Mariusz Wieczorek

mariusz.wieczorek@kabat.pl

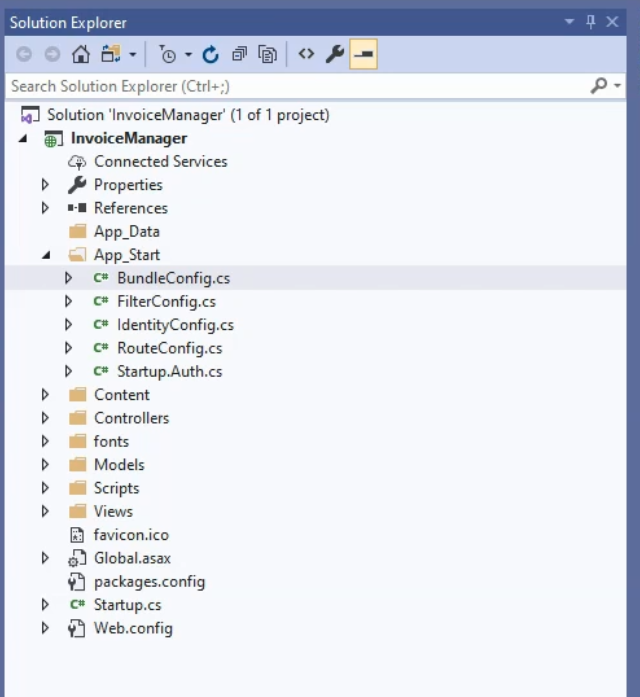
Streszczenie

Omówienie plików powstałych w projekcie po wygenerowaniu szablonu aplikacji

ASP.NET MVC

Tworzenie aplikacji cd.

**Omówienie plików**



**BundleConfig.cs** (Bundle – pakiet, paczka) klasa w której wiążemy kilka skryptów css lub js w jeden plik, którego kod ładuje się szybciej, łatwiej możemy się do niego odwołać w kodzie.

|  |
| --- |
| public class BundleConfig  {  // For more information on bundling, visit https://go.microsoft.com/fwlink/?LinkId=301862  public static void RegisterBundles(BundleCollection bundles)  {  bundles.Add(new ScriptBundle("~/bundles/jquery").Include(  "~/Scripts/jquery-{version}.js"));  bundles.Add(new ScriptBundle("~/bundles/jqueryval").Include(  "~/Scripts/jquery.validate\*"));  // Use the development version of Modernizr to develop with and learn from. Then, when you're  // ready for production, use the build tool at https://modernizr.com to pick only the tests you need.  bundles.Add(new ScriptBundle("~/bundles/modernizr").Include(  "~/Scripts/modernizr-\*"));  bundles.Add(new ScriptBundle("~/bundles/bootstrap").Include(  "~/Scripts/bootstrap.js"));  bundles.Add(new StyleBundle("~/Content/css").Include(  "~/Content/bootstrap.css",  "~/Content/site.css"));  }  } |

Dzięki zapisowi version w nawiasach klamrowych, jak wersja się zmieni to nadal zapis ten będzie działał.  
Dalej zostaną załączone wszystkie skrypty, które rozpoczynają się od jquery.validate.

**FilterConfig.cs** - tu można własny filtr, który będzie wywoływany przed każdym requestem.

**RouteConfig.cs** - można tu zdefiniować jak ma działać routing w naszej aplikacji, jak mają wyglądać adresy URL i jakie akcje mają być wywoływane.

**Content** – folder z zasobami, zawiera pliki CSS np. **Site.css** gdzie będziemy definiować własne style CSS, bootstrap’owe css’y , obrazki i inne zasoby.

**Controllers** - folder z plikami zawierającymi kontrolery. Mamy zdefiniowane na wstępie 3 kontrolery. **AccountControler.cs** i **ManageController.cs** mają już w sobie mnóstwo logiki odpowiedzialnej za logowanie i zarządzanie kontami. **HomeController.cs** jest prawie pusty. Kontrolerem w ASP.NET MVC jest klasa C#, która dziedziczy po klasie Controller. Metody kontrolera nazywamy **Akcjami**.

**Models** – zawiera modele związane z logiką związaną z użytkownikami

**Scripts** – zawiera skrypty w JavaScript

**Views** – zawiera widoki. Posiada on kolejne foldery, które wg. konwencji nazewniczej mają i muszą mieć nazwę powiązaną z kontrolerami pewną konwencją:  
dla **Home**Controller.cs jest tu katalog Views\**Home**\  
dla **Manage**Controller.cs jest katalog Views\**Manage**\

W katalogu jest tyle plików widoku \*.cshtml ile akcji w kontrolerze, jest to domyślna konwencja, niewymuszona.

W katalogu Views znajduje się jeszcze katalog Shared w którym mamy widoki wspólne dla wszystkich kontrolerów.

\_ViewStart.cshtml - widok startowy, zdefiniowany jest w nim layout   
Web.config – różne konfiguracje dla widoków

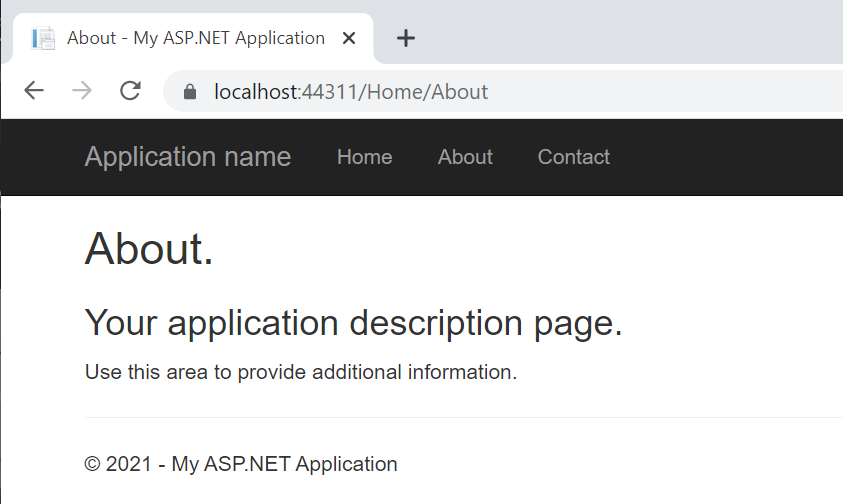
Global.asax – wywołuje przy starcie metody konfiguracyjne

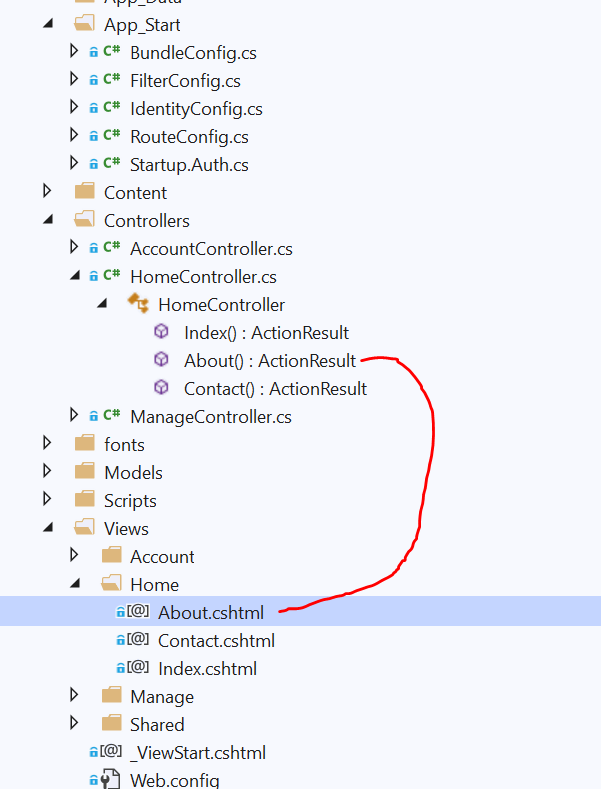
**Packages.config** – informacje o zainstalowanych nugetach

**Web.config** – ogólny plik konfiguracyjny analogiczny do App.config w aplikacjach desktopowych.  
connectionstring,

Po uruchomieniu aplikacji nastąpi połączenie z bazą i utworzenie tabel.

|  |
| --- |
| public class RouteConfig  {  public static void RegisterRoutes(RouteCollection routes)  {  routes.IgnoreRoute("{resource}.axd/{\*pathInfo}");  routes.MapRoute(  name: "Default",  url: "{controller}/{action}/{id}",  defaults: new { controller = "Home", action = "Index", id = UrlParameter.Optional }  );  }  } |





Po kliknięci na about w kontrolerze home zostaje wywołana akcja about().

|  |
| --- |
| public ActionResult About()  {  ViewBag.Message = "Your application description page.";  return View();  } |

|  |
| --- |
| @{  ViewBag.Title = "About";  }  <h2>@ViewBag.Title.</h2>  <h3>@ViewBag.Message</h3>  <p>Use this area to provide additional information.</p> |

Widok ten jest częścią layoutu \_Layout.cshtml  
Jest on wstawiony w szablon w miejsce @RenderBody()

|  |
| --- |
| @Html.Partial("\_LoginPartial")  </div>  </div>  </div>  <div class="container body-content">  @RenderBody()  <hr />  <footer>  <p>&copy; @DateTime.Now.Year - My ASP.NET Application</p>  </footer>  </div>  @Scripts.Render("~/bundles/jquery")  @Scripts.Render("~/bundles/bootstrap")  @RenderSection("scripts", required: false)  </body>  </html> |

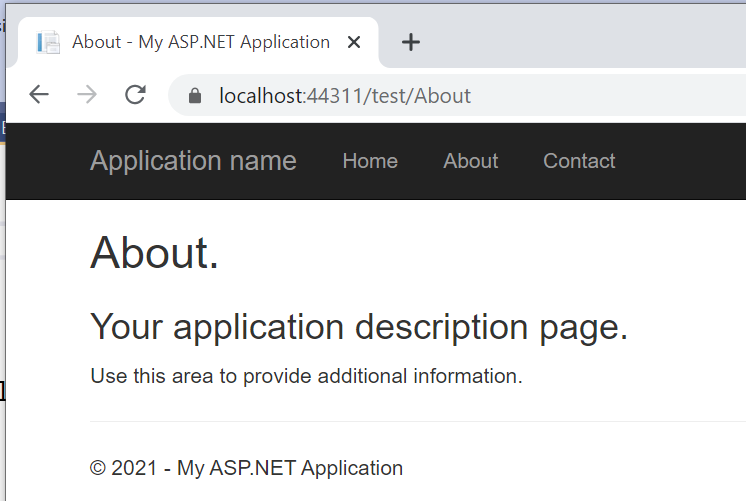
Gdy w widoku wpiszemy Layout = null; Szablon strony się nie wyświetli.

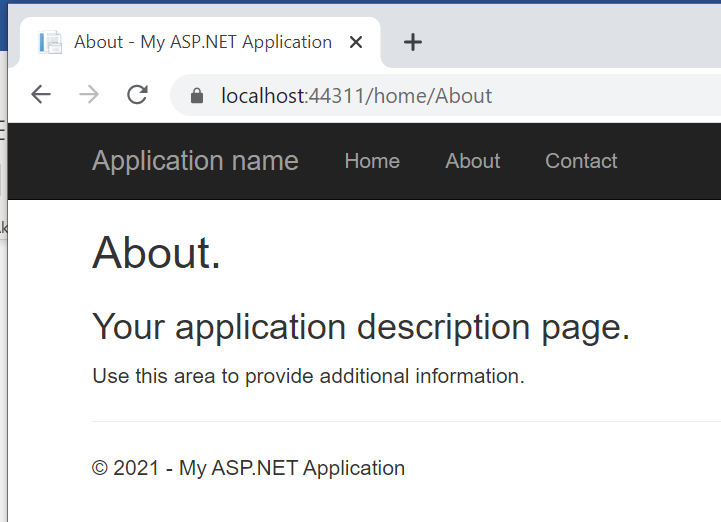
|  |
| --- |
| @{  ViewBag.Title = "About";  Layout = null;  }  <h2>@ViewBag.Title.</h2>  <h3>@ViewBag.Message</h3>  <p>Use this area to provide additional information.</p> |

Pisząc link w pasku adresu, zostaje on sprawdzony przez routing, jeżeli routing jest dobry, to wywoływana jest odpowiednia akcja z kontrolera, następnie kontroler wyświetla widok zgodnie z konwencją, czyli w katalogu z nazwą kontrolera i widok z nazwą akcji. Jeżeli go tam nie będzie przeszukany zostanie katalog shared, jeżeli widoku również tam nie będzie to wyświetlony zostanie błąd.

|  |
| --- |
| routes.MapRoute(  name: "My",  url: "test/{action}/{id}",  defaults: new { controller = "Home", action = "Index", id = UrlParameter.Optional }  );  routes.MapRoute(  name: "Default",  url: "{controller}/{action}/{id}",  defaults: new { controller = "Home", action = "Index", id = UrlParameter.Optional }  ); |

Ścieżki routingu możemy różnie konfigurować, zapisane jak powyżej, to wpisując home/about czy też test/about trafimy w to samo miejsce





Jeżeli w kontrolerze nie podamy nazwy widoku, to zostanie wywołany widok o nazwie akcji,

|  |
| --- |
| public ActionResult About()  {  ViewBag.Message = "Your application description page.";  return View();  } |

Ale możemy jako parametr przekazać nazwę widoku np.:

|  |
| --- |
| public ActionResult About()  {  ViewBag.Message = "Your application description page.";  return View("Contact");  } |

Teraz wciskając przycisk about wywołamy widok Contact

Przekazywanie danych do widoku,   
Można jako parametr, ale metoda View() oczekuje widoku, więc taki tekst musimy rzutować  
na object

|  |
| --- |
| public ActionResult About()  {  ViewBag.Message = "Your application description page.";  var tekst = "121212 1212121 1212121";  return View((object)tekst);  } |

|  |
| --- |
| @model string  @{  ViewBag.Title = "About";  }  <h2>@ViewBag.Title.</h2>  <h3>@ViewBag.Message</h3>  <p>Use this area to provide additional information.</p>  <div> móje tekst @Model </div> |

Helper, który wygeneruje nam sam link czyli np. <http://google.com> , wskażemy nazwę akcji i następnie nazwę kontrolera. Zamieni on ten kod na linka

|  |
| --- |
| <a class="navbar-brand" href="@Url.Action("Index","Home")">Strona Główna</a> |

Inny helper, który wygeneruje cały kod linka html czyli <a …..>….</a>  
parametry: nazwa wyświetlana, nazwa akcji, nazwa kontrolera, null,   
następnie html attributes , użyliśmy słowa kluczowego z c# class, jeżeli chcemy użyć go jawnie musimy poprzedzić go znakiem @

|  |
| --- |
| @Html.ActionLink("Kontakt", "Contact", "Home", null, new { @class = "nav-link" }) |

Partial View

|  |
| --- |
| @Html.Partial("\_LoginPartial") |