KOMPENDIUM WIEDZY O PLATFORMIE ASP.NET MVC 5!

Apress

ASP.NET MVC 5

Zaawansowane programowanie

Adam Freeman

Helion

Tytuł oryginału: Pro ASP.NET MVC 5

Tłumaczenie: Robert Górczyński

ISBN: 978-83-283-0651-6

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/asp5zp.zip

Drogi Czytelniku!
Jeżeli chcesz ocenić tę książkę, zajrzyj pod adres
http://helion.pl/user/opinie/asp5zp
Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

Printed in Poland.

Spis treści

ROZDZIAŁ 14.

Przegląd projektu MVC

Zanim zagłębię się w szczegółach funkcji platformy MVC, podam nieco informacji ogólnych. W tym rozdziale przedstawię strukturę i naturę aplikacji ASP.NET MVC, w tym domyślną strukturę projektu oraz konwencje nazewnictwa. Niektóre konwencje są opcjonalne, z kolei inne na sztywno definiują sposób, w jaki działa platforma MVC.

Korzystanie z projektów MVC z Visual Studio

Gdy tworzymy nowy projekt MVC, Visual Studio daje nam możliwość wyboru jednego z kilku punktów startowych. Celem jest ułatwienie procesu nauki nowym programistom, a także zastosowanie pewnych pozwalających na oszczędność czasu najlepszych praktyk podczas implementacji najczęściej używanych funkcji. Tego rodzaju wsparcie oferowane programistom ma postać szablonów wykorzystywanych do tworzenia kontrolerów i widoków przygotowywanych z użyciem kodu szablonu do wymiany obiektów danych, edycji właściwości modelu itd.

W Visual Studio 2013 oraz MVC 5 firma Microsoft uaktualniła szablony i tak zwane *szkielety kodu*, niwelując różnice między poszczególnymi rodzajami projektów ASP.NET. Ma to na celu dostarczenie szerszej gamy szablonów projektów oraz konfiguracji.

Po lekturze pierwszej części książki nie powinieneś mieć wątpliwości, że nie jestem fanem podejścia polegającego na użyciu szablonów projektów. Intencje Microsoftu są dobre, ale wykonanie pozostawia sporo do życzenia. Jedną z cech charakterystycznych, którą niezwykle cenię w platformach ASP.NET i MVC, jest ogromna elastyczność pozwalająca na dostosowanie platformy do preferowanego przez daną osobę stylu programowania. Tworzone i wypełniane kodem przez Visual Studio projekty, klasy i widoki sprawiają, że czuję się ograniczony i zmuszony do pracy w stylu zupełnie kogoś innego. Ponadto automatycznie generowana zawartość i konfiguracja wydają się być zbyt ogólne, aby stały się szczególnie użyteczne. W rozdziale 10. wspomniałem, że jednym z niebezpieczeństw użycia układu responsywnego dla urządzeń mobilnych jest uzyskanie przeciętnego kodu, który jest dopasowany do jak największej liczby urządzeń. W podobny sposób można określić szablony Visual Studio. Microsoft nie wie, jakiego rodzaju aplikacje będziesz chciał tworzyć, i dlatego stara się zapewnić obsługę maksymalnej liczby scenariuszy. Wynik jest tak bezbarwny i uogólniony, że zawartość generowaną przez Visual Studio wyrzucam od razu na początku pracy z projektem.

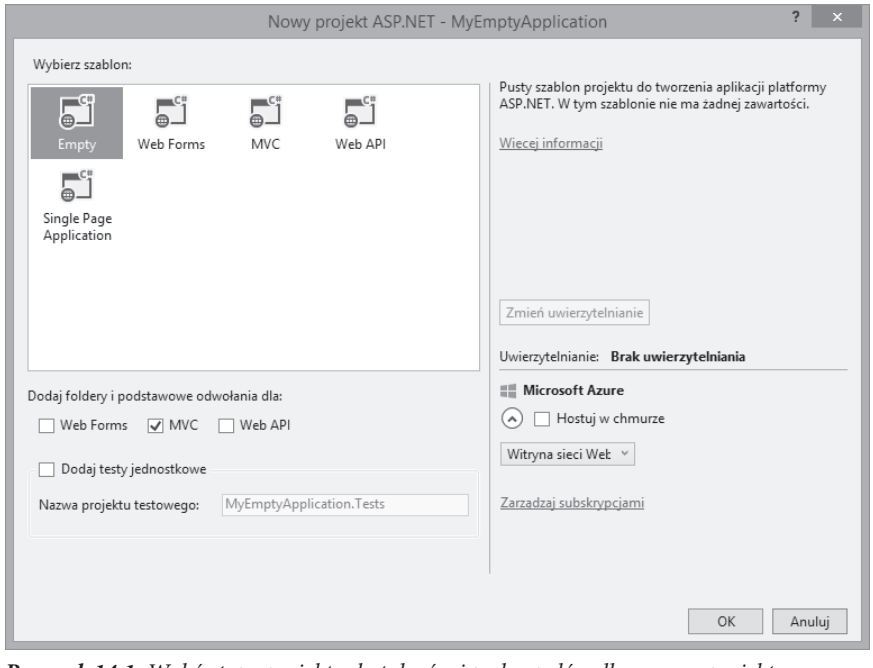
Moja rada (udzielana każdemu, kto popełnia błąd, pytając o nią) brzmi: rozpoczynaj pracę z pustym projektem, a następnie dodawaj niezbędne katalogi, pliki i pakiety. Dzięki takiemu podejściu nie tylko lepiej poznasz sposób działania platformy MVC, ale również zachowasz pełną kontrolę nad zawartością aplikacji.

Moje osobiste preferencje nie muszą pasować do Twojego doświadczenia w zakresie programowania. Być może dostarczane przez Visual Studio szablony i szkielety kodu uznasz za dużo bardziej użyteczne, niż są dla mnie, zwłaszcza jeżeli dopiero zaczynasz programowanie na platformie ASP.NET i nie wykształciłeś jeszcze

swojego stylu programowania. Ponadto szablony projektów możesz uznać za użyteczny zasób i źródło idei. Powinieneś jednak zachować ostrożność podczas dodawania funkcji do aplikacji, zanim dokładnie nie poznasz jej sposobu działania.

Tworzenie projektu

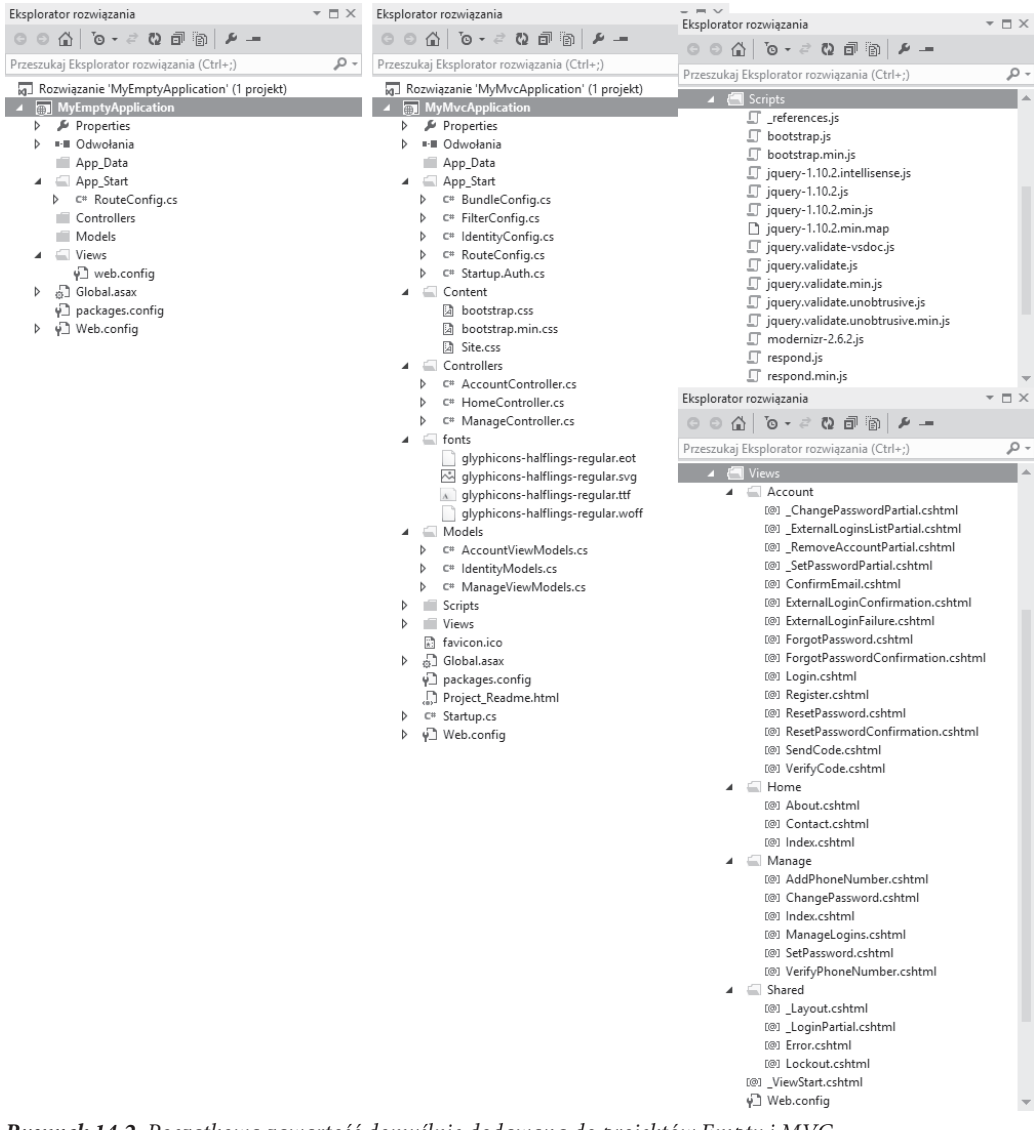
Kiedy po raz pierwszy tworzysz projekt MVC, do dyspozycji masz dwa punkty startowe: szablony *Empty* i *MVC*. Nazwy szablonów są nieco mylące, ponieważ podstawowe katalogi i podzespoły niezbędne dla platformy MVC można dodać do każdego szablonu projektu. W tym celu należy zaznaczyć pole wyboru *MVC* w sekcji *Dodaj foldery i podstawowe odwołania dla:* okna dialogowego *Nowy projekt*, jak pokazano na rysunku 14.1. W przypadku szablonu projektu *MVC* odpowiednia opcja jest zaznaczona domyślnie.



Rysunek 14.1. *Wybór typu projektu, katalogów i podzespołów dla nowego projektu*

Faktyczna różnica polega na umieszczeniu dodatkowej zawartości w szablonie projektu *MVC*. W ten sposób, tworząc nowy projekt, programista otrzymuje prawdziwy punkt startowy, zawierający pewne domyślne kontrolery, widoki, konfigurację zabezpieczeń, popularne pakiety JavaScript i CSS (na przykład jQuery i Bootstrap), a układ jest oparty na bibliotece Bootstrap, dostarczającej motyw graficzny dla zawartości aplikacji. Z kolei szablon *Empty* zawiera po prostu podstawowe odwołania wymagane przez platformę MVC oraz najprostszą strukturę katalogów. Szablon *MVC* dodaje znaczną ilość różnego rodzaju kodu, a różnica między omawianymi szablonami jest wyraźnie widoczna na rysunku 14.2, który pokazuje zawartość dwóch nowo utworzonych projektów. Projekt po lewej stronie utworzono na podstawie szablonu *Empty* wraz z zaznaczonym polem wyboru `MVC`. Okna po prawej stronie pokazują zawartość projektu utworzonego na podstawie szablonu *MVC*. Aby zmieścić na stronie książki listę wszystkich plików, zawartość niektórych katalogów projektu musiałem otworzyć w oddzielnych oknach. W przeciwnym razie cała lista nie zmieściłaby się na stronie książki.

Wprawdzie liczba dodatkowych plików umieszczanych w projekcie opartym na szablonie *MVC* może przerażać, ale nie jest tak źle. Część plików jest powiązana z mechanizmem uwierzytelniania, inne to pliki JavaScript i CSS dostarczane w postaci zarówno zwykłej, jak i zminimalizowanej. (Sposób użycia tych plików przedstawię w rozdziale 26.).



Rysunek 14.2. *Początkowa zawartość domyślnie dodawana do projektów Empty i MVC*

■ **Wskazówka** Podzespoły Visual Studio są przez szablon *MVC* tworzone za pomocą pakietów NuGet. Oznacza to, że użyte pakiety możesz zobaczyć po wybraniu opcji *Narzędzia/Menedżer pakietów NuGet/Zarządzaj pakietami NuGet dla rozwiązania…*. To jednocześnie wskazuje na możliwość dodawania tych samych pakietów do dowolnego projektu, w tym także utworzonego na podstawie szablonu *Empty*. (Takie rozwiązanie zastosowałem w pierwszej części książki).

Niezależnie od rodzaju, szablony pozwalają na tworzenie projektów o podobnej strukturze. Niektóre z elementów projektu mają specjalne role, wbudowane w ASP.NET lub platformę MVC. Inne są wynikiem konwencji nazewnictwa. Każdy z tych plików i katalogów został opisany w tabeli 14.1. Część plików może nie znajdować się w domyślnych projektach, ale zostaną omówione w dalszych rozdziałach.

Tabela 14.1. *Podsumowanie elementów projektu MVC*

Katalog lub plik	Opis	Uwagi
/App_Data	W katalogu tym umieszczamy prywatne dane, takie jak pliki XML lub bazy danych wykorzystywane przez SQL Server Express, SQLite bądź inne repozytoria plikowe.	IIS nie udostępnia zawartości tego katalogu.
/App_Start	Ten katalog zawiera pewne ustawienia początkowe projektu, między innymi definicje tras, filtry oraz paczki plików.	System routingu zostanie omówiony w rozdziałach 15. i 16., filtry w rozdziale 18., natomiast paczki plików w rozdziale 26.
/Areas	Obszary umożliwiają partycjonowanie ogromnej aplikacji na mniejsze fragmenty.	Obszary zostaną omówione w rozdziale 15.
/bin	Umieszczane są tu skompilowane podzespoły aplikacji MVC wraz z wszystkimi wykorzystywanymi podzespołami, które nie znajdują się w GAC.	Nie zobaczysz katalogu *bin* w oknie *Eksplorator rozwiązania*, o ile nie klikniesz przycisku *Pokaż wszystkie pliki*. Ponieważ te pliki binarne są generowane w czasie kompilacji, nie powinieneś ich przechowywać w systemie kontroli wersji.
/Content	Jest to katalog na statyczną treść, na przykład pliki CSS oraz obrazy.	Jest to konwencja, ale niewymagana. Statyczne dane można umieścić w dowolnym odpowiadającym nam miejscu projektu.
/Controllers	Znajdują się tu klasy kontrolerów.	Jest to konwencja. Klasy kontrolerów można umieszczać w dowolnym katalogu, ponieważ są kompilowane do tego samego podzespołu.
/Models	Jest to miejsce na klasy modelu widoku oraz modelu domeny, choć oprócz najprostszych aplikacji lepiej jest definiować model domeny w dedykowanym projekcie, jak pokazałem to w aplikacji SportsStore.	Jest to konwencja. Klasy modelu można definiować w dowolnym katalogu projektu lub w osobnym projekcie.
/Scripts	Jest to katalog przeznaczony na biblioteki JavaScript dla naszej aplikacji.	Jest to konwencja. Pliki skryptów można umieścić w dowolnej lokalizacji, ponieważ są one innym typem zawartości statycznej. Więcej informacji na temat zarządzania plikami skryptów znajdziesz w rozdziale 26.
/Views	Katalog ten jest przeznaczony na widoki i widoki częściowe, zwykle grupowane w katalogach mających nazwy kontrolerów, z którymi są skojarzone.	Plik */Views/Web.config* uniemożliwia udostępnianie zawartości tych katalogów. Widoki są generowane za pomocą metod akcji.
/Views/Shared	Katalog ten jest przeznaczony na pliki układów i widoków, które nie są skojarzone z konkretnym kontrolerem.	
/Views/ Web.config	To *nie* jest plik konfiguracyjny dla aplikacji. Zawiera on konfigurację wymaganą do tego, aby widoki działały w ASP.NET, oraz blokuje możliwość udostępniania widoków przez IIS. Przestrzenie nazw są domyślnie importowane do widoków.	

Tabela 14.1. *Podsumowanie elementów projektu MVC (ciąg dalszy)*

Katalog lub plik	Opis	Uwagi
/Global.asax	Definiuje globalną klasę aplikacji ASP.NET. Jego klasa kodu ukrytego (*/Global.asax.cs*) jest miejscem, w którym rejestrujemy konfigurację routingu, jak również dodajemy kod, jaki powinien wykonać się w czasie inicjalizacji lub wyłączenia aplikacji albo w przypadku wystąpienia nieobsłużonego wyjątku.	Plik *Global.asax* ma w aplikacji MVC taką samą funkcję jak w aplikacji Web Forms.
/Web.config	Jest to plik konfiguracyjny dla naszej aplikacji.	Plik *Web.config* ma w aplikacji MVC taką samą funkcję jak w aplikacji Web Forms.

Przedstawienie konwencji MVC

W projektach MVC występują dwa rodzaje konwencji. Pierwszy rodzaj to raczej sugestia na temat tego, w jaki sposób możemy tworzyć strukturę projektu. Jest to na przykład konwencja zachęcająca nas do umieszczenia wszystkich plików JavaScript w katalogu *Scripts*. Programiści MVC oczekują, że znajdą je w tym właśnie katalogu. Menedżer pakietów NuGet również umieszcza w nim pliki JavaScript dołączane do projektu MVC. Możemy jednak zmienić nazwę katalogu *Scripts* lub całkiem go usunąć i umieścić skrypty w dowolnym innym miejscu. Nie spowoduje to, że platforma MVC nie będzie w stanie uruchomić aplikacji.

Inny rodzaj konwencji wynika z zasady *konwencja przed konfiguracją*, która była jedną z przyczyn tak ogromnej popularności Ruby on Rails. Konwencja przed konfiguracją oznacza, że nie musimy jawnie konfigurować połączeń pomiędzy kontrolerami i ich widokami. Po prostu stosujemy określone konwencje nazewnictwa i wszystko działa bez zarzutu. W przypadku tego typu konwencji mamy mniejsze możliwości zmiany struktury projektu. W kolejnych punktach przedstawimy konwencje stosowane zamiast konfiguracji.

■ **Wskazówka** Wszystkie konwencje mogą być zmienione przez użycie własnego silnika widoku, co opiszę w rozdziale 20., ale to nie jest łatwe zadanie. W większości przypadków w projektach MVC będziesz miał jednak do czynienia z wymienionymi konwencjami.

Stosowanie konwencji dla klas kontrolerów

Klasa kontrolera *musi* kończyć się słowem `Controller`, np.: `ProductsController`, `AdminController` czy też `HomeController`. Odwołując się do kontrolera z poziomu projektu, na przykład podczas użycia metody pomocniczej HTML, podajemy pierwszą część nazwy (na przykład `Product`), a platforma MVC automatycznie doda `Controller` do nazwy i zacznie szukać klasy kontrolera.

■ **Wskazówka** Można zmienić to zachowanie przez utworzenie własnej implementacji interfejsu `IControllerFactory`, co opiszę w rozdziale 19.

Stosowanie konwencji dla widoków

Widoki i widoki częściowe powinny być umieszczone w katalogu */Views/nazwakontrolera*. Na przykład widok skojarzony z klasą `ProductController` powinien znajdować się w katalogu */Views/Product*.

■ **Wskazówka** Zwróć uwagę, że pomijamy drugą część nazwy klasy w podkatalogu *Views*; używamy katalogu */Views/Product*, a nie */Views/ProductController*. Może Ci się to wydawać na początku mało intuicyjne, ale szybko stanie się Twoją drugą naturą.

Platforma MVC oczekuje, że domyślny widok dla metody akcji powinien nosić nazwę tej metody. Na przykład widok skojarzony z metodą akcji `List` powinien mieć nazwę *List.cshtml*. Dlatego domyślny widok dla metody akcji `List` z klasy `ProductController` powinien znajdować się w */Views/Product/List.cshtml*. Domyślny widok jest używany, gdy z metody akcji zwrócimy wynik wywołania metody `View`, na przykład:

```
...
return View();
...
```

Możemy również podać nazwę innego widoku, na przykład:

```
...
return View("InnyWidok");
...
```

Zwróć uwagę, że nie podajemy rozszerzenia nazwy pliku ani ścieżki dostępu do widoku. Platforma MVC szuka widoku w katalogu o nazwie kontrolera, a następnie w katalogu */Views/Shared*. Dlatego widoki stosowane przez więcej niż jeden kontroler możemy umieścić w katalogu */Views/Shared*; platforma znajdzie je w razie potrzeby ich użycia.

Stosowanie konwencji dla układów

Konwencją nazewnictwa dla układów jest poprzedzanie ich nazw znakiem podkreślenia. Pliki układów są umieszczane w katalogu */Views/Shared*. Visual Studio tworzy plik układu o nazwie *_Layout.cshtml*, który wchodzi w skład wszystkich szablonów projektów poza *Pusta*. Układ ten jest stosowany domyślnie do wszystkich widoków poprzez plik */Views/_ViewStart.cshtml*. Jeżeli nie chcesz, aby domyślny widok był stosowany do widoku, możesz zmienić ustawienie w pliku *_ViewStart.cshtml*, definiując w nim inny plik układu (lub usuwając ten plik).

```
@{
    Layout = "~/Views/Shared/_MyLayout.cshtml";
}
```

Można również zablokować wszystkie układy dla pojedynczego widoku:

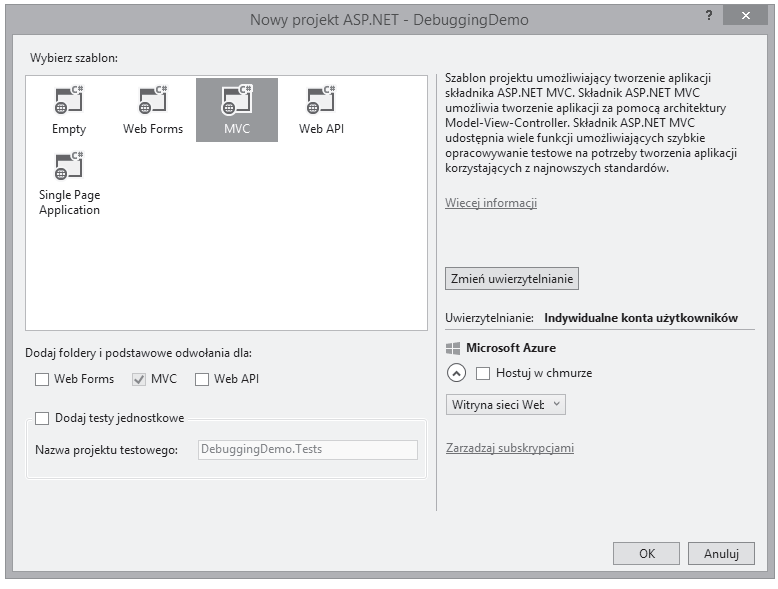
```
@{
    Layout = null;
}
```

Debugowanie aplikacji MVC

Aplikację ASP.NET MVC można debugować dokładnie tak samo jak aplikację ASP.NET Web Forms. Debuger w Visual Studio jest niezwykle zaawansowanym i elastycznym narzędziem, które ma wiele funkcji i zastosowań. W książce tej przedstawię jedynie kilka podstawowych funkcji. Pokażę, jak skonfigurować debuger i przeprowadzać różne zadania związane z usuwaniem błędów w projekcie MVC.

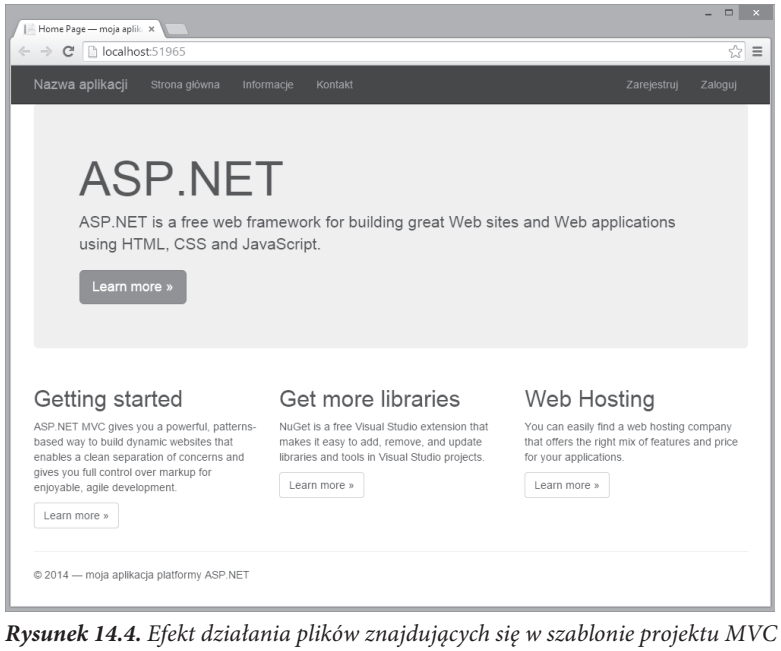
Tworzenie przykładowego projektu

Aby zademonstrować działanie debugera, utworzymy nowy projekt MVC, korzystający z szablonu *MVC*. W ten sposób będziesz mógł zobaczyć, jak przygotowywana jest zawartość i konfiguracja podstawowa projektu, a także jaki efekt ma zastosowanie motywu domyślnego w widokach. Nazwijmy nasz projekt `DebuggingDemo`. Jak pokazano na rysunku 14.3, jako uwierzytelnianie wybrano opcję `Indywidualne konta użytkowników`, która oznacza użycie podstawowego systemu uwierzytelniania użytkowników.



Rysunek 14.3. *Tworzenie projektu DebuggingDemo*

Po kliknięciu przycisku *OK* Visual Studio utworzy projekt, umieści w nim katalogi i pliki pakietów domyślnych znajdujących się w szablonie *MVC*. Dodane do projektu pliki i sposób ich konfiguracji możesz zobaczyć po uruchomieniu aplikacji (rysunek 14.4).



Rysunek 14.4. *Efekt działania plików znajdujących się w szablonie projektu MVC*

Projekt zawiera pewne miejsca zarejestrowane pozwalające na podanie nazwy aplikacji i promocję marki, a także oferuje łącza do dokumentów MVC, pakietów NuGet oraz opcji dotyczących hostingu.

Pasek nawigacyjny znajduje się na górze strony i ma taką samą postać, jakiej użyłem w aplikacji SportsStore. Ponadto w kodzie zastosowano pewne funkcje układu responsywnego. Aby się o tym przekonać, zmień szerokość okna przeglądarki internetowej.

Tworzenie kontrolera

Wprawdzie Visual Studio tworzy kontroler `Home` jako część projektu początkowego, ale jego kod zastąpimy przedstawionym na listingu 14.1.

Listing 14.1. *Zawartość pliku HomeController.cs*

```
using System.Web.Mvc;

namespace DebuggingDemo.Controllers
{
    public class HomeController : Controller
    {
        public ActionResult Index()
        {
            int firstVal = 10;
            int secondVal = 5;
            int result = firstVal / secondVal;

            ViewBag.Message = "Witamy na platformie ASP.NET MVC!";

            return View(result);
        }
    }
}
```

Tworzenie widoku

Visual Studio tworzy także plik widoku *Views/Home/Index.cshtml*, jako część pierwotnej zawartości projektu. Ponieważ nie potrzebujemy zawartości domyślnej tego widoku, zastąp ją kodem przedstawionym na listingu 14.2.

Listing 14.2. *Zawartość pliku Index.cshtml*

```
@model int

@{
    Layout = null;
}

<!DOCTYPE html>

<html>
<head>
    <meta name="viewport" content="width=device-width" />
    <link href="~/Content/Site.css" rel="stylesheet" type="text/css" />
    <title>Index</title>
</head>
<body>
     <h2 class="message">@ViewData["Message"]</h2>
    <p>
        Wynik obliczeń: @Model
    </p>
</body>
</html>
```

Ostatnim krokiem w trakcie tych przygotowań projektu jest dodanie stylu do pliku */Content/Site.css* przedstawionego na listingu 14.3 oraz zmiana jednego z istniejących. Plik *Site.css* jest tworzony przez Visual Studio jako część szablonu projektu *MVC*, stanowi domyślne miejsce dla stylów CSS aplikacji. (W przedstawionym na listingu 14.2 kodzie widoku dodałem element `<link>` importujący plik CSS do widoku *Index.cshtml*).

Listing 14.3. *Dodanie stylu do pliku /Content/Site.css*

```
body { padding-top: 5px; padding-bottom: 5px; }
.field-validation-error { color: #b94a48; }
.field-validation-valid { display: none; }
input.input-validation-error { border: 1px solid #b94a48; }
input[type="checkbox"].input-validation-error { border: 0 none; }
.validation-summary-errors { color: #b94a48; }
.validation-summary-valid { display: none; }
.no-color { background-color: white; border-style:none; }
.message { font-size: 20pt; text-decoration: underline; }
```

Uruchamianie debugera Visual Studio

Domyślnie Visual Studio włącza możliwość debugowania nowych projektów, choć warto wiedzieć, jak można samodzielnie to zmienić. Najważniejsze ustawienie znajduje się w pliku *Web.config*, położonym w katalogu głównym aplikacji. Odpowiednie ustawienie jest w elemencie `<system.web>`, jak pokazano na listingu 14.4.

■ **Ostrzeżenie** Nie należy instalować aplikacji na serwerze produkcyjnym bez wcześniejszego ustawienia wartości `false` opcji `debug`. Jeżeli do wdrożenia aplikacji używasz Visual Studio (podobnie jak to pokazałem w rozdziale 13.), wówczas odpowiednia zmiana zostanie wprowadzona automatycznie po wybraniu konfiguracji *Release* w projekcie.

Listing 14.4. *Atrybut Debug w pliku Web.config*

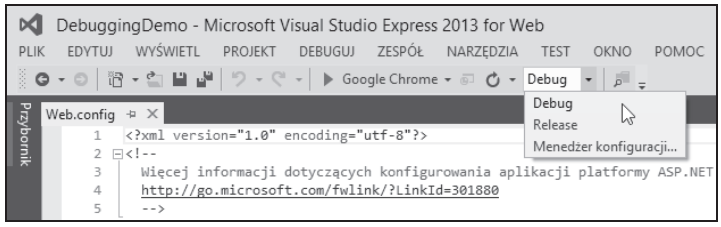
```
...
<system.web>
    <httpRuntime targetFramework="4.5.1" />
    <compilation debug="true" targetFramework="4.5.1" />
</system.web>
...
```

Spora liczba operacji kompilacji w projekcie MVC jest przeprowadzana, gdy aplikacja działa w serwerze IIS. W trakcie prac nad aplikacją musisz więc się upewnić, że atrybutowi `debug` jest przypisana wartość `true`. W ten sposób debuger będzie mógł operować na plikach klas wygenerowanych podczas kompilacji na żądanie.

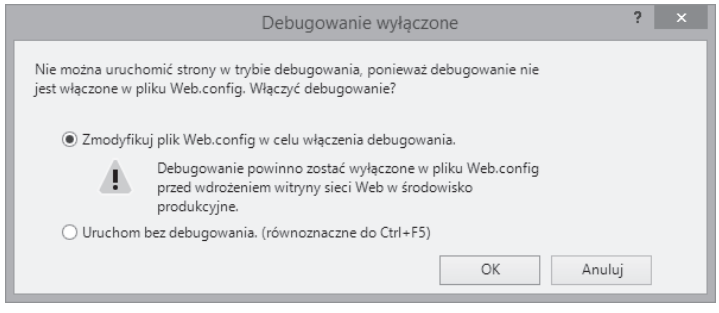
Oprócz zmiany w pliku konfiguracyjnym *Web.config*, konieczne jest upewnienie się, że Visual Studio umieszcza informacje debugowania w tworzonych plikach klas. Wprawdzie to nie ma znaczenia krytycznego, ale może powodować problemy, jeśli poszczególne ustawienia debugowania nie są zsynchronizowane. Upewnij się o wybraniu opcji konfiguracyjnej *Debug* na pasku narzędziowym Visual Studio, jak pokazano na rysunku 14.5.

Aby rozpocząć debugowanie aplikacji na platformie MVC, wybierz opcję *Start Debugging* z menu *Debuguj* w Visual Studio lub kliknij zieloną ikonę strzałki na pasku narzędziowym Visual Studio (wspomnianą ikonę widać na rysunku 14.5 tuż obok nazwy przeglądarki internetowej używanej do wyświetlenia aplikacji — w omawianym przykładzie jest to Google Chrome).

Jeżeli w pliku konfiguracyjnym *Web.config* wartością atrybutu `debug` jest `false`, wtedy podczas uruchamiania debugera Visual Studio wyświetli okno dialogowe pokazane na rysunku 14.6. Wybierz opcję pozwalającą Visual Studio na przeprowadzenie modyfikacji pliku *Web.config*, a następnie kliknij przycisk *OK*. Debuger zostanie uruchomiony.



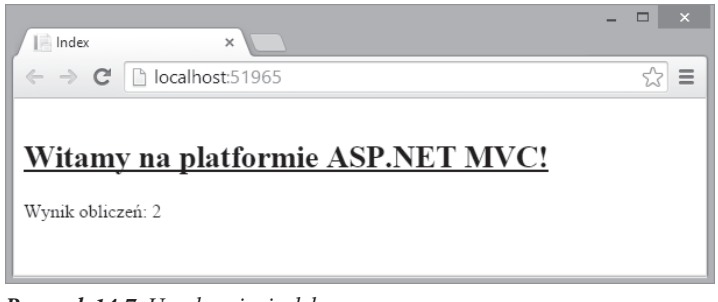
Rysunek 14.5. *Wybór opcji konfiguracyjnej Debug*



Rysunek 14.6. *Okno dialogowe wyświetlane przez Visual Studio, gdy debugowanie jest wyłączone w pliku Web.config*

W tym momencie aplikacja jest uruchomiona i wyświetlona w oknie przeglądarki, jak pokazano na rysunku 14.7.

Debuger został dołączony do naszej aplikacji, ale nie zauważymy żadnej różnicy do momentu przerwania jej działania przez debuger (przedstawię to w następnym punkcie). Aby zatrzymać debuger, wybierz opcję *Stop Debugging* z menu *Debuguj* lub zamknij okno przeglądarki internetowej.



Rysunek 14.7. *Uruchomienie debugera*

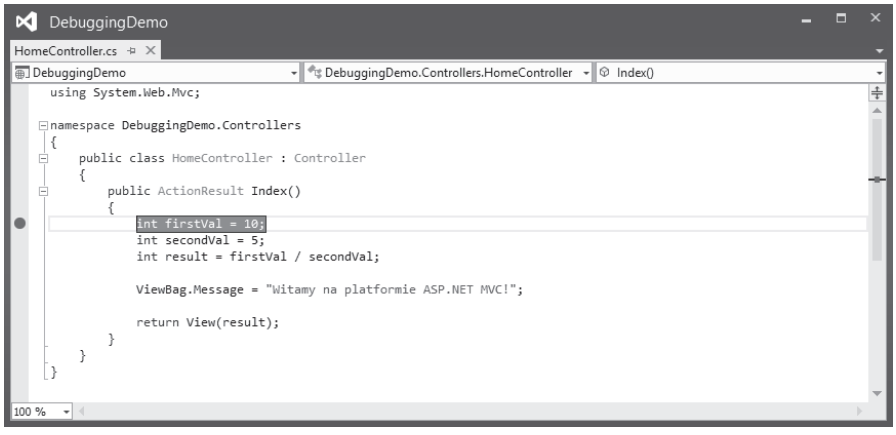
Przerywanie pracy aplikacji przez debuger Visual Studio

Aplikacja działająca z podłączonym debugerem zachowuje się normalnie do momentu wystąpienia *przerwania*, w którym działanie aplikacji jest zatrzymywane i sterowanie jest przekazywane do debugera. W tym momencie możemy przeglądać i modyfikować stan aplikacji. Przerwania pojawiają się z dwóch głównych powodów: gdy zostanie napotkany punkt przerwania lub gdy wystąpi nieobsłużony wyjątek. Przykłady przedstawię w kolejnych punktach.

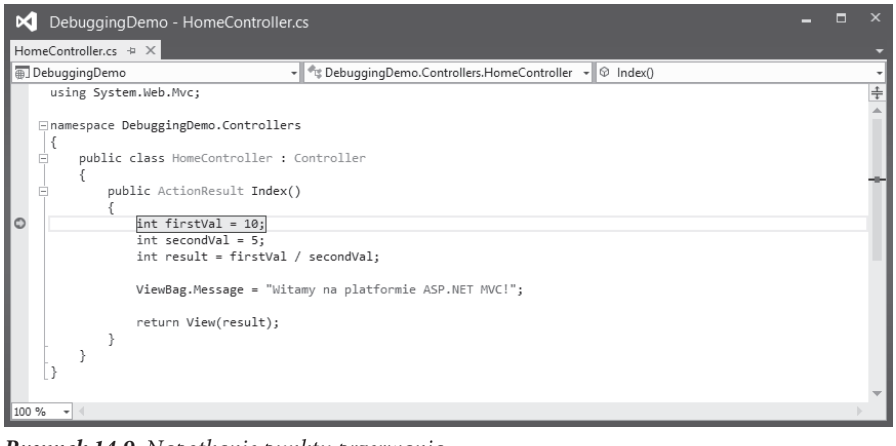
Użycie punktów przerwania

Punkt przerwania to instrukcja informująca debuger o konieczności zatrzymania wykonywania aplikacji i przekazania kontroli programiście. W tym momencie możemy przeglądać stan aplikacji, sprawdzać, co się w niej dzieje i — opcjonalnie — wznowić działanie aplikacji.

Aby utworzyć punkt przerwania, kliknij prawym przyciskiem myszy kod i wybierz opcję *Punkt przerwania/Insert Breakpoint* z menu kontekstowego. W celu zademonstrowania działania punktów przerwania umieszczamy jeden taki punkt w pierwszym poleceniu metody akcji `Index`, w klasie `Home`. Na marginesie edytora tekstów pojawi się czerwona kropka (rysunek 14.8).



Rysunek 14.8. *Dodawanie punktu przerwania w pierwszym poleceniu metody akcji Index*



Rysunek 14.9. *Napotkanie punktu przerwania*

Aby zobaczyć efekt dodania punktu przerwania, musisz uruchomić debuger poprzez wybranie opcji *Start Debugging* z menu *Debuguj* w Visual Studio. Aplikacja będzie działała aż do chwili dotarcia do polecenia oznaczonego punktem przerwania. W tym momencie debuger przerwie działanie aplikacji i przekaże kontrolę programiście. Jak pokazano na rysunku 14.9, Visual Studio podświetla wiersz kodu, w którym nastąpiło zatrzymanie działania aplikacji.

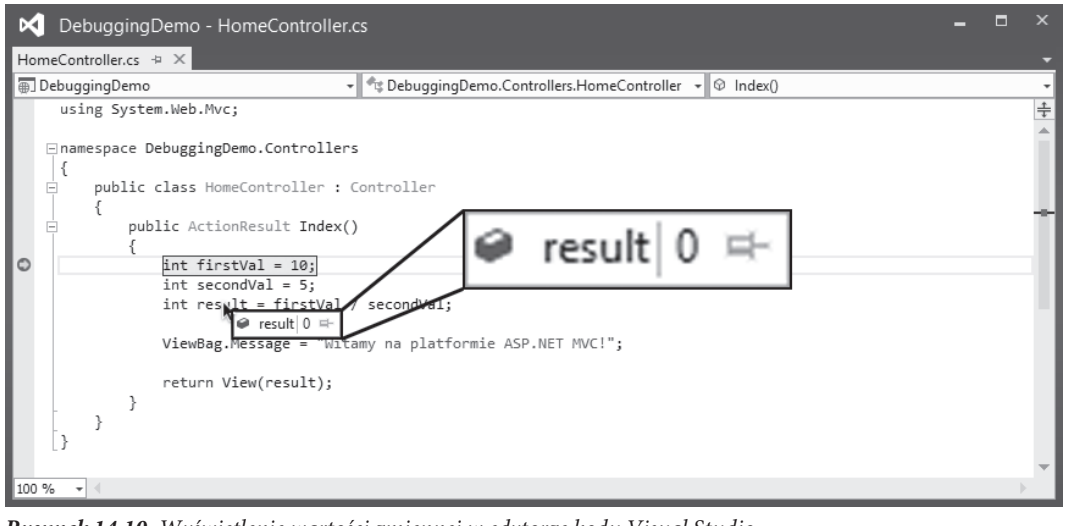
> **Uwaga** Punkt przerwania działa tylko w momencie, gdy skojarzona z nim instrukcja jest wykonywana. W naszym przykładzie punkt przerwania jest osiągany od razu po uruchomieniu aplikacji, ponieważ znajduje się w metodzie akcji wywoływanej w chwili otrzymania żądania domyślnego adresu URL. Jeżeli umieścisz punkt przerwania wewnątrz innej metody akcji, musisz użyć przeglądarki do wywołania adresu URL skojarzonego z tą metodą. Może to oznaczać, że konieczne będzie skorzystanie z aplikacji w taki sposób, w jaki korzystają z niej użytkownicy, lub bezpośrednie przejście do adresu URL w oknie przeglądarki.

Po przejęciu kontroli nad wykonywaniem aplikacji możesz przejść do kolejnego polecenia, śledzić wykonywanie w innych metodach i ogólnie przeglądać stan aplikacji. Do tego celu wykorzystujesz przyciski znajdujące się na pasku narzędzi w Visual Studio bądź opcje dostępne w menu *Debuguj*. Oprócz kontroli nad wykonywaniem aplikacji, Visual Studio dostarcza Ci także wielu użytecznych informacji dotyczących stanu aplikacji. W rzeczywistości wspomnianych informacji jest tak wiele, że w niniejszej książce nie ma wystarczająco dużo miejsca na przedstawienie czegokolwiek więcej poza podstawami.

Przeglądanie wartości danych w edytorze kodu

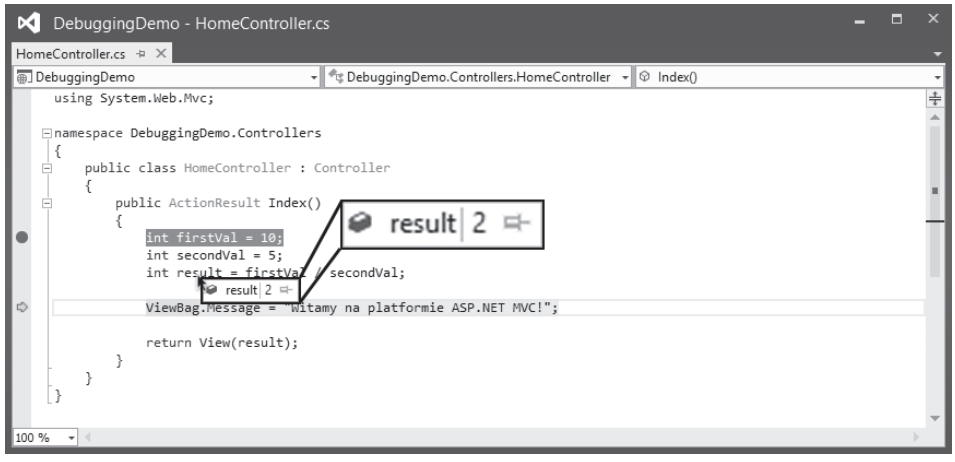
Najczęstszym sposobem użycia punktów przerwania jest próba znalezienia błędów w kodzie. Zanim będziesz mógł usunąć błąd z kodu, najpierw musisz ustalić, co tak naprawdę się dzieje. Jedną z najbardziej użytecznych funkcji oferowanych przez Visual Studio jest możliwość przeglądania i monitorowania wartości zmiennych bezpośrednio w edytorze kodu.

Przykładowo, uruchom aplikację w debugerze i zaczekaj na zatrzymanie działania aplikacji po dotarciu do dodanego wcześniej punktu przerwania. Kiedy debuger zatrzyma działanie aplikacji, umieść kursor myszy nad poleceniem definiującym zmienną `result`. Po chwili na ekranie zobaczysz małe wyskakujące okno przedstawiające bieżącą wartość wspomnianej zmiennej (rysunek 14.10). Ponieważ wspomniane okno jest małe, na rysunku pokazano także jego powiększoną wersję.



Rysunek 14.10. *Wyświetlenie wartości zmiennej w edytorze kodu Visual Studio*

Wykonywanie poleceń w metodzie akcji `Index` nie dotarło do miejsca, w którym następuje przypisanie wartości zmiennej `result`, więc Visual Studio pokazuje wartość domyślną wymienionej zmiennej, czyli `0` dla typu `int`. Wybieraj opcję *Step Over* w menu *Debuguj* (lub naciskaj klawisz *F10*) dopóty, dopóki nie dotrzesz do polecenia, w którym następuje zdefiniowanie właściwości `ViewBag.Message`. Teraz ponownie umieść kursor myszy nad zmienną `result`. Po wykonaniu polecenia przypisującego wartość zmiennej `result` wynik wykonania tego polecenia możesz zobaczyć na rysunku 14.11.



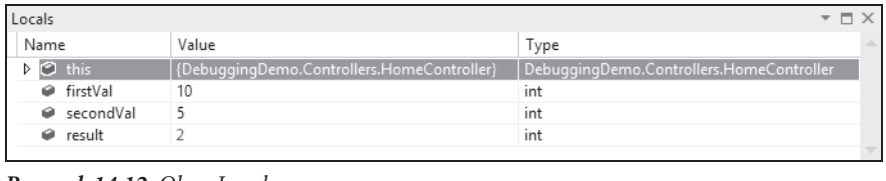
Rysunek 14.11. *Efekt przypisania wartości zmiennej*

Funkcji tej używamy w celu rozpoczęcia procesu wyszukiwania błędu, ponieważ daje ona natychmiastowy wgląd do tego, co się dzieje w aplikacji. Omawiana funkcja okazuje się szczególnie użyteczna w wykrywaniu wartości `null` oznaczających, że zmiennej nie została przypisana wartość (to źródło wielu błędów, jak wynika z mojego doświadczenia).

W wyświetlonym oknie, po prawej stronie wartości, możesz dostrzec ikonę pinezki. Jeżeli ją klikniesz, dane okno na stałe pozostanie wyświetlone na ekranie i będzie wskazywało zmianę wartości zmiennej. W ten sposób możesz monitorować jedną lub więcej zmiennych i natychmiast dowiadywać się o zmianie ich wartości i poznawać nowo przypisane wartości.

Przegląd stanu aplikacji w oknie debugera

Visual Studio zawiera wiele różnych okien, które można wykorzystać do pobierania informacji o aplikacji, gdy jej działanie zostało zatrzymane w punkcie przerwania. Pełna lista okien jest dostępna w menu *Debuguj/Okna*, ale dwa najważniejsze z nich to *Locals* i *Call Stack*. W oknie *Locals* automatycznie są wyświetlane wartości wszystkich zmiennych w aktualnym zasięgu, co pokazano na rysunku 14.12. W ten sposób otrzymujesz pojedynczy widok zawierający wszystkie zmienne, którymi możesz być zainteresowany.



Rysunek 14.12. *Okno Locals*

Zmienne, których wartości uległy zmianie w trakcie poprzednio wykonanego polecenia, są wyświetlone w kolorze czerwonym. Na rysunku widać, że wartość zmiennej `result` jest wyświetlona na czerwono, ponieważ w poprzednim poleceniu nastąpiło przypisanie jej wartości.

■ **Wskazówka** Zestaw zmiennych wyświetlonych w oknie *Locals* ulega zmianie wraz z poruszaniem się po aplikacji. Jeżeli chcesz globalnie śledzić wartość wybranej zmiennej, kliknij ją prawym przyciskiem myszy, a następnie z menu kontekstowego wybierz opcję *Add Watch*. Elementy w oknie *Watch* nie ulegają zmianie podczas wykonywania kolejnych poleceń w aplikacji i tym samym masz doskonałe miejsce na ich śledzenie.

W oknie *Call Stack* jest wyświetlana sekwencja wywołań, które doprowadziły do aktualnego stanu aplikacji. To może być bardzo użyteczne, jeśli próbujesz znaleźć powód dziwnego zachowania aplikacji — możesz wówczas przejrzeć stos wywołań i poznać przyczyny, które doprowadziły do wywołania danego punktu przerwania. (Na rysunku nie pokazano okna *Call Stack*, ponieważ w omawianej prostej aplikacji nie wystąpiło wystarczająco dużo wywołań, aby zapewnić użyteczny wgląd w nie. Zachęcam Cię do zapoznania się z omawianymi oraz pozostałymi oknami w Visual Studio, aby w ten sposób dowiedzieć się, jakie informacje możesz uzyskać z debugera).

■ **Wskazówka** Możliwe jest debugowanie widoków przez wstawianie do nich punktów przerwania. Może to być bardzo pomocne w kontrolowaniu wartości właściwości modelu widoku. Aby dodać punkt przerwania do widoku, należy wykonać taką samą operację jak w przypadku pliku kodu — kliknąć prawym przyciskiem myszy interesującą nas instrukcję Razor i wybrać *Punkt przerwania*|*Insert Breakpoint*.

Przerywanie aplikacji przez wyjątki

Nieobsłużone wyjątki są smutnym faktem. Jednym z powodów wykonywania wielu testów jednostkowych i integracyjnych jest minimalizacja prawdopodobieństwa wystąpienia takiego wyjątku w środowisku produkcyjnym. Debuger Visual Studio jest uruchamiany automatycznie w przypadku pojawienia się nieobsłużonego wyjątku.

■ **Uwaga** Jedynie *nieobsłużone* wyjątki powodują wywołanie debugera. Wyjątek staje się *obsłużony*, gdy przechwycimy go w bloku `try ... catch`. Obsłużone wyjątki są użytecznym narzędziem programistycznym. Są one wykorzystywane do obsługiwania scenariuszy, w których metoda nie jest w stanie dokończyć zadania i musimy poinformować o tym wywołującego. Nieobsłużone wyjątki są mankamentem, ponieważ reprezentują nieoczekiwane warunki w aplikacji (i powodują wyświetlenie użytkownikowi informacji o błędzie).

Aby zademonstrować przerwanie pracy aplikacji w przypadku wyjątku, do naszej metody akcji `Index` wprowadzimy małą zmianę pokazaną na listingu 14.5.

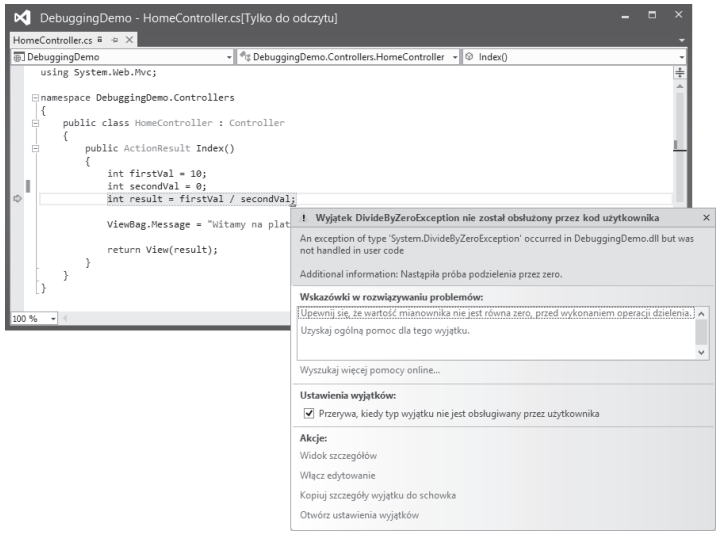
Listing 14.5. *Dodatkowe polecenie w pliku HomeController.cs powodujące wystąpienie wyjątku*

```
using System.Web.Mvc;

namespace DebuggingDemo.Controllers {
    public class HomeController : Controller {

        public ActionResult Index() {
            int firstVal = 10;
            int secondVal = 0;
            int result = firstVal / secondVal;

            ViewBag.Message = "Witamy w ASP.NET MVC!";

            return View(result);
        }
    }
}
```

Zmieniliśmy wartość zmiennej `secondVal` na `0`, co spowoduje wyjątek w instrukcji, w której `firstVal` jest dzielona przez `secondVal`.

■ **Uwaga** Z metody akcji `Index` usunięto także punkt przerwania poprzez jego kliknięcie prawym przyciskiem myszy i wybranie opcji *Delete Breakpoint* z wyświetlonego menu kontekstowego.

Jeżeli uruchomisz debuger, aplikacja będzie działała do momentu zgłoszenia wyjątku, gdy pojawi się okno informacji o wyjątku pokazane na rysunku 14.13.



Rysunek 14.13. *Okno pomocnicze obsługi wyjątku*

W tym oknie pomocniczym znajdują się informacje na temat wyjątku. Gdy debuger zostanie wywołany w wierszu powodującym wyjątek, możemy skontrolować stan aplikacji i sterować jej działaniem, podobnie jak w przypadku punktu przerwania.

Użycie opcji Edit and Continue

Jedną z najbardziej interesujących funkcji debugera Visual Studio jest *Edit and Continue*. Gdy zostanie wywołany debuger, można zmodyfikować kod, a następnie kontynuować debugowanie. Visual Studio ponownie skompiluje aplikację, po czym odtworzy jej stan w momencie aktywowania debugera.

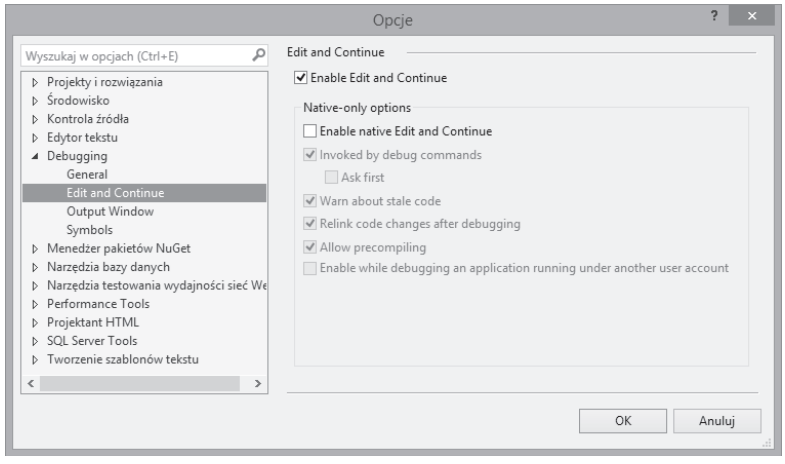
Włączanie opcji Edit and Continue

Konieczne jest włączenie opcji *Edit and Continue* w dwóch miejscach:

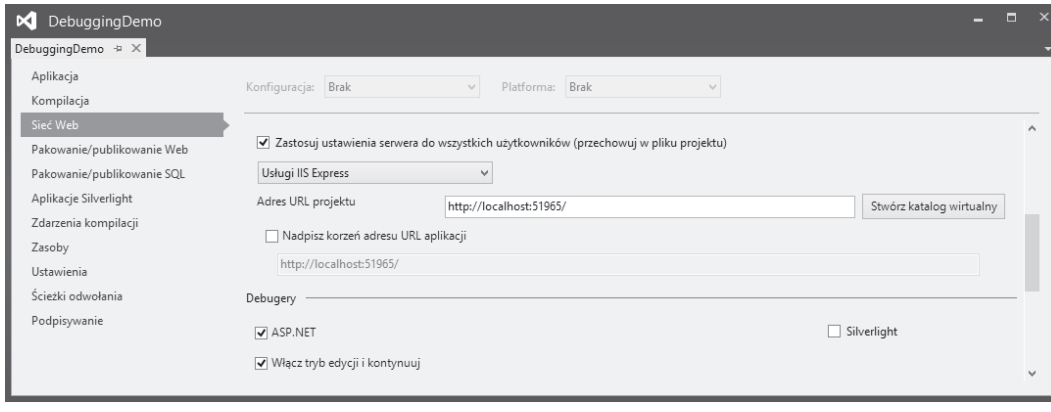
- Upewnij się, że w sekcji *Edit and Continue* dla opcji *Debugging* zaznaczona jest opcja *Enable Edit and Continue* (wybierz *Opcje…* z menu *Narzędzia*), jak pokazano na rysunku 14.14.
- We właściwościach projektu (wybierz *Właściwości DebuggingDemo…* z menu *Projekt*) przejdź do sekcji *Sieć Web* i upewnij się, że zaznaczona jest opcja *Włącz tryb edycji i kontynuuj* (rysunek 14.15).

Modyfikowanie projektu

Funkcja *Edit and Continue* jest nieco kapryśna. Istnieją przypadki, w których nie będzie ona działać. Jeden z nich jest pokazany dla metody `Index` z klasy `HomeController` — użyte są w niej obiekty dynamiczne. Rozwiązaniem problemu jest umieszczenie znaków komentarza na początku wiersza, w którym korzystamy z funkcji `ViewBag`, w pliku *HomeController.cs*, jak przedstawiono na listingu 14.6.



Rysunek 14.14. *Włączenie opcji Edit and Continue w oknie dialogowym Opcje*



Rysunek 14.15. *Włączanie trybu edycji i kontynuacji we właściwościach projektu*

Listing 14.6. *Usunięcie wywołania ViewBag z metody Index w pliku HomeController.cs*

```
using System.Web.Mvc;

namespace DebuggingDemo.Controllers {
    public class HomeController : Controller {

        public ActionResult Index() {

            int firstVal = 10;
            int secondVal = 0;
            int result = firstVal / secondVal;

            // poniższe polecenie zostało poprzedzone znakiem komentarza
            // ViewBag.Message = "Witamy w ASP.NET MVC!";

            return View(result);
        }
    }
}
```

Analogiczną zmianę musimy wykonać w widoku *Index.cshtml*, co jest pokazane na listingu 14.7.

Listing 14.7. *Usunięcie wywołania ViewBag z widoku*

```
@model int

@{
    Layout = null;
}

<!DOCTYPE html>

<html>
<head>
    <meta name="viewport" content="width=device-width" />
    <link href="~/Content/Site.css" rel="stylesheet" type="text/css" />
    <title>Index</title>
</head>
<body>
    <!-- Poniższy element został umieszczony w komentarzu. -->
    <!--<h2 class="message">@ViewData["Message"]</h2>-->
    <p>
        Wartość obliczeń to: @Model
    </p>
</body>
</html>
```

Edycja i kontynuowanie pracy

Jesteśmy już gotowi do użycia funkcji *Edit and Continue*. Zaczniemy od wybrania opcji *Start Debugging* z menu *Debuguj*. Aplikacja uruchomi się z dołączonym debugerem i będzie realizowała metodę `Index` do momentu wykonania wiersza, w którym przeprowadzamy obliczenia. Wartość drugiego parametru wynosi zero, co spowoduje zgłoszenie wyjątku. W tym momencie debuger przerwie działanie i wyświetli się okno informacji o wyjątku (jak pokazano na wcześniejszym rysunku 14.13).

Kliknij łącze *Włącz edytowanie* w oknie wyjątku. W edytorze kodu zmień wyrażenie obliczające wartość zmiennej `result` na następujące:

```
...
int result = firstVal / 2;
...
```

Usunęliśmy odwołanie do zmiennej `secondVal` i zastąpiliśmy je wartością 2. Z menu *Debuguj* wybierz *Continue*. Aplikacja będzie kontynuowała działanie. Nowa wartość przypisana zmiennej zostanie użyta do wygenerowania wyniku zmiennej `result`, a przeglądarka wyświetli stronę, zamieszczoną na rysunku 14.16.

Rysunek 14.16. *Efekt usunięcia błędu dzięki użyciu funkcji Edit and Continue*

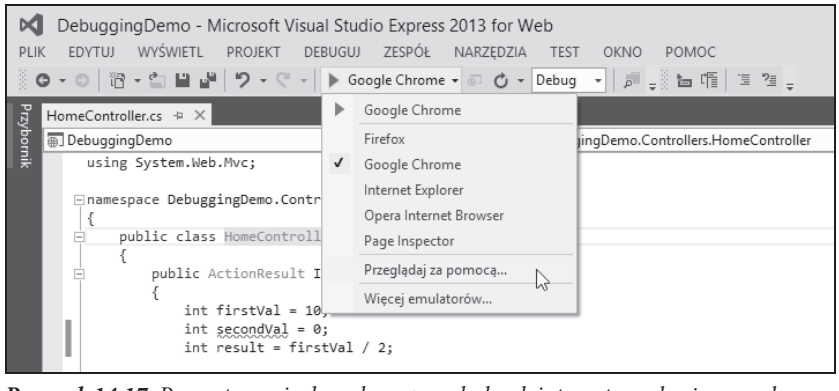
Poświęć chwilę na analizę wyniku tych działań. Uruchomiliśmy aplikację zawierającą błąd — próbę dzielenia przez zero. Debuger wykrył wyjątek i zatrzymał wykonywanie programu. Aby poprawić błąd, zmodyfikowaliśmy kod, zamieniając odwołanie do zmiennej na literał o wartości 5. Następnie wznowiliśmy działanie debugera. W tym momencie aplikacja została ponownie skompilowana przez Visual Studio, dzięki czemu nasza zmiana jest

uwzględniona w procesie kompilacji, stan aplikacji jest przywrócony, a następnie kontynuowany w normalny sposób. Przeglądarka otrzyma wygenerowany wynik uwzględniający naszą poprawkę. Bez opcji *Edit and Continue* musielibyśmy zatrzymać aplikację, wprowadzić zmiany, skompilować aplikację i ponownie uruchomić debuger. Następnie musielibyśmy powtórzyć w przeglądarce kroki, które doprowadziły do momentu wywołania debugera. Uniknięcie tego ostatniego kroku jest tu najważniejsze. Odtworzenie skomplikowanych błędów może wymagać wykonania wielu operacji w aplikacji, a możliwość testowania potencjalnych rozwiązań bez potrzeby powtarzania tych kroków pozwala zaoszczędzić czas i nerwy programisty.

Użycie funkcji połączonych przeglądarek

Visual Studio 2013 zawiera funkcję o nazwie *połączone przeglądarki*, która pozwala na jednoczesne wyświetlanie aplikacji w wielu przeglądarkach internetowych i ich odświeżanie po wprowadzeniu zmiany. Ta funkcja okazuje się najbardziej użyteczna, gdy działanie aplikacji jest stabilne i pozostało już tylko dopracowanie kodu HTML i CSS generowanego przez widoki (wkrótce to wyjaśnię).

W celu użycia funkcji *połączonych przeglądarek* na pasku narzędzi w Visual Studio kliknij mały trójkąt skierowany w dół obok nazwy wybranej przeglądarki internetowej, a następnie z menu wybierz opcję *Przeglądaj za pomocą…*, jak pokazano na rysunku 14.17.

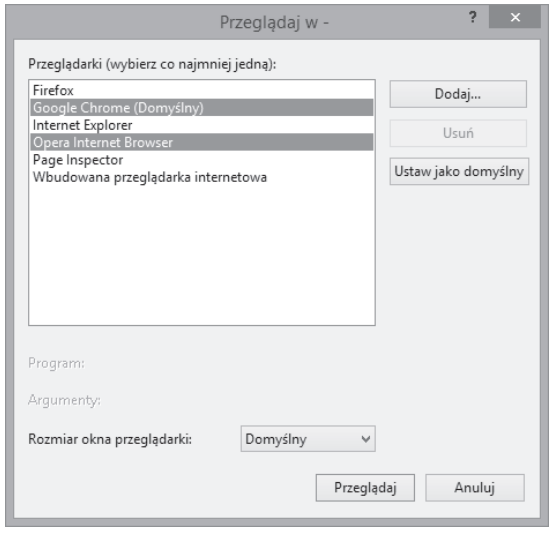


Rysunek 14.17. *Przygotowanie do wyboru przeglądarek internetowych używanych wraz z funkcją Browser Link*

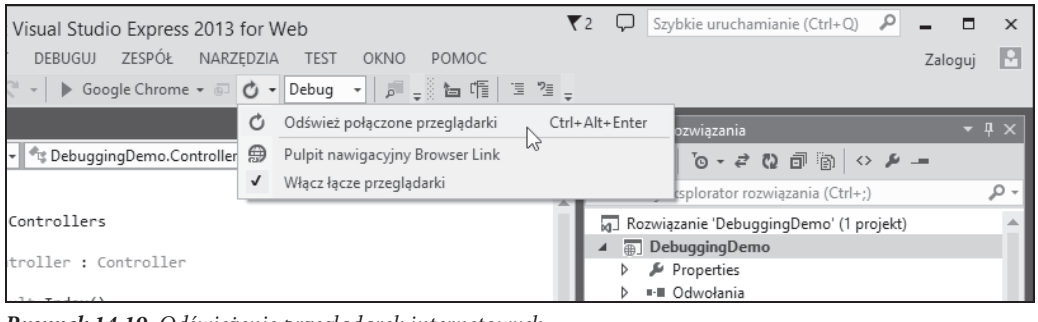
Na ekranie zostanie wyświetlone okno dialogowe *Przeglądaj w*. Naciśnij klawisz *Control*, a następnie zaznacz przeglądarki internetowe, których chcesz używać. Na rysunku 14.18 widać, że wybrałem Google Chrome i Opera Internet Browser. Za pomocą tego okna dialogowego możesz również dodać nowe przeglądarki, choć Visual Studio całkiem dobrze radzi sobie z wykrywaniem większości najważniejszych przeglądarek internetowych.

Po kliknięciu przycisku *Przeglądaj* Visual Studio uruchomi wybrane przeglądarki internetowe i wczyta aktualny projekt w każdej z nich. Teraz możesz przeprowadzić edycję kodu w aplikacji, a następnie uaktualnić wszystkie okna przeglądarek internetowych, wybierając opcję *Odśwież połączone przeglądarki* z paska narzędzi w Visual Studio, jak pokazano na rysunku 14.19. Aplikacja zostanie automatycznie skompilowana i będziesz mógł zobaczyć wprowadzone zmiany.

Omawiana funkcja działa przez wysłanie przeglądarce internetowej pewnego kodu JavaScript w dokumencie HTML i zapewnia elegancki sposób programowania iteracyjnego. Zalecam jej stosowanie jedynie podczas pracy z widokami, ponieważ wtedy istnieje najmniejsze prawdopodobieństwo, że serwer IIS wyśle przeglądarce internetowej komunikaty błędów HTTP. Wspomniane komunikaty są generowane, gdy w kodzie występuje błąd. Kod JavaScript nie jest dodawany do odpowiedzi dotyczących błędów, a tym samym następuje utrata połączenia między Visual Studio i przeglądarkami internetowymi. W takim przypadku trzeba ponownie przejść do okna dialogowego *Przeglądaj w*. Funkcja połączonych przeglądarek jest użyteczna, ale użycie kodu JavaScript okazuje się problemem. Podczas pracy nad projektami w innych technologiach niż ASP.NET korzystam z podobnego narzędzia, o nazwie LiveReload (*http://livereload.com/*). Wymienione narzędzie



Rysunek 14.18. *Wybór dwóch przeglądarek internetowych*



Rysunek 14.19. *Odświeżenie przeglądarek internetowych*

oferuje lepsze podejście, ponieważ jego działanie opiera się na wtyczkach przeglądarek internetowych, na które komunikaty błędów HTTP nie mają wpływu. Wartość funkcji połączonych przeglądarek w Visual Studio będzie ograniczona, dopóki Microsoft nie zastosuje podobnego rozwiązania.

Podsumowanie

W rozdziale tym omówiłem strukturę projektu Visual Studio MVC i wyjaśniłem, jak są połączone ze sobą jej części. Wskazałem również jedną z najważniejszych cech platformy MVC — możliwość stosowania konwencji. Do omówionych tematów będę stale wracać w kolejnych rozdziałach, przedstawiając sposób działania platformy MVC.

Skorowidz

C

D

E

F

G

H

I

J

K

L

Ł

M

N

O

P

R

S

Ś

T

U

V

W

Z

Ź

Ż

LEKTURA OBOWIĄZKOWA KAŻDEGO PROGRAMISTY!

ASP.NET MVC 5
Zaawansowane programowanie

Platforma ASP.NET MVC powstała jako alternatywa dla ASP.NET Web Forms. Dzięki zastosowaniu architektury model-widok-kontroler (ang. Model-View-Controller) tworzony kod jest przejrzysty oraz zdecydowanie łatwiejszy w utrzymaniu. Ciągły rozwój tej platformy zaowocował wydaniem kolejnej wersji, oznaczonej numerem 5. Dzięki zawartym tu nowościom stworzysz jeszcze lepszy kod w krótszym czasie. Przekonaj się sam!

Niniejsze wydanie tej cenionej przez programistów książki zostało rozszerzone o nowości wprowadzone na platformie ASP.NET MVC 5. Wśród nich znajdziesz opisy sposobu definiowania tras za pomocą atrybutów C# oraz metody nadpisywania filtrów. Przeczytasz tu również o bibliotekach Bootstrap oraz poznasz możliwości zintegrowanego środowiska programistycznego Visual Studio 2013. Ponadto dowiesz się, jak zabezpieczać poszczególne obszary aplikacji, w jaki sposób korzystać z filtrów lub routingu oraz jak obsługiwać żądania AJAX. Ta książka jest kompletnym przewodnikiem po ASP.NET MVC 5. Musisz ją mieć!

Dzięki tej książce:

- poznasz nowości wprowadzone w wersji 5 platformy ASP.NET MVC
- zgłębisz architekturę MVC i z jej pomocą zbudujesz aplikację
- zaznajomisz się z nową wersją środowiska programistycznego Visual Studio 2013
- zdobędziesz dogłębną wiedzę o ASP.NET MVC 5

Adam Freeman — specjalista z branży IT o ogromnym doświadczeniu. Przez wiele lat zajmował stanowiska kierownicze. Obecnie jest dyrektorem ds. technologii oraz dyrektorem naczelnym w międzynarodowym banku. Pisze książki i jest zapalonym biegaczem.

Helion

32734 numer katalogowy

księgarnia internetowa

http://helion.pl

zamówienia telefoniczne

0 801 339900

0 601 339900

Informatyka w najlepszym wydaniu

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

sięgnij po WIĘCEJ

KOD KORZYŚCI

ISBN 978-83-283-0651-6

9 788328 306516

cena: 119,00 zł

Apress