "User")'>Back to List</button>
                        <button class="btn btn-
primary" type="button" onclick="location.href=@Url.Action("Edit", "User", n
ew { id = @Model.UserName})'">Edit</button>
                    </div>
                </div>
            </div>
        </div>
    </div>
</div>
</div>

```

Create.cshtml

Berikut adalah antarmuka untuk input data user.



Gambar 170. View User - Create.cshtml

Berikut adalah kode untuk membuat antarmuka di atas.

```
Create.cshtml
@model SqlServerBookStore.Models.UserCreateFormViewModel

<div class="">
    <div class="page-title">
        <div class="title_left">
            <h3>Users</h3>
        </div>

        <div class="clearfix"></div>

        <div class="row">
            <div class="col-md-12 col-sm-12 col-xs-12">
                <div class="x_panel">
                    <div class="x_title">
                        <h2>Form of User <small>create data</small></h2>
                        <div class="clearfix"></div>
                    </div>
                    <div class="x_content">
                        <br />
                        <form asp-controller="User" asp-
action="Create" data-parsley-validate class="form-horizontal form-label-
left">
                            <div asp-validation-summary="All"></div>
                            <div class="form-group">
                                <label asp-for="UserName" class="control-
label col-md-3 col-sm-3 col-xs-12"></label>
                                <div class="col-md-6 col-sm-6 col-xs-12">
                                    <input asp-
for="UserName" required="required" class="form-control col-md-7 col-xs-12">
                                    <span asp-validation-
for="UserName"></span>
                                </div>
                            </div>
                            <div class="form-group">
```

```

        <label asp-for="RoleID" class="control-
label col-md-3 col-sm-3 col-xs-12"></label>
            <div class="col-md-9 col-sm-9 col-xs-12">
                <select asp-for="RoleID" asp-
items="@ViewBag.Roles" class="select2_multiple form-
control" multiple="multiple"></select>
                    <span asp-validation-
for="RoleID"></span>
                </div>
            </div>
            <div class="form-group">
                <label asp-for="Password" class="control-
label col-md-3 col-sm-3 col-xs-12"></label>
                <div class="col-md-6 col-sm-6 col-xs-12">
                    <input asp-
for="Password" required="required" class="form-control col-md-7 col-xs-12">
                    <span asp-validation-
for="Password"></span>
                </div>
            </div>
            <div class="form-group">
                <label asp-
for="PasswordConfirm" class="control-label col-md-3 col-sm-3 col-xs-
12"></label>
                <div class="col-md-6 col-sm-6 col-xs-12">
                    <input asp-
for="PasswordConfirm" required="required" class="form-control col-md-7 col-
xs-12">
                    <span asp-validation-
for="PasswordConfirm"></span>
                </div>
            </div>
            <div class="form-group">
                <label asp-for="Email" class="control-
label col-md-3 col-sm-3 col-xs-12"></label>
                <div class="col-md-6 col-sm-6 col-xs-12">
                    <input asp-
for="Email" required="required" class="form-control col-md-7 col-xs-12">
                    <span asp-validation-for="Email"></span>
                </div>
            </div>
            <div class="form-group">
                <label asp-for="FullName" class="control-
label col-md-3 col-sm-3 col-xs-12"></label>
                <div class="col-md-6 col-sm-6 col-xs-12">
                    <input asp-
for="FullName" required="required" class="form-control col-md-7 col-xs-12">
                    <span asp-validation-
for="FullName"></span>
                </div>
            </div>
            <div class="ln_solid"></div>
            <div class="form-group">
                <div class="col-md-6 col-sm-6 col-xs-12 col-
md-offset-3">
                    <button class="btn btn-
primary" type="button" onclick="location.href='@Url.Action("Index", "User")'>Cancel</button>
                    <button class="btn btn-
primary" type="reset" onclick='document.forms[0].reset();return false;'>Rese-
t</button>

```

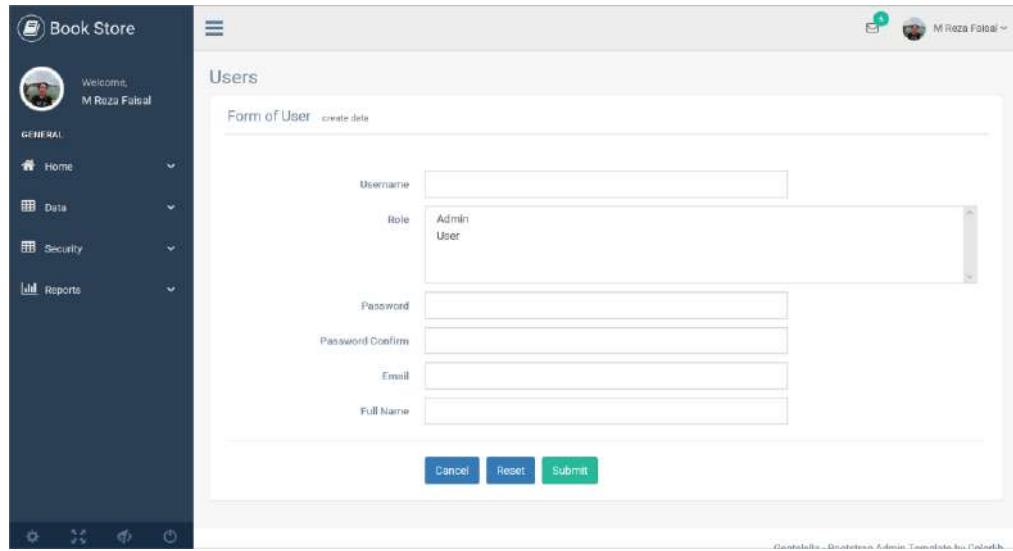
```

        <button type="submit" class="btn btn-success">Submit</button>
    </div>
</div>
</form>
</div>
</div>
</div>
</div>
</div>
</div>

```

Edit.cshtml

Berikut ini adalah antarmuka untuk mengedit data user yang dipilih.



Gambar 171. View User - Edit.

Berikut adalah kode untuk membuat antarmuka di atas.

```

Edit.cshtml
@model SqlServerBookStore.Models.UserEditFormViewModel

<div class="">
    <div class="page-title">
        <div class="title_left">
            <h3>Users</h3>
        </div>
        <div class="clearfix"></div>
    <div class="row">
        <div class="col-md-12 col-sm-12 col-xs-12">
            <div class="x_panel">
                <div class="x_title">
                    <h2>Form of User <small>edit data</small></h2>
                    <div class="clearfix"></div>
                </div>
                <div class="x_content">
                    <br />
                    <form asp-controller="User" asp-action="Edit" data-parsley-validate class="form-horizontal form-label-left">
                        <div asp-validation-summary="All"></div>

```

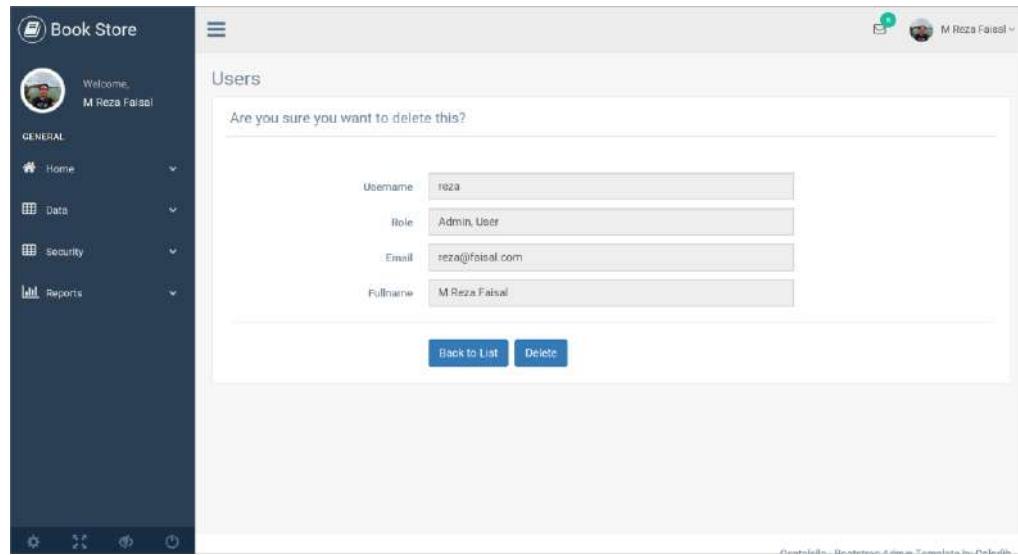
```

        <div class="form-group">
            <label asp-for="UserName" class="control-
label col-md-3 col-sm-3 col-xs-12"></label>
            <div class="col-md-6 col-sm-6 col-xs-12">
                <input asp-
for="UserName" required="required" readonly="readonly" class="form-
control col-md-7 col-xs-12">
                <span asp-validation-
for="UserName"></span>
            </div>
        </div>
        <div class="form-group">
            <label asp-for="RoleID" class="control-
label col-md-3 col-sm-3 col-xs-12"></label>
            <div class="col-md-9 col-sm-9 col-xs-12">
                <select asp-for="RoleID" asp-
items="@ViewBag.Roles" class="select2_multiple form-
control" multiple="multiple"></select>
                <span asp-validation-
for="RoleID"></span>
            </div>
        </div>
        <div class="form-group">
            <label asp-for="Email" class="control-
label col-md-3 col-sm-3 col-xs-12"></label>
            <div class="col-md-6 col-sm-6 col-xs-12">
                <input asp-
for="Email" required="required" class="form-control col-md-7 col-xs-12">
                <span asp-validation-for="Email"></span>
            </div>
        </div>
        <div class="form-group">
            <label asp-for="FullName" class="control-
label col-md-3 col-sm-3 col-xs-12"></label>
            <div class="col-md-6 col-sm-6 col-xs-12">
                <input asp-
for="FullName" required="required" class="form-control col-md-7 col-xs-12">
                <span asp-validation-
for="FullName"></span>
            </div>
        </div>
        <div class="ln_solid"></div>
        <div class="form-group">
            <div class="col-md-6 col-sm-6 col-xs-12 col-
md-offset-3">
                <button class="btn btn-
primary" type="button" onclick="location.href='@Url.Action("Index", "User")'">Cancel</button>
                <button type="submit" class="btn btn-
success">Update</button>
            </div>
        </div>
    </form>
</div>
</div>
</div>
</div>
</div>

```

Delete.cshtml

Berikut adalah antarmuka untuk menampilkan halaman konfirmasi untuk menghapus data user yang telah dipilih.



Gambar 172. View User - Delete.

Berikut adalah kode untuk membuat antarmuka di atas.

```
>Delete.cshtml
@model SqlServerBookStore.Models.UserViewModel

<div class="">
    <div class="page-title">
        <div class="title_left">
            <h3>Users</h3>
        </div>

        <div class="clearfix"></div>

        <div class="row">
            <div class="col-md-12 col-sm-12 col-xs-12">
                <div class="x_panel">
                    <div class="x_title">
                        <h2>Are you sure you want to delete this?</h2>
                        <div class="clearfix"></div>
                    </div>
                    <div class="x_content">
                        <br />
                        <form asp-controller="User" asp-
action="Delete" data-parsley-validate class="form-horizontal form-label-
left">
                            <div asp-validation-summary="All"></div>
                            <input type="hidden" asp-for="UserName" />
                            <div class="form-group">
                                <label asp-for="UserName" class="control-
label col-md-3 col-sm-3 col-xs-12"></label>
                                <div class="col-md-6 col-sm-6 col-xs-12">
                                    <input asp-
for="UserName" readonly="readonly" class="form-control col-md-7 col-xs-12">
                                </div>
                            </div>
                        </form>
                    </div>
                </div>
            </div>
        </div>
    </div>
</div>
```

```

                </div>
                <div class="form-group">
                    <label asp-for="RoleName" class="control-
label col-md-3 col-sm-3 col-xs-12"></label>
                    <div class="col-md-6 col-sm-6 col-xs-12">
                        <input asp-
for="RoleName" readonly="readonly" class="form-control col-md-7 col-xs-12">
                    </div>
                </div>
                <div class="form-group">
                    <label asp-for="Email" class="control-
label col-md-3 col-sm-3 col-xs-12"></label>
                    <div class="col-md-6 col-sm-6 col-xs-12">
                        <input asp-
for="Email" readonly="readonly" class="form-control col-md-7 col-xs-12">
                    </div>
                </div>
                <div class="form-group">
                    <label asp-for="FullName" class="control-
label col-md-3 col-sm-3 col-xs-12"></label>
                    <div class="col-md-6 col-sm-6 col-xs-12">
                        <input asp-
for="FullName" readonly="readonly" class="form-control col-md-7 col-xs-12">
                    </div>
                </div>
                <div class="ln_solid"></div>
                <div class="form-group">
                    <div class="col-md-6 col-sm-6 col-xs-12 col-
md-offset-3">
                        <button class="btn btn-
primary" type="button" onclick="location.href='@Url.Action("Index", "User")'">Back to List</button>
                        <button class="btn btn-
primary" type="submit">Delete</button>
                    </div>
                </div>
            </div>
        </div>
    </div>
</div>

```

Implementasi Keamanan

Pada bab ini akan dilakukan pengamanan aplikasi dengan cara yang umum dilakukan yaitu dengan integrasi sistem pemeriksaan otentifikasi dengan menggunakan halaman login. Selanjutnya adalah integrasi sistem otorisasi yang berfungsi untuk menentukan akses suatu halaman agar hanya dapat diakses oleh user yang telah ditentukan.

Modifikasi Startup.cs

Fungsi utama dari proses otentifikasi adalah user dapat melakukan login untuk memasuki sistem dan logout untuk keluar dari sistem. Langkah yang pertama untuk implementasi proses otentifikasi adalah menentukan method action yang akan digunakan untuk login dan logout.

Sedangkan fungsi utama proses otorisasi adalah memberikan pesan kepada user yang tidak memiliki hak mengakses suatu halaman atau suatu method action. Pesan dapat disampaikan dengan mengirimkan user tersebut ke suatu halaman atau method action.

Untuk menentukan method action yang akan digunakan untuk login, logout dan pesan penolakan hak akses digunakan penambahan kode pada method ConfigureServices di dalam file Startup.cs seperti yang terlihat pada baris yang dicetak tebal.

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddDbContext<ApplicationDbContext>(options =>
        options.UseSqlServer(Configuration.GetConnectionString("DefaultConnection")));

    services.AddIdentity<ApplicationUser, ApplicationRole>(p => {
        p.Password.RequireDigit = false;
        p.Password.RequireLowercase = false;
        p.Password.RequireUppercase = false;
        p.Password.RequireNonAlphanumeric = false;
        p.Password.RequiredLength = 6;
    })
    .AddEntityFrameworkStores<ApplicationDbContext>()
    .AddDefaultTokenProviders();

    services.AddAuthentication(o => {
        o.DefaultAuthenticateScheme = CookieAuthenticationDefaults.AuthenticationScheme;
        o.DefaultChallengeScheme = CookieAuthenticationDefaults.AuthenticationScheme;
        o.DefaultSignInScheme = CookieAuthenticationDefaults.AuthenticationScheme;
    })
    .AddCookie(CookieAuthenticationDefaults.AuthenticationScheme, o => {
        o.LoginPath = new PathString("/account/login/");
        o.LogoutPath = new PathString("/Home/Logout");
        o.AccessDeniedPath = new PathString("/Home/AccessDenied");
    });
}

// Add application services.
services.AddTransient<IEmailSender, EmailSender>();
```

```
        services.AddMvc();
    }
```

Otentikasi

Otentikasi atau authentication adalah proses verifikasi user mempunyai hak untuk mengakses aplikasi. Biasanya proses ini melibatkan username dan password. Pada buku ini proses otentikasi dilakukan dengan bantuan ASP.NET Core Identity.

Model: UserLoginFormViewModel.cs

Model UserLoginFormViewModel digunakan untuk pengiriman data dari komponen view yang berfungsi sebagai form login ke komponen controller yang akan melakukan proses otentikasi.

Berikut ini adalah kode dari file UserLoginFormViewModel.cs

```
UserLoginFormViewModel.cs
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using System.ComponentModel.DataAnnotations;

namespace SqlServerBookStore.Models
{
    public class UserLoginFormViewModel
    {
        [Display(Name = "Username")]
        [Required(ErrorMessage = "{0} harus diisi.")]
        public String UserName { set; get; }

        [Display(Name = "Password")]
        [Required(ErrorMessage = "{0} harus diisi.")]
        [DataType(DataType.Password)]
        public String Password { set; get; }
    }
}
```

View

Untuk komponen view akan dibuat dua file yaitu Login.cshtml dan AccessDenied.cshtml.

Login.cshtml

Komponen view Login.cshtml ini digunakan sebagai form untuk login yang memungkinkan user memasukkan nilai username dan password. Berikut adalah kode lengkap dari file Views/Home/Login.cshtml.

```
Login.cshtml
@{
    Layout = null;
}

@model SqlServerBookStore.Models.UserLoginFormViewModel
```

```

<!DOCTYPE html>
<html lang="en">
<head>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
    <!-- Meta, title, CSS, favicons, etc. -->
    <meta charset="utf-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1">

    <title>ASP.NET Core MVC: Book Store</title>

    <!-- Bootstrap -->
    <link href="~/vendors/bootstrap/dist/css/bootstrap.min.css" rel="stylesheet">
    <!-- Font Awesome -->
    <link href="~/vendors/font-awesome/css/font-awesome.min.css" rel="stylesheet">
    <!-- NProgress -->
    <link href="~/vendors/nprogress/nprogress.css" rel="stylesheet">
    <!-- Animate.css -->
    <link href="~/vendors/animate.css/animate.min.css" rel="stylesheet">

    <!-- Custom Theme Style -->
    <link href="~/build/css/custom.min.css" rel="stylesheet">
</head>

<body class="login">
    <div>
        <a class="hiddenanchor" id="signup"></a>
        <a class="hiddenanchor" id="signin"></a>

        <div class="login_wrapper">
            <div class="animate form login_form">
                <section class="login_content">
                    <form asp-controller="Home" asp-action="Login">
                        <h1><i class="fa fa-book"></i> Book Store</h1>
                        <div>
                            <input asp-for="UserName" class="form-control" placeholder="Username" required="" />
                        </div>
                        <div>
                            <input asp-for="Password" class="form-control" placeholder="Password" required="" />
                        </div>
                        <div>
                            <button type="submit" class="btn btn-success">Login</button>
                        </div>

                        <div class="clearfix"></div>

                        <div class="separator">

                            <div class="clearfix"></div>
                            <br />

                            <div>
                                <p>©2016 All Rights Reserved. Gentelella Ale
                            la! is a Bootstrap 3 template. Privacy and Terms</p>
                            </div>
                        </div>
                </form>
            </section>
        </div>
    </div>
</body>

```

```
        </form>
    </section>
</div>
</div>
</body>
</html>
```

Pada kode di atas dapat dilihat kode berikut ini. Kode ini berfungsi untuk menyatakan halaman ini tidak akan menggunakan layout yang telah ada, yaitu _Layout.cshtml.

```
@{
    Layout = null;
}
```

Kemudian untuk menggunakan model UserLoginFormViewModel digunakan kode berikut ini.

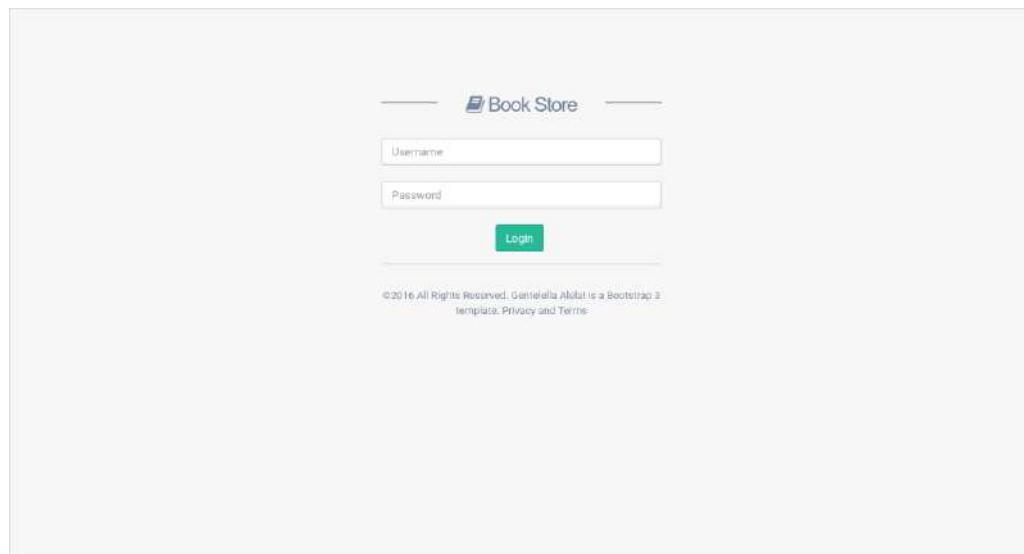
```
@model SqlServerBookStore.Models.UserLoginFormViewModel
```

Selanjutnya digunakan komponen form, input dan button sebagai form login.

```
. . .
<form asp-controller="Home" asp-action="Login">
    <h1><i class="fa fa-book"></i> Book Store</h1>
    <div>
        <input asp-for="UserName" class="form-control" placeholder="Username"
required="" />
    </div>
    <div>
        <input asp-for="Password" class="form-control" placeholder="Password"
required="" />
    </div>
    <div>
        <button type="submit" class="btn btn-success">Login</button>
    </div>
. . .
</form>
. . .
```

Pada kode di atas dapat dilihat proses login akan ditangani oleh method action Login pada HomeController.

Berikut ini adalah tampilan dari form login.



Gambar 173. Implementasi Keamanan - Form Login.

AccessDenied.cshtml

Berikut adalah isi halaman AccessDenied.cshtml.

```
AccessDenied.cshtml
<div class="">
    <h2>Anda tidak memiliki hak akses ke halaman tersebut.</h2>
</div>
```

Berikut ini adalah tampilan dari halaman ini.



Gambar 174. Implementasi Keamanan - AccessDenied.cshtml.

Controller: HomeController

Berikut ini adalah kode lengkap class HomeController.

```
HomeController.cs
using System;
using System.Collections.Generic;
using System.Linq;
```

```

using System.Security.Claims;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Authentication;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Identity;
using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.Mvc.Rendering;
using Microsoft.Extensions.Logging;
using Microsoft.Extensions.Options;
using SqlServerBookStore.Models;

namespace SqlServerBookStore.Controllers
{
    [Authorize]
    public class HomeController : Controller
    {
        private readonly SignInManager< ApplicationUser > _signInManager;

        public HomeController(SignInManager< ApplicationUser > signInManager)
        {
            _signInManager = signInManager;
        }

        public IActionResult Index()
        {
            return View();
        }

        [HttpGet]
        [AllowAnonymous]
        public async Task< IActionResult > Login(string returnUrl = null)
        {
            // Clear the existing external cookie to ensure a clean login process
            await HttpContext.SignOutAsync(IdentityConstants.ExternalScheme);

            ViewData["ReturnUrl"] = returnUrl;
            return View();
        }

        [HttpPost]
        [AllowAnonymous]
        [ValidateAntiForgeryToken]
        public async Task< IActionResult > Login(UserLoginFormViewModel item,
string returnUrl = null)
        {
            ViewData["ReturnUrl"] = returnUrl;

            if (ModelState.IsValid)
            {
                var result = await _signInManager.PasswordSignInAsync(item.UserName, item.Password, isPersistent: false, lockoutOnFailure: false);
                if (result.Succeeded)
                {
                    return RedirectToAction("Index");
                }
            }
        }

        return View();
    }
}

```

```
        }

        [HttpGet]
        public async Task<IActionResult> Logout()
        {
            await _signInManager.SignOutAsync();
            return RedirectToAction("Login");
        }

        [HttpGet]
        public IActionResult AccessDenied()
        {
            return View();
        }
    }
}
```

Method Action [HttpGet] Login

Pada method ini akan ditampilkan halaman untuk form login. Pada method ini terdapat proses untuk menghapus cookies agar proses login dapat dilakukan dengan baik.

```
await HttpContext.SignOutAsync(IdentityConstants.ExternalScheme);
```

Method Action [HttpPost] Login

Method action ini berfungsi untuk memproses username dan password yang dikirimkan dari form login. Pada method ini dapat dilihat proses otentikasi dilakukan dengan menggunakan method PasswordSignInAsync dari class SignInManager.

```
var result = await _signInManager.PasswordSignInAsync(item.UserName, item.Password, isPersistent: false, lockoutOnFailure: false);
if (result.Succeeded)
{
    return RedirectToAction("Index");
}
```

Jika proses login berhasil, maka user akan diantarkan ke halaman Index.cshtml dengan cara memanggil method action Index.

Method Action [HttpGet] Logout

Untuk logout digunakan kode berikut ini.

```
await _signInManager.SignOutAsync();
```

Sedangkan untuk proses logout dipanggil oleh link yang ada pada _Layout.cshtml.

```
<a data-toggle="tooltip" data-placement="top" title="Logout" asp-controller="Home" asp-action="Logout">
    <span class="glyphicon glyphicon-off" aria-hidden="true"></span>
</a>
```

Method Action [HttpGet] AccessDenied

Method ini berfungsi untuk menampilkan halaman AccessDenied.cshtml. Proses penentuan seorang user yang tidak berhak akan diantarkan ke halaman ini dapat dilihat pada setting yang telah dilakukan pada file Startup.cs yang telah dilakukan pada sub bab di atas.

Otorisasi

Pada sub bab di atas sudah dijelaskan penggunaan atribut [Authorize] dan [AllowAnonymous]. Kedua atribut ini adalah merupakan komponen utama pada proses otorisasi. Untuk menggunakan atribut tersebut pada class controller perlu ditambahkan namespace berikut ini.

```
using Microsoft.AspNetCore.Authorization;
```

Untuk mengetahui lebih lanjut pemanfaatan kedua atribut tersebut pada proses otorisasi, maka akan dilakukan untuk menyelesaikan kasus proses otorisasi pada aplikasi Book Store. Aplikasi ini memiliki 5 fitur utama untuk mengelola:

1. Kategori buku yang menggunakan CategoriesController.
2. Pengarang buku yang menggunakan AuthorsController.
3. Buku yang menggunakan BooksController.
4. Role yang menggunakan RoleController.
5. User yang menggunakan UserController.

Kelima fitur itu akan dibagi dua untuk diakses oleh 2 role, yaitu:

1. admin.
2. user.

Roles			
List of Role			
Role ID	Role Name	Description	Action
admin	Admin	Super User	Detail Edit Delete
user	User	User Biasa	Detail Edit Delete

Gambar 175. Implementasi Keamanan - Role.

User dengan role admin dapat mengakses seluruh fitur-fitur tersebut. Sedangkan user dengan role user hanya dapat mengakses fitur pengelolaan kategori buku, pengarang buku dan buku. Untuk membuat hal tersebut maka kelima class controller dimodifikasi seperti berikut ini.

```
CategoriesController.cs
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.Mvc.Rendering;
using Microsoft.EntityFrameworkCore;
using Microsoft.AspNetCore.Authorization;
using SqlServerBookStore.Data;
```

```
using SqlServerBookStore.Models;

namespace SqlServerBookStore.Controllers
{
    [Authorize(Roles = "user")]
    public class CategoriesController : Controller
    {
        .
        .
        .
    }
    .
    .
}
```

```
AuthorsController.cs
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.Mvc.Rendering;
using Microsoft.EntityFrameworkCore;
using Microsoft.AspNetCore.Authorization;
using SqlServerBookStore.Data;
using SqlServerBookStore.Models;

namespace SqlServerBookStore.Controllers
{
    [Authorize(Roles = "user")]
    public class AuthorsController : Controller
    {
        .
        .
        .
    }
    .
    .
}
```

```
BooksController.cs
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.Mvc.Rendering;
using Microsoft.EntityFrameworkCore;
using Microsoft.AspNetCore.Authorization;
using SqlServerBookStore.Data;
using SqlServerBookStore.Models;

namespace SqlServerBookStore.Controllers
{
    [Authorize(Roles = "user")]
    public class BooksController : Controller
    {
        .
        .
        .
    }
    .
    .
}
```

```
RoleController.cs
using System;
using System.Collections.Generic;
```

```

using System.Linq;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.Identity;
using Microsoft.AspNetCore.Authorization;
using SqlServerBookStore.Models;
using SqlServerBookStore.Data;

namespace SqlServerBookStore.Controllers
{
    [Authorize(Roles = "admin")]
    public class RoleController : Controller
    {
        . . .
    }
    . . .
}

```

```

UserController.cs
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.Mvc.Rendering;
using Microsoft.AspNetCore.Identity;
using Microsoft.AspNetCore.Authorization;
using SqlServerBookStore.Models;
using SqlServerBookStore.Data;

namespace SqlServerBookStore.Controllers
{
    [Authorize(Roles = "admin")]
    public class UserController : Controller
    {
        . . .
    }
    . . .
}

```

Dari kode di atas dapat dilihat penggunaan atribut [Authorize] untuk menentukan role yang dapat mengakses method action pada class controller. Berikut adalah sintaks penggunaan atribut ini.

[Authorize]

Atau

[Authorize(Roles = "role-1, role-2, ... , role-n")]

Dari sintaks di atas dapat dilihat nilai parameter Roles dapat berisi satu nama role atau lebih. Atribut ini juga dapat digunakan langsung di atas method action yang diinginkan. Sebagai contoh jika ingin agar method action Index pada class RoleController dapat diakses oleh role user maka dapat dilakukan modifikasi sebagai berikut.

RoleController.cs

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Mvc;

```

```

using Microsoft.AspNetCore.Identity;
using Microsoft.AspNetCore.Authorization;
using SqlServerBookStore.Models;
using SqlServerBookStore.Data;

namespace SqlServerBookStore.Controllers
{
    public class RoleController : Controller
    {
        private readonly RoleManager<ApplicationRole> db;

        public RoleController(RoleManager<ApplicationRole> roleManager)
        {
            this.db = roleManager;
        }

        [HttpGet]
        [Authorize(Roles = "admin, user")]
        public IActionResult Index()
        {

            var items = new List<RoleViewModel>();
            items = db.Roles.Select(r => new RoleViewModel
            {
                RoleID = r.Id,
                RoleName = r.Name,
                Description = r.Description
            }).ToList();

            return View(items);
        }

        [HttpGet]
        [Authorize(Roles = "admin, user")]
        public async Task<IActionResult> Detail(string id)
        {
            RoleViewModel item = new RoleViewModel();
            ApplicationRole role = await db.FindByIdAsync(id);
            if (role != null)
            {
                item.RoleID = role.Id;
                item.RoleName = role.Name;
                item.Description = role.Description;
            }
            return View(item);
        }

        [HttpGet]
        [Authorize(Roles = "admin")]
        public IActionResult Create()
        {
            return View();
        }

        [HttpPost]
        [Authorize(Roles = "admin")]
        public async Task<IActionResult> Create(RoleViewModel item)
        {
            if (ModelState.IsValid)
            {

```

```

        ApplicationRole role = new ApplicationRole();
        role.Id = item.RoleID;
        role.Name = item.RoleName;
        role.Description = item.Description;

        var result = await db.CreateAsync(role);

        return RedirectToAction("Index");
    }

    return View();
}

[HttpGet]
[Authorize(Roles = "admin")]
public async Task<IActionResult> Edit(string id)
{
    RoleViewModel item = new RoleViewModel();
    ApplicationRole role = await db.FindByIdAsync(id);
    if (role != null)
    {
        item.RoleID = role.Id;
        item.RoleName = role.Name;
        item.Description = role.Description;
    }
    return View(item);
}

[HttpPost]
[Authorize(Roles = "admin")]
[ValidateAntiForgeryToken]
public async Task<IActionResult> Edit([Bind("RoleID,RoleName,Description")] RoleViewModel item)
{
    if (ModelState.IsValid)
    {
        ApplicationRole role = await db.FindByIdAsync(item.RoleID);
        if (role != null)
        {
            role.Id = item.RoleID;
            role.Name = item.RoleName;
            role.Description = item.Description;
            var result = await db.UpdateAsync(role);
        }
        return RedirectToAction("Index");
    }
    return View();
}

[HttpGet]
[Authorize(Roles = "admin")]
public async Task<IActionResult> Delete(string id)
{
    RoleViewModel item = new RoleViewModel();
    ApplicationRole role = await db.FindByIdAsync(id);
    if (role != null)
    {
        item.RoleID = role.Id;
        item.RoleName = role.Name;
        item.Description = role.Description;
    }
}

```

```

        }
        return View(item);
    }

[HttpPost, ActionName("Delete")]
[Authorize(Roles = "admin")]
[ValidateAntiForgeryToken]
public async Task<IActionResult> DeleteConfirmed(string id)
{
    if (ModelState.IsValid)
    {
        ApplicationRole role = await db.FindByIdAsync(id);
        var result = await db.DeleteAsync(role);

        return RedirectToAction("Index");
    }

    return View();
}
}

```

Pada contoh di atas atribut [Authorize] tidak digunakan di atas nama class controller, tetapi digunakan pada setiap method action. Kemudian dapat dilihat seluruh method action hanya dapat diakses oleh role admin, kecuali method action Index yang dapat diakses oleh role admin dan user. Aturan otorisasi tersebut membuat user dengan role user dapat mengakses halaman daftar role tetapi tidak dapat melakukan penambahan, edit dah hhapus data.

Dari penjelasan di atas maka pembaca dapat memiliki pengetahuan bagaimana membuat aturan otorisasi sesuai dengan kebutuhan.

9

Penutup

Sebagai penutup, buku ini ditulis untuk para web developer yang ingin membangun aplikasi web lintas platform dengan ASP.NET Core dengan database MS SQL Server dengan tool development Visual Studio 2017.

Akhir kalimat, jika ada kritik dan saran dapat ditujukan langsung via email ke alamat reza.faisal@gmail.com.