

See discussions, stats, and author profiles for this publication at: <https://www.researchgate.net/publication/316597650>

# Seri Belajar ASP.NET: ASP.NET Core MVC & MySQL dengan Visual Studio Code

Book · May 2017

---

CITATIONS

0

READS

14,498

1 author:



M Reza Faisal

Universitas Lambung Mangkurat

63 PUBLICATIONS 158 CITATIONS

[SEE PROFILE](#)

Some of the authors of this publication are also working on these related projects:



DNA Sequence Classification [View project](#)



Effect of features Generated from additional segments in protein sequence classification [View project](#)



INDONESIA  
.NET DEVELOPER COMMUNITY



Seri Belajar  
**ASP.NET**  
ASP.NET Core MVC & MySQL  
dengan Visual Studio Code  
M Reza Faisal

# **Kata Pengantar**

Puji dan syukur diucapkan kepada Allah SWT atas selesainya buku sederhana yang berjudul Seri Belajar ASP.NET: ASP.NET Core MVC & MySQL dengan Visual Studio Code.

Framework ASP.NET Core adalah versi terbaru dari framework ASP.NET. Kelebihan utama framework ASP.NET Core adalah multiplatform, yaitu dapat digunakan untuk membangun aplikasi web yang dapat dideploy pada berbagai sistem operasi seperti MS Windows, Linux dan Mac OS X. Pada buku ini akan dipaparkan dasar-dasar pemrograman web dengan menggunakan framework ASP.NET Core MVC dan bahasa pemrograman C#. Untuk memberikan pemahaman yang lebih baik maka akan dibuat proyek membangun aplikasi web Book Store yang berfungsi untuk mengelola buku. Aplikasi web ini menggunakan database MySQL dan tool development yang akan digunakan adalah Visual Studio Code. Tool development ini merupakan tool development multi platform yang dapat digunakan pada sistem operasi Windows, Linux dan Mac OS X.

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, Mei 2017

M Reza Faisal

([reza.faisal@gmail.com](mailto:reza.faisal@gmail.com))

# ***Daftar Isi***

<i>Kata Pengantar</i> .....	I
<i>Daftar Isi</i> .....	II
<i>Daftar Gambar</i> .....	VI
<b>1 Pendahuluan</b> .....	<b>10</b>
.NET Core.....	10
ASP.NET Core.....	11
Web Server .....	12
Kestrel.....	12
Internet Information Services (IIS).....	12
Visual Studio Code.....	13
MySQL .....	14
Bahan Pendukung.....	14
Buku .....	14
Source Code .....	17
<b>2 .NET Core SDK &amp; Runtime</b> .....	<b>18</b>
Installasi.....	18
Windows .....	18
Mac.....	19
Linux.....	20
Uji Coba .....	21
.NET Core Command Line Tool.....	22
Info & Bantuan .....	23
Membuat Project .....	24
Restore .....	26
Build.....	27
Run .....	27
Migrasi Project.....	29
Kesimpulan .....	30
<b>3 Visual Studio Code &amp; MySQL</b> .....	<b>31</b>
Visual Studio Code.....	31
Installasi .....	31

Antarmuka .....	34
Tool Tambahan.....	35
Membuat Project .....	37
Fitur-Fitur.....	41
<b>MySQL .....</b>	<b>47</b>
MySQL Extension for Visual Studio Code .....	47
Koneksi .....	48
Memilih Database .....	49
Eksekusi Query .....	49
<b>Kesimpulan .....</b>	<b>51</b>
<b>4 Pengenalan ASP.NET Core MVC.....</b>	<b>52</b>
<b>Cara Kerja ASP.NET Core .....</b>	<b>52</b>
<b>File &amp; Folder Utama ASP.NET Core.....</b>	<b>53</b>
Membuat Project ASP.NET Core .....	54
File Utama .....	54
Folder Utama .....	56
Folder ASP.NET Core MVC .....	57
<b>Cara Kerja ASP.NET Core MVC .....</b>	<b>57</b>
Modifikasi File *.csproj.....	57
Modifikasi File Startup.cs .....	58
Controller .....	59
View .....	61
Model.....	63
Catatan.....	66
<b>ASP.NET Core MVC &amp; MySQL.....</b>	<b>67</b>
MySQL Data Core .....	67
MySQL Entity Framework Core .....	77
<b>Kesimpulan .....</b>	<b>83</b>
<b>5 Model-View-Controller .....</b>	<b>84</b>
<b>Persiapan.....</b>	<b>84</b>
Aplikasi Book Store .....	84
Template Aplikasi Web .....	84
Database .....	85
Membuat Project .....	86

<b>Model.....</b>	<b>88</b>
API .....	88
Tipe Class Model.....	89
Display & Format.....	97
Validasi.....	99
Book Store: Class Model & Atribut.....	102
<b>View.....</b>	<b>106</b>
Akses File .....	106
Razor .....	106
Layout & Antarmuka .....	107
Sintaks Dasar Razor .....	114
HTML Helper .....	123
Tag Helper .....	133
Book Store: Komponen View .....	143
<b>Controller.....</b>	<b>167</b>
View Bag .....	169
LINQ .....	170
Book Store: Komponen Controller.....	173
<b>6 Otentikasi dan Otorisasi .....</b>	<b>191</b>
<b>Library Otentikasi &amp; Otorisasi.....</b>	<b>191</b>
ASP.NET Identity.....	191
Cookie Authentication Middleware .....	192
Implementasi .....	192
<b>Persiapan.....</b>	<b>199</b>
Modifikasi File Startup.cs .....	199
Database .....	199
Class Entity Model.....	200
Class Data Context.....	201
<b>Pengelolaan Role &amp; User .....</b>	<b>203</b>
Modifikasi File MasterLayout.cshtml .....	203
Mengelola Role.....	203
Mengelola User .....	209
<b>Implementasi Otentikasi .....</b>	<b>221</b>
Login.....	221
Logout.....	226

<b>Implementasi Otorisasi.....</b>	<b>226</b>
Otorisasi Method Action.....	.226
<b>Demo.....</b>	<b>231</b>
<b>7 Penutup.....</b>	<b>233</b>

# ***Daftar Gambar***

Gambar 1. Arsitektur .NET Core.....	10
Gambar 2. Berikut adalah daftar .NET Core 1.1.....	11
Gambar 3. Proses request-response pada Kestrel. ....	12
Gambar 4. Proses request-response pada IIS.....	13
Gambar 5. <a href="https://code.visualstudio.com">https://code.visualstudio.com</a> .....	14
Gambar 6. Seri ASP.NET: ASP.NET Core 1.0: Installasi & Deployment. ....	15
Gambar 7. Seri Belajar ASP.NET: ASP.NET MVC untuk Pemula. ....	15
Gambar 8. Seri Belajar ASP.NET: Pengenalan ASP.NET Web API. ....	16
Gambar 9. Seri Belajar ASP.NET: Membangun Aplikasi Web Mudah & Cepat.....	16
Gambar 10. Installasi .NET Core SDK. ....	18
Gambar 11. .NET Core runtime.....	19
Gambar 12. Aplikasi .NET Core console - HelloWorldApp.....	22
Gambar 13. Daftar file dan folder project ASP.NET Core Empty.....	25
Gambar 14. Daftar file dan folder project ASP.NET Core Web. ....	25
Gambar 15. Daftar file dan folder project ASP.NET Core Web dengan otentifikasi. ....	26
Gambar 16. dotnet restore. ....	26
Gambar 17. dotnet build. ....	27
Gambar 18. Tampilan aplikasi web ASP.NET Core Empty.....	28
Gambar 19. Aplikasi ASP.NET Core Web.....	28
Gambar 20. Aplikasi ASP.NET Core Web dengan fitur otentifikasi.....	29
Gambar 21. Web Visual Studio Code. ....	31
Gambar 22. Mac OS - Visual Studio Code.....	32
Gambar 23. Mac OS - Visual Studio Code setelah dijalankan. ....	32
Gambar 24. Proses installasi Visual Studio Code pada Linux Ubuntu. ....	33
Gambar 25. Menjalankan Visual Studio Code pada Linux Ubuntu. ....	33
Gambar 26. Visual Studio Code pada Linux Ubuntu. ....	34
Gambar 27. Layout dasar antarmuka Visual Studio Code. ....	34
Gambar 28. Daftar extension. ....	35
Gambar 29. Pencarian extension. ....	36
Gambar 30. Extension berhasil diinstall. ....	36
Gambar 31. Integrated terminal.....	37
Gambar 32. Project helloworld pada Visual Studio Code. ....	38

Gambar 33. Visual Studio – Informasi proses restore pada bagian OUTPUT .....	38
Gambar 34. Proses debug pada Visual Studio Code. ....	39
Gambar 35. Visual Studio Code – Menjalakan aplikasi console. ....	39
Gambar 36. Visual Studio Code – HelloWorldASPNETCore. ....	40
Gambar 37. Aplikasi web HelloWorldASPNETCore pada web browser.....	40
Gambar 38. Fitur - Hover.....	41
Gambar 39. Fitur - Parameter Hint.....	41
Gambar 40. Fitur - IntelliSense. ....	42
Gambar 41. Fitur - Split Editor.....	42
Gambar 42. Fitur - Go to definition.....	43
Gambar 43. Fitur - Error & Warning.....	43
Gambar 44. Fitur - Folding. ....	44
Gambar 45. Fitur - Komentar. ....	44
Gambar 46. Popular extension.....	45
Gambar 47. Fitur - Debugging. ....	46
Gambar 48. Fitur - Debugging - Debug Console.....	46
Gambar 49. Extension vscode-database. ....	47
Gambar 50. Extension vscode-database - Show All Commands. ....	48
Gambar 51. Extension vscode-database - Koneksi ke database. ....	48
Gambar 52. Extension vscode-database - Daftar database. ....	49
Gambar 53. Extension vscode-database - Eksekusi Query. ....	49
Gambar 54. Extension vscode-database - Show Databases. ....	50
Gambar 55. Extension vscode-database - Operasi tabel.....	50
Gambar 56. Extension vscode-database - Insert & Select.....	51
Gambar 57. Cara kerja ASP.NET klasik pada IIS. ....	52
Gambar 58. Cara kerja aplikasi ASP.NET Core.....	53
Gambar 59. Cara kerja aplikasi ASP.NET dengan IIS. ....	53
Gambar 60. Cara kerja Pattern MVC.....	57
Gambar 61. Restore library Microsoft.AspNetCore.Mvc dan Microsoft.AspNetCore.StaticFiles. ....	58
Gambar 62. Tombol New Folder. ....	60
Gambar 63. Tombol New File. ....	60
Gambar 64. Error - File Index.cshtml tidak ditemukan.....	61
Gambar 65. Tampilkan Index.cshtml.....	62
Gambar 66. Tampilan Error.cshtml.....	63

Gambar 67. Antarmuka GuestBook.....	63
Gambar 68. Hasil ketika form diisi dan tombol Send diklik. ....	66
Gambar 69. Catatan 1 tentang ASP.NET Core MVC. ....	66
Gambar 70. Catatan 2 tentang ASP.NET Core MVC. ....	66
Gambar 71. GuestBook - Index.cshtml. ....	76
Gambar 72. GuestBook - Create.cshtml. ....	76
Gambar 73. GuestBook - Index.cshtml dengan data baru. ....	77
Gambar 74. Template admin Gentellela. ....	84
Gambar 75. Tabel BookStore. ....	85
Gambar 76. Book Store - Daftar kategori buku. ....	94
Gambar 77. Book Store - Daftar pengarang buku. ....	94
Gambar 78. Book Store - Daftar pengarang buku. ....	97
Gambar 79. Display atribut model sebagai label pada header tabel. ....	98
Gambar 80. Display atribut model sebagai label pada form input. ....	99
Gambar 81. File dan folder template gentelella. ....	107
Gambar 82. Layout aplikasi. ....	108
Gambar 83. Master layout template gentelella. ....	108
Gambar 84. Razor - Pesan kesalahan. ....	116
Gambar 85. Razor - Penggunaan simbol @@.....	116
Gambar 86. Razor - Ekspresi implisit & eksplisit. ....	118
Gambar 87. Razor - Blok kode. ....	119
Gambar 88. Razor - Blok kode. ....	120
Gambar 89. Razor - Pengulangan. ....	121
Gambar 90. Daftar data tamu. ....	122
Gambar 91. Tampilan form sebelum proses validasi. ....	131
Gambar 92. Tampilan form setelah proses validasi. ....	131
Gambar 93. Antarmuka design form. ....	132
Gambar 94. Contoh antarmuka implementasi tag helper input. ....	139
Gambar 95. Contoh antarmuka implementasi tag helper select. ....	141
Gambar 96. Contoh antarmuka implementasi tag helper select dengan atribut multiple. ....	142
Gambar 97. Book Store - Kategori buku - Index.cshtml. ....	146
Gambar 98. Book Store - Kategori buku - Create.cshtml. ....	148
Gambar 99. Book Store - Kategori buku - Edit.cshtml. ....	150
Gambar 100. Book Store - Pengarang - Index.cshtml. ....	153
Gambar 101. Book Store - Pengarang - Create.cshtml. ....	155

Gambar 102. Book Store - Pengarang - Edit.cshtml.. ..	157
Gambar 103. Book Store - Buku - Index.cshtml.....	161
Gambar 104. Book Store - Buku - Create.cshtml. ....	164
Gambar 105. Book Store - Buku - Edit.cshtml.....	167
Gambar 106. Daftar buku. ....	184
Gambar 107. Form menambah buku. ....	186
Gambar 108. Contoh kasus otentikasi. ....	198
Gambar 109. Tabel users dan roles pada database BookStore. ....	199
Gambar 110. Book Store - Role - Index.cshtml. ....	206
Gambar 111. Book Store - Role - Create.cshtml. ....	207
Gambar 112. Book Store - Role - Edit.cshtml. ....	209
Gambar 113. Daftar role yang ditambahkan.....	209
Gambar 114. Tabel users - password user yang diacak dengan method EncryptString().....	214
Gambar 115. Book Store - User - Index.cshtml. ....	216
Gambar 116. Book Store - User - Create.cshtml.....	218
Gambar 117. Book Store - User - Edit.cshtml.....	220
Gambar 118. Daftar user.....	221
Gambar 119. Book Store - Login.cshtml. ....	223
Gambar 120. Book Store - Logout.....	226
Gambar 121. Book Store - Halaman AccessDenied.cshtml.....	232

# 1

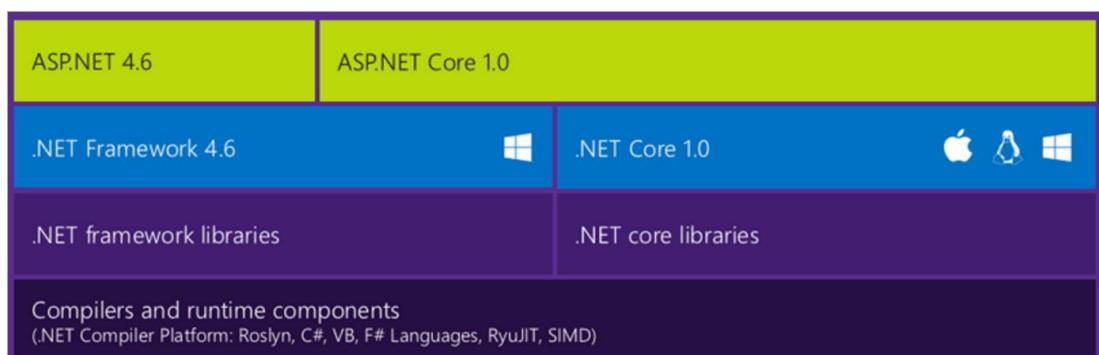
## Pendahuluan

### .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 4.6.2. .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.

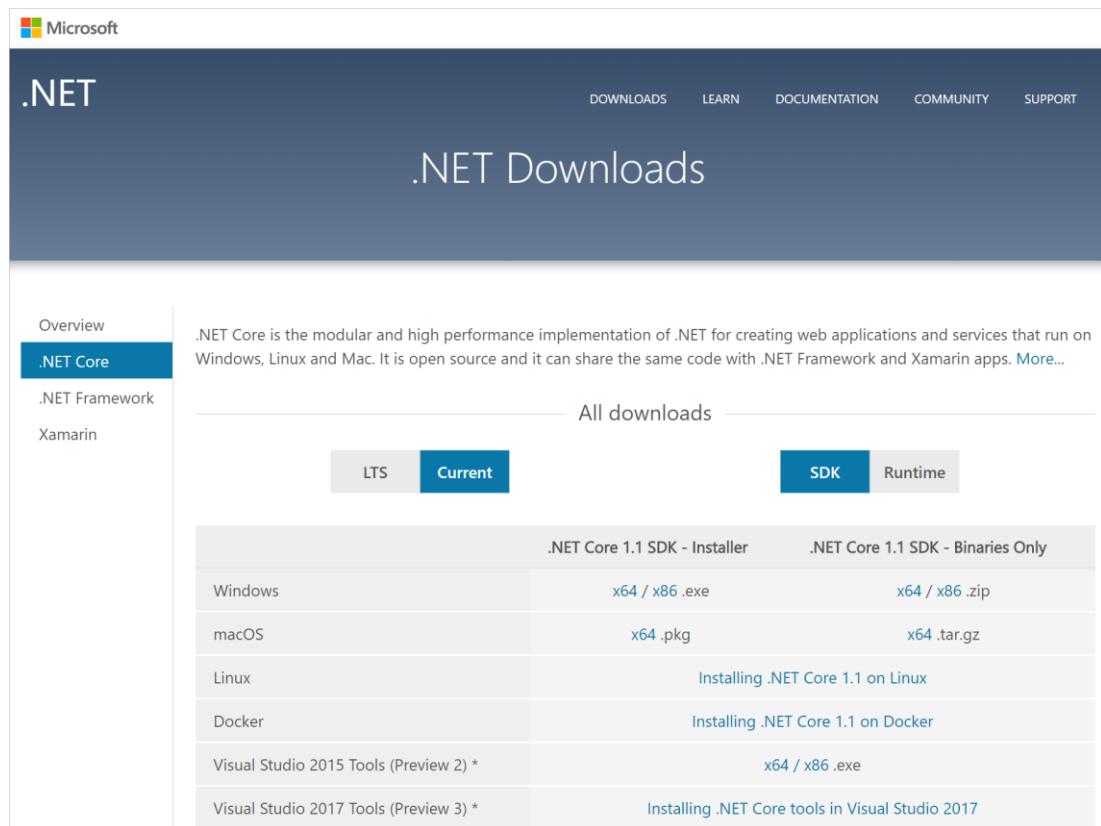
Saat ini Microsoft telah mengembangkan .NET Core, yaitu “.NET Framework” yang bersifat open-source dan multiplatform. Artinya .NET Core dapat dijalankan pada platform Windows, Linux dan Mac OS. Sehingga aplikasi-aplikasi yang dibangun di atas framework ini juga dapat dijalankan di atas banyak platform. Saat ini .NET Core hanya mendukung bahasa pemrograman C# dan F#. Saat buku ini ditulis .NET Core telah mencapai versi 1.1.

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



Gambar 1. Arsitektur .NET Core.

Untuk mendapatkan .NET Core dapat diunduh pada link berikut <https://www.microsoft.com/net/core>. Pada buku akan digunakan .NET Core 1.1 SDK - Installer yang dapat diunduh pada link berikut <https://www.microsoft.com/net/download/core#/current>.



Gambar 2. Berikut adalah daftar .NET Core 1.1.

## 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 1.

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:

1. Aplikasi lebih kecil.
2. Aplikasi menjadi lebih aman.
3. Mengurangi service.
4. 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 4.6 akan ditemui pada framework ini. Saat ASP.NET Core hanya dapat ditulis dengan bahasa pemrograman C# berbeda dengan framework sebelumnya yang memungkinkan menggunakan bahasa pemrograman C# dan VB.NET.

## 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.

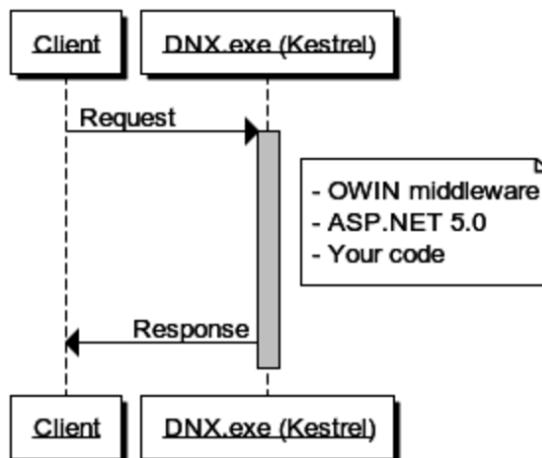
### Kestrel

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.

Kestrel dibangun di atas library berikut ini:

1. libuv adalah library open-source untuk asynchronous event yang digunakan oleh Node.js. Library ini menyediakan asynchronous TCP socket dalam OS-agnostic.
2. SslStream adalah class .NET framework untuk mengubah stream biasa menjadi stream TLS sehingga memungkinkan dukungan HTTPS pada Kestrel.

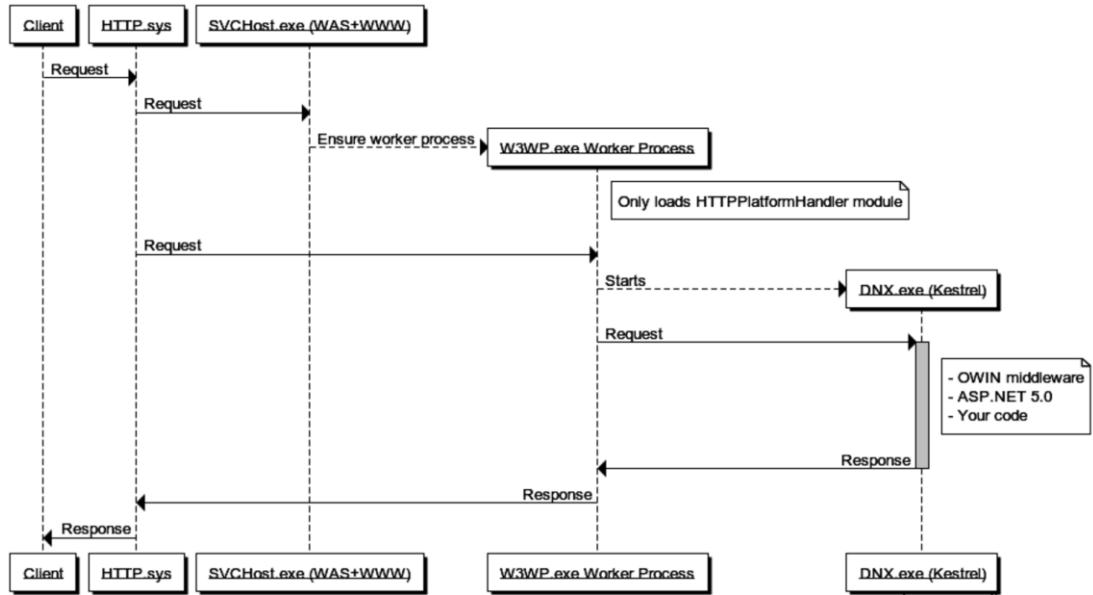
Berikut ini adakan proses request-response pada Kestrel.



Gambar 3. Proses request-response pada Kestrel.

### 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 yang dapat dilihat pada proses request-response di bawah ini.



Gambar 4. Proses request-response pada IIS.

## Visual Studio Code

Visual Studio adalah integrated development environment (IDE) yang dikembangkan oleh Microsoft untuk mempermudah software developer mengembangkan aplikasi pada platform milik Microsoft. Visual Studio 2015 adalah versi stabil terbaru saat buku ini ditulis. Dan sedang dikembangkan Visual Studio 2017. Visual Studio dapat digunakan untuk mengembangkan aplikasi mobile, web, desktop dan cloud. Bahasa yang didukung oleh Visual Studio 2015 adalah Visual Basic, C#, C++, Python, Javascript dan masih banyak lagi. Tetapi Visual Studio 2015 hanya dapat digunakan pada sistem operasi Microsoft Windows.

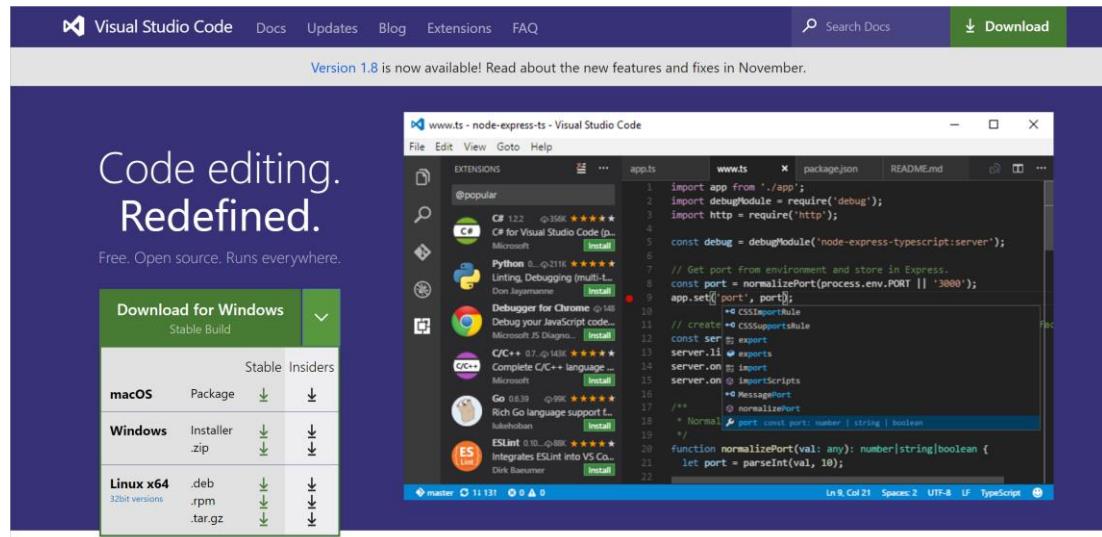
Tetapi saat ini Microsoft telah mengembangkan Visual Studio Code. Visual Studio Code adalah source code editor multiplatform yang dapat digunakan pada sistem operasi Windows, Linux dan Mac OSX. Visual Studio Code juga mendukung banyak bahasa pemrograman seperti halnya Visual Studio 2015 ditambah bahasa pemrograman PHP, Node.js dan lain-lain.

Fitur-fitur utama dari Visual Studio Code adalah:

1. Intelligent code completion, fitur ini akan membantu software developer untuk melengkapi variable, method dan modul yang ditulis.
2. Streamlined debugging, fitur ini berfungsi untuk melakukan debug terhadap kode yang ditulis.
3. Linters, multi-cursor editing, parameter hints.
4. Code navigation.
5. Refactoring.
6. Dukungan akses Git.

Semua fitur-fitur di atas dan fitur-fitur lainnya akan dibahas pada bab selanjutnya

Saat buku ini ditulis, Visual Studio Code telah mencapai versi 1.8. Untuk mengunduh file installer Visual Studio Code dapat mengunjungi link berikut ini <https://code.visualstudio.com/>.



Gambar 5. <https://code.visualstudio.com>.

Dari gambar di atas dapat dilihat Visual Studio Code tersedia untuk:

1. macOS.
2. Windows yang terdiri atas file installer dan zip.
3. Linux x64 dalam format .deb, .rpm, dan .tar.gz.

Visual Studio Code digunakan sebagai code editor yang digunakan untuk membuat aplikasi web yang akan dicontohkan pada buku ini.

## MySQL

MySQL adalah relational database management system (RDBMS). MySQL dikenal sebagai salah satu komponen LAMP open-source web application software stack. LAMP adalah akronim dari Linux, Apache, MySQL dan PHP. MySQL telah banyak dipakai pada banyak produk seperti Wordpress, Joomla, Drupal dan lain-lain.

Saat ini MySQL telah menjadi bagian dari Oracle. Dan MySQL memiliki dua versi yaitu versi Enterprise yang berbayar dan versi Community yang gratis.

Pada buku ini akan digunakan MySQL sebagai database server yang akan menyimpan data dari contoh aplikasi yang akan dibangun pada buku ini.

## Bahan Pendukung

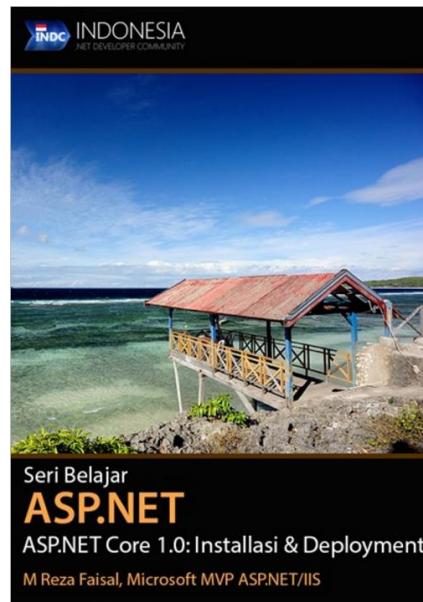
### Buku

1. Seri Belajar ASP.NET: ASP.NET Core 1.0 Installasi & Deployment

Buku ini memberikan detail bagaimana menginstall .NET Core, ASP.NET Core pada berbagai sistem operasi Windows, Linux Mac OSX dan juga Docker. Buku ini juga memberikan langkah-langkah deployment secara detail pada sistem operasi Windows, Linux, Mac OSX dan Docker.

Buku ini dapat diunduh pada Google Play di alamat berikut ini:

[https://play.google.com/store/books/details/M\\_Reza\\_Faisal\\_Seri\\_Belajar\\_ASP\\_NET\\_ASP\\_NET\\_Core\\_1?id=HnsIDAAAQBAJ](https://play.google.com/store/books/details/M_Reza_Faisal_Seri_Belajar_ASP_NET_ASP_NET_Core_1?id=HnsIDAAAQBAJ).



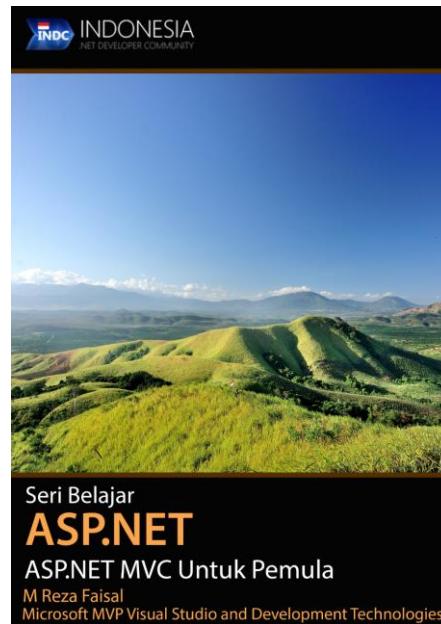
Gambar 6. Seri ASP.NET: ASP.NET Core 1.0: Installasi & Deployment.

2. Seri Belajar ASP.NET: ASP.NET MVC untuk Pemula

Pada buku ini akan dipaparkan apa saja hal-hal yang mesti diketahui web developer dalam memulai membangun aplikasi web dengan ASP.NET MVC dengan bantuan tool Visual Studio 2015. Pada buku ini akan diberikan contoh-contoh yang dapat diikuti oleh pembaca sehingga konsep yang diberikan pada buku ini dapat langsung dicoba. Harapannya hal ini akan membantu web developer untuk cepat memahami paparan pada buku.

Buku ini dapat diunduh pada Google Play di alamat berikut ini:

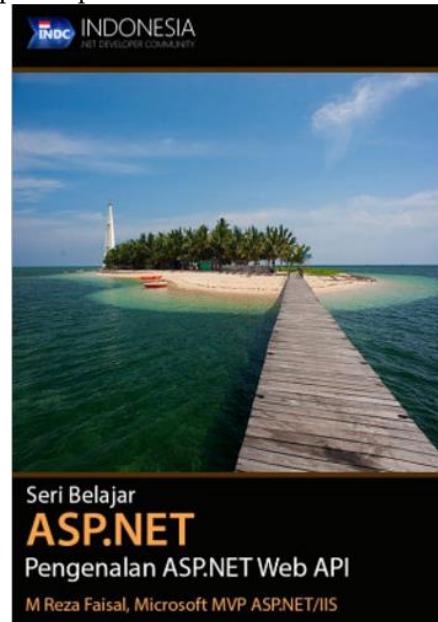
[https://play.google.com/store/books/details/M\\_Reza\\_Faisal\\_Seri\\_Belajar\\_ASP\\_NET\\_ASP\\_NET\\_MVC\\_Unt?id=4vPzDQAAQBAJ](https://play.google.com/store/books/details/M_Reza_Faisal_Seri_Belajar_ASP_NET_ASP_NET_MVC_Unt?id=4vPzDQAAQBAJ)



Gambar 7. Seri Belajar ASP.NET: ASP.NET MVC untuk Pemula.

3. Seri Belajar ASP.NET: Pengenalan ASP.NET Web API.

Pada buku ini akan dikenalkan tentang ASP.NET Web API dengan bantuan tool Visual Studio 2013. Selain itu juga akan diberikan contoh-contoh penggunaannya serta pemanfaatannya pada aplikasi web dan mobile.

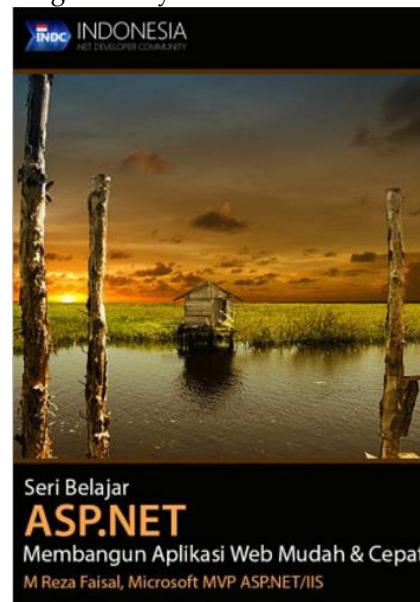


Gambar 8. Seri Belajar ASP.NET: Pengenalan ASP.NET Web API.

Buku ini dapat diunduh pada Google Play di alamat berikut ini:  
[https://play.google.com/store/books/details/M\\_Reza\\_Faisal\\_Seri\\_Belajar\\_ASP\\_NET\\_Pengenalan\\_ASP?id=ewBsCQAAQBAJ](https://play.google.com/store/books/details/M_Reza_Faisal_Seri_Belajar_ASP_NET_Pengenalan_ASP?id=ewBsCQAAQBAJ).

4. Seri Belajar ASP.NET: Membangun Aplikasi Web Mudah & Cepat.

Pada buku terdapat penjelasan dasar tentang pemrograman C# dan pemrograman web dengan menggunakan ASP.NET Web Form. Framework data access yang digunakan pada buku ini adalah LINQ to SQL. Pada buku ini juga dijelaskan tentang ASP.NET Membership sebagai library otentikasi dan otorisasi.



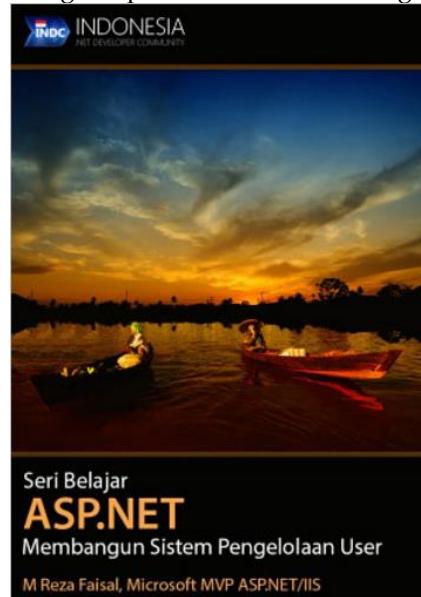
Gambar 9. Seri Belajar ASP.NET: Membangun Aplikasi Web Mudah & Cepat.

Buku ini dapat diunduh pada Google Play di alamat berikut ini:

[https://play.google.com/store/books/details/M\\_Reza\\_Faisal\\_Seri\\_Belajar\\_ASP\\_NET\\_Membangun\\_Aplik?id=m2s-CQAAQBAJ&hl=en](https://play.google.com/store/books/details/M_Reza_Faisal_Seri_Belajar_ASP_NET_Membangun_Aplik?id=m2s-CQAAQBAJ&hl=en).

5. Seri Belajar ASP.NET: Membangun Sistem Pengelolaan User.

Buku ini menjelaskan penggunaan framework ASP.NET Web Form dan ASP.NET Membership untuk membangun aplikasi web Sistem Pengelolaan User.



Buku ini dapat diunduh pada Google Play di alamat berikut ini:

[https://play.google.com/store/books/details/M\\_Reza\\_Faisal\\_Seri\\_Belajar\\_ASP\\_NET\\_Membangun\\_Siste?id=5FI-CQAAQBAI](https://play.google.com/store/books/details/M_Reza_Faisal_Seri_Belajar_ASP_NET_Membangun_Siste?id=5FI-CQAAQBAI).

## Source Code

---

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

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

# 2

## **.NET Core SDK & Runtime**

Pada bab ini akan dijelaskan langkah-langkah untuk melakukan installasi .NET Core dan ASP.NET Core pada sistem operasi Windows, Linux dan Mac OS X.

Pada buku sebelumnya yaitu Seri Belajar ASP.NET: ASP.NET Core 1.0 Installasi & Deployment (lihat Bab 1 Pendahuluan sub bab Bahan Pendukung) telah membahas cara installasi DNVM ( dotnet version manager) yang menjadi dasar lingkungan .NET Core. Tetapi saat ini DNVM sudah tidak digunakan lagi dan digantikan dengan .NET Core SDK. Dibandingkan dengan DNVM, cara isntallasi NET Core SDK lebih mudah.

---

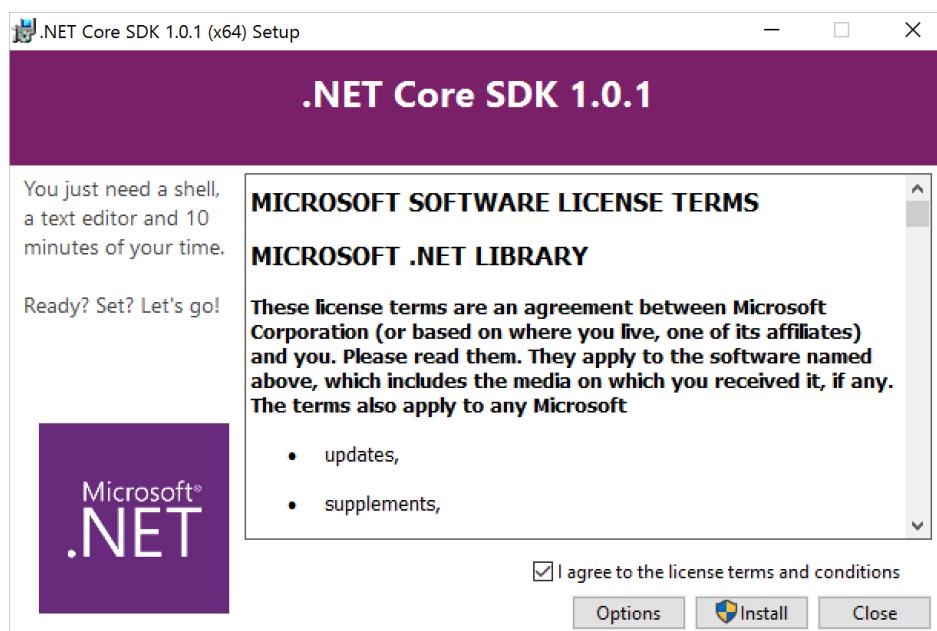
### Installasi

#### **Windows**

---

Untuk installasi pada platform MS Windows digunakan installer dari link berikut ini <https://www.microsoft.com/net/core#windowscmd>. Saat buku ini ditulis versi yang digunakan adalah .NET Core 1.1.1 SDK.

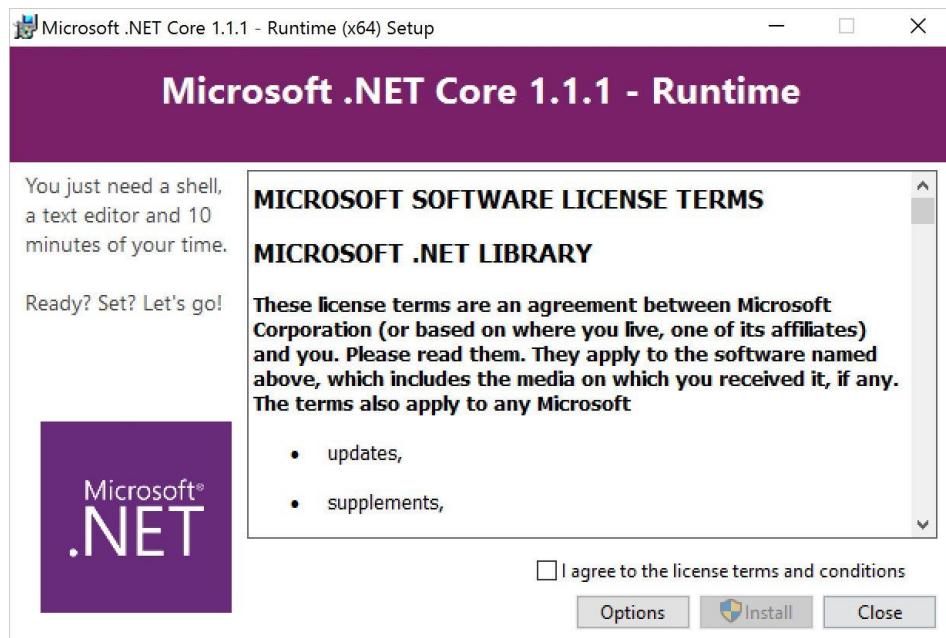
Setelah file dotnet-1.1.1-sdk-win-x64.exe sebesar 103MB selesai diunduh, selanjutnya installasi dapat dilakukan dengan menjalankan file tersebut.



Gambar 10. Installasi .NET Core SDK.

Setelah installasi .NET Core SDK selesai, maka dapat dilanjutkan dengan melakukan installasi .NET Core Runtime. File installer .NET Core Runtime dapat diunduh di link berikut ini <https://www.microsoft.com/net/download/core#/runtime>, pada daftar pilih .NET Core 1.1.1

runtime yang sesuai dengan lingkungan komputer yang digunakan. Sebagai contoh digunakan lingkungan 64 bit maka file installer yang akan digunakan adalah sebagai berikut dotnet-win-x64.1.1.1.exe. File ini berukuran 27MB. Jalankan file tersebut untuk memulai installasi.



Gambar 11. .NET Core runtime.

## Mac

Untuk installasi pada platform Mac OS terlebih dahulu dilakukan installasi OpenSSL terbaru dengan cara menjalankan perintah-perintah berikut ini pada terminal.

```
brew update
brew install openssl
mkdir -p /usr/local/lib
ln -s /usr/local/opt/openssl/lib/libcrypto.1.0.0.dylib /usr/local/lib/
ln -s /usr/local/opt/openssl/lib/libssl.1.0.0.dylib /usr/local/lib/
```

Selanjutnya unduh file .NET Core SDK untuk MacOS pada link berikut ini <https://www.microsoft.com/net/core#macos>.

Nama file installer adalah dotnet-1.1.1-sdk-osx-x64.pkg dengan ukuran file 123MB. Jalankan file installer ini untuk memulai proses installasi.

Langkah selanjutnya adalah menginstall .NET Core runtime untuk MacOS yang filenya dapat diunduh pada link berikut ini <https://www.microsoft.com/net/download/core#/runtime>. File yang diunduh adalah pada bagian macOS (x64) PKG installer dengan nama file dotnet-osx-x64.1.1.1.pkg sebesar 29MB. Jalankan file ini untuk memulai proses installasi.

## Linux

---

Untuk installasi pada platform Linux terdapat berbagai macam cara tergantung dari distro Linux yang digunakan. Berikut adalah perintah-perintah yang harus dijalankan sesuai dengan distro dan versi Linux yang digunakan.

### Ubuntu/Linux Mint

Sebelumnya harus dilakukan terlebih dahulu persiapan dengan menjalankan perintah-perintah berikut ini.

Untuk Ubuntu 14.04/ Linux Mint 17 digunakan perintah berikut ini.

```
sudo sh -c 'echo "deb [arch=amd64] https://apt-mo.trafficmanager.net/repos/dotnet-release/ trusty main" > /etc/apt/sources.list.d/dotnetdev.list'

sudo apt-key adv --keyserver hkp://keyserver.ubuntu.com:80 --recv-keys 417A0893

sudo apt-get update
```

Untuk Ubuntu 16.04 digunakan perintah berikut ini.

```
sudo sh -c 'echo "deb [arch=amd64] https://apt-mo.trafficmanager.net/repos/dotnet-release/ xenial main" > /etc/apt/sources.list.d/dotnetdev.list'

sudo apt-key adv --keyserver hkp://keyserver.ubuntu.com:80 --recv-keys 417A0893

sudo apt-get update
```

Untuk Ubuntu 16.10 digunakan perintah berikut ini.

```
sudo sh -c 'echo "deb [arch=amd64] https://apt-mo.trafficmanager.net/repos/dotnet-release/ yakkety main" > /etc/apt/sources.list.d/dotnetdev.list'

sudo apt-key adv --keyserver hkp://keyserver.ubuntu.com:80 --recv-keys 417A0893

sudo apt-get update
```

Setelah itu dapat untuk semua distro dan versi di atas dapat melanjutkan untuk melakukan installasi .NET SDK dan runtime dengan perintah berikut ini.

```
sudo apt-get install dotnet-dev-1.0.1
```

### Red Hat Enterprise Linux 7 Server

Pada RHEL langkah pertama yang dilakukan adalah persiapan dengan menjalankan perintah-perintah berikut ini.

```
subscription-manager repos --enable=rhel-7-server-dotnet-rpms
yum install scl-utils
```

Selanjutnya untuk installasi .NET Core SDK digunakan perintah berikut ini.

```
yum install rh-dotnetcore11
scl enable rh-dotnetcore11 bash
```

### Debian 8

Untuk instalasi pada Linux Debian 8 digunakan perintah-perintah berikut ini.

```
sudo apt-get install curl libunwind8 gettext
```

```
curl -sSL -o dotnet.tar.gz https://go.microsoft.com/fwlink/?linkid=843453
sudo mkdir -p /opt/dotnet && sudo tar zxf dotnet.tar.gz -C /opt/dotnet
sudo ln -s /opt/dotnet/dotnet /usr/local/bin
```

## Fedora

Untuk Linux Fedora 23 digunakan perintah-perintah berikut ini untuk melakukan persiapan dan proses pengunduhan file binary.

```
sudo dnf install libunwind libicu
curl -sSL -o dotnet.tar.gz https://go.microsoft.com/fwlink/?linkid=843457
```

Sedangkan untuk Linux Fedora 24 digunakan perintah-perintah berikut ini.

```
sudo dnf install libunwind libicu
curl -sSL -o dotnet.tar.gz https://go.microsoft.com/fwlink/?linkid=843461
```

Selanjutnya adalah proses installasi dengan menggunakan perintah-perintah berikut ini.

```
sudo mkdir -p /opt/dotnet && sudo tar zxf dotnet.tar.gz -C /opt/dotnet
sudo ln -s /opt/dotnet/dotnet /usr/local/bin
```

## CentOS & Oracle Linux

Untuk CentOS 7.1 dan Oracle Linux 7.1 digunakan perintah-perintah berikut ini untuk melakukan installasi.

```
sudo yum install libunwind libicu
curl -sSL -o dotnet.tar.gz https://go.microsoft.com/fwlink/?linkid=843449
sudo mkdir -p /opt/dotnet && sudo tar zxf dotnet.tar.gz -C /opt/dotnet
sudo ln -s /opt/dotnet/dotnet /usr/local/bin
```

## OpenSUSE

Langkah pertama adalah persiapan dan proses pengunduhan. Untuk OpenSUSE 13.2 digunakan perintah-perintah berikut ini.

```
sudo zypper install libunwind libicu
curl -sSL -o dotnet.tar.gz https://go.microsoft.com/fwlink/?linkid=843447
```

Sedangkan untuk OpenSUSE 42.1 digunakan perintah-perintah berikut ini.

```
sudo zypper install libunwind libicu
curl -sSL -o dotnet.tar.gz https://go.microsoft.com/fwlink/?linkid=843451
```

Setelah itu dapat dilakukan proses installasi dan konfigurasi PATH dengan perintah-perintah berikut ini.

```
sudo mkdir -p /opt/dotnet && sudo tar zxf dotnet.tar.gz -C /opt/dotnet
sudo ln -s /opt/dotnet/dotnet /usr/local/bin
```

---

## Uji Coba

Untuk mengetahui apakah proses installasi .NET Core SDK dan runtime berhasil, maka dapat dilakukan uji coba berikut ini.

Untuk menguji .NET Core SDK dapat dilakukan dengan perintah berikut ini.

```
dotnet new console -o HelloWorldApp
```

Perintah di atas bertujuan untuk membuat project aplikasi console. Nama project yang dibuat adalah HelloWorldApp. Selanjutnya adalah melakukan proses restore, untuk mengunduh

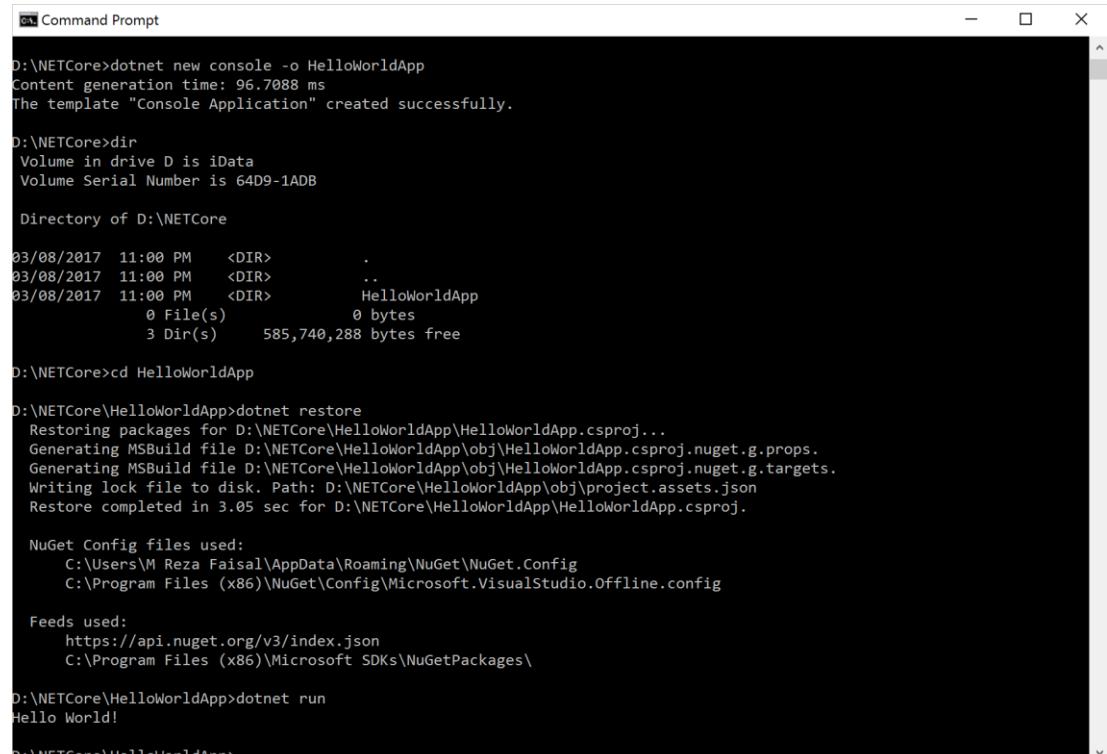
package-package yang diperlukan oleh aplikasi. Perintah yang digunakan untuk proses restore adalah sebagai berikut.

```
dotnet restore
```

Dan untuk menjalankan aplikasi digunakan perintah berikut.

```
dotnet run
```

Dan hasilnya dapat dilihat pada gambar di bawah ini.



```
D:\.NETCore>dotnet new console -o HelloWorldApp
Content generation time: 96.7088 ms
The template "Console Application" created successfully.

D:\.NETCore>dir
 Volume in drive D is iData
 Volume Serial Number is 64D9-1ADB

 Directory of D:\.NETCore

03/08/2017  11:00 PM    <DIR>      .
03/08/2017  11:00 PM    <DIR>      ..
03/08/2017  11:00 PM    <DIR>      HelloWorldApp
          0 File(s)   0 bytes
          3 Dir(s)  585,740,288 bytes free

D:\.NETCore>cd HelloWorldApp

D:\.NETCore\HelloWorldApp>dotnet restore
Restoring packages for D:\.NETCore\HelloWorldApp\HelloWorldApp.csproj...
Generating MSBuild file D:\.NETCore\HelloWorldApp\obj\HelloWorldApp.csproj.nuget.g.props.
Generating MSBuild file D:\.NETCore\HelloWorldApp\obj\HelloWorldApp.csproj.nuget.g.targets.
Writing lock file to disk. Path: D:\.NETCore\HelloWorldApp\obj\project.assets.json
Restore completed in 3.05 sec for D:\.NETCore\HelloWorldApp\HelloWorldApp.csproj.

NuGet Config files used:
  C:\Users\M Reza 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\  
D:\.NETCore\HelloWorldApp>dotnet run
Hello World!

D:\.NETCore\HelloWorldApp>
```

Gambar 12. Aplikasi .NET Core console - HelloWorldApp.

Dengan berhasilnya perintah-perintah di atas, maka dapat diketahui .NET Core SDK dan runtime telah berhasil di install.

## .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.

```
> dotnet --info

.NET Command Line Tools (1.0.0)

Product Information:
Version:          1.0.0
Commit SHA-1 hash: e53429feb4

Runtime Environment:
OS Name:         Windows
OS Version:      10.0.14393
OS Platform:     Windows
RID:             win10-x64
Base Path:       C:\Program Files\dotnet\sdk\1.0.0
```

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.

```
>dotnet --help

.NET Command Line Tools (1.0.0)
Usage: dotnet [host-options] [command] [arguments] [common-options]

Arguments:
[command]           The command to execute
[arguments]         Arguments to pass to the command
[host-options]      Options specific to dotnet (host)
[common-options]    Options common to all commands

Common options:
-v|--verbose        Enable verbose output
-h|--help           Show help

Host options (passed before the command):
-d|--diagnostics   Enable diagnostic output
--version          Display .NET CLI Version Number
--info             Display .NET CLI Info

Commands:
new                Initialize .NET projects.
restore            Restore dependencies specified in the .NET project.
build              Builds a .NET project.
publish            Publishes a .NET project for deployment (including the runtime).
run                Compiles and immediately executes a .NET project.
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.

Project modification commands:
add                Add items to the project
remove             Remove items from the project
list               List items in the project

Advanced Commands:
nuget              Provides additional NuGet commands.
```

<code>msbuild</code>	Runs Microsoft Build Engine (MSBuild).
<code>vstest</code>	Runs Microsoft Test Execution Command Line Tool.

## 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.

```
>dotnet new --help

Template Instantiation Commands for .NET Core CLI.

Usage: dotnet new [arguments] [options]

Arguments:
  template  The template to instantiate.

Options:
  -l|--list      List templates containing the specified name.
  -lang|--language Specifies the language of the template to create
  -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.
  -h|--help       Displays help for this command.
  -all|--show-all Shows all templates

Templates          Short Name    Language    Tags
-----
Console Application  console      [C#], F#   Common/Console
Class library        classlib    [C#], F#   Common/Library
Unit Test Project   mstest      [C#], F#   Test/MSTest
xUnit Test Project  xunit       [C#], F#   Test/xUnit
ASP.NET Core Empty   web         [C#]      Web/Empty
ASP.NET Core Web App mvc        [C#], F#   Web/MVC
ASP.NET Core Web API webapi     [C#]      Web/WebAPI
Solution File        sln         -         Solution

Examples:
  dotnet new mvc --auth None --framework netcoreapp1.1
  dotnet new console --framework netcoreapp1.1
  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:

1. ASP.NET Core Empty, template ini digunakan untuk membuat aplikasi web kosong.
2. 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
wwwroot	3/9/2017 1:56 PM	File folder	
Program.cs	3/9/2017 1:56 PM	Visual C# Source f...	1 KB
Startup.cs	3/9/2017 1:56 PM	Visual C# Source f...	2 KB
webempty.csproj	3/9/2017 1:56 PM	Visual C# Project f...	1 KB

Gambar 13. 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	3/9/2017 1:59 PM	File folder	
Views	3/9/2017 1:59 PM	File folder	
wwwroot	3/9/2017 1:59 PM	File folder	
.bowerrc	3/9/2017 1:59 PM	BOWERC File	1 KB
appsettings.Development.json	3/9/2017 1:59 PM	JSON File	1 KB
appsettings.json	3/9/2017 1:59 PM	JSON File	1 KB
bower.json	3/9/2017 1:59 PM	JSON File	1 KB
bundleconfig.json	3/9/2017 1:59 PM	JSON File	1 KB
Program.cs	3/9/2017 1:59 PM	Visual C# Source f...	1 KB
Startup.cs	3/9/2017 1:59 PM	Visual C# Source f...	2 KB
webmvc.csproj	3/9/2017 1:59 PM	Visual C# Project f...	1 KB

Gambar 14. 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 otentikasi 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	3/9/2017 2:15 PM	File folder	
Data	3/9/2017 2:15 PM	File folder	
Models	3/9/2017 2:15 PM	File folder	
Services	3/9/2017 2:15 PM	File folder	
Views	3/9/2017 2:15 PM	File folder	
wwwroot	3/9/2017 2:15 PM	File folder	
.bowerrc	3/9/2017 2:15 PM	BOWERRC File	1 KB
appsettings.Development.json	3/9/2017 2:15 PM	JSON File	1 KB
appsettings.json	3/9/2017 2:15 PM	JSON File	1 KB
bower.json	3/9/2017 2:15 PM	JSON File	1 KB
bundleconfig.json	3/9/2017 2:15 PM	JSON File	1 KB
Program.cs	3/9/2017 2:15 PM	Visual C# Source f...	1 KB
Startup.cs	3/9/2017 2:15 PM	Visual C# Source f...	4 KB
webmvcauth.csproj	3/9/2017 2:15 PM	Visual C# Project f...	2 KB
webmvcauth.db	3/9/2017 2:15 PM	Data Base File	104 KB

Gambar 15. Daftar file dan folder project ASP.NET Core Web dengan otentikasi.

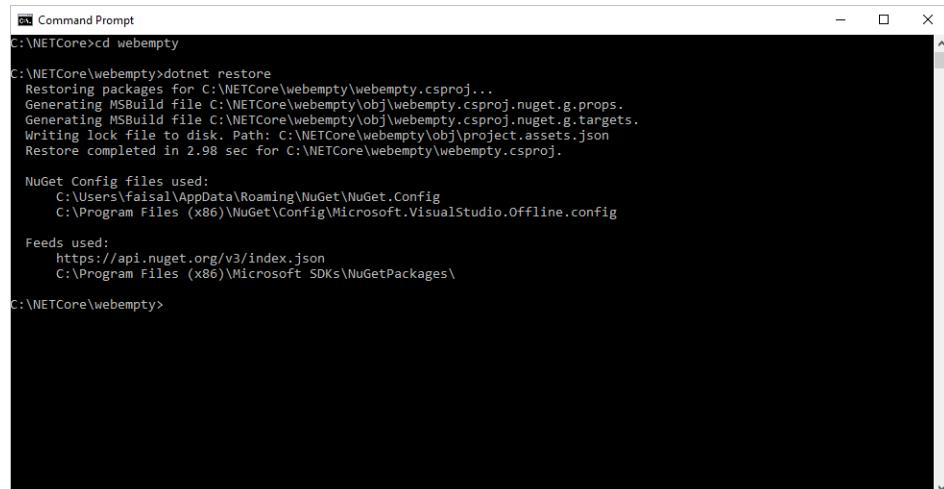
Dari gambar di atas dapat dilihat aplikasi ini memiliki komponen Model, View dan Controller. Selain itu juga dapat dilihat file webmvcauth.db yang merupakan file database SQLite.

## 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.



```
Command Prompt
C:\NETCore>cd webempty
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 16. dotnet restore.

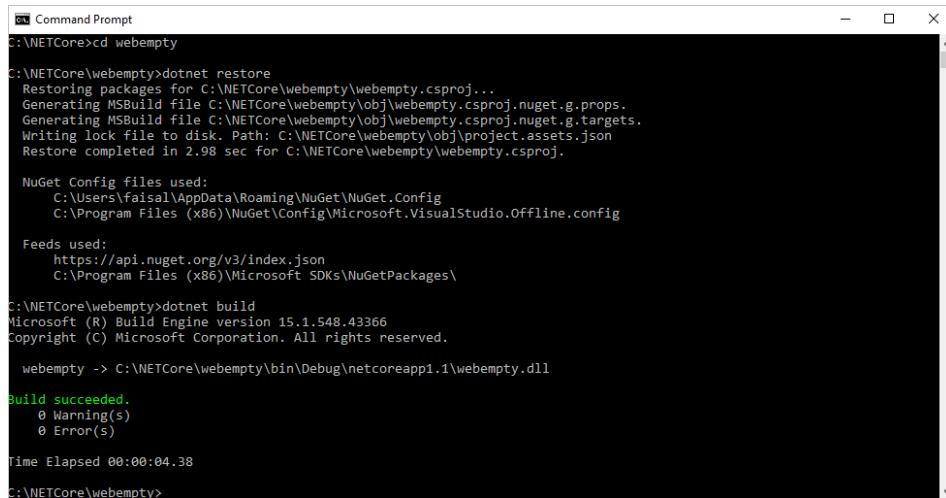
Lakukan proses restore untuk ketiga project yang telah dibuat.

## Build

---

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

```
dotnet build
```



The screenshot shows a Windows Command Prompt window titled "Command Prompt". The working directory is "C:\NETCore\webempty". The user runs "dotnet restore", which restores packages for the project. Then, "dotnet build" is run, which completes successfully with 0 warnings and 0 errors. The total build time is 00:00:04.38. The output also includes information about the Microsoft Build Engine version and the NuGet configuration.

```
C:\NETCore>cd webempty
C:\NETCore\webeempty>dotnet restore
Restoring packages for C:\NETCore\webeempty\webeempty.csproj...
Generating MSBuild file C:\NETCore\webeempty\obj\webeempty.csproj.nuget.g.props.
Generating MSBuild file C:\NETCore\webeempty\obj\webeempty.csproj.nuget.g.targets.
Writing lock file to disk. Path: C:\NETCore\webeempty\obj\project.assets.json
Restore completed in 2.98 sec for C:\NETCore\webeempty\webeempty.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\webeempty>dotnet build
Microsoft (R) Build Engine version 15.1.548.43366
Copyright (C) Microsoft Corporation. All rights reserved.

  webeempty -> C:\NETCore\webeempty\bin\Debug\netcoreapp1.1\webeempty.dll

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

Time Elapsed 00:00:04.38
C:\NETCore\webeempty>
```

Gambar 17. dotnet build.

## Run

---

Untuk menjalankan aplikasi digunakan perintah ini.

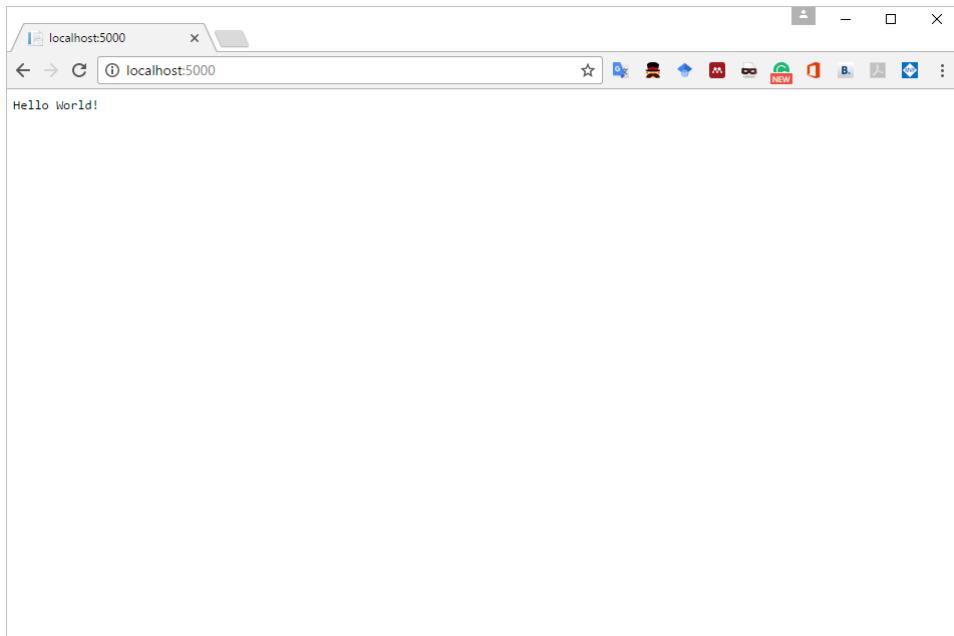
```
dotnet run
```

Hasilnya dapat dilihat pada informasi di bawah ini.

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

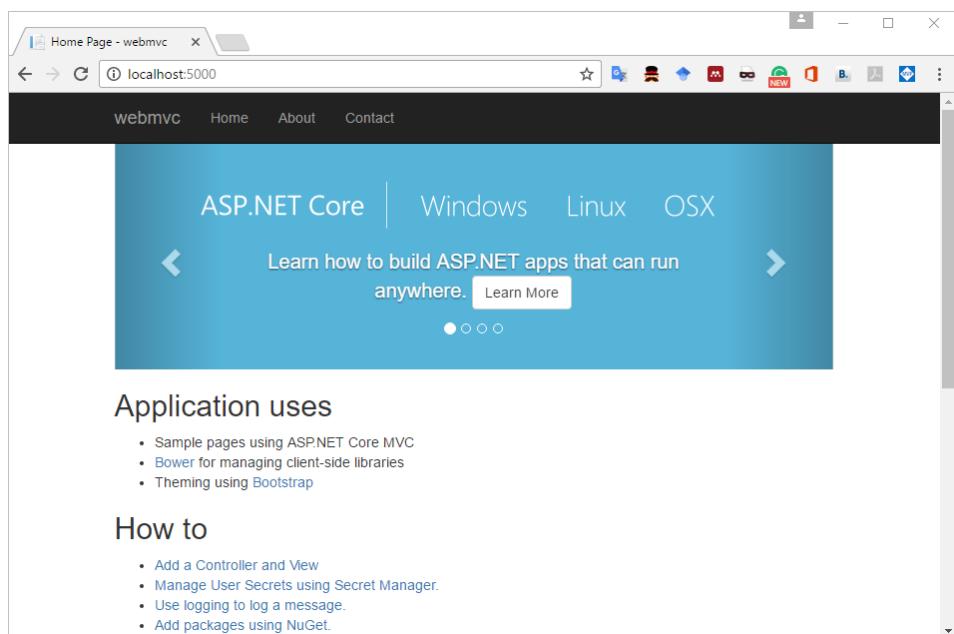
Hosting environment: Production
Content root path: C:\NETCore\webeempty
Now listening on: http://localhost:5000
Application started. Press Ctrl+C to shut down.
```

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.



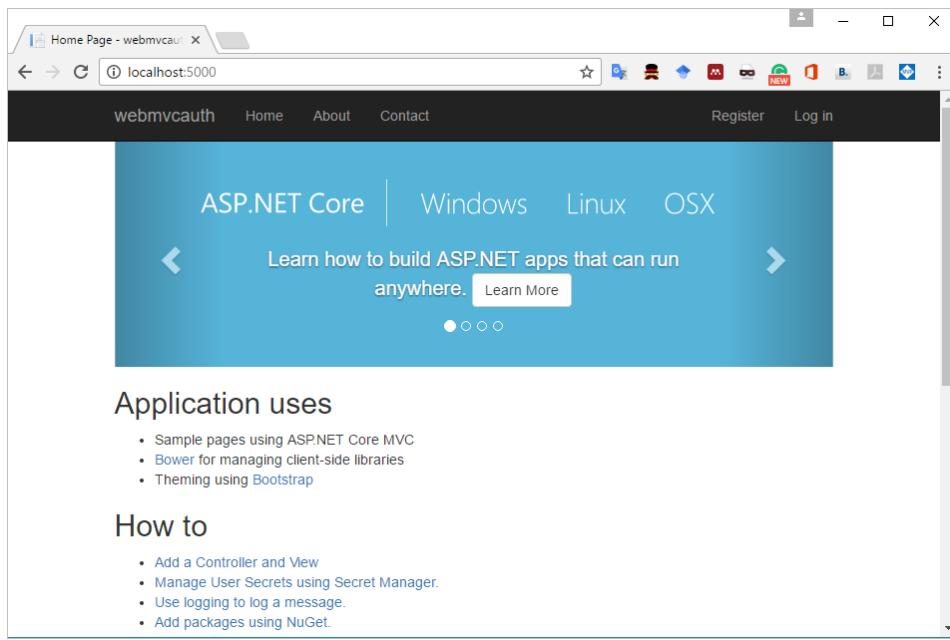
Gambar 18. Tampilan aplikasi web ASP.NET Core Empty.

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 19. 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 20. 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.

---

## **Kesimpulan**

Pada bab ini dijelaskan cara installasi .NET Core dan ASP.NET Core pada berbagai platform hal ini menunjukkan aplikasi yang dibangun dengan framework ini juga dapat dijalankan pada berbagai platform.

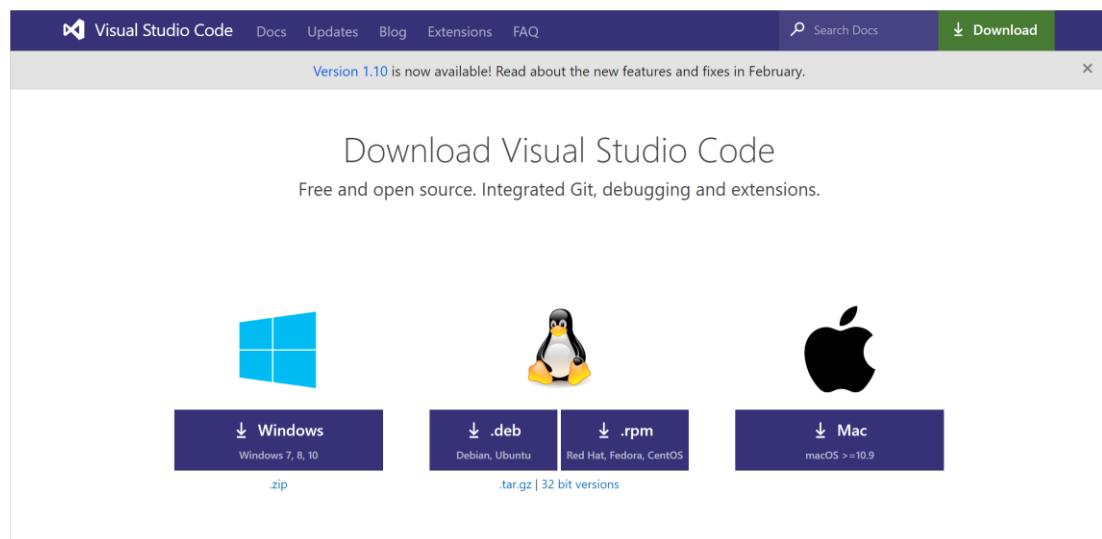
# 3

## ***Visual Studio Code & MySQL***

---

### **Visual Studio Code**

File installer Visual Studio Code dapat diunduh pada link berikut <https://code.visualstudio.com/download>.



Gambar 21. Web Visual Studio Code.

---

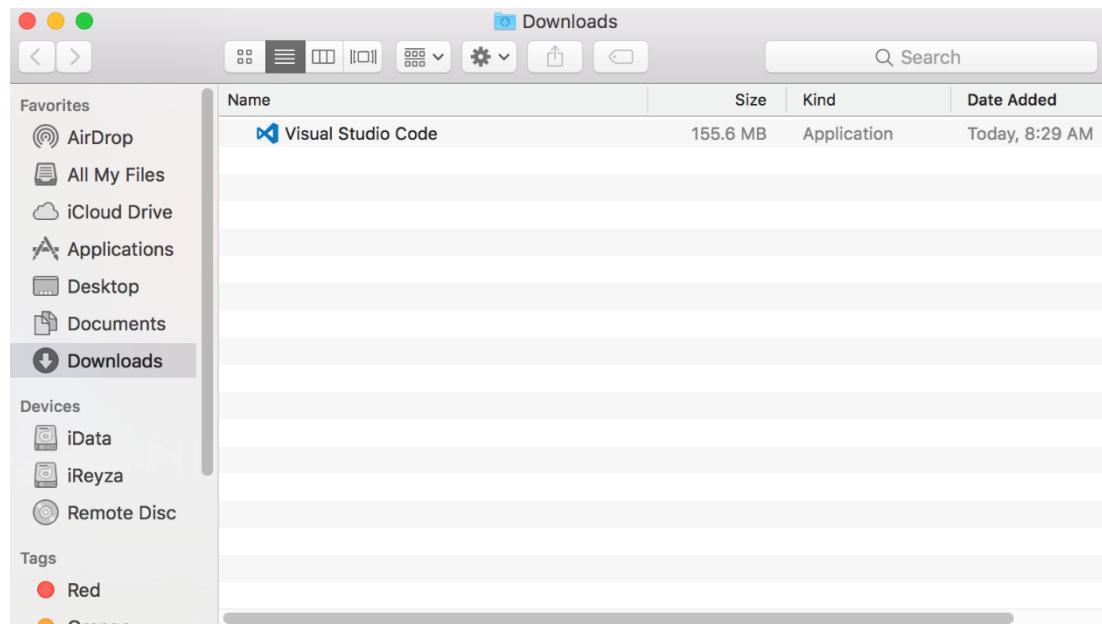
### **Installasi**

#### **Windows**

Untuk platform Windows tersedia dua pilihan file, yaitu installer dan .zip. Untuk file .zip tidak perlu melakukan proses installasi, cukup melakukan proses extract file .zip. Kemudian Visual Studio Code dapat dijalankan dengan mengeksekusi file Code.exe.

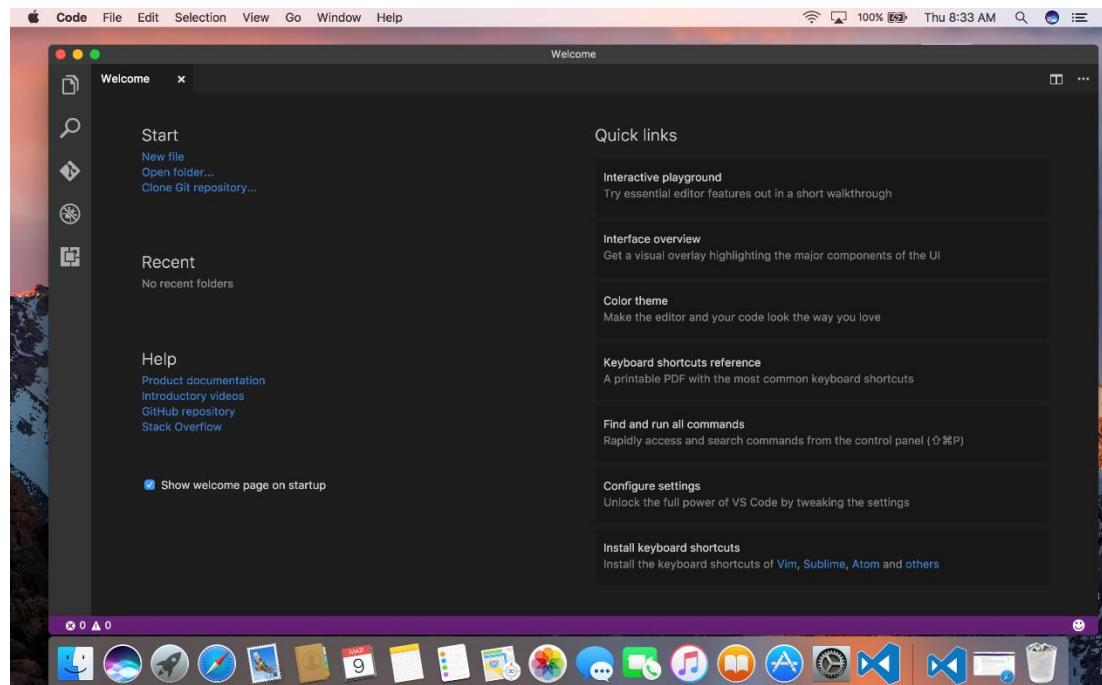
#### **Mac OS**

Untuk platform Mac OS terdapat file zip yang dapat diunduh. Cukup extract file tersebut maka hasilnya adalah sebagai berikut.



Gambar 22. Mac OS - Visual Studio Code.

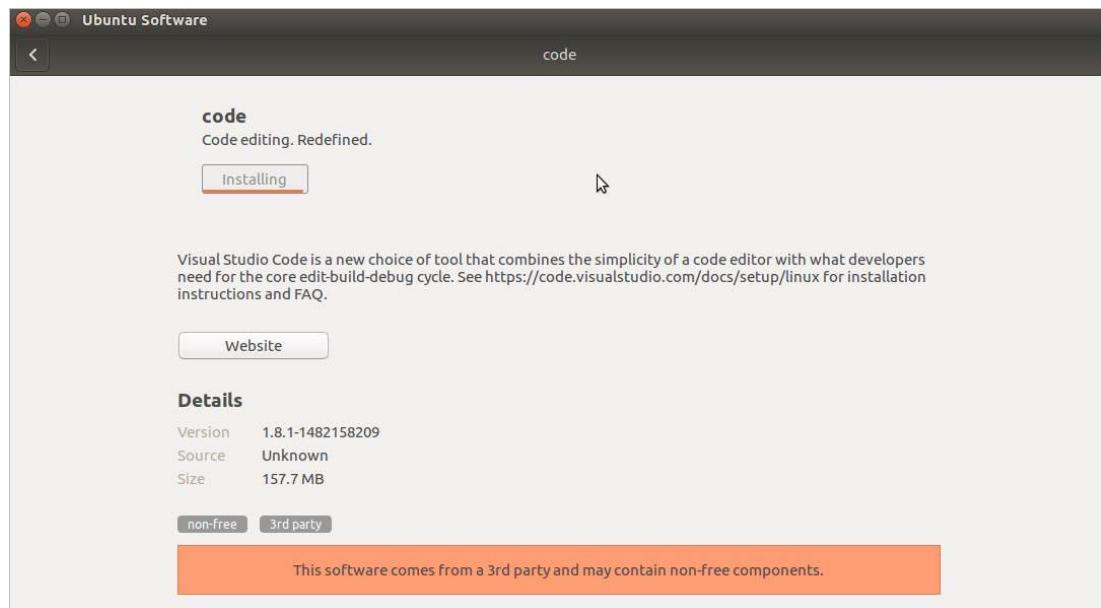
Setelah diklik maka akan dapat dilihat antarmuka sebagai berikut.



Gambar 23. Mac OS - Visual Studio Code setelah dijalankan.

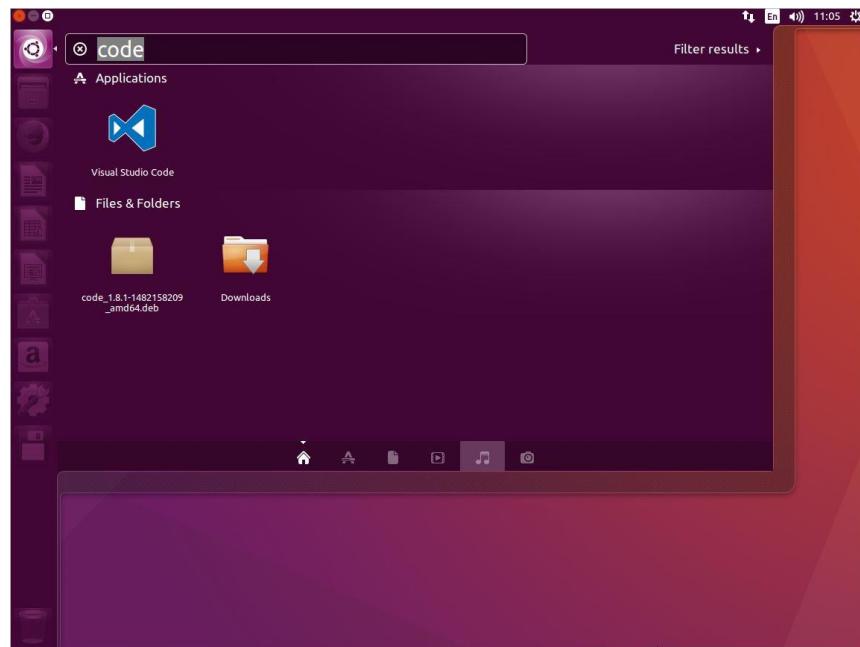
## Linux

Bagi pengguna Linux Ubuntu, installer Visual Studio Code yang digunakan adalah file .deb. Setelah file diunduh, klik dua kali pada file tersebut. Kemudian proses installasi akan dilakukan seperti pada gambar di bawah ini.



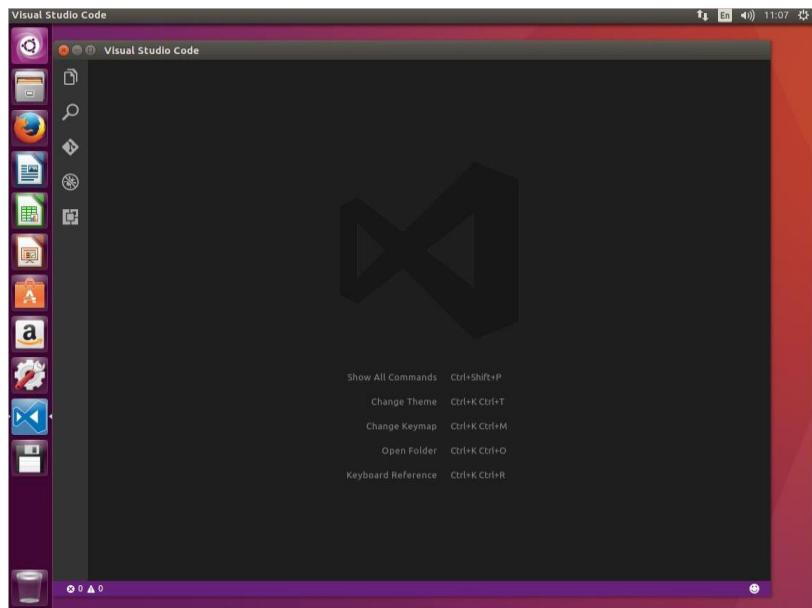
**Gambar 24. Proses installasi Visual Studio Code pada Linux Ubuntu.**

Setelah proses installasi selesai maka untuk menjalankannya dapat dilakukan dengan cara mencari dari fitur “search your computer”, kemudian ketik code pada kolom pencarian. Maka akan ditampilkan item hasil pencarian seperti pada gambar di bawah ini.



**Gambar 25. Menjalankan Visual Studio Code pada Linux Ubuntu.**

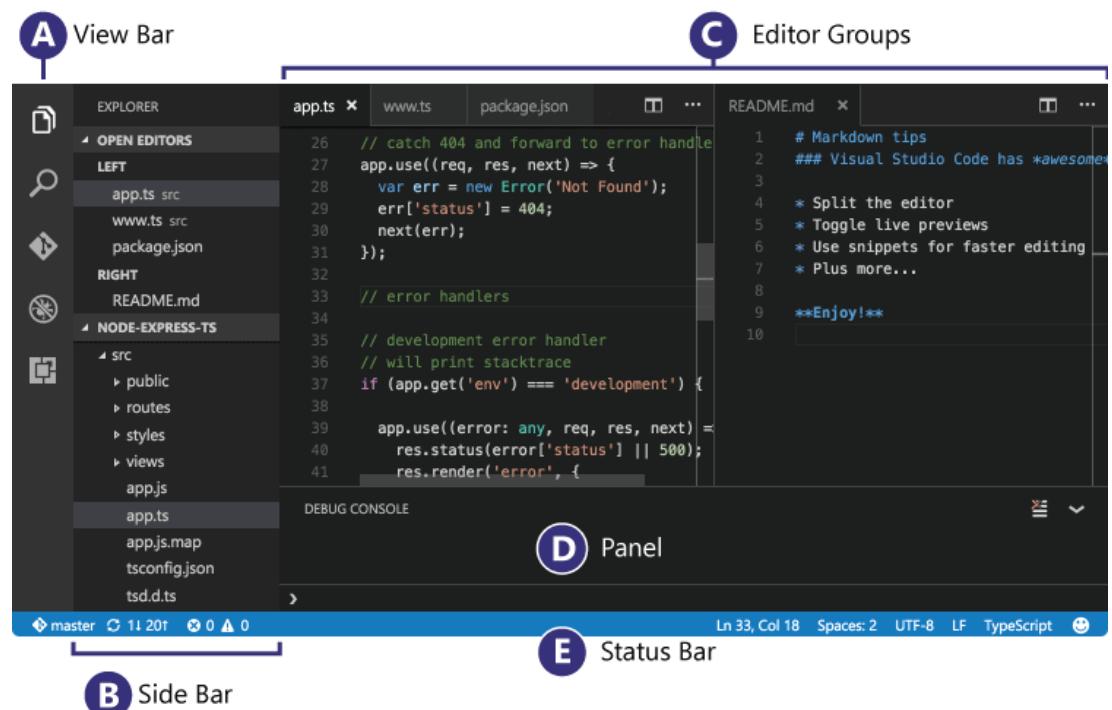
Pada gambar di atas dapat dilihat terdapat Visual Studio Code pada group Applications. Klik icon tersebut untuk menjalankannya.



Gambar 26. Visual Studio Code pada Linux Ubuntu.

## Antarmuka

Pada gambar di bawah ini dapat dilihat layout dasar antarmuka Visual Studio Code.



Gambar 27. Layout dasar antarmuka Visual Studio Code.

Dari gambar di atas dapat dilihat layout terbagi atas 5 bagian yaitu:

- View Bar, berisi tombol-tombol yaitu:
  - Explorer yang berfungsi untuk melihat daftar folder dan file-file didalamnya.
  - Search untuk melakukan pencarian.
  - Git untuk melihat daftar folder yang disimpan pada Git repository.

- o Debug.
  - o Extension untuk melihat daftar extension yang tersedia pada repository.
- b. Side Bar, akan menampilkan view sesuai dengan tombol pada View Bar yang dipilih.
- c. Editor Group, akan menampilkan window-window editor dari file yang dibuka. Artinya jika dibuka 2 file maka akan dapat terlihat 2 windows seperti pada gambar di atas.
- d. Panel akan memberikan informasi output dari hasil proses debug, error, warning dan dapat juga berfungsi sebagai integrated terminal.
- e. Status Bar berfungsi untuk menampilkan informasi dari project dan file yang sedang diedit.

## Tool Tambahan

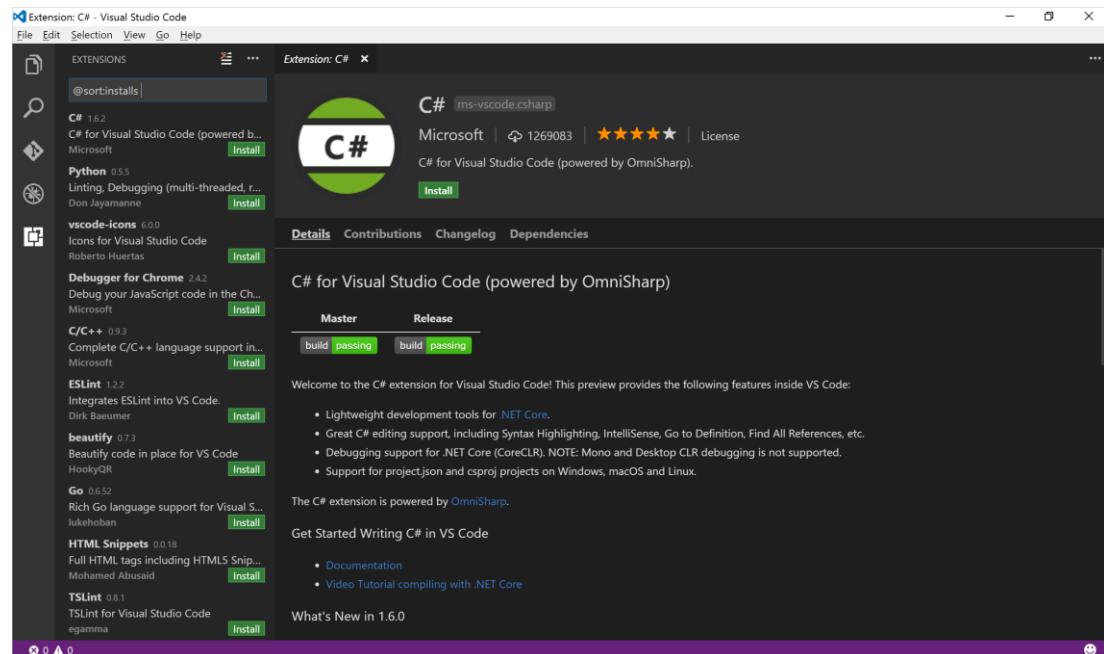
---

Pada sub bab ini akan diterangkan cara untuk menambah tool atau extension yang tersedia pada repository. Akses ke repository memerlukan akses internet. Cara menginstall extension dapat dilakukan dengan dua cara, yaitu dengan menggunakan:

1. Extensions.
2. Integrated Terminal.

### Extension

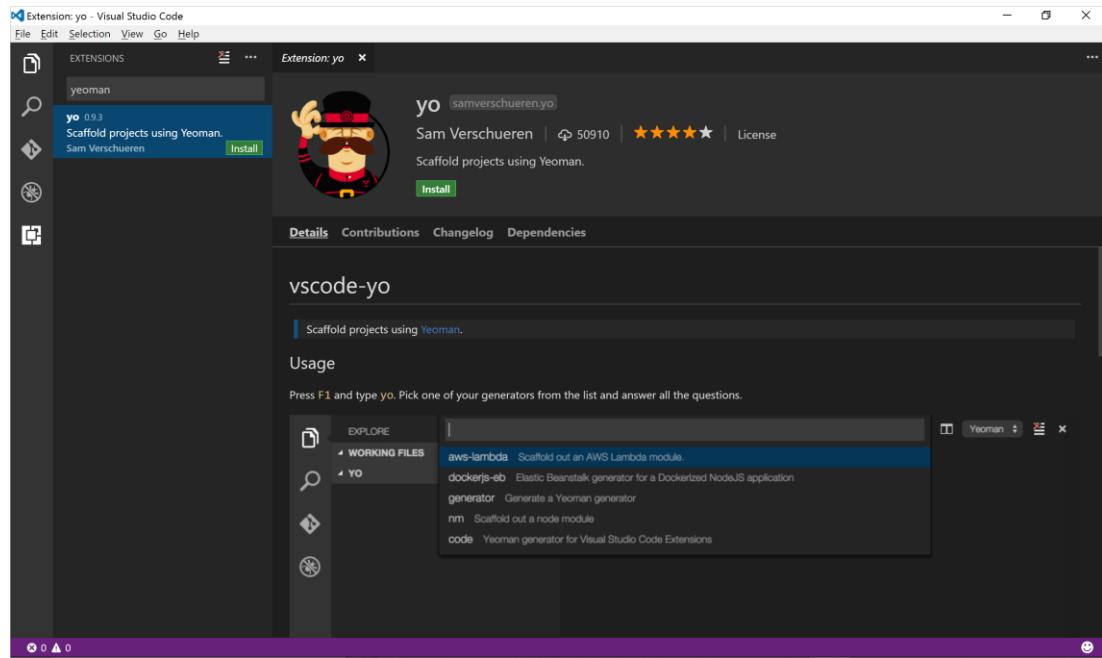
Untuk mengakses daftar extension yang tersedia dapat mengklik tombol Extensions pada View Bar. Berikut adalah tampilan daftar extension.



Gambar 28. Daftar extension.

Klik tombol Install kemudian setelah installasi selesai, klik tombol Reload.

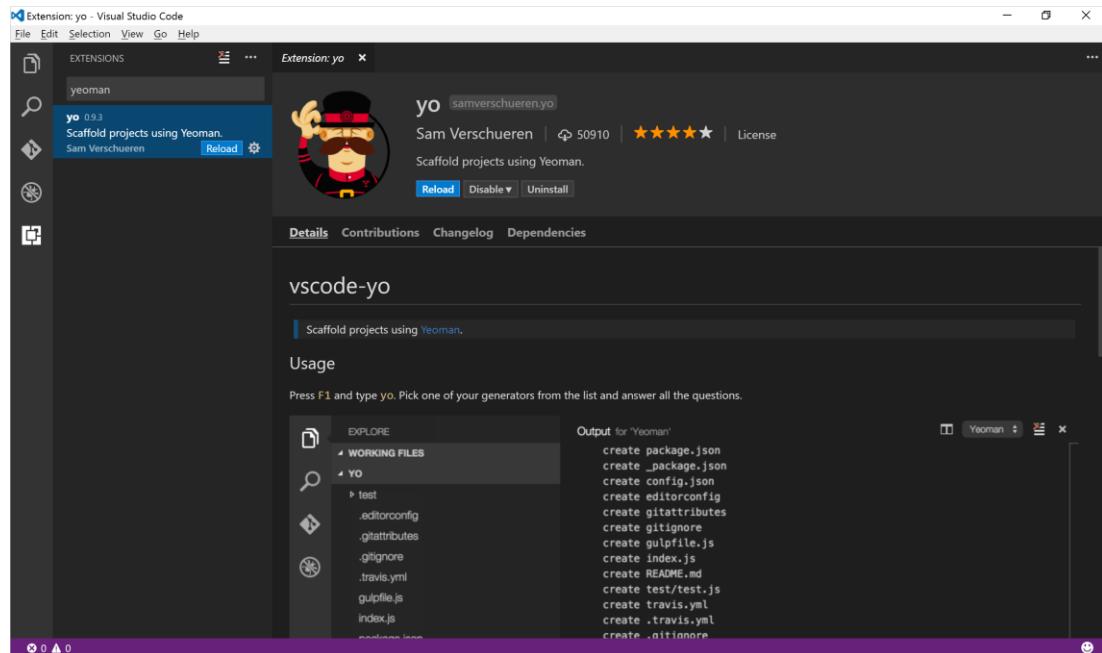
Untuk mencari extension yang diinginkan dapat mengetikkan keyword pada kolom pencarian.



Gambar 29. Pencarian extension.

Sebagai contoh, keyword yang digunakan pada kolom pencarian adalah yeoman. Maka akan ditampilkan extension yang sesuai dengan keyword. Jika ingin melihat detail dari extension maka dapat diklik. Maka detail extension yo dapat dilihat seperti pada gambar. Klik tombol install yang berwarna hijau untuk menginstall extension ini.

Setelah proses installasi selesai maka akan dapat dilihat perubahan status pada tombol install menjadi seperti pada gambar di bawah ini.

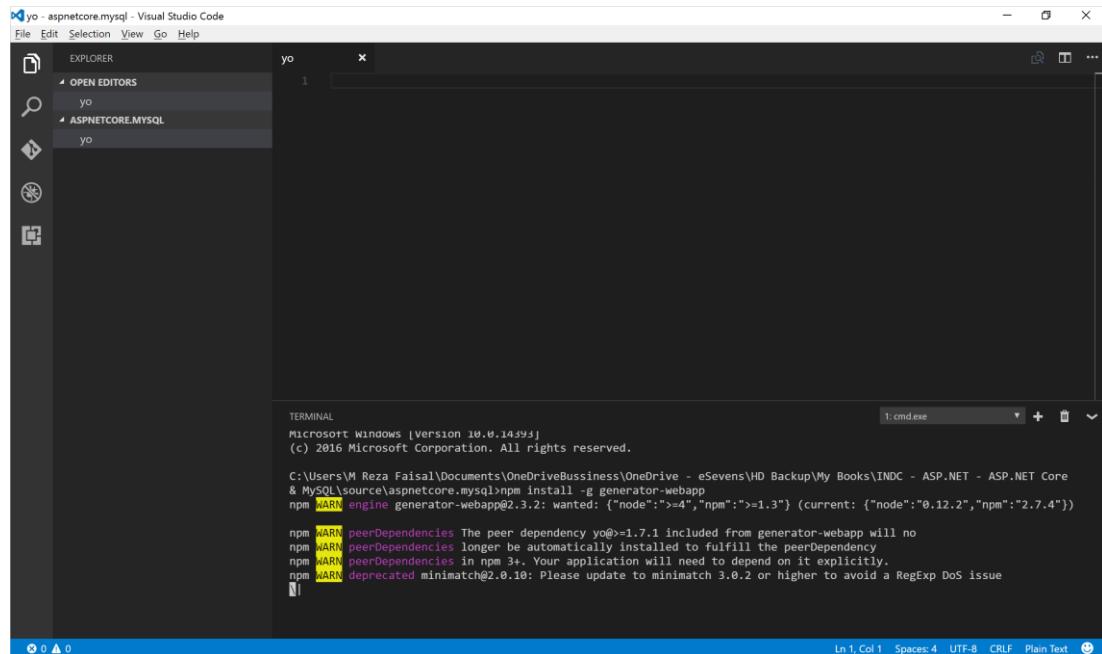


Gambar 30. Extension berhasil diinstall.

Klik tombol Reload untuk mengaktifkan extension ini.

## Integrated Terminal

Cara kedua adalah dengan Terminal. Untuk menampilkan terminal dapat dilakukan dengan cara memilih menu View > Integrated Terminal.



Gambar 31. Integrated terminal.

Integrated Terminal seperti command prompt yang memungkinkan pengguna menuliskan perintah. Perintah yang dapat digunakan untuk menginstall extension adalah npm (nuget package manager).

Sebagai contoh, jika ingin menginstall extension generator-web maka dapat diketik perintah berikut ini pada terminal.

```
npm install -g generator-webapp
```

Maka hasilnya akan dapat dilihat pada gambar di atas.

## Membuat Project

Secara default Visual Studio Code hanya berfungsi untuk membuat project yang telah ada yaitu project yang telah dibuat sebelumnya dengan Visual Studio 2015. Visual Studio Code tidak memiliki feature untuk membuat Project, tidak terdapat menu File > New > Project.

Pada sub bab ini akan diberikan langkah-langkah untuk membuat project baru sehingga bagi pengguna yang tidak memiliki Visual Studio 2015 tetap dapat membangun aplikasi web ASP.NET Core dari awal.

### Project Console HelloWorld

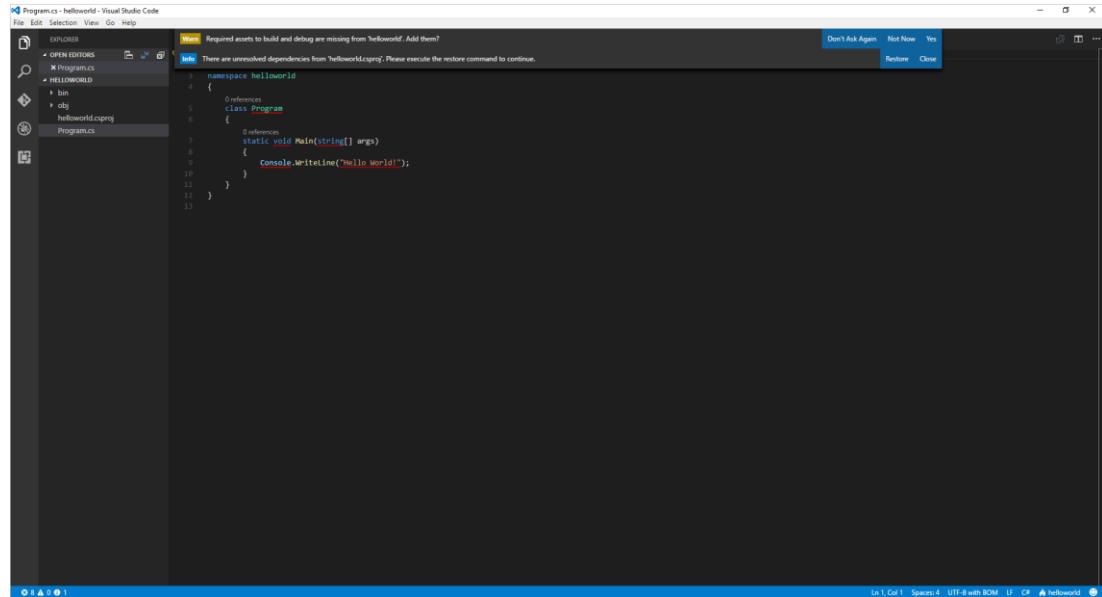
Berikut langkah-langkah untuk membuat project. Nama project yang akan dibuat adalah "helloworld". Langkah pertama adalah membuat folder "helloworld".

```
dotnet new console -o helloworld
```

Hasil dari perintah ini berupa dua file, yaitu:

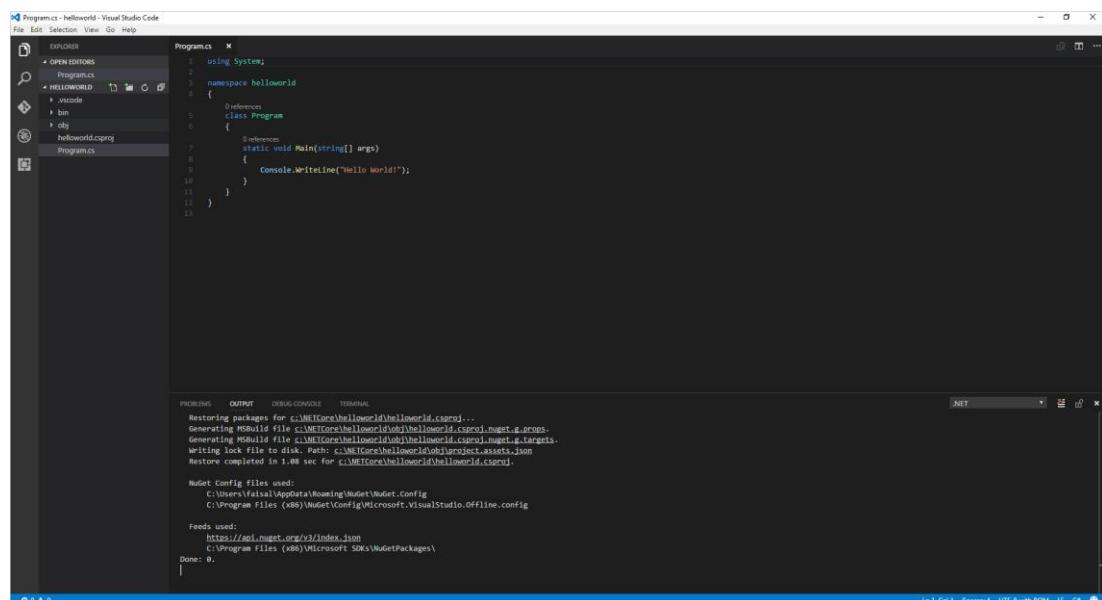
1. Program.cs.
2. helloworld.csproj, file ini adalah file penting yang harus dimiliki oleh sebuah project.

Untuk membuka project helloworld pada Visual Studio Code dapat dilakukan dengan memilih menu File > Open Folder. Kemudian pilih lokasi folder helloworld yang telah dibuat sebelumnya. Kemudian buka file Program.cs maka akan dapat dilihat tampilan seperti pada gambar di bawah ini.



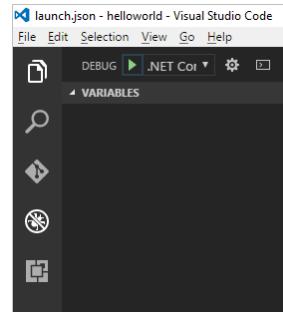
Gambar 32. Project helloworld pada Visual Studio Code.

Akan ditampilkan dialog konfirmasi seperti pada gambar di atas, klik tombol Yes dan Restore. Selanjutnya akan dilakukan proses restore oleh Visual Studio Code.



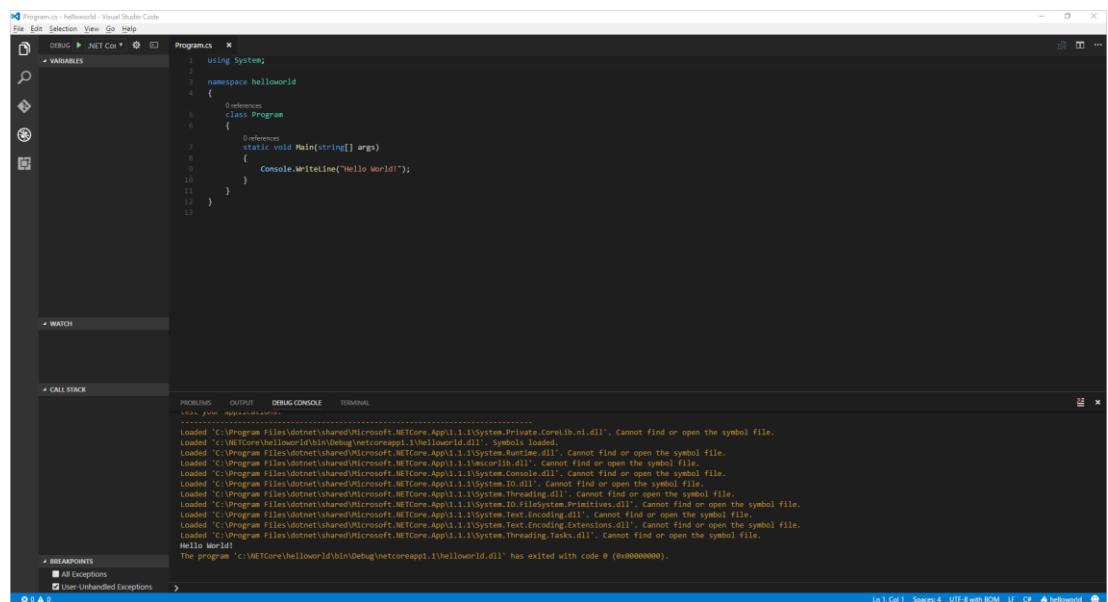
Gambar 33. Visual Studio – Informasi proses restore pada bagian OUTPUT.

Selanjutnya untuk proses debug klik tombol Debug. Kemudian klik tombol segitiga berwarna hijau disamping label DEBUG seperti yang terlihat pada gambar di bawah ini.



Gambar 34. Proses debug pada Visual Studio Code.

Selanjutnya dilakukan proses menjalankan aplikasi. Dan hasilnya dapat dilihat pada gambar di bawah ini.



Gambar 35. Visual Studio Code – Menjalakan aplikasi console.

## Project Web HelloWorldASPNETCore

Langkah pertama adalah membuat project HelloWorldASPNETCore dari template project ASP.NET Core Web App dengan perintah berikut ini.

```
dotnet new mvc -o HelloWorldASPNETCore
```

Kemudian buka project ini dengan menggunakan Visual Studio Code. Berikut dapat dilihat isi file dan folder project ini pada bagian Explorer di Visual Studio Code.

```

Startup.cs
1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Threading.Tasks;
5  using Microsoft.AspNetCore.Builder;
6  using Microsoft.AspNetCore.Hosting;
7  using Microsoft.Extensions.Configuration;
8  using Microsoft.Extensions.DependencyInjection;
9  using Microsoft.Extensions.Logging;
10 
11  namespace HelloWorldASPNETCore
12  {
13      // ...
14      public class Startup
15      {
16          // ...
17          public Startup(IHostingEnvironment env)
18          {
19              var builder = new ConfigurationBuilder()
20                  .SetBasePath(env.ContentRootPath)
21                  .AddJsonFile("appsettings.json", optional: false, reloadOnChange: true)
22                  .AddJsonFile($"appsettings.{env.EnvironmentName}.json", optional: true)
23                  .AddEnvironmentVariables();
24              Configuration = builder.Build();
25          }
26          // ...
27          public IConfiguration Configuration { get; }
28          // This method gets called by the runtime. Use this method to add services to the container.
29      }
30  }
31  // ...
32  2 references
33  public IConfigurationRoot Configuration { get; }
34  // ...
35  // This method gets called by the runtime. Use this method to add services to the container.
36  
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

Restoring packages for c:\NETCore\HelloWorld\HelloWorldASPNETCore\HelloWorldASPNETCore.csproj...
Generating NuGet file c:\NETCore\HelloWorld\HelloWorldASPNETCore\HelloWorldASPNETCore.csproj.nupkg ...
Generating NuGet file c:\NETCore\HelloWorld\HelloWorldASPNETCore\HelloWorldASPNETCore\HelloWorldASPNETCore.csproj.nupkg.g.targets ...
Writing NuGet file to disk. Please go to c:\NETCore\HelloWorld\HelloWorldASPNETCore\HelloWorldASPNETCore\HelloWorldASPNETCore.csproj.
Restore completed in 1.95 sec for c:\NETCore\HelloWorld\HelloWorldASPNETCore\HelloWorldASPNETCore.csproj.

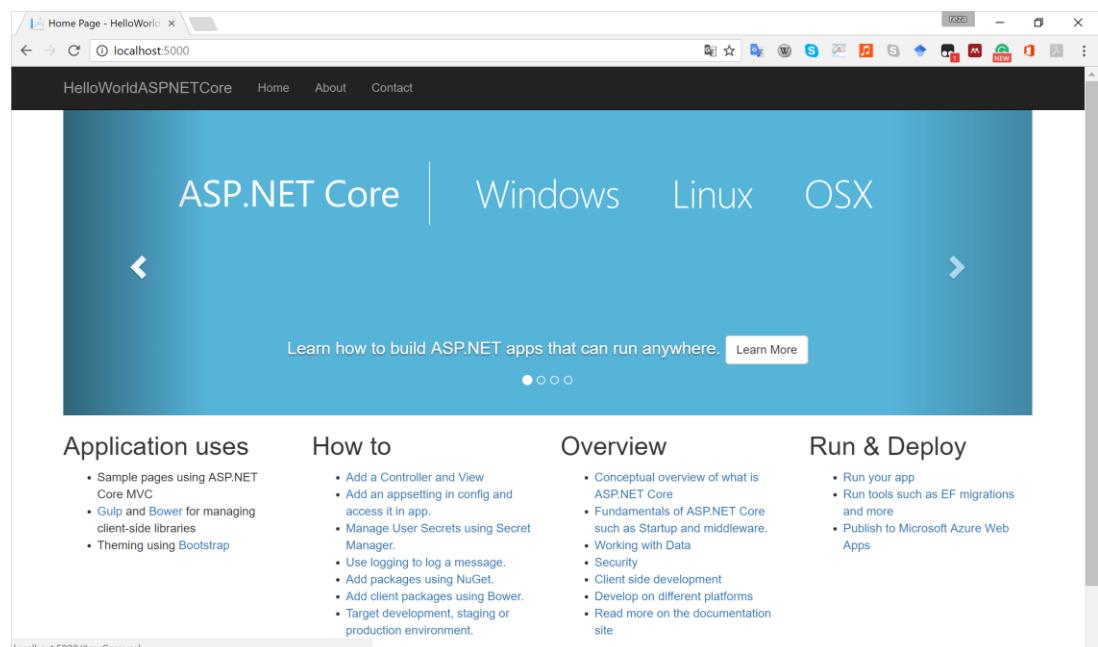
NuGet Config files used:
C:\Users\atul\appdata\Roaming\NuGet\NuGet.Config
C:\Program Files (x86)\NuGet\Config\Microsoft.VisualStudio.Offline.config

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

Done: 9

Gambar 36. Visual Studio Code – HelloWorldASPNETCore.

Selanjutnya untuk menjalakan project ini dapat dilakukan dengan melakukan proses Debug seperti yang telah dicontohkan pada sub bab Project Console HelloWorld. Untuk project aplikasi web, setelah proses build dan run sukses maka secara otomatis Visual Studio Code akan menjalankan Web Browser kemudian secara otomatis mengakses alamat default <http://localhost:5000> seperti pada gambar di bawah ini.



Gambar 37. Aplikasi web HelloWorldASPNETCore pada web browser.

## Fitur-Fitur

---

Berikut adalah fitur-fitur yang dimiliki oleh Visual Studio Code.

### Hover

Fitur ini akan memberikan informasi dari fungsi yang ditunjuk oleh cursor. Sebagai contoh jika cursor diarahkan ke method WriteLine maka akan dapat informasinya seperti gambar di bawah ini.

The screenshot shows a code editor window for 'Program.cs'. The cursor is hovering over the word 'value' in the line 'Console.WriteLine(string value)'. A tooltip appears with the following text: 'Writes the specified string value, followed by the current line terminator, to the standard output stream. value: The value to write. System.IO.IOException: An I/O error occurred.' The code in 'Program.cs' is as follows:

```
1  using System;
2
3  namespace ConsoleApplication
4  {
5      0 references
6      public class Program
7      {
8          0 references
9          public static void Main()
10         {
11             Console.WriteLine("Hello World!");
12         }
13     }
14 }
```

Gambar 38. Fitur - Hover.

### Parameter Hint

Method WriteLine dapat ditulis dengan cara yang berbeda-beda atau dikenal sebagai method overloading. Artinya nama method yang sama tetapi berbeda jumlah parameter dan tipe data yang digunakan pada method tersebut. Visual Studio Code juga dapat memberikan hint sampai level parameter pada kasus method over loading.

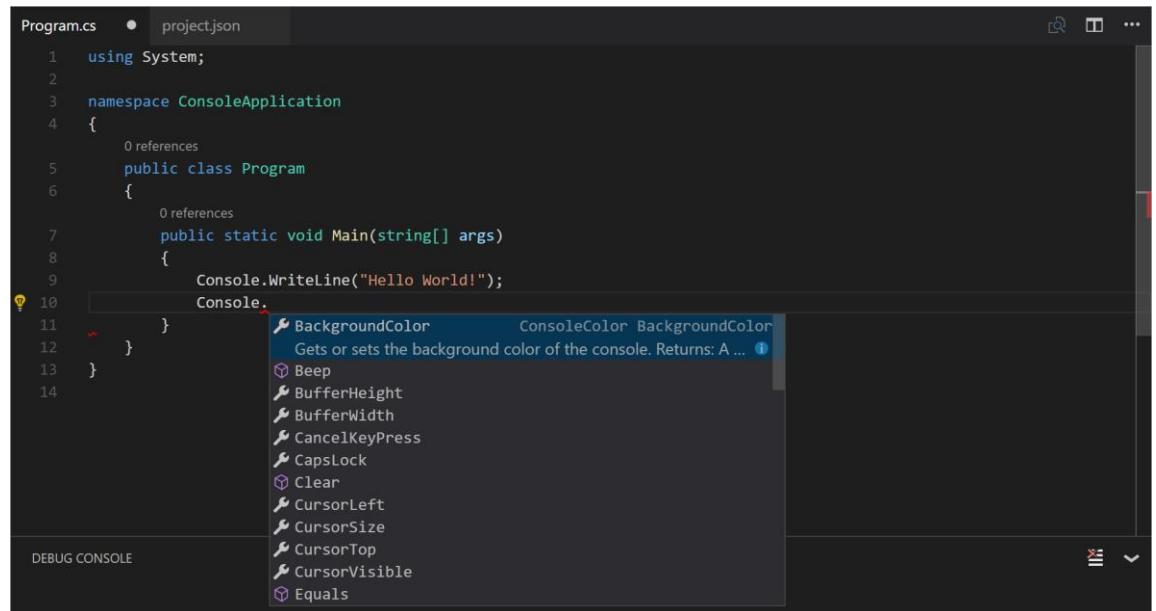
The screenshot shows a code editor window for 'Program.cs'. The cursor is hovering over the parameter 'value' in the line 'Console.WriteLine(ulong value)'. A tooltip appears with the following text: 'Writes the text representation of the specified 64-bit unsigned integer value, followed by the current line terminator, to the standard output stream.' The code in 'Program.cs' is as follows:

```
1  using System;
2
3  namespace ConsoleApplication
4  {
5      0 references
6      public class Program
7      {
8          0 references
9          public static void Main()
10         {
11             Console.WriteLine(
12             Console.WriteLine()
13         )
14     }
15 }
```

Gambar 39. Fitur - Parameter Hint.

## IntelliSense

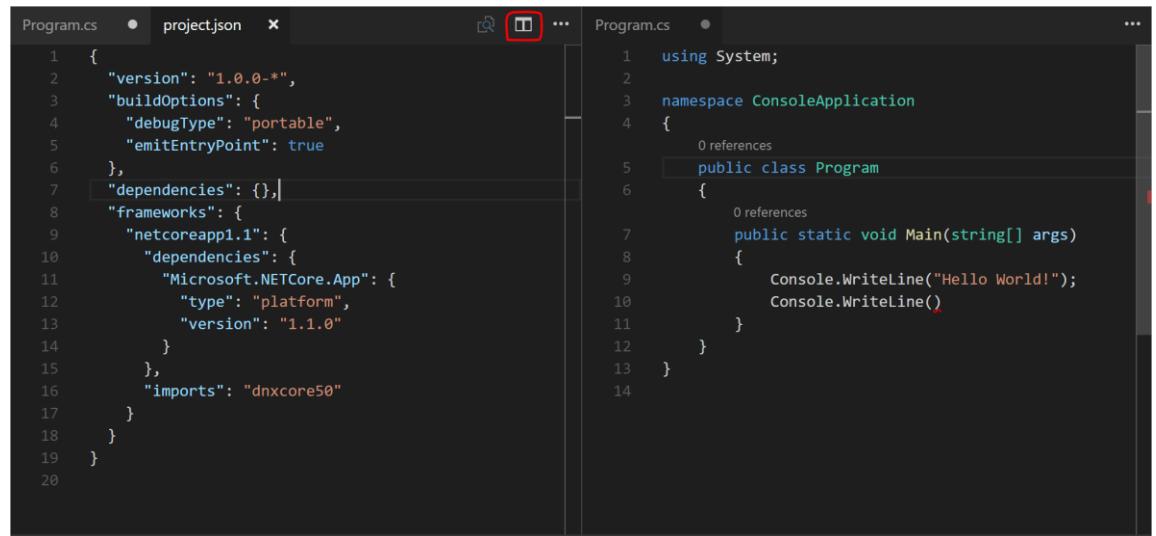
Fitur ini umum ditemui pada setiap editor kode pemrograman. Fungsi fitur ini adalah sebagai membantu melengkapi kode yang sedang ditulis dan memberikan hint.



Gambar 40. Fitur - IntelliSense.

## Split Editor

Dengan mengklik tombol Split Editor (tombol dengan lingkaran merah) maka dapat ditampilkan window editor seperti pada gambar di bawah ini. Fitur ini dapat memudahkan developer untuk membandingkan dua file atau bekerja pada dua file secara bergantian.



Gambar 41. Fitur - Split Editor

## Go to Definition

Fitur ini berfungsi untuk menuju ke source code dari method yang dipilih. Cara untuk menggunakan fitur ini adalah dengan menekan tombol Ctrl dan tahan kemudian klik method yang diinginkan dengan cursor mouse (Ctrl+Click).

Jika ingin membuka source code definition langsung pada window baru maka dapat digunakan kombinasi Ctrl+Alt+Click. Maka hasilnya akan seperti pada gambar di bawah ini.

```
Program.cs • project.json
1 using System;
2
3 namespace ConsoleApplication
4 {
5     public class Program
6     {
7         public static void Main(string[] args)
8         {
9             Console.WriteLine("Hello World!");
10            Console.WriteLine();
11        }
12    }
13 }
14
```

[metadata] Console.cs x

```
499 // T:System.FormatException:
500 //   The format specification in format is invalid.
501 public static void Write(string format, object value);
502 //
503 // Summary:
504 //   Writes the current line terminator to the standard output stream.
505 //
506 // Exceptions:
507 //   T:System.IO.IOException:
508 //     An I/O error occurred.
509 public static void WriteLine();
510 //
511 // Summary:
512 //   Writes the text representation of the specified object to the standard output stream.
513 //   current line terminator, to the standard output stream.
514 //
515 // Parameters:
516 //   value:
517 //     The value to write.
518 //
519 // Exceptions:
520 //   T:System.FormatException:
521 //     The format specification in format is invalid.
522 //   T:System.IO.IOException:
523 //     An I/O error occurred.
```

Gambar 42. Fitur - Go to definition.

## Error & Warning

Fitur ini untuk memberikan informasi jika terjadi kesalahan pada kode yang ditulis. Informasi tentang kesalahan dan peringatan dapat dilihat langsung pada baris yang salah dengan tanda garis merah. Sedangkan rangkuman kesalahan dan peringatan dapat dilihat pada area di bawah editor seperti yang dapat dilihat pada gambar di bawah ini.

```
File Edit Selection View Go Help
```

EXPLORER OPEN EDITORS 1 UNSAVED

- Program.cs
- project.json
- HELLOWORLD**
  - .vscode
  - bin
  - obj
  - Program.cs
  - project.json
  - project.lock.json

Program.cs • project.json

```
1 using System;
2
3 namespace ConsoleApplication
4 {
5     public class Program
6     {
7         public static void Main(string[] args)
8         {
9             Console.WriteLine("Hello World!");
10            console.WriteLine();
11            Console.Wrote();
12        }
13    }
14 }
```

3 ERRORS

Program.cs 3

- The name 'console' does not exist in the current context [netcoreapp1.1] (10, 13)
- 'Console' does not contain a definition for 'Write' [netcoreapp1.1] (11, 21)
- ; expected [netcoreapp1.1] (11, 28)

Gambar 43. Fitur - Error & Warning.

## Folding

Fitur ini berfungsi untuk menutup dan membuka "lipatan" suatu blok pada kode. Pada bahasa pemrograman C# suatu blok adalah bagian yang dibuka dengan tanda "{" dan ditutup dengan tanda "}".

Untuk membuka dan menutup "lipatan" blok dapat menggunakan tombol toggle disamping kiri tanda "{" dan "}" seperti yang terlihat pada gambar di bawah ini.

```
Program.cs    project.json
1  using System;
2
3  namespace ConsoleApplication
4  {
5      public class Program
6      {
7          public static void Main(string[] args)
8          {
9              Console.WriteLine("Hello World!");
10         }
11     }
12 }
```

Gambar 44. Fitur - Folding.

Selain itu juga dapat digunakan tombol shortcut berikut ini:

1. Untuk menutup daerah dimana cursor aktif digunakan kombinasi tombol Ctrl+Shift+[.
2. Untuk membuka daerah dimana cursor aktif digunakan kombinasi tombol Ctrl+Shift+].
3. Untuk menutup seluruh blok dapat digunakan kombinasi tombol Ctrl+K kemudian Ctrl+0.
4. Untuk membuka seluruh blok yang tertutup dapat digunakan kombinasi tombol Ctrl+K kemudian Ctrl+J.

## Komentar

Seperti halnya code editor pada umumnya, Visual Studio Code juga memiliki fitur untuk membuat baris-baris kode menjadi komentar seperti gambar berikut ini. Baris ke-5 sampai ke-11 menjadi komentar dengan cara memilih baris-baris tersebut kemudian tekan kombinasi tombol Ctrl+K+C.

```
Program.cs    project.json
1  using System;
2
3  namespace ConsoleApplication
4  {
5      // public class Program
6      // {
7          // public static void Main(string[] args)
8          // {
9              // Console.WriteLine("Hello World!");
10         }
11     }
12 }
```

Gambar 45. Fitur - Komentar.

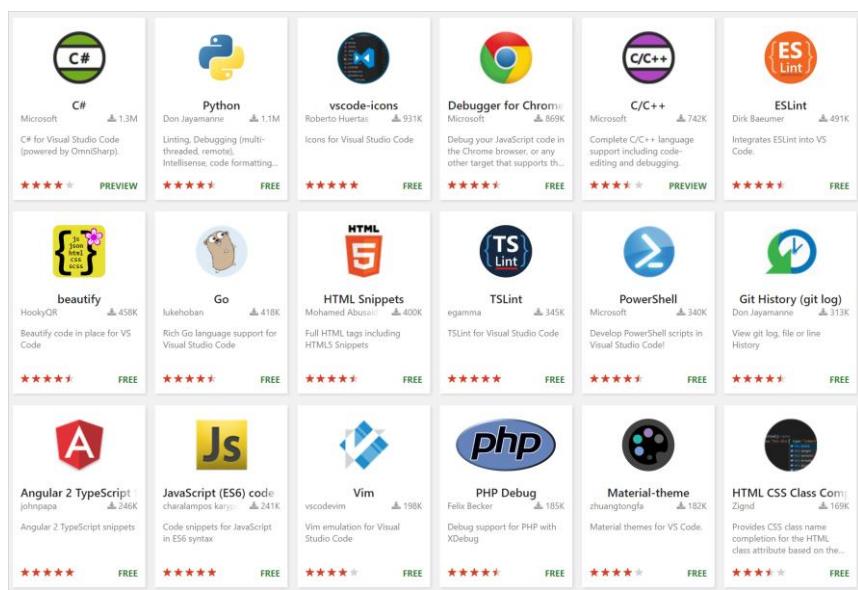
Sedangkan untuk membuka komentar pada baris-baris tersebut dapat dilakukan dengan cara memilih baris-baris tersebut kemudian menekan kombinasi tombol Ctrl+K+U.

## Debugging

Visual Studio Code tidak sekedar kode editor saja. Kelebihan utamanya adalah kemampuan proses debug kode program yang ditulis. Visual Studio Code mampu melakukan debug berbagai macam bahasa pemrograman. Caranya adalah dengan menginstall extension debug untuk bahasa pemrograman yang digunakan.

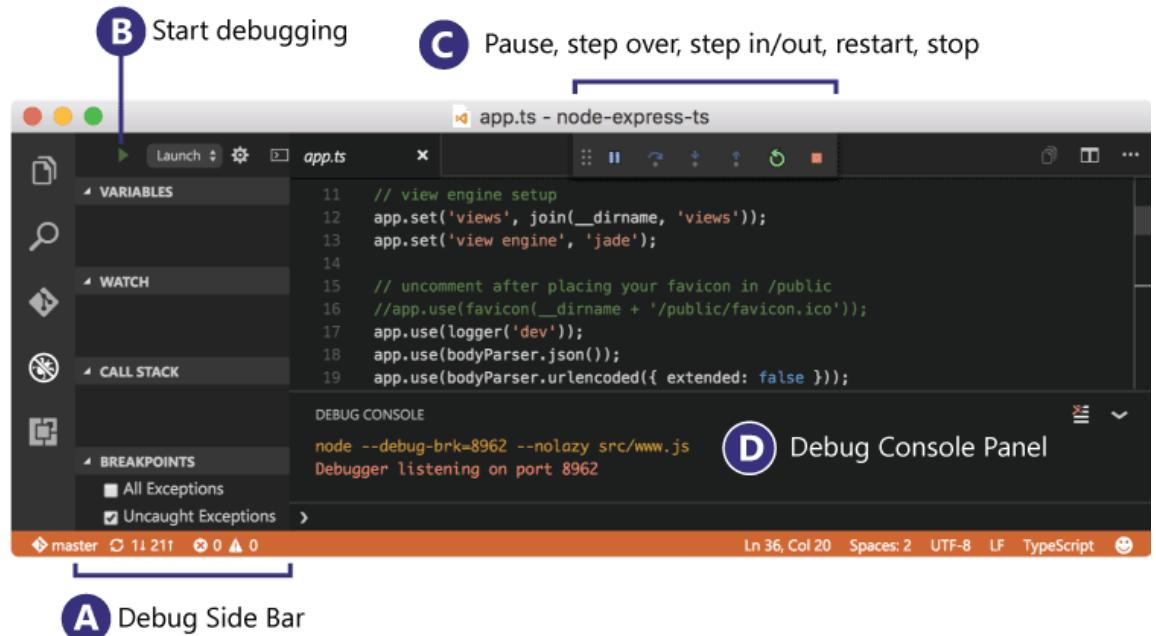
Pada gambar di bawah ini adalah beberapa extension populer berdasarkan jumlah download (<https://marketplace.visualstudio.com/search?target=VSCode&sortBy=Downloads>). Dari sebagain daftar di bawah ini dapat dilihat beberapa extension bahasa pemrograman dan extension lain yang berguna membangun aplikasi web, seperti:

1. C#.
2. Python.
3. C/C++.
4. Go.
5. HTML.
6. PHP.
7. Javascript.
8. AngularJS.
9. Dan lain-lain.



**Gambar 46. Popular extension.**

Saat proses debug dilakukan maka pada Visual Studio Code akan dapat dilihat antarmuka akan terbagi menjadi area-area seperti berikut ini.



Gambar 47. Fitur - Debugging.

Keterangan:

- Debug Side Bar akan ditampilkan ketika tombol Debug diklik.
- Untuk memulai proses debug dapat dilakukan dengan mengklik tombol yang ditunjuk oleh B.
- Saat proses debug sedang berjalan akan dapat dilihat tombol control yang ditunjuk oleh C.
- Debug Console Panel berfungsi untuk menampilkan informasi hasil proses debug. Area ini juga dapat menampilkan output dari program, seperti yang dapat dilihat pada gambar di bawah ini.

The "DEBUG CONSOLE" panel displays the following output:

```

DEBUG CONSOLE
-----
You may only use the Microsoft .NET Core Debugger (clrdbg) with Visual Studio Code, Visual Studio or Visual Studio for Mac software to help you develop and test your applications.
-----
Loaded 'C:\Program Files\dotnet\shared\Microsoft.NETCore.App\1.1.0\System.Private.CoreLib.ni.dll'. Cannot find or open the symbol file.
Loaded 'd:\Data\My Books\INDC - ASP.NET - ASP.NET Core & MySQL\source\helloworld\bin\Debug\netcoreapp1.1\helloworld.dll'. Symbols loaded.
Loaded 'C:\Program Files\dotnet\shared\Microsoft.NETCore.App\1.1.0\System.Runtime.dll'. Cannot find or open the symbol file.
Loaded 'C:\Program Files\dotnet\shared\Microsoft.NETCore.App\1.1.0\mscorlib.dll'. Cannot find or open the symbol file.
Loaded 'C:\Program Files\dotnet\shared\Microsoft.NETCore.App\1.1.0\System.Console.dll'. Cannot find or open the symbol file.
Loaded 'C:\Program Files\dotnet\shared\Microsoft.NETCore.App\1.1.0\System.IO.dll'. Cannot find or open the symbol file.
Loaded 'C:\Program Files\dotnet\shared\Microsoft.NETCore.App\1.1.0\System.Threading.dll'. Cannot find or open the symbol file.
Loaded 'C:\Program Files\dotnet\shared\Microsoft.NETCore.App\1.1.0\System.FileSystem.Primitives.dll'. Cannot find or open the symbol file.
Loaded 'C:\Program Files\dotnet\shared\Microsoft.NETCore.App\1.1.0\System.Text.Encoding.dll'. Cannot find or open the symbol file.
Loaded 'C:\Program Files\dotnet\shared\Microsoft.NETCore.App\1.1.0\System.Text.Encoding.Extensions.dll'. Cannot find or open the symbol file.
Loaded 'C:\Program Files\dotnet\shared\Microsoft.NETCore.App\1.1.0\System.Threading.Tasks.dll'. Cannot find or open the symbol file.
Hello World!
The program 'd:\Data\My Books\INDC - ASP.NET - ASP.NET Core & MySQL\source\helloworld\bin\Debug\netcoreapp1.1\helloworld.dll' has exited with code 0 (0x00000000).

```

Gambar 48. Fitur - Debugging - Debug Console.

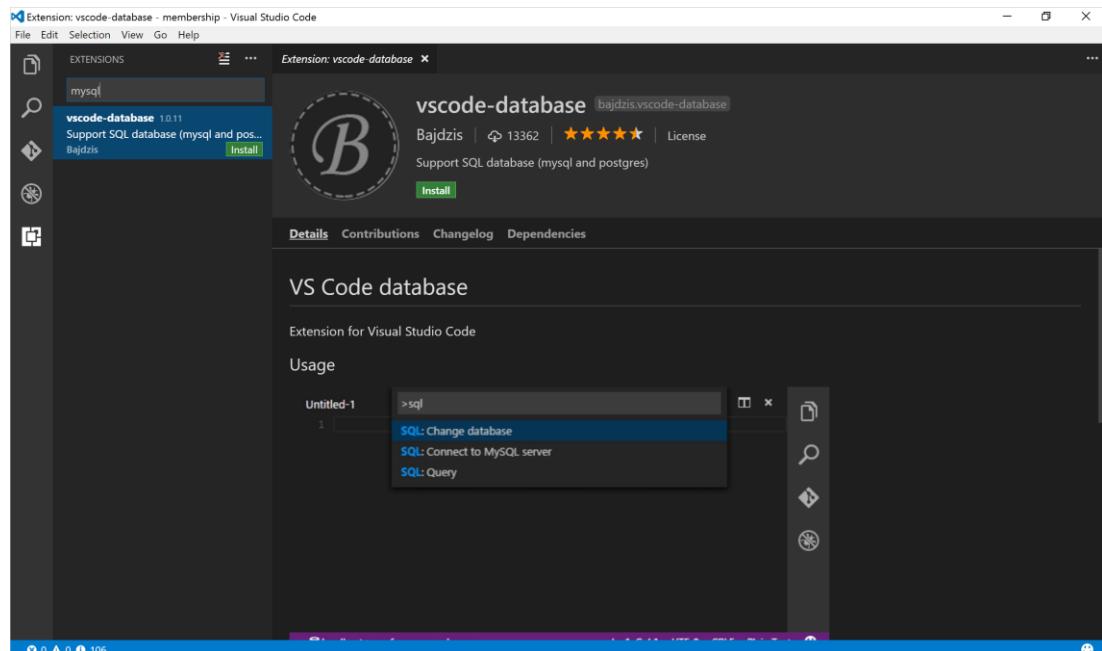
## MySQL

Pembaca diharapkan telah menginstall MySQL Server sebelum memulai bab ini. Karena pada bagian ini tidak akan dijelaskan langkah-langkah menginstall MySQL. Pada bab ini juga tidak akan memberikan tentang dasar-dasar SQL. Diharapkan pembaca telah mengenal dasar-dasar SQL.

### MySQL Extension for Visual Studio Code

MySQL Extension for Visual Studio Code ini adalah MySQL Client yang terintegrasi dengan Visual Studio Code. Sehingga pengguna dapat melakukan mengeksekusi perintah MySQL dan SQL.

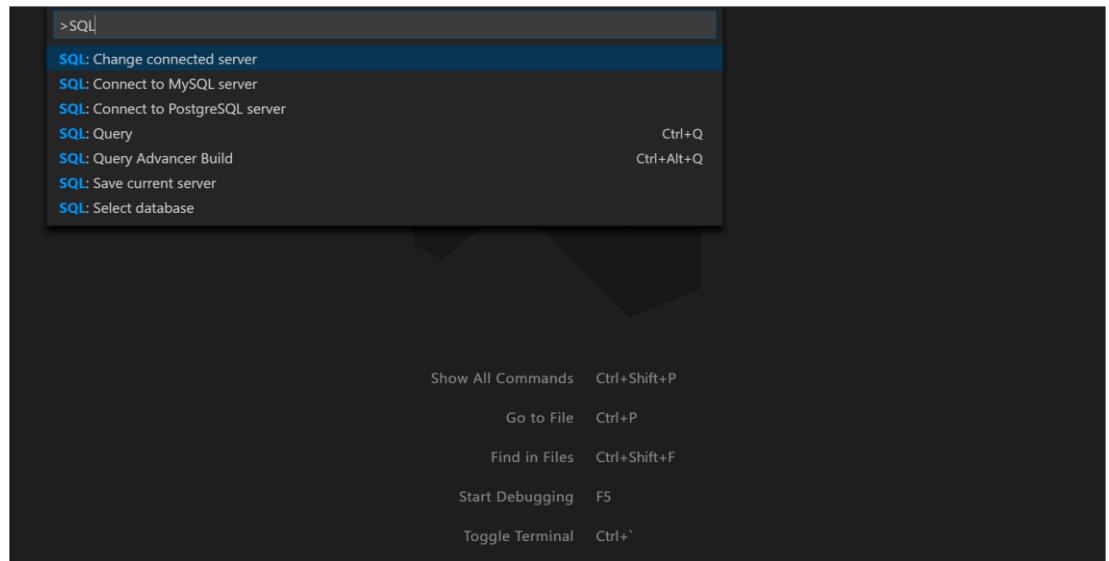
Untuk menginstall extension ini dapat dilakukan dari fitur Extension di Visual Studio Code. Ketikkan mysql pada kolom pencarian, maka akan dapat dilihat extension vscode-database seperti pada gambar di bawah ini.



Gambar 49. Extension vscode-database.

Klik tombol Install, kemudian klik tombol Reload.

Untuk menggunakan extention dapat dilakukan dengan menekan kombinasi tombol Ctrl+Shift+P. Kemudian akan ditampilkan kolom penulisan perintah seperti pada gambar di bawah ini. Ketik SQL untuk melihat perintah-perintah MySQL Client yang tersedia.



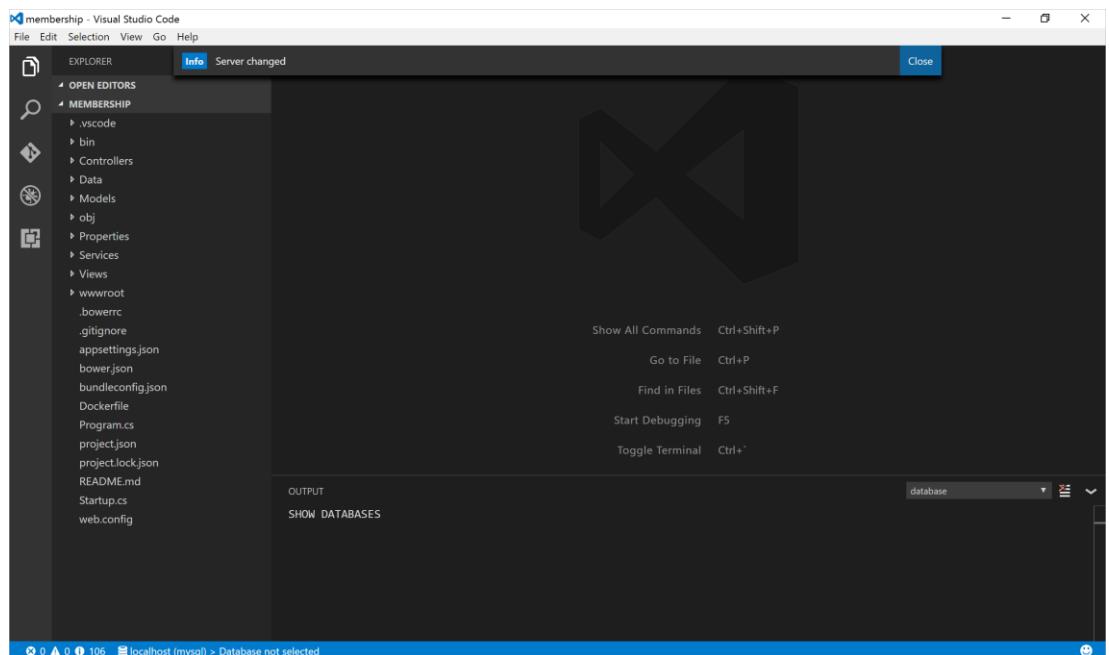
Gambar 50. Extension vscode-database - Show All Commands.

## Koneksi

Untuk melakukan koneksi ke MySQL Server pertama kali pilih “SQL: Connect to MySQL server” kemudian tekan tombol Enter. Kemudian akan diminta memasukkan nilai-nilai ini secara berurutan, yaitu:

1. Alamat server (contoh: localhost).
2. Username (contoh: root).
3. Password (contoh: rahasia).

Jika koneksi berhasil dilakukan akan dilihat dialog window yang berisi informasi “Server changed”.

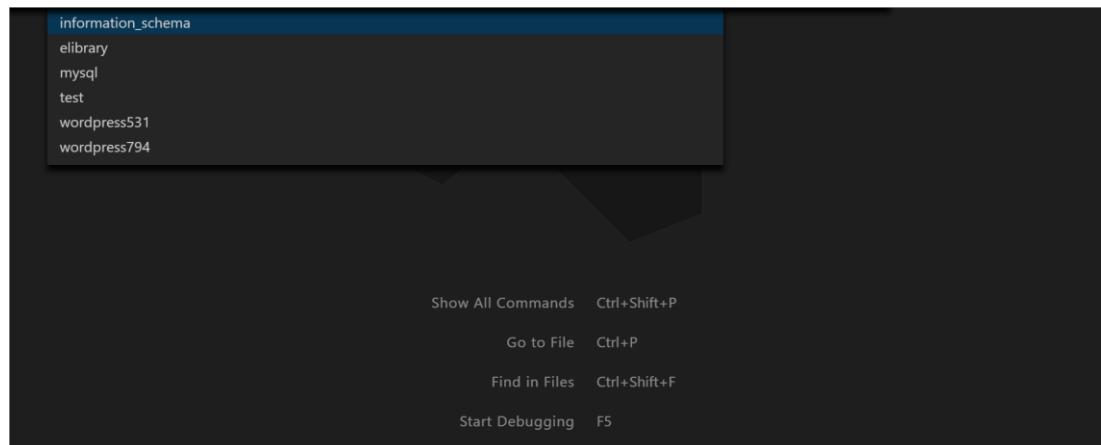


Gambar 51. Extension vscode-database - Koneksi ke database.

## Memilih Database

---

Untuk memilih database dapat digunakan perintah “SQL: Select database”. Kemudian akan ditampilkan daftar database yang ada di server. Pilih database yang ingin digunakan.

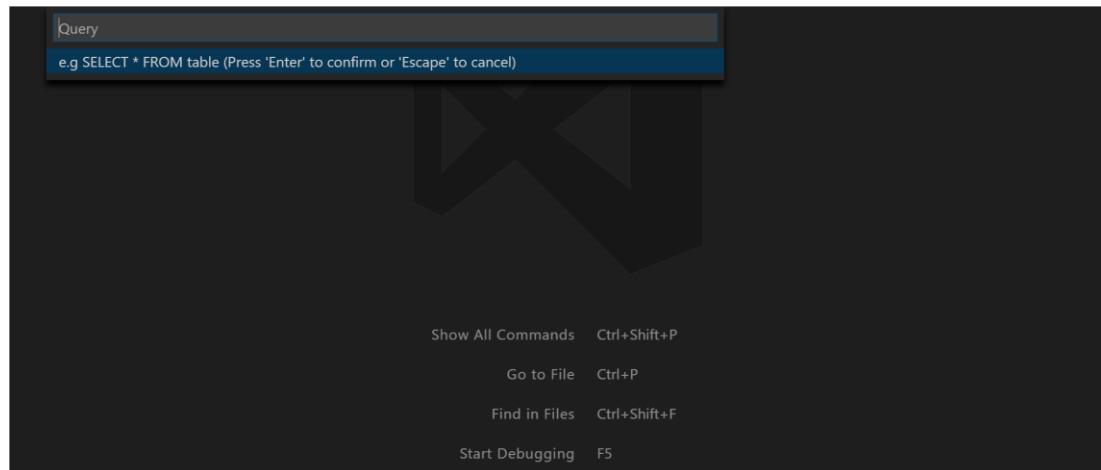


Gambar 52. Extension vscode-database - Daftar database.

## Eksekusi Query

---

Untuk melakukan query dapat dilakukan dengan cara cepat yaitu dengan menekan kombinasi Ctrl+Q. Akan ditampilkan kolom input untuk mengetikkan query seperti pada gambar ini. Kemudian tulis query pada kolom input yang disediakan.



Gambar 53. Extension vscode-database - Eksekusi Query.

Misal query yang ditulis adalah untuk menampilkan daftar database.

```
show databases
```

Maka dapat dilihat output dari query ini pada sebagai berikut.

```
OUTPUT
SHOW DATABASES
show databases

+-----+
| Database      |
+-----+
| information_schema |
| mysql          |
| test           |
| wordpress531   |
| wordpress794   |
+-----+
```

Gambar 54. Extension vscode-database - Show Databases.

Untuk membuat database dapat digunakan query berikut ini. Sebagai contoh nama database yang dibuat adalah elibrary.

```
create database elibrary
```

Pilih database elibrary dengan cara yang telah diberikan pada sub bab sebelumnya.

Untuk membuat tabel, sebagai contoh nama tabel yang dibuat adalah contoh\_tbl, maka digunakan query seperti berikut ini.

```
create table contoh_tbl(
contoh_id INT NOT NULL AUTO_INCREMENT,
isi VARCHAR(100) NOT NULL,
PRIMARY KEY ( contoh_id )
);
```

Untuk menampilkan daftar tabel pada database yang telah dipilih digunakan query berikut ini.

```
show tables
```

Berikut adalah contoh output dari kedua query di atas.

```
OUTPUT
create table contoh_tbl( contoh_id INT NOT NULL AUTO_INCREMENT, isi VARCHAR(100) NOT NULL, PRIMARY KEY ( contoh_id ) );

+-----+-----+-----+-----+-----+
| fieldCount | affectedRows | insertId | serverStatus | warningCount | changedRows |
+-----+-----+-----+-----+-----+
| 0          | 0            | 0        | 2            | 0            | 0            |
+-----+-----+-----+-----+-----+
show tables

+-----+
| Tables_in_elibrary |
+-----+
| contoh_tbl         |
+-----+
```

Gambar 55. Extension vscode-database - Operasi tabel.

Untuk menambahkan data pada tabel contoh\_tbl dapat digunakan query berikut.

```
INSERT INTO contoh_tbl (isi) VALUES ("isi 1")
```

Dan untuk menampilkan seluruh data pada tabel contoh\_tbl digunakan query berikut ini.

```
SELECT * FROM contoh_tbl
```

Hasil kedua query di atas akan ditampilkan dengan output seperti pada gambar di bawah ini.

The screenshot shows the 'OUTPUT' tab of the 'vscode-database' extension in VS Code. It displays the results of a MySQL query. The first part of the output shows the result of an 'INSERT' statement:

```
INSERT INTO contoh_tbl (isi) VALUES ("isi 1")
```

Below it, the results of a 'select \* from contoh\_tbl' query are shown:

```
+-----+-----+-----+-----+-----+
| fieldCount | affectedRows | insertId | serverStatus | warningCount | changedRows |
+-----+-----+-----+-----+-----+
| 0          | 1            | 1        | 2            | 0            | 0            |
+-----+-----+-----+-----+-----+
select * from contoh_tbl
```

```
+-----+-----+
| contoh_id | isi    |
+-----+-----+
| 1         | isi 1  |
+-----+-----+
```

Gambar 56. Extension vscode-database - Insert & Select.

## Kesimpulan

Pada bab ini telah diberikan penjelasan mengenai tool pemrograman yang dapat digunakan untuk memrograman aplikasi web ASP.NET Core, yaitu Visual Studio Code. Selain digunakan untuk memrograman, tool ini juga dapat digunakan untuk melakukan operasi dengan database MySQL.

# 4

## **Pengenalan ASP.NET Core MVC**

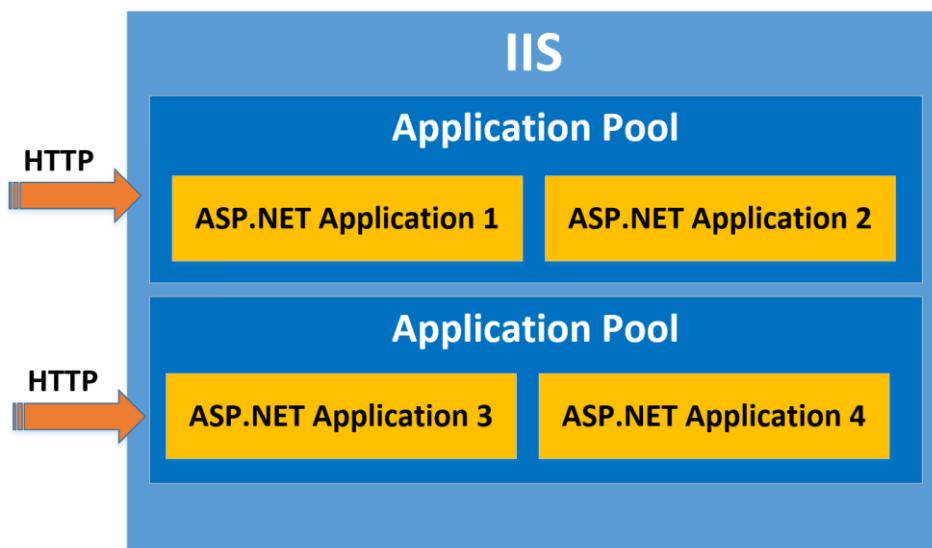
Pada bab ini akan dijelaskan perbedaan cara kerja ASP.NET dibandingkan versi ASP.NET sebelumnya, kemudian dijelaskan konsep pattern Model-View-Controller dan bagaimana implementasinya pada framework ASP.NET Core. Penjelasan konsep dalam bentuk contoh kasus dengan penjelasan langkah demi langkah sehingga akan mudah diikuti dan dimengerti.

---

### **Cara Kerja ASP.NET Core**

Di awal buku ini telah disebutkan bahwa ASP.NET Core adalah framework yang ditulis dari awal, bukan melanjutkan framework ASP.NET sebelumnya. Cara kerjanya pun juga berbeda.

Gambar di bawah ini adalah cara kerja ASP.NET versi sebelumnya. Aplikasi web ASP.NET versi sebelumnya dideploy pada web server Internet Information Services (IIS). Yang mana setiap aplikasi web akan dikelola oleh application pool. Application pool akan menjalankan aplikasi web ASP.NET tersebut.



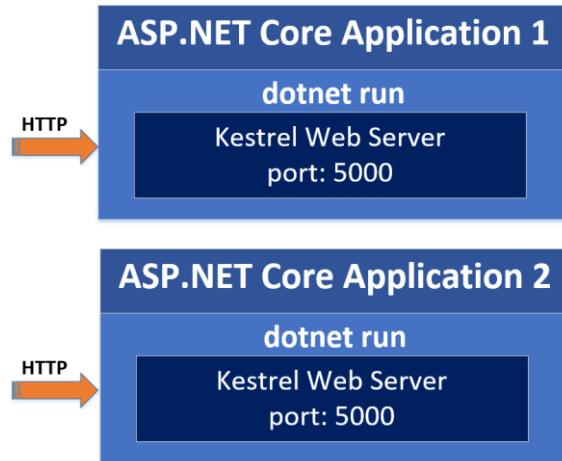
**Gambar 57. Cara kerja ASP.NET klasik pada IIS.**

Jika client ingin mengakses ASP.NET Application 1 maka proses yang terjadi adalah:

1. Client melakukan request ke IIS sebagai web server.
2. Request dikirim ke application pool tempat ASP.NET Application 1 berada.
3. Response diberikan oleh ASP.NET Application 1.

Tetapi karena IIS adalah web server yang hanya tersedia pada sistem operasi MS Windows, maka aplikasi web ASP.NET versi sebelumnya hanya dapat dijalankan pada sistem operasi MS Windows juga.

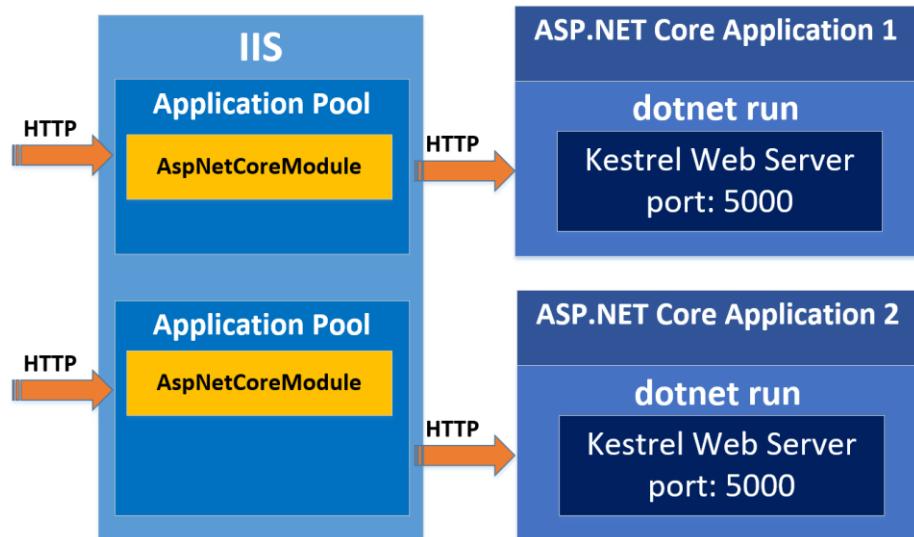
Aplikasi ASP.NET Core bukanlah aplikasi web. Aplikasi ASP.NET Core adalah aplikasi standalone console yang dipanggil oleh perintah runtime **dotnet**. Selanjutnya akan dijalankan web server Kestrel yang akan melakukan response dari request client via jalur HTTP.



Gambar 58. Cara kerja aplikasi ASP.NET Core.

Web server Kestrel merupakan bagian dari runtime ASP.NET Core yang jalan dijalankan pada berbagai platform seperti MS Windows, Linux dan Mac OSX. Hal ini membuat aplikasi ASP.NET Core juga bersifat multiplatform.

Walau aplikasi ASP.NET Core menggunakan web server Kestrel, aplikasi ini juga tetap dapat berkolaborasi dengan web server IIS dengan skema seperti pada gambar di bawah ini.



Gambar 59. Cara kerja aplikasi ASP.NET dengan IIS.

AspNetCoreModule menjadi modul pada IIS setelah .NET Core SDK diinstall. Dari gambar di atas dapat dilihat IIS berperan sebagai front end proxy.

## File & Folder Utama ASP.NET Core

Pada sub bab ini akan dijelaskan file-file utama yang mesti ada pada aplikasi ASP.NET Core. Dan juga akan diperlihatkan alasan kenapa aplikasi ASP.NET Core disebut sebagai aplikasi standalone console.

Pada sub bab ini akan dimulai dengan penjelasan pembuatan aplikasi ASP.NET Core dengan menggunakan perintah “dotnet new”. Kemudian akan dijelaskan file dan folder penting pada aplikasi ASP.NET Core.

## Membuat Project ASP.NET Core

---

Pada sub bab ini akan dijelaskan langkah-langkah untuk membuat project. Berbeda dengan pembuatan project yang telah dibahas pada bab sebelumnya, pada sub bab ini akan membuat project dengan template “ASP.NET Core Empty”. Nama project yang akan dibuat adalah “BelajarASPNETCoreMVC”. Berikut adalah perintah yang digunakan untuk membuat project ini.

```
dotnet new web -o BelajarASPNETCoreMVC
```

## File Utama

---

Berikut ini adalah file-file yang harus ada pada project ASP.NET Core.

### **BelajarASPNETCoreMVC.csproj**

File ini digunakan untuk mendefinisikan metadata project, informasi untuk proses kompilasi dan daftar file atau library yang digunakan. Ini adalah contoh file BelajarASPNETCoreMVC.csproj dari project BelajarASPNETCoreMVC.

```
BelajarASPNETCoreMVC.csproj
<Project Sdk="Microsoft.NET.Sdk.Web">

    <PropertyGroup>
        <TargetFramework>netcoreapp1.1</TargetFramework>
    </PropertyGroup>

    <ItemGroup>
        <Folder Include="wwwroot\" />
    </ItemGroup>

    <ItemGroup>
        <PackageReference Include="Microsoft.AspNetCore" Version="1.1.1" />
    </ItemGroup>

</Project>
```

## Startup.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;
using Microsoft.Extensions.Logging;
```

```

namespace BelajarASPNETCoreMVC
{
    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, ILoggerFactory loggerFactory)
            {
                loggerFactory.AddConsole();

                if (env.IsDevelopment())
                {
                    app.UseDeveloperExceptionPage();
                }

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

```

Pada class Startup terdapat dua method utama yaitu:

1. 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.
2. ConfigureServices, method ini digunakan untuk menambahkan service ke container. Method ini dipanggil sebelum method Configure.

## Program.cs

Ketiga file yang telah dijelaskan di atas dapat ditemui pada aplikasi web ASP.NET klasik ataupun ASP.NET Core. Tetapi file Program.cs tidak akan ditemui pada aplikasi ASP.NET klasik. File Program.cs umumnya ditemui pada aplikasi standalone baik pada aplikasi console ataupun windows form.

File ini ada pada aplikasi ASP.NET Core karena aplikasi ini adalah aplikasi standalone console. Berikut ini adalah isi file Program.cs dari project BelajarASPNETCoreMVC.

Program.cs
<pre> using System; using System.Collections.Generic; using System.IO; using System.Linq; using System.Threading.Tasks; using Microsoft.AspNetCore.Hosting;  namespace BelajarASPNETCoreMVC { </pre>

```

public class Program
{
    public static void Main(string[] args)
    {
        var host = new WebHostBuilder()
            .UseKestrel()
            .UseContentRoot(Directory.GetCurrentDirectory())
            .UseIISIntegration()
            .UseStartup<Startup>()
            .Build();

        host.Run();
    }
}

```

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 menjalankan web server Kestrel.

Aplikasi standalone pada umumnya akan dikompilasi menjadi file .exe dan untuk menjalankannya dapat dilakukan dengan mengeksekusi file .exe tersebut.

Pada aplikasi ASP.NET Core proses kompilasi dilakukan dengan menggunakan perintah.

```
dotnet build
```

Selanjutnya untuk menjalankan aplikasi ASP.NET Core dengan perintah. Cara ini membedakan aplikasi ASP.NET Core dengan aplikasi standalone.

```
dotnet run
```

Selanjutnya dapat dilihat web server Kestrel dijalankan seperti yang terlihat pada pesan berikut.

```

D:\Data\My Books\INDC - ASP.NET - ASP.NET Core &
MySQL\source\BelajarASPNETCoreMVC>dotnet run
Project BelajarASPNETCoreMVC (.NETCoreApp,Version=v1.1) was previously
compiled. Skipping compilation.
Hosting environment: Production
Content root path: D:\Data\My Books\INDC - ASP.NET - ASP.NET Core &
MySQL\source\BelajarASPNETCoreMVC
Now listening on: http://localhost:5000
Application started. Press Ctrl+C to shut down.

```

## Folder Utama

---

Berikut ini akan dijelaskan beberapa folder penting pada project ASP.NET Core.

### **bin**

Folder ini umumnya digunakan untuk menyimpan library dalam bentuk file .dll. Folder ini juga digunakan untuk menyimpan hasil kompilasi project ASP.NET Core.

### **wwwroot**

Folder ini digunakan untuk menyimpan file-file statik yang digunakan oleh aplikasi ASP.NET Core. File-file statik yang umum digunakan pada aplikasi web adalah sebagai berikut:

1. File style .css.
2. File-file gambar.
3. File icon .ico.

- File script seperti Javascript (.js).

## Folder ASP.NET Core MVC

Folder-folder penting pada project ASP.NET Core MVC tidak berbeda dengan versi ASP.NET MVC sebelumnya yaitu:

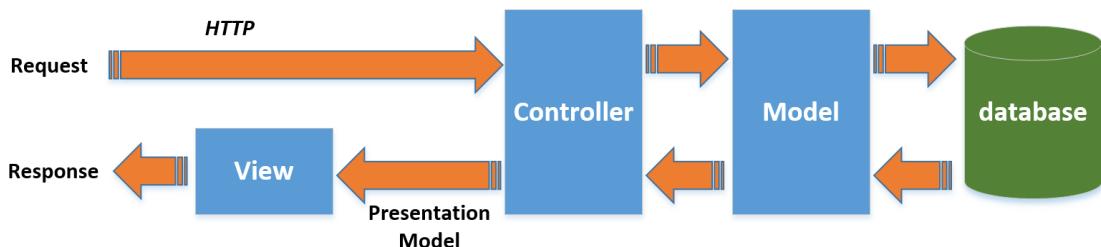
- Views, folder ini menyimpan komponen View.
- Models, folder ini menyimpan komponen Model.
- Controllers, folder ini menyimpan komponen Controller.

Fungsi ketiga folder tersebut akan diperlihatkan pada sub 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 60. 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 BelajarASPNETCoreMVC yang telah dibuat sebelumnya.

## Modifikasi File \*.csproj

Modifikasi file BelajarASPNETCoreMVC.csproj ini bertujuan untuk menambahkan library berikut ini:

- Microsoft.AspNetCore.Mvc, adalah library untuk implementasi framework MVC.
- Microsoft.AspNetCore.StaticFiles, adalah library middleware yang berfungsi memberikan layanan file statik, directory browsing dan default file.

Berikut ini adalah cara menambahkan library ini. Pertama adalah mendeklarasikan kedua library tersebut pada file BelajarASPNETCoreMVC.csproj. Caranya dapat dilihat pada dua baris yang dicetak tebal.

```
BelajarASPNETCoreMVC.csproj
<Project Sdk="Microsoft.NET.Sdk.Web">
  <PropertyGroup>
    <TargetFramework>netcoreapp1.1</TargetFramework>
  </PropertyGroup>
```

```

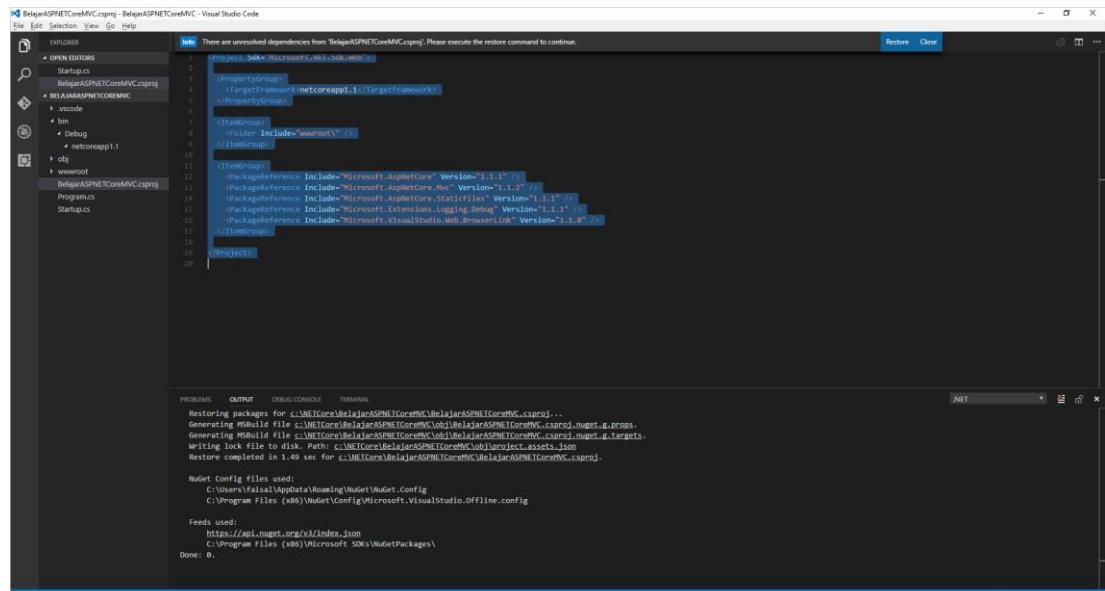
<ItemGroup>
  <Folder Include="wwwroot\" />
</ItemGroup>

<ItemGroup>
  <PackageReference Include="Microsoft.AspNetCore" Version="1.1.1" />
  <PackageReference Include="Microsoft.AspNetCore.Mvc" Version="1.1.2" />
  <PackageReference Include="Microsoft.AspNetCore.StaticFiles" Version="1.1.1" />
  <PackageReference Include="Microsoft.Extensions.Logging.Debug" Version="1.1.1" />
  <PackageReference Include="Microsoft.VisualStudio.Web.BrowserLink" Version="1.1.0" />
</ItemGroup>

</Project>

```

Setelah file disimpan maka Visual Studio Code akan memberikan dialog konfirmasi seperti pada gambar di bawah ini.



**Gambar 61. Restore library Microsoft.AspNetCore.Mvc dan Microsoft.AspNetCore.StaticFiles.**

Karena telah ditambahkan deklarasi penggunaan dua library pada file \*.csproj, maka dengan menekan tombol **Restore** akan dijalankan perintah `dotnet restore` yang akan melakukan proses restore paket-paket library berdasarkan file \*.csproj terbaru.

## Modifikasi File Startup.cs

Modifikasi file `Startup.cs` akan mengubah isi dari method `ConfigureServices` dan `Configure`.

Pada method `ConfigureServices` ditambahkan satu baris seperti pada kode di bawah ini.

```

public void ConfigureServices(IServiceCollection services)
{
    services.AddMvc();
}

```

Penambahanan method `AddMvc()` untuk menambahkan service MVC sehingga project ini dapat menggunakan method-method yang umum digunakan pada aplikasi web MVC.

Selanjutnya memodifikasi method `Configure`. Berikut ini adalah isi dari method `Configure` awal.

```

public void Configure(IApplicationBuilder app, IHostingEnvironment env, ILoggerFactory loggerFactory)
{
    loggerFactory.AddConsole();
}

```

```

    if (env.IsDevelopment())
    {
        app.UseDeveloperExceptionPage();
    }

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

```

Isi method ini akan diubah menjadi sebagai berikut.

```

public void Configure(IApplicationBuilder app, IHostingEnvironment env, ILoggerFactory loggerFactory)
{
    loggerFactory.AddConsole();

    if (env.IsDevelopment())
    {
        app.UseDeveloperExceptionPage();
        app.UseBrowserLink();
    }
    else
    {
        app.UseExceptionHandler("/Home/Error");
    }

    app.UseStaticFiles();

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

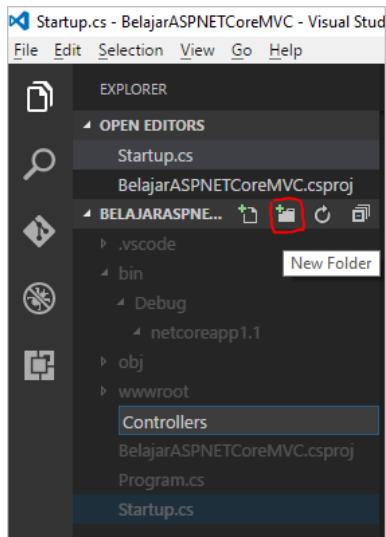
```

Pada kode di atas terdapat 2 method penting yang dipanggil, yaitu:

1. UseStaticFiles() adalah method untuk menangani file static yang secara default disimpan pada folder wwwroot.
2. UseMvc() adalah method yang digunakan untuk membuat routing. Pada contoh di atas digunakan pola routing yang umum digunakan yaitu Controller + Action + ID. Controller default yang digunakan adalah Home. Sedangkan Action default yang digunakan adalah Index. Sedangkan untuk ID terdapat tanda "?" adalah optional artinya dapat diisi atau dikosongkan.

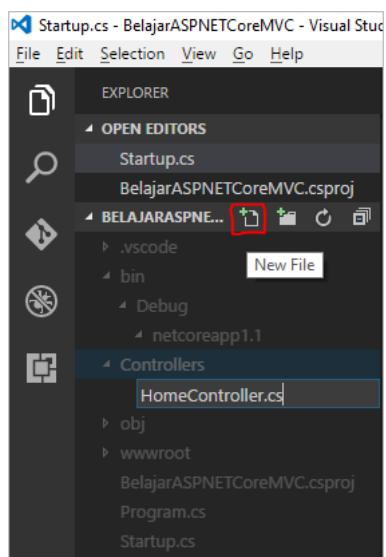
## Controller

Langkah pertama adalah membuat komponen controller. Komponen controller disimpan pada folder Controllers. Jika pada project belum terdapat folder ini maka folder Controllers perlu dibuat. Caranya dengan mengklik tombol “New Folder” pada lingkaran merah seperti yang terlihat pada gambar di bawah ini.



Gambar 62. Tombol New Folder.

Selanjutnya membuat class controller di dalam folder Controllers. Klik folder Controllers pada kemudian klik tombol New File pada lingkaran merah seperti pada gambar di bawah ini.



Gambar 63. Tombol New File.

Nama file controller yang dibuat adalah HomeController.cs. Kemudian ketik kode berikut di file tersebut.

```
HomeController.cs
using Microsoft.AspNetCore.Mvc;

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

        [HttpGet]
        public IActionResult Error()
```

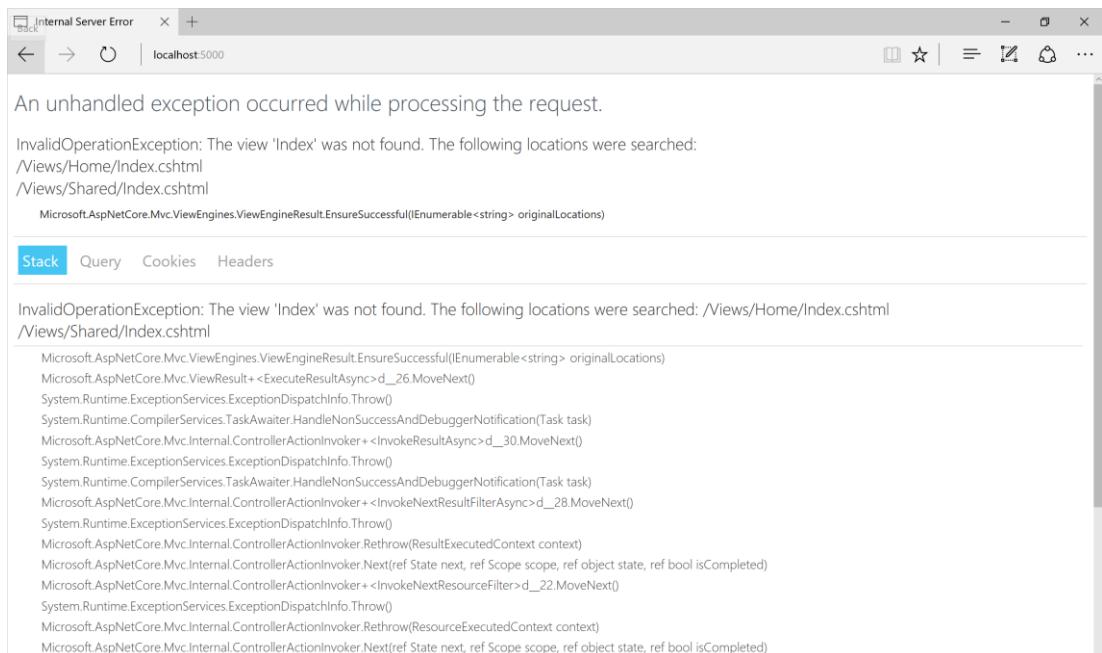
```

    {
        return View();
    }
}

```

Dari kode di atas dapat dilihat bahwa class HomeController merupakan turunan dari class Controller. Hal ini juga akan berlaku untuk semua class controller yang akan dibuat nanti. Di dalam class HomeController 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.

Jika aplikasi ini dijalankan maka akan dapat dilihat pesan kesalahan seperti pada gambar di bawah ini. Hal ini terjadi karena method action pada controller tidak menemukan file view yang sesuai dengan nama method action tersebut.



**Gambar 64. Error - File Index.cshtml tidak ditemukan.**

Selain method Index, pada class HomeController juga memiliki method Error yang akan menampilkan halaman Error.cshtml.

## View

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

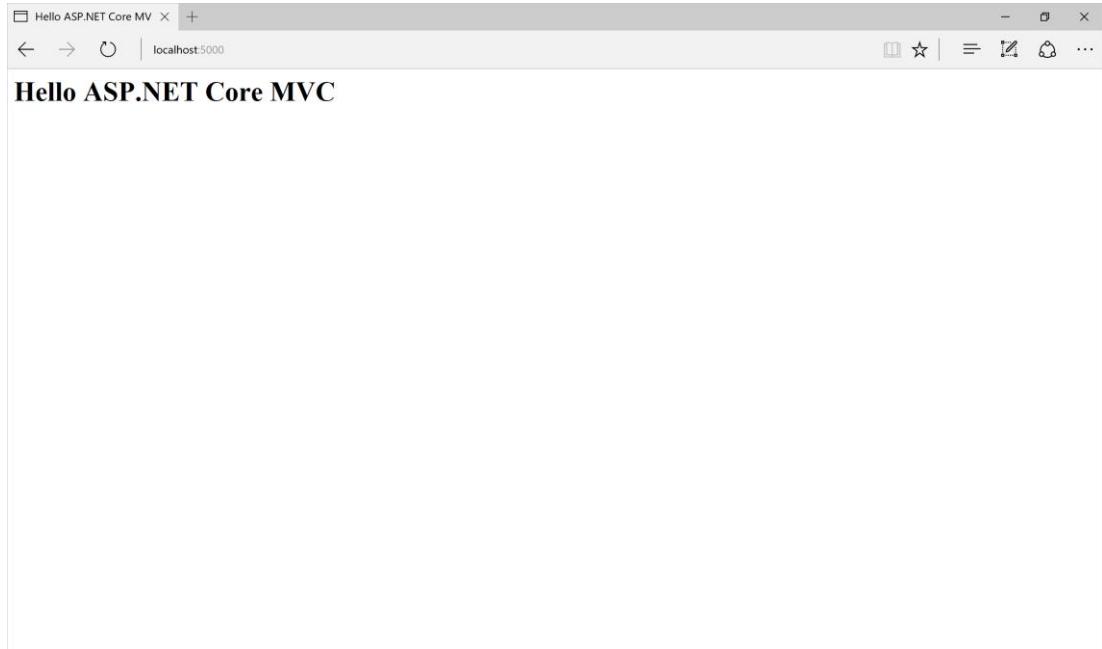
Untuk setiap method action pada class controller harus dibuat sebuah file \*.cshtml dengan nama yang sesuai dengan nama method action tersebut. Sebagai contoh jika pada class HomeController dimiliki satu method action dengan nama Index, maka perlu dibuat file Index.cshtml di dalam folder Views\Home.

Index.cshtml
<html>
<head>
<title>Hello ASP.NET Core MVC</title>

```
</head>

<body>
    <h1>Hello ASP.NET Core MVC</h1>
</body>
</html>
```

Jika project ini didebug dan dijalankan maka error yang sebelumnya ditampilkan sudah tidak ada lagi, dan akan ditampilkan hasil seperti pada gambar di bawah ini.



**Gambar 65. Tampilkan Index.cshtml.**

Jika diperhatikan alamat pada web browser adalah sebagai berikut:

```
http://localhost:5000
```

Halaman di atas juga dapat diakses dengan cara sesuai pola routing yang telah dibahas pada sub bab “Modifikasi File Startup.cs” yaitu: {controller=Home}/{action=Index}/{id?}. Sebagai contoh alamat adalah:

```
http://localhost:5000/Home/Index
```

Tetapi karena secara default telah ditentukan nilai default seperti pola di atas, maka jika controller dan action tidak ditentukan otomatis akan digunakan nilai seperti pola di atas.

Contoh lain adalah dibuat komponen view untuk method action Error. Berikut ini adalah contoh isi file Error.cshtml.

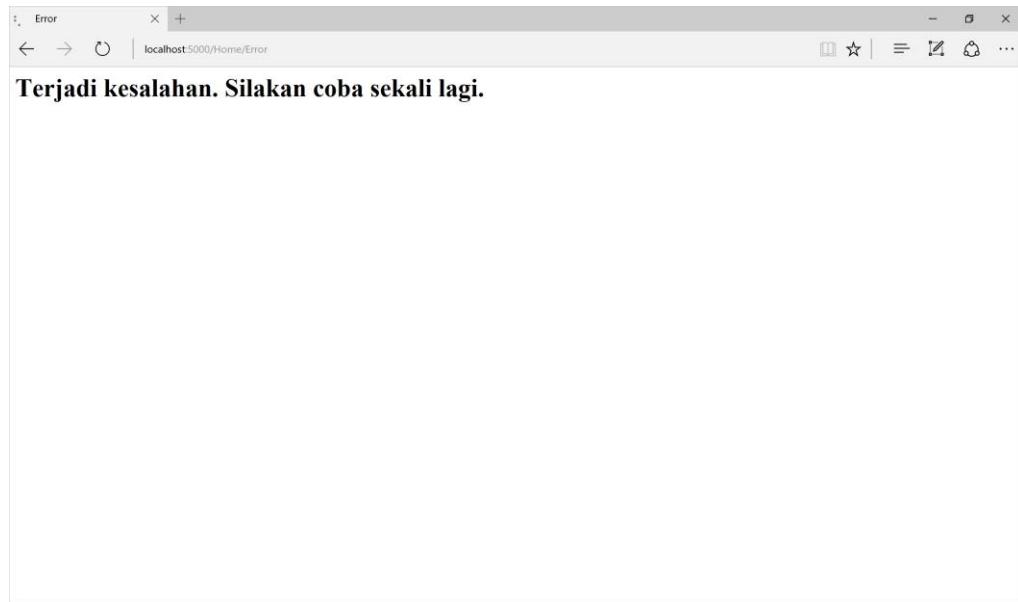
```
Error.cshtml
<html>
<head>
    <title>Error</title>
</head>

<body>
    <h1>Terjadi kesalahan. Silakan coba sekali lagi.</h1>
</body>
</html>
```

Untuk mengakses halaman ini dapat dilakukan dengan menuliskan alamat sebagai berikut:

```
http://localhost:5000/Home/Error
```

Dan hasilnya dapat dilihat pada gambar di bawah ini.

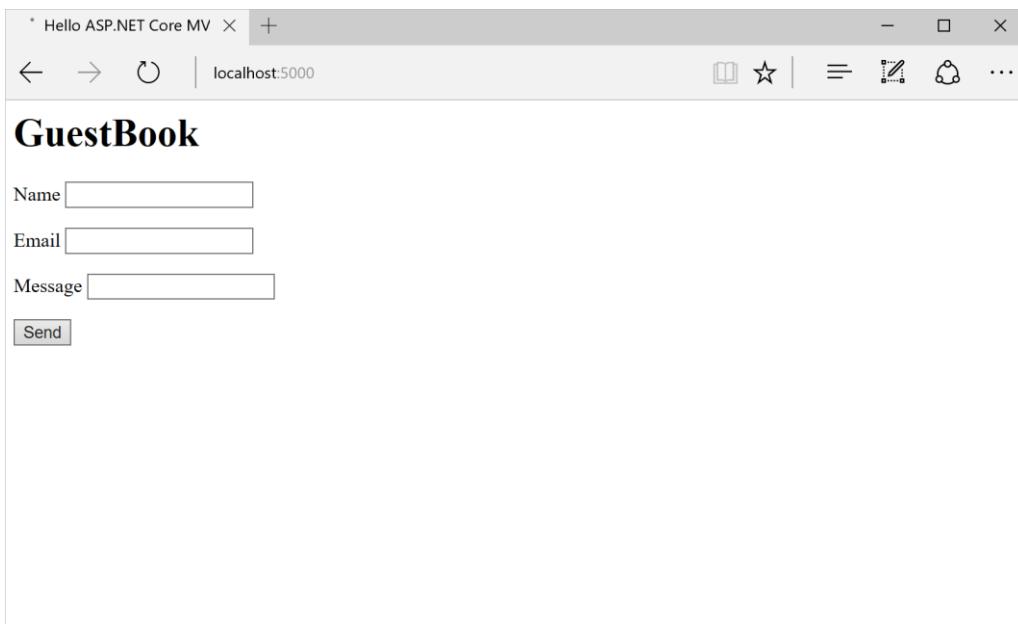


Gambar 66. Tampilan Error.cshtml.

## 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 67. Antarmuka GuestBook.

Komponen model akan disimpan di dalam folder Models. Kemudian tambahkan file GuestBook.cs dengan isi sebagai berikut.

```

GuestBook.cs
using System;

namespace BelajarASPNETCoreMVC.Models
{
    public class GuestBook{
        public String Name {set; get;}
        public String Email {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
@{
    Layout = null;
}

@model BelajarASPNETCoreMVC.Models.GuestBook

<html>
<head>
    <title>Hello ASP.NET Core MVC</title>
</head>

<body>
    <h1>GuestBook</h1>
    <div>
        @using (Html.BeginForm() )
        {
            <p>
                @Html.LabelFor(m=>m.Name)
                @Html.TextBoxFor(m=>m.Name)
            </p>
            <p>
                @Html.LabelFor(m=>m.Email)
                @Html.TextBoxFor(m=>m.Email)
            </p>
            <p>
                @Html.LabelFor(m=>m.Message)
                @Html.TextBoxFor(m=>m.Message)
            </p>
            <p>
                @ViewBag.GuestBookMessage
            </p>
            <p>
                <input type="submit" value="Send" />
            </p>
        }
    </div>
</body>
</html>

```

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 BelajarASPNETCoreMVC.Models.GuestBook

```

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 HomeController.cs.

```
HomeController.cs
using Microsoft.AspNetCore.Mvc;
using BelajarASPNETCoreMVC.Models;

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

        [HttpPost]
        public IActionResult Index(GuestBook data)
        {
            ViewBag.GuestBookMessage = "Hello " + data.Name +
                "(" + data.Email +
                ") menulis " + data.Message;
            return View();
        }

        [HttpGet]
        public IActionResult Error()
        {
            return View();
        }
    }
}
```

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

```
using BelajarASPNETCoreMVC.Models;
```

Pada file class HomeController.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.

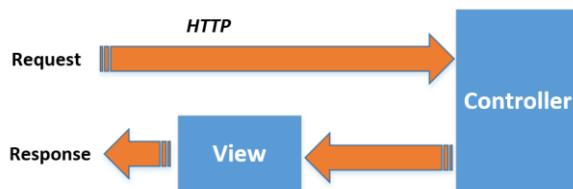
Berikut adalah tampilan halaman setelah tombol Send diklik.

Gambar 68. 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:

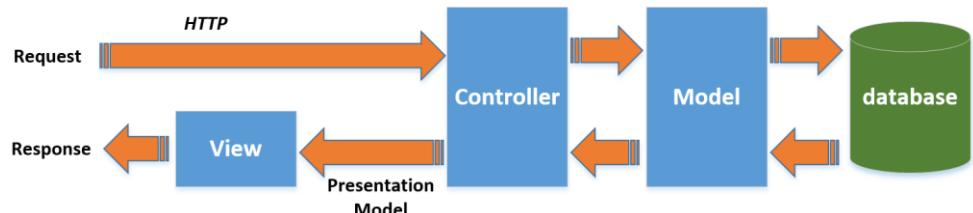
1. Untuk membuat web dengan halaman statik tanpa ada interaksi seperti form atau penggunaan data maka cukup digunakan dua komponen saja yaitu Controller dan View.



Gambar 69. 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 70. 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.

- 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.

---

## ASP.NET Core MVC & MySQL

Pada bab ini akan dijelaskan bagaimana cara melakukan koneksi dan operasi ke database MySQL dari aplikasi ASP.NET Core. Aplikasi yang dibuat adalah pengelolaan buku tamu (guest book). Akan digunakan form input data guest book yang telah dibuat sebelumnya. Kemudian akan ditambahkan kode untuk menyimpan nilai yang diisi pada form guest book ke dalam database MySQL. Dan juga akan ditampilkan data yang telah disimpan di dalam database ke halaman web.

### MySQL Data Core

---

#### Membuat Project

Untuk latihan ini akan dibuat project baru dengan nama MyCoreGuestBook dengan perintah berikut ini.

```
dotnet new web -o MyCoreGuestBook
```

Kemudian modifikasi isi file MyCoreGuestBook.csproj agar isinya sama dengan file BelajarASPNETCoreMVC.csproj yang telah dibuat pada sub bab sebelumnya.

Selanjutnya modifikasi isi file Startup.cs dengan isi yang sama dengan file Startup.cs pada project BelajarASPNETCoreMVC. Setelah itu sesuaikan namespace yang digunakan pada file Startup.cs menjadi MyCoreGuestBook.

Kemudian salin folder Models, Controllers dan View berserta file-file didalamnya. Dan modifikasi nilai namespace yang digunakan pada setiap file-file tersebut agar menjadi MyCoreGuestBook.

### Persiapan

#### Membuat Tabel

Untuk menyimpan data buku tamu diperlukan tabel pada database. Berikut ini adalah script SQL untuk membuat tabel buku\_tamu.

```
create table guestbooks(
    guestbook_id INT NOT NULL AUTO_INCREMENT,
    guest_name VARCHAR(100) NOT NULL,
    guest_email VARCHAR(100) NOT NULL,
    message VARCHAR(256) NOT NULL,
    PRIMARY KEY ( guestbook_id )
);
```

Eksekusi script di atas pada Visual Studio code dengan cara yang telah diberikan pada bab Persiapan Database MySQL > Eksekusi Query.

### **Modifikasi File MyCoreGuestBook.csproj**

Langkah selanjutnya adalah melakukan modifikasi file MyCoreGuestBook.csproj. Akan ditambahkan library MySql.Data.Core. Library ini berisi class dan method-method untuk mengakses database MySQL.

```
<Project Sdk="Microsoft.NET.Sdk.Web">

<PropertyGroup>
    <TargetFramework>netcoreapp1.1</TargetFramework>
</PropertyGroup>

<ItemGroup>
    <Folder Include="wwwroot\" />
</ItemGroup>

<ItemGroup>
    <PackageReference Include="Microsoft.AspNetCore" Version="1.1.1" />
    <PackageReference Include="Microsoft.AspNetCore.Mvc" Version="1.1.2" />
    <PackageReference Include="Microsoft.AspNetCore.StaticFiles" Version="1.1.1" />
    <PackageReference Include="Microsoft.Extensions.Logging.Debug" Version="1.1.1" />
    <PackageReference Include="Microsoft.VisualStudio.Web.BrowserLink" Version="1.1.0" />
    <PackageReference Include="MySql.Data.Core" Version="7.0.4-IR-191" />
</ItemGroup>

</Project>
```

Cara menambahkan library MySql.Data.Core dapat dilihat pada baris cetak tebal. Setelah file MyCoreGuestBook.csproj disimpan maka akan ada proses restore yang dilakukan oleh Visual Studio Code.

### **Membuat File appsettings.json**

Untuk menyimpan variable yang dapat digunakan pada aplikasi web maka variable tersebut dapat disimpan pada suatu file. Sebagai contoh, variable yang menyimpan connection string database. Tapi sebelumnya terlebih dahulu file appsettings.json perlu dibuat di dalam folder project BelajarASPNETCoreMVC. Setelah itu tuliskan variable seperti contoh di bawah ini.

```
{
    "ConnectionStrings": {
        "Default": "server=localhost; userid=root;
                    password=rahasia; database=elibrary;
                    SslMode=None"
    }
}
```

#### **Perhatian:**

Baris section Default yang berisi connection string harus ditulis dalam satu baris tanpa terpotong oleh enter seperti pada contoh di atas. Contoh di atas ditulis dengan tujuan agar mudah dibaca di buku ini.

### **Modifikasi File Startup.cs**

Tujuan memodifikasi file Startup.cs adalah untuk membuat property statik yang akan menyimpan nilai dari nilai connection string yang ditulis pada file appsettings.json. Langkah pertama adalah menambahkan baris berikut ini.

```
using Microsoft.Extensions.Configuration;
```

Baris ini bertujuan untuk penggunaan class Configuration. Langkah kedua adalah menambahkan property static ConnectionString sehingga property ini dapat diakses oleh

seluruh class di dalam project. Selain itu juga ditambahkan provider Configuration yang dapat digunakan untuk membaca file konfigurasi.

```
public static string ConnectionString {get; private set;}  
public static IConfigurationRoot Configuration { get; set; }
```

Selanjutnya menambahkan method Startup seperti berikut ini.

```
public Startup(IHostingEnvironment env)  
{  
    var builder = new ConfigurationBuilder()  
        .SetBasePath(env.ContentRootPath)  
        .AddJsonFile("appsettings.json", optional: true, reloadOnChange: true)  
        .AddEnvironmentVariables();  
  
    Configuration = builder.Build();  
  
    ConnectionString = Configuration.GetConnectionString("Default");  
}
```

Pada bagian pertama method ini berfungsi untuk menentukan nama dan lokasi dari file konfigurasi yang akan dibaca. Dapat dilihat jika file yang akan dibaca adalah appsettings.json. Sedangkan pada baris terakhir digunakan provider Configuration untuk membaca nilai Default pada section ConnectionStrings pada file appsettings.json.

Berikut adalah isi lengkap 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;  
using Microsoft.Extensions.Logging;  
using Microsoft.Extensions.Configuration;  
  
namespace MyCoreGuestBook  
{  
    public class Startup  
    {  
        public static string ConnectionString {get; private set;}  
        public static IConfigurationRoot Configuration { get; set; }  
  
        public Startup(IHostingEnvironment env)  
        {  
            var builder = new ConfigurationBuilder()  
                .SetBasePath(env.ContentRootPath)  
                .AddJsonFile("appsettings.json", optional: true, reloadOnChang  
e: true)  
                .AddEnvironmentVariables();  
  
            Configuration = builder.Build();  
  
            ConnectionString = Configuration.GetConnectionString("Default");  
        }  
  
        // 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, visi  
t https://go.microsoft.com/fwlink/?LinkID=398940  
        public void ConfigureServices(IServiceCollection services)  
        {  
            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, ILoggerFactory loggerFactory)
    {
        loggerFactory.AddConsole();

        if (env.IsDevelopment())
        {
            app.UseDeveloperExceptionPage();
            app.UseBrowserLink();
        }
        else
        {
            app.UseExceptionHandler("/Home/Error");
        }

        app.UseStaticFiles();

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

```

## Pembuatan GuestBook

Setelah proses persiapan di atas telah selesai dilakukan, maka selanjutnya akan dilakukan penulisan program untuk melengkapi kode program pada komponen controller dan view agar menjadi halaman lengkap untuk mengelola buku tamu (guestbook).

Penjelasan pembuatan pengelolaan buku tamu ini akan terdiri atas fitur-fitur sebagai berikut:

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

Untuk merealisasikan fitur-fitur tersebut maka akan dilakukan hal-hal berikut ini:

1. Memodifikasi file class HomeController.cs dengan memodifikasi dan membuat method action untuk menampilkan data dari tabel dan menambah data ke dalam tabel.
2. Memodifikasi file Index.cshtml untuk menampilkan data dari tabel.
3. Menambahkan file Create.cshtml untuk menambah data ke dalam tabel.

### Index

Pada bagian ini akan diperlihatkan modifikasi method action Index beserta penjelasannya. Langkah pertama yang dilakukan adalah menambahkan kedua baris berikut ini pada file HomeController.cs.

```

using System.Collections.Generic;
using MySql.Data.MySqlClient;

```

Baris pertama ditambahkan agar dapat menggunakan class IList(), class ini merupakan class collection untuk menampung object. Sedangkan baris kedua ditambahkan agar dapat digunakan class dan method untuk mengakses database MySQL.

Berikut adalah memodifikasi method action Index menjadi kode di bawah ini..

```
[HttpGet]
public IActionResult Index()
{
    IList<GuestBook> items = new List<GuestBook>();

    // koneksi database
    MySqlConnection conn = new MySqlConnection{
        ConnectionString = Startup.ConnectionString
    };
    conn.Open();

    // menyiapkan query
    MySqlCommand cmd = new MySqlCommand("SELECT * FROM guestbooks;", conn)
;

    // membaca data
    MySqlDataReader dataReader = cmd.ExecuteReader();
    while (dataReader.Read()) {
        // menyimpan record ke object model
        GuestBook item = new GuestBook();
        item.Email = Convert.ToString(dataReader["guest_email"]);
        item.Name = Convert.ToString(dataReader["guest_name"]);
        item.Message = Convert.ToString(dataReader["message"]);

        // menyimpan object model ke collection
        items.Add(item);
    }
    dataReader.Close();
    conn.Close();

    return View(items);
}
```

Pada bagian “koneksi database” dapat dilihat penggunaan class MySqlConnection untuk membuat object untuk melakukan koneksi dengan menggunakan nilai connection string dari Startup.ConnectionString. Dari contoh ini dapat dilihat untuk mengambil nilai connection string dari file appsettings.json dapat dengan memanggil property Connection dari class Startup.

Pada bagian “eksekusi query” dapat dilihat penggunaan class MySqlCommand untuk mengeksekusi query SQL.

Sedangkan pada baris pertama bagian “membaca data” dapat dilihat query SQL dieksekusi dengan cara memanggil method ExecuteReader() dari object cmd. Hasil pembacaan data pada tabel guestbooks ditampung oleh object dataReader yang merupakan instance dari class MySqlDataReader. Tahap selanjutnya adalah membaca setiap record yang ditampung oleh object dataReader kemudian setiap record disimpan dalam sebuah object model item (object hasil instansiasi class GuestBook) dan kemudian object model GuestBook disimpan dalam collection items.

Kemudian pada baris terakhir dapat dilihat cara mengirim object items ke komponen view dengan menggunakan method View(items).

Selanjutnya adalah mengubah isi file Index.cshtml menjadi sebagai berikut.

```
Index.cshtml
@{
    Layout = null;
}

@model IList<MyCoreGuestBook.Models.GuestBook>

<html>
```

```

<head>
    <title>ASP.NET Core MVC - MySQL: GuestBook</title>
</head>

<body>
    <h1>Daftar GuestBook</h1>
    <div>
        <p>@Html.ActionLink("Tambah data", "Create")</p>
        <table style="width:100%; border: solid 1px #000">
            <tr>
                <td>Nama</td>
                <td>Email</td>
                <td>Message</td>
            </tr>
            @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>
                </tr>
            }
        </div>
    </body>
</html>

```

Pada kode di atas dapat dilihat ada perubahan cara penentuan komponen model yang digunakan pada komponen view. Karena komponen controller mengirimkan object model dalam object collection IList maka dapat dilihat penentuan model seperti berikut ini.

```
@model IList< MyCoreGuestBook.Models.GuestBook>
```

Selanjutnya untuk menampilkan data yang disimpan pada object collection mesti digunakan pengulangan dengan sintaks Razor seperti kode di bawah ini.

```

@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>
    </tr>
}

```

Sedangkan untuk membuat hyperlink untuk ke form input buku tamu digunakan sintaks Razor seperti berikut ini. Parameter pertama teks yang akan dilihat pada halaman web, sedangkan parameter kedua adalah method action yang akan menampilkan view Create.cshtml.

```
@Html.ActionLink("Tambah data", "Create")
```

Sintaks-sintaks Razor di atas akan dijelaskan secara detail pada bab tersendiri.

## Create

Untuk membuat form input buku tamu akan dibuat dua method action Create. Berikut adalah isi dari kedua method action Create tersebut.

```

[HttpGet]
public IActionResult Create() {
    return View();
}

[HttpPost]
public IActionResult Create(GuestBook item) {
    if(ModelState.IsValid) {
        MySqlConnection conn = new MySqlConnection{

```

```

        ConnectionString = Startup.ConnectionString
    };
    conn.Open();

    MySqlCommand command = conn.CreateCommand();
    command.CommandText = "INSERT INTO guestbooks (guest_name, guest_email, message) VALUES (?name, ?email, ?message)";

    command.Parameters.AddWithValue("?name", item.Name);
    command.Parameters.AddWithValue("?email", item.Email);
    command.Parameters.AddWithValue("?message", item.Message);
    command.ExecuteNonQuery();

    conn.Close();

    return RedirectToAction("Index");
}
return View();
}

```

Method action Create yang pertama akan memberikan respon jika diakses dengan method GET. Method action ini akan menampilkan komponen view file Create.cshtml yang berfungsi sebagai form input buku tamu. Sedangkan ketika tombol Send diklik maka method action Create method POST akan dipanggil.

Pada method action Create method POST, dapat dilihat input dari method ini adalah object model GuestBook dari komponen view. Selanjutnya object tersebut akan disimpan ke dalam tabel guestbooks. Untuk langkah penyimpanan data dimulai dengan melakukan pembuatan object connection. Kemudian menyiapkan query SQL dengan method CommandText milik object command yang merupakan hasil instansiasi class MySqlCommand. Pada query SQL di atas dapat dilihat terdapat kata yang diawali dengan tanda "?" seperti berikut:

1. ?name.
2. ?email.
3. ?message.

Ketiganya akan diisi oleh nilai-nilai dari object item dengan cara sebagai berikut.

```

command.Parameters.AddWithValue("?name", item.Name);
command.Parameters.AddWithValue("?email", item.Email);
command.Parameters.AddWithValue("?message", item.Message);

```

Setelah semua value dari atribut tabel guestbooks diisi, maka query SQL dieksekusi dengan method ExecuteNonQuery() seperti contoh di bawah ini.

```

command.ExecuteNonQuery();

```

Method ini khusus digunakan untuk query memodifikasi data, seperti:

1. Tambah data.
2. Update data.
3. Hapus data.

Setelah data disimpan ke dalam tabel, selanjutnya akan di eksekusi method action Index dengan perintah di bawah ini untuk mengantarkan ke halaman yang menampilkan daftar buku tamu.

```

return RedirectToAction("Index");

```

Sedangkan untuk komponen view dibuat file Create.cshtml dengan isi sebagai berikut.

```
Create.cshtml
@{
    Layout = null;
}

@model MyCoreGuestBook.Models.GuestBook

<html>
<head>
    <title>ASP.NET Core MVC - MySQL: GuestBook</title>
</head>

<body>
    <h1>Form GuestBook</h1>
    <div>
        @using (Html.BeginForm())
        {
            <p>
                @Html.LabelFor(m=>m.Name)
                @Html.TextBoxFor(m=>m.Name)
            </p>
            <p>
                @Html.LabelFor(m=>m.Email)
                @Html.TextBoxFor(m=>m.Email)
            </p>
            <p>
                @Html.LabelFor(m=>m.Message)
                @Html.TextBoxFor(m=>m.Message)
            </p>
            <p>
                @ViewBag.GuestBookMessage
            </p>
            <p>
                <input type="submit" value="Send" />
            </p>
        }
    </div>
</body>
</html>
```

## Kode Lengkap

Kode lengkap file class HomeController.cs dapat dilihat di bawah ini.

```
HomeController.cs
using System;
using System.Collections.Generic;
using Microsoft.AspNetCore.Mvc;
using MyCoreGuestBook.Models;

using MySql.Data.MySqlClient;

namespace MyCoreGuestBook.Controllers
{
    public class HomeController : Controller
    {
        [HttpGet]
        public IActionResult Index()
        {
            IList<GuestBook> items = new List<GuestBook>();

            // koneksi database
            MySqlConnection conn = new MySqlConnection{
                ConnectionString = Startup.ConnectionString
            };

```

```

        conn.Open();

        // menyiapkan query
        MySqlCommand cmd = new MySqlCommand("SELECT * FROM guestbooks;
", conn);

        // membaca data
        MySqlDataReader dataReader = cmd.ExecuteReader();
        while (dataReader.Read()){
            // menyimpan record ke object model
            GuestBook item = new GuestBook();
            item.Email = Convert.ToString(dataReader["guest_email"]);
            item.Name = Convert.ToString(dataReader["guest_name"]);
            item.Message = Convert.ToString(dataReader["message"]);

            // menyimpan object model ke collection
            items.Add(item);
        }
        dataReader.Close();

        return View(items);
    }

    [HttpGet]
    public IActionResult Create(){
        return View();
    }

    [HttpPost]
    public IActionResult Create(GuestBook item){
        if(ModelState.IsValid){
            MySqlConnection conn = new MySqlConnection{
                ConnectionString = Startup.ConnectionString
            };
            conn.Open();

            MySqlCommand command = conn.CreateCommand();
            command.CommandText = "INSERT INTO guestbooks (guest_name,
guest_email, message) VALUES (?name, ?email, ?message)";
            command.Parameters.AddWithValue("?name", item.Name);
            command.Parameters.AddWithValue("?email", item.Email);
            command.Parameters.AddWithValue("?message", item.Message);
            command.ExecuteNonQuery();

            conn.Close();

            return RedirectToAction("Index");
        }
        return View();
    }

    [HttpGet]
    public IActionResult Error()
    {
        return View();
    }
}

```

## Demo

Jalankan project ini dengan menggunakan fitur Debug pada Visual Studio Code. Method action pertama yang akan dijalankan adalah Index, yang akan memanggil file view Index.cshtml dengan tampilan seperti gambar di bawah ini.

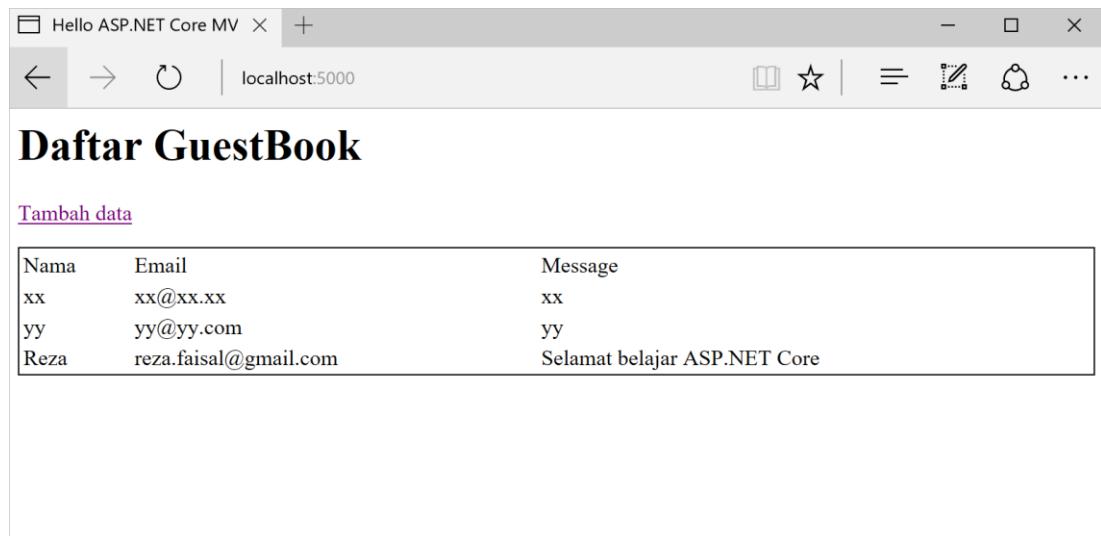
Nama	Email	Message
xx	xx@xx.xx	xx
yy	yy@yy.com	yy

Gambar 71. GuestBook - Index.cshtml.

Pada gambar di atas dapat dilihat daftar data dari tabel guestbooks. Kemudian jika hyperlink Tambah data diklik maka akan dijalankan method action Create method GET yang akan memanggil file view Create.cshtml seperti pada gambar di bawah ini.

Gambar 72. GuestBook - Create.cshtml.

Kemudian setelah form input di atas diisi dan tombol Send diklik maka akan dipanggil method action Create method POST. Setelah data disimpan kemudian akan dipanggil method action Index untuk menampilkan kembali file view Index.cshtml seperti pada gambar di bawah ini. Pada gambar di bawah dapat dilihat data yang baru dimasukkan telah ada di daftar buku tamu.



Gambar 73. GuestBook - Index.cshtml dengan data baru.

## MySQL Entity Framework Core

---

Pada sub bab ini akan diberikan penjelasan pembuatan aplikasi web buku tamu dengan menggunakan Entity Framework Core.

### 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.

## Persiapan

### Membuat Project

Untuk latihan ini akan dibuat project baru dengan nama EFCoreGuestBook. Aplikasi ini mempunyai fitur yang sama persis dengan aplikasi buku tamu yang sudah dibuat pada sub bab sebelumnya. Sehingga struktur folder dan file pada project ini akan persis sama dengan project MyCoreGuestBook.

Langkah pertama yang dilakukan adalah membuat project dengan perintah berikut ini.

```
dotnet new web -o EFCoreGuestBook
```

Kemudian modifikasi file EFCoreGuestBook.csproj agar berisi sama dengan isi file MyCoreGuestBook.csproj yang telah dibuat pada sub bab sebelumnya. Kemudian salin file-file dan folder-folder berikut ini:

1. File Startup.cs.
2. File appsettings.json.
3. Folder Models beserta file-file didalamnya.
4. Folder Views beserta file-file didalamnya.
5. Folder Controllers beserta file-file didalamnya

Kemudian ganti nama namespace pada setiap file menjadi EFCoreGuestBook. Begitu juga namespace yang digunakan sebagai model pada file Create.cshtml dan Index.cshtml.

### Modifikasi File EFCoreGuestBook.csproj

Untuk implementasi EF Core pada project ASP.NET Core MVC dapat dilakukan dengan penambahan baris berikut ini ke dalam file EFCoreGuestBook.csproj. Perbedaan isi file ini dengan isi file MyCoreGuestBook.csproj adalah baris-baris yang dicetak tebal.

```
<Project Sdk="Microsoft.NET.Sdk.Web">  
  
  <PropertyGroup>  
    <TargetFramework>netcoreapp1.1</TargetFramework>  
  </PropertyGroup>  
  
  <ItemGroup>  
    <Folder Include="wwwroot\" />  
  </ItemGroup>  
  
  <ItemGroup>  
    <PackageReference Include="Microsoft.AspNetCore" Version="1.1.1" />  
    <PackageReference Include="Microsoft.AspNetCore.Mvc" Version="1.1.2" />  
    <PackageReference Include="Microsoft.AspNetCore.StaticFiles" Version="1.1.1" />  
    <PackageReference Include="Microsoft.Extensions.Logging.Debug" Version="1.1.1" />  
    <PackageReference Include="Microsoft.VisualStudio.Web.BrowserLink" Version="1.1.0" />  
    <PackageReference Include=" MySql.Data.Core" Version="7.0.4-IR-191" />  
    <PackageReference Include="Microsoft.EntityFrameworkCore" Version="1.1.1" />  
    <PackageReference Include="Microsoft.EntityFrameworkCore.Tools" Version="1.1.0" />  
    <PackageReference Include=" MySql.Data.EntityFrameworkCore" Version="7.0.4-IR-191" />  
  </ItemGroup>  
</Project>
```

```
</Project>
```

## Pembuatan Guest Book

### Modifikasi File Class Model GuestBook

Pada class model GuestBook sebelumnya belum dimiliki property Id. Maka perlu ditambahkan property Id sebagai primary key. Berikut adalah isi file class model GuestBook yang telah dimodifikasi.

```
GuestBook.cs
using System;

namespace EFCoreGuestBook.Models
{
    public partial class GuestBook{

        public int Id {set; get;}
        public String Email {set; get;}
        public String Name {set; get;}
        public String Message {set; get;}
    }
}
```

### Membuat File Class GuestBookDataContext.cs

File class ini merupakan turunan dari class DbContext. Class ini adalah class utama yang bertanggung jawab agar interaksi data sebagai object dapat dilakukan. Class ini mengelola entity object selama aplikasi berjalan, untuk melakukan pembentukan object-object yang menampung data dari database, melacak perubahan dan menyimpan data ke database.

File class GuestBookDataContext.cs disimpan di dalam folder Models. Berikut adalah isi dari file ini.

```
GuestBookDataContext.cs
using Microsoft.EntityFrameworkCore;
using MySQL.Data.EntityFrameworkCore.Extensions;

namespace EFCoreGuestBook.Models
public class GuestBookDataContext : DbContext
{
    public static string ConnectionString { get; set; }

    protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)
    {
        optionsBuilder.UseMySQL(ConnectionString);
    }

    protected override void OnModelCreating(ModelBuilder modelBuilder)
    {
        modelBuilder.Entity<GuestBook>(entity =>
        {
            entity.Property(e => e.Id).HasColumnName("guestbook_id");
            entity.Property(e => e.Email).HasColumnName("guest_email");
            entity.Property(e => e.Name).HasColumnName("guest_name");
            entity.Property(e => e.Message).HasColumnName("message");
        });
    }

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

Pada baris pertama dapat dilihat penggunaan namespace Microsoft.EntityFrameworkCore untuk implementasi EF Core dan agar class DbContext dapat digunakan. Sehingga dapat dilihat class GuestBookDataContext dibuat sebagai turunan dari class DbContext.

Pada baris kedua digunakan namespace MySQL.Data.EntityFrameworkCore.Extensions sebagai implementasi provider MySQL untuk EF Core. Sehingga dapat dilihat penggunaan method UseMySQL pada object optionBuilder.

Hal penting selanjutnya adalah melakukan mapping antara property-property class model GuestBook dengan atribut-atribut pada tabel guestbooks seperti pada penggalan kode berikut.

```
modelBuilder.Entity<GuestBook>(entity =>
{
    entity.Property(e => e.Id).HasColumnName("guestbook_id");
    entity.Property(e => e.Email).HasColumnName("guest_email");
    entity.Property(e => e.Name).HasColumnName("guest_name");
    entity.Property(e => e.Message).HasColumnName("message");
});
```

Dari kode di atas dapat dilihat pemetaan sebagai berikut:

1. Property Id dengan atribut guestbook\_id.
2. Property Email dengan atribut guest\_email.
3. Property Name dengan atribut guest\_name.
4. Property Message dengan atribut message.

Sehingga dapat disimpulkan untuk melakukan mapping antara property class dengan atribut tabel digunakan sintaks berikut ini.

```
entity.Property(e => e.{ObjectProperty}).HasColumnName("{TableAttribute}")
```

Keterangan:

1. ObjectProperty adalah nama property dari class.
2. TableAttribute adalah nama atribut dari table.

Terakhir memetakan antara class GuestBook dengan tabel guestbooks dengan sintaks berikut ini.

```
public virtual DbSet<{ClassName}> {TableName} { get; set; }
```

### Modifikasi File Startup.cs

Yang perlu ditambahkan pada file ini adalah baris berikut ini ke dalam method ConfigureServices. Baris ini berfungsi untuk menentukan connection string yang akan digunakan class GuestBookDataContext.

```
GuestBookDataContext.ConnectionString = Configuration.GetConnectionString("Default");
```

Sehingga dapat dilihat isi lengkap file Startup.cs sebagai berikut.

```
Startup.cs
using Microsoft.AspNetCore.Builder;
using Microsoft.AspNetCore.Hosting;
using Microsoft.Extensions.DependencyInjection;
using Microsoft.Extensions.Logging;
using Microsoft.Extensions.Configuration;

using EFCoreGuestBook.Models;

namespace EFCoreGuestBook
{
    public class Startup
    {
```

```

public static IConfigurationRoot Configuration { get; set; }

public Startup(IHostingEnvironment env)
{
    var builder = new ConfigurationBuilder()
        .SetBasePath(env.ContentRootPath)
        .AddJsonFile("appsettings.json", optional: true, reloadOnChange: true)
        .AddEnvironmentVariables();

    Configuration = builder.Build();
}
// 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)
{
    services.AddMvc();

    GuestBookDataContext.ConnectionString = 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, ILoggerFactory loggerFactory)
{
    loggerFactory.AddConsole();

    if (env.IsDevelopment())
    {
        app.UseDeveloperExceptionPage();
    }
    else
    {
        app.UseExceptionHandler("/Home/Error");
    }

    app.UseStaticFiles();

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

```

### **Modifikasi File Class Controller HomeController**

Langkah selanjutnya adalah implementasi EF Core pada HomeController untuk menampilkan data dan menyimpan data ke dalam database. Langkah pertama adalah menambahkan baris berikut ini.

```
using EFCoreGuestBook.Models;
```

Dengan menambahkan baris tersebut maka class-class pada namespace EFCoreGuestBook.Models dapat digunakan pada class HomeController. Berikut ini adalah kode lengkap isi file class controller HomeController.cs.

```

HomeController.cs
using System.Linq;
using Microsoft.AspNetCore.Mvc;
using EFCoreGuestBook.Models;

namespace EFCoreGuestBook.Controllers
{
    public class HomeController : Controller
    {
        [HttpGet]
        public IActionResult Index()
        {
            GuestBookDataContext db = new GuestBookDataContext();
            var items = db.GuestBooks.ToList();

            return View(items);
        }

        [HttpGet]
        public IActionResult Create()
        {
            return View();
        }

        [HttpPost]
        public IActionResult Create(GuestBook item)
        {
            if (ModelState.IsValid)
            {
                GuestBookDataContext db = new GuestBookDataContext();
                db.Add(item);
                db.SaveChanges();

                return RedirectToAction("Index");
            }
            return View();
        }

        [HttpGet]
        public IActionResult Error()
        {
            return View();
        }
    }
}

```

Untuk mengambil data dari tabel dapat dilihat pada method action Index, yang cukup dilakukan dengan dua baris kode berikut ini.

```

GuestBookDataContext db = new GuestBookDataContext();
var items = db.GuestBooks.ToList();

```

Dapat pada baris kedua cara untuk mengambil data dari tabel pada database, kemudian datanya disimpan dalam bentuk object items. Selanjutnya object items dikirimkan ke komponen view Index.cshtml untuk ditampilkan dengan cara berikut.

```

return View(items);

```

Sedangkan untuk menginput data ke dalam tabel yang sesuai dengan object GuestBook dapat dilakukan dengan langkah berikut. Pertama object yang diisi nilai-nilainya dari komponen view Create.cshtml sehingga pada method action Create dapat dilihat parameter item seperti kode berikut ini.

```

public IActionResult Create(GuestBook item)
{
}

```

Kemudian object item akan disimpan ke dalam database dengan cara di bawah ini.

```

GuestBookDataContext db = new GuestBookDataContext();
db.Add(item);
db.SaveChanges();

```

Dan sekarang aplikasi EFCoreGuestBook telah dapat dijalankan dan digunakan. Tidak perlu ada modifikasi atau perubahan pada komponen view.

---

## Kesimpulan

Pada bab ini dijelaskan cara kerja ASP.NET Core MVC dan cara-cara untuk melakukan koneksi dan operasi ke database MySQL dari aplikasi web ASP.NET Core MVC. Sehingga pembaca sudah dapat menyiapkan project dan melakukan konfigurasi. Pada bab selanjutnya akan mengulang penjelasan tentang komponen model, view dan controller, namun penjelasan lebih detail.

# 5

## 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.

### Persiapan

#### Aplikasi Book Store

Aplikasi ini akan memiliki fitur-fitur sebagai berikut:

1. Mengelola pengarang buku.  
Fitur ini dapat digunakan untuk menampilkan, menambah, mengedit dan menghapus data pengarang buku.
2. Mengelola kategori buku.  
Fitur ini dapat digunakan untuk menampilkan, menambah, mengedit dan menghapus data kategori buku.
3. 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 74. 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.

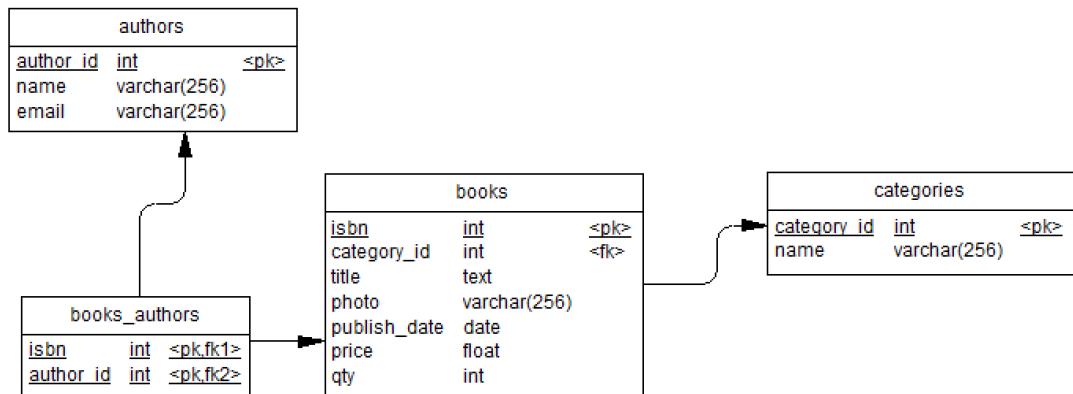
Template admin yang digunakan pada buku ini adalah Gentellela yang menggunakan framework Bootstrap 3. Template ini dapat diunduh dari link berikut <https://github.com/puikinsh/gentelella>. Cara untuk mengubah template ini agar bisa digunakan pada aplikasi web ASP.NET Core akan dijelaskan pada sub bab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

## Database

Aplikasi book store ini akan menggunakan database baru dengan nama bookstore yang memiliki fitur sebagai berikut:

1. Mengelola pengarang.
2. Mengelola kategori buku.
3. Mengelola buku.

Dari fitur-fitur tersebut diperlukan database dengan tabel-tabel sebagai berikut.



Gambar 75. Tabel BookStore.

Langkah pertama adalah terhubung dengan MySQL server dengan menggunakan Visual Studio Code dengan menekan tombol **ctrl+shift+P**. Kemudian ketik “SQL: Connect to MySQL Server”. Kemudian masukkan nama server, username dan password. Kemudian tekan tombol **ctrl+Q** untuk mengeksekusi query, kemudian ketikan query berikut untuk membuat database dengan nama corebookstore.

```
create database corebookstore
```

Selanjutnya dengan cara yang sama jalankan query-query berikut untuk membuat table-table di atas.

```

create table authors
(
    author_id      int not null auto_increment,
    name           varchar(256),
    email          varchar(256),
    primary key (author_id)
);

create table categories
(
    category_id    int not null auto_increment,
    name           varchar(256),

```

```

        primary key (category_id)
    );

create table books
(
    isbn                int not null,
    category_id         int,
    title               text,
    photo               varchar(256),
    publish_date        date,
    price               float,
    qty                 int,
    primary key (isbn)
);

create table books_authors
(
    isbn                int not null,
    author_id           int not null,
    primary key (isbn, author_id)
);

```

Sedangkan untuk membuat relasi antar tabel digunakan query-query berikut ini.

```

alter table books add constraint FK_REFERENCE_2 foreign key (category_id)
references categories (category_id) on delete restrict on update restrict;

alter table books_authors add constraint FK_REFERENCE_3 foreign key (isbn)
references books (isbn) on delete restrict on update restrict;

alter table books_authors add constraint FK_REFERENCE_4 foreign key
(author_id) references authors (author_id) on delete restrict on update
restrict;

```

Langkah selanjutnya adalah membuat model yang akan diterangkan pada sub bab berikutnya.

## Membuat Project

Nama project yang akan dibuat adalah EFCoreBookStore dengan cara menggunakan perintah berikut ini.

```
dotnet new web -o EFCoreBookStore
```

Selanjutnya memodifikasi file-file berikut ini agar mempunyai isi yang sama seperti isi file yang dimiliki pada project EFCoreGuestBook.

1. EFCoreBookStore.csproj dimodifikasi agar isinya sama dengan EFCoreGuestBook.csproj.
2. Startup.cs.

Kemudian ganti nilai namespace EFCoreGuestBook menjadi EFCoreBookStore. Berikut adalah isi lengkap file Startup.cs pada project ini.

```

Startup.cs
using Microsoft.AspNetCore.Builder;
using Microsoft.AspNetCore.Hosting;
using Microsoft.Extensions.DependencyInjection;
using Microsoft.Extensions.Logging;
using Microsoft.Extensions.Configuration;

using EFCoreBookStore.Models;

namespace EFCoreBookStore

```

```

{
    public class Startup
    {
        public static IConfigurationRoot Configuration { get; set; }

        public Startup(IHostingEnvironment env)
        {
            var builder = new ConfigurationBuilder()
                .SetBasePath(env.ContentRootPath)
                .AddJsonFile("appsettings.json", optional: true, reloadOnChange: true)
                .AddEnvironmentVariables();

            Configuration = builder.Build();
        }

        // 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)
        {
            services.AddMvc();

            BookStoreDataContext.ConnectionString = 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, ILoggerFactory loggerFactory)
        {
            loggerFactory.AddConsole();

            if (env.IsDevelopment())
            {
                app.UseDeveloperExceptionPage();
            }
            else
            {
                app.UseExceptionHandler("/Home/Error");
            }

            app.UseStaticFiles();

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

```

## Membuat File appsettings.json

Modifikasi file appsetting.json untuk melakukan koneksi ke database yang baru saja dibuat pada langkah di atas.

```

appsettings.json
{
    "ConnectionStrings": {
        "Default": "server=localhost;userid=root;password=rahasia;database=corebookstore;SslMode=None"
    }
}

```

```
    }  
}
```

## Membuat File GuestBookDataContext.cs

Langkah pertama adalah membuat BookStoreDataContext.cs yang disimpan pada folder Models. Kemudian mengganti isi filenya menjadi seperti berikut ini.

```
BookStoreDataContext.cs  
using Microsoft.EntityFrameworkCore;  
using MySQL.Data.EntityFrameworkCore.Extensions;  
  
namespace EFCoreGuestBook.Models{  
    public class BookStoreDataContext : DbContext  
    {  
        public static string ConnectionString { get; set; }  
  
        protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)  
        {  
            optionsBuilder.UseMySQL(ConnectionString);  
        }  
  
        protected override void OnModelCreating(ModelBuilder modelBuilder)  
        {  
        }  
    }  
}
```

---

## Model

Pada bab sebelumnya 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 data dari database. Kemudian pada contoh berikutnya class model GuestBook dimodifikasi agar memiliki property-property yang sesuai dengan atribut-atribut pada tabel, dan 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 penampung data dalam proses pertukaran data.

Pada bab ini akan dijelaskan bagaimana membuat model untuk kebutuhan menampung data dan memanfaatkan model untuk operasi database. Selain itu juga akan diberikan fungsi model lainnya yang dapat digunakan untuk label yang ditampilkan pada form atau header tabel pada komponen view. Selain itu juga cara pemanfaatan validasi pada form di komponen view.

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:

3. Data annotations.
4. 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.
2. Class view model yang berfungsi untuk menampilkan data ke komponen view saja.

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. Hal lain yang membuat sebuah class menjadi class entity adalah dengan mendaftarkan class tersebut pada class data context.

Konfigurasi class agar menjadi class entity model dapat dilakukan dengan menggunakan Fluent API. Konfigurasi dengan menggunakan Fluent API dilakukan di dalam method OnModelCreating pada class data context. Class utama yang digunakan untuk melakukan konfigurasi adalah ModelBuilder. Maka akan dapat dilihat method OnModelCreating seperti berikut ini.

```
protected override void OnModelCreating(ModelBuilder modelBuilder){}
```

Berikut ini adalah contoh pembuat 4 class entity model dengan konfigurasi menggunakan Fluent API. Class entity yang pertama adalah class Category yang merupakan representasi tabel categories. Class ini memiliki dua property yaitu:

1. CategoryID yang merupakan representasi atribut category\_id.
2. Name yang merupakan representasi atribut name.

```
Category.cs
using System;
using System.Collections.Generic;

namespace EFCoreBookStore.Models
{
    public partial class Category{
        public int CategoryID {set; get;}
        public String Name {set; get;}
    }
}
```

Kemudian tambahkan kode berikut ini ke dalam class data context, yaitu class BookStoreDataContext. Kode berikut ditambahkan ke dalam method OnModelCreating.

```
modelBuilder.Entity<Category>().ToTable("categories");

modelBuilder.Entity<Category>(entity =>
{
    entity.Property(e => e.CategoryID).HasColumnName("category_id");
    entity.Property(e => e.Name).HasColumnName("name");
});

modelBuilder.Entity<Category>().HasKey(e => new { e.CategoryID});
```

Pada baris pertama adalah cara untuk melakukan mapping antara class entity Category dengan tabel categories. Sintaks cara mapping class entity dengan tabel adalah sebagai berikut.

```
modelBuilder.Entity<NAMA_CLASS_ENTITY>().ToTable("NAMA_TABEL");
```

Kemudian untuk melakukan mapping property-property class dengan atribut-atribut tabel dapat dilihat pada bagian tengah kode di atas. Dari kode tersebut dapat dilihat sintaksnya sebagai berikut ini.

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

Konfigurasi selanjutnya adalah menentukan property dari class Category yang akan menjadi primary key yang dapat dilihat dari baris terakhir. Sintaks dari konfigurasi ini adalah sebagai berikut.

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

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

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

Nama property biasanya merupakan bentuk jamak dari nama class entity. Jika nama class entity adalah Category maka bentuk jamaknya adalah Categories. Contoh lain jika nama class entity adalah Book maka bentuk jamaknya adalah Books.

Sintaks dari kode di atas adalah sebagai berikut.

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

Selanjutnya adalah membuat class entity Author, Book dan BookAuthor. Berikut adalah kode lengkap dari ketiga class tersebut.

Class entity berikutnya adalah class Author yang merupakan representasi tabel authors. Class ini memiliki tiga property yaitu:

1. AuthorID, property ini merupakan representasi dari atribut author\_id.
2. Name, property ini merupakan representasi dari atribut name.
3. Email, property ini merupakan representasi dari atribut email.

Author.cs

```
using System;

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

Class entity berikutnya adalah class Books dengan kode sebagai berikut.

Book.cs

```
using System;

namespace EFCoreBookStore.Models
{
    public partial class Book{
        public int ISBN {set; get;}
        public int CategoryID {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;}
    }
}
```

```
}
```

Dan selanjutnya adalah class entity BooksAuthor.

```
BookAuthor.cs
```

```
using System;

namespace EFCoreBookStore.Models
{
    public partial class BookAuthor{

        public int ISBN {set; get; }

        public int AuthorID {set; get; }
    }
}
```

Selanjutnya adalah melakukan mapping setiap class-class entity di atas dengan tabel-tabel pada database yang dilakukan pada class data context BookStoreDataContex di bawah ini.

```
BookStoreDataContext.cs
```

```
using Microsoft.EntityFrameworkCore;
using MySQL.Data.EntityFrameworkCore.Extensions;
using Microsoft.Extensions.Configuration;

namespace EFCoreBookStore.Models{
    public class BookStoreDataContext : DbContext
    {
        public static string ConnectionString { get; set; }

        protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)
        {
            optionsBuilder.UseMySQL(ConnectionString);
        }

        protected override void OnModelCreating(ModelBuilder modelBuilder)
        {
            modelBuilder.Entity<Category>().ToTable("categories");
            modelBuilder.Entity<Category>(entity =>
            {
                entity.Property(e => e.CategoryID).HasColumnName("category_id");
                entity.Property(e => e.Name).HasColumnName("name");
            });
            modelBuilder.Entity<Category>().HasKey(e => new { e.CategoryID });

            modelBuilder.Entity<Author>().ToTable("authors");
            modelBuilder.Entity<Author>(entity =>
            {
                entity.Property(e => e.AuthorID).HasColumnName("author_id");
                entity.Property(e => e.Name).HasColumnName("name");
                entity.Property(e => e.Email).HasColumnName("email");
            });
            modelBuilder.Entity<Author>().HasKey(e => new { e.AuthorID });

            modelBuilder.Entity<Book>().ToTable("books");
            modelBuilder.Entity<Book>(entity =>
            {
                entity.Property(e => e.ISBN).HasColumnName("isbn");
                entity.Property(e => e.CategoryID).HasColumnName("category_id");
                entity.Property(e => e.Title).HasColumnName("title");
                entity.Property(e => e.Photo).HasColumnName("photo");
                entity.Property(e => e.PublishDate).HasColumnName("publish_date");
                entity.Property(e => e.Price).HasColumnName("price");
            });
        }
    }
}
```

```

        entity.Property(e => e.Quantity).HasColumnName("qty");
    });
modelBuilder.Entity<Book>().HasKey(e => new { e.ISBN });

modelBuilder.Entity<BookAuthor>().ToTable("books_authors");
modelBuilder.Entity<BookAuthor>(entity =>
{
    entity.Property(e => e.ISBN).HasColumnName("isbn");
    entity.Property(e => e.AuthorID).HasColumnName("author_id");
});
modelBuilder.Entity<BookAuthor>().HasKey(e => new { e.ISBN, e.
AuthorID });
}
}
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; }
}
}

```

Pada kode di atas dapat dilihat juga bagaimana cara memberikan 2 primary key pada class entity, yaitu dengan cara sebagai berikut.

```
modelBuilder.Entity<BookAuthor>().HasKey(e => new { e.ISBN, e. AuthorID });
```

Class data context dan class entity model harus disiapkan pada awal pembangunan aplikasi web. Jika menggunakan pendekatan Database First, maka kedua class ini dapat dibuat secara otomatis jika menggunakan Visual Studio (cara penggunaan Visual Studio untuk membuat kedua class ini dapat dilihat pada buku Seri Belajar ASP.NET: ASP.NET MVC Untuk Pemula). Tetapi jika menggunakan Visual Studio Code maka kedua class tersebut harus dibuat secara manual.

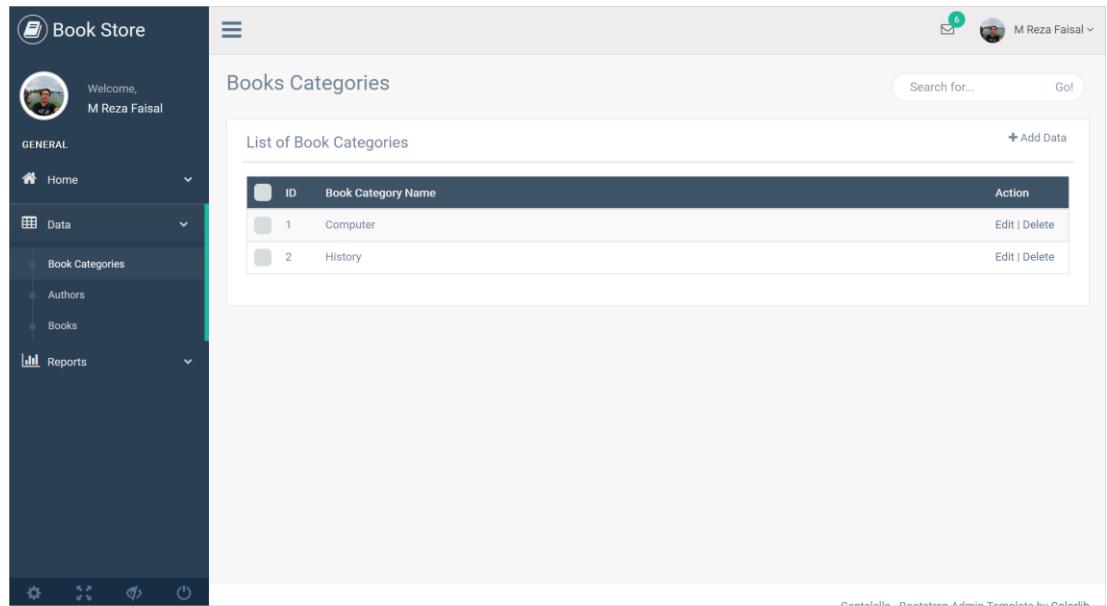
Terdapat dua fungsi dari class entity model, yaitu:

1. Menampung data untuk komunikasi antara komponen controller dan view. Fungsi ini adalah fungsi yang pasti dimiliki oleh setiap model.
2. Sedangkan fungsi spesifik yang hanya dimiliki oleh class entity model ini adalah sebagai representasi dari tabel di database seperti yang telah disebutkan di atas.

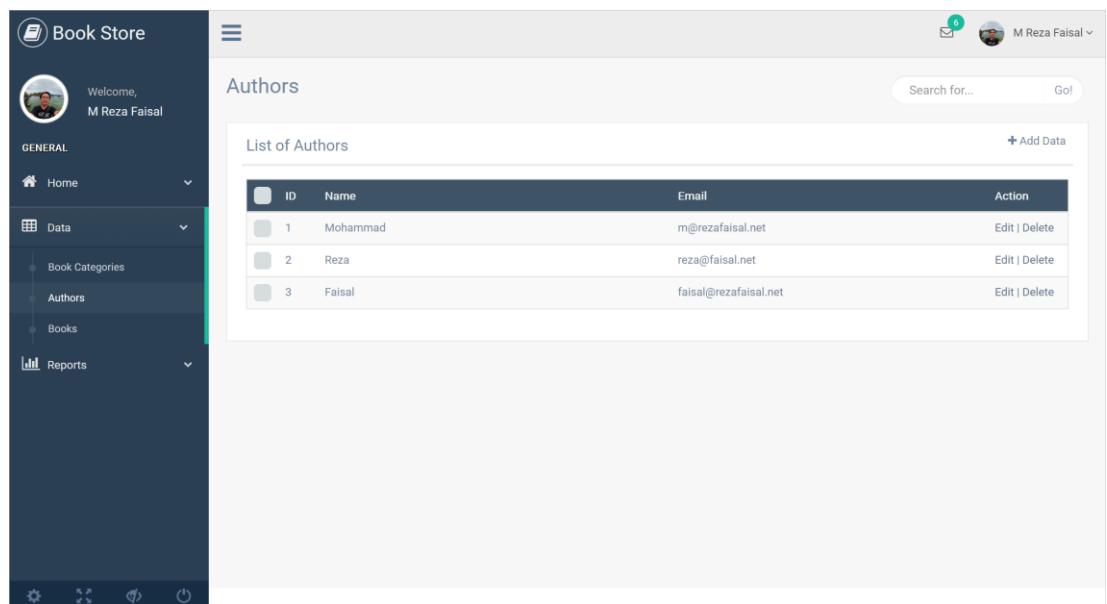
Dari ketiga fitur aplikasi web book store yang telah disebutkan di atas, hanya 2 fitur saja yang dapat dibuat dengan menggunakan class entity model yang telah dibuat di atas, yaitu:

1. Mengelola pengarang dengan menggunakan class Author. Fitur ini berfungsi untuk menampilkan, menambah, mengedit dan menghapus data pengarang.
2. Mengelola kategori buku dengan menggunakan class Category. Fitur ini berfungsi untuk menampilkan, menambah, mengedit dan menghapus data kategori buku.

Berikut adalah gambar antarmuka kedua fitur book store. Cara membuat kedua fitur ini akan dijelaskan pada sub bab model dan controller.



Gambar 76. Book Store - Daftar kategori buku.



Gambar 77. Book Store - Daftar pengarang buku.

### Class View Model

Class view model dibuat dengan tujuan untuk mengatur antaramuka komponen view untuk interaksi data, baik untuk menampilkan data dalam bentuk tabel atau dalam bentuk form untuk input data yang tidak bisa dilakukan oleh class entity model.

Fitur aplikasi web book store yang memerlukan class view model adalah fitur mengelola buku. 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 books 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.

Table books\_authors adalah table yang menyimpan relasi antara nama pengarang dan buku, tetapi isinya hanya isbn dan author\_id 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, books\_authors, authors dan categories.

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

```
BookViewModel.cs
using System;

namespace EFCoreBookStore.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;}
    }
}
```

Jika class entity model digunakan merupakan representasi sebuah tabel pada database, sehingga harus dipetakan dengan tabel. Sedangkan class view model bukan representasi dari tabel yang ada di database. Setelah class view model menjadi object maka object ini hanya akan digunakan untuk menampung data yang didapat object dari class entity model.

Berikut adalah contoh bagaimana object dari class BookViewModel menampung data dari object class entity model Author, Book dan Category. Langkah pertama adalah membuat object collection yang berisi dari object BookViewModel.

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

Kemudian mengambil daftar buku dari database dengan cara sebagai berikut.

```
BookStoreDataContext db = new BookStoreDataContext();
var bookList = db.Books.ToList();
```

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.

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

    item.ISBN = book.ISBN;
    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 sebagai berikut.

```

foreach(Books book in bookList) {
    BooksViewModel item = new BooksViewModel();

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

    var category = db.Categories.Where(p=>p.CategoryID.Equals(book.CategoryID)).Single<Category>();
    item.CategoryName = category.Name;
    . . .
}

```

Langkah selanjutnya adalah menyiapkan variable authorNameList untuk menampung nama-nama pengarang. Kemudian membaca tabel books\_authors untuk mendapatkan daftar pengarang buku berdasarkan ISBN.

```

foreach(Books book in bookList) {
    BooksViewModel item = new BooksViewModel();

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

    var category = db.Categories.Where(p=>p.CategoryID.Equals(book.CategoryID)).Single<Categories>();
    item.CategoryName = category.Name;

    string authorNameList = string.Empty;
    var booksAuthorsList = db.BooksAuthors.Where(p=>p.ISBN.Equals(book.ISBN));
    . . .
}

```

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

```

foreach(Books book in bookList) {
    BooksViewModel item = new BooksViewModel();

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

    var category = db.Categories.Where(p=>p.CategoryID.Equals(book.CategoryID)).Single<Categories>();
    item.CategoryName = category.Name;

```

```

        string authorNameList = string.Empty;
        var booksAuthorsList = db.BooksAuthors.Where(p=>p.ISBN.Equals(book.ISBN));

        foreach(BookAuthor booksAuthors in booksAuthorsList){
            BookStoreDataContext db2 = new BookStoreDataContext();
            var author = db2.Authors.Where(p=>p.AuthorID.Equals(booksAuthors.AuthorID)).Single<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 sebagai berikut ini.

Cover	Category	Title	Publish Date	Qty	Action
	Computer	Seri Belajar ASP.NET : ASP.NET MVC Untuk Pemula	1/1/2017 12:00:00 AM	10	Edit   Delete

Gambar 78. Book Store - Daftar pengarang buku.

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.

## Display & Format

Pada 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;
```

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.ComponentModel.DataAnnotations;

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

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

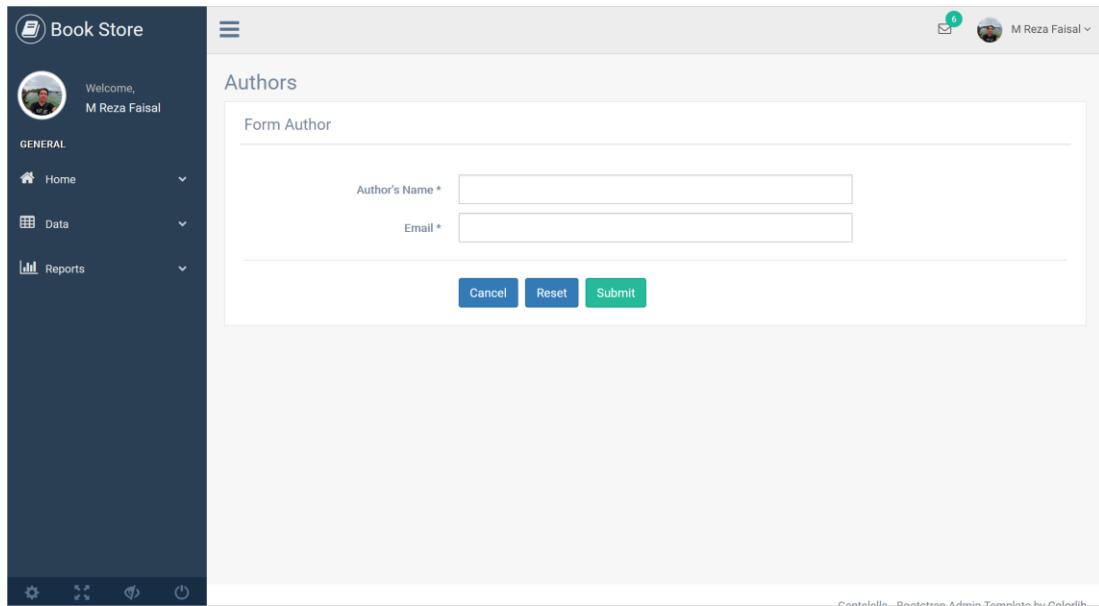
        [Display(Name ="Email")]
        public String Email {set; get; }
    }
}
```

The screenshot shows a web application interface. On the left, there is a sidebar with a dark blue background and white text. It has sections for 'GENERAL' (Home, Data, Books, Reports) and 'LOGGED IN AS' (M Reza Faisal). The 'Data' section is expanded, showing 'Book Categories', 'Authors', and 'Books'. The main content area has a light gray background. At the top right, there is a user profile icon with a green notification badge showing '6' and the name 'M Reza Faisal'. Below the profile, there is a search bar with 'Search for...' and a 'Go!' button. The main title is 'Authors' and below it is 'List of Authors'. A table is displayed with the following data:

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 79. 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 80. 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-MMMM-yyyy}")]
public DateTime PublishDate {set; get;}
```

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

## 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.")]
```

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.ComponentModel.DataAnnotations;

namespace EFCoreBookStore.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;}
    }
}
```

```
}
```

## 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.ComponentModel.DataAnnotations;

namespace EFCoreBookStore.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;}
    }
}
```

## 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.ComponentModel.DataAnnotations;

namespace EFCoreBookStore.Models
{
    public partial class 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 ISBN {set; get; }

        [Display(Name ="Category ID")]
        [Required(ErrorMessage = "{0} harus diisi.")]
    }
}
```

```

[RegularExpression("^[0-9]*$", ErrorMessage = "{0} harus angka")]
public int CategoryID {set; get;}

[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;}
}
}

```

## 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 Berikut adalah kode lengkap class entity model BookAuthor yang telah diberikan atribut-atribut.

```

BookAuthor.cs
using System;
using System.ComponentModel.DataAnnotations;

namespace EFCoreBookStore.Models
{
    public partial class BookAuthor{

        [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 ISBN {set; get;}

        [Display(Name ="AuthorID")]
        [Required(ErrorMessage = "{0} harus diisi.")]
        public int AuthorID {set; get;}
    }
}

```

## 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.ComponentModel.DataAnnotations;

```

```

namespace EFCoreBookStore.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.ComponentModel.DataAnnotations;
using System.Collections.Generic;
using Microsoft.AspNetCore.Http;

namespace EFCoreBookStore.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.

---

## 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/gentelella>, maka langkah selanjutnya adalah extract file gentelella-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	BOWERC 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 81. File dan folder template gentelella.**

Selanjutnya salin folder-folder berikut ke dalam folder wwwroot pada project EFBookStore:

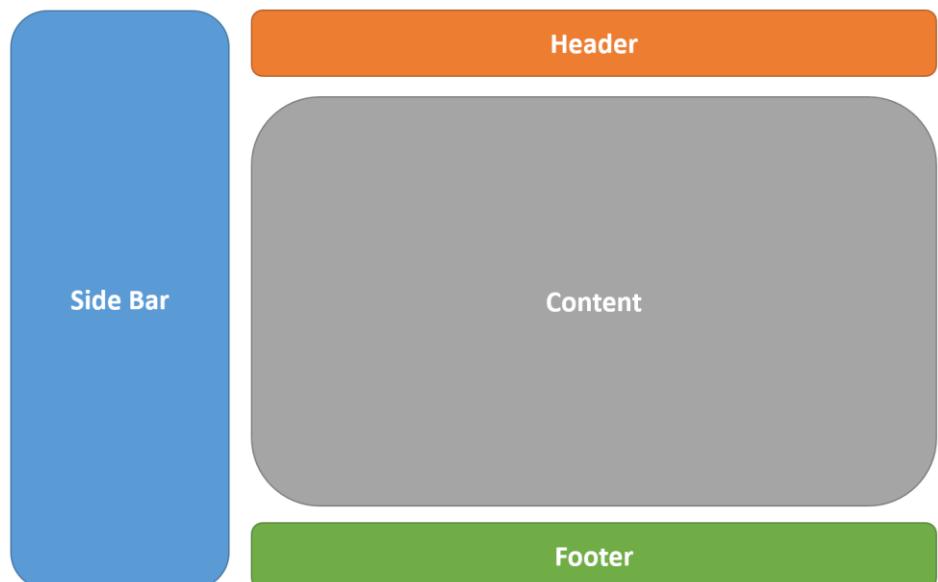
- build.
- vendors.

Selanjutnya di dalam folder production, salin folder-folder berikut ini ke dalam folder wwwroot:

- css.
- images.
- js.

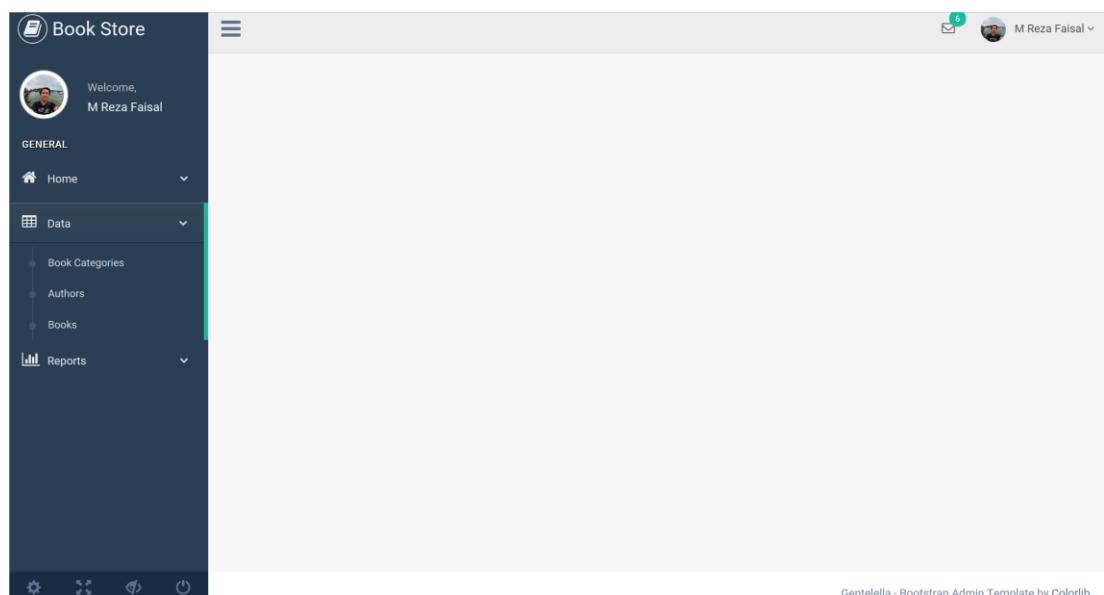
### **Master Layout**

Langkah selanjutnya adalah membuat master layout. Layout aplikasi Book Store memiliki bagian atau area sebagai berikut.



**Gambar 82.** Layout aplikasi.

Dengan menggunakan template gentelella maka dapat dibuat master layout seperti gambar berikut ini.



**Gambar 83.** Master layout template gentelella.

Berikut adalah langkah-langkah yang dilakukan untuk membuat master layout pada aplikasi web ASP.NET Core MVC.

Langkah pertama adalah membuat folder Shared di dalam folder Views. Kemudian dilanjutkan dengan membuat file MasterLayout.cshtml di dalam folder Views\Shared. Untuk membuat MasterLayout.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.

Isi lengkap file MasterLayout.cshtml adalah sebagai berikut.

```
MasterLayout.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>
        <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-bar-chart-o"></i> Reports <span class="fa fa-chevron-down"></span></a>
                    <ul class="nav child_menu">
                        <li><a href="#">Book by Category Chart</a></li>
                        <li><a href="#">Book Selling</a></li>
                    </ul>
                </li>
            </ul>
        </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" href="login.html">

```

```

></span>
</a>
</div>
<!-- /menu footer buttons --&gt;
&lt;/div&gt;
&lt;/div&gt;

<!-- top navigation --&gt;
&lt;div class="top_nav"&gt;
&lt;div class="nav_menu"&gt;
&lt;nav&gt;
&lt;div class="nav_toggle"&gt;
&lt;a id="menu_toggle"&gt;&lt;i class="fa fa-bars"&gt;&lt;/i&gt;&lt;/a&gt;
&lt;/div&gt;

&lt;ul class="nav navbar-nav navbar-right"&gt;
&lt;li class=""&gt;
&lt;a href="#" class="user-profile dropdown-toggle" data-toggle="dropdown" aria-expanded="false"&gt;
&lt;img src="/images/reza.jpg" alt=""&gt;M Reza Faisal
&lt;span class="fa fa-angle-down"&gt;&lt;/span&gt;
&lt;/a&gt;
&lt;ul class="dropdown-menu dropdown-usermenu pull-right"&gt;
&lt;li&gt;&lt;a href="#"&gt; Profile&lt;/a&gt;&lt;/li&gt;
&lt;li&gt;
&lt;a href="#"&gt;
&lt;span class="badge bg-red pull-right">50%</span>
<span>Settings</span>
</a>
</li>
<li><a href="#">Help</a></li>
<li><a href="login.html"><i class="fa fa-sign-out pull-right"></i> Log Out</a></li>
</ul>
</li>

<li role="presentation" class="dropdown">
<a href="#" 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"></span>
<span>
<span>John Smith</span>
<span class="time">3 mins ago</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 ago</span>
</span>
</a>
</li>

```

```


    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 ago</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 ago</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</strong>
            <i class="fa fa-angle-right"></i>
        </a>
    </div>
</li>
</ul>
</li>
</ul>
</nav>
</div>
</div>
<!-- /top navigation -->

<!-- ----- -->
<!-- page content -->
<div class="right_col" role="main">
@RenderBody()
</div>
<!-- /page content -->
<!-- ----- -->

<!-- footer content -->
<footer>
    <div class="pull-right">
        Gentelella - Bootstrap Admin Template by <a href="https://colorlib.com">Colorlib</a>
    </div>

```

```

        <div class="clearfix"></div>
    </footer>
    <!-- /footer content -->
</div>
</div>

<!-- jQuery -->
<script src="~/vendors/jquery/dist/jquery.min.js"></script>
<!-- Bootstrap -->
<script src="~/vendors/bootstrap/dist/js/bootstrap.min.js"></script>
<!-- FastClick -->
<script src="~/vendors/fastclick/lib/fastclick.js"></script>
<!-- NProgress -->
<script src="~/vendors/nprogress/nprogress.js"></script>
<!-- Chart.js -->
<script src="~/vendors/Chart.js/dist/Chart.min.js"></script>
<!-- gauge.js -->
<script src="~/vendors/gauge.js/dist/gauge.min.js"></script>
<!-- bootstrap-progressbar -->
<script src="~/vendors/bootstrap-progressbar/bootstrap-
progressbar.min.js"></script>
<!-- iCheck -->
<script src="~/vendors/iCheck/icheck.min.js"></script>
<!-- Skycons -->
<script src="~/vendors/skycons/skycons.js"></script>
<!-- Flot -->
<script src="~/vendors/Flot/jquery.flot.js"></script>
<script src="~/vendors/Flot/jquery.flot.pie.js"></script>
<script src="~/vendors/Flot/jquery.flot.time.js"></script>
<script src="~/vendors/Flot/jquery.flot.stack.js"></script>
<script src="~/vendors/Flot/jquery.flot.resize.js"></script>
<!-- Flot plugins -->
<script src="~/vendors/flot.orderbars/js/jquery.flot.orderBars.js"><s
cript>
    <script src="~/vendors/flot-
spline/js/jquery.flot.spline.min.js"></script>
    <script src="~/vendors/flot.curvedlines/curvedLines.js"></script>
    <!-- DateJS -->
    <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"><s
cript>
        <!-- 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 MasterLayout.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 master layout atau dapat juga menggunakan master layout.

Jika file view ingin berdiri sendiri tanpa menggunakan master 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/MasterLayout.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 MasterLayout.cshtml sebagai master layout maka pada 40 file view tersebut harus ditambahkan baris di atas.

Untuk otomatisasi agar seluruh file-file view menggunakan file master 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/MasterLayout.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 EFCoreBookStore.Models  
@addTagHelper *, Microsoft.AspNetCore.Mvc.TagHelpers
```

Baris pertama berfungsi untuk mendeklarasikan namespace EFCoreBookStore.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 Microsoft.AspNetCore.Mvc;

namespace EFCoreGuestBook.Controllers
{
    public class LatihanController : Controller
    {

        [HttpGet]
        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;
}

<html>
    <head>
        <title>Latihan</title>
    </head>
    <body>
        <h1>Latihan</h1>

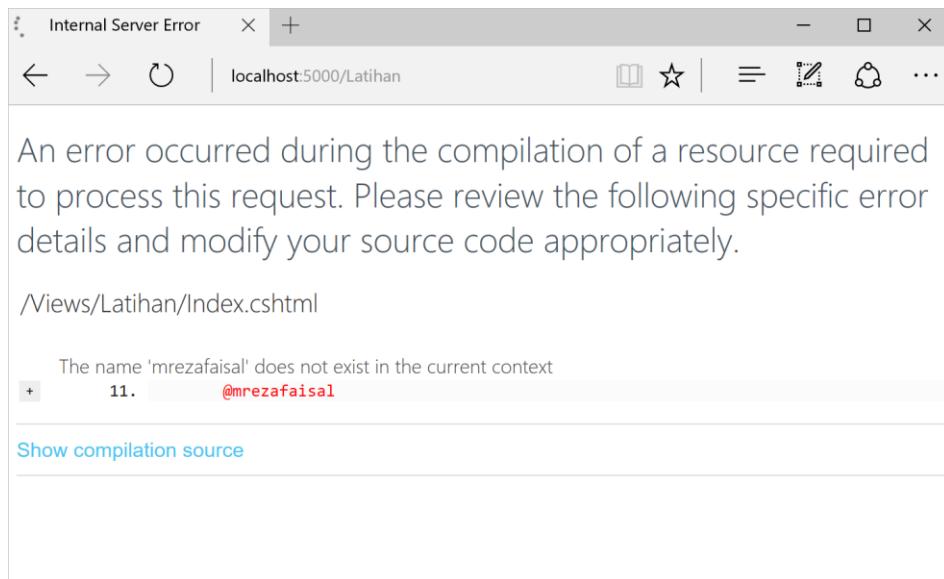
    </body>
</html>
```

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.

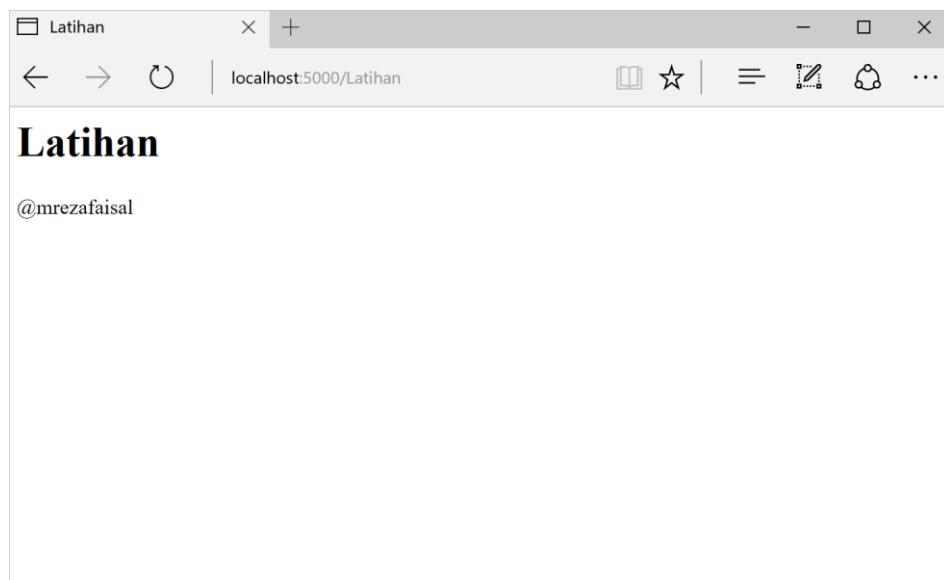


Gambar 84. Razor - Pesan kesalahan.

Untuk menghindari kesalahan terjadi maka perlu ditambahkan simbol @ sehingga menjadi sebagai berikut.

```
@@mrezafaisal
```

Hasilnya dapat dilihat pada gambar di bawah ini.



Gambar 85. Razor - Penggunaan simbol @@.

Untuk penggunaan 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.

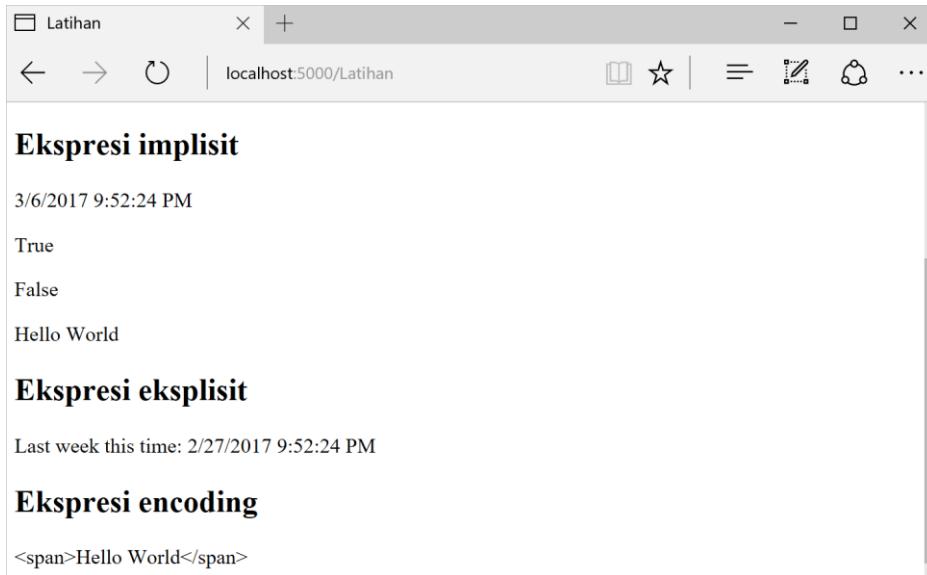
```
<h2>Ekspresi eksplisit </h2>
<p>Last week this time: @(DateTime.Now - TimeSpan.FromDays(7))</p>
```

- 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 86. Razor - Ekspresi implisit & eksplisit.

## Blok Kode

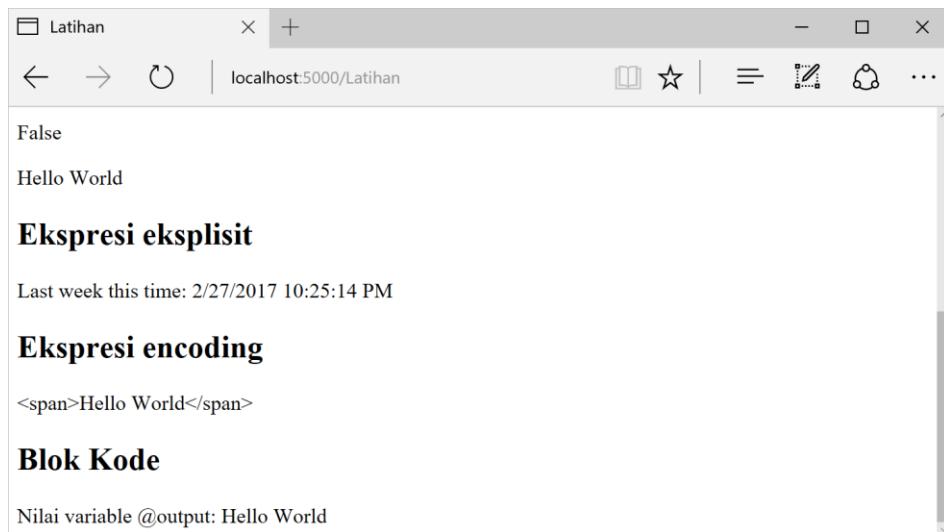
Blok kode Razor diawali dengan simbol @ kemudian diikuti dengan tanda {}. Berbeda dengan ekspresi, kode yang berada di dalam blok kode ini tidak dirender. Berikut adalah contoh implementasi blok kode.

```
@{  
    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 87. 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 @:

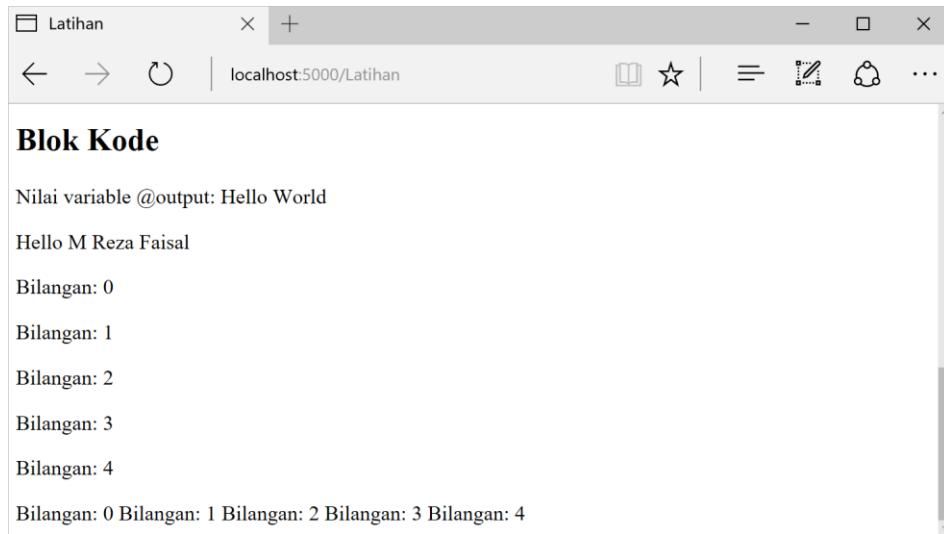
Penggunaan simbol @: dapat digunakan menggantikan tag HTML yang dicontohkan pada cara di atas. Berikut adalah implementasi cara ini.

```

@for (var i = 0; i < 5; i++)
{
    @:Bilangan: @i
}

```

Hasil dari ketiga cara di atas dapat dilihat pada gambar di bawah ini.



Gambar 88. 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>
}

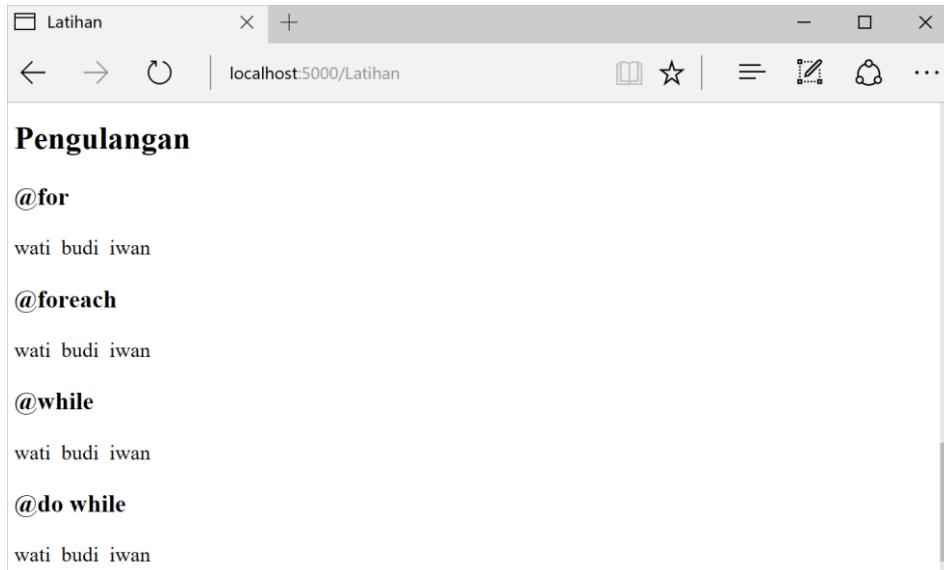
<h3>@@foreach</h3>
@foreach (var name in names) {
    <text>@name &nbsp;</text>
}

<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 89. 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.

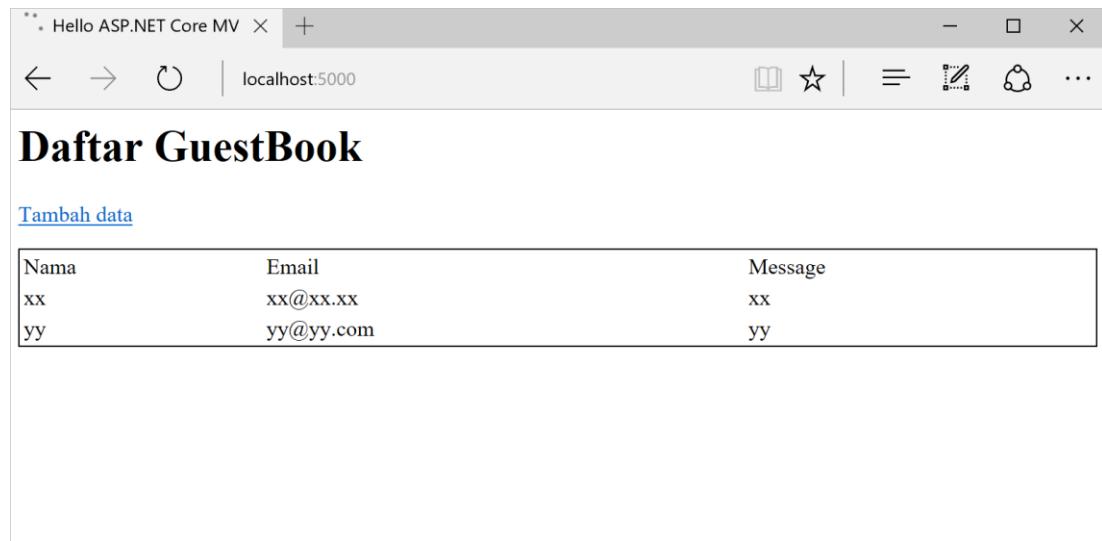
```
...
@model EFCoreGuestBook.Models.GuestBook
...
```

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 dark header "Daftar GuestBook". Below it is a table with three columns: "Nama", "Email", and "Message". The table contains two rows of data:

Nama	Email	Message
xx	xx@xx.xx	xx
yy	yy@yy.com	yy

Gambar 90. 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 MyGuestBook dan EFCoreGuestBook.

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 EFCoreBookStore.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. Contoh dari kasus ini dapat dilihat pada file Create.cshtml pada folder Views/Home di proyek EFCoreGuestBook.

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 IList<EFCoreGuestBook.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 pada aplikasi EFCoreBookStore.

```
@model EFCoreBookStore.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 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 EFCoreBookStore.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 EFCoreBookStore.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 EFCoreBookStore.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 EFCoreBookStore.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 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 91. Tampilan form sebelum proses validasi.

Jika input textbox tidak diisi dan tombol submit diklik maka akan ditampilkan pesan kesalahan berikut ini.

Gambar 92. 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 \*', 'Middle Name / Initial', 'Gender' (with options 'Male' and 'Female' where 'Female' is highlighted in blue), and 'Date Of Birth \*'. Below the fields are three buttons: 'Cancel', 'Reset', and 'Submit' (which is highlighted in green).

Gambar 93. 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>
</form>
```

```

        </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">
                    &nbsp; Male &nbsp;
                </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 EFCoreBookStore akan menggunakan menggunakan tag helper pada setiap halaman view.

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 di baris awal halaman tersebut.

```
@addTagHelper *, Microsoft.AspNetCore.Mvc.TagHelpers
```

Jika ingin secara otomatis seluruh halaman view menggunakan ini maka perlu dibuat file \_ViewImports.cshtml yang disimpan pada folder Views. Dan isi dari file ini adalah baris deklarasi di atas.

## 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 EFCoreBookStore.Models.Author  
  
<label asp-for="Name"></label>
```

Tag helper di atas akan dirender menjadi tag HTML lengkap di bawah ini.

```
<label for="Name">Author's Name</label>
```

## Image

Untuk membuat elemen <img> menjadi tag helper dapat dilakukan dengan sintaks berikut ini.

```

```

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.

```

```

Tag helper tersebut akan dirender menjadi sebagai berikut.

```

```

Tag helper <img> 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 EFCoreBookStore.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 EFCoreBookStore.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
using System;
using System.ComponentModel.DataAnnotations;

namespace EFCoreBookStore.Models
{
    public partial 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 EFCoreBookStore.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/>

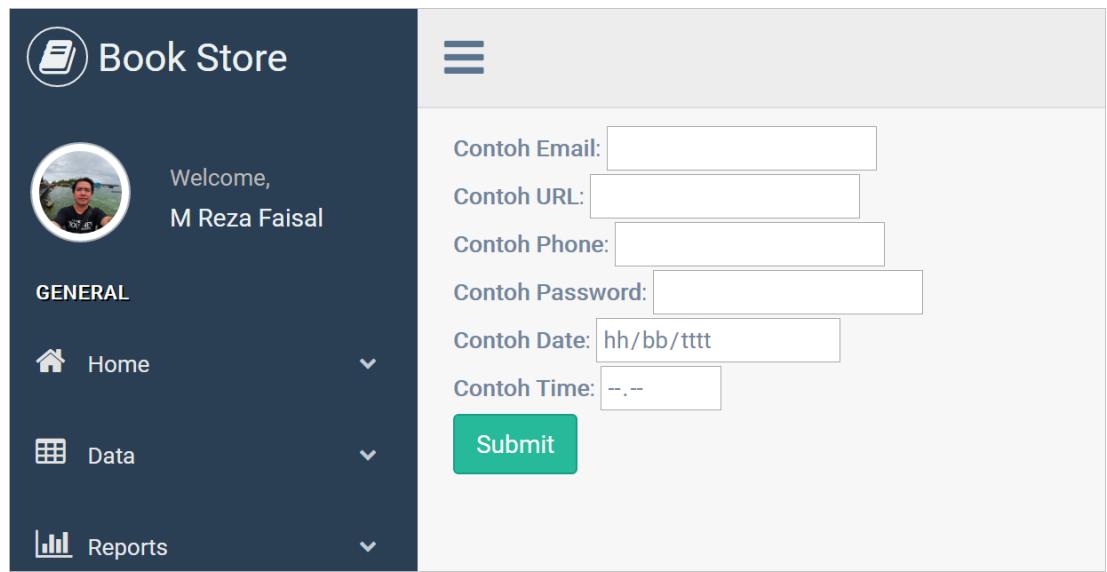
    <label asp-for="ContohDate"></label>:
    <input asp-for="ContohDate" />
    <br/>

    <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 94. 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 EFCoreBookStore.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 EFCoreBookStore.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.Linq;
using Microsoft.AspNetCore.Mvc;
using EFCoreBookStore.Models;

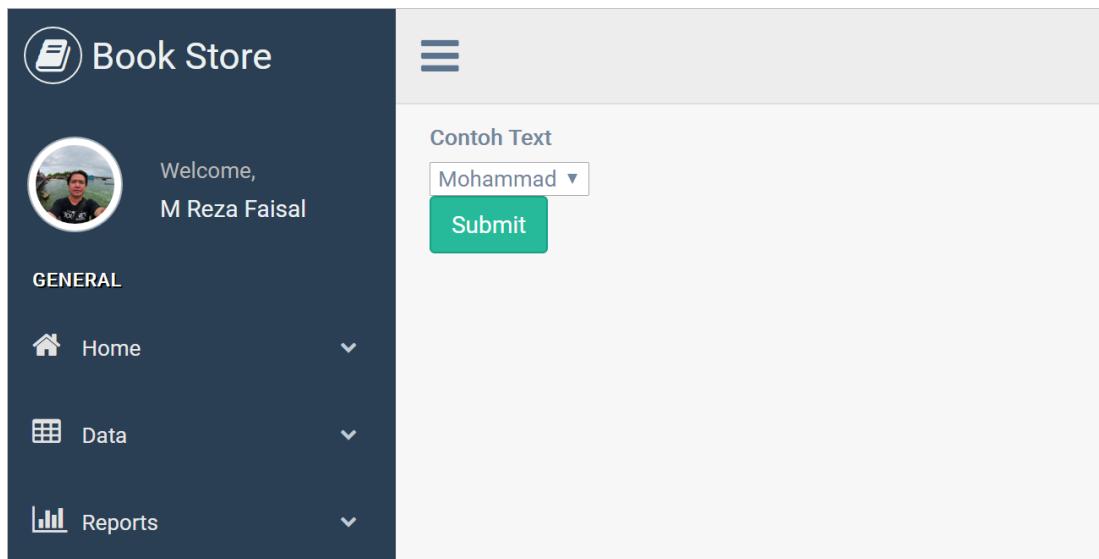
namespace EFCoreBookStore.Controllers
{
    public class LatihanController : Controller
    {

        [HttpGet]
        public IActionResult Index()
        {
            return View();
        }

        [HttpGet]
        public IActionResult Template()
        {
            BookStoreDataContext db = new BookStoreDataContext();
            ViewBag.Authors = db.Authors.ToList();
            return View();
        }
    }
}

```

Hasilnya dapat dilihat pada gambar di bawah ini.



**Gambar 95. 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>

```

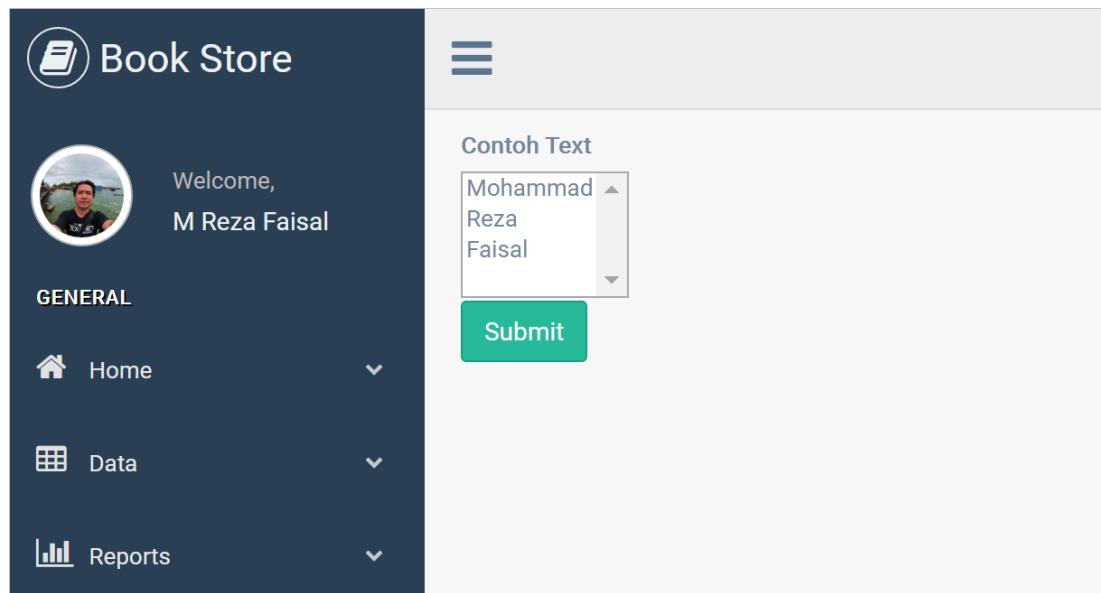
Tag <select> di atas hanya dapat digunakan untuk memilih satu nilai dari daftar nilai yang disediakan. Jika ingin menggunakan tag <select> agar dapat digunakan untuk memilih lebih dari satu nilai maka dapat digunakan atribut multiple seperti contoh berikut ini.

```

<select asp-for="ContohText"
       asp-items=
"@(new SelectList(ViewBag.Authors, "AuthorID", "Name"))"
       multiple="multiple">
</select>

```

Hasilnya dapat dilihat pada gambar di bawah ini.



Gambar 96. 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">Simpan</button>
<button type="reset">Batal</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 <span> 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 EFCoreBookStore.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.
  - Create.cshtml untuk form input untuk menambah data kategori buku.
  - Edit.cshtml untuk form mengedit data kategori buku.
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.
  - Create.cshtml untuk form input untuk menambah data pengarang buku.
  - Edit.cshtml untuk form mengedit data pengarang buku.
3. Mengelola buku
  - Index.cshtml yang akan menampilkan tabel daftar buku. Pada halaman ini juga terdapat fitur untuk menghapus data buku.
  - Create.cshtml untuk form input untuk menambah data buku.
  - Edit.cshtml untuk form mengedit data buku.

### Kategori

File-file ini disimpan pada folder Category di dalam folder Views.

#### **Index.cshtml**

Berikut ini adalah kode lengkap Index.cshtml pada folder View/Category. File ini berfungsi untuk menampilkan daftar kategori buku.

```

Index.cshtml
@model IQueryable<EFCoreBookStore.Models.Category>

<div class="">
    <div class="page-title">
        <div class="title_left">
            <h3>Books Categories</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>

```

```

                </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 Book Categories</h2>
                        <ul class="nav navbar-right panel_toolbox">
                            <li><a asp-controller="Category" 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>
                                            <input type="checkbox" id="check-all" class="flat">
                                        </th>
                                        <th class="column-title" style="width: 5%">@Html.DisplayNameFor(model => model.FirstOrDefault().CategoryID)</th>
                                        <th class="column-title" style="width: 85%">@Html.DisplayNameFor(model => model.FirstOrDefault().Name)</th>
                                        <th class="column-title no-link last" style="width: 10%"><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="a-center ">
                                                <input type="checkbox" class="flat" name="table_records">
                                            </td>
                                            <td class=" " >@item.CategoryID</td>
                                            <td class=" " >@item.Name</td>
                                            <td class=" last">
                                                <a asp-controller="Category" asp-action="Edit" asp-route-id="@item.CategoryID">Edit</a> | 
                                                <a asp-controller="Category" asp-action="Delete" asp-route-id="@item.CategoryID">Delete</a>
                                            </td>
                                        </tr>
                                        odd = !odd;
                                    }
                                </tbody>
                            </table>
                        </div>
                    </div>
                </div>
            </div>
        </div>
    </div>

```

Pada kode di atas diperlihatkan penggunaan beberapa sintaks Razor, HTML helper dan tag helper. Yang pertama adalah penggunaan sintaks Razor untuk menentukan model yang akan digunakan pada halaman view Index.cshtml.

```
@model IList<EFCoreBookStore.Models.Category>
```

Dari sintaks di atas dapat dilihat jika model yang digunakan adalah collection dari object class Category.

Selanjutnya adalah penggunaan tag helper untuk membuat tombol “Add Data” dengan cara berikut ini.

```
<a asp-controller="Category" asp-action="Create">
    <i class="fa fa-plus"></i> Add Data
</a>
```

Dari cara di atas akan dihasilkan link dengan alamat /Category/Create, link tersebut akan mengakses method action Create yang dimiliki oleh class controller CategoryController.cs. Untuk menampilkan kolom-kolom pada tabel digunakan HTML helper berikut ini. HTML helper ini akan mengambil nilai atribut property dari class model.

```
@Html.DisplayNameFor(model => model.FirstOrDefault().CategoryID)
@Html.DisplayNameFor(model => model.FirstOrDefault().Name)
```

Sedangkan untuk menampilkan data dari model sebagai baris-baris pada tabel digunakan pengulangan dengan menggunakan sintaks Razor berikut ini.

```
@{ var odd = false;
@foreach (var item in Model){
<tr class="@ (odd ? "odd": "even") pointer">
    .
    .
</tr>
odd = !odd;
}}
```

Pada kode di atas dapat dilihat pengulangan dilakukan terhadap tag <tr></tr> yang bertujuan untuk mengulangan pembuatan baris pada tabel. Pada kode di atas setiap baris akan menggunakan style yang berbeda, hal itu dapat dilakukan dengan cara pemrograman singkat seperti yang terlihat pada kode di atas. Pada kode di atas object Model merupakan collection object class Category, sehingga object item pada pengulangan dengan menggunakan foreach berisi sebuah object class Category. Sehingga dengan menggunakan sintaks Razor di bawah ini nilai property object class Category dapat ditampilkan.

```
@item.CategoryID
@item.Name
```

Yang terakhir adalah membuat link untuk mengedit dan menghapus record dengan menggunakan tag helper berikut ini.

```
<a asp-controller="Category" asp-action="Edit" asp-route-
id="@item.CategoryID">Edit</a>

<a asp-controller="Category" asp-action="Delete" asp-route-
id="@item.CategoryID">Delete</a>
```

Pada tag helper untuk membuat tag link di atas digunakan tiga atribut, yaitu:

- asp-controller adalah class controller yang dituju.
- asp-action adalah method action yang dituju.
- asp-route-id digunakan untuk mengirim id dari object class Category yang akan diedit atau dihapus. Atribut ini sebenarnya adalah hanya “asp-route” saja, penambahan “id” di belakang atribut ini bertujuan untuk menentukan variable yang digunakan. Sehingga untuk mengakses variable ini pada method action di dalam class controller digunakan cara seperti berikut ini.

```
public IActionResult Edit(int? id)
{
    . . .
}
```

Contoh lain jika ingin menggunakan nama variable “categoryId” maka atribut yang dapat digunakan adalah “asp-route-categoryId”. Sehingga untuk mengakses variable ini pada method action di dalam class controller digunakan cara seperti berikut ini.

```
public IActionResult Edit(int? categoryId)
{
    . . .
}
```

ID	Book Category Name	Action
1	Computer	Edit   Delete
2	History	Edit   Delete

Gambar 97. Book Store - Kategori buku - Index.cshtml.

### Create.cshtml

Berikut ini adalah kode lengkap Create.cshtml pada folder View/Category.

```
Create.cshtml
@model EFCoreBookStore.Models.Category



146


```

```

<form asp-controller="Category" 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" type="text" />
            <span asp-validation-for="Name" class="text-danger"></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">Cancel</button>
            <button class="btn btn-primary" type="reset">Reset</button>
            <button type="submit" class="btn btn-success">Submit</button>
        </div>
    </div>
</div>
</div>
</div>
</div>
</div>
</div>

```

File komponen Create.cshtml di atas berfungsi untuk menambah data kategori buku. Untuk menambah data kategori digunakan class entity model Category. Penentuan class entity model Category ini dapat dilihat pada awal baris kode di atas.

```
@model EFCoreBookStore.Models.Category
```

Untuk membuat form input data diperlukan tag helper <form> seperti berikut.

```
<form asp-controller="Category" asp-action="Create" . . . >
. . .
</form>
```

Dari tag helper di atas dapat dilihat, form akan mengirimkan data ke alamat /Category/Create, artinya data akan diproses oleh method action Create pada class controller CategoryController.cs. Secara default metode pengiriman data pada form menggunakan metode POST.

Selanjutnya adalah menambahkan tag helper untuk input data berikut ini.

```
...
<label asp-for="Name" . . . ></label>
<input asp-for="Name" . . . >
<span asp-validation-for="Name" class="text-danger"></span>
...
```

Dari kode di atas dapat dilihat 3 tag helper yaitu:

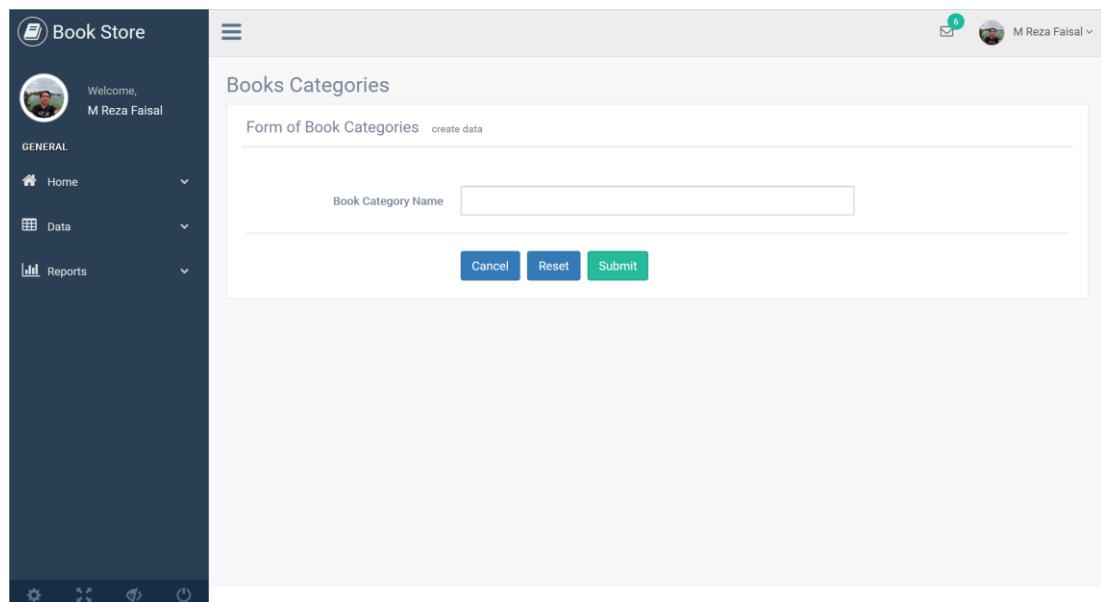
- Tag helper <label> menggunakan atribut asp-for dengan nilai "Name" yang akan menampilkan nilai atribut "Display" dari property Name yang dimiliki oleh class entity model Category.cs. Hasilnya akan ditampilkan label "Book Category Name" seperti yang terlihat pada gambar di bawah ini.

- Tag helper <input> sebagai komponen input nilai pada form. Tag helper ini menggunakan atribut asp-for dengan nilai "Name". Hal tersebut berfungsi untuk menentukan tipe input secara otomatis sesuai dengan tipe data dari property Name. Selain itu juga berfungsi untuk menentukan nilai property Name sesuai dengan nilai yang diisikan user pada input ini.
- Tag helper <span> menggunakan atribut asp-validation-for dengan nilai "Name" di atas berfungsi untuk menampilkan pesan kesalahan jika nilai yang dimasukkan tidak sesuai dengan syarat yang telah ditentukan untuk property Name pada class entity model Category.cs. Jika dilihat pada class tersebut maka terdapat 2 syarat validasi untuk property Name yaitu tidak boleh kosong dan nilai tidak boleh berisi lebih dari 256 karakter.

Selanjutnya adalah menambahkan <button> dengan tipe submit. Ketika tombol ini diklik oleh user maka proses validasi akan secara otomatis dilakukan, jika semua nilai pada input telah sesuai dengan syarat validasi maka object class Category akan dikirim untuk diproses oleh method action Create tipe HttPost pada class controller CategoryController.cs.

```
<button type="submit" class="btn btn-success">Submit</button>
```

Berikut ini adalah gambar antarmuka halaman Create.cshtml.



Gambar 98. Book Store - Kategori buku - Create.cshtml.

### Edit.cshtml

Berikut ini adalah kode lengkap Edit.cshtml pada folder View/Category.

```
Edit.cshtml
@model EFCoreBookStore.Models.Category



### Books Categories


```

```

<div class="col-md-12 col-sm-12 col-xs-12">
    <div class="x_panel">
        <div class="x_title">
            <h2>Form of Book Categories <small>edit data</small></h2>
            <div class="clearfix"></div>
        </div>
        <div class="x_content">
            <br />
            <form asp-controller="Category" 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="CategoryID" 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="CategoryID" readonly="readonly" required="required" class="form-control col-md-7 col-xs-12" type="text" value="CategoryID" />
                        <span asp-validation-for="CategoryID" class="ln_solid"></span>
                    </div>
                </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" type="text" value="Name" />
                        <span asp-validation-for="Name" class="ln_solid"></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">Cancel</button>
                        <button class="btn btn-primary" type="reset">Reset</button>
                        <button type="submit" class="btn btn-success">Submit</button>
                    </div>
                </div>
            </div>
        </div>
    </div>
</div>

```

Form di atas berfungsi untuk mengedit data kategori buku. Isi file Edit.cshtml ini berisi sintaks Razor dan tag helper yang mirip dengan isi file Create.cshtml. Perbedaan yang mendasar antara keduanya pada tag helper <form>.

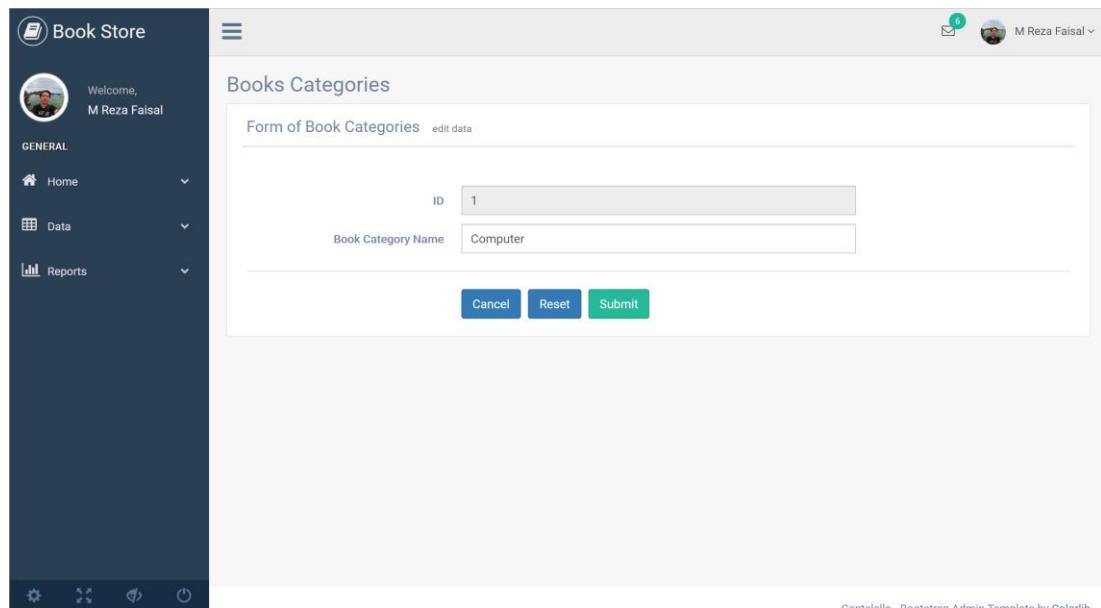
```

<form asp-controller="Category" asp-action="Edit" . . . >
    . . .
</form>

```

Dari nilai atribut asp-controller dan asp-action di atas maka object class Category yang telah diisi akan dikirim dan diproses oleh method action Edit tipe HttpPost pada class controller CategoryController.cs.

Berikut adalah antarmuka file Edit.cshtml.



**Gambar 99. Book Store - Kategori buku - Edit.cshtml.**

### Catatan

Dari ketiga file komponen view di atas, ketiganya menggunakan sebuah class entity model yaitu class Category.cs.

### Pengarang

File-file ini disimpan di dalam folder Author yang berada pada folder Views.

#### Index.cshtml

Berikut ini adalah kode lengkap Index.cshtml pada folder View/Author. File ini berfungsi untuk menampilkan daftar pengarang buku.

```
Index.cshtml
@model IQueryable<EFCoreBookStore.Models.Author>



150


```

```

<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 Authors</h2>
                <ul class="nav navbar-right panel_toolbox">
                    <li><a asp-controller="Author" 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>
                                    <input type="checkbox" id="check-all" class="flat">
                                </th>
                                <th class="column-title" style="width: 5%">@Html.DisplayNameFor(model => model.FirstOrDefault().AuthorID)</th>
                                <th class="column-title" style="width: 45%">@Html.DisplayNameFor(model => model.FirstOrDefault().Name)</th>
                                <th class="column-title" style="width: 40%">@Html.DisplayNameFor(model => model.FirstOrDefault().Email)</th>
                                <th class="column-title no-link last" style="width: 10%"><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="a-center ">
                                        <input type="checkbox" class="flat" name="table_records">
                                    </td>
                                    <td class=" " >@item.AuthorID</td>
                                    <td class=" " >@item.Name</td>
                                    <td class=" " >@item.Email</td>
                                    <td class=" last">
                                        <a asp-controller="Author" asp-action="Edit" asp-route-id="@item.AuthorID">Edit</a> | 
                                        <a asp-controller="Author" asp-action="Delete" asp-route-id="@item.AuthorID">Delete</a>
                                    </td>
                                </tr>
                                odd = !odd;
                            }
                        </tbody>
                    </table>
                </div>
            </div>
        </div>
    </div>
</div>

```

Pada kode di atas diperlihatkan penggunaan beberapa sintaks Razor, HTML helper dan tag helper. Yang pertama adalah penggunaan sintaks Razor untuk menentukan model yang akan digunakan pada halaman view Index.cshtml.

```
@model IList<EFCoreBookStore.Models.Author>
```

Dari sintaks di atas dapat dilihat jika model yang digunakan adalah collection dari object class Author.

Selanjutnya adalah penggunaan tag helper untuk membuat tombol “Add Data” dengan cara berikut ini.

```
<a asp-controller="Author" asp-action="Create">
    <i class="fa fa-plus"></i> Add Data
</a>
```

Dari cara di atas akan dihasilkan link dengan alamat /Author/Create, link tersebut akan mengakses method action Create yang dimiliki oleh class controller AuthorController.cs. Untuk menampilkan kolom-kolom pada tabel digunakan HTML helper berikut ini. HTML helper ini akan mengambil nilai atribut property dari class model.

```
@Html.DisplayNameFor(model => model.FirstOrDefault().AuthorID)
@Html.DisplayNameFor(model => model.FirstOrDefault().Name)
@Html.DisplayNameFor(model => model.FirstOrDefault().Email)
```

Sedangkan untuk menampilkan data dari model sebagai baris-baris pada tabel digunakan pengulangan dengan menggunakan sintaks Razor berikut ini.

```
@{ var odd = false;
@foreach (var item in Model){
<tr class="@(odd ? "odd": "even") pointer">
    ...
</tr>
odd = !odd;
}}
```

Pada kode di atas dapat dilihat pengulangan dilakukan terhadap tag <tr> </tr> yang bertujuan untuk mengulangan pembuatan baris pada tabel. Pada kode di atas setiap baris akan menggunakan style yang berbeda, hal itu dapat dilakukan dengan cara pemrograman singkat seperti yang terlihat pada kode di atas. Pada kode di atas object Model merupakan collection object class Author, sehingga object item pada pengulangan dengan menggunakan foreach berisi sebuah object class Author. Sehingga dengan menggunakan sintaks Razor di bawah ini nilai property object class Author dapat ditampilkan.

```
@item.AuthorID
@item.Name
@item.Email
```

Yang terakhir adalah membuat link untuk mengedit dan menghapus record dengan menggunakan tag helper berikut ini.

```
<a asp-controller="Author" asp-action="Edit" asp-route-
id="@item.AuthorID">Edit</a>

<a asp-controller="Author" asp-action="Delete" asp-route-
id="@item.AuthorID">Delete</a>
```

Pada tag helper untuk membuat tag link di atas digunakan tiga atribut, yaitu:

- asp-controller adalah class controller yang dituju.
- asp-action adalah method action yang dituju.
- asp-route-id digunakan untuk mengirim id dari object class Author yang akan diedit atau dihapus. Atribut ini sebenarnya adalah hanya “asp-route” saja, penambahanan “id” di belakang atribut ini bertujuan untuk menentukan

variable yang digunakan. Sehingga untuk mengakses variable ini pada method action di dalam class controller digunakan cara seperti berikut ini.

```
public IActionResult Edit(int? id)
{
    . . .
}
```

Contoh lain jika ingin menggunakan nama variable “authorId” maka atribut yang dapat digunakan adalah “asp-route-authorId”. Sehingga untuk mengakses variable ini pada method action di dalam class controller digunakan cara seperti berikut ini.

```
public IActionResult Edit(int? authorId)
{
    . . .
}
```

Berikut ini adalah antarmuka file Index.cshtml.

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
6	Adi Faisal	adi@adi.com	Edit   Delete

Gambar 100. Book Store - Pengarang - Index.cshtml.

### Create.cshtml

Berikut ini adalah kode lengkap Create.cshtml pada folder View/Author.

```
Create.cshtml
@model EFCoreBookStore.Models.Author



### Author



Create | Index



## Form of Author <small>create data</small>



Cancel | Save


```

```
<br />
<form asp-controller="Author" 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">Cancel</button>
            <button class="btn btn-
primary" type="reset">Reset</button>
            <button type="submit" class="btn btn-
success">Submit</button>
        </div>
    </div>
</form>
</div>
</div>
</div>
</div>
</div>
</div>
```

File komponen Create.cshtml di atas berfungsi untuk menambah data pengarang buku. Untuk menambah data pengarang buku digunakan class entity model Author. Penentuan class entity model Author ini dapat dilihat pada awal baris kode di atas.

```
@model EFCoreBookStore.Models.Author
```

Untuk membuat form input data diperlukan tag helper <form> seperti berikut.

```
<form asp-controller="Author" asp-action="Create" . . . >
. . .
</form>
```

Dari tag helper di atas dapat dilihat, form akan mengirimkan data ke alamat /Author/Create, artinya data akan diproses oleh method action Create pada class controller AuthorController.cs. Secara default metode pengiriman data pada form menggunakan metode POST.

Selanjutnya adalah menambahkan tag helper untuk input data berikut ini.

```
<label asp-for="Name" . . .></label>
<input asp-for="Name" . . .>
<span asp-validation-for="Name"></span>
```

```
<label asp-for="Email" . . .></label>
<input asp-for="Email" . . .>
<span asp-validation-for="Email"></span>
```

Dari kode di atas dapat dilihat 3 tag helper yaitu:

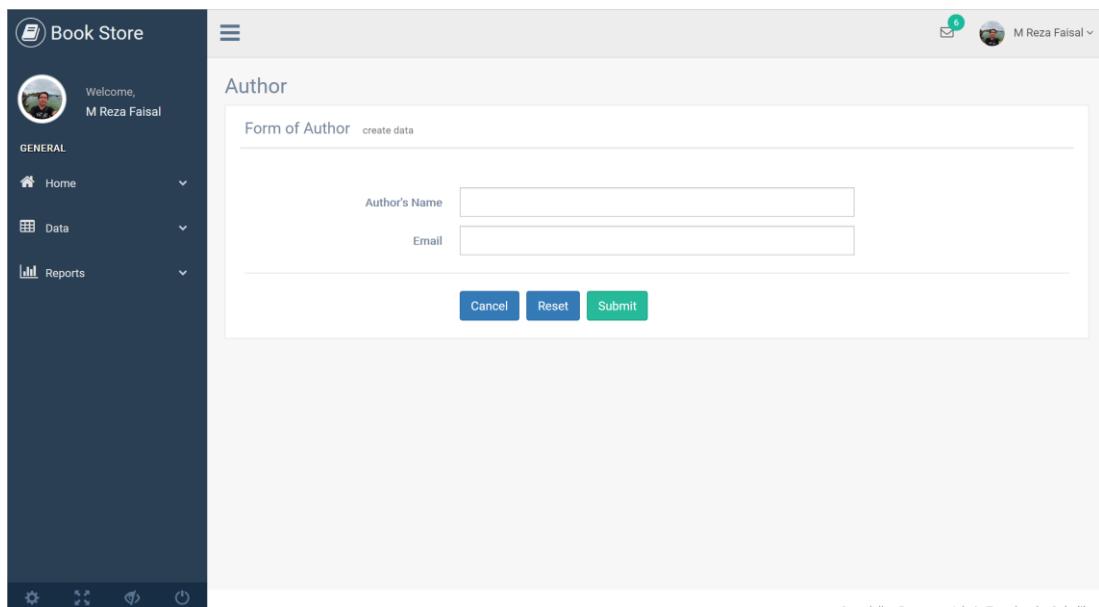
- Tag helper <label> menggunakan atribut asp-for dengan nilai "Name" yang akan menampilkan nilai atribut "Display" dari property Name yang dimiliki oleh class entity model Author.cs. Hasilnya akan ditampilkan label "Author's Name" seperti yang terlihat pada gambar di bawah ini.
- Tag helper <input> sebagai komponen input nilai pada form. Tag helper ini menggunakan atribut asp-for dengan nilai "Name". Hal tersebut berfungsi untuk menentukan tipe input secara otomatis sesuai dengan tipe data dari property Name. Selain itu juga berfungsi untuk menentukan nilai property Name sesuai dengan nilai yang diisikan user pada input ini.
- Tag helper <span> menggunakan atribut asp-validation-for dengan nilai "Name" di atas berfungsi untuk menampilkan pesan kesalahan jika nilai yang dimasukkan tidak sesuai dengan syarat yang telah ditentukan untuk property Name pada class entity model Author.cs. Jika dilihat pada class tersebut maka terdapat 2 syarat validasi untuk property Name yaitu tidak boleh kosong dan nilai tidak boleh berisi lebih dari 256 karakter.

Selanjutnya ketiga tag helper di atas digunakan kembali untuk property Email seperti yang dapat dilihat pada penggalan kode di atas.

Sedangkan keperluan mengirimkan data yang telah diisi pada form maka perlu dilakukan penambahan <button> dengan tipe submit. Ketika tombol ini diklik oleh user maka proses validasi akan secara otomatis dilakukan, jika semua nilai pada input telah sesuai dengan syarat validasi maka object class Author akan dikirim untuk diproses oleh method action Create tipe HttPost pada class controller AuthorController.cs.

```
<button type="submit" class="btn btn-success">Submit</button>
```

Antarmuka file Create.cshtml dapat dilihat pada gambar di bawah ini.



Gambar 101. Book Store - Pengarang - Create.cshtml.

## Edit.cshtml

Berikut ini adalah kode lengkap Edit.cshtml pada folder View/Author.

```
>Edit.cshtml
@model EFCoreBookStore.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 of Author <small>edit data</small></h2>
                    <div class="clearfix"></div>
                </div>
                <div class="x_content">
                    <br />
                    <form asp-controller="Author" 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="AuthorID" 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="AuthorID" readonly="readonly" required="required" class="form-
control col-md-7 col-xs-12">
                            </div>
                        </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">Cancel</button>
                                    </div>
                                </div>
                            </div>
                        </div>
                    </form>
                </div>
            </div>
        </div>
    </div>
</div>
```

```

        <button class="btn btn-primary" type="reset">Reset</button>
        <button type="submit" class="btn btn-success">Submit</button>
    </div>

```

Form di atas berfungsi untuk mengedit data pengarang buku. Isi file Edit.cshtml ini berisi sintaks Razor dan tag helper yang mirip dengan isi file Create.cshtml. Perbedaan yang mendasar antara keduanya pada tag helper <form>.

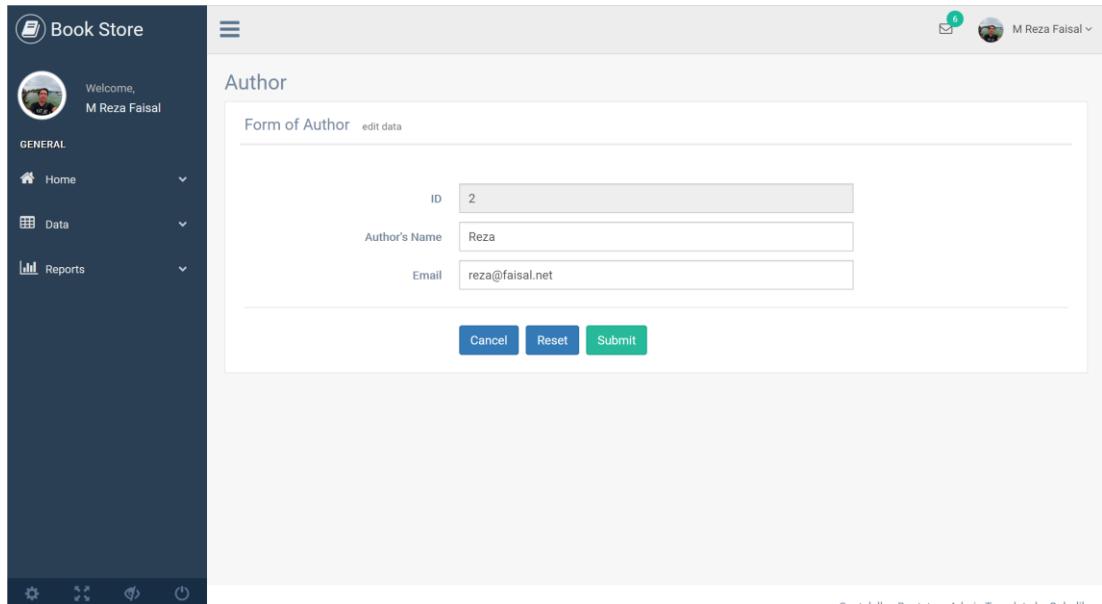
```

<form asp-controller="Author" asp-action="Edit" . . . >
. . .
</form>

```

Dari nilai atribut asp-controller dan asp-action di atas maka object class Author yang telah diisi akan dikirim dan diproses oleh method action Edit tipe HttpPost pada class controller AuthorController.cs.

Antarmuka file Edit.cshtml dapat dilihat pada gambar di bawah ini.



**Gambar 102. Book Store - Pengarang - Edit.cshtml..**

### **Catatan**

Dari ketiga file komponen view di atas, ketiganya menggunakan sebuah class entity model yaitu class Author.cs.

## Buku

File-file ini disimpan di dalam folder Books yang berada pada folder Views.

### Index.cshtml

Berikut ini adalah kode lengkap Index.cshtml pada folder View/Book. File ini berfungsi untuk menampilkan daftar buku.

```
Index.cshtml
@model IList<EFCoreBookStore.Models.BookViewModel>



### Books



Go!



## List of Books



- <a href="#" asp-action="Create"><i class="fa fa-plus"></i> Add Data</a>



| <input class="flat" id="checkbox-all" type="checkbox"/> | Cover | Category | Title | Publish Date |
|---------------------------------------------------------|-------|----------|-------|--------------|
|                                                         |       |          |       |              |


```

```
 Qty | Action |
@{ var odd = false; }
@foreach (var item in Model){
|  |  |  |  |  |  |  | |
|---|---|---|---|---|---|---|---|
|  |  | @item.CategoryName | @item.Title  ISBN: @item.ISBN  Authors: @item.AuthorNames  Price: @item.Price | @item.PublishDate | @item.Quantity | Edit |  Delete |

odd = !odd;
}

```

Pada kode di atas diperlihatkan penggunaan beberapa sintaks Razor, HTML helper dan tag helper. Yang pertama adalah penggunaan sintaks Razor untuk menentukan model yang akan digunakan pada halaman view Index.cshtml.

```
@model IList<EFCoreBookStore.Models.BookViewModel>
```

Dari sintaks di atas dapat dilihat jika model yang digunakan adalah collection dari object class BookViewModel.

Selanjutnya adalah penggunaan tag helper untuk membuat tombol “Add Data” dengan cara berikut ini.

```
<a asp-controller="Book" asp-action="Create">
    <i class="fa fa-plus"></i> Add Data
</a>
```

Dari cara di atas akan dihasilkan link dengan alamat /Book/Create, link tersebut akan mengakses method action Create yang dimiliki oleh class controller BookController.cs. Untuk

Untuk menampilkan data dari model sebagai baris-baris pada tabel digunakan pengulangan dengan menggunakan sintaks Razor berikut ini.

```
@{ var odd = false; }
@foreach (var item in Model){
<tr class="@(odd ? "odd": "even") pointer">
. .
</tr>
odd = !odd;
}
```

Pada kode di atas dapat dilihat pengulangan dilakukan terhadap tag <tr></tr> yang bertujuan untuk mengulangan pembuatan baris pada tabel. Pada kode di atas setiap baris akan menggunakan style yang berbeda, hal itu dapat dilakukan dengan cara pemrograman singkat seperti yang terlihat pada kode di atas. Pada kode di atas object Model merupakan collection object class BookViewModel, sehingga object item pada pengulangan dengan menggunakan foreach berisi sebuah object class BookViewModel. Sehingga dengan menggunakan sintaks Razor di bawah ini nilai property object class BookViewModel dapat ditampilkan.

```
@item.CategoryName
@item.Title
@item.ISBN
@item.AuthorNames
@item.Price
@item.PublishDate
@item.Quantity
```

Sedangkan untuk menampilkan gambar digunakan tag helper berikut ini.

```

```

Yang terakhir adalah membuat link untuk mengedit dan menghapus record dengan menggunakan tag helper berikut ini.

```
<a asp-controller="Book" asp-action="Edit" asp-route-
id="@item.ISBN">Edit</a>

<a asp-controller="Book" asp-action="Delete" asp-route-
id="@item.ISBN">Delete</a>
```

Pada tag helper untuk membuat tag link di atas digunakan tiga atribut, yaitu:

- asp-controller adalah class controller yang dituju.
- asp-action adalah method action yang dituju.
- asp-route-id digunakan untuk mengirim id dari object class BookViewModel yang akan diedit atau dihapus. Atribut ini sebenarnya adalah hanya “asp-route” saja, penambahanan “id” di belakang atribut ini bertujuan untuk menentukan variable yang digunakan. Sehingga untuk mengakses variable ini pada method action di dalam class controller digunakan cara seperti berikut ini.

```
public IActionResult Edit(int? id)
{
. .
}
```

Contoh lain jika ingin menggunakan nama variable “isbn” maka atribut yang dapat digunakan adalah “asp-route-isbn”. Sehingga untuk mengakses variable ini pada method action di dalam class controller digunakan cara seperti berikut ini.

```
public IActionResult Edit(int? isbn)
{
. .
}
```

Gambar di bawah ini adalah antarmuka dari file Index.cshtml.

The screenshot shows the 'Book Store' application interface. On the left, there is a sidebar with a user profile picture of 'M Reza Faisal', a general menu with 'Home', 'Data' (selected), 'Reports', and system settings, and a navigation tree under 'Data' for 'Book Categories', 'Authors', and 'Books'. The main content area is titled 'Books' and displays a table titled 'List of Books'. The table has columns for 'Cover', 'Category', 'Title', 'Publish Date', 'Qty', and 'Action'. A single row is shown for a book titled 'Seri Belajar ASP.NET : ASP.NET MVC Untuk Pemula', categorized as 'Computer', published on 1/1/2017 at 12:00:00 AM, with a quantity of 10. The 'Action' column contains links for 'Edit' and 'Delete'. There is also a search bar and a 'Add Data' button at the top right of the table.

Gambar 103. Book Store - Buku - Index.cshtml.

### Create.cshtml

Berikut ini adalah kode lengkap Create.cshtml pada folder View/Book.

```
Create.cshtml
@model EFCoreBookStore.Models.BookFormViewModel



### Book



View All



## Form of Book <small>create data</small>



<form method="POST" enctype="multipart/form-data" asp-controller="Book" asp-action="Create" data-parsley-validate class="form-horizontal form-label-left">



* ISBN



* Title



* Category



* Author


```

```

        <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>

```

```
<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">Cancel</button>
        <button class="btn btn-
primary" type="reset">Reset</button>
        <button type="submit" class="btn btn-
success">Submit</button>
    </div>
    </div>
</div>
</div>
</div>
</div>
</div>
</div>
```

File komponen Create.cshtml di atas berfungsi untuk menambah data buku. Untuk menambah data buku digunakan class view model BookFormViewModel untuk pengiriman data dari komponen view ke komponen controller. Sedangkan untuk menyimpan data ke database digunakan class entity model Book dan BookAuthor yang nanti dilihat penggunaanya pada sub bab Controller. Penentuan class view model BookFormViewModel ini dapat dilihat pada awal baris kode di atas.

```
@model EFCoreBookStore.Models.BookFormViewModel
```

Untuk membuat form input data diperlukan tag helper <form> seperti berikut.

```
<form asp-controller="Book" asp-action="Create" . . . >
. . .
</form>
```

Dari tag helper di atas dapat dilihat, form akan mengirimkan data ke alamat /Book/Create, artinya data akan diproses oleh method action Create pada class controller BookController.cs. Secara default metode pengiriman data pada form menggunakan metode POST.

Selanjutnya adalah menambahkan tag helper untuk input data berikut ini.

```
<label asp-for="ISBN" . . .></label>
<input asp-for="ISBN" . . .>
<span asp-validation-for="ISBN"></span>

<label asp-for="CategoryID" . . .></label>
<select asp-for="CategoryID" asp-items="@ViewBag.Categories" . . .>
<option>Choose Category</option>
</select>
<span asp-validation-for="CategoryID"></span>

<label asp-for="Title" . . .></label>
<input asp-for="Title" . . .>
<span asp-validation-for="Title"></span>

<label asp-for="PublishDate" . . .></label>
<input asp-for="PublishDate" . . .>
<span asp-validation-for="PublishDate"></span>

<label asp-for="Price" . . .></label>
<input asp-for="Price" . . .>
```

```

<span asp-validation-for="Price"></span>
<label asp-for="Quantity" . . .></label>
<input asp-for="Quantity" . . .>
<span asp-validation-for="Quantity"></span>
<label asp-for="Photo" . . .></label>
<input asp-for="Photo" type="file" . . .>

```

Dari kode di atas dapat dilihat 3 tipe tag helper yaitu:

- Tag helper <label> pada kode di atas berfungsi untuk membuat label dengan nilai sesuai dengan atribut “Display” dari property yang diisikan pada atribut asp-for.
- Tag helper untuk mengisi data adalah tag helper <input> dan <select>. Tag helper <input> terdiri atas dua tipe input untuk mengisi data text seperti untuk mengisi nilai ISBN, title, publish date, price, quantity. Tipe yang lain adalah input untuk memilih file foto yang akan diupload. Pada input untuk upload foto digunakan tambahan atribut type=”file”.
- Tag helper <span> yang digunakan untuk menampilkan pesan validasi dengan menambahkan atribut asp-validation-for yang diisi dengan nama property. Pesan dari setiap tag helper ini akan sesuai dengan nilai atribut validasi dari setiap property.

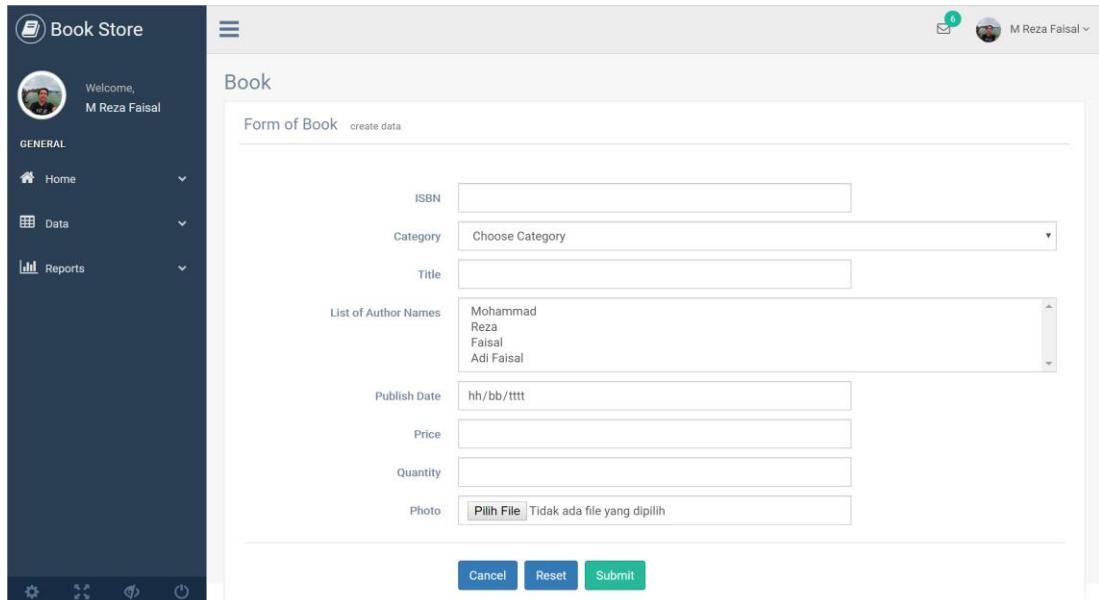
Sedangkan keperluan mengirimkan data yang telah diisi pada form maka perlu dilakukan penambahan <button> dengan tipe submit. Ketika tombol ini diklik oleh user maka proses validasi akan secara otomatis dilakukan, jika semua nilai pada input telah sesuai dengan syarat validasi maka object class BookFormViewModel akan dikirim untuk diproses oleh method action Create tipe HttPost pada class controller BookController.cs.

```

<button type="submit" class="btn btn-success">Submit</button>

```

Antarmuka file Create.cshtml dapat dilihat pada gambar di bawah ini.



Gambar 104. Book Store - Buku - Create.cshtml.

## Edit.cshtml

Berikut ini adalah kode lengkap dari file Edit.cshtml.

```
Editor.cshtml
@model EFCoreBookStore.Models.BookFormViewModel



### Book



## Form of Book <small>create data</small></h2>



<div class="asp-validation-for" asp-for="ISBN"></div>
                            
                            <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 class="form-group">
                            <label asp-for="CategoryID" class="control-label col-md-3 col-sm-3 col-xs-12">
                                <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>
                            </label>
                            <div class="form-group">
                                <label asp-for="Title" class="control-label col-md-3 col-sm-3 col-xs-12">
                                    <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>
                                </label>
                                <div class="form-group">
                                    <label asp-for="AuthorIDs" class="control-label col-md-3 col-sm-3 col-xs-12">
                                        <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">
                                                </select>
                                            </div>
                                        </label>
                                    </div>
                                </div>
                            </div>
                        </div>


```

```

    <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>
<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">Cancel</button>
<button class="btn btn-primary" type="reset">Reset</button>
<button type="submit" class="btn btn-success">Submit</button>
</div>
</div>
</form>
</div>
</div>
</div>
</div>
</div>
</div>
</div>

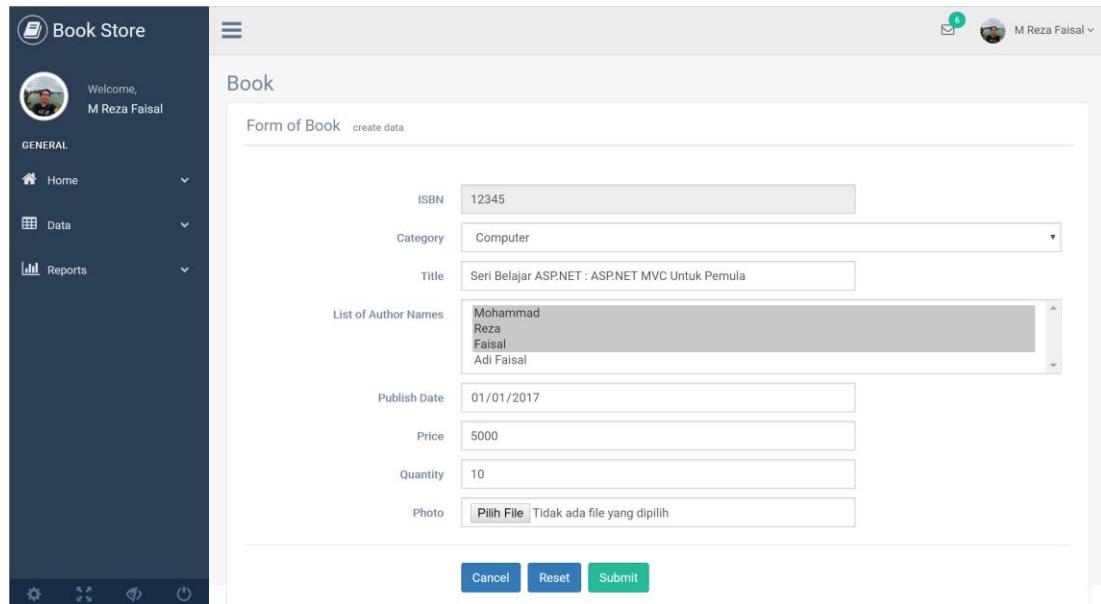
```

Form di atas berfungsi untuk mengedit data buku. Isi file Edit.cshtml ini berisi sintaks Razor dan tag helper yang mirip dengan isi file Create.cshtml. Perbedaan yang mendasar antara keduanya pada tag helper <form>.

```
<form asp-controller="Book" asp-action="Edit" . . . >
. . .
</form>
```

Dari nilai atribut asp-controller dan asp-action di atas maka object class Book yang telah diisi akan dikirim dan diproses oleh method action Edit tipe HttpPost pada class controller BookController.cs.

Antarmuka file Edit.cshtml dapat dilihat pada gambar di bawah ini.



Gambar 105. Book Store - Buku - Edit.cshtml.

### Catatan

Ada perbedaan antara pengelolaan kategori dan pengarang buku dengan pengelolaan buku. Pada pengelolaan kategori dan pengarang buku cukup digunakan class entity model pada seluruh file komponen view, yaitu pada file Index.cshtml, Create.cshtml dan Edit.cshtml. Sedangkan pada pengelolaan buku, tidak menggunakan class entity model pada komponen view, tetapi menggunakan class view model. Class view model yang digunakan pada halaman Index.cshtml adalah BookViewModel.cs, sedangkan pada halaman Create.cshtml dan Edit.cshtml digunakan class view model BookFormViewModel.cs.

Pada komponen view untuk mengelola buku ini tidak ada penggunaan class entity model. class entity model Book.cs hanya akan digunakan pada komponen controller.

---

## 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 Category maka nama file yang harus digunakan adalah CategoryController.cs. Contoh lain, jika nama controller adalah Book maka nama file yang harus digunakan adalah BookController.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:

- Attribut adalah

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:

- CategoryController.cs.
- AuthorController.cs.
- BookController.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 LatihanViewBag, selanjutnya tambahkan file LatihanViewBag.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 LatihanViewBag.cshtml berikut ini.

```
@ ViewBag.VariableInt <br/>
@ ViewBag.VariableString <br/>
```

Contoh penggunaan object ViewBag lainnya dapat dapat dilihat pada bab Pengenalan ASP.NET Core MVC.

## 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 diperlihatkan 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 ouput 2 \* x. Sedangkan untuk untuk contoh kasus penggunaan extension method Select() di atas, input p berisi nilai dari db.Authors. Sedangkan yang menjadi ouput 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 ouput 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**

---

### **Class Controller Category**

Untuk fitur pengelolaan kategori buku dapat digunakan untuk:

- Menampilkan daftar kategori buku.
- 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.

Berikut ini adalah kode lengkap dari class CategoryController.cs.

```
CategoryController.cs
using System.Linq;
using Microsoft.AspNetCore.Mvc;
using EFCoreBookStore.Models;

namespace EFCoreBookStore.Controllers
{
    public class CategoryController : Controller
    {
        BookStoreDataContext db = new BookStoreDataContext();

        [HttpGet]
        public IActionResult Index()
        {
            var items = db.Categories.Select(p => p);

            return View(items);
        }

        [HttpGet]
        public IActionResult Create()
        {
            return View();
        }

        [HttpPost]
        public IActionResult Create(Category item)
        {
            if(ModelState.IsValid){
                db.Add(item);
                db.SaveChanges();

                return RedirectToAction("Index");
            }

            return View();
        }

        [HttpGet]
        public IActionResult Edit(int? id)
        {
```

```

        var item =
db.Categories.SingleOrDefault(p=>p.CategoryID.Equals(id));

        return View(item);
    }

[HttpPost]
[ValidateAntiForgeryToken]
public IActionResult Edit([Bind("CategoryID,Name")] Category item)
{
    if(ModelState.IsValid){
        db.Update(item);
        db.SaveChanges();

        return RedirectToAction("Index");
    }

    return View();
}

[HttpGet]
public IActionResult Delete(int id)
{
    if(ModelState.IsValid)
    {
        var item = db.Categories.Find(id);
        db.Categories.Remove(item);
        db.SaveChanges();

        return RedirectToAction("Index");
    }

    return View();
}
}

```

#### **Method Action [HttpGet] Index**

Untuk fitur menampilkan daftar kategori buku dapat dilihat pada method action Index dengan atribut [HttpGet]. Pada method action ini dapat dilihat pengambilan data dari tabel categories dengan menggunakan Entity Framework Core dan LINQ.

```
var items = db.Categories.Select(p => p);
```

Objek items akan menyimpan data dari tabel category dalam bentuk IQueryable. Selanjutnya object items akan dikirim ke komponen view Index.cshtml dengan cara berikut.

```
return View(items);
```

#### **Method Action [HttpGet] Create**

Method action ini bertujuan untuk menampilkan form untuk menambah data kategori buku. File komponen view yang digunakan sebagai form ini adalah Create.cshtml.

#### **Method Action [HttpPost] Create**

Method action ini bertujuan untuk menyimpan object Category yang property-propertynya 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(Category item)
{
    . . .
}
```

Selanjutnya untuk memvalidasi item sebelum disimpan dapat dilakukan dengan cara berikut.

```
if (ModelState.IsValid) {
    . . .
}
```

Sedangkan untuk menyimpan object Category dapat dilakukan dengan cara berikut.

```
db.Add(item);
db.SaveChanges();
```

Setelah data sukses disimpan, selanjutnya akan kembali ditampilkan daftar kategori buku dengan menggunakan baris berikut ini.

```
return RedirectToAction("Index");
```

### **Method Action [HttpGet] Edit**

Method action ini bertujuan untuk menampilkan form untuk mengedit data kategori buku yang dipilih. Untuk mendapatkan data kategori buku yang akan diedit maka pada method ini memiliki parameter input untuk menerima kiriman id kategori buku dari komponen view Index.cshtml.

```
public IActionResult Edit(int? id)
{
    . . .
}
```

Dengan memanfaatkan parameter input id tersebut maka dapat dilakukan pengambilan data dengan menggunakan cara berikut ini.

```
var item = db.Categories.SingleOrDefault(p=>p.CategoryID.Equals(id));
```

Data kategori 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 bertujuan untuk menyimpan object Category yang property-propertynya 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("CategoryID,Name")] Category item)
{
    . . .
}
```

Pada parameter input di atas dapat dilihat object item tipe Category. 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) {  
    . . .  
}
```

Kemudian digunakan kode berikut ini untuk mengupdate object yang dipilih ke dalam database.

```
db.Update(item);  
db.SaveChanges();
```

Setelah data sukses disimpan, selanjutnya akan kembali ditampilkan daftar kategori buku dengan menggunakan baris berikut ini.

```
return RedirectToAction("Index");
```

### **Method Action [HttpGet] Delete**

Method ini berfungsi untuk menghapus data kategori buku yang dipilih. Agar method ini dapat menerima id kategori 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)  
{  
    . . .  
}
```

Langkah selanjutnya adalah memilih data yang akan dihapus dengan baris berikut ini.

```
var item = db.Categories.Find(id);
```

Setelah object didapatkan maka data dihapus dengan kode berikut.

```
db.Categories.Remove(item);  
db.SaveChanges();
```

Setelah data sukses dihapus, selanjutnya akan kembali ditampilkan daftar kategori buku dengan menggunakan baris berikut ini.

```
return RedirectToAction("Index");
```

## **Class Controller Author**

Untuk fitur pengelolaan pengarang buku dapat digunakan untuk:

- Menampilkan daftar pengarang buku.
- 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.

Berikut ini adalah kode lengkap dari class AuthorController.cs.

```
AuthorController.cs
using System.Linq;
using Microsoft.AspNetCore.Mvc;
using EFCoreBookStore.Models;

namespace EFCoreBookStore.Controllers
{
    public class AuthorController : Controller
    {
        BookStoreDataContext db = new BookStoreDataContext();

        [HttpGet]
        public IActionResult Index()
        {
            var items = db.Authors.Select(p => p);

            return View(items);
        }

        [HttpGet]
        public IActionResult Create()
        {
            return View();
        }

        [HttpPost]
        public IActionResult Create(Author item)
        {
            if (ModelState.IsValid){
                db.Add(item);
                db.SaveChanges();

                return RedirectToAction("Index");
            }

            return View();
        }

        [HttpGet]
        public IActionResult Edit(int? id)
        {
            var item = db.Authors.SingleOrDefault(p=>p.AuthorID.Equals(id));

            return View(item);
        }

        [HttpPost]
        [ValidateAntiForgeryToken]
        public IActionResult Edit([Bind("AuthorID,Name, Email")] Author item)
        {
            if (ModelState.IsValid){
                db.Update(item);
            }
        }
    }
}
```

```

        db.SaveChanges();

        return RedirectToAction("Index");
    }

    return View(item);
}

[HttpGet]
public IActionResult Delete(int id)
{
    if(ModelState.IsValid)
    {
        var item = db.Authors.Find(id);
        db.Authors.Remove(item);
        db.SaveChanges();

        return RedirectToAction("Index");
    }

    return View();
}
}

```

### **Method Action [HttpGet] Index**

Untuk fitur menampilkan daftar pengarang buku dapat dilihat pada method action Index dengan atribut [HttpGet]. Pada method action ini dapat dilihat pengambilan data dari tabel authors dengan menggunakan Entity Framework Core dan LINQ.

```
var items = db.Authors.Select(p => p);
```

Objek items akan menyimpan data dari tabel author dalam bentuk IQueryable. Selanjutnya object items akan dikirim ke komponen view Index.cshtml dengan cara berikut.

```
return View(items);
```

### **Method Action [HttpGet] Create**

Method action ini bertujuan untuk menampilkan form untuk menambah data pengarang buku. File komponen view yang digunakan sebagai form ini adalah Create.cshtml.

### **Method Action [HttpPost] Create**

Method action ini bertujuan untuk menyimpan object Author yang property-propertynya telah diisi melalui form pada file Create.cshtml. Agar method ini dapat menerima kirim dari file Create.cshtml maka dapat dilihat parameter input item seperti pada kode berikut ini.

```
public IActionResult Create(Author item)
{
    . . .
}
```

Selanjutnya untuk memvalidasidi item sebelum disimpan dapat dilakukan dengan cara berikut.

```
if (ModelState.IsValid) {
    . . .
}
```

```
}
```

Sedangkan untuk menyimpan object Author dapat dilakukan dengan cara berikut.

```
db.Add(item);  
db.SaveChanges();
```

Setelah data sukses disimpan, selanjutnya akan kembali ditampilkan daftar pengarang buku dengan menggunakan baris berikut ini.

```
return RedirectToAction("Index");
```

### **Method Action [HttpGet] Edit**

Method action ini bertujuan untuk menampilkan form untuk mengedit data pengarang buku yang dipilih. Untuk mendapatkan data pengarang 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)  
{  
    . . .  
}
```

Dengan memanfaatkan parameter input id tersebut maka dapat dilakukan pengambilan data dengan menggunakan cara berikut ini.

```
var item = db.Authors.SingleOrDefault(p=>p.AuthorID.Equals(id));
```

Data kategori 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 bertujuan untuk menyimpan object Author yang property-propertynya 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("AuthorID,Name, Email")] Author item)  
{  
    . . .  
}
```

Pada parameter input di atas dapat dilihat object item tipe Author. 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){  
    . . .  
}
```

Kemudian digunakan kode berikut ini untuk mengupdate object yang dipilih ke dalam database.

```
db.Update(item);  
db.SaveChanges();
```

Setelah data sukses disimpan, selanjutnya akan kembali ditampilkan daftar pengarang buku dengan menggunakan baris berikut ini.

```
return RedirectToAction("Index");
```

### **Method Action [HttpGet] Delete**

Method ini berfungsi untuk menghapus data pengarang buku yang dipilih. Agar method ini dapat menerima id pengarang 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)
{
    . . .
}
```

Langkah selanjutnya adalah memilih data yang akan dihapus dengan baris berikut ini.

```
var item = db.Authors.Find(id);
```

Setelah object didapatkan maka data dihapus dengan kode berikut.

```
db.Authors.Remove(item);
db.SaveChanges();
```

Setelah data sukses dihapus, selanjutnya akan kembali ditampilkan daftar pengarang buku dengan menggunakan baris berikut ini.

```
return RedirectToAction("Index");
```

## **Class Controller Book**

Untuk fitur pengelolaan buku dapat digunakan untuk:

- Menampilkan daftar buku.
- 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.

Berikut ini adalah kode lengkap dari class BookController.cs.

```
BookController.cs
using System;
using System.Linq;
using System.Collections.Generic;
using System.IO;
```

```

using System.Net.Http.Headers;
using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.Mvc.Rendering;
using Microsoft.AspNetCore.Hosting;
using EFCoreBookStore.Models;

namespace EFCoreBookStore.Controllers
{
    public class BookController : Controller
    {
        BookStoreDataContext db = new BookStoreDataContext();
        private IHostingEnvironment _environment;

        public BookController(IHostingEnvironment environment)
        {
            _environment = environment;
        }

        [HttpGet]
        public IActionResult Index()
        {
            var bookList = db.Books.ToList();

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

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

                var category = db.Categories.Where(p=>p.CategoryID.Equals(book.CategoryID)).Single<Category>();
                item.CategoryName = category.Name;

                string authorNameList = string.Empty;
                var booksAuthorsList = db.BooksAuthors.Where(p=>p.ISBN.Equals(book.ISBN));
                foreach (BookAuthor booksAuthors in booksAuthorsList)
                {
                    BookStoreDataContext db2 = new BookStoreDataContext();
                    var author = db2.Authors.Where(p=>p.AuthorID.Equals(booksAuthors.AuthorID)).Single<Author>();
                    authorNameList = authorNameList + author.Name + ", ";
                }
                item.AuthorNames = authorNameList.Substring(0, authorNameList.Length - 2);

                items.Add(item);
            }

            return View(items);
        }

        [HttpGet]
        public IActionResult Create()
        {
            ViewBag.Categories = new SelectList(db.Categories.ToList(), "CategoryID", "Name");
            ViewBag.Authors = new MultiSelectList(db.Authors.ToList(), "AuthorID", "Name");

            return View();
        }
    }
}

```

```

[HttpPost]
public IActionResult Create(BookFormViewModel item)
{
    if (ModelState.IsValid){
        Book book = new Book();
        book.ISBN = item.ISBN;
        book.CategoryID = item.CategoryID;
        book.Title = item.Title;
        book.PublishDate = item.PublishDate;
        book.Price = item.Price;
        book.Quantity = item.Quantity;
        db.Add(book);

        foreach(int authorId in item.AuthorIDs){
            BookAuthor bookAuthor = new BookAuthor();
            bookAuthor.ISBN = item.ISBN;
            bookAuthor.AuthorID = authorId;
            db.Add(bookAuthor);
        }

        db.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();
}

[HttpGet]
public IActionResult Edit(int? id)
{
    ViewBag.Categories = new SelectList(db.Categories.ToList(),
"CategoryID", "Name");
    ViewBag.Authors = new MultiSelectList(dbAuthors.ToList(),
"AuthorID", "Name");

    var book = db.Books.SingleOrDefault(p=>p.ISBN.Equals(id));

    BookFormViewModel item = new BookFormViewModel();
    item.ISBN = book.ISBN;
    item.Title = book.Title;
    item.PublishDate = book.PublishDate;
    item.Price = book.Price;
    item.Quantity = book.Quantity;
    item.CategoryID = book.CategoryID;

    var authorList = db.BooksAuthors.Where(p =>
p.ISBN.Equals(book.ISBN)).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,
    PublishDate, Price, Quantity, AuthorIDs")] BookFormViewModel item)
    {
        if (ModelState.IsValid){
            db.BooksAuthors.RemoveRange(db.BooksAuthors.Where(p =>
p.ISBN.Equals(item.ISBN)));
            db.SaveChanges();

            Book book = db.Books.SingleOrDefault(p =>
p.ISBN.Equals(item.ISBN));
            book.CategoryID = item.CategoryID;
            book.Title = item.Title;
            book.PublishDate = item.PublishDate;
            book.Price = item.Price;
            book.Quantity = item.Quantity;
            db.Update(book);

            foreach(int authorId in item.AuthorIDs){
                BookAuthor bookAuthor = new BookAuthor();
                bookAuthor.ISBN = item.ISBN;
                bookAuthor.AuthorID = authorId;
                db.Add(bookAuthor);
            }

            db.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();
}

[HttpGet]
public IActionResult Delete(int id)
{
    if(ModelState.IsValid)
    {
        var item = db.Books.Find(id);
        db.BooksAuthors.RemoveRange(db.BooksAuthors.Where(p =>
p.ISBN.Equals(item.ISBN)));
        db.SaveChanges();

        db.Books.Remove(item);
        db.SaveChanges();

        return RedirectToAction("Index");
    }

    return View();
}

```

```

    }
}

```

### **Method Action [HttpGet] Index**

Untuk fitur menampilkan daftar buku dapat dilihat pada method action Index dengan atribut [HttpGet]. Untuk menampilkan data buku seperti di bawah ini maka perlu mengambil data dari tabel categories, authors, books dan books\_authors.

Cover	Category	Title	Publish Date	Qty	Action
	Computer	Seri Belajar ASP.NET : ASP.NET MVC Untuk Pemula ISBN: 12345 Authors: Mohammad, Reza, Faisal Price: 5000	1/1/2017 12:00:00 AM	10	Edit   Delete
	History	History 101 ISBN: 223344 Authors: Reza Price: 22	12/12/2012 12:00:00 AM	22	Edit   Delete

**Gambar 106. Daftar buku.**

Langkah pertama yang dilakukan adalah mengambil data buku dari tabel books dengan cara berikut ini.

```
var bookList = db.Books.ToList();
```

Cara di atas adalah cara lain untuk mendapatkan seluruh data selain yang telah diterangkan pada class controller CategoryController.cs dan AuthorController.cs. 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.ISBN;
    item.Title = book.Title;
    item.Photo = book.Photo;
    item.PublishDate = book.PublishDate;
    item.Price = book.Price;
    item.Quantity = book.Quantity;
    ...
}
```

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.

Pada object book hanya memiliki nilai CategoryID, untuk mendapatkan nama kategori buku yang sesuai dengan nilai CategoryID maka digunakan kode berikut ini.

```
var category =
    db.Categories.Where(p=>p.CategoryID.Equals(book.CategoryID)).Single<Category>();

item.CategoryName = category.Name;
```

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.

```
string authorNameList = string.Empty;

var booksAuthorsList = db.BooksAuthors.Where(p=>p.ISBN.Equals(book.ISBN));
```

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.

```
foreach(BookAuthor booksAuthors in booksAuthorsList){
    BookStoreDataContext db2 = new BookStoreDataContext();

    var author =
    db2.Authors.Where(p=>p.AuthorID.Equals(booksAuthors.AuthorID)).Single<Author>();

    authorNameList = authorNameList + author.Name + ", ";
}
```

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] Create**

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 107. Form menambah buku.**

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 cara berikut.

```
ViewBag.Categories = new SelectList(db.Categories.ToList(), "CategoryID", "Name");

ViewBag.Authors = new MultiSelectList(db.Authors.ToList(), "AuthorID", "Name");
```

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**

Method action ini bertujuan untuk menyimpan nilai-nilai dari object BookFormViewModel yang property-propertynya 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 memvalidasi 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.ISBN = item.ISBN;
book.CategoryID = item.CategoryID;
book.Title = item.Title;
book.PublishDate = item.PublishDate;
book.Price = item.Price;
book.Quantity = item.Quantity;
db.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.ISBN = item.ISBN;
    bookAuthor.AuthorID = authorId;
    db.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.

```

db.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**

Method action ini bertujuan untuk menampilkan form untuk mengedit data pengarang buku yang dipilih. Untuk mendapatkan data pengarang 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(db.Categories.ToList(), "CategoryID",
"Name");

```

```
ViewBag.Authors = new MultiSelectList(db.Authors.ToList(), "AuthorID", "Name");
```

Kemudian dengan memanfaatkan parameter input id tersebut maka dapat dilakukan pengambilan data dengan menggunakan cara berikut ini.

```
var book = db.Books.SingleOrDefault(p=>p.ISBN.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.ISBN;
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 = db.BooksAuthors.Where(p =>
p.ISBN.Equals(book.ISBN)).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**

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.

```
db.BooksAuthors.RemoveRange(db.BooksAuthors.Where(p =>
    p.ISBN.Equals(item.ISBN)));
db.SaveChanges();
```

Langkah selanjutnya adalah memilih object buku yang diedit untuk ditampung pada object entity model.

```
Book book = db.Books.SingleOrDefault(p => p.ISBN.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;
db.Update(book);
```

Kemudian menyimpan data pengarang buku yang dipilih dengan cara berikut ini.

```
foreach(int authorId in item.AuthorIDs) {
    BookAuthor bookAuthor = new BookAuthor();
    bookAuthor.ISBN = item.ISBN;
    bookAuthor.AuthorID = authorId;
    db.Add(bookAuthor);
}

db.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**

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)
{
    . . .
}
```

Langkah selanjutnya adalah memilih data yang akan dihapus dengan baris berikut ini.

```
var item = db.Books.Find(id);
```

Selanjutnya adalah menghapus data pengarang buku dari tabel books\_authors berdasarkan ISBN buku yang akan dihapus dengan menggunakan kode berikut ini.

```
db.BooksAuthors.RemoveRange(db.BooksAuthors.Where(p =>
    p.ISBN.Equals(item.ISBN)));
db.SaveChanges();
```

Selanjutnya menghapus buku yang dipilih.

```
db.Books.Remove(item);
db.SaveChanges();
```

Setelah data sukses dihapus, selanjutnya akan kembali ditampilkan daftar buku dengan menggunakan baris berikut ini.

```
return RedirectToAction("Index");
```

# 6

## **Otentikasi dan Otorisasi**

Salah satu pengamanan yang umum digunakan adalah dengan melindungi halaman dengan pemeriksaan otentikasi dengan terlebih dahulu dengan 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 otentikasi dan otorisasi dengan ASP.NET Core dan database MySQL.

---

### **Library Otentikasi & Otorisasi**

Pada sub bab ini dijelaskan tentang library-library otentikasi dan otorisasi pada framework ASP.NET Core.

---

#### **ASP.NET Identity**

Pada tahun 2005 telah dikenal ASP.NET Membership yang berfungsi untuk mengelola proses otentikasi 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. Kedua buku tersebut dapat diunduh pada link yang telah diberikan pada sub bab Bahan Pendukung > Buku.

Sekarang ini telah dikenal ASP.NET Identity, yang memungkinkan pengelolaan proses otentikasi yang lebih bebas. Termasuk kemampuan untuk melakukan otentikasi 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. Penjelasan dan contoh implementasi ASP.NET Identity dapat dibaca pada buku berjudul Seri Belajar ASP.NET : ASP.NET MVC Untuk Pemula. Buku ini dapat diunduh pada link yang diberikan pada sub bab Bahan Pendukung > Buku.

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 otentikasi dan otorisasi yang dapat langsung digunakan oleh software developer. Sebagai contoh, telah ada method untuk proses SignIn dan SignOut yang langsung dapat digunakan.

## Cookie Authentication Middleware

---

Cookie Authentication Middleware atau disingkat Cookie Middleware adalah library yang dapat digunakan untuk mempermudah implementasi otentikasi dan otorisasi.

Penggunaan library ini tidak bergantung pada aturan tertentu dalam penyimpanan data user dan role seperti pada library ASP.NET Membership dan ASP.NET Identity. Sehingga software developer dapat dengan leluasa membuat design tabel untuk menyimpan user dan role sesuai dengan kebutuhan.

Library ini tidak selengkap kedua library yang telah dijelaskan di atas. Sebagai contoh, library ini tidak memiliki method SignIn dan SignOut untuk proses login dan logout. Sehingga method ini harus dibuat sendiri.

Berikut adalah langkah-langkah implementasi Cookie Middleware untuk proses otentikasi dan otorisasi:

- Form Login, user harus memasukkan username dan password.
- Jika username dan password cocok maka menyimpan identitas dengan menggunakan class Claim, ClaimIdentity dan ClaimPrincipal.
- Melakukan proses login dengan menggunakan method HttpContext.Authentication.SignInAsync dan menyimpan informasi ke dalam cookie.

Pada buku ini akan digunakan library ini untuk proses otentikasi dan otorisasi. Penjelasan detail penggunaan library ini akan dijelaskan pada sub bab selanjutnya.

## Implementasi

---

Pada sub bab ini diberikan contoh sederhana implementasi otentikasi dan otorisasi. Langkah pertama adalah menyiapkan project EFCoreBookStore dengan memodifikasi file EFCoreBookStore.csproj dan Startup.cs.

### Modifikasi File EFCoreBookStore.csproj

Modifikasi ini bertujuan untuk menambah library untuk implementasi otentikasi dengan menggunakan library Microsoft.AspNetCore.Authentication.Cookies. Library ini adalah implementasi ASP.NET Core middleware yang mampu mengaktifkan aplikasi agar menggunakan otentikasi berbasis cookie.

Untuk menambahkan library ini pada aplikasi web Book Store dapat dilakukan dengan cara memodifikasi file EFCoreBookStore.csproj seperti kode di bawah ini.

```
EFCoreBookStore.csproj
<Project Sdk="Microsoft.NET.Sdk.Web">

  <PropertyGroup>
    <TargetFramework>netcoreapp1.1</TargetFramework>
  </PropertyGroup>

  <ItemGroup>
    <Folder Include="wwwroot\" />
  </ItemGroup>

  <ItemGroup>
    <PackageReference Include="Microsoft.AspNetCore" Version="1.1.1" />
    <PackageReference Include="Microsoft.AspNetCore.Mvc" Version="1.1.2" />
    <PackageReference Include="Microsoft.AspNetCore.StaticFiles" Version="1.1.1" />
    <PackageReference Include="Microsoft.Extensions.Logging.Debug" Version="1.1.1" />
    <PackageReference Include="Microsoft.VisualStudio.Web.BrowserLink" Version="1.1.0" />
    <PackageReference Include="MySQL.Data.Core" Version="7.0.4-IR-191" />
    <PackageReference Include="Microsoft.EntityFrameworkCore" Version="1.1.1" />
    <PackageReference Include="Microsoft.EntityFrameworkCore.Tools" Version="1.1.0" />
  </ItemGroup>
```

```

<PackageReference Include=" MySql.Data.EntityFrameworkCore" Version="7.0.4-IR-191" />
<PackageReference Include=" Microsoft.AspNetCore.Mvc.TagHelpers" Version="1.1.2" />
<PackageReference Include=" Microsoft.AspNetCore.Authentication.Cookies" Version="1.1.1" />
</ItemGroup>
</Project>

```

## Modifikasi Startup.cs

Langkah selanjutnya adalah menambah beberapa baris kode untuk pengelolaan cookies dan setting untuk menangani pengelolaan otentikasi.

```

Startup.cs
using Microsoft.AspNetCore.Builder;
using Microsoft.AspNetCore.Hosting;
using Microsoft.Extensions.DependencyInjection;
using Microsoft.Extensions.Logging;
using Microsoft.Extensions.Configuration;
using Microsoft.AspNetCore.Http;
using System;
using System.Linq;
using System.Security.Claims;
using System.Threading.Tasks;

using EFCoreBookStore.Models;

namespace EFCoreBookStore
{
    public class Startup
    {
        public static IConfigurationRoot Configuration { get; set; }

        public Startup(IHostingEnvironment env)
        {
            var builder = new ConfigurationBuilder()
                .SetBasePath(env.ContentRootPath)
                .AddJsonFile("appsettings.json", optional: true,
reloadOnChange: true)
                .AddEnvironmentVariables();

            Configuration = builder.Build();
        }
        // 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)
        {
            services.AddMvc();

            BookStoreDataContext.ConnectionString =
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, ILoggerFactory loggerFactory)
        {
            loggerFactory.AddConsole();

            if (env.IsDevelopment())
            {
                app.UseDeveloperExceptionPage();
            }
            else
            {

```

```

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

    app.UseStaticFiles();

    app.UseCookieAuthentication(new CookieAuthenticationOptions
    {
        AuthenticationScheme = "Cookies",
        AutomaticAuthenticate = true,
        AutomaticChallenge = true,
        LoginPath = new PathString("/ContohOtentikasi/Login"),
        AccessDeniedPath = new PathString("/Home/AccessDenied/")
    });

    app.UseClaimsTransformation(context =>
    {
        if (context.Principal.Identity.IsAuthenticated)
        {
            context.Principal.Identities.First().AddClaim(new Claim("now", DateTime.Now.ToString()));
        }

        return Task.FromResult(context.Principal);
    });

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

```

Pada kode di atas dapat dilihat penambahan 2 hal, yaitu:

- Setting penggunaan otentikasi berbasis cookie.
- Setting transformasi Claim.

Untuk setting penggunaan otentikasi berbasis cookie dapat dilihat kode berikut di dalam method `Configure`.

```

app.UseCookieAuthentication(new CookieAuthenticationOptions
{
    AuthenticationScheme = "Cookies",
    AutomaticAuthenticate = true,
    AutomaticChallenge = true,
    LoginPath = new PathString("/ContohOtentikasi/Login")
});

```

Kode `app.UserCookieAuthentication()` ini ditulis di atas `app.UserMvc()`. Parameter di dalam method `UserCookieAuthentication` berisi object dari instansiasi class `CookieAuthenticationOption`. Class ini berisi property-property sebagai berikut:

- `AuthenticationScheme` adalah skema otentikasi yang digunakan yaitu Cookies.
- `AutomaticAuthenticate` bernilai true.
- `AutomaticChallenge` bernilai true.
- `LoginPath` adalah alamat halaman login. Untuk aplikasi web Book Store ini halaman login dikelola oleh method action `Login` dari class controller `ContohOtentikasi.cs`. Class ini akan dibuat pada sub bab selanjutnya.

Selanjutnya untuk melakukan transformasi Claim maka terlebih dahulu harus ditambahkan namespace berikut ini.

```
using System.Security.Claims;
```

Kemudian pada method Configure ditambahkan kode berikut ini, kode ini terletak di atas baris app.UserMvc().

```
app.UseClaimsTransformation(context =>
{
    if (context.Principal.Identity.IsAuthenticated)
    {
        context.Principal.Identities.First().AddClaim(new Claim("now",
DateTime.Now.ToString()));
    }

    return Task.FromResult(context.Principal);
});
```

Kode di atas berfungsi untuk menangani atribut waktu pada identitas yang berhasil melakukan proses otentikasi.

## Contoh Kasus

Contoh kasus sederhana pada sub bab ini akan memperlihatkan halaman Secret.cshtml yang hanya bisa diakses oleh user yang telah melewati proses otentikasi. Proses otentikasi dilakukan dengan cara memasukkan username dan password pada form login. Untuk pengiriman data username dan password digunakan model UserLoginFormViewModel.cs berikut ini.

Berikut ini adalah isi dari class view model UserLoginFormViewModel.cs.

```
UserLoginFormViewModel.cs
using System;
using System.ComponentModel.DataAnnotations;

namespace EFCoreBookStore.Models
{
    public partial 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;}
    }
}
```

Selanjutnya adalah membuat class controller ContohOtentikasiController.cs dengan kode lengkap berikut ini.

```
ContohOtentikasiController.cs
using System;
using System.Linq;
using System.Collections.Generic;
using System.IO;
using System.Net.Http.Headers;
using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.Mvc.Rendering;
using Microsoft.AspNetCore.Hosting;
using System.Security.Cryptography;
using System.Text;
using System.Security.Claims;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Authorization;
```

```

using EFCoreBookStore.Models;

namespace EFCoreBookStore.Controllers
{
    public class ContohOtentikasiController : Controller
    {
        [HttpGet]
        public IActionResult Login()
        {
            return View();
        }

        [HttpPost]
        public async Task<IActionResult> Login(UserLoginFormViewModel item)
        {
            if(ModelState.IsValid){
                if(item.UserName.Equals("admin") &&
item.Password.Equals("rahasia")){
                    var claims = new List<Claim>
                    {
                        new Claim("username", "admin"),
                        new Claim(ClaimTypes.Name, "M Reza Faisal"),
                        new Claim(ClaimTypes.Email, "admin@faisal.net"),
                        new Claim(ClaimTypes.Role, "admin")
                    };
                    var id = new ClaimsIdentity(claims, "password");
                    var principal = new ClaimsPrincipal(id);

                    await HttpContext.Authentication.SignInAsync("Cookies", principal);
                }
            }
            return View();
        }

        [HttpGet]
        public async Task<IActionResult> Logout()
        {
            await HttpContext.Authentication.SignOutAsync("Cookies");
            return RedirectToAction("Login");
        }

        [HttpGet]
        [Authorize]
        public IActionResult Secret()
        {
            return View();
        }
    }
}

```

Dari kode di atas dapat dilihat class controller ContohOtentikasiController.cs memiliki 4 method action, yaitu:

- [HttpGet] Login, method action ini berfungsi untuk menampilkan form login.
- [HttpPost] Login, method action berfungsi untuk melakukan otentikasi berdasarkan nilai username dan password yang dimasukkan. Pada contoh ini dapat dilihat proses otentikasi tidak melibatkan data dari database, tapi dilakukan secara manual. Kemudian proses membuat object claims yang berisi

- daftar informasi tentang user tersebut. Dan terakhir adalah proses otentikasi dengan mengeksekusi method SignInAsync(). Method ini berisi 2 parameter, parameter pertama adalah skema otentikasi yang digunakan yaitu Cookies. Parameter yang kedua adalah informasi user yang berhasil melakukan otentikasi.
- [HttpGet] Logout, method action ini berfungsi untuk melakukan proses logout dengan cara mengeksekusi method SignOutAsync().
  - [HttpPost] Secret, method action ini berfungsi untuk menampilkan file Secret.cshtml. Jika diperhatikan pada method action ini memiliki tambahan atribut [Authorize]. Atribut ini menyatakan bahwa method action ini hanya dapat diakses jika telah melewati proses otentikasi. Ini adalah contoh otorisasi, yaitu proses yang menentukan method action apa yang dapat diakses oleh user.

Untuk komponen view dibuat file Login.cshtml dengan kode lengkap berikut ini.

```
Login.cshtml
@{
    Layout = null;
}
@model EFCoreBookStore.Models.UserLoginFormViewModel

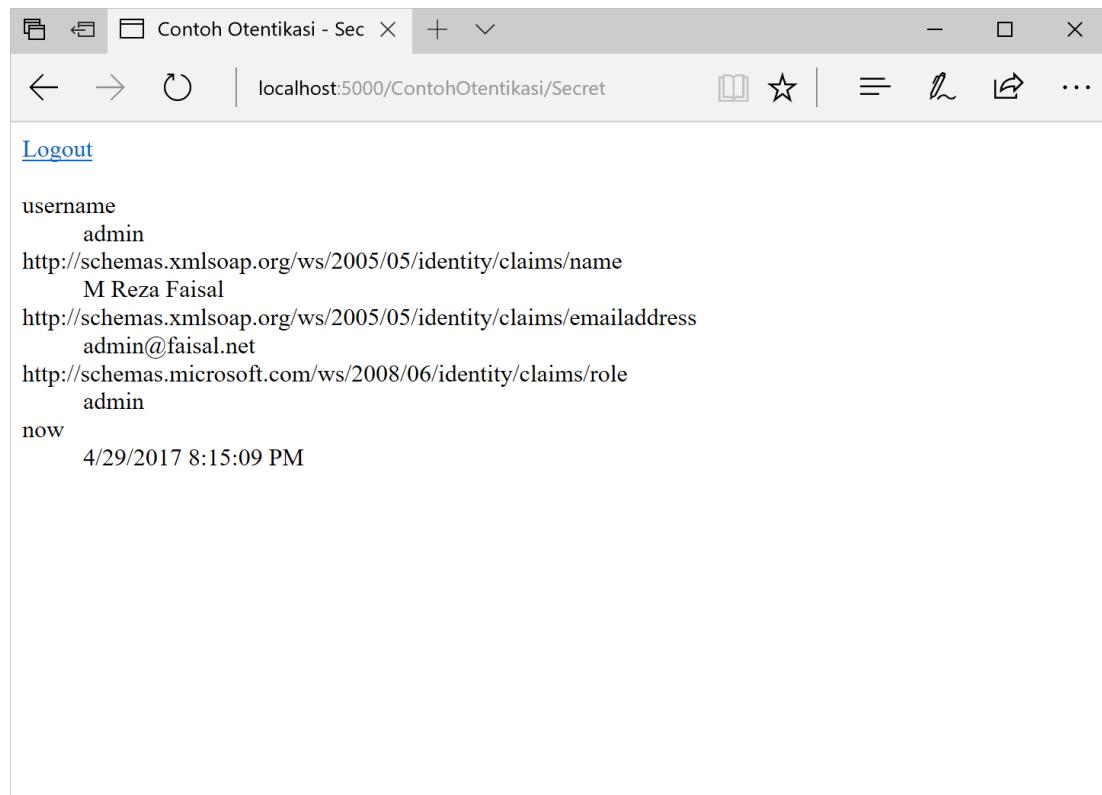
<html>
    <head>
        <title>Contoh Otentikasi - Login</title>
    </head>
    <body>
        <form asp-controller="ContohOtentikasi" asp-action="Login">
            Username: <input asp-for="UserName" /><br/>
            Password: <input asp-for="Password" /><br/>
            <button type="submit" class="btn btn-success">Login</button>
        </form>
    </body>
</html>
```

Dan berikut ini adalah kode file Secret.cshtml.

```
Secret.cshtml
@{
    Layout = null;
}

<html>
    <head>
        <title>Contoh Otentikasi - Secret</title>
    </head>
    <body>
        <a asp-controller="ContohOtentikasi" asp-action="Logout">
            Logout
        </a>
        <br/><br/>
        @foreach (var claim in User.Claims)
        {
            <dt>@claim.Type</dt>
            <dd>@claim.Value</dd>
        }
    </body>
</html>
```

Berikut adalah tampilan dari halaman ini.



Gambar 108. Contoh kasus otentikasi.

## Catatan

Dari penjelasan dan contoh di atas dapat dilihat beberapa hal penting yang terkait dengan proses otentikasi. Karena sebuah halaman komponen view ditampilkan oleh method action maka untuk melindungi suatu halaman agar hanya bisa diakses ketika user telah terotentikasi dapat dilakukan dengan memberikan atribut [Authorize]. Sehingga memberikan atribut ini pada seluruh method action akan membuat halaman komponen view dapat diakses setelah melewati proses otentikasi.

Tetapi harus diperhatikan tidak seluruh method action harus diberikan atribut [Authorize], sebagai contoh adalah method action untuk menampilkan form login dan method action untuk proses otentikasi.

Atribut [Authorize] juga dapat digunakan untuk melakukan otorisasi. Artinya dapat ditentukan role yang dapat mengakses method action tersebut. Sebagai contoh jika terdapat 2 role yaitu Admin dan User. Implementasi otorisasi dengan atribut ini dapat dilihat pada contoh di bawah ini.

```
[HttpGet]
[Authorize(Roles = "Admin")]
public IActionResult Index()
{
    . . .
}
```

Pada sebuah method action dapat dibuat agar bisa diakses oleh lebih dari satu role saja. Sebagai contoh jika method action ingin diakses oleh dua role maka dapat ditulis seperti contoh di bawah ini.

```
[Authorize(Roles = "Admin, User")]
```

## Persiapan

Setelah mendapatkan penjelasan mengenai cara otentikasi dan otorisasi dengan menggunakan Cookie Middleware, maka pada sub bab ini diberikan contoh implementasi otentikasi dan otorisasi dengan menggunakan data user dan role yang disimpan pada database. Oleh karena itu pada sub bab ini juga diberikan diberikan pengelolaan data role dan user.

## Modifikasi File Startup.cs

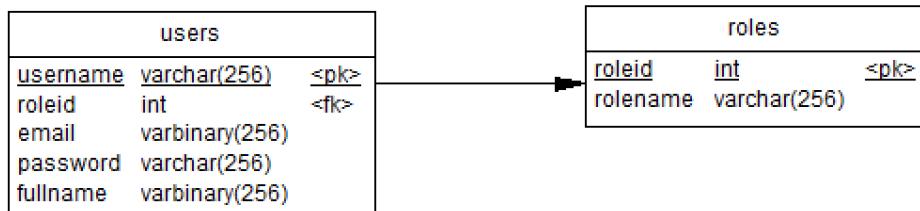
Untuk proses proses login dan logout akan dikelola oleh class controller HomeController.cs. Sehingga perlu dilakukan modifikasi file Startup.cs untuk setting otentikasi pada method Configure. Berikut adalah kode yang telah diubah.

```
app.UseCookieAuthentication(new CookieAuthenticationOptions
{
    AuthenticationScheme = "Cookies",
    AutomaticAuthenticate = true,
    AutomaticChallenge = true,
    LoginPath = new PathString("/Home/Login"),
    AccessDeniedPath = new PathString("/Home/AccessDenied/")
});
```

Modifikasi hanya dilakukan pada baris LoginPath dan menambahkan baris AccessDeniedPath untuk menentukan url yang akan diakses ketika user melanggar hak otorisasi.

## Database

Pada sub bab ini akan dibuat tabel untuk menyimpan data user dan role. Berikut ini adalah tabel user dan roles akan digunakan untuk kebutuhan otentikasi dan otorisasi.



Gambar 109. Tabel users dan roles pada database BookStore.

Berikut adalah kode SQL untuk membuat kedua tabel di atas.

```
create table users
(
    username      varchar(50) not null,
    roleid        int,
    email         varbinary(256),
    password      varchar(256),
    fullname      varbinary(256),
    primary key (username)
);

create table roles
(
    roleid        int not null,
    rolename      varchar(256),
    primary key (roleid)
);
```

```
alter table users add constraint FK_REFERENCE_5 foreign key (roleid)
references roles (roleid) on delete restrict on update restrict;
```

## Class Entity Model

Langkah kedua adalah melakukan persiapan class entity model sebagai representasi kedua tabel users dan roles. Class entity model User.cs sebagai representasi tabel users dan class entity model Role.cs sebagai representasi tabel roles.

Berikut ini adalah kode class User.cs.

```
User.cs
using System;
using System.ComponentModel.DataAnnotations;

namespace EFCoreBookStore.Models
{
    public partial class User{
        [Display(Name ="Username")]
        [Required(ErrorMessage = "{0} harus diisi.")]
        public int UserName {set; get; }

        [Display(Name ="Role")]
        [Required(ErrorMessage = "{0} harus diisi.")]
        public int RoleID {set; get; }

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

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

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

Dan berikut ini adalah isi dari file class Role.cs.

```
Role.cs
using System;
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations;

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

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

```
}
```

## Class Data Context

Selanjutnya adalah melakukan pemetaan antara class entity model dengan tabel dengan cara melakukan modifikasi pada class BookStoreDataContext. Dengan menambahkan kode berikut pada method OnModelCreating.

```
modelBuilder.Entity<User>().ToTable("users");
modelBuilder.Entity<User>(entity =>
{
    entity.Property(e => e.UserName).HasColumnName("username");
    entity.Property(e => e.RoleID).HasColumnName("roleid");
    entity.Property(e => e.Email).HasColumnName("email");
    entity.Property(e => e.Password).HasColumnName("password");
    entity.Property(e => e.Fullname).HasColumnName("fullname");
});
modelBuilder.Entity<User>().HasKey(e => new { e.UserName });

modelBuilder.Entity<Role>().ToTable("roles");
modelBuilder.Entity<Role>(entity =>
{
    entity.Property(e => e.RoleID).HasColumnName("roleid");
    entity.Property(e => e.RoleName).HasColumnName("rolename");
});
modelBuilder.Entity<Role>().HasKey(e => new { e.RoleID});
```

Kemudian menambahkan property-property berikut ini pada class BookStoreDataContext.

```
public virtual DbSet<User> Users { get; set; }
public virtual DbSet<Role> Roles { get; set; }
```

Sehingga dapat dilihat kode lengkap class BookStoreDataContext sebagai berikut.

```
BookStoreDataContext.cs
using Microsoft.EntityFrameworkCore;
using MySQL.Data.EntityFrameworkCore.Extensions;
using Microsoft.Extensions.Configuration;

namespace EFCoreBookStore.Models{
    public class BookStoreDataContext : DbContext
    {
        public static string ConnectionString { get; set; }

        protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)
        {
            optionsBuilder.UseMySQL(ConnectionString);
        }

        protected override void OnModelCreating(ModelBuilder modelBuilder)
        {
            modelBuilder.Entity<Category>().ToTable("categories");
            modelBuilder.Entity<Category>(entity =>
            {
                entity.Property(e => e.CategoryID).HasColumnName("category_id");
                entity.Property(e => e.Name).HasColumnName("name");
            });
            modelBuilder.Entity<Category>().HasKey(e => new { e.CategoryID});

            modelBuilder.Entity<Author>().ToTable("authors");
            modelBuilder.Entity<Author>(entity =>
            {
```

```

        entity.Property(e =>
e.AuthorID).HasColumnName("author_id");
            entity.Property(e => e.Name).HasColumnName("name");
            entity.Property(e => e.Email).HasColumnName("email");
        });
modelBuilder.Entity<Author>().HasKey(e => new { e.AuthorID});

modelBuilder.Entity<Book>().ToTable("books");
modelBuilder.Entity<Book>(entity =>
{
    entity.Property(e => e.ISBN).HasColumnName("isbn");
    entity.Property(e =>
e.CategoryID).HasColumnName("category_id");
        entity.Property(e => e.Title).HasColumnName("title");
        entity.Property(e => e.Photo).HasColumnName("photo");
        entity.Property(e =>
e.PublishDate).HasColumnName("publish_date");
        entity.Property(e => e.Price).HasColumnName("price");
        entity.Property(e => e.Quantity).HasColumnName("qty");
    });
modelBuilder.Entity<Book>().HasKey(e => new { e.ISBN});

modelBuilder.Entity<BookAuthor>().ToTable("books_authors");
modelBuilder.Entity<BookAuthor>(entity =>
{
    entity.Property(e => e.ISBN).HasColumnName("isbn");
    entity.Property(e =>
e.AuthorID).HasColumnName("author_id");
});
modelBuilder.Entity<BookAuthor>().HasKey(e => new { e.ISBN, e.
AuthorID });

modelBuilder.Entity<User>().ToTable("users");
modelBuilder.Entity<User>(entity =>
{
    entity.Property(e =>
e.UserName).HasColumnName("username");
        entity.Property(e => e.RoleID).HasColumnName("roleid");
        entity.Property(e => e.Email).HasColumnName("email");
        entity.Property(e =>
e.Password).HasColumnName("password");
        entity.Property(e =>
e.Fullname).HasColumnName("fullname");
    });
modelBuilder.Entity<User>().HasKey(e => new { e.UserName });

modelBuilder.Entity<Role>().ToTable("roles");
modelBuilder.Entity<Role>(entity =>
{
    entity.Property(e => e.RoleID).HasColumnName("roleid");
    entity.Property(e =>
e.RoleName).HasColumnName("rolename");
});
modelBuilder.Entity<Role>().HasKey(e => new { e.RoleID});

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; }
public virtual DbSet<User> Users { get; set; }
public virtual DbSet<Role> Roles { get; set; }

}
}

```

## Pengelolaan Role & User

Pada sub bab ini akan dijelaskan pembuatan fitur untuk mengelola role dan user.

### Modifikasi File MasterLayout.cshtml

Langkah pertama yang dilakukan adalah menambahkan menu untuk akses kekedua fitur tersebut pada file MasterLayout.cshtml.

Berikut adalah kode yang ditambahkan pada file tersebut.

```
<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>
```

### Mengelola Role

Role berfungsi untuk mengelompokkan user. Dalam kasus ini setiap user hanya dapat memiliki sebuah role saja.

Pada sub bab ini akan diberikan contoh kode untuk mengelola role. Pada sub bab sebelumnya telah diberikan kode komponen model role yaitu Role.cs.

### RoleController.cs

Selanjutnya diberikan contoh komponen controller yaitu class RoleController.cs berikut ini.

```
RoleController.cs
using System.Linq;
using Microsoft.AspNetCore.Mvc;
using EFCoreBookStore.Models;

namespace EFCoreBookStore.Controllers
{
    public class RoleController : Controller
    {
        BookStoreDataContext db = new BookStoreDataContext();

        [HttpGet]
        public IActionResult Index()
        {
            var items = db.Roles.Select(p => p);

            return View(items);
        }

        [HttpGet]
        public IActionResult Create()
        {
            return View();
        }

        [HttpPost]
        public IActionResult Create(Role item)
        {
```

```

        if(ModelState.IsValid){
            db.Add(item);
            db.SaveChanges();

            return RedirectToAction("Index");
        }

        return View();
    }

    [HttpGet]
    public IActionResult Edit(int? id)
    {
        var item = db.Roles.SingleOrDefault(p=>p.RoleID.Equals(id));

        return View(item);
    }

    [HttpPost]
    [ValidateAntiForgeryToken]
    public IActionResult Edit([Bind("RoleID,RoleName")] Role item)
    {
        if(ModelState.IsValid){
            db.Update(item);
            db.SaveChanges();

            return RedirectToAction("Index");
        }

        return View();
    }

    [HttpGet]
    public IActionResult Delete(int id)
    {
        if(ModelState.IsValid)
        {
            var item = db.Roles.Find(id);
            db.Roles.Remove(item);
            db.SaveChanges();

            return RedirectToAction("Index");
        }

        return View();
    }
}

```

## Index.cshtml

Selanjutnya membuat komponen view Index.cshtml, Create.cshtml dan Edit.cshtml. Ketiganya akan disimpan pada folder Views/Role.

Berikut adalah kode file komponen view Index.cshtml.

Index.cshtml
<pre>@model IQueryble&lt;EFCoreBookStore.Models.Role&gt;  &lt;div class=""&gt;     &lt;div class="page-title"&gt;         &lt;div class="title_left"&gt;             &lt;h3&gt;Roles&lt;/h3&gt;         &lt;/div&gt;          &lt;div class="title_right"&gt;</pre>

```

        <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 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 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>
                                        <input type="checkbox" id="check-all" class="flat">
                                    </th>
                                    <th class="column-title" style="width: 25%">@Html.DisplayNameFor(model => model.FirstOrDefault().RoleID)</th>
                                    <th class="column-title" style="width: 65%">@Html.DisplayNameFor(model => model.FirstOrDefault().RoleName)</th>
                                    <th class="column-title no-link last" style="width: 10%"><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="a-center">
                                            <input type="checkbox" class="flat" name="table_records">
                                        </td>
                                        <td class=" ">@item.RoleID</td>
                                        <td class=" ">@item.RoleName</td>
                                        <td class=" last">
                                            <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>
    
```

```

        </table>
    </div>
</div>
</div>
</div>
</div>

```

Dan berikut adalah tampilan antarmuka dari file ini.

Role ID	Role Name	Action
1	Admin	Edit   Delete
2	User	Edit   Delete

Gambar 110. Book Store - Role - Index.cshtml.

## Create.cshtml

Berikut adalah kode lengkap dari file Create.cshtml.

```

Create.cshtml
@model EFCoreBookStore.Models.Role



206

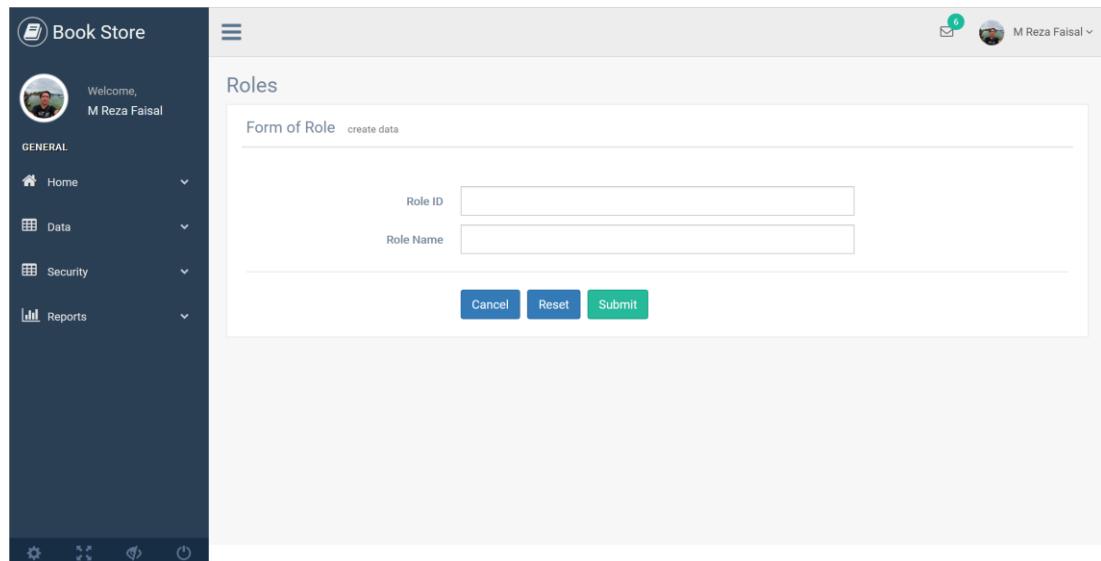

```

```

        </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" type="text" value="Role Name" />
            <span asp-validation-for="RoleName" class="text-danger">Role Name is required</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">Cancel</button>
            <button class="btn btn-primary" type="reset">Reset</button>
            <button class="btn btn-primary" type="submit" value="Submit" type="button">Submit</button>
        </div>
    </div>
    </div>
</div>

```

Berikut adalah antarmuka file Create.cshtml.



Gambar 111. Book Store - Role - Create.cshtml.

## Edit.cshtml

Berikut adalah kode lengkap file Edit.cshtml.

```

Edit.cshtml
@model EFCoreBookStore.Models.Role

<div class="">
    <div class="page-title">
        <div class="title-left">

```

```

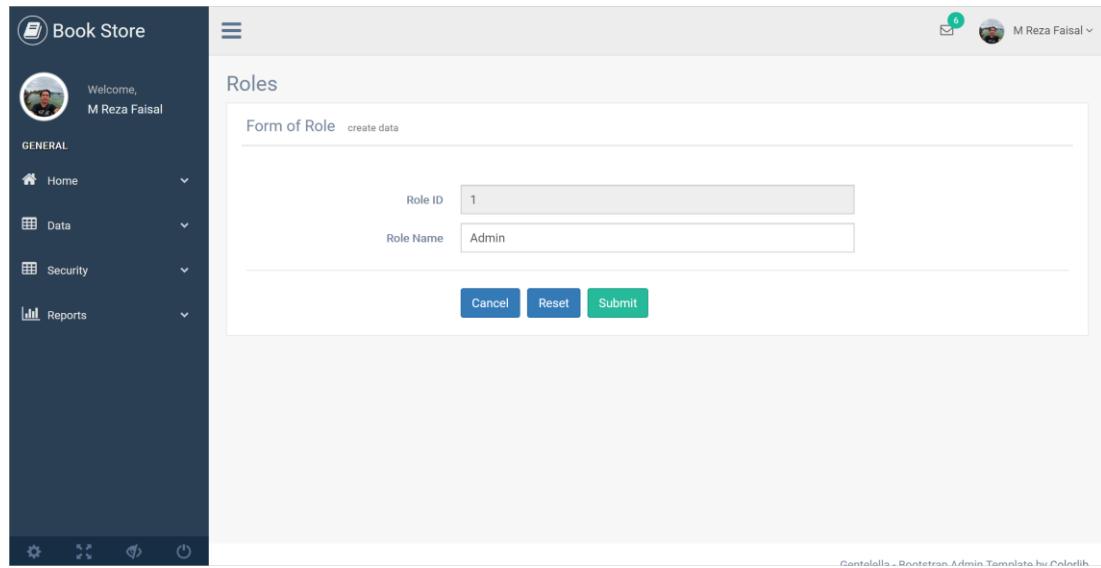
<h3>Roles</h3>
</div>

<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="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"
readonly="readonly" 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="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>
            </div>
        </div>
    </div>
</div>
</div>

```

Berikut adalah antarmuka file Edit.cshtml.



Gambar 112. Book Store - Role - Edit.cshtml.

## Mengisi Data

Setelah langkah-langkah di atas telah dilakukan maka selanjutnya dilakukan pengisian data. Data ini akan diperlukan pada sub bab selanjutnya untuk menguji proses otentikasi dan otorisasi pada aplikasi ini.

Role yang harus ditambahkan adalah:

- Admin.
- User.

List of Role			<a href="#">+ Add Data</a>
	Role ID	Role Name	Action
	1	Admin	<a href="#">Edit   Delete</a>
	2	User	<a href="#">Edit   Delete</a>

Gambar 113. Daftar role yang ditambahkan.

---

## Mengelola User

Pada sub bab ini akan diberikan contoh kode untuk mengelola user.

### Class View Model

Untuk mengelola user diperlukan tambahan 2 class view model yaitu:

- `UserViewModel.cs`, class ini digunakan untuk menampilkan daftar user pada komponen view `Index.cshtml`.
- `UserCreateFormViewModel.cs`, class ini digunakan untuk proses menambah data user yang melibatkan komponen view `Create.cshtml`.
- `UserEditFormViewModel.cs`, class ini digunakan untuk proses edit data user yang melibatkan komponen view `Edit.cshtml`.

```
UserViewModel.cs
using System;
using System.ComponentModel.DataAnnotations;

namespace EFCoreBookStore.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;}
    }
}
```

Karena pada komponen view Index.cshtml hanya digunakan untuk menampilkan data user pada tabel sehingga tidak diperlukan atribut untuk validasi.

```
UserCreateFormViewModel.cs
using System;
using System.ComponentModel.DataAnnotations;

namespace EFCoreBookStore.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 int 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 ="Fullname")]
        [Required(ErrorMessage = "{0} harus diisi.")]
    }
}
```

```

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

```

Pada class view model di atas digunakan pada pada form input Create.cshtml dan form edit Edit.cshtml. pada class di atas ditambahkan property PasswordConfirm sebagai input tambahan agar password dimasukkan dua kali. Pada property ini juga ditambahkan atribut untuk validasi untuk memeriksa kedua password yang dimasukkan adalah sama.

Berikut ini adalah kode lengkap class UserEditFormViewModel.cs.

```

UserEditFormViewModel.cs
using System;
using System.ComponentModel.DataAnnotations;

namespace EFCoreBookStore.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 int 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")]
        [StringLength(256, ErrorMessage = "{0} tidak boleh lebih {1}
karakter.")]
        [DataType(DataType.Password)]
        public String Password {set; get; }

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

```

Kode class ini mirip dengan class UserCreateFormViewModel.cs, perbedaannya pada property Password dan PasswordConfirm yang tidak memiliki atribut [Required] untuk melakukan validasi untuk mengisi nilai kedua property ini. Dengan tidak adanya atribut ini maka dimungkinkan untuk mengosongkan nilai untuk kedua property ini. Sehingga jika pada proses edit kolom password dan password confirm tidak diisi artinya password untuk user yang diedit tersebut tidak diubah. Password untuk user yang diedit diubah ketika kedua kolom diisi.

## UserController.cs

```
UserController.cs
using System;
using System.Linq;
using System.Collections.Generic;
using System.IO;
using System.Net.Http.Headers;
using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.Mvc.Rendering;
using Microsoft.AspNetCore.Hosting;
using System.Security.Cryptography;
using System.Text;
using EFCoreBookStore.Models;

namespace EFCoreBookStore.Controllers
{
    public class UserController : Controller
    {
        BookStoreDataContext db = new BookStoreDataContext();
        string key = "E546C8DF278CD5931069B522E695D4F2";

        public static string EncryptString(string text, string keyString)
        {
            var key = Encoding.UTF8.GetBytes(keyString);

            using (var aesAlg = Aes.Create())
            {
                using (var encryptor = aesAlg.CreateEncryptor(key,
aesAlg.IV))
                {
                    using (var msEncrypt = new MemoryStream())
                    {
                        using (var csEncrypt = new CryptoStream(msEncrypt,
encryptor, CryptoStreamMode.Write))
                        using (var swEncrypt = new StreamWriter(csEncrypt))
                        {
                            swEncrypt.Write(text);
                        }

                        var iv = aesAlg.IV;

                        var decryptedContent = msEncrypt.ToArray();

                        var result = new byte[iv.Length +
decryptedContent.Length];

                        Buffer.BlockCopy(iv, 0, result, 0, iv.Length);
                        Buffer.BlockCopy(decryptedContent, 0, result,
iv.Length, decryptedContent.Length);

                        return Convert.ToString(result);
                    }
                }
            }
        }

        [HttpGet]
        public IActionResult Index()
        {
            var userList = db.Users.ToList();
            IList<UserViewModel> items = new List<UserViewModel>();
            foreach(User user in userList){
                UserViewModel item = new UserViewModel();
                item.UserName = user.UserName;
```

```

        item.Email = user.Email;
        item.Fullname = user.Fullname;

        Role      role      =      db.Roles.SingleOrDefault(p      =>
p.RoleID.Equals(user.RoleID));
        item.RoleName = role.RoleName;

        items.Add(item);
    }

    return View(items);
}

[HttpGet]
public IActionResult Create()
{
    ViewBag.Roles = new SelectList(db.Roles.ToList(), "RoleID",
"RoleName");

    return View();
}

[HttpPost]
public IActionResult Create(UserCreateFormViewModel item)
{
    ViewBag.Roles = new SelectList(db.Roles.ToList(), "RoleID",
"RoleName");

    if(ModelState.IsValid){
        User user = new User();
        user.UserName = item.UserName;
        user.RoleID = item.RoleID;
        user.Email = item.Email;
        user.Password = EncryptString(item.Password, key);
        user.Fullname = item.Fullname;

        db.Add(user);
        db.SaveChanges();

        return RedirectToAction("Index");
    }

    return View();
}

[HttpGet]
public IActionResult Edit(string id)
{
    ViewBag.Roles = new SelectList(db.Roles.ToList(), "RoleID",
"RoleName");

    var user = db.Users.SingleOrDefault(p=>p.UserName.Equals(id));
    UserEditFormViewModel item = new UserEditFormViewModel();
    item.UserName = user.UserName;
    item.RoleID = user.RoleID;
    item.Email = user.Email;
    item.Fullname = user.Fullname;

    return View(item);
}

[HttpPost]
[ValidateAntiForgeryToken]
public IActionResult Edit([Bind("UserName, RoleID, Email, Password,
PasswordConfirm, Fullname")] UserEditFormViewModel item)
{
}

```

```

        ViewBag.Roles = new SelectList(db.Roles.ToList(), "RoleID",
"RoleName");

        if(ModelState.IsValid){
            User user = db.Users.SingleOrDefault(p =>
p.UserName.Equals(item.UserName));
            user.UserName = item.UserName;
            user.RoleID = item.RoleID;
            user.Email = item.Email;
            if(!String.IsNullOrEmpty(item.Password)){
                user.Password = EncryptString(item.Password, key);
            }
            user.Fullname = item.Fullname;

            db.Update(user);
            db.SaveChanges();

            return RedirectToAction("Index");
        }

        return View();
    }

    [HttpGet]
    public IActionResult Delete(string id)
    {
        if(ModelState.IsValid)
        {
            var item = db.Users.Find(id);
            db.Users.Remove(item);
            db.SaveChanges();

            return RedirectToAction("Index");
        }

        return View();
    }
}

```

Pada class ini dapat dilihat selain method action juga ditambahkan method EncryptString yang berfungsi untuk melakukan proses encrypt password. Penggunaan method ini dapat dilihat pada method action [HttpPost] Create dan [HttpPost] Edit.

Jika diperhatikan isi tabel users maka dapat dilihat nilai password untuk setiap user disimpan sebagai string yang diacak seperti yang terlihat pada gambar di bawah ini.



```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL database
select * from users

+-----+-----+-----+-----+
| username | roleid | email      | password          | fullname   |
+-----+-----+-----+-----+
| faisal   | 2       | faisal@yahoo.com | c8DdbpggIDI1qcIjLUEsNtv5ywwdvM5bJEdjWGlUkg= | faisal reza |
| reyza    | 1       | reza@faisal.id  | 9PsP5csQRKgz/yJA8XEzuGj5SwI8CE1fTSroIMWcVGs= | M Reza Faisal |
+-----+-----+-----+-----+

```

**Gambar 114. Tabel users - password user yang diacak dengan method EncryptString().**

## Index.cshtml

Berikut adalah kode file Index.cshtml.

```
Index.cshtml
```

```

@model IList<EFCoreBookStore.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" class="fa fa-plus"></a> 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>
                                    <input type="checkbox" id="check-all" class="flat">
                                </th>
                                <th class="column-title" style="width: 25%">@Html.DisplayNameFor(model => model.FirstOrDefault().UserName)</th>
                                <th class="column-title" style="width: 25%">@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: 10%"><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="a-center ">
                                        <input type="checkbox" class="flat" name="table_records">
                                    </td>
                                    <td>
                                        <span class="label label-success">@(item.Status == 1 ? "Active" : "InActive")</span>
                                    </td>
                                    <td>
                                        <span class="label label-primary">@(item.RoleName)</span>
                                    </td>
                                    <td>
                                        <span class="label label-primary">@(item.UserName)</span>
                                    </td>
                                    <td>
                                        <span class="label label-primary">@(item.Email)</span>
                                    </td>
                                    <td>
                                        <span class="label label-primary">@(item.Fullname)</span>
                                    </td>
                                    <td class="a-center ">
                                        <button class="btn btn-primary" type="button" asp-action="Edit" asp-route-id="@item.Id">Edit</button>
                                        <button class="btn btn-danger" type="button" asp-action="Delete" asp-route-id="@item.Id">Delete</button>
                                    </td>
                                </tr>
                            }
                        </tbody>
                    </table>
                </div>
            </div>
        </div>
    </div>


```

```

        </td>
        <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="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>
</div>
</div>

```

Gambar berikut ini adalah antarmuka dari Index.cshtml.

Username	Role	Email	Fullname	Action
faisal	User	faisal@yahoo.com	faisal reza	Edit   Delete
reyza	Admin	reza@faisal.id	M Reza Faisal	Edit   Delete

**Gambar 115. Book Store - User - Index.cshtml.**

## Create.cshtml

Berikut ini adalah isi file Create.cshtml.

```

Create.cshtml
@model EFCoreBookStore.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-6 col-sm-6 col-xs-12">
                    <select asp-for="RoleID" asp-
items="@ViewBag.Roles" class="form-control">
                        <option>Choose Role</option>
                    </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">
                </div>
            </div>
        </form>
    </div>

```

```

        <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">Cancel</button>
            <button class="btn btn-primary" type="reset">Reset</button>
            <button type="submit" class="btn btn-success">Submit</button>
        </div>
    </div>
</div>
</div>
</div>
</div>
</div>
</div>

```

The screenshot shows the Book Store application's user creation interface. On the left is a dark sidebar with a user profile picture and the text "Welcome, M Reza Faisal". Below this are sections for "GENERAL", "Home", "Data", "Security", and "Reports". The main content area has a title "Users" and a sub-title "Form of User". It includes fields for "Username", "Role" (with a dropdown menu), "Password", "Password Confirm", "Email", and "Fullscreen". At the bottom are "Cancel", "Reset", and "Submit" buttons. The URL "localhost:5000/User/index.html" is visible at the bottom of the browser window.

**Gambar 116. Book Store - User - Create.cshtml.**

## Edit.cshtml

Berikut adalah isi file Edit.cshtml.

```

Edit.cshtml
@model EFCoreBookStore.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>create 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-6 col-sm-6 col-xs-12">
                <select asp-for="RoleID" asp-
items="@ViewBag.Roles" class="form-control">
                    <option>Choose Role</option>
                </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" 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"
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">Cancel</button>
```

```
        <button class="btn btn-primary"
```

```
type="reset">Reset</button>
```

```
        <button type="submit" class="btn btn-
```

```
success">Submit</button>
```

```
    </div>
```

```
    </div>
```

```
    </form>
```

```
    </div>
```

```
    </div>
```

```
</div>
```

```
</div>
```

Book Store

Welcome,  
M Reza Faisal

GENERAL

- Home
- Data
- Security
- Reports

Users

Form of User create data

Username	reyza
Role	Admin
Password	
Password Confirm	
Email	reza@faisal.id
Fullname	M Reza Faisal

[Cancel](#) [Reset](#) [Submit](#)

localhost:5000/User/Edit/index.html

Gambar 117. Book Store - User - Edit.cshtml.

## Mengisi Data

Setelah langkah-langkah di atas telah dilakukan maka selanjutnya dilakukan pengisian data. Data ini akan diperlukan pada sub bab selanjutnya untuk menguji proses otentifikasi dan otorisasi pada aplikasi ini.

Perlu ditambahkan masing-masing sebuah user yang mewakili setiap role yang telah dibuat pada sub bab Mengelola Role. Berikut adalah contoh daftar user yang telah dibuat.

List of User					<a href="#">+ Add Data</a>
	Username	Role	Email	Fullscreen	Action
<input type="checkbox"/>	faisal	User	faisal@yahoo.com	faisal reza	<a href="#">Edit   Delete</a>
<input type="checkbox"/>	reyza	Admin	reza@faisal.id	M Reza Faisal	<a href="#">Edit   Delete</a>

Gambar 118. Daftar user.

Nama user yang ditambahkan adalah bebas, yang perlu dipastikan adalah role user. Perlu ditambahkan 1 user dengan role Admin dan 1 user dengan role User seperti contoh pada gambar di atas.

## Implementasi Otentikasi

### Login

Untuk form login ada beberapa komponen yang harus dibuat yaitu:

- Komponen model yaitu class model view UserLoginViewModel.cs.
- Komponen view yaitu file Login.cshtml.
- Komponen controller yaitu class HomeController.cs. File class ini telah dibuat di awal project ini dibuat. Yang perlu dilakukan adalah menambahkan method action untuk login dan logout.

### Login.cshtml

File Login.cshtml merupakan file view dari method action Login yang berada pada class controller HomeController.cs, sehingga file ini disimpan pada folder Views/Home. Berikut adalah kode lengkap dari file ini.

```
Login.cshtml
@{
    Layout = null;
}

@model EFCoreBookStore.Models.UserLoginViewModel

<!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>

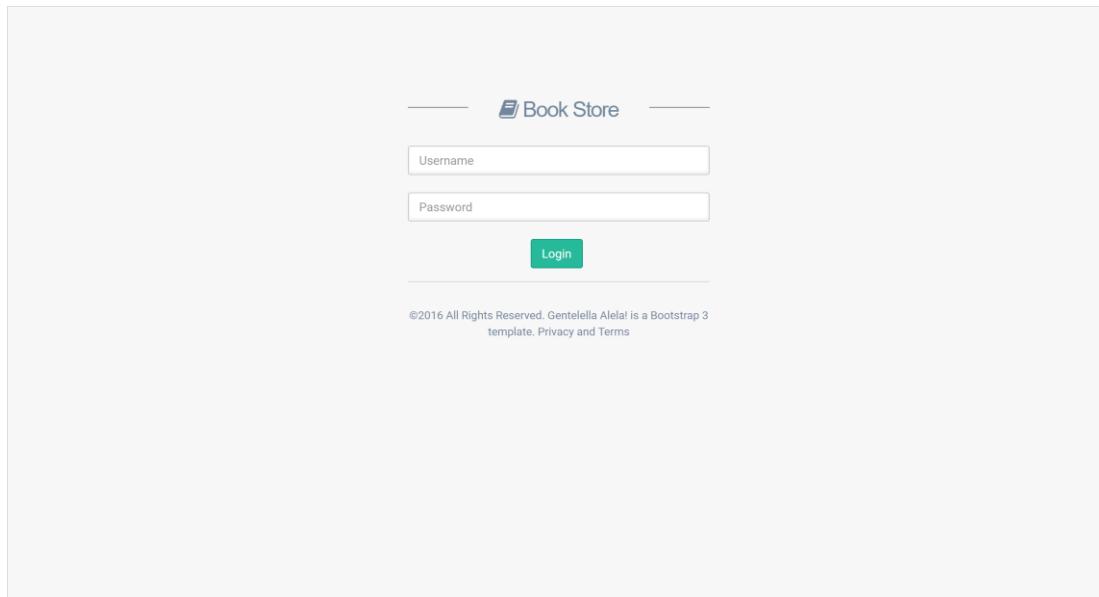
<div class="clearfix"></div>

<div class="separator">
<div class="clearfix"></div>
<br />

<div>
<p>©2016 All Rights Reserved. Gentelella Alela! is a Bootstrap 3 template. Privacy and Terms</p>
</div>
</div>
</form>
</section>
</div>
</div>
</body>
</html>

```

Gambar berikut ini adalah antarmuka dari file Login.cshtml.



Gambar 119. Book Store - Login.cshtml.

Dari gambar di atas dapat dilihat terdapat perbedaan antara antarmuka sebelumnya. Hal ini dikarenakan file ini tidak menggunakan layout MasterLayout.cshtml.

### Modifikasi HomeController.cs

Selanjutnya adalah menambahkan method action berikut ini:

- [HttpGet] Login yang berfungsi untuk menampil form login yang ada pada file komponen view Login.cshtml.
- [HttpPost] Login yang berfungsi untuk melakukan proses otentikasi user yang melakukan proses login.
- [HttpGet] Logout yang berfungsi untuk melakukan proses logout.
- [HttpGet] AccessDenied yang berfungsi untuk menampilkan halaman AccessDenied.cshtml ketika ada user yang melanggar aturan otentikasi.

Selain itu juga ditambahkan method untuk melakukan proses decrypt password agar string password yang terenkripsi bisa dikembalikan menjadi nilai password asli.

Berikut ini adalah kode lengkap dari file HomeController.cs.

```
HomeController.cs
using System;
using System.Linq;
using System.Collections.Generic;
using System.IO;
using System.Net.Http.Headers;
using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.Mvc.Rendering;
using Microsoft.AspNetCore.Hosting;
using System.Security.Cryptography;
using System.Text;
using System.Security.Claims;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Authorization;

using EFCoreBookStore.Models;

namespace EFCoreBookStore.Controllers
{
    public class HomeController : Controller
```

```

    {
        BookStoreDataContext db = new BookStoreDataContext();
        string key = "E546C8DF278CD5931069B522E695D4F2";
        public static string DecryptString(string cipherText, string
keyString)
        {
            var fullCipher = Convert.FromBase64String(cipherText);

            var iv = new byte[16];
            var cipher = new byte[16];

            Buffer.BlockCopy(fullCipher, 0, iv, 0, iv.Length);
            Buffer.BlockCopy(fullCipher, iv.Length, cipher, 0, iv.Length);
            var key = Encoding.UTF8.GetBytes(keyString);

            using (var aesAlg = Aes.Create())
            {
                using (var decryptor = aesAlg.CreateDecryptor(key, iv))
                {
                    string result;
                    using (var msDecrypt = new MemoryStream(cipher))
                    {
                        using (var csDecrypt = new CryptoStream(msDecrypt,
decryptor, CryptoStreamMode.Read))
                        {
                            using (var srDecrypt = new
StreamReader(csDecrypt))
                            {
                                result = srDecrypt.ReadToEnd();
                            }
                        }
                    }
                }
            }

            return result;
        }
    }

    [HttpGet]
    [Authorize]
    public IActionResult Index()
    {
        return View();
    }

    [HttpGet]
    public IActionResult Login(string returnUrl = null)
    {
        ViewData["ReturnUrl"] = returnUrl;

        return View();
    }

    [HttpPost]
    public async Task<IActionResult> Login(UserLoginFormViewModel
item, string returnUrl = null)
    {
        ViewData["ReturnUrl"] = returnUrl;

        if(ModelState.IsValid){
            var user = db.Users.SingleOrDefault(p =>
p.UserName.Equals(item.UserName));
            if(user != null){
                if(!String.IsNullOrEmpty(user.Password)){
                    string decryptPassword =
DecryptString(user.Password, key);

```

```

        if(item.Password.Equals(decryptPassword)) {
            var role = db.Roles.SingleOrDefault(p =>
p.RoleID.Equals(user.RoleID));

            var claims = new List<Claim>
            {
                new Claim("username", user.UserName),
                new Claim("fullname", user.Fullname),
                new Claim(ClaimTypes.Role, role.RoleName)
            };

            var id = new ClaimsIdentity(claims,
"password");
            var principal = new ClaimsPrincipal(id);

            await
HttpContext.Authentication.SignInAsync("Cookies", principal);

            return RedirectToAction("Index");
        }
    }

    return View();
}

[HttpGet]
public async Task<IActionResult> Logout()
{
    await HttpContext.Authentication.SignOutAsync("Cookies");
    return RedirectToAction("Login");
}

[HttpGet]
[Authorize]
public IActionResult AccessDenied()
{
    return View();
}
}

```

Karena ada penambahan method action AccessDenied yang memerlukan halaman view maka pada folder Views/Home ditambahkan file AccessDenied.cshtml dengan isi sebagai berikut.

```

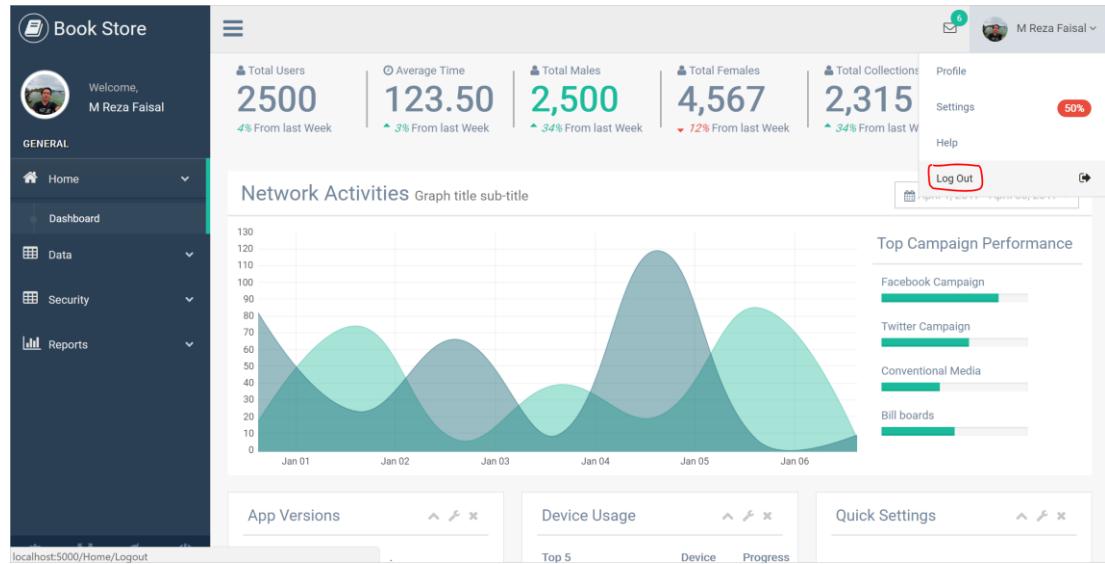
AccessDenied.cshtml
<div class="">
    <div class="page-title">
        <div class="title_left">
            <h3>Access Denied</h3>
        </div>
    </div>
    <div class="clearfix"></div>

    <div class="row">
        <h2 style="color: red">
            Anda tidak memiliki hal akses ke halaman tersebut.
        </h2>
    </div>
</div>

```

## Logout

Pada gambar di bawah dapat dilihat link Log Out. Pada sub bab ini akan dilakukan modifikasi file MasterLayout.cshtml agar link Log Out tersebut mengakses method action Logout pada class controller HomeController.cs.



Gambar 120. Book Store - Logout.

Baris kode yang harus dimodifikasi adalah baris ke-135. Berikut adalah kode setelah diubah.

```
<li>
    <a asp-controller="Home" asp-action="Logout">
        <i class="fa fa-sign-out pull-right"></i> Log Out
    </a>
</li>
```

## Implementasi Otorisasi

Pada sub bab Library Otentikasi & Otorisasi > Implementasi > Contoh Kasus telah diberikan contoh penggunaan atribut [Authorize] pada method action. Atribut dapat digunakan untuk menentukan method action mana yang dapat diakses oleh user tanpa melakukan otentikasi dan method mana yang dapat diakses oleh user setelah melewati proses otentikasi. Atribut ini juga dapat digunakan menentukan user dengan role apa yang bisa mengakses suatu method action.

## Otorisasi Method Action

Pada bab ini akan dilakukan pemberian atribut pada method-method yang ada pada seluruh class controller pada aplikasi web Book Store ini. Pada aplikasi ini terdapat 6 class controller utama yaitu:

- HomeController.cs.
- CategoryController.cs.
- AuthorController.cs.
- BookController.cs.
- RoleController.cs.
- UserController.cs.

Pengguna aplikasi ini akan dikelompokkan ke dalam 2 role yaitu:

- Admin.
- User.

Dari keenam class controller tersebut terdapat dua class controller yang hanya dapat diakses oleh role Admin yaitu:

- RoleController.cs.
- UserController.cs.

Untuk membuat hal itu terjadi maka langkah pertama yang harus dilakukan adalah menambahkan baris ini pada setiap class controller.

```
using Microsoft.AspNetCore.Authorization;
```

Selanjutnya menambahkan atribut [Authorize] pada 4 class controller yang dapat diakses oleh seluruh user yang sudah melewati proses otentikasi. Untuk kode class HomeController.cs dapat dilihat pada bab ini sub bab Login > Modifikasi HomeController.cs. Pada class controller ini hanya method action Index dan AccessDenied yang menggunakan atribut [Authorize], sedangkan method action lainnya tidak menggunakan atribut itu karena method action tersebut harus dapat diakses oleh user tanpa melakukan proses otentikasi.

Berikut adalah kode pada class controller CategoryController.cs.

```
CategoryController.cs
. . .
using Microsoft.AspNetCore.Authorization;

namespace EFCoreBookStore.Controllers
{
    public class CategoryController : Controller
    {
        . . .

        [HttpGet]
        [Authorize]
        public IActionResult Index()
        {
            . . .
        }

        [HttpGet]
        [Authorize]
        public IActionResult Create()
        {
            . . .
        }

        [HttpPost]
        [Authorize]
        public IActionResult Create(Category item)
        {
            . . .
        }

        [HttpGet]
        [Authorize]
        public IActionResult Edit(int? id)
        {
            . . .
        }

        [HttpPost]
        [Authorize]
```

```

[ValidateAntiForgeryToken]
public IActionResult Edit([Bind("CategoryID,Name")] Category item)
{
    . . .
}

[HttpGet]
[Authorize]
public IActionResult Delete(int id)
{
    . . .
}
}

```

Berikut adalah kode class controller AuthorCategory.cs setelah ditambahkan atribut [Authorize].

```

AuthorCategory.cs
. . .
using Microsoft.AspNetCore.Authorization;

namespace EFCoreBookStore.Controllers
{
    public class AuthorController : Controller
    {
        . . .

        [HttpGet]
        [Authorize]
        public IActionResult Index()
        {
        }

        [HttpGet]
        [Authorize]
        public IActionResult Create()
        {
        }

        [HttpPost]
        [Authorize]
        public IActionResult Create(Author item)
        {
        }

        [HttpGet]
        [Authorize]
        public IActionResult Edit(int? id)
        {
        }

        [HttpPost]
        [Authorize]
        [ValidateAntiForgeryToken]
        public IActionResult Edit([Bind("AuthorID,Name, Email")] Author
item)
        {
        }

        [HttpGet]
        [Authorize]
        public IActionResult Delete(int id)
        {
        }
    }
}

```

Berikut ini adalah kode class controller BookCategory.cs setelah ditambahkan atribut [Authorize].

```
BookController.cs
. . .
using Microsoft.AspNetCore.Authorization;

namespace EFCoreBookStore.Controllers
{
    public class BookController : Controller
    {
        . . .

        [HttpGet]
        [Authorize]
        public IActionResult Index()
        {
            . . .
        }

        [HttpPost]
        [Authorize]
        public IActionResult Create(BookFormViewModel item)
        {
            . . .
        }

        [HttpGet]
        [Authorize]
        public IActionResult Edit(int? id)
        {
            . . .
        }

        [HttpPost]
        [Authorize]
        [ValidateAntiForgeryToken]
        public IActionResult Edit([Bind("ISBN, CategoryID, Title, Photo,
        PublishDate, Price, Quantity, AuthorIDs")] BookFormViewModel item)
        {
            . . .
        }

        [HttpGet]
        [Authorize]
        public IActionResult Delete(int id)
        {
            . . .
        }
    }
}
```

Sedangkan untuk class controller RoleController.cs dan UserController.cs ditambahkan atribut berikut ini.

```
[Authorize(Roles = "Admin")]
```

Artinya seluruh method action pada kedua class controller itu hanya dapat diakses oleh user dengan role Admin. Berikut adalah kode class RoleController.cs yang telah dimodifikasi.

```
RoleController.cs
. . .
using Microsoft.AspNetCore.Authorization;

namespace EFCoreBookStore.Controllers
{
```

```

public class RoleController : Controller
{
    . . .

    [HttpGet]
    [Authorize(Roles = "Admin")]
    public IActionResult Index()
    {
        . . .
    }

    [HttpGet]
    public IActionResult Create()
    {
        . . .
    }

    [HttpPost]
    [Authorize(Roles = "Admin")]
    public IActionResult Create(Role item)
    {
        . . .
    }

    [HttpGet]
    public IActionResult Edit(int? id)
    {
        . . .
    }

    [HttpPost]
    [Authorize(Roles = "Admin")]
    [ValidateAntiForgeryToken]
    public IActionResult Edit([Bind("RoleID,RoleName")] Role item)
    {
        . . .
    }

    [HttpGet]
    [Authorize(Roles = "Admin")]
    public IActionResult Delete(int id)
    {
        . . .
    }
}

```

Dan berikut adalah kode class controller UserController.cs yang telah dimodifikasi.

```

UserController.cs
. .
using Microsoft.AspNetCore.Authorization;

namespace EFCoreBookStore.Controllers
{
    public class UserController : Controller
    {
        . . .

        [HttpGet]
        [Authorize(Roles = "Admin")]
        public IActionResult Index()
        {
            . . .
        }

        [HttpGet]
        [Authorize(Roles = "Admin")]

```

```

public IActionResult Create()
{
    . . .
}

[HttpPost]
[Authorize(Roles = "Admin")]
public IActionResult Create(UserCreateViewModel item)
{
    . . .
}

[HttpGet]
[Authorize(Roles = "Admin")]
public IActionResult Edit(string id)
{
    . . .
}

[HttpPost]
[Authorize(Roles = "Admin")]
[ValidateAntiForgeryToken]
public IActionResult Edit([Bind("UserName, RoleID, Email, Password,
PasswordConfirm, Fullname")] UserEditFormViewModel item)
{
    . . .
}

[HttpGet]
[Authorize(Roles = "Admin")]
public IActionResult Delete(string id)
{
    . . .
}
}

```

## Demo

Setelah semua persiapan dan modifikasi di atas telah dilakukan, maka pada sub bab ini akan diperlihatkan demo bagaimana proses otentikasi dan otorisasi bekerja.

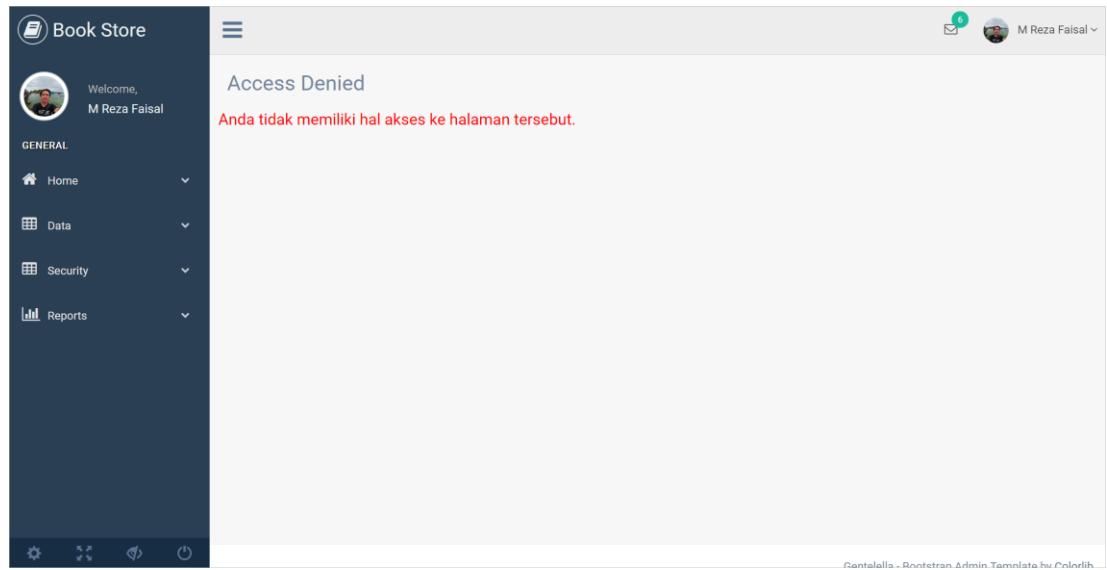
Langkah pertama adalah mencoba mengakses daftar kategori buku tanpa melakukan login.

```
http://localhost:5000/Category
```

Maka secara otomatis akan ditampilkan form login. Hal ini membuktikan proses otorisasi untuk melindungi method action pada class controller CategoryController.cs.

Kemudian pada form login masukkan user faisal atau user dengan role User dan akan ditampilkan halaman Index.cshtml yang merupakan view dari method action Index pada class controller HomeController.cs. Hal ini membuktikan proses otorisasi telah berhasil dilakukan.

Berikutnya akan dicoba pembuktian proses otorisasi yang lain, dengan cara mengakses menu Role dan User. Hasilnya akan dapat dilihat tampilan berikut ini.



Gambar 121. Book Store - Halaman AccessDenied.cshtml.

Hal ini membuktikan jika method action pada kedua class controller tersebut tidak bisa diakses oleh user dengan role User.

Langkah selanjutnya adalah user faisal melakukan proses logout dengan mengklik tombol Log Out yang telah disediakan. Maka secara otomatis akan ditampilkan kembali halaman form login.

Selanjutnya lakukan login dengan user reyza atau user dengan role Admin, kemudian akses seluruh menu yang ada termasuk menu Role dan User. Dan dapat dilihat bahwa halaman-halaman yang pada kedua menu tersebut dapat diakses. Hal ini juga membuktikan proses otorisasi berjalan dengan baik.

# **7**

## ***Penutup***

Sebagai penutup, buku ini ditulis untuk para web developer yang ingin membangun aplikasi web lintas platform dengan ASP.NET Core dengan database MySQL dengan tool development Visual Studio Code.

Akhir kalimat, jika ada kritik dan saran dapat ditujukan langsung via email ke alamat reza.faisal@gmail.com.