

ASP.NET MVC 5

Zaawansowane
programowanie

Adam Freeman

Tytuł oryginału: Pro ASP.NET MVC 5

Tłumaczenie: Robert Górczyński

ISBN: 978-83-283-0651-6

Original edition copyright © 2013 by Adam Freeman.
All rights reserved.

Polish edition copyright © 2015 by HELION SA.
All rights reserved.

All rights reserved. No part of this book may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying, recording or by any information storage retrieval system, without permission from the Publisher.

Wszelkie prawa zastrzeżone. Nieautoryzowane rozpowszechnianie całości lub fragmentu niniejszej publikacji w jakiegokolwiek postaci jest zabronione. Wykonywanie kopii metodą kserograficzną, fotograficzną, a także kopiowanie książki na nośniku filmowym, magnetycznym lub innym powoduje naruszenie praw autorskich niniejszej publikacji.

Wszystkie znaki występujące w tekście są zastrzeżonymi znakami firmowymi bądź towarowymi ich właścicieli.

Autor oraz Wydawnictwo HELION dołożyli wszelkich starań, by zawarte w tej książce informacje były kompletne i rzetelne. Nie biorą jednak żadnej odpowiedzialności ani za ich wykorzystanie, ani za związane z tym ewentualne naruszenie praw patentowych lub autorskich. Autor oraz Wydawnictwo HELION nie ponoszą również żadnej odpowiedzialności za ewentualne szkody wynikłe z wykorzystania informacji zawartych w książce.

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.

- [Kup książkę](#)
- [Poleć książkę](#)
- [Oceń książkę](#)

- [Księgarnia internetowa](#)
- [Lubię to! » Nasza społeczność](#)

Spis treści

	O autorze	17
	O recenzencie technicznym	18
Rozdział 1.	ASP.NET MVC w szerszym kontekście	19
	Krótką historia programowania witryn WWW	19
	Co poszło nie tak z ASP.NET Web Forms?	20
	Programowanie witryn WWW — stan obecny	21
	Standardy sieciowe oraz REST	21
	Programowanie zwinne i sterowane testami	22
	Ruby on Rails	22
	Node.js	22
	Najważniejsze zalety ASP.NET MVC	23
	Architektura MVC	23
	Rozszerzalność	24
	Ścisła kontrola nad HTML i HTTP	24
	Łatwość testowania	24
	Zaawansowany system routingu	25
	Zbudowany na najlepszych częściach platformy ASP.NET	25
	Nowoczesne API	26
	ASP.NET MVC jest open source	26
	Co powinienem wiedzieć?	26
	Jaka jest struktura książki?	27
	Część I. Wprowadzenie do ASP.NET MVC 5	27
	Część II. Szczegółowe omówienie platformy ASP.NET MVC	27
	Co nowego w ASP.NET MVC 5?	27
	Gdzie znajdę przykładowe fragmenty kodu?	28
	Jakiego oprogramowania będę potrzebował?	28
	Bootstrap	29
	Podsumowanie	29

Rozdział 2.	Pierwsza aplikacja MVC	31
	Przygotowanie Visual Studio	31
	Tworzenie nowego projektu ASP.NET MVC	31
	Dodawanie pierwszego kontrolera	34
	Poznajemy trasy	37
	Generowanie stron WWW	37
	Tworzenie i generowanie widoku	37
	Dynamiczne dodawanie treści	41
	Tworzenie prostej aplikacji wprowadzania danych	42
	Przygotowanie sceny	42
	Projektowanie modelu danych	43
	Łączenie metod akcji	44
	Budowanie formularza	47
	Zdefiniowanie początkowego adresu URL	49
	Obsługa formularzy	50
	Dodanie kontroli poprawności	53
	Nadanie stylu zawartości	58
	Kończymy przykład	63
	Podsumowanie	64
Rozdział 3.	Wzorzec MVC	65
	Historia MVC	65
	Wprowadzenie do wzorca MVC	66
	Budowa modelu domeny	66
	Implementacja MVC w ASP.NET	67
	Porównanie MVC z innymi wzorcami	67
	Budowanie luźno połączonych komponentów	70
	Wykorzystanie wstrzykiwania zależności	71
	Użycie kontenera wstrzykiwania zależności	72
	Zaczynamy testy automatyczne	74
	Zadania testów jednostkowych	74
	Zadania testów integracyjnych	79
	Podsumowanie	79
Rozdział 4.	Najważniejsze cechy języka	81
	Utworzenie przykładowego projektu	81
	Dodanie podzespołu System.Net.Http	83
	Użycie automatycznie implementowanych właściwości	83
	Użycie inicjalizatorów obiektów i kolekcji	86
	Użycie metod rozszerzających	88
	Stosowanie metod rozszerzających do interfejsów	90
	Tworzenie filtrujących metod rozszerzających	92
	Użycie wyrażeń lambda	93
	Automatyczna inferencja typów	97
	Użycie typów anonimowych	97
	Wykonywanie zapytań LINQ	98
	Opóźnione zapytania LINQ	102
	Użycie metod asynchronicznych	103
	Użycie słów kluczowych async i await	105
	Podsumowanie	106

Rozdział 5. Praca z silnikiem Razor	107
Utworzenie przykładowego projektu	107
Definiowanie modelu	108
Definiowanie kontrolera	108
Tworzenie widoku	109
Korzystanie z obiektów modelu	109
Praca z układami	111
Tworzenie układu	112
Stosowanie układu	113
Użycie pliku ViewStart	114
Użycie układów współdzielonych	115
Użycie wyrażeń Razor	118
Wstawianie wartości danych	119
Przypisanie wartości atrybutu	121
Użycie konstrukcji warunkowych	123
Wyświetlanie zawartości tablic i kolekcji	125
Praca z przestrzenią nazw	127
Podsumowanie	128
Rozdział 6. Ważne narzędzia wspierające MVC	129
Utworzenie przykładowego projektu	130
Utworzenie klas modelu	130
Dodanie kontrolera	132
Dodanie widoku	132
Użycie Ninject	133
Zrozumienie problemu	133
Dodawanie Ninject do projektu Visual Studio	135
Zaczynamy korzystać z Ninject	136
Konfiguracja wstrzykiwania zależności na platformie MVC	137
Tworzenie łańcucha zależności	140
Definiowanie wartości właściwości i parametrów konstruktora	142
Użycie łączenia warunkowego	143
Ustawienie obiektu zakresu	144
Testy jednostkowe w Visual Studio	147
Tworzenie projektu testów jednostkowych	147
Tworzenie testów jednostkowych	148
Uruchamianie testów (nieudane)	152
Implementacja funkcji	152
Testowanie i poprawianie kodu	153
Użycie Moq	155
Zrozumienie problemu	155
Dodawanie Moq do projektu Visual Studio	157
Dodanie obiektu imitacyjnego do testu jednostkowego	157
Tworzenie bardziej skomplikowanych obiektów Mock	160
Podsumowanie	162

Rozdział 7. SportsStore — kompletna aplikacja	163
Zaczynamy	164
Tworzenie rozwiązania i projektów w Visual Studio	164
Instalacja pakietów narzędziowych	166
Dodawanie odwołań między projektami	166
Konfigurowanie kontenera DI	167
Uruchamiamy aplikację	168
Tworzenie modelu domeny	168
Tworzenie abstrakcyjnego repozytorium	169
Tworzenie imitacji repozytorium	169
Wyświetlanie listy produktów	171
Dodawanie kontrolera	171
Dodawanie układu, pliku ViewStart i widoku	172
Konfigurowanie domyślnej trasy	173
Uruchamianie aplikacji	174
Przygotowanie bazy danych	175
Tworzenie bazy danych	176
Definiowanie schematu bazy danych	177
Dodawanie danych do bazy	179
Tworzenie kontekstu Entity Framework	180
Tworzenie repozytorium produktów	182
Dodanie stronicowania	184
Wyświetlanie łączy stron	185
Ulepszanie adresów URL	193
Dodawanie stylu	194
Instalacja pakietu Bootstrap	194
Zastosowanie w aplikacji stylów Bootstrap	195
Tworzenie widoku częściowego	196
Podsumowanie	199
Rozdział 8. SportsStore — nawigacja	201
Dodawanie kontrolki nawigacji	201
Filtrowanie listy produktów	201
Ulepszanie schematu URL	205
Budowanie menu nawigacji po kategoriach	208
Poprawianie licznika stron	213
Budowanie koszyka na zakupy	216
Definiowanie encji koszyka	217
Tworzenie przycisków koszyka	221
Implementowanie kontrolera koszyka	222
Wyświetlanie zawartości koszyka	223
Podsumowanie	226
Rozdział 9. SportsStore — ukończenie koszyka na zakupy	227
Użycie dołączania danych	227
Tworzenie własnego łącznika modelu	227
Kończenie budowania koszyka	231
Usuwanie produktów z koszyka	232
Dodawanie podsumowania koszyka	233

Składanie zamówień	236
Rozszerzanie modelu domeny	236
Dodawanie procesu składania zamówienia	236
Implementowanie mechanizmu przetwarzania zamówień	242
Rejestrowanie implementacji	244
Zakończenie pracy nad kontrolerem koszyka	246
Wyświetlanie informacji o błędach systemu kontroli poprawności	249
Wyświetlanie strony podsumowania	251
Podsumowanie	252
Rozdział 10. SportsStore — wersja mobilna	253
Kontekst programowania sieciowego dla urządzeń mobilnych	253
Odstępianie od działania (lub jego podjęcie na minimalnym możliwym poziomie)	254
Użycie układu responsywnego	255
Utworzenie responsywnego nagłówka	256
Tworzenie responsywnej listy produktów	260
Utworzenie zawartości specjalnie dla urządzeń mobilnych	267
Utworzenie układu dla urządzeń mobilnych	268
Utworzenie widoków dla urządzeń mobilnych	269
Podsumowanie	272
Rozdział 11. SportsStore — administracja	275
Dodajemy zarządzanie katalogiem	275
Tworzenie kontrolera CRUD	276
Tworzenie nowego pliku układu	277
Implementowanie widoku listy	278
Edycja produktów	282
Tworzenie nowych produktów	295
Usuwanie produktów	298
Podsumowanie	301
Rozdział 12. SportsStore — bezpieczeństwo i ostatnie usprawnienia	303
Zabezpieczanie kontrolera administracyjnego	303
Zdefiniowanie prostej polityki bezpieczeństwa	303
Realizacja uwierzytelniania z użyciem filtrów	305
Tworzenie dostawcy uwierzytelniania	306
Tworzenie kontrolera AccountController	308
Tworzenie widoku	309
Przesyłanie zdjęć	312
Rozszerzanie bazy danych	312
Rozszerzanie modelu domeny	313
Tworzenie interfejsu użytkownika do przesyłania plików	314
Zapisywanie zdjęć do bazy danych	316
Implementowanie metody akcji GetImage	317
Wyświetlanie zdjęć produktów	321
Podsumowanie	322

Rozdział 13. Wdrażanie aplikacji	323
Przygotowanie do użycia Windows Azure	324
Tworzenie witryny internetowej i bazy danych	324
Przygotowanie bazy danych do zdalnej administracji	325
Tworzenie schematu bazy danych	326
Wdrażanie aplikacji	328
Podsumowanie	332
Rozdział 14. Przegląd projektu MVC	333
Korzystanie z projektów MVC z Visual Studio	333
Tworzenie projektu	334
Przedstawienie konwencji MVC	337
Debugowanie aplikacji MVC	338
Tworzenie przykładowego projektu	338
Uruchamianie debugera Visual Studio	341
Przerywanie pracy aplikacji przez debuger Visual Studio	342
Użycie opcji Edit and Continue	347
Użycie funkcji połączonych przeglądarek	350
Podsumowanie	351
Rozdział 15. Routing URL	353
Utworzenie przykładowego projektu	353
Utworzenie przykładowych kontrolerów	355
Utworzenie widoku	356
Ustawienie początkowego adresu URL i przetestowanie aplikacji	356
Wprowadzenie do wzorców URL	357
Tworzenie i rejestrowanie prostej trasy	358
Użycie prostej trasy	363
Definiowanie wartości domyślnych	363
Użycie statycznych segmentów adresu URL	366
Definiowanie własnych zmiennych segmentów	370
Użycie własnych zmiennych jako parametrów metod akcji	372
Definiowanie opcjonalnych segmentów URL	373
Definiowanie tras o zmiennej długości	375
Definiowanie priorytetów kontrolerów na podstawie przestrzeni nazw	377
Ograniczenia tras	380
Ograniczanie trasy z użyciem wyrażeń regularnych	380
Ograniczanie trasy do zbioru wartości	381
Ograniczanie tras z użyciem metod HTTP	381
Użycie ograniczeń dotyczących typu i wartości	383
Definiowanie własnych ograniczeń	385
Użycie atrybutów routingu	387
Włączanie i stosowanie atrybutów routingu	387
Tworzenie tras za pomocą zmiennych segmentu	389
Zastosowanie ograniczeń trasy	390
Użycie prefiksu trasy	392
Podsumowanie	393

Rozdział 16. Zaawansowane funkcje routingu	395
Utworzenie przykładowego projektu	396
Uproszczenie tras	396
Dodanie pakietu optymalizacyjnego	396
Uaktualnienie projektu testów jednostkowych	397
Generowanie wychodzących adresów URL w widokach	397
Użycie systemu routingu do wygenerowania wychodzącego adresu URL	397
Użycie innych kontrolerów	400
Przekazywanie dodatkowych parametrów	401
Definiowanie atrybutów HTML	403
Generowanie w pełni kwalifikowanych adresów URL w łączach	404
Generowanie adresów URL (nie łączy)	405
Generowanie wychodzących adresów URL w metodach akcji	406
Generowanie adresu URL na podstawie wybranej trasy	407
Dostosowanie systemu routingu	408
Tworzenie własnej implementacji RouteBase	408
Tworzenie własnego obiektu obsługi trasy	412
Korzystanie z obszarów	414
Tworzenie obszaru	414
Wypełnianie obszaru	416
Rozwiązywanie problemów z niejednoznacznością kontrolerów	417
Tworzenie obszarów za pomocą atrybutów	418
Generowanie łącz do akcji z obszarów	419
Routing żądań dla plików dyskowych	420
Konfiguracja serwera aplikacji	421
Definiowanie tras dla plików na dysku	422
Pomijanie systemu routingu	424
Najlepsze praktyki schematu adresów URL	424
Twórz jasne i przyjazne dla człowieka adresy URL	425
GET oraz POST — wybierz właściwie	426
Podsumowanie	426
Rozdział 17. Kontrolery i akcje	427
Utworzenie przykładowego projektu	428
Ustawienie początkowego adresu URL	428
Wprowadzenie do kontrolerów	428
Tworzenie kontrolera z użyciem interfejsu IController	428
Tworzenie kontrolera przez dziedziczenie po klasie Controller	430
Odczytywanie danych wejściowych	432
Pobieranie danych z obiektów kontekstu	432
Użycie parametrów metod akcji	433
Tworzenie danych wyjściowych	435
Wyniki akcji	436
Zwracanie kodu HTML przez generowanie widoku	440
Przekazywanie danych z metody akcji do widoku	443
Wykonywanie przekierowań	447
Zwracanie błędów i kodów HTTP	452
Podsumowanie	453

Rozdział 18. Filtry	455
Utworzenie przykładowego projektu	456
Ustawienie początkowego adresu URL i przetestowanie aplikacji	458
Użycie filtrów	458
Wprowadzenie do podstawowych typów filtrów	459
Dołączanie filtrów do kontrolerów i metod akcji	460
Użycie filtrów autoryzacji	461
Użycie własnego filtra autoryzacji	462
Użycie wbudowanego filtra autoryzacji	463
Użycie filtrów uwierzytelniania	464
Interfejs <code>IAutenticationFilter</code>	464
Implementacja sprawdzenia uwierzytelniania	466
Połączenie filtrów uwierzytelniania i autoryzacji	468
Obsługa ostatniego uwierzytelnienia w żądaniu	469
Użycie filtrów wyjątków	470
Tworzenie filtra wyjątku	470
Użycie filtra wyjątków	471
Użycie widoku w celu reakcji na wyjątek	474
Użycie wbudowanego filtra wyjątków	476
Użycie filtrów akcji	478
Implementacja metody <code>OnActionExecuting</code>	479
Implementacja metody <code>OnActionExecuted</code>	481
Używanie filtra wyniku	482
Użycie wbudowanych klas filtrów akcji i wyniku	483
Użycie innych funkcji filtrów	485
Filtrowanie bez użycia atrybutów	485
Użycie filtrów globalnych	487
Określanie kolejności wykonywania filtrów	489
Nadpisywanie filtrów	491
Podsumowanie	494
Rozdział 19. Rozszerzanie kontrolerów	495
Utworzenie przykładowego projektu	496
Ustawienie początkowego adresu URL	498
Tworzenie własnej fabryki kontrolerów	498
Przygotowanie kontrolera zapasowego	500
Utworzenie klasy kontrolera	500
Implementacja innych metod interfejsu	501
Rejestrowanie własnej fabryki kontrolerów	501
Wykorzystanie wbudowanej fabryki kontrolerów	502
Nadawanie priorytetów przestrzeniom nazw	502
Dostosowywanie sposobu tworzenia kontrolerów w <code>DefaultControllerFactory</code>	504
Tworzenie własnego obiektu wywołującego akcje	506
Użycie wbudowanego obiektu wywołującego akcje	508
Użycie własnych nazw akcji	508
Selekcja metod akcji	509

Poprawianie wydajności z użyciem specjalizowanych kontrolerów	515
Użycie kontrolerów bezstanowych	515
Użycie kontrolerów asynchronicznych	517
Podsumowanie	521
Rozdział 20. Widoki	523
Tworzenie własnego silnika widoku	523
Tworzenie przykładowego projektu	526
Tworzenie własnej implementacji IView	527
Tworzenie implementacji IViewEngine	528
Rejestrowanie własnego silnika widoku	529
Testowanie silnika widoku	529
Korzystanie z silnika Razor	531
Tworzenie przykładowego projektu	531
Sposób generowania widoków przez Razor	532
Konfigurowanie wyszukiwania lokalizacji widoków	533
Dodawanie dynamicznych treści do widoku Razor	536
Zastosowanie sekcji układu	536
Użycie widoków częściowych	541
Użycie akcji potomnych	544
Podsumowanie	546
Rozdział 21. Metody pomocnicze	547
Tworzenie przykładowego projektu	548
Ustawienie początkowego adresu URL	549
Przetestowanie aplikacji	549
Tworzenie własnej metody pomocniczej	549
Tworzenie wewnętrznej metody pomocniczej HTML	549
Tworzenie zewnętrznej metody pomocniczej HTML	551
Zarządzanie kodowaniem ciągów tekstowych w metodzie pomocniczej	554
Użycie wbudowanych metod pomocniczych	559
Przygotowania do obsługi formularzy	559
Określenie trasy używanej przez formularz	565
Użycie metod pomocniczych do wprowadzania danych	567
Tworzenie znaczników select	571
Podsumowanie	573
Rozdział 22. Szablone metody pomocnicze	575
Przygotowanie przykładowego projektu	576
Używanie szablonych metod pomocniczych	578
Generowanie etykiety i wyświetlanie elementów	581
Użycie szablonych metod pomocniczych dla całego modelu	583
Użycie metadanych modelu	586
Użycie metadanych do sterowania edycją i widocznością	586
Użycie metadanych dla etykiet	589
Użycie metadanych wartości danych	590
Użycie metadanych do wybierania szablonu wyświetlania	591
Dodawanie metadanych do klasy zaprzyjaźnionej	593
Korzystanie z parametrów typów złożonych	595

Dostosowywanie systemu szablonowych metod pomocniczych	596
Tworzenie własnego szablonu edytora	596
Tworzenie szablonu ogólnego	597
Zastępowanie szablonów wbudowanych	599
Podsumowanie	599
Rozdział 23. Metody pomocnicze URL i Ajax	601
Przygotowanie przykładowego projektu	602
Definiowanie dodatkowych stylów CSS	603
Instalacja pakietów NuGet	603
Tworzenie podstawowych łączy i adresów URL	603
Nieprzeszkadzający Ajax	605
Tworzenie widoku formularza synchronicznego	606
Włączanie i wyłączanie nieprzeszkadzających wywołań Ajax	607
Utworzenie nieprzeszkadzających formularzy Ajax	608
Przygotowanie kontrolera	608
Tworzenie formularza Ajax	610
Sposób działania nieprzeszkadzających wywołań Ajax	612
Ustawianie opcji Ajax	612
Zapewnienie kontrolowanej degradacji	612
Informowanie użytkownika o realizowanym żądaniu Ajax	614
Wyświetlanie pytania przed wysłaniem żądania	615
Tworzenie łączy Ajax	616
Zapewnienie kontrolowanej degradacji dla łączy	618
Korzystanie z funkcji wywołania zwrotnego w technologii Ajax	618
Wykorzystanie JSON	621
Dodanie obsługi JSON do kontrolera	621
Przetwarzanie JSON w przeglądarce	622
Przygotowanie danych do kodowania	624
Wykrywanie żądań Ajax w metodach akcji	626
Podsumowanie	628
Rozdział 24. Dołączanie modelu	629
Przygotowanie przykładowego projektu	630
Użycie dołączania modelu	632
Użycie domyślnego łącznika modelu	633
Dołączanie typów prostych	634
Dołączanie typów złożonych	636
Dołączanie tablic i kolekcji	643
Ręczne wywoływanie dołączania modelu	648
Obsługa błędów dołączania modelu	650
Dostosowanie systemu dołączania modelu	650
Tworzenie własnego dostawcy wartości	651
Tworzenie własnego łącznika modelu	653
Rejestracja własnego łącznika modelu	655
Podsumowanie	656

Rozdział 25. Kontrola poprawności modelu	657
Utworzenie przykładowego projektu	658
Utworzenie układu	659
Utworzenie widoków	660
Jawna kontrola poprawności modelu	661
Wyświetlenie użytkownikowi błędów podczas kontroli poprawności	662
Wyświetlanie komunikatów kontroli poprawności	664
Wyświetlanie komunikatów kontroli poprawności poziomu właściwości	667
Użycie alternatywnych technik kontroli poprawności	668
Kontrola poprawności w łączniku modelu	668
Definiowanie zasad poprawności za pomocą metadanych	670
Definiowanie modeli automatycznie przeprowadzających kontrolę	675
Użycie kontroli poprawności po stronie klienta	677
Aktywowanie i wyłączanie kontroli poprawności po stronie klienta	678
Użycie kontroli poprawności po stronie klienta	679
Jak działa kontrola poprawności po stronie klienta?	680
Wykonywanie zdalnej kontroli poprawności	681
Podsumowanie	684
Rozdział 26. Paczki	685
Utworzenie przykładowego projektu	685
Dodanie pakietów NuGet	685
Utworzenie modelu i kontrolera	686
Utworzenie układu i widoku	687
Profilowanie wczytywania skryptów i arkuszy stylów	689
Używanie paczek stylów i skryptów	691
Dodanie pakietu NuGet	691
Definiowanie paczki	692
Stosowanie paczek	694
Optymalizacja plików JavaScript i CSS	695
Podsumowanie	697
Rozdział 27. Web API i aplikacje w postaci pojedynczej strony	699
Aplikacja w postaci pojedynczej strony	700
Utworzenie przykładowego projektu	700
Tworzenie modelu	701
Dodanie pakietów NuGet	702
Tworzenie kontrolera Home	703
Dodanie układu i widoków	703
Ustawienie początkowego adresu URL i przetestowanie aplikacji	705
Zrozumienie Web API	706
Tworzenie kontrolera Web API	707
Testowanie kontrolera API	707
Jak działa kontroler API?	709
Jak wybierana jest akcja kontrolera API?	710
Mapowanie metod HTTP na metody akcji	711

Użycie Knockout do utworzenia aplikacji typu SPA	712
Dodanie bibliotek JavaScript do układu	712
Implementacja podsumowania	713
Implementacja funkcji tworzenia rezerwacji	719
Ukończenie aplikacji	722
Uproszczenie kontrolera Home	722
Zarządzanie wyświetlaniem zawartości	723
Podsumowanie	725
Skorowidz	727

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.

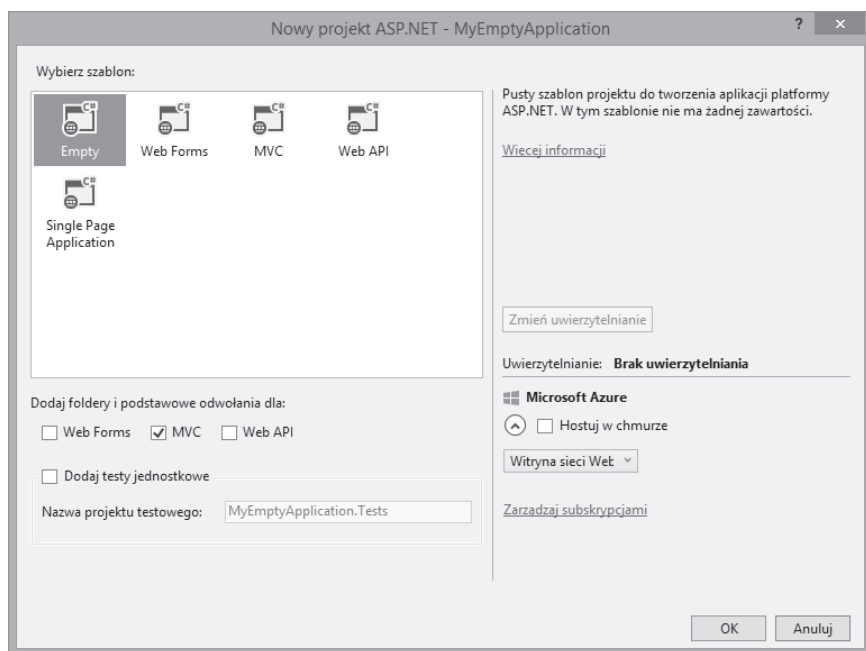
Moja rada (udzielana każdemu, kto popełnia błąd, pytając o nią) brzmi: rozpoczynaj pracę z pustym projektem, a następnie dodawaj niezbędne katalogi, pliki i pakiety. Dzięki takiemu podejściu nie tylko lepiej poznasz sposób działania platformy MVC, ale również zachowasz pełną kontrolę nad zawartością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

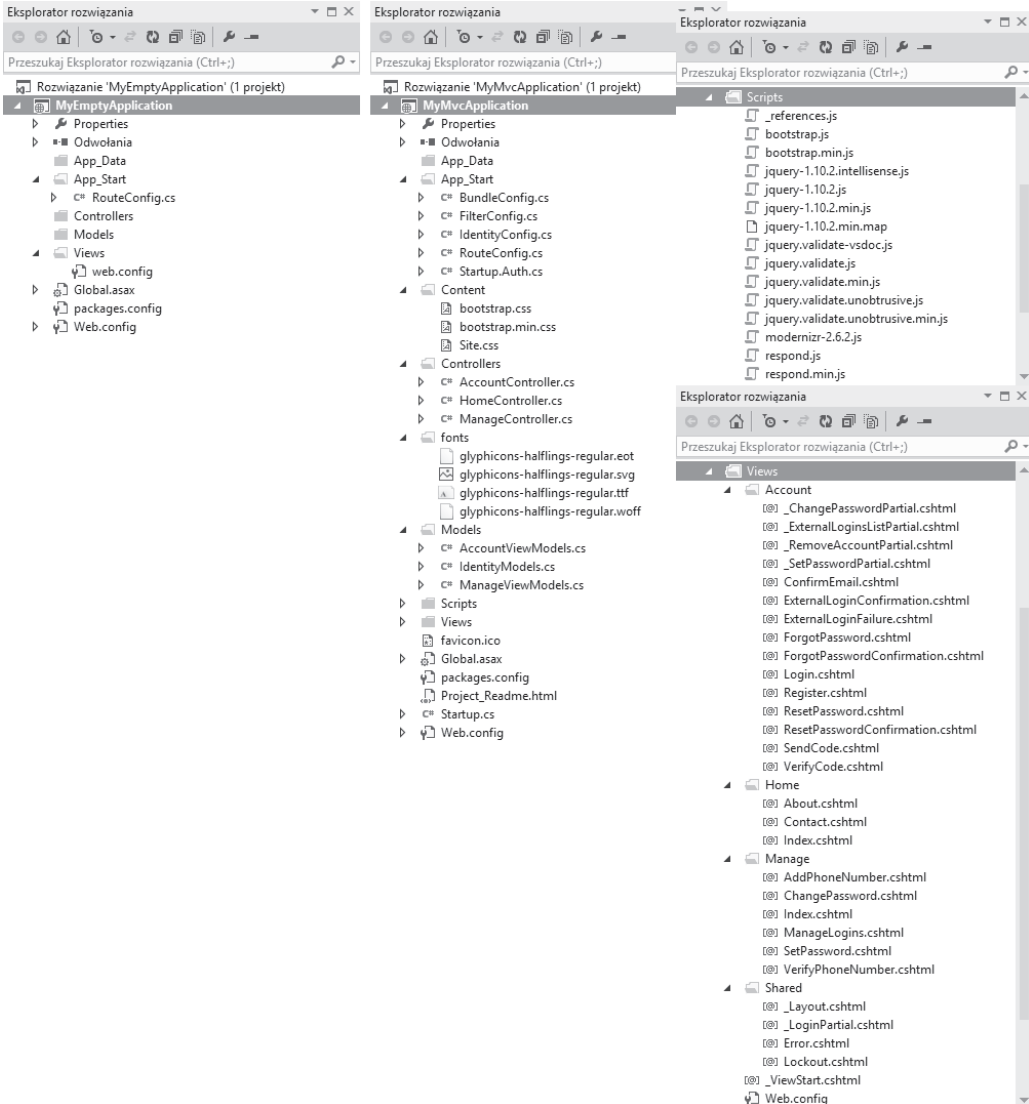
Kiedy po raz pierwszy tworzysz projekt MVC, do dyspozycji masz dwa punkty startowe: szablony *Empty* i *MVC*. Nazwy szablonów są nieco mylące, ponieważ podstawowe katalogi i podzespoły niezbędne dla platformy MVC można dodać do każdego szablonu projektu. W tym celu należy zaznaczyć pole wyboru *MVC* 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 <i>bin</i> w oknie <i>Eksplorator rozwiązania</i> , o ile nie klikniesz przycisku <i>Pokaż wszystkie pliki</i> . 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 <i>/Views/Web.config</i> 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 <i>nie</i> 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
<i>/Global.asax</i>	Definiuje globalną klasę aplikacji ASP.NET. Jego klasa kodu ukrytego (<i>/Global.asax.cs</i>) 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 <i>Global.asax</i> ma w aplikacji MVC taką samą funkcję jak w aplikacji Web Forms.
<i>/Web.config</i>	Jest to plik konfiguracyjny dla naszej aplikacji.	Plik <i>Web.config</i> 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 znajdują 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 `ControllerFactory`, co opiszę w rozdziale 19.

Stosowanie konwencji dla widoków

Widoki i widoki częściowe powinny być umieszczone w katalogu */Views/nazwaskontrolera*. 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:

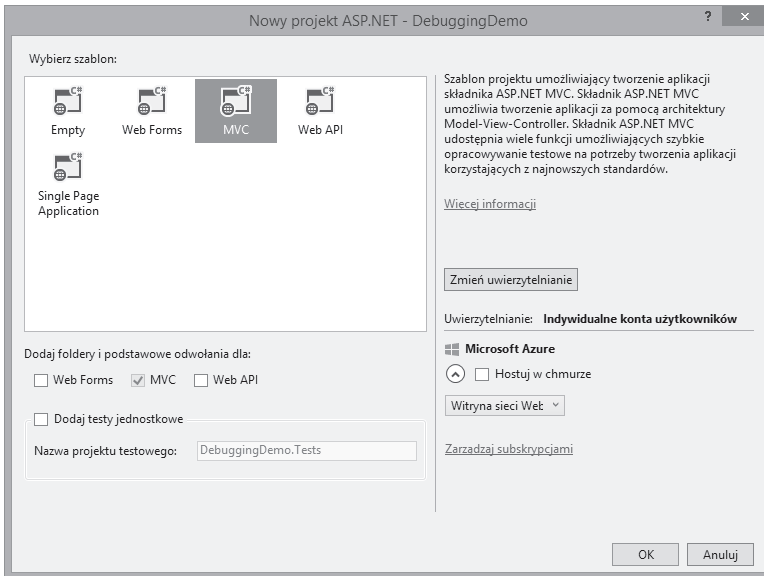
```
@{
    Layout = null;
}
```

Debugowanie aplikacji MVC

Aplikację ASP.NET MVC można debugować dokładnie tak samo jak aplikację ASP.NET Web Forms. Debugger 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 debugger i przeprowadzać różne zadania związane z usuwaniem błędów w projekcie MVC.

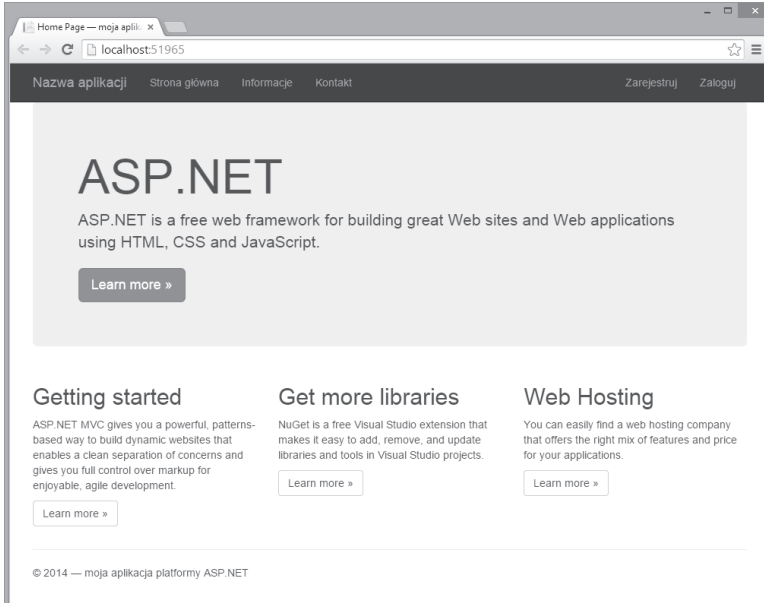
Tworzenie przykładowego projektu

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 uwiaryzalnianie wybrano opcję Indywidualne konta użytkowników, która oznacza użycie podstawowego systemu uwiaryzalniania 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`

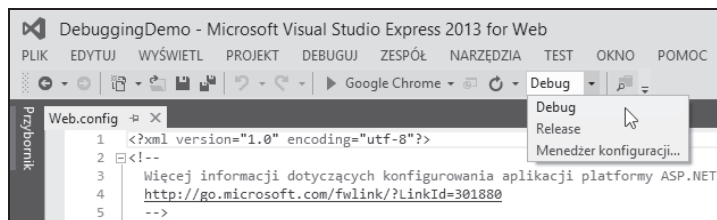
```
...
<system.web>
  <httpRuntime targetFramework="4.5.1" />
  <compilation debug="true" targetFramework="4.5.1" />
</system.web>
...
```

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 debugger będzie mógł operować na plikach klas wygenerowanych podczas kompilacji na żądanie.

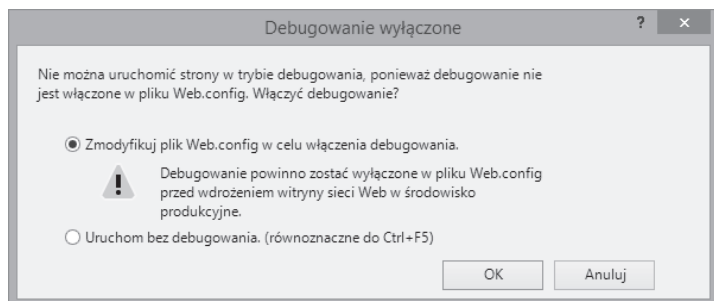
Oprócz zmiany w pliku konfiguracyjnym `Web.config`, konieczne jest upewnienie się, że Visual Studio umieszcza informacje debugowania w tworzonych plikach klas. Wprawdzie to nie ma znaczenia krytycznego, ale może powodować problemy, jeśli poszczególne ustawienia debugowania nie są zsynchronizowane. Upewnij 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`. Debugger zostanie uruchomiony.



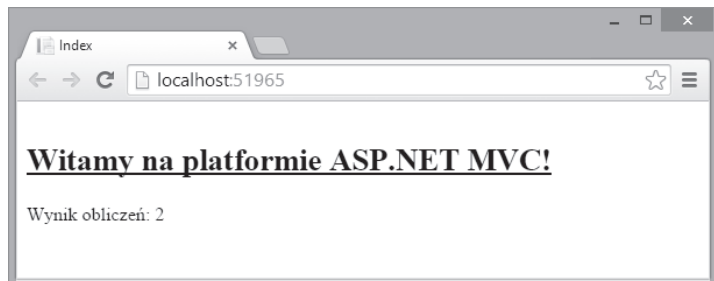
Rysunek 14.5. Wybór opcji konfiguracyjnej Debug



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.

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



Rysunek 14.7. Uruchomienie debugera

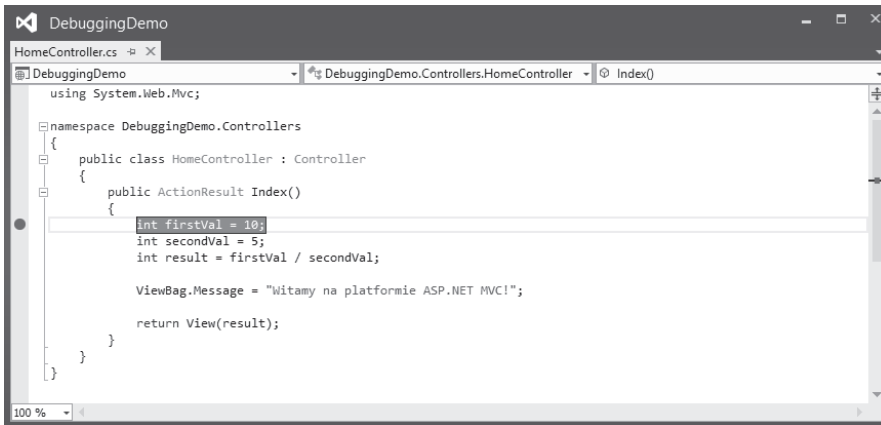
Przerywanie pracy aplikacji przez debugger Visual Studio

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.

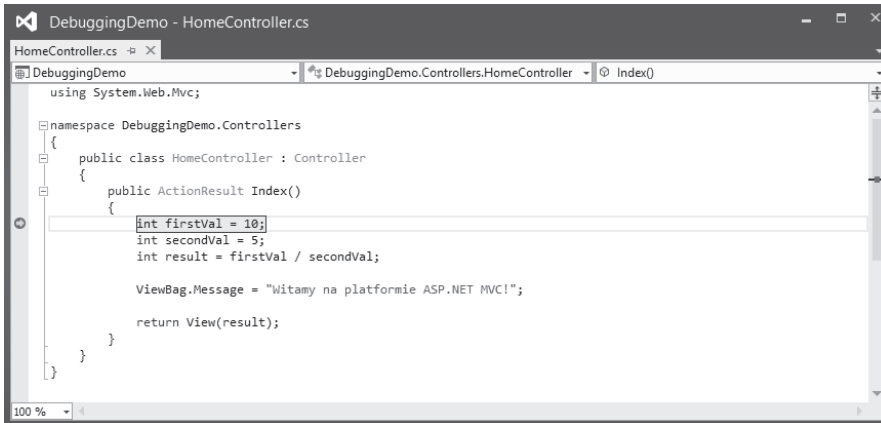
Użycie punktów przerwania

Punkt przerwania to instrukcja informująca debugger 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 debugger 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 debugger przerwie działanie aplikacji i przekaże kontrolę programiście. Jak pokazano na rysunku 14.9, Visual Studio podświetla wiersz kodu, w którym nastąpiło zatrzymanie działania aplikacji.

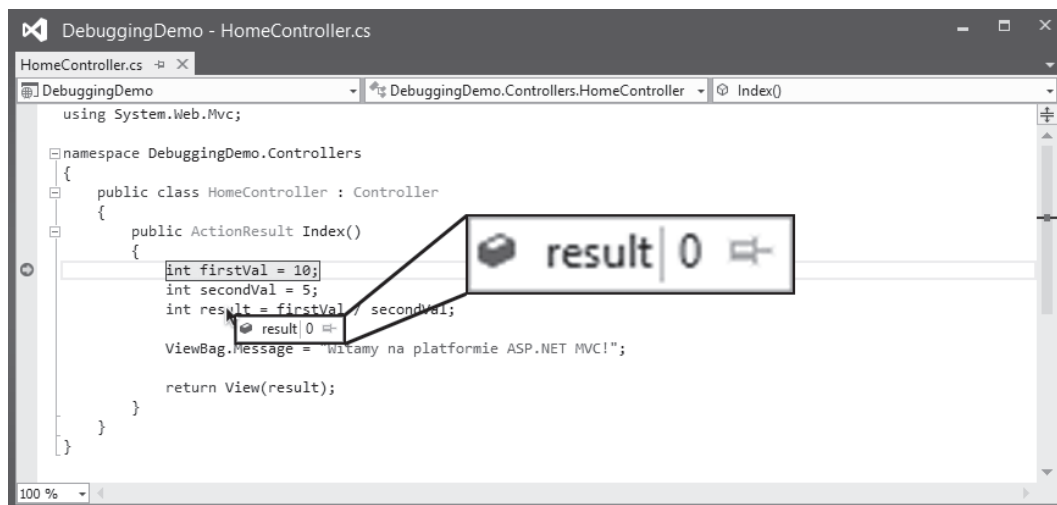
- **Uwaga** Punkt przerwania działa tylko w momencie, gdy skojarzona z nim instrukcja jest wykonywana. W naszym przykładzie punkt przerwania jest osiągnięty od razu po uruchomieniu aplikacji, ponieważ znajduje się w metodzie akcji wywoływanej w chwili otrzymania żądania domyślnego adresu URL. Jeżeli 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 przejściu 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

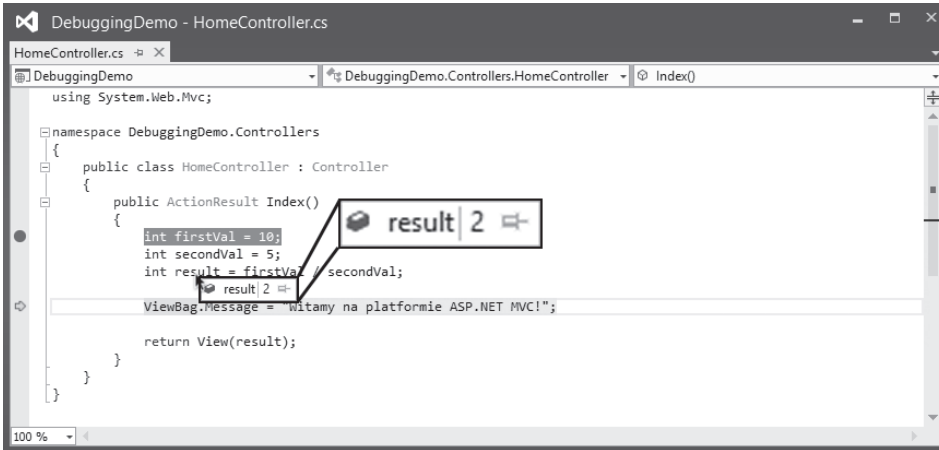
Najczęstszym sposobem użycia punktów przerwania jest próba znalezienia błędów w kodzie. Zanim będziesz mógł usunąć błąd z kodu, najpierw musisz ustalić, co tak naprawdę się dzieje. Jedną z najbardziej użytecznych funkcji oferowanych przez Visual Studio jest możliwość przeglądania i monitorowania wartości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

Przeгляд 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.

Name	Value	Type
▶ this	{DebuggingDemo.Controllers.HomeController}	DebuggingDemo.Controllers.HomeController
firstVal	10	int
secondVal	5	int
result	2	int

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. Debugger 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 obsługi 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;

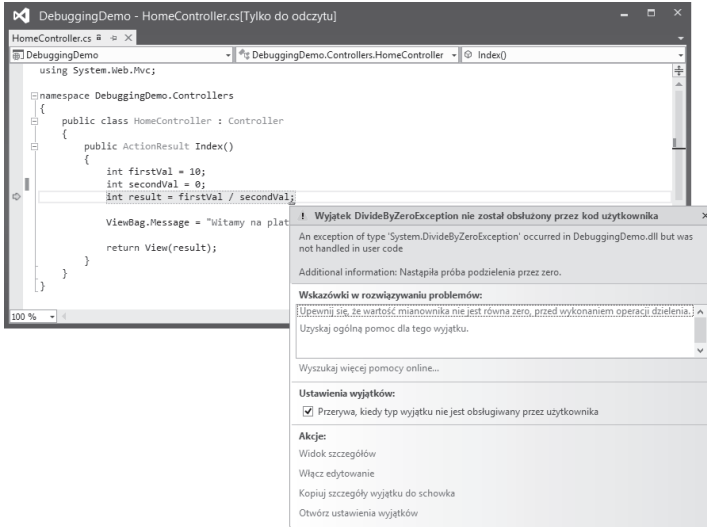
            ViewBag.Message = "Witamy w ASP.NET MVC!";

            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 debugger, 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 debugger 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 debugger, można zmodyfikować kod, a następnie kontynuować debugowanie. Visual Studio ponownie skompiluje aplikację, po czym odtworzy jej stan w momencie aktywowania debugera.

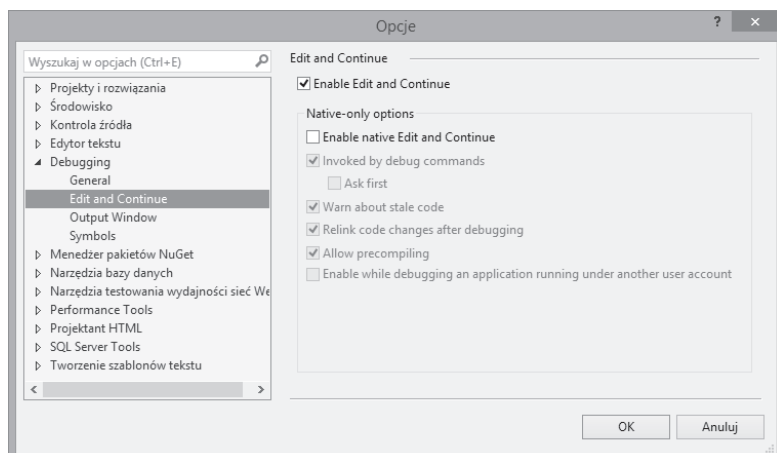
Włączanie opcji Edit and Continue

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

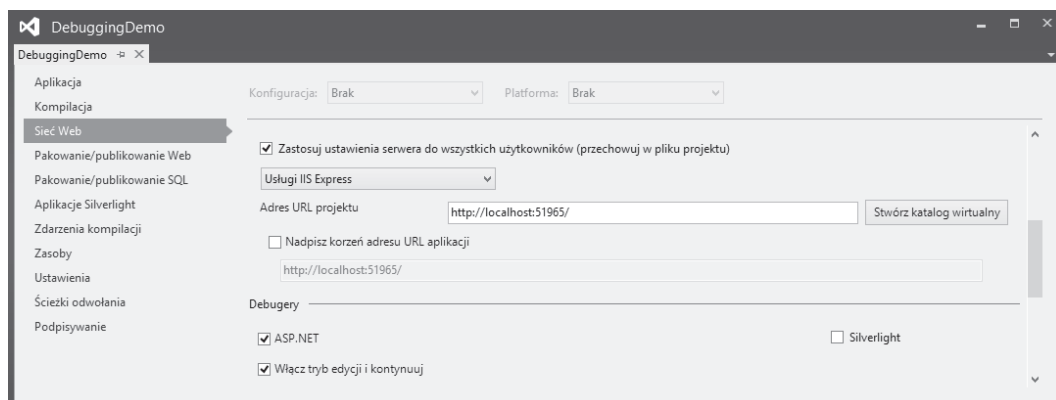
- Upewnij się, że w sekcji *Edit and Continue* dla opcji *Debugging* zaznaczona jest opcja *Enable Edit and Continue* (wybierz *Opcje...* z menu *Narzędzia*), jak pokazano na rysunku 14.14.
- We właściwościach projektu (wybierz *Właściwości DebuggingDemo...* z menu *Projekt*) przejdź do sekcji *Sieć 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>
```

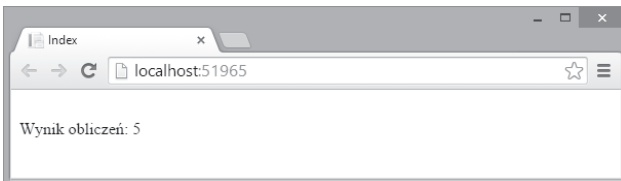
Edycja i kontynuowanie pracy

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*

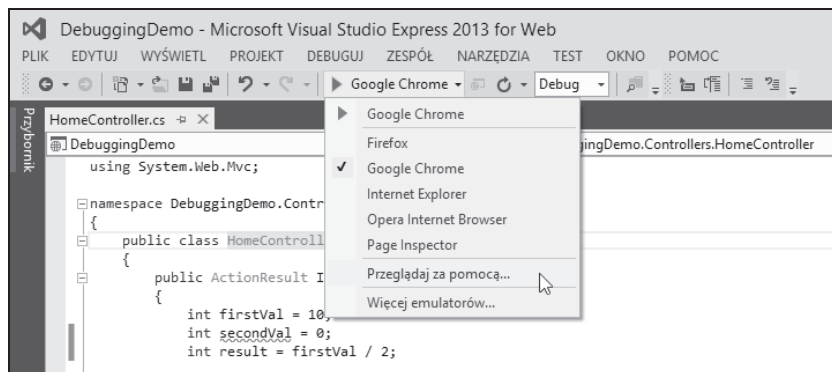
Poświęć chwilę na analizę wyniku tych działań. Uruchomiliśmy aplikację zawierającą błąd — próbę dzielenia przez zero. Debugger wykrył wyjątek i zatrzymał wykonywanie programu. Aby poprawić błąd, zmodyfikowaliśmy kod, zamieniając odwołanie do zmiennej na literał o wartości 5. Następnie wznowiliśmy działanie 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 debugger. 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 *Przełączaj za pomocą...*, jak pokazano na rysunku 14.17.

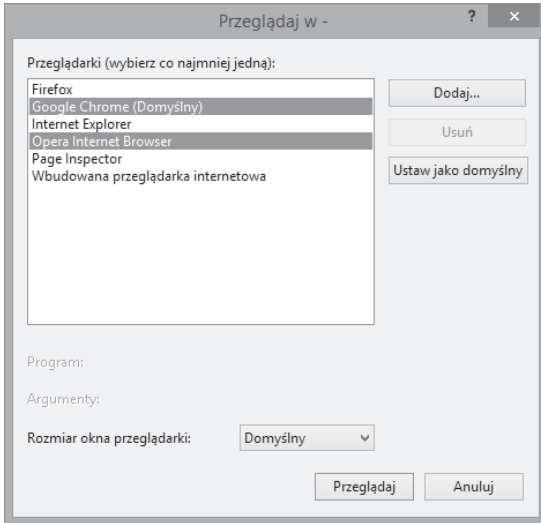


Rysunek 14.17. Przygotowanie do wyboru przeglądarek internetowych używanych wraz z funkcją *Browser Link*

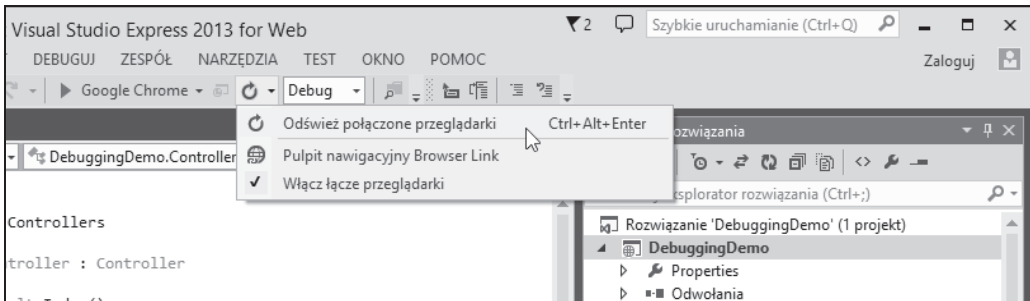
Na ekranie zostanie wyświetlone okno dialogowe *Przełączaj w*. Naciśnij klawisz *Control*, a następnie zaznacz przeglądarki internetowe, których chcesz używać. Na rysunku 14.18 widać, że wybrałem Google Chrome i Opera Internet Browser. Za pomocą tego okna dialogowego możesz również dodać nowe przeglądarki,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

Po kliknięciu przycisku *Przełączaj* 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 *Przełączaj w*. Funkcja *połączonych przeglądarek* jest użyteczna, ale użycie kodu JavaScript okazuje się problemem. Podczas pracy nad projektami w innych technologiach niż ASP.NET korzystałem 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

A

- abstrakcja, 20
- adres URL, 49, 193, 425
- adres URL z systemu routingu, 449
- AJAX, Asynchronous JavaScript and XML, 605
- akcja, 427
 - CustomVariable, 371, 372
 - GetPeopleDataJson, 628
 - GetRemoteData, 519
 - kontrolera API, 710
 - ValidateDate, 684
- akcje
 - potomne, 544, 545
 - w kontrolerze Customer, 390
- aktualizowanie repozytorium, 286
- aktywator
 - kontrolerów, 504
 - zadań, 505
- aliasy dla kontrolera, 369
- antywzorzec, 68
- API Node.js, 23
- aplikacja SportsStore, 163
- aplikacje
 - mobilne, 255
 - MVC, 31
 - Rails, 22
 - SPA, 28, 699, 700, 712
- architektura
 - model-widok, 68
 - model-widok-prezenter, 70
 - model-widok-widok-model, 70
 - MVC, 23
 - trójwarstwowa, 69
- arkusz stylów, 56
- asercja, 362
- ASP.NET MVC, 19
 - API, 26
 - architektura MVC, 23
 - kontrola nad HTML, 24
 - łatwość testowania, 24
 - routing, 25
 - rozszerzalność, 24
- ASP.NET MVC 5, 27
- ASP.NET Web Forms, 20
- asynchroniczność, 23
- atrybut
 - ActionName, 509
 - AttributeUsage, 489
 - Authorize, 305
 - Bind, 642
 - class, 266, 565
 - data-bind, 716
 - data-transition, 271
 - DataType, 590, 591
 - Debug, 341, 695, 696
 - Display, 590
 - DisplayName, 589, 590
 - HiddenInput, 586, 587
 - href, 58
 - HttpPost, 510, 560, 637
 - loginUrl, 304, 456
 - NonAction, 511
 - Remote, 683
 - Route, 388–401
 - RouteArea, 419
 - SessionState, 516
 - UIHint, 592, 598

- atrybuty
 - autoryzacji, 462
 - routingu, 387, 388
 - warunkowe, 266
- automatyczna
 - inferencja typów, 97
 - kontrola, 675
- automatyczne wyróżnianie błędów, 58
- automatycznie implementowane
 - właściwości, 83, 85
- automatyzacja interfejsu użytkownika, 22, 79

B

- baza danych, 175, 325
 - Azure, 325
 - LocalDB, 177
 - SportsStore, 177
- bezpieczeństwo, 303
- biblioteka
 - Bootstrap, 24, 29, 59, 63
 - jQuery, 608, 693, 713
 - jQuery Mobile, 268, 271
 - Knockout, 24, 28, 712–715, 719
 - LINQ, 98
 - Modernizr, 24
 - Moq, 155
 - TPL, 517
 - WatiN, 79
- błąd, 153, 377, 634
 - 401, 453
 - 404, 367, 452, 480

- błąd
 - kontroli poprawności, 56, 58, 665, 669
 - niejednoznaczności kontrolera, 417
 - błędy
 - dołączania modelu, 650
 - kontroli poprawności, 664
 - na poziomie modelu, 676
 - na poziomie właściwości, 676
 - weryfikacji, 54
 - budowa modelu domeny, 66
- C**
- C#, 81
 - ciągi tekstowe, 554
 - CRUD, 275
 - cykl życia strony, 20
 - czas życia obiektów, 73, 144
- D**
- DAL, data access layer, 69
 - dane
 - JSON, 624, 707, 708
 - modelu widoku, 188
 - statystyczne, 690
 - stronicowania, 190, 214
 - uwierzytelniające, 456
 - wejściowe, 432
 - wyjściowe, 435
 - XML, 708
 - debuger Visual Studio, 342
 - debugowanie, 166, 175, 402
 - aplikacji, 338
 - atributów HTML, 403
 - encji koszyka, 217
 - funkcji Ajax, 714
 - intencji, 436
 - interfejsu, 242
 - definiowanie
 - kontrolera, 108
 - mechanizmu dołączania, 715
 - metody akcji, 646
 - metody rozszerzającej, 88
 - modelu, 108, 715
 - opcjonalnych segmentów URL, 373
 - paczki, 692
 - priorytetów kontrolerów, 377
 - sekcji, 536
 - wartości domyślnych, 363
 - własnych ograniczeń, 385
 - własnych prefiksów, 640
 - zmiennych segmentów, 370
 - degradowanie łączy Ajax, 618
 - deklarowanie zależności, 72
 - DI, dependency injection, 71
 - dotatkowe style CSS, 603
 - dotywanie
 - bibliotek JavaScript, 712
 - danych do bazy, 179
 - danych modelu widoku, 188
 - danych tabeli, 327
 - dynamicznych treści, 536
 - filtra, 460
 - filtra globalnego, 487
 - klasy modelu, 44
 - kolumn, 313
 - kontrolerek nawigacji, 201
 - kontrolera, 34, 35, 132, 171
 - kontroli poprawności, 53, 292
 - łącza do widoku, 45
 - łączy Ajax, 617
 - metadanych, 593
 - metody asynchronicznej, 520
 - modelu widoku, 186
 - Moq, 157
 - Ninject, 135
 - nowego produktu, 297
 - nowego widoku, 46
 - obiektu imitacyjnego, 157
 - obszaru, 414
 - odwołań, 166
 - pakietów NuGet, 678, 685, 691, 702
 - parametru metody akcji, 372
 - podsumowania koszyka, 233
 - przestrzeni nazw, 188
 - przycisków, 221
 - punktu przerwania, 343
 - responsywnej zawartości, 258
 - stronicowania, 184
 - stylu, 194
 - trasy, 193, 398
 - układu, 172
 - widoku, 39, 115, 132, 172
 - widoku HTML, 43
 - wielu filtrów, 489
 - zależności, 141
 - dołączanie
 - danych, 227, 715, 716
 - do kolekcji, 645
 - do tablic, 643
 - kodu HTML, 638
 - kolekcji, 646
 - modelu, 51, 432, 629, 653
 - dostosowanie systemu, 650
 - ręczne, 648
 - tablic i kolekcji, 643
 - typów prostych, 634
 - typów złożonych, 636
 - warunkowe, 144
 - właściwości, 640, 642
 - domyślne
 - przeszukiwanie przestrzeni nazw, 379
 - wartości parametrów, 434
 - zachowanie filtra, 492
 - domyślny
 - adres URL, 49, 363
 - kod, 47
 - łącznik modelu, 635
 - widok, 38
 - dopasowanie
 - adresów URL, 358, 365, 373, 376
 - tras, 399
 - dostawca
 - uwierzytelniania, 306, 307
 - wartości, 651, 653
 - dostosowanie systemu routingu, 408
 - dynamiczne dodawanie treści, 41, 536
 - dynamiczny widok, 42
 - dyrektywa click, 716
 - działanie
 - biblioteki Knockout, 715
 - filtra Authorize, 306
 - filtra wyniku, 483
 - filtrów akcji globalnych, 493
 - kontrolera API, 709
 - koszyka, 217
 - nieprzeszkadzających wywołań, 612
 - silnika wyniku, 530
 - dziedziczenie, 430
 - dziedziczenie po atrybutach, 672

E

edycja produktów, 282, 293
 edytor
 kodu, 344
 produktów, 287
 efekt
 kliknięcia łącza, 399
 selektywnego kodowania treści, 558
 użycia filtra, 489
 użycia łącznika modelu, 656
 użycia stylów, 586
 użycia widoku częściowego, 543
 zastosowania ograniczenia, 391
 element
 <credentials>, 305
 <div>, 62
 <form>, 233, 567
 , 316
 <input>, 48, 62, 233, 568
 <label>, 62, 239
 <link>, 58, 250
 <p>, 557
 <script>, 58, 294
 <select>, 62, 571
 <style>, 60, 233
 <table>, 233
 elementy projektu, 336
 eliminowanie powielania
 widoków, 264
 encje koszyka, 217
 Entity Framework, 181
 etykieta, 241

F

fabryka
 dostawcy wartości, 652
 kontrolerów, 380, 498, 501
 wbudowana, 502
 własna, 501
 filtr, 431, 455
 akcji, 478
 Authorize, 306
 autoryzacji, 461–463, 468
 uwierzytelniania, 305, 464, 467, 468
 wyjątku, 470, 471, 474, 476
 wyniku, 482

filtrowanie
 bez użycia atrybutów, 485
 kolekcji obiektów, 92
 listy produktów, 201
 obiektów, 95
 według kategorii, 204
 filtry globalne, 487
 format JSON, 624
 formatowanie danych, 119
 formaty
 danych, 708
 daty, 636
 formularz, 47, 50, 221, 560, 562
 Ajax, 608, 610
 RSVP, 44
 synchroniczny, 607
 szczegółów wysyłki, 239
 formy wyrażen lambda, 96
 framework
 Node.js, 23
 Rhino Mocks, 129
 funkcja
 \$.ajax, 714
 Browser Link, 350
 Edit and Continue, 349
 getAllItems, 715, 716
 Łącze przeglądarki, 690
 paczki, 685
 processData, 623
 removeItem, 718
 sendAjaxRequest, 714
 ViewBag, 119
 ViewData, 527
 funkcje
 aplikacji, 174
 platformy ASP.NET, 25
 routingu, 395
 typu anonimowego, 62
 w MVC 5, 27

G

generowanie
 adresów URL, 397, 404–407
 danych widoku, 173
 elementu <input>, 568
 etykiet, 241, 581
 listy kategorii, 209
 łączy, 419
 sekcji opcjonalnych, 540

stron WWW, 37
 widoku, 37, 53, 440–443, 532, 582
 wychodzących adresów URL, 406, 411
 gettery, 84
 graficzny interfejs użytkownika,
 GUI, 19
 GridView, 192

H

hasło, 304
 HTML, 24
 HTTP, 24

I

IIS, Internet Information Services, 323
 imitacja
 określonych wartości, 161
 repozytorium, 169, 205
 zachowania klasy, 160
 zakresu wartości, 162
 implementacja
 elementów <input>, 721
 filtra wyjątku, 477
 funkcji, 152
 funkcji tworzenia rezerwacji, 719
 interfejsu, 90, 243
 interfejsu IControllerActivator, 504
 IProductRepository, 170
 IView, 527, 528
 IViewEngine, 528
 IValueProvider, 649
 kontrolera koszyka, 222
 mechanizmu przetwarzania
 zamówień, 242
 metody Menu, 209
 metody OnActionExecuted, 481
 MVC, 67
 podsumowania, 713
 repozytorium, 183
 RouteBase, 408
 sprawdzenia uwierzytelniania, 466
 widoku listy, 278
 inferencja typów, 97

informacja

- dotycząca usługi Azure, 329
- o adresie, 639
- o żądaniu Ajax, 615

inicjowanie

- kolekcji, 87
- obiektów, 86, 87
- tablic, 87

instalacja

- pakietów narzędziowych, 166
- pakietów NuGet, 603
- pakietu Bootstrap, 59, 194

IntelliSense, 48, 67, 433, 444

interakcje w aplikacji, 67

interfejs, 90, 134

- IActionFilter, 478
- IActionInvoker, 506
- IAuthenticationFilter, 464
- IController, 428
- IControllerActivator, 504
- IControllerFactory, 498, 515
- IDependencyResolver, 138
- IDiscountHelper, 141, 156
- IEmailSender, 71, 73
- IEnumerable<T>, 91
- IHttpHandler, 413
- IKernel, 137
- IModelBinder, 228, 633
- IOrderProcessor, 242
- IOVERRIDEFilter, 492
- IProductRepository, 169
- IResultFilter, 482
- IRouteConstraint, 385
- IRouteHandler, 413
- IValidatableObject, 677
- IValueCalculator, 139, 145
- IViewEngine, 524
- repozytorium, 287
- typu CRUD, 275
- użytkownika, 314

IoC, inversion of control, 71

J

jawna kontrola poprawności, 661

język

- C#, 81
- HTML5, 21
- JavaScript, 614

JSON, JavaScript Object Notation, 621

K

katalog

- AdditionalControllers, 386
- Content, 420
- Controllers, 208
- Infrastructure, 137, 307, 482, 534, 672
- Models, 140, 630
- produktów, 264
- Scripts, 337, 678
- Shared, 631

kierowanie

- adresów URL, 409
- żądań, 411

klasa

- ActionExecutedContext, 482
- ActionExecutingContext, 479
- ActionFilterAttribute, 460, 483
- ActionMethodSelectorAttribute, 512
- AddressSummary, 640, 643, 654
- AddressSummaryBinder, 653
- AdminController, 75
- AjaxOptions, 611, 618
- Appointment, 674, 679
- AreaRegistrationContext, 416
- Assert, 150
- AuthenticationChallenge
 - ↳ Context, 465
- AuthenticationContext, 466
- AuthorizeAttribute, 463
- btn, 60
- Cart, 217, 218
- CartController, 247
- CartLine, 217
- CartModelBinder, 228
- Controller, 430, 431, 453
- ControllerContext, 228, 470
- CountryValueProvider, 651
- CustomHelpers, 555
- CustomRedirectResult, 437
- CustomValueProviderFactory, 651
- DebugDataView, 528
- DefaultControllerFactory, 502–506
- DefaultDiscountHelper, 141
- DefaultModelBinder, 634, 636, 669

DependencyResolver, 500, 501, 504

- EFProductRepository, 183
- EmailSettings, 244
- ExecutionContext, 470
- FilterConfig, 487
- FormCollection, 649
- FormsAuthentication, 307
- FutureDateAttribute, 673
- HandleErrorAttribute, 476
- HandleErrorInfo, 478
- HomeController, 36, 82, 139, 378
- HtmlHelper, 552, 558
- HttpMethodConstraint, 382
- HttpRequestBase, 361
- HttpStatusCodeResult, 452
- IControllerFactory, 515
- It, 159
- LegacyRoute, 409
- LinqValueCalculator, 131, 133, 139, 141, 145, 155
- MinimumDiscountHelper, 147, 149, 153, 160
- MvcRouteHandler, 360
- MyAsyncMethod, 103
- NavController, 210
- NinjectDependencyResolver, 138, 139
- PagingHelpers, 186
- PasswordResetHelper, 71, 72
- Product, 168, 281, 293
- ProductController, 502
- RemoteService, 518
- RequestContext, 429, 499
- RouteCollection, 359
- RouteValueDictionary, 412
- ShoppingCart, 88, 90, 131, 134, 156
- System.Diagnostics.Debug, 145
- TagBuilder, 553
- text-center, 60
- User, 76
- ViewBag, 41
- ViewContext, 526, 552
- ViewDataDictionary, 526
- ViewEngineResult, 524, 525
- ViewResult, 212
- WebMail, 64

- klasy
 - Base, 429
 - CartLine, 218
 - ograniczania trasy, 384
 - Wrapper, 430
 - zaprzyżnżone, 593
 - Knockout, 712
 - kod
 - 401, 453
 - 404, 367, 452, 480
 - źródłowy biblioteki MVC, 26
 - kodowanie
 - treści metody pomocniczej, 556
 - wartości danych, 557
 - kody statusu HTTP, 398, 453
 - kolejność
 - przeszukiwania danych parametrów, 634
 - przetwarzania żądania Ajax, 620
 - tras, 367
 - wykonywania filtrów, 489, 491
 - wykorzystania przestrzeni nazw, 378
 - wyszukiwania szablonów, 597
 - kolekcja, 86, 125
 - ValueProviderFactories.
 - ↳Factories, 652
 - ViewEngine.Engines, 529
 - komponent MyEmailSender, 70
 - komunikat
 - kontroli poprawności, 669
 - potwierdzający, 290
 - kontroli poprawności, 664
 - o błędach, 168, 249, 513, 541, 667, 673, 680
 - konfiguracja
 - bazy danych, 177, 325
 - domyślnej trasy, 173
 - filtra globalnego, 488
 - kontenera DI, 167
 - paczek, 693
 - pliku widoku, 39
 - routingu, 37, 407, 418, 617
 - serwera aplikacji, 421
 - uwierzytelniania formularzy, 304
 - wstrzykiwania zależności, 137
 - konflikt nazw, 377, 418
 - konsola JavaScript, 718
 - konstrukcja warunkowa, 123
 - konstruktor
 - EmailOrderProcessor, 244
 - MvcHtmlString, 556
 - kontekst Entity Framework, 180
 - kontener DI, 73, 167
 - kontrola nad HTML, 20, 24
 - kontrola poprawności, 53, 249, 250, 292
 - alternatywne techniki, 668
 - dla pojedynczych właściwości, 668
 - formularza, 54
 - jawna, 661
 - metody akcji, 684
 - modelu, 292, 657, 661
 - po stronie klienta, 677–681
 - unikanie konfliktów, 681
 - w jQuery, 681
 - w łączniku modelu, 668
 - wbudowane atrybuty, 670
 - własny atrybut, 671, 674
 - wyświetlanie komunikatów, 664
 - wyświetlenie błędów, 662
 - zdalna, 681, 684
 - kontroler, 34, 66, 427
 - AdminController, 308
 - AdminController, 295
 - API, 709
 - CRUD, 276
 - Customer, 390
 - DerivedController, 437
 - GoogleAccountController, 465
 - Home, 136, 472, 658, 703, 722
 - HomeController, 82, 132, 456, 630
 - koszyka, 222, 230, 246
 - nadzorujący, 70
 - nawigacji, 208
 - RemoteData, 521
 - Web API, 707
 - kontrolery
 - asynchroniczne, 517, 519
 - bezstanowe, 515
 - zapasowe, 500
 - kontrolka
 - GridView, 192
 - podsumowania koszyka, 235
 - kontrolki nawigacji, 201
 - kontrolowana degradacja formularza, 613
 - konwencje MVC, 337
 - koszyk, 216–227
 - dodanie produktu, 225
 - działanie, 217
 - encje, 217
 - implementowanie kontrolera, 222
 - kontroler, 230, 246
 - podsumowanie, 233
 - przyciski, 221
 - testowanie, 218
 - usuwanie towarów, 233
 - wartość towarów, 220
 - wyświetlanie zawartości, 223, 226
 - kwalifikowane adresy URL, 404
- ## L
- licznik stron, 213
 - link do widoku, 57
 - LINQ, Language Integrated Query, 26, 81, 98, 625
 - lista
 - kategorii, 209
 - produktów, 171, 201, 260
 - lokalizacja widoków, 533
- ## Ł
- łącza, 45
 - Ajax, 616
 - do akcji, 419
 - do formularza RSVP, 44
 - kategorii, 211
 - nawigacji, 192
 - przeglądarki, 690
 - stron, 185, 191, 214
 - z atrybutami, 403
 - łączenie
 - filtrów, 468, 485
 - metod akcji, 44
 - ograniczeń, 392
 - sekcji, 538
 - statycznych segmentów URL, 368
 - warunkowe, 143
 - łącznik modelu, 229, 633, 635, 642, 644, 647, 649, 653

M

- mapowanie, 423
 - metod HTTP, 711
 - obiektoowo-relacyjne, ORM, 22
- mechanizm
 - IntelliSense, 48, 67, 433, 444
 - przetwarzania zamówień, 242
 - rozwiązywania zależności, 137
 - TempData, 451
 - View Bag, 212
- menedżer pakietów NuGet, 337
- menu nawigacji, 208
- metadane, 284, 589–593, 670
- metadane modelu, 586
- metoda
 - @RenderBody, 113
 - Action, 194, 207, 264, 405, 546
- metoda
 - ActionLink, 45, 397–404
 - Add, 503, 655
 - AddModelError, 666
 - Application_Start, 359, 416, 529
 - AreEqual, 150
 - Assert, 362
 - Authenticate, 311
 - BeginForm, 49, 221, 564–566, 613
 - BeginRouteForm, 566
 - BindModel, 654
 - CanChangeLoginName, 77
 - ChangeLoginName, 76
 - Checkout, 237, 238
 - ContainsPrefix, 651
 - ContinueWith, 104
 - Create, 295
 - CreateController, 499
 - CreatePerson, 637, 638
 - CustomVariable, 372
 - DateTime, 683
 - Delete, 299
 - DisplayMessage, 556
 - DisplaySummary, 641
 - Edit, 283, 288
 - Editor, 579
 - EditorFor, 579, 639
 - EditorForModel, 284
 - ExecuteResult, 437, 440
 - FilterByCategory, 92, 93
 - FilterTest, 486
 - Get, 137
 - GetAll, 138
 - GetCart, 222, 229
 - GetControllerSessionBehavior, 501
 - GetImage, 317
 - GetLegacyURL, 409
 - GetNames, 572
 - GetPeopleData, 611
 - GetRemoteData, 518
 - GetRemoteDataAsync, 521
 - getTestObject, 149
 - GetValue, 655
 - GetVirtualPath, 411
 - HandleUnknownAction, 514
 - Helper, 579
 - HelperFor, 579
 - HttpClient.GetAsync, 104, 105
 - IgnoreRoute, 424
 - Index, 36, 223, 276
 - InRequestScope, 146
 - InvokeAction, 506
 - Is, 162
 - IsAjaxRequest, 627
 - IsInRange, 162
 - IsSectionDefined, 541
 - IsValid, 674
 - IsValidForRequest, 512
 - Keep, 452
 - List, 189
 - MakeBooking, 659
 - MapRoute, 360, 367, 380
 - Menu, 209, 263
 - Names, 643
 - OnActionExecuted, 481
 - OnActionExecuting, 479
 - OnAuthentication, 467
 - OnAuthenticationChallenge, 466
 - OnResultExecuted, 482
 - OrderByDescending, 100
 - Partial, 198
 - PartialView, 263
 - Peek, 451
 - ProduceOutput, 436
 - RedirectToAction, 406, 450, 451
 - RedirectToRoute, 449
 - RegisterRoutes, 363, 399
 - ReleaseController, 501
 - RemoveLine, 219
 - Render, 695, 696
 - RenderBody, 261, 278
 - Returns, 159
 - RouteLink, 211
 - RsvpForm, 46
 - SaveChanges, 288
 - SaveProduct, 287
 - Select, 100
 - SetResolver, 138
 - TestIncomingRoutes, 369
 - TestIncomingRouteResult, 362
 - TestRouteMatch, 361, 363
 - Throws, 162
 - ToConstant, 170
 - TryUpdateModel, 650
 - UpdateModel, 649
 - UpdateReservation, 711
 - ValidationMessageFor, 667
 - ValidationSummary, 54, 664, 665
 - ValueProducts, 141
 - View, 662
 - WebMail, 64
- metody
 - akcji, 45, 125, 431, 433
 - akcji i trasy, 390
 - asynchroniczne, 103, 520, 521
 - dołączania warunkowego, 144
 - dostarczające dane, 627
 - HTTP, 381, 711
- klasy
 - Assert, 150
 - DefaultControllerFactory, 506
 - DefaultModelBinder, 669
 - It, 159
- kontrolera filtra, 485
- MapRoute, 416
- OnActionExecuting, 490
- pomocnicze, 45, 547
 - Ajax, 601
 - HTML, 186, 191, 551, 604
 - szablonowe, 575–599
 - szkieletów, 584, 585
 - URL, 601
 - wbudowane, 559
 - wewnętrzne, 549
 - zewnętrzne, 551
- przeciążone, 51
- rozszerzające, 88–91

metody

- filtrujące, 92
- LINQ, 101
- opóźnione, 102
- PageLinks, 187
- użycie delegata, 93
- testowe, 149
- zakresu, 146

mobilne przeglądarki internetowe, 253

model, 66

- code-behind, 20
- danych, 43
- domeny, 66, 108, 164, 168, 313
- widoku, 186

modyfikowanie

- kodu HTML widoku, 40
- konstruktora, 144
- projektu, 347

monitorowanie żądań, 717

Moq, 155

MVC, 65, 67

MVP, 70

MVVM, 70

N

nadawanie

- priorytetu kontrolerom, 378
- stylów, 58, 196
- stylu polom wyboru, 663

nadpisywanie

- filtra, 491–494
- metod, 505

nagłówki

- ContentLength, 104
- user-agent, 386

narzędzia, 129

- automatyzacji UI, 22
- testów jednostkowych, 22

narzędzie Moq, 129

nawias klamrowy, 357

nawigacja, 363

nazywanie tras, 360, 408

niejednoznaczność

- kontrolerów, 417
- metod akcji, 514

Node.js, 22

nowoczesne API, 26

O

obiekt, 86

- ActionResult, 437, 453
- Cart, 230
- EmailSettings, 245
- HttpResponseBase, 436
- HttpResponseMessage, 104
- kernel, 136
- Mock, 159
- ModelMetaData, 240
- Person, 582, 633
- Product[], 126
- RedirectResult, 438
- Reservation, 721
- Route, 400
- RouteCollection, 399
- RouteData, 410
- RouteValues, 410
- ViewBag, 41, 445
- ViewResult, 296

obiekty

- .NET, 629
- ActionResult, 523
- imitujące, 155–158
- kontekstu, 432, 433
- modelu, 109
- modelu widoku, 443, 445
- Reservation, 714
- wywołujące akcje, 506, 507
 - wbudowane, 508
 - własne, 507
- zakresu, 144
- zarządzające zależnościami, 504

obsługa

- atrybutów routingu, 388
- błędów, 662
- błędów dołączania modelu, 650
- danych zdjęcia, 316
- formularzy, 50, 559
- IntelliSense, 67
- JavaScriptu, 614
- JSON, 621
- kategorii, 202
- komunikatu, 291
- łańcucha zależności, 73
- nieprzeszkadzających wywołań, 607

nieznanych akcji, 514

ostatniego uwierzytelnienia, 469

trasy, 412, 413

własnych błędów, 477

wyjątku, 347, 473

zdarzeń, 721

żądania POST, 50

żądań, 414, 288

żądań JSON, 626

obszary, 414

odczyt

danych, 432, 446

zdjęć, 319

odmiany MVC, 69

odświeżenie przeglądarki

internetowych, 351

odwołania między projektami, 166

odwrócenie kontroli, IoC, 71

ograniczanie

liczby adresów, 424

łącznika, 649

pobierania wartości, 653

ograniczenia

trasy, 380

do zbioru wartości, 381

klasy, 384

własne, 385

z użyciem metod HTTP, 381

zastosowanie, 390

typu i wartości, 383

układu responsywnego, 267

okno

Dodaj szkielec, 35

Dodawanie widoku, 39

Eksplorator rozwiązania, 32, 165

Eksplorator testów, 77

Eksploratora serwera, 178

Locals, 345

Menedżer odwołań, 83

opcja Edit and Continue, 347

opcje Ajax, 612

opcjonalne segmenty URL, 373, 375

operacje CRUD, 279

opóźnione zapytania LINQ, 102

optymalizacja plików

CSS, 695

JavaScript, 695

OSI, Open Source Initiative, 26

P

- paczki, 685
- pakiet
 - Bootstrap, 59, 194
 - narzędziowy, 166
 - NuGet, 335, 603, 702
 - optymalizacyjny, 396
- parametr
 - Cart, 247
 - Func, 94
 - ShippingDetails, 247
- parametry
 - generyczne, 508
 - konstruktora, 73, 142
 - obowiązkowe, 434
 - opcjonalne, 434
 - typów złożonych, 595
- pasywna implementacja widoku, 70
- pętla foreach, 126, 128
- pierwsza aplikacja, 31
- platforma
 - ASP.NET, 333
 - Azure, 332
- plik
 - _AdminLayout.cshtml, 277, 290, 294
 - _BasicLayout.cshtml, 112, 113, 116
 - _Layout.cshtml, 172, 208, 235, 250, 256, 260, 263, 537–540, 561, 603, 608, 631, 659, 703
 - _Layout.Mobile.cshtml, 268, 269
 - _ViewStart.cshtml, 114, 195
 - AccountController.cs, 309, 457
 - ActionName.cshtml, 400–404, 412
 - Address.cshtml, 646
 - AddressSummary.cs, 647
 - AddressSummaryBinder.cs, 654
 - AdminAreaRegistration.cs, 415
 - AdminController.cs, 276, 288, 299, 305, 316, 356
 - AdminSecurityTests.cs, 311
 - AdminTests.cs, 281
 - Appointment.cs, 658, 672, 676, 679, 686
 - BasicController.cs, 428, 435
 - Boolean.cshtml, 599
 - bootstrap.css, 60
 - bootstrap-theme.css, 60
 - Cart.cs, 217
 - CartController.cs, 222, 229, 234, 237, 246
 - CartItemViewModel.cs, 223
 - CartItemBinder.cs, 227
 - Checkout.cshtml, 238, 240, 249
 - Completed.cshtml, 251, 660
 - CountryValueProvider.cs, 651
 - CreatePerson.cshtml, 563–572, 577, 595
 - CustomActionAttribute.cs, 479
 - CustomActionInvoker.cs, 506
 - CustomAuthAttribute.cs, 461
 - CustomControllerFactory.cs, 498, 515
 - Customer.cs, 488
 - CustomerController.cs, 355, 388, 408, 492, 511
 - CustomHelpers.cs, 555
 - CustomLocationViewEngine.cs, 534
 - CustomOverrideActionFiltersAttribute.cs, 493
 - CustomRedirectResult.cs, 437
 - CustomRouteHandler.cs, 413
 - CustomValueProviderFactory.cs, 652
 - CustomVariable.cshtml, 371
 - Data.cshtml, 518
 - DebugDataView.cs, 527
 - DebugDataViewEngine.cs, 528
 - DemoArray.cshtml, 126
 - DemoExpression.cshtml, 120
 - DerivedController.cs, 438
 - DerivedControllers.cs, 431
 - Discount.cs, 140, 142, 143
 - DisplayPerson.cshtml, 582
 - DisplaySummary.cshtml, 641
 - Edit.cs, 292
 - Edit.cshtml, 283, 286, 296, 314
 - Editor.cshtml, 705
 - EfDbContext.cs, 181
 - EFProductRepository.cs, 182, 287, 317
 - EmailOrderProcessor.cs, 243
 - Enum.cshtml, 598
 - ErrorStyles.css, 250
 - FastController.cs, 516
 - FilterConfig.cs, 487
 - FlexibleDiscountHelper.cs, 143
 - FlexMenu.cshtml, 265
 - FlexMenu.Mobile.cshtml, 269
 - FormsAuthProvider.cs, 307
 - FutureDateAttribute.cs, 673
 - GetLegacyURL.cshtml, 409
 - GetPeople.cshtml, 606, 613, 619, 622
 - GetPeopleData.cshtml, 609
 - Global.asax, 415, 488, 501, 505, 529, 534
 - Global.asax.cs, 228, 359
 - GoogleAccountController.cs, 465
 - GoogleAuthAttribute.cs, 465, 467, 469
 - GuestResponse.cs, 44
 - HomeController.cs, 36, 41, 50, 84–91, 94–97, 115, 132, 340, 480, 512, 630, 703, 722
 - IAuthProvider.cs, 307
 - Index.cshtml, 40, 59, 109, 110, 132, 224, 232, 237, 279, 340, 416, 444, 548, 604, 704, 719, 723
 - Index.js, 713
 - IOrderProcessor.cs, 242
 - IProductRepository.cs, 169, 287, 298
 - IValueCalculator.cs, 134
 - LegacyController.cs, 409
 - LegacyRoute.cs, 409, 411
 - LinqValueCalculator.cs, 131, 144, 155
 - List.cshtml, 173, 191, 196, 535, 542
 - LocalAttribute.cs, 512
 - Login.cshtml, 310, 457
 - LoginViewModel.cs, 308
 - MakeBooking.cshtml, 660, 664, 667, 687
 - Menu.cshtml, 211, 213
 - MinimumDiscountHelper.cs, 147
 - MustBeTrueAttribute.cs, 672
 - MyAsyncMethods.cs, 103
 - MyExtensionMethods.cs, 88–93
 - MyPartial.cshtml, 542
 - MyStronglyTypedPartial.cshtml, 543
 - MyView.cshtml, 431
 - NameAndPrice.cshtml, 117

- Names.cshtml, 644
- NavController.cs, 208, 212, 263, 266
- NinjectDependencyResolver.cs, 137, 141, 145, 167, 170, 182, 245, 307
- NinjectWebCommon.cs, 139
- NoJoeOnMondayAttribute.cs, 674
- PagingHelpers.cs, 186
- PagingInfo.cs, 186
- PeopleController.cs, 602, 609, 621, 625, 626
- Person.cs, 559, 576, 587, 630
- PersonMetadata.cs, 594
- Product.cs, 83, 85, 169
- ProductController.cs, 171, 189, 202, 214, 318, 497
- ProductsListViewModel.cs, 189, 201
- ProductSummary.cs, 197, 321
- ProductSummary.cshtml, 221
- ProductSummary.Mobile.cshtml, 272
- ProfileActionAttribute.cs, 481
- ProfileAllAttribute.cs, 484
- ProfileResultAttribute.cs, 482
- RangeError.cshtml, 474, 475
- RangeErrorPage.html, 471
- RangeExceptionAttribute.cs, 471, 474
- RemoteDataController.cs, 518, 520
- RemoteService.cs, 518, 520
- Reservation.cs, 701
- ReservationRepository.cs, 701
- Result.cs, 496
- Result.cshtml, 82
- Role.cshtml, 596
- RouteConfig.cs, 173, 193, 205, 358–375, 378, 402, 407, 411, 413, 418, 421, 424, 565, 632
- RouteTests.cs, 360
- RsvpForm.cshtml, 46, 54, 61
- ShippingDetails.cs, 236
- ShippingDetails.cshtml, 241
- ShoppingCart.cs, 88, 131
- SimpleMessageAttribute.cs, 489, 491
- Site.css, 341
- StaticContent.html, 420
- Styles.css, 57
- Summary.cshtml, 234, 258, 704
- Thanks.cshtml, 64
- Time.cshtml, 545
- UnitTest1.cs, 149, 151
- UnitTest2.cs, 157, 160
- UserAgentConstraint.cs, 385
- View.cshtml, 532
- ViewStart, 172
- Web.config, 181, 183, 188, 245, 304–307, 341, 695
- WebApiConfig.cs, 709
- WebController.cs, 707, 711
- pliki**
 - .aspx, 39
 - .cshtml, 39, 534
 - CSS, 695
 - JavaScript, 695
 - układu, 112
 - widoków, 113
- pobieranie**
 - danych, 99, 432
 - schematu bazy danych, 327
- podgląd wartości danych, 344**
- podpowiadanie nazwy, 111**
- podsumowanie**
 - danych profilowania, 696
 - koszyka, 233, 235
- podzespół System.Net.Http, 83, 104**
- poła z błędami, 56**
- pole**
 - __VIEWSTATE, 55
 - wyboru, 567
- polecenia**
 - Razor, 109
 - SQL, 328
- polecenie**
 - @if, 126
 - @model, 109
 - if, 124
 - Install-Package, 59
 - switch, 123, 124, 125
 - using, 149
- połączenie**
 - ograniczeń trasy, 384
 - z bazą danych, 181, 326
- połączone przeglądarki, 350**
- pomijanie systemu routingu, 424**
- poprawianie**
 - kodu, 153
 - licznika stron, 213
- potwierdzenie pobierania danych, 717**
- powiązanie z repozytorium, 182**
- powielanie widoków, 264**
- prefiks, 638**
 - @, 124
 - ~/, 393
 - trasy, 392
 - własny, 640
- priorytety**
 - kontrolera, 378
 - przestrzeni nazw, 502
- procedura obsługi zdarzeń, 721**
- profilowanie**
 - aplikacji, 696
 - wczytywania skryptów, 689
- programowanie**
 - sieciowe, 26, 253
 - sterowane testami, 74, 78
 - sterowane testami, TDD, 22
 - witryn WWW, 19, 21
 - zwinne, 22
- projekt, 333**
 - ClientFeatures, 685
 - ControllerExtensibility, 496
 - ControllersAndActions, 428
 - DebuggingDemo, 339
 - EssentialTools, 130
 - Filters, 456
 - HelperMethods, 548, 576, 602
 - LanguageFeatures, 81
 - ModelValidation, 658
 - MvcModels, 630
 - PartyInvites, 32
 - Razor, 107
 - SportsStore, 164, 201, 227, 253, 275, 303
 - SportsStore.WebUI, 256
 - UrlAndRoutes, 353
 - UrlsAndRoutes, 396
 - Views, 526
 - WebServices, 700
 - WorkingWithRazor, 531
- projektowanie modelu danych, 43**
- projekty**
 - sklepów internetowych, 163
 - testów jednostkowych, 147
 - Visual Studio, 135
- przechwytywanie informacji**
 - o adresie, 639
- przeciążanie metod, 564, 665**

przeгляд projektu, 333
 przeglądarka internetowa, 350, 423, 691
 przekazywanie
 danych
 do widoku, 443
 z użyciem ViewBag, 445
 parametrów, 401
 przekierowanie, 406, 407, 447, 449
 do innej metody akcji, 450
 trwałe, 448
 z użyciem tras, 449
 aplikacji, 346
 przestrzeń nazw, 127, 377
 EssentialTools.Models, 149
 System.Web, 430
 System.Web.Http, 711
 przesyłanie
 danych edycji, 289
 plików, 314
 zdjęć, 312
 przetwarzanie
 danych, 119
 JSON, 622
 właściwości modelu, 634
 zamówień, 242, 247
 przychodzące żądania URL, 353
 przycisk
 Dodaj do koszyka, 216
 Publish, 331
 Update Database, 179
 usuwania, 232
 zamówienia, 237
 Złóż zamówienie, 237
 przyciski koszyka, 221
 przygotowanie obiektów danych, 626
 przypisanie wartości atrybutu, 121
 przrostek Attribute, 488
 publikowanie aplikacji sieciowej, 329
 punkt przerwania, 343, 344
 pusty ciąg znaków, 666

R

raportowanie kategorii, 212
 Razor, 107, 532
 reakcja na wyjątek, 474
 refaktoring
 aplikacji, 196
 kontrolera Home, 139
 metod akcji, 609

rejestrowanie
 aktywatora kontrolerów, 505
 dostawcy uwierzytelniania, 307
 fabryki kontrolerów, 501
 implementacji, 244
 łącznika modelu, 655, 656
 silnika widoku, 529
 trasy, 358–360
 repozytorium
 abstrakcyjne, 169
 produktów, 182, 286
 resetowanie hasła, 70
 responsywne
 funkcje CSS, 257
 listy produktów, 260
 nagłówki, 256
 siatki, 262
 REST, 21
 routing, 25, 37, 387, 395
 URL, 353
 żądań, 420
 rozbudowa modelu, 721
 rozdzielanie komponentów, 71
 rozszerzalność, 24
 rozszerzanie
 bazy danych, 312
 kontrolerów, 495
 modelu domeny, 236, 313
 wyrażenia filtrującego, 96
 rozwiązywanie zależności, 137, 138, 142
 Ruby on Rails, 22
 rzutowanie
 na IController, 501
 parametru, 409
 wartości, 123

S

schemat
 adresów URL, 205, 357, 424
 opcjonalny, 373
 statyczny, 366
 własne zmienne, 370
 bazy danych, 177, 326, 327
 segmenty statyczne adresu URL, 366
 selekcja metod akcji, 509, 512
 selektywne
 kodowanie treści, 558
 kodowanie wartości danych, 558

Selenium RC, 79
 separacja zadań, 374
 serializowanie wartości, 55
 serwer
 aplikacji, 421
 IIS Express, 422
 settery, 84
 silnie typowane metody, 570
 silnik
 Razor, 107, 531, 536, 541
 widoku, 67, 107, 523
 składanie zamówień, 236
 składnia Knockout, 719
 składniki klasy TagBuilder, 553
 słowo kluczowe
 async, 105
 await, 105
 class, 62
 model, 444
 new, 73, 136
 return, 104
 using, 49, 100
 SPA, Single Page Application, 28, 699
 SportsStore, 163
 administracja, 275
 bezpieczeństwo, 303
 naszyk, 216, 227
 nawigacja, 201
 promowanie marki, 257
 usprawnienia, 303
 wersja mobilna, 253
 sprawdzanie
 błędów, 54
 istnienia sekcji, 540
 uwierzytelniania, 466
 stan
 aplikacji, 345
 sesji, 515, 516
 standardy sieciowe, 21
 statyczne segmenty URL, 366, 368
 sterowanie edycją i widocznością, 586
 stosowanie
 konwencji dla klas kontrolerów, 337
 konwencji dla układów, 338
 konwencji dla widoków, 337
 metod rozszerzających, 90
 układu, 113
 strona podsumowania, 251

stronicowanie, 184
 struktura plików i katalogów, 33
 styl widoku
 Index, 59
 RsvpForm, 61
 Thanks, 62
 style, 194
 Bootstrap, 195
 CSS, 603
 system
 kontrolni poprawności, 249
 routingu, 37, 353, 357
 szablon
 Empty, 196
 projektu, 32, 339
 szablonowe metody pomocnicze,
 575–599
 szablonowy
 ogólne, 597
 wbudowane, 599
 szkielec, 583

Ś

ścieżka dostępu, 443

T

tabela Products, 180
 tablica, 125
 TDD, test-driven development, 74,
 78
 test jednostkowy, 22, 74, 147, 155,
 163
 adres URL, 360
 akcja Index, 276
 aktualizowanie, 203
 dane stronicowania, 190
 filtrowanie według kategorii,
 205
 generowanie listy kategorii, 210
 generowanie widoku, 441
 kody statusu HTTP, 453
 kontroler koszyka, 230
 kontrolery i akcje, 438
 koszyk na zakupy, 218
 łącza stron, 187
 obiekty modelu widoku, 445
 odczyt zdjęć, 319
 ograniczenia tras, 382
 opcjonalne segmenty URL, 375

przekierowanie, 448
 przesyłanie danych edycji, 289
 przetwarzanie zamówień, 247
 raportowanie kategorii, 212
 segmenty statyczne, 369
 stronicowanie, 184
 usuwanie produktów, 299
 uwierzytelnianie, 311
 ViewBag, 446
 wartości domyślne, 365
 zliczanie produktów, 216
 zmienne segmentów, 372
 zmienne segmenty
 przechwytyjące, 376
 testowanie, 24, 153
 aplikacji, 133, 356, 458, 549,
 705
 funkcji tworzenia rezerwacji,
 722
 kontrolera API, 707
 koszyka, 218
 operacji dołączania danych,
 717
 przychodzących adresów URL,
 360
 regresyjne, 74
 segmentów statycznych, 369
 silnika widoku, 529
 stronicowania, 184
 widoku, 41
 zmiennych segmentów, 372
 zmiennych segmentów
 przechwytyjących, 376
 testy
 automatyczne, 74
 integracyjne, 74, 79
 jednostkowe, 22, 74, 147, 155,
 163
 TPL, Task Parallel Library, 517
 trasa formularza, 565
 trasy, 37, 207, 357
 dla plików na dysku, 422
 domyślne, 173
 nazwane, 408
 o zmiennej długości, 375
 tworzenie
 abstrakcyjnego repozytorium,
 169
 adresów URL, 603, 605
 akcji potomnych, 544
 aplikacji SPA, 712

arkusza stylów, 56
 atrybutu kontroli poprawności,
 672, 674
 bazy danych, 176, 324
 danych wyjściowych, 435
 dostawcy uwierzytelniania, 306
 dostawcy wartości, 651
 edytora, 596
 elementu select, 572
 fabryki kontrolerów, 498
 filtra wyjątku, 470
 filtrujących metod
 rozszerzających, 92
 formularza Ajax, 610
 formularzy, 562
 imitacji repozytorium, 169, 205
 implementacji IViewEngine,
 528
 implementacji RouteBase, 408
 interfejsu użytkownika, 314
 klasy kontrolera, 500
 klasy Product, 168
 kodu zabezpieczeń, 461
 kontekstu Entity Framework,
 180
 kontrolera, 340, 355, 428, 430
 AccountController, 308
 asynchronicznego, 519, 520
 CRUD, 276
 Home, 703
 nawigacji, 208
 Web API, 707
 koszyka, 216
 łańcucha zależności, 140
 łącznika modelu, 227, 653
 łączy, 603, 605
 łączy Ajax, 616
 metody akcji, 45
 metody akcji Edit, 283
 metody pomocniczej, 549
 modelu, 701
 modelu domeny, 168
 nieprzeszkadzających
 formularzy, 608
 nowego projektu, 31
 nowych produktów, 295
 obiektów .NET, 629
 obiektów parametrów, 434
 obiektu domeny, 66
 obiektu imitacji, 158
 obiektu obsługi trasy, 412

- tworzenie
 - obszaru, 414, 418
 - pliku układu, 277
 - procedury obsługi zdarzeń, 721
 - projektu, 334, 338
 - projektu testów
 - jednostkowych, 147
 - przycisków koszyka, 221
 - repozytorium produktów, 182
 - responsywnego nagłówka, 256
 - responsywnej listy produktów, 260
 - rezerwacji, 719, 722, 725
 - rozwiązania, 164
 - schematu bazy danych, 326
 - sekcji opcjonalnej, 541
 - silnika widoku, 523
 - szablonu ogólnego, 597
 - tabeli, 178, 327
 - tablicy obiektów, 97
 - testów automatycznych, 20
 - testów jednostkowych, 148
 - tras, 389
 - trasy, 358
 - typu anonimowego, 97
 - układu, 112, 659, 687
 - widoków, 660
 - widoku, 37, 109, 210, 309, 340, 356, 687
 - widoku częściowego, 196, 197, 542
 - widoku edycji, 283
 - widoku formularza
 - synchronicznego, 606
 - widoku Index, 278
 - wielu formularzy, 221
 - witryny internetowej, 324
 - zewnętrznej metody
 - pomocniczej, 551
 - znaczników select, 571
 - typ
 - bool, 54
 - click, 721
 - enum, 572
 - JsonRequestBehavior, 622
 - Role, 572
 - SessionStateBehavior, 515
 - value, 721
 - typy
 - anonimowe, 97
 - filtrów, 459
 - niestandardowe, 646
 - proste, 634
 - ściśle określone, 570
 - wyliczeniowe, 572
 - złożone, 595, 636
- ## U
- uaktualnienie projektu testów
 - jednostkowych, 397
 - układ, 111
 - _AdminLayout.cshtml, 310
 - dla urządzeń mobilnych, 268
 - układy
 - responsywne, 255, 267
 - współdzielone, 115
 - ukrywanie właściwości obiektu, 588
 - ulepszanie adresów URL, 193, 205
 - upraszczanie
 - kontrolera Home, 722
 - tras, 396
 - uruchamianie
 - aplikacji, 168, 174
 - debugera, 341
 - testów, 152
 - usługa Azure, 324, 329
 - usprawnienie funkcji removeItem, 718
 - ustawianie
 - danych widoku, 41
 - obiektu zakresu, 144
 - opcji Ajax, 612
 - ustawienia
 - aplikacji, 245, 330
 - regionalne, 636
 - usuwanie
 - atrybutu class, 266
 - jawnej kontroli poprawności, 675
 - produktów, 232, 298–300
 - rezerwacji, 718
 - towarów, 233
 - uwierzelnianie, 311, 466
 - formularzy, 304
 - z użyciem filtrów, 305
 - użycie
 - @using, 188
 - akcji potomnych, 544, 546
 - aktywatora kontrolerów, 504
 - atrybutów routing, 387
 - atrybutu
 - Bind, 642
 - DataType, 590, 591
 - Display, 241, 242, 590
 - DisplayName, 589
 - HiddenInput, 586, 587
 - HttpPost, 510
 - NonAction, 511
 - Remote, 683
 - UIHint, 592, 598
 - automatycznie
 - implementowanych
 - właściwości, 83
 - biblioteki Moq, 159
 - Bootstrap, 196
 - delegata, 93
 - dołączania danych, 227
 - dołączania modelu, 51, 632
 - fabryki kontrolerów, 502
 - filtra HandleErrorAttribute, 477
 - filtrów, 458, 459
 - akcji, 478–481
 - autoryzacji, 461
 - globalnych, 487, 489
 - uwierzelniania, 464
 - wbudowanych, 483
 - wyjątków, 470, 471
 - wyników, 482
 - filtrującej metody
 - rozszerzającej, 92
 - formularza HTML, 561
 - funkcji połączonych
 - przeglądarek, 350
 - inferencji typów, 97
 - inicjalizatorów obiektów, 86
 - interfejsu IController, 428
 - JavaScript, 23
 - Knockout, 712
 - kolekcji, 645
 - konstrukcji warunkowych, 123
 - kontenera DI, 73
 - kontrolerek, 570
 - kontrolerów asynchronicznych, 517
 - kontrolerów bezstanowych, 515
 - kontroli poprawności, 679
 - LINQ, 99
 - łączenia warunkowego, 143
 - łącznika modelu, 633
 - mechanizmu ViewBag, 212, 446

metadanych, 284, 589, 590, 591
 metadanych modelu, 586
 metod
 asynchronicznych, 103, 520, 521
 kontrolera filtra, 485
 pomocniczych, 582
 pomocniczych szkieletów, 584
 rozszerzających, 88
 metody Html.ActionLink, 397
 Moq, 155
 niestandardowego ograniczenia trasy, 386
 Ninject, 133
 notacji kropki, 100
 obiektu imitacyjnego, 157
 obiektu Mock, 159
 obiektu modelu widoku, 443
 opcji Edit and Continue, 347
 opcjonalnych segmentów URL, 374
 opóźnionych metod
 rozszerzających LINQ, 102
 paczek stylów i skryptów, 691
 parametrów metod akcji, 433
 parametrów opcjonalnych, 434
 parametru konstruktora, 143
 pliku ViewStart, 114
 polecenia warunkowego, 124
 prefiksu trasy, 392
 punktów przerwania, 343
 responsywnego układu
 Bootstrap, 258
 responsywnej siatki, 262
 sekcji w widoku, 538
 słowa async, 105
 słowa await, 105
 stylów, 586
 szablonowych metod
 pomocniczych, 578
 szablonu niestandardowego, 599
 tras nazwanych, 408
 trasy, 363
 typów anonimowych, 97
 układów współdzielonych, 115
 układu responsywnego, 255
 warunkowego dołączania, 144
 wbudowanego filtra
 autoryzacji, 463
 wbudowanego filtra wyjątków, 476

wbudowanych metod
 pomocniczych, 559
 widoków częściowych, 198, 541, 543
 wielu egzemplarzy, 145
 wielu tras, 379
 Windows Azure, 324
 własnych lokalizacji, 536
 własnych nazw akcji, 508
 własnych zmiennych, 372
 właściwości
 Layout, 113
 Order, 490
 Response, 436
 RouteData.Values, 372
 wyrażenia
 @model, 110
 @Model, 109
 @using, 127
 wyrażień
 lambda, 93, 95
 Razor, 118, 121
 regularnych, 380
 wywołań zwrotnych, 619
 zakresu żądania, 145
 zewnętrznej metody
 pomocniczej, 553
 zmiennych segmentu, 390

V

ViewState, 20
 Visual Studio, 31, 135, 147, 157
 debuger, 341, 342
 projekty MVC, 333
 tworzenie rozwiązania, 164
 Visual Studio Express 2013 for Web, 28

W

warstwa dostępu do danych, 69
 wartości
 domyślne, 363, 365
 domyślne parametru, 636
 typu DataType, 591
 typu wyliczeniowego, 515
 wartość null, 54, 320, 635
 warunkowe dołączanie, 144
 WatiN, 79

wbudowane
 atrybuty kontroli poprawności, 670
 szablony widoku, 592, 593
 typy ActionResult, 439
 wbudowany filtr wyjątków, 476
 wczytywanie
 arkuszy stylów, 689
 skryptów, 689
 wdrażanie aplikacji, 323, 328–331
 Web API, 699, 706
 wersja
 biurowa, 259
 mobilna, 259
 weryfikacja danych, 55
 wewnętrzna metoda pomocnicza, 549
 widok, 37, 523–46
 Address, 647
 Completed, 660, 664
 CreatePerson, 572
 DebugData, 529
 DemoExpression.cshtml, 120
 DisplayPerson, 582, 583
 DisplaySummary, 641
 dla urządzeń mobilnych, 269, 273
 edycji, 283
 formularza synchronicznego, 606
 GetPeople, 606, 622
 Index, 59, 60, 281, 417
 Index.cshtml, 548
 List.cshtml, 191, 195, 198
 listy, 278, 280
 Login, 310
 MakeBooking, 660, 662, 689
 NameAndPrice, 118
 RangeError, 474, 475
 Razor, 109, 532
 Role, 596
 RsvpForm, 50, 61
 ściśle określonego typu, 46
 Thanks, 52, 62, 63
 z formularzem, 47
 widoki, 66
 beztypowe, 444
 częściowe, 196–198, 541
 silnie typowane, 444
 wielokrotne wykorzystanie zmiennych segmentów, 402

- wiersz poleceń NuGet, 59
- właściwości
 - automatyczne, 85, 86
 - klasy
 - ActionExecutedContext, 482
 - ActionExecutingContext, 479
 - AjaxOptions, 611, 618
 - ControllerContext, 470
 - ExceptionContext, 470
 - HandleErrorAttribute, 476
 - HandleErrorInfo, 478
 - HtmlHelper, 552
 - ModelBindingContext, 654
 - RequestContext, 499
 - ViewContext, 526, 552
 - ViewDataDictionary, 526
 - konstruktora, 142
 - wyszukiwania, 534
- właściwość
 - @Model, 109
 - AjaxOptions.Confirm, 615
 - AjaxOptions.LoadingElementId, 615
 - AppRelativeCurrentExecution
 - ↳FilePath, 361
 - ClientName, 662
 - controller, 502
 - CurrentCategory, 202
 - Date, 658
 - DiscountSize, 142
 - DisplayName, 242
 - Exclude, 643
 - HomeAddress, 595, 639
 - HttpContext.Session, 516
 - IDENTITY, 178
 - Layout, 113, 115
 - LoadingElementDuration, 615
 - LoadingElementId, 614
 - Model, 655
 - ModelState.IsValid, 662
 - Order, 490
 - PersonId, 588
 - Response, 436
 - RouteData.Values, 372
 - RouteExistingFiles, 420
 - ViewBag, 212
 - ViewContext, 265
- włączanie opcji Edit and Continue, 347
- wprowadzanie danych, 567, 570
- wskazywanie
 - trasy, 566
 - widoku, 581
- wstawianie wartości danych, 119
- wstrzykiwanie zależności, DI, 71, 129, 137
- wstrzyknięcie konstruktora, 140
- wybieranie szablonu wyświetlania, 591
- wybór
 - pliku układu, 116, 117
 - przeglądarki, 350
 - pustego kontrolera, 35
 - silnika widoku, 67
 - widoku, 262
- wychodzące adresy URL, 395, 398, 399
- wyjątek, 346
 - ArgumentOutOfRangeException
 - ↳Exception, 471
 - DivideByZeroException, 476
 - System.ArgumentException, 183
- wykonywanie
 - intencji, 436
 - przekierowań, 447
 - zapytań, 98
 - zapytań LINQ, 98, 103
- wykrywanie żądań Ajax, 626
- wylączenie
 - kontroli poprawności, 678
 - nieprzeszkadzających wywołań
 - Ajax, 607
 - właściwości, 588
- wymuszanie separacji zadań, 374
- wynik akcji, 431
- wypełnianie obszaru, 416
- wyrażenia
 - Razor, 118
 - regularne, 380
- wyrażenie
 - @foreach, 126
 - @model, 110, 714
 - @Model, 111, 119, 133
 - @Model.ClientName, 716
 - @using, 127, 128
 - @ViewBag, 119
 - lambda, 93–96
- wyróżnianie
 - bieżącej kategorii, 212–214
 - błędów, 58
 - pól, 56
- wysyłanie
 - formularza, 641
 - kodów, 452
 - koðu 401, 453
 - koðu 404, 452
- wyszukiwanie
 - lokalizacji widoków, 533
 - szablonów, 597
- wyświetlanie
 - błędów, 662
 - danych JSON, 625
 - elementów, 581
 - komunikatów, 250, 667
 - komunikatów kontroli
 - poprawności, 664
 - komunikatu potwierdzającego, 290
 - liczby stron, 215
 - listy produktów, 171
 - łączy nawigacji, 192
 - łączy stron, 185, 191, 207
 - nieobsługiwane go widoku, 530
 - niewłaściwych łączy, 214
 - strony podsumowania, 251
 - wartości domyślnej zmiennej, 371
 - wartości zmiennej, 371
 - zawartości, 723
 - zawartości koszyka, 223, 226
 - zawartości tablic, 125
 - zdjęć produktów, 321
- wywołania zwrotne, 618
- wywołania zwrotne Ajax, 620
- wywoływanie
 - akcji potomnej, 545
 - metod akcji, 495
- względne adresy URL, 404
- wzorzec
 - architektury trójwarstwowej, 69
 - model-widok, 69
 - MVC, 65
 - POST-Redirect-GET, 447
 - repozytorium, 169
 - Smart UI, 67
 - URL, 357, 363, 366

Z

zabezpieczanie kontrolera administracyjnego, 303
 zadania testów
 integracyjnych, 79
 jednostkowych, 74
 zagnieżdżona klasa modelu, 638
 zakres wartości, 162
 zalety ASP.NET MVC, 23
 zamówienia, 236
 zapisywanie zdjęć, 316
 zapytania
 LINQ, 98
 opóźnione, 102
 złożone, 100
 zarządzanie
 katalogiem, 275
 kodowaniem ciągów tekstowych, 554
 stanem sesji, 515, 516
 wyświetlaniem zawartości, 723
 zależnościami, 504
 zasady poprawności, 670
 zastępowanie szablonów, 599
 zastosowanie, *Patrz także* użycie atrybutu kontroli poprawności, 672
 atrybutu Route, 389

filtra
 akcji, 481
 uwierzytelniania, 467
 wyjątku, 473
 wyniku, 483
 interfejsu, 134, 135
 metadanych, 285
 ograniczeń trasy, 390
 paczek, 694, 696
 sekcji układu, 536
 stylów Bootstrap, 195
 zdalna kontrola poprawności, 681, 684
 zdarzenia, 721
 zdjęcia produktów, 321
 zgłaszanie wyjątku, 161
 zintegrowane środowisko programistyczne, IDE, 129
 zliczanie produktów, 216
 zmiana kodu
 metody pomocniczej, 550
 znaczników, 551
 zmienianie hasła, 75
 zmienne
 lokalne, 265
 przechwytyjące, 375
 segmentu, 370, 389, 402
 znacznik, *Patrz* element

znak
 @, 62, 109, 123, 533
 cudzysłowu, 188
 zrywanie zależności, 72
 zwracanie
 błędów, 452
 kodów HTTP, 452
 kodu HTML, 440
 wyniku, 159

Ź

źródło danych łącznika, 649

Ż

żądania
 Ajax, 614
 asynchroniczne, 615
 dla plików dyskowych, 420
 żądanie
 adresu URL, 400, 511
 GET, 50, 426, 622
 pliku dyskowego, 423
 pliku statycznego, 422
 POST, 50, 288, 426, 622
 przesłania pliku, 421
 wartości null, 635

PROGRAM PARTNERSKI

GRUPY WYDAWNICZEJ HELION



- 1. ZAREJESTRUJ SIĘ**
- 2. PREZENTUJ KSIĄŻKI**
- 3. ZBIERAJ PROWIZJĘ**

Zmień swoją stronę WWW
w działający bankomat!

Dowiedz się więcej i dołącz już dzisiaj!

<http://program-partnerski.helion.pl>

GRUPA WYDAWNICZA

 **Helion SA**

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 telefonicznie

☎ 0 801 339900

☎ 0 601 339900

Informatyka w najlepszym wydaniu

Sprawdź najnowsze promocje:
 ● <http://helion.pl/promocje>
 Książki najchętniej czytane:
 ● <http://helion.pl/bestsellery>
 Zamów informacje o nowościach:
 ● <http://helion.pl/nowosci>

Helion SA
 ul. Koszalińska 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



cena: 119,00 zł

Apress