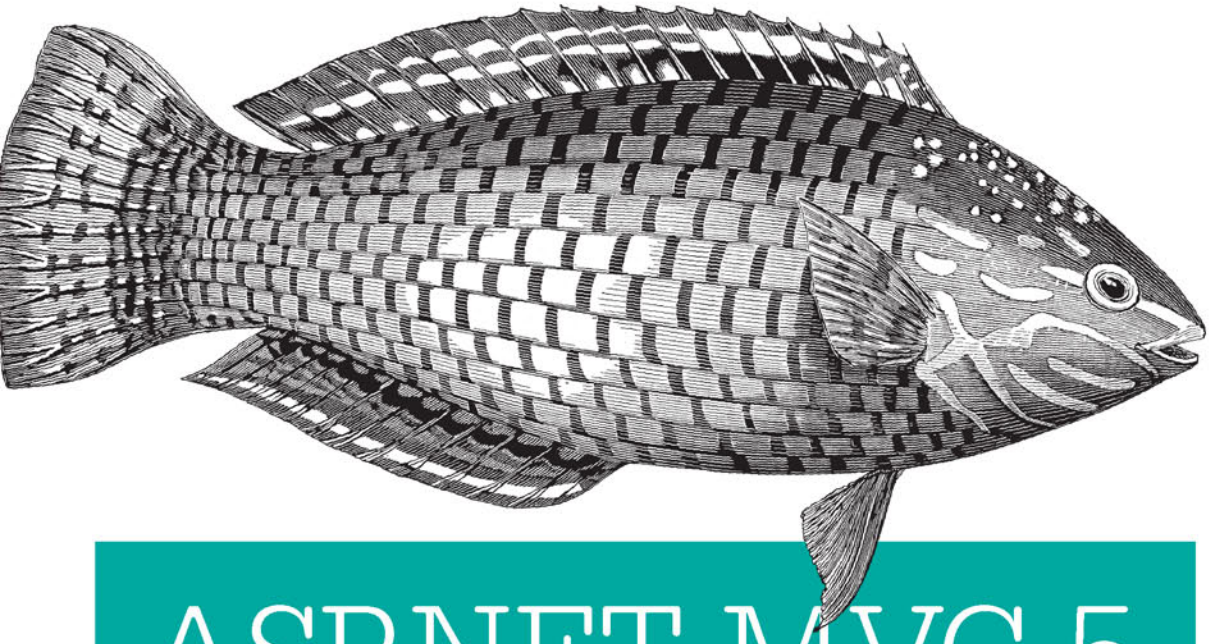
O'REILLY®



ASP.NET MVC 5, Bootstrap i Knockout.js

TWORZENIE DYNAMICZNYCH I ELASTYCZNYCH APLIKACJI INTERNETOWYCH

Helion

Jamie Munro

Tytuł oryginału: ASP.NET MVC 5 with Bootstrap and Knockout.js

Tłumaczenie: Piotr Pilch

ISBN: 978-83-283-2050-5

Wydawnictwo HELION
ul. Kościuszki 1c, 44-100 GLIWICE
tel. 32 231 22 19, 32 230 98 63
e-mail: *helion@helion.pl*
WWW: *http://helion.pl* (księgarnia internetowa, katalog książek)

Pliki z przykładami omawianymi w książce można znaleźć pod adresem:
ftp://ftp.helion.pl/przyklady/aspboo.zip

Drogi Czytelniku!
Jeżeli chcesz ocenić tę książkę, zajrzyj pod adres
http://helion.pl/user/opinie/aspboo
Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

Printed in Poland.

Spis treści

ROZDZIAŁ 1.

Wprowadzenie do wzorca architektury MVC

MVC to wzorzec architektury. Skrót wywodzi się od terminu *Model-View-Controller*. Moją definicję wzorca MVC mogę podsumować w następujący sposób:

- Model zarządza danymi aplikacji. Każdy model reprezentuje przeważnie jedną tabelę lub większą liczbę tabel w bazie danych.
- Widok zawiera wizualną reprezentację aplikacji. W witrynach internetowych osiąga się to zwykle z wykorzystaniem języków HTML i JavaScript oraz arkuszy stylów CSS.
- Kontroler pełni funkcję pośrednika między modelem i widokiem. Typowy kontroler będzie żądać danych od modelu i przekazywać je widokowi, aby użyć ich podczas wyświetlania danych. W czasie zapisywania danych kolejność działań byłaby odwrotna. Kontroler odebrałby dane z widoku i przekazał je modelowi w celu ich zapisania.

ASP.NET MVC 5 to środowisko implementujące wzorzec architektury MVC.

Termin **MVC** będzie wielokrotnie powtarzany w książce. W większości sytuacji będę się odwoływać do środowiska MVC implementującego wzorzec MVC.

Tworzenie pierwszego projektu

Oprogramowanie Visual Studio oferuje różne szablony ułatwiające rozpoczęcie projektu. W książce skoncentrowałem się na dwóch konkretnych szablonach: MVC i Web API.

Szablon MVC umożliwia tworzenie aplikacji internetowych, które korzystają z wzorca architektury *Model-View-Controller*. Szablon ten zostanie bardziej szczegółowo omówiony w książce.

Szablon Web API pozwala na tworzenie aplikacji internetowych RESTful. REST to kolejny typ wzorca architektury oprogramowania, powszechnie wykorzystywany do tworzenia interfejsów API lub aplikacji klient-serwer. Szablon Web API z łatwością integrowany jest z wzorcem architektury MVC, co umożliwia ponowne wykorzystanie kodu w projektach opartych na szablonach MVC i Web API. Zademonstruję to w dalszej części książki.

W ramach pierwszego przykładu skupię się na szablonie MVC. W oknie *Visual Studio Start Page* lub z menu *File/New* wybierz opcję *Project*. Jak widać, dostępnych jest wiele różnych typów projektów do utworzenia. W menu *Templates* rozwiń pozycję *Visual C#*. Z menu wybierz opcję *Web*.

W swojej domyślnej instalacji dysponuję tylko jedną opcją aplikacji internetowej ASP.NET. Wybierz ją i wprowadź nazwę w udostępnionym polu. Wpisałem nazwę **`BootstrapIntroduction`** (wprowadzenie do środowiska Bootstrap).

Po wybraniu nazwy projektu kliknij przycisk *OK*. Zostanie zaprezentowanych kilka różnych szablonów aplikacji internetowych. Wybierz szablon wzorca MVC, a zauważysz dodatkowe opcje umożliwiające dołączenie interfejsu Web API oraz projektu testu jednostkowego. Na potrzeby przykładu opcje pozostawię niezaznaczone.

W wypadku szablonu MVC dostępne są też cztery różne opcje uwierzytelniania:

No Authentication

Wszystkie strony witryny internetowej będą publicznie dostępne.

Individual User Accounts

Opcja sprawia, że aplikacja internetowa umożliwi użytkownikom rejestrowanie się i logowanie za pomocą utworzonej nazwy użytkownika i hasła. Opcja zapewnia też kilka różnych wariantów logowania w serwisach społecznościowych, w tym na Facebooka i Twittera, na konto Google i konto Microsoft. W każdym z tych wypadków informacje będą przechowywane w automatycznie tworzonej bazie danych członków.

Work And School Accounts

Opcja powoduje, że aplikacja internetowa zostanie zintegrowana z usługami Active Directory, Microsoft Azure Active Directory lub Office 365 w celu zabezpieczenia stron w obrębie aplikacji.

Windows Authentication

Dzięki tej opcji aplikacja internetowa jest zabezpieczana za pomocą modułu Windows Authentication IIS. Taki wariant jest powszechnie stosowany w wypadku aplikacji intranetowych, w których wszystkie konta użytkowników istnieją na serwerze WWW obsługującym aplikację.

Na potrzeby omawianego przykładu opcję zabezpieczeń zmieniłem na *No Authentication* (rysunek 1.1).

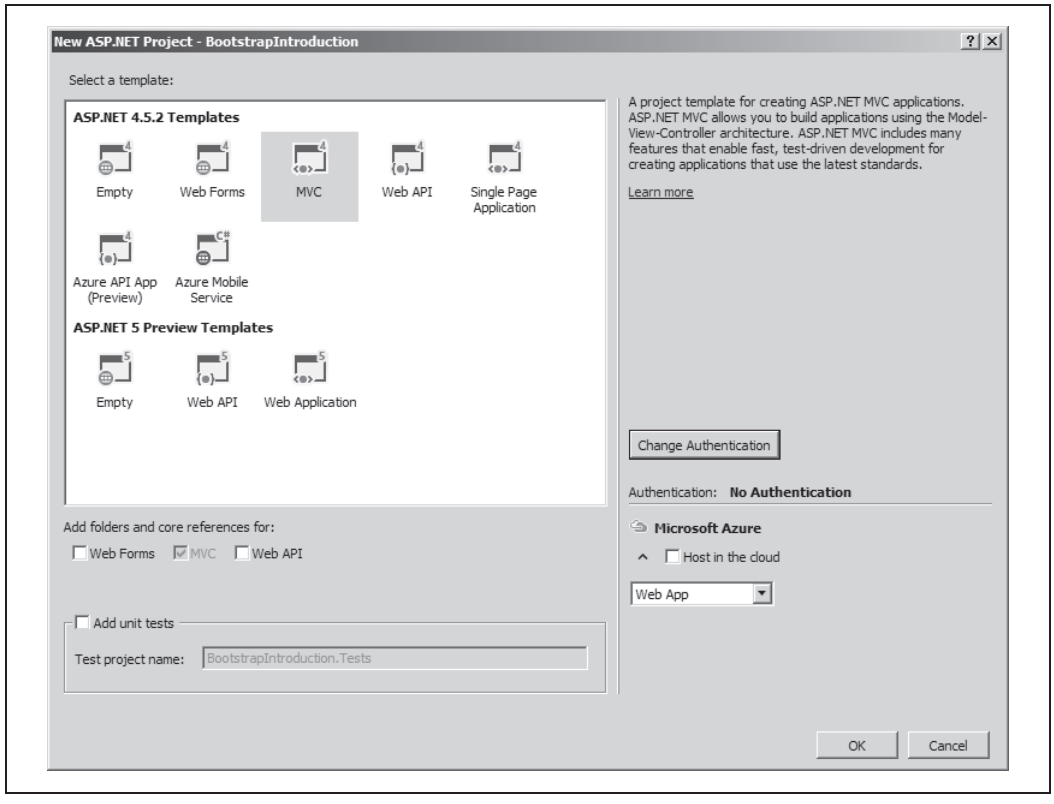
Gdy będziesz gotowy do zakończenia tworzenia projektu, kliknij przycisk *OK*. W zależności od wydajności komputera minutę albo dwie może potrwać dla projektu proces konfigurowania i wstępnego ładowania wielu przydatnych plików i folderów, które ułatwią rozpoczęcie pracy.

Po zakończeniu procesu możliwe będzie wybranie opcji *Start Debugging* z menu *Debug*. Spowoduje ona uruchomienie nowej aplikacji internetowej w domyślnej przeglądarce internetowej.

Po przyjrzeniu się folderom zauważysz, że z każdą rolą we wzorcu MVC powiązany jest osobny folder. Folder *Views* zawiera zwykle podfolder o nazwie zgodnej z nazwą kontrolera, ponieważ w folderze tym znajduje się zwykle kilka widoków, aby w prosty sposób umożliwić uporządkowanie plików.

Jeśli rozwiniesz folder *Views*, ujrzysz dwa podfoldery: *Home* i *Shared*. Folder *Home* dopasowany jest do klasy `HomeController` istniejącej w folderze *Controllers*. Folder *Shared* jest używany na potrzeby widoków zawierających wspólny kod innych widoków.

Widoki te obejmują układy lub widoki częściowe. Układy to specjalne widoki zawierające na każdej stronie widok wielokrotnego użycia. Widok częściowy jest widokiem do wielokrotnego zastosowania, posiadającym podzbiór danych, które mogą być dołączone do jednej strony lub ich większej liczby.



Rysunek 1.1. Tworzenie projektu

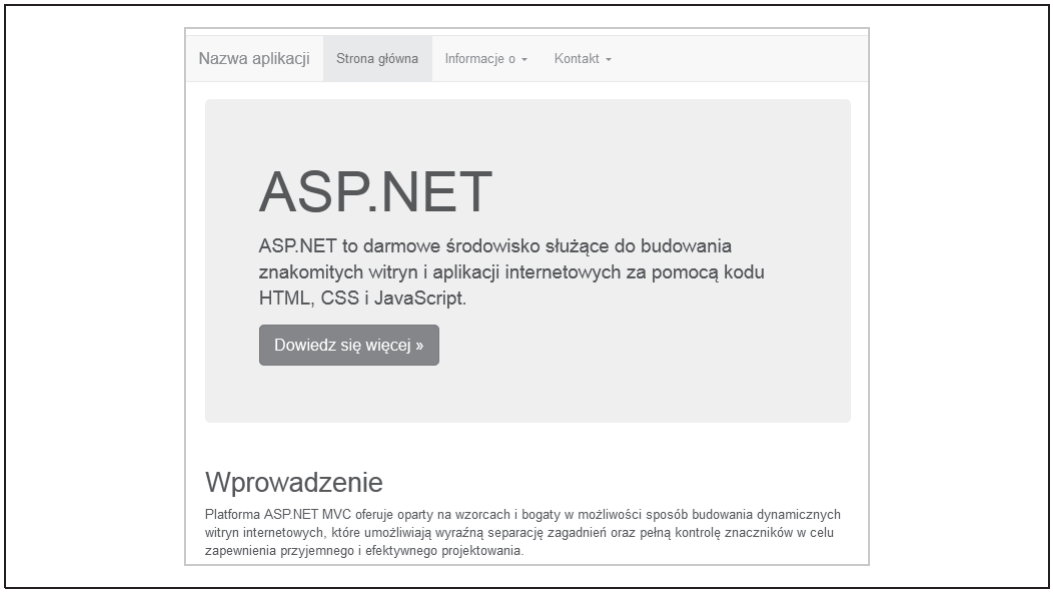
Rysunek 1.2 prezentuje zrzut ekranu domyślnej witryny internetowej tworzonej z wykorzystaniem nowego projektu MVC. Menu zawiera trzy odnośniki: *Strona główna*, *Informacje o* i *Kontakt*. Pamiętajmy o nich, rozpoczynając poznawanie kodu.

Analizowanie kontrolera HomeController

Zacznijmy od przyjrzenia się kontrolerowi. W folderze *Controllers* znajduje się plik o nazwie *HomeController.cs*, którego zawartość powinna być podobna do zaprezentowanej w przykładzie 1.1.

Przykład 1.1. Plik HomeController.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.Mvc;
namespace BootstrapIntroduction.Controllers
{
  public class HomeController : Controller
  {
    public ActionResult Index()
    {
      return View();
    }
```



Rysunek 1.2. Domyślna witryna internetowa

```
        public ActionResult About()
        {
          ViewBag.Message = "Strona z opisem aplikacji";
          return View();
        }
        public ActionResult Contact()
        {
          ViewBag.Message = "Strona z informacjami kontaktowymi";
          return View();
        }
      }
    }
```

`HomeController` to klasa zawierająca trzy metody: `Index`, `About` i `Contact`. W terminologii wzorca MVC są one często określane mianem *akcji*. Akcja o nazwie `Index` to zwykle główny punkt wejścia kontrolera. Zauważ, że akcje `About` i `Contact` dopasowują nazwy utworzonych odnośników — wyjątkiem jest akcja `Home`, ponieważ odnosi się ona do akcji `Index`.

Wszystkie kontrolery w aplikacji MVC będą rozszerzać podstawową klasę `Controller`. Każda metoda w klasie zwraca typ o nazwie `ActionResult`. W większości sytuacji typ ten będzie zwracany przez wszystkie używane akcje. W zamieszczonych dalej przykładach zostaną objaśnione inne zwracane typy.

Akcje `About` i `Contact` wprowadzają też właściwość `ViewBag` klasy `Controller`. Właściwość ta umożliwia dynamiczne przekazywanie danych widokowi. Przykład 1.2 zademonstruje sposób jej użycia w widoku.

Właściwość ViewBag
Właściwość `ViewBag` pozwala na współużytkowanie danych między kontrolerami i widokami. Zmienna ta jest definiowana jako typ dynamiczny i nie posiada żadnych predefiniowanych właściwości. Dzięki temu możliwe jest podanie dowolnej nazwy właściwości, która może zawierać dowolny typ danych.

I wreszcie — każda akcja jest zwracana z wywołaniem funkcji `View`. Metoda ta istnieje w klasie `Controller`, rozszerzanej przez wszystkie kontrolery. W wyniku jej zastosowania ładowany jest widok i wykonywany jest kod Razor zawarty w pliku *.cshtml*. W przykładzie 1.1 nie są przekazywane funkcji żadne parametry, co oznacza, że domyślnie wzorzec MVC będzie szukać widoku o takiej samej nazwie jak nazwa funkcji.

Analizowanie widoków

Po rozwinięciu zawartości folderu *Views* widoczny jest podfolder o nazwie *Home*. Znajdują się w nim trzy pliki: *About.cshtml*, *Contact.cshtml* i *Index.cshtml*. Nazwa każdego pliku jest zgodna z nazwą jego akcji w kontrolerze. Rozszerzenie *.cshtml* to skrót od C# HTML. Widoki te umożliwiają użycie składni silnika Razor (*http://bit.ly/razor-syntax*), która stanowi kombinację języków HTML i C#. Daje to możliwość implementowania typowych technik tworzenia kodu, takich jak instrukcje warunkowe, pętle i zwracanie danych dynamicznych (na przykład wcześniej wspomniana właściwość `ViewBag`).

W przykładzie 1.2 zaprezentowano zawartość pliku *About.cshtml* domyślnej strony tworzonej wraz z projektem. Elementy korzystające ze składni silnika Razor zaczynają się od symbolu `@`. W połączeniu ze znakiem nawiasu klamrowego `{` pozwala to na stosowanie wielowierszowego kodu C#. W tym przykładzie dla właściwości `ViewBag` ustawiany jest tytuł strony (wartość `"Informacje o"`). Właściwość `Title` powiązana z właściwością `ViewBag` jest powszechnie używana w układzie współużytkowanym do ustawiania tytuły strony wyświetlanej przez przeglądarkę. W omawianym przykładzie właściwość `Title` jest też umieszczona w znaczniku `h2`.

Przykład 1.2. Plik About.cshtml

```
@{
  ViewBag.Title = "Informacje o";
}
<h2>@ViewBag.Title.</h2>
<h3>@ViewBag.Message</h3>
<p>Użyj tego obszaru do udostępnienia dodatkowych informacji.</p>
```

Oprócz właściwości `Title` widoczna jest właściwość `Message`, powiązana w znaczniku `h3` z właściwością `ViewBag`. Jak już wspomniano, właściwość `Message` została ustawiona w akcji kontrolera dla metod `About` i `Contact`.

Jest to znakomity przykład tego, jak kod Razor jest wstawiany do standardowego kodu HTML w celu wygenerowania treści dynamicznej w momencie wykonywania operacji renderowania w przeglądarce.

W czasie przeglądania strony *Informacje o* możesz zauważyć, że zawiera ona znacznie więcej kodu HTML niż tylko kilka elementów. Jest to wynikiem użycia układu współużytkowanego. Domyślnie wszystkie widoki są umieszczane w obrębie szablonu domyślnego, który znajduje się w innym folderze, o nazwie *Shared*, umieszczonym w folderze *Views*. Jeśli rozwiniesz folder *Shared*, ujrzysz plik *_Layout.cshtml* (przykład 1.3).

Przykład 1.3. Plik _Layout.cshtml

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8" />
```

```
<meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>@ViewBag.Title – Moja aplikacja ASP.NET</title>
  @Styles.Render("~/Content/css")
  @Scripts.Render("~/bundles/modernizr")
</head>
<body>
  <div class="navbar navbar-inverse navbar-fixed-top">
    <div class="container">
      <div class="navbar-header">
        <button type="button" class="navbar-toggle" data-toggle="collapse"
                data-target=".navbar-collapse">
          <span class="icon-bar"></span>
          <span class="icon-bar"></span>
          <span class="icon-bar"></span>
        </button>
        @Html.ActionLink("Nazwa aplikacji", "Index", "Home",
                  new { area = "" }, new { @class = "navbar-brand" })
      </div>
      <div class="navbar-collapse collapse">
        <ul class="nav navbar-nav">
          <li>@Html.ActionLink("Strona główna", "Index", "Home")</li>
          <li>@Html.ActionLink("Informacje o", "About", "Home")</li>
          <li>@Html.ActionLink("Kontakt", "Contact", "Home")</li>
        </ul>
      </div>
    </div>
  </div>
  <div class="container body-content">
    @RenderBody()
    <hr />
    <footer>
      <p>&copy; @DateTime.Now.Year – Moja aplikacja ASP.NET</p>
    </footer>
  </div>
  @Scripts.Render("~/bundles/jquery")
  @Scripts.Render("~/bundles/bootstrap")
  @RenderSection("scripts", required: false)
</body>
</html>
```

Układ domyślny zawiera kod HTML wielokrotnego użytku, pojawiający się na każdej stronie w obrębie witryny (są to takie elementy jak tytuł strony, nagłówek, stopka, arkusze CSS czy kod JavaScript).

Renderowany widok wstawiany jest do tego układu w momencie wywołania metody `RenderBody` za pośrednictwem kodu Razor.

W wypadku tego układu współużytkowanego ma miejsce wiele innych rzeczy. Układ korzysta z kilku klas pomocniczych zapewnianych przez środowisko MVC, takich jak klasa `HtmlHelper` (*http://bit.ly/htmlhelper*) — do tworzenia odnośników, `Scripts` (*http://bit.ly/scripts-render*) — do dołączania plików JavaScript, `Styles` (*http://bit.ly/styles-render*) — do dołączania plików CSS, a także klasa `RenderSection`, która umożliwia widokom wskazywanie konkretnej treści do wstawienia w określonym miejscu układu współużytkowanego. Zostanie to zaprezentowane w rozdziale 2.

Struktura adresu URL

W momencie uruchomienia domyślnej witryny internetowej zostały utworzone następujące trzy odnośniki:

- *Strona główna*. Kieruje do głównego katalogu witryny (*/*).
- *Informacje o*. Kieruje do katalogu */Home/About*.
- *Kontakt*. Kieruje do katalogu */Home/Contact*.

Odnośniki te działają, ponieważ na początku tworzenia projektu MVC odpowiednio skonfigurowano trasę domyślną. Trasy umożliwiają witrynie internetowej informowanie wzorca MVC o tym, jak powinien odwzorować adres URL na konkretny kontroler i akcję w projekcie.

Trasy są konfigurowane w pliku *App_Start/RouteConfig.cs*. Przykład 1.4 prezentuje trasę domyślną skonfigurowaną z wykorzystaniem nowego projektu MVC.

Przykład 1.4. Trasa domyślna

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

W przykładowej konfiguracji trasy definiowane są następujące trzy istotne elementy:

- Nazwa. Każda trasa musi mieć unikatową nazwę.
- Adres URL. Jest to względny adres URL występujący po nazwie domeny witryny internetowej. Adres URL może zawierać kombinację statycznego tekstu ze zmiennymi.
- Konfiguracja domyślna. Jeśli nie zapewniono dowolnej ze zmiennych w adresie URL, mogą być dla nich ustawione wartości domyślne.

Jeśli ponownie przeanalizujemy wymienione wcześniej odnośniki, to dojdziemy do wniosku, że ich działanie wynika z następujących powodów:

- W wypadku przejścia do katalogu / adres URL nie zawiera żadnego kontrolera, akcji ani identyfikatora. Ustawiane wartości domyślne wskazują, że zostanie użyty kontroler `HomeController` i akcja `Index`, a identyfikator nie musi istnieć.
- W wypadku przejścia do katalogów */Home/About* i */Home/Contact* nie są używane żadne wartości domyślne, ponieważ w adresie URL są określone zarówno kontroler, jak i akcja.

Okazuje się, że odnośnik *Strona główna* może być też dostępny za pośrednictwem katalogów */Home* i */Home/Index*. Gdy jednak adresy URL są tworzone w obrębie widoku, wzorzec MVC wybiera spośród nich najkrótszy i najwłaściwszy adres URL, czyli po prostu katalog /.

W wypadku takiej trasy możesz utworzyć nowy kontroler i (lub) akcję, a ponadto trasa może być automatycznie dostępna przy użyciu jej nazwy i nazwy akcji.

Podsumowanie

Jeśli dopiero rozpoczynasz przygodę z wzorcem architektury MVC, treść tego rozdziału może sprawiać na Tobie przytłaczające wrażenie. Predefiniowane szablony MVC w środowisku Visual Studio są dość rozbudowane i zapewniają projektantom sporą „zaliczkę" w momencie realizowania projektów.

Dlatego też trudno omówić każdy szczegół dotyczący tego, co jest tworzone. W tym wprowadzającym rozdziale przedstawiłem wiele podstawowych funkcji, które umożliwią rozpoczęcie pracy.

Istnieje znacznie więcej szczegółów do omówienia, związanych z wzorcem architektury *Model-View-Controller*. W kolejnych rozdziałach książki, prezentujących bardziej zaawansowane zagadnienia, szczegóły te będą w znacznym stopniu przybliżane.

Skorowidz

F

G

H

I

K

L

Ł

M

N

O

P

R

S

Ś

T

U

V

W

Z

Ż

O'REILLY®

Praktyczne wprowadzenie do świata AngularJS!

Budowa aplikacji internetowych często wymaga integracji różnych technologii. Praca programisty staje się wówczas nieco trudniejsza, ale za to utworzone aplikacje mogą działać na wielu nowoczesnych urządzeniach bez potrzeby pisania kodu dla każdego sprzętu z osobna. W tej książce przedstawiono wyjątkowo udane połączenie trzech technologii, czyli środowiska ASP.NET MVC 5 umożliwiającego budowę zaawansowanych aplikacji internetowych, interakcję z bazą danych oraz dynamiczne renderowanie kodu HTML, środowiska Bootstrap pozwalającego na tworzenie ładnych i elastycznych widoków, a także biblioteki Knockout.js, która łączy te technologie, a jednocześnie rozszerza elastyczny projekt aplikacji internetowej dzięki dynamicznym interakcjom po stronie klienta, sterowanym przez serwerową aplikację internetową.

Książka, którą trzymasz w dłoniach, to podręcznik, który umożliwi Ci płynne tworzenie aplikacji zgodnych z wzorcem MVC (Model-View-Controller) za pomocą trzech technologii: ASP.NET MVC 5, środowiska Bootstrap i biblioteki Knockout.js. Poznasz sposoby, które pozwolą Ci napisać bardzo dobrze zorganizowane i łatwe w utrzymaniu projekty.

Dowiedz się, jak:

- tworzyć dobrze zorganizowane i łatwe w utrzymaniu aplikacje internetowe
- budować serwerowe aplikacje internetowe na platformie ASP.NET MVC 5, korzystać z baz danych i w dynamiczny sposób renderować strony HTML
- tworzyć elastyczne widoki za pomocą środowiska Bootstrap i umożliwiać ich renderowanie na przeróżnych nowoczesnych urządzeniach
- ulepszać projekt elastycznej aplikacji internetowej za pomocą biblioteki Knockout.js z wykorzystaniem szybkich interakcji po stronie klienta

Jamie Munro – od ponad 15 lat zajmuje się projektowaniem witryn i aplikacji internetowych. Od blisko dekady jest mentorem młodszych programistów: przekazuje im swoją wiedzę i pozwala korzystać ze swojego wieloletniego doświadczenia.

Helion

41501 numer katalogowy
księgarnia internetowa
http://helion.pl
zamówienia telefoniczne
0 801 339900
0 601 339900

Sprawdź najnowsze promocje:
http://helion.pl/promocje
Książki najchętniej czytane:
http://helion.pl/bestseliery
Zamów informacje o nowościach:
http://helion.pl/nowosci

Helion SA
ul. Kościuszki 1c, 44-100 Gliwice
tel.: 32 230 98 63
e-mail: helion@helion.pl
http://helion.pl

Informatyka w najlepszym wydaniu

sięgnij po WIĘCEJ
KOD KORZYŚCI

ISBN 978-83-283-2050-5
9 788328 320505

cena: 49,00 zł