

ASP.NET MVC 4

Programowanie aplikacji webowych

Zbigniew Fryźlewicz, Ewa Bukowska, Daniel Nikończuk

ASP.NET ASP.NET ASP.NET
MODEL MODEL MODEL
VIEW VIEW VIEW
ER CONTROLLER

4

Programuj z ASP.NET MVC 4 – i zostań
deweloperem przyszłości!

ASP.NET MVC 4, czyli po co komu
wzorce projektowe

Dodatki zewnętrzne, czyli jak wspomagać
aplikację innymi technologiami

Studia przypadku, czyli od czego zacząć
i na czym skończyć w praktyce



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.

Redaktor prowadzący: Ewelina Burska

Projekt okładki: ULABUKA

Materiały graficzne na okładce zostały wykorzystane za zgodą Shutterstock.

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)

Drogi Czytelniku!

Jeżeli chcesz ocenić tę książkę, zajrzyj pod adres

<http://helion.pl/user/opinie?aspm4w>

Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

Materiały do książki można znaleźć pod adresem:

<ftp://ftp.helion.pl/przyklady/aspm4w.zip>

ISBN: 978-83-246-6534-1

Copyright © Helion 2013

Printed in Poland.

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

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

Spis treści

- Wstęp 7
- Rozdział 1. Model-Widok-Kontroler 11
 - 1.1. Krótka historia MVC 11
 - 1.2. Klasyczna postać wzorca MVC 12
 - 1.3. MVC jako wzorec złożony 14
 - 1.3.1. Obserwator 15
 - 1.3.2. Strategia 15
 - 1.3.3. Kompozyt 16
 - 1.4. MVC dla aplikacji webowych 17
 - 1.5. Warianty MVC 18
 - 1.5.1. Wzorec Model-Widok-Prezenter (MVP) 18
 - 1.5.2. Wzorec Prezentacja-Abstrakcja-Kontrola (PAC) 19
 - 1.5.3. Wzorec Model-Widok-Widok Modelu (MVVM) 19
 - 1.6. Implementacja MVC w językach programowania 19
 - 1.6.1. Ruby: Ruby on Rails 19
 - 1.6.2. Python: Django 20
 - 1.6.3. Java: Spring, Struts 20
 - 1.6.4. PHP: Zend Framework 20
 - 1.7. Podsumowanie 20
- Rozdział 2. Framework ASP.NET MVC 4 23
 - 2.1. Założenia projektowe frameworku ASP.NET MVC 25
 - 2.1.1. Rozszerzalność 26
 - 2.1.2. Konwencja nad konfiguracją 26
 - 2.1.3. Pełna kontrola nad HTML i HTTP 26
 - 2.1.4. Testowalność 27
 - 2.1.5. Elastyczny mechanizm routingu 27
 - 2.1.6. Dostępność sprawdzonych funkcji i elementów frameworku ASP.NET 27
 - 2.1.7. Nowoczesne API 27
 - 2.1.8. Dostęp do kodu źródłowego 28
 - 2.2. Wersje i usprawnienia frameworku ASP.NET MVC 28
 - 2.3. Routing URL 29
 - 2.4. Testowanie 30
 - 2.4.1. Testowanie zwróconego widoku 31
 - 2.4.2. Testowanie zwróconych danych 31
 - 2.4.3. Testowanie zwróconego rezultatu 32
 - 2.5. Struktura aplikacji w ASP.NET MVC 4 33
 - 2.6. Podsumowanie 37

Rozdział 3. Kontrolery	39
3.1. Akcje i parametry	41
3.2. Model binding	44
3.3. Filtry	46
3.4. Wbudowane atrybuty	49
3.4.1. NonAction	49
3.4.2. Atrybut ActionNameAttribute	49
3.4.3. Atrybut AcceptVerbsAttribute	50
3.5. Podsumowanie	50
Rozdział 4. Widoki	53
4.1. Przekazywanie danych do widoków	54
4.2. Silniki widoków: Razor i ASPX	57
4.3. Definicja szablonu widoku	60
4.4. Metody pomocnicze HTML	61
4.5. Helpery Razor	63
4.6. Podsumowanie	63
Rozdział 5. Modele	65
5.1. Entity Framework	66
5.1.1. Konwencje i klasa kontekstowa w Code First	67
5.2. Walidacja i adnotacje danych	68
5.3. Asocjacje (związki)	70
5.3.1. Związek jeden-do-jednego	70
5.3.2. Związek jeden-do-wielu (opcjonalny) i związek jeden-do-wielu (obligatoryjny)	71
5.3.3. Związek wiele-do-wielu	71
5.3.4. Związek wiele-do-wielu, klasa asocjacyjna z dodatkowymi atrybutami	72
5.3.5. Generalizacja	72
5.4. Podsumowanie	73
Rozdział 6. Zagadnienia zaawansowane	75
6.1. Routing, czyli przekierowania	75
6.1.1. Deklaracja tras	76
6.1.2. Przykład deklaracji własnej trasy	77
6.1.3. Używanie trasowania do generowania linków	79
6.2. AJAX	80
6.2.1. AJAX Helpers	80
6.3. Entity Framework + Database Migrations	83
6.3.1. Podejście Code First	84
6.3.2. Database Migrations	87
6.4. Mechanizm grupowania i minimalizacji	90
6.4.1. Wpływ grupowania i minimalizacji	91
6.4.2. Kontrola i konfiguracja w projekcie MVC 4	92
6.5. Podsumowanie	94
Rozdział 7. Studium przypadku: serwis Subskrypcja	95
7.1. Biznesowa wizja systemu	95
7.2. Projektowa wizja systemu	96
7.3. Przygotowanie projektu	97
7.4. Przygotowanie bazy danych	99
7.5. Dodanie modelu	101
7.6. Utworzenie kontrolera Subscriber	104
7.7. Dodanie widoków dla kontrolera Subscriber	109
7.8. Modyfikacja routingu oraz dostosowanie strony wzorcowej	119

7.9. Dodanie modelu Newsletter	121
7.10. Utworzenie kontrolera Newsletter	122
7.11. Dodanie widoków dla kontrolera Newsletter	125
7.12. Utworzenie użytkownika Administrator	127
7.13. Utworzenie modelu Administrator	130
7.14. Utworzenie kontrolera Administrator	130
7.15. Dodanie widoków dla kontrolera Administrator	132
7.16. Instalacja aplikacji na serwerze WWW	136
7.17. Podsumowanie	137
Rozdział 8. Studium przypadku: serwis Portal Usług	139
8.1. Biznesowa wizja systemu	139
8.2. Projektowa wizja systemu	140
8.3. Projekt aplikacji	142
8.3.1. Projektowanie modelu	142
8.3.2. Projektowanie kontrolerów	145
8.4. Przygotowywanie projektu	146
8.5. Przygotowywanie bazy danych	146
8.6. Przygotowywanie modelu	147
8.7. Przygotowywanie repozytorium	156
8.8. Rejestrowanie użytkowników	169
8.8.1. Zmiana kontrolera Account	171
8.8.2. Zmiana widoków kontrolera Account	182
8.8.3. Dodanie komponentu CAPTCHA	189
8.8.4. Implementacja mechanizmu komunikatów	193
8.9. Funkcjonalność dodawania i edycji kategorii usług i kategorii komentarzy	194
8.9.1. Dodawanie i edycja kategorii usług	194
8.9.2. Dodawanie i edycja kategorii komentarzy	201
8.10. Funkcjonalność wysyłania newslettera	204
8.11. Utworzenie użytkownika Administrator	208
8.12. Implementacja menu	209
8.13. Funkcjonalność usługobiorcy	214
8.13.1. Lista usługobiorców	216
8.13.2. Edycja usługobiorców	225
8.13.3. Usuwanie usługobiorców	229
8.14. Funkcjonalność usługodawcy	232
8.14.1. Lista usługodawców	235
8.14.2. Edycja usługodawców	239
8.14.3. Usuwanie usługodawców	242
8.15. Funkcjonalność usług i komentarzy	245
8.15.1. Dodawanie usług	246
8.15.2. Lista usług	251
8.15.3. Edycja usług	260
8.15.4. Usuwanie usług	264
8.15.5. Szczegóły usług	267
8.15.6. Dodawanie komentarzy	275
8.15.7. Edycja i usuwanie komentarzy	279
8.16. Szczegóły usługodawców	282
8.17. Modyfikacja strony głównej	289
8.18. Implementacja widoków mobilnych	291
8.18.1. Dostosowanie strony wzorcowej	292
8.18.2. Widoki mobilne dla usług	294
8.18.3. Widoki mobilne dla wysyłania newslettera	296
8.19. Instalacja aplikacji na serwerze WWW	297
8.20. Podsumowanie	298

Rozdział 9. Studium przypadku: serwis Moje Zdjęcia w Windows Azure	301
9.1. Biznesowa wizja systemu	302
9.2. Projektowa wizja systemu	302
9.3. Moje Zdjęcia — wersja lokalna	304
9.3.1. Tworzenie projektu	304
9.3.2. Dodanie modelu	306
9.3.3. Wygenerowanie kontrolera i widoków	308
9.3.4. Modyfikacja kontrolera	309
9.3.5. Modyfikacje widoków	320
9.3.6. Dodanie metody-akcji i widoku Wyszukaj	328
9.3.7. Dodanie mechanizmu autouzupełniania	332
9.3.8. Pliki konfiguracyjne Windows Azure	335
9.4. Moje Zdjęcia — wersja w chmurze	337
9.4.1. Zakładanie subskrypcji Windows Azure	337
9.4.2. Zakładanie usług Cloud Service, Azure Storage i bazy SQL	340
9.4.3. Pobranie poświadczeń do bazy SQL i Azure Storage	342
9.4.4. Modyfikacja plików konfiguracyjnych	343
9.4.5. Wdrożenie serwisu do chmury	344
9.5. Podsumowanie	346
Rozdział 10. Studium przypadku: serwis Planner	349
10.1. Własności ASP.NET Web API	349
10.2. Tworzenie projektu ASP.NET Web API	351
10.2.1. Biznesowa wizja systemu	351
10.2.2. Przygotowanie projektu	351
10.2.3. Struktura projektu ASP.NET Web API	353
10.2.4. Implementacja modelu Task i bazy danych	355
10.2.5. Implementacja repozytorium	357
10.2.6. Implementacja kontrolera Tasks	359
10.2.7. Testowanie akcji kontrolera Tasks	362
10.2.8. Implementacja interfejsu użytkownika	366
10.3. Instalacja aplikacji na serwerze WWW	377
10.4. Podsumowanie	378
Rozdział 11. Podsumowanie	379
Dodatek A. Bibliografia	381
Książki	381
Zasoby internetowe	381
Skorowidz	383

Rozdział 3.

Kontrolery

Kontroler to zasadniczo pośrednik pomiędzy działaniami użytkownika a danymi zawartymi w modelu. Pośrednik, który obsługując żądania HTTP, może dokonywać zmian w modelu i uruchamiać odpowiedni widok z prezentacją danego fragmentu modelu.

Zgodnie z konwencją obowiązującą w ASP.NET MVC klasa, która jest kontrolerem, musi mieć nazwę zakończoną przyrostkiem `Controller` oraz implementować przynajmniej interfejs `IController` — listing 3.1.

Listing 3.1. Interfejs `IController`

```
public interface IController
{
    void Execute(RequestContext requestContext);
}
```

Interfejs `IController` jest bardzo prosty i zawiera tylko jedną metodę `Execute`. Wewnątrz tej metody można dostać się do obiektów takich jak `HttpContext`, `Request` czy `Response`. Taki sposób definiowania kontrolera jest rzadko użyteczny, bo w ten sposób pomija się wszystkie udogodnienia frameworka związane np. z bezpieczeństwem, wiązaniem modelu (ang. *model binding*) czy też wynikami akcji. Traci się także możliwość definiowania metod akcji — wszystkie żądania przychodzące do tego kontrolera są obsługiwane centralnie przez metodę `Execute`. Z tych powodów w praktyce nigdy nie implementuje się bezpośrednio interfejsu `IController`. Podejście praktyczne to dziedziczenie z jednej z dwóch klas bazowych — `ControllerBase` i `Controller`.

Framework ASP.NET MVC 4 posiada abstrakcyjną klasę, która jest klasą bazową dla wszystkich kontrolerów MVC. Klasa `ControllerBase` bezpośrednio implementuje interfejs `IController` oraz dostarcza dodatkowe funkcjonalności, np. odpowiada za tworzenie obiektu `ControllerContext`, dostarcza właściwość `ViewData` etc. `ControllerBase` nie dostarcza jednak innych wysoce pożądaných funkcjonalności, takich jak np. metody akcji. Te i inne funkcjonalności zawiera dopiero abstrakcyjna klasa `Controller`. Dostarcza ona trzy kluczowe funkcje: **akcje** (ang. *Action Methods*), **wyniki akcji** (ang. *Action Results*) oraz **filtry** (ang. *Filters*).

Akcje umożliwiają rozszerzenie zachowania kontrolera. Stosowanie pojedynczej akcji `Execute`, którą oferuje interfejs `IController`, jest mało wygodne. Przy zaimplementowaniu klasy kontrolera dziedziczącego po `Controller` każda akcja jest ściśle powiązana z adresem URL oraz parametrami przekazywanymi do niej, a wydobytymi z przychodzącego żądania.

Wyniki akcji umożliwiają zwracanie różnych bytów z poziomu wykonywanej akcji, np. wygenerowanie widoku, przekierowanie do innej akcji czy zwrócenie jakiegoś pliku. Taka jawna separacja wyników zwracanych przez akcję upraszcza przeprowadzanie testów jednostkowych.

W praktyce właśnie na podstawie klasy `Controller` tworzone są wszystkie klasy kontrolerów używanych w aplikacjach opartych na frameworku ASP.NET MVC 4.

Podczas projektowania aplikacji zachodzi zwykle konieczność ustalenia ziarnistości kontrolerów. W tym procesie wszystko zależy od projektanta. Jeżeli projektant przewidzi w aplikacji jeden kontroler, odpowiednio organizując w nim akcje, to aplikacja może działać prawidłowo. Wadą takiego podejścia jest jednak zwykle duży przyrost akcji oraz metod, które będą musiały obsługiwać żądania z szerokiej grupy klas biznesowych.

Bardziej wskazanym podejściem jest utworzenie kontrolerów w liczbie występujących w systemie klas biznesowych, które wymagają bezpośrednich operacji na danych. Wtedy w każdym z tych kontrolerów tworzy się zestaw akcji, które umożliwiają utworzenie, odczyt, modyfikację oraz usunięcie danych dla konkretnej klasy biznesowej.

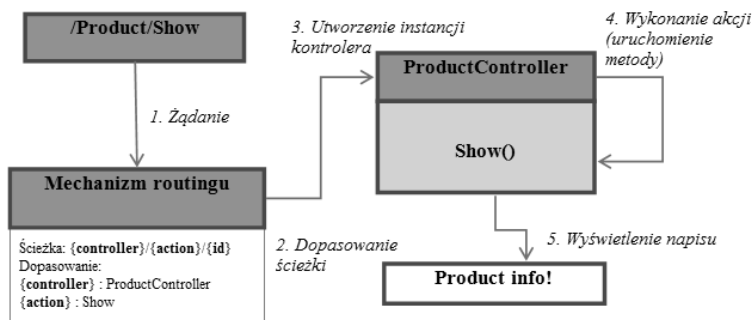
Na listingu 3.2 przedstawiono fragment kodu, za pomocą którego utworzono kontroler `Product`.

Listing 3.2. *Przykładowy kontroler*

```
public class ProductController : Controller
{
    public void Show()
    {
        Response.Write("Product info!");
    }
}
```

W klasie kontrolera `Product` dodano metodę `Show`, której wynikiem wykonania jest wyświetlenie napisu *Product info*. Na rysunku 3.1 przedstawiono sposób, w jaki odbywa się wykonanie akcji `Show` w kontrolerze `Product`. W pierwszym kroku użytkownik aplikacji — poprzez interakcję z widokiem — przekazuje do aplikacji żądanie */Product/Show*. System dopasowuje otrzymane żądanie do wzorca ścieżki przy użyciu mechanizmu routingu. W trzecim kroku, kiedy ścieżka została dopasowana, system tworzy instancję kontrolera `ProductController` (odeczytanego z żądania), po czym wykonuje akcję (poprzez uruchomienie metody) `Show`, która również odczytana została z żądania HTTP. Ostatnim krokiem jest wyświetlenie tekstu.

Rysunek 3.1.
Tworzenie
kontrolera
i uruchamianie akcji



3.1. Akcje i parametry

Domyślnie w aplikacji ASP.NET MVC ścieżka URL rozbijana jest — przy użyciu mechanizmu routingu — na trzy główne elementy: `/ {controller} / {action} / {id}`, gdzie `controller` to nazwa kontrolera, `action` oznacza nazwę akcji, a `id` to parametr dla akcji.

W kontrolerze może znajdować się wiele akcji (metod publicznych), z których każda może (ale nie musi) przyjmować parametry.

Listing 3.3 przedstawia zmodyfikowaną wersję kontrolera z poprzedniego przykładu (listing 3.2), która zawiera jeden parametr `id`.

Listing 3.3. Akcja i parametr

```
public class ProductController : Controller
{
    public void Show(string id)
    {
        Response.Write("Product" + id + "info!");
    }
}
```

Teraz gdy nastąpi żądanie: `/Product/Show/TVsets`, w odpowiedzi wyświetlony zostanie napis: `Product TVsets info!`. Taki sam rezultat zostanie osiągnięty, jeśli żądanie będzie wyglądało tak: `/Product/Show?id=TVsets`. Łatwo zauważyć, że pierwszy sposób jest wygodniejszy dla użytkownika. Takie przekazywanie parametru możliwe jest dzięki odpowiednio skonfigurowanemu mechanizmowi routingu, który sam zajmuje się przekazywaniem metodzie parametru odczytanego z adresu.

Oczywiście jest możliwość przekazywania większej liczby parametrów. Ilustracją jest zmodyfikowany kontroler z poprzedniego przykładu — listing 3.4.

Listing 3.4. Akcja i większa liczba parametrów

```
public class ProductController : Controller
{
```

```
public void Show(String p1, String p2, int p3)
{
    Response.Write("[id:" + p3 + "] name:" + p1 + " info " + p2);
}
```

Jeśli teraz do aplikacji trafi żądanie: */Product/Show?p1=TVsets&p2=test&p3=1*, wówczas wyświetlony zostanie tekst: *[id:1] name: TVsets info test*. Aby adres mógł być bardziej czytelny dla użytkownika, należy zdefiniować odpowiednią ścieżkę w mechanizmie routingu — szczegóły przedstawia listing 3.5.

Listing 3.5. *Przykładowa ścieżka z kilkoma parametrami*

```
routes.MapRoute(
    "productInfo", "Product/Show/{p1}/{p2},{p3}",
    new { Controller = "Product", action = "Show" }
)
```

Przy tak zdefiniowanej ścieżce jak na listingu 3.5 można użyć bardziej przyjaznej dla użytkownika wersji adresu URL, w postaci: */Product/Show/TVsets/test,1*.

W prezentowanych wcześniej przykładach zastosowano tymczasowo metodę `Response.Write`, która umożliwiała wyświetlenie tekstu. Użycie tej metody w znacznym stopniu naruszyło jednak zasadę rozdziału kompetencji ról poszczególnych warstw wzorca MVC. Kontrolery nie mogą zajmować się widokiem danych; dane prezentowane są w bytach zwanych widokami.

Aby zapewnić zgodność z konwencją, akcje w kontrolerach zwracają różnego rodzaju wyniki. W ASP.NET MVC 4 realizacja takiego podejścia następuje przy wykorzystaniu obiektów klasy `ActionResult` — listing 3.6.

Listing 3.6. *Klasa ActionResult*

```
public abstract class ActionResult
{
    protected ActionResult()
    {
    }

    public abstract void ExecuteResult(ControllerContext context);
}
```

Akcja w kontrolerze zwraca obiekt typu `ActionResult`, jednak nie służy on jako kontener dla danych. `ActionResult` jest abstrakcyjną klasą oferującą wspólny interfejs, który wykonuje pewne dalsze działania w imieniu akcji. Poprzez nadpisanie metody `ExecuteResult` pochodna klasy uzyskuje dostęp do wszelkich danych dostarczonych przez wykonanie akcji i wyzwała jakąś późniejszą akcję. Są to kolejne działania związane z wytworzeniem odpowiedzi dla przeglądarki.

Ponieważ klasa `ActionResult` jest abstrakcyjna, każda akcja zdefiniowana w kontrolerze może zwrócić jeden z predefiniowanych typów `ActionResult` — tabela 3.1.

Tabela 3.1. *Typy ActionResult*

Typ	Opis
ContentResult	Zwraca bezpośrednio podaną treść (niekoniecznie HTML) do przeglądarki. Metoda <code>ExecuteResult</code> serializuje przekazywaną treść.
EmptyResult	Nie zwraca niczego do przeglądarki. Metoda <code>ExecuteResult</code> nic nie robi.
FileContentResult	Zwraca zawartość pliku do przeglądarki. Zawartość pliku wyrażona jest jako tablica bajtów. Metoda <code>ExecuteResult</code> przekazuje zawartość do strumienia wyjściowego.
FilePathResult	Zwraca zawartość pliku do przeglądarki. Plik identyfikowany jest poprzez ścieżkę. Metoda <code>ExecuteResult</code> przekierowuje przeglądarkę bezpośrednio do pliku.
FileStreamResult	Zwraca zawartość pliku do przeglądarki. Zawartość pliku reprezentowana jest przez obiekt <code>Stream</code> . Metoda <code>ExecuteResult</code> kopiuje zawartość pliku z dostarczonego strumienia do strumienia wyjściowego.
HttpNotFoundResult	Zwraca kod HTTP 404 do przeglądarki. Podany kod identyfikuje żądanie jako niepowodzenie — żądany zasób nie został odnaleziony.
HttpUnauthorizedResult	Zwraca kod HTTP 401 do przeglądarki. Podany kod wskazuje nieuprawnione żądanie. W efekcie mechanizm uwierzytelniania prosi o zalogowanie.
JavaScriptResult	Zwraca kod JavaScript do przeglądarki. Metoda <code>ExecuteResult</code> zwraca skrypt i ustawia odpowiedni typ. Skrypt będzie wykonany po stronie klienta.
JsonResult	Serializuje obiekty .NET do formatu JSON i wysyła jako odpowiedź do przeglądarki. Metoda <code>ExecuteResult</code> określa typ obiektu, który ma zostać zwrócony, oraz wywołuje klasę <code>JavaScriptSerializer</code> w celu dokonania serializacji dostarczonego obiektu do formatu JSON.
PartialViewResult	Zwraca zawartość HTML do przeglądarki; zwracana zawartość reprezentuje fragment widoku strony.
RedirectResult	Zwraca kod HTTP 302 do przeglądarki w celu jej przekierowania na inny adres URL. Metoda <code>ExecuteResult</code> wywołuje po prostu <code>Response.Redirect</code> .
RedirectToRouteResult	Działa podobnie jak <code>RedirectResult</code> ; w tym przypadku przekierowuje użytkownika na adres URL określony w parametrach routingu.
ViewResult	Zwraca zawartość HTML do przeglądarki; zawartość reprezentuje pełny widok strony.

W tabeli 3.1 można odnaleźć typy do obsługi kodów HTTP, możliwości zwracania plików, przekierowania i wsparcia dla JavaScriptu. Można też uzyskać asynchroniczne przekazywanie danych w formie serializowanej do formatu JSON.

3.2. Model binding

Model binding (wiązanie modelu) to mechanizm odwzorowywania danych przychodzących w żądaniu HTTP bezpośrednio na parametry akcji i obiekty *.NET* (model). Mechanizm jest wykorzystywany w tzw. modelu wejściowym danych, czyli takim, który opisuje dane przychodzące do aplikacji. Każde przychodzące żądanie jest rozpatrywane pod kątem kontrolera i akcji, której rezultat jest odpowiedzią na żądanie. Najpierw rozpoznawany jest kontroler, później akcja. Jeśli oba elementy zostaną odnalezione, to odszukiwane są dane oczekiwane przez akcję, a zdefiniowane jako parametry tej akcji.

Działanie mechanizmu *model binding* stanie się jasne, gdy najpierw przeanalizowany zostanie sposób pobierania danych z przychodzącego żądania HTTP. Na listingu 3.7 przedstawiono przykładowy model wraz z akcją *Create*. Akcja ma za zadanie utworzyć nową usługę i dodać ją do systemu.

Listing 3.7. *Pobieranie danych z żądania*

```
public class Service
{
    public int ID { get; set; }
    public String Name { get; set; }
    public String Content { get; set; }
    public DateTime PostedDate { get; set; }
}

public ActionResult Create()
{
    var service = new Service(){
        Name = Request["name"],
        Content = Request["content"],
        PostedDate = DateTime.Parse(Request["postedDate"])
    };
    // ...
}
```

W przykładzie przedstawionym na listingu 3.7 akcja *Create* tworzy nowy obiekt *Service* i wypełnia go danymi bezpośrednio pobranymi z żądania. Ponieważ dane odczytywane z żądania są ciągiem znaków, to dla każdego pola z modelu *Service*, który nie jest typu *string*, należy pobrać wartość przekształcić na odpowiedni typ (jak w przypadku pola *PostedDate*).

Tworzenie akcji w taki sposób jest bardzo kłopotliwe i nieefektywne. W przypadku wykorzystania w akcji obiektu, który będzie miał wiele pól różnych typów, samo tworzenie obiektu będzie bardzo skomplikowane i przesłoni główny cel akcji, jakim jest — w powyższym przykładzie — dodanie nowej usługi do systemu.

W celu uniknięcia takich mało efektywnych metod można tworzyć akcje inaczej. Zamiast pobierać dane bezpośrednio z żądania można deklarować je jako parametry akcji — listing 3.8.

Listing 3.8. Pobieranie danych z parametrów akcji — typy proste

```
public ActionResult Create(string title, string content, DateTime postedDate)
{
    var service = new Service(){
        Title = title,
        Content = content
        PostedDate = postedDate
    };
    // ...
}
```

W przypadku akcji `Create` z listingu 3.8 zadziała mechanizm *model binding*. Przed uruchomieniem akcji `Create` dwa ważne elementy, `ValueProviderDictionary` oraz `DefaultModelBinder`, wykonają pewną pracę. `ValueProviderDictionary` pobierze wartości z żądania HTTP i zapisze je jako ciągi znaków. Pobieranie odbywa się kolejno z następujących miejsc (na przykładzie parametru `string title`):

- ◆ `Request.Form["title"]`; jeśli element nie istnieje, to:
- ◆ `RouteData.Values["title"]`; jeśli element nie istnieje, to:
- ◆ `Request.QueryString["title"]`; jeśli element nie istnieje, to zwraca `null`.

Po pobraniu z żądania odpowiednich wartości przez `ValueProviderDictionary` klasa `DefaultModelBinder` bierze na siebie odpowiedzialność przekształcenia tych ciągów znaków na odpowiednie obiekty *.NET*.

Tak się dzieje, jeśli parametry oparte są o typy proste. Sytuacja się nieco komplikuje, gdy parametrem jest obiekt złożony (model), tak jak na listingu 3.9. Wtedy z pomocą przychodzi mechanizm **refleksji** (ang. *reflection*). Przy udziale refleksji wpisywane są — do wszystkich publicznych właściwości typu złożonego — wartości pobrane przez klasę `ValueProviderDictionary`. Warto wspomnieć również o tym, że wszystkie publiczne właściwości typu złożonego, które również są typem złożonym, w ten sam sposób otrzymują wartość.

Listing 3.9. Pobieranie danych z parametrów akcji — typ złożony

```
public ActionResult Create(Service service)
{
    // ...
}
```

Czasami zachodzi potrzeba, aby niektóre pola złożonego typu (modelu) wyłączyć z mechanizmu *model binding*. Do tego celu służy atrybut `Bind`. Można go wykorzystać na dwa sposoby: lokalnie w akcji — listing 3.10 lub globalnie w klasie modelu — listing 3.11. Przy wykorzystaniu atrybutu `Bind` można użyć dwóch parametrów `Include` lub `Exclude`, które odpowiednio włączają wymienione właściwości modelu do mechanizmu *model binding* lub wyłączają właściwości modelu z tego mechanizmu.

Listing 3.10. Lokalny sposób użycia atrybutu *Bind*

```
public ActionResult Create([Bind(Include = "Title,Content,PostedDate")] Service service)
{
    // Mechanizm model binding zadziała tylko dla właściwości Title, Content i PostedDate
    // ...
}

// lub
public ActionResult Create([Bind(Exclude = "ID, PostedDate")] Service service)
{
    // mechanizm model binding nie zadziała dla właściwości ID.
    // ...
}
```

Listing 3.11. Globalny sposób użycia atrybutu *Bind*

```
[Bind(Include = "Name,Content,PostedDate")]
public class Service
{
    public int ID { get; set; }
    public String Name { get; set; }
    public String Content { get; set; }
    public DateTime PostedDate { get; set; }
}

// lub
[Bind(Exclude = "ID")]
public class Service
{
    public int ID { get; set; }
    public String Name { get; set; }
    public String Content { get; set; }
    public DateTime PostedDate { get; set; }
}
```

Podane przykłady użycia mechanizmu *model binding* pokazują, że wykorzystanie tego narzędzia pozwala na tworzenie bardziej czytelnego kodu aplikacji, a także na jego znaczne skrócenie.

3.3. Filtry

Mechanizm filtrów w ASP.NET MVC 4 umożliwia wstrzyknięcie dodatkowej logiki do potoku przetwarzania żądania. Umożliwia to prosty i elegancki sposób realizacji przekrojowych problemów dotyczących logowania, autoryzacji czy buforowania. Na listingu 3.12 przedstawiono fragment definicji kontrolera zawierającego filtry.

Listing 3.12. Przykładowy kontroler z filtrami

```
[Authorize(Roles = "role")]
public class ExampleController : Controller
{
    [ShowMessage]
    [OutputCache(Duration = 60)]
```

```
public ActionResult Index()
{
    // ... ..
}
}
```

Filtry zapisuje się w formie atrybutów (pochodzą z przestrzeni nazw `System.Attribute`) i mogą być dołączane do innych elementów kodu, w tym klas, metod, właściwości i pól. Celem ich użycia jest dostarczenie dodatkowych informacji do skompilowanego kodu. W języku C# elementy te umieszcza się w nawiasach kwadratowych, które dodatkowo mogą zawierać parametry, np. `[MyAttribute(SomeProperty = value)]`.

Przykład z listingu 3.12 zawiera kilka filtrów. Pierwszy w postaci `[Authorize(Roles = "role")]` oznacza, że użytkownik, który może korzystać z tego kontrolera, musi być autoryzowany i dodatkowo musi występować w roli `role`. Ten filtr obejmuje cały kontroler wraz ze wszystkimi akcjami.

Filtr zapisany w postaci `[OutputCache(Duration = 60)]` dotyczy tylko akcji `Index` i oznacza uruchomienie buforowania wyników tej akcji przez okres czasu wyrażony w sekundach i podany w parametrze `Duration`. Domyślnie dane przechowywane są w trzech miejscach: na serwerze WWW, na wszystkich serwerach proxy oraz w przeglądarce internetowej.

Framework ASP.NET MVC 4 obsługuje różne rodzaje filtrów:

- ♦ **Filtry zezwoleń** (*Authorization*) — filtr implementuje interfejs `IAuthorizationFilter` i służy do ograniczania dostępu do kontrolerów i akcji kontrolerów. Przykładami tego filtra są klasy `AuthorizeAttribute` oraz `RequireHttpsAttribute`. Filtr `Authorization` działa przed wszystkimi innymi filtrami.
- ♦ **Filtry akcji** (*Action*) — filtr implementuje interfejs `IActionFilter` i „opakowuje” działanie akcji kontrolera. Interfejs `IActionFilter` deklaruje dwie metody: `OnActionExecuting` i `OnActionExecuted`. `OnActionExecuting` uruchamiana jest przed wykonaniem akcji i może dostarczać akcji dodatkowych danych lub sprawdzać poprawność wprowadzanych danych. Metoda `OnActionExecuted` uruchamiana jest natomiast po wykonaniu akcji i może służyć do kontroli danych wynikowych lub anulowania wykonywania rezultatu akcji. Przykładem takiego filtru jest klasa `ActionFilterAttribute`.

Filtry rezultatu (*Result*) — filtr implementuje interfejs `IResultFilter` i „opakowuje” obiekt wyniku działania akcji — `ActionResult`. Interfejs `IResultFilter` deklaruje dwie metody: `OnResultExecuting` i `OnResultExecuted`. Metoda `OnResultExecuting` uruchamiana jest przed zwróceniem rezultatu akcji (np. przed wyświetleniem widoku lub przekierowaniem do innej akcji), a `OnResultExecuted` działa po zwróceniu wyniku akcji. Obie metody mogą wprowadzać dodatkowe przetwarzanie rezultatu akcji, np. mogą modyfikować odpowiedź HTTP. Przykładem takiego filtru jest klasa `OutputCacheAttribute`.

- ◆ **Filtry wyjątku** (Exception) — filtr implementuje interfejs `IExceptionHandler` i wykonywany jest, gdy podczas przetwarzania akcji wystąpi nieobsługiwany wyjątek. Filtr ten może być wykorzystywany do rejestrowania lub wyświetlania strony błędu. Przykładem implementacji filtra wyjątku jest klasa `HandleErrorAttribute`.

W celu zdefiniowania filtra wystarczy utworzyć klasę, która będzie implementowała jeden z wymienionych wyżej interfejsów, a w metodach tego interfejsu zdefiniować dodatkowe działania. Przykładem takiej implementacji może być atrybut `CaptchaValidatorAttribute` wykorzystany w aplikacji *Portal Usług* (rozdział 8.).

Definicję `CaptchaValidatorAttribute` przedstawiono na listingu 3.13. Klasa atrybutu dziedziczy po klasie `ActionFilterAttribute`. Nazwa klasy musi być — według przyjętej we frameworku ASP.NET MVC konwencji — zakończona przyrostkiem *Attribute*. Filtr nadpisuje metodę `OnActionExecuting`, która uruchamiana jest przed wykonaniem akcji. Metoda sprawdza wartość przekazywaną w modelu (`model.Captcha`) i na jej podstawie dodaje do akcji nowy parametr (`captchaValid`), który można odczytać podczas wykonywania akcji — listing 3.14.

Listing 3.13. Przykładowa implementacja filtra akcji

```
public class CaptchaValidatorAttribute : ActionFilterAttribute
{
    public override void OnActionExecuting(ActionExecutingContext filterContext)
    {
        RegisterModel model = filterContext.ActionParameters["model"] as RegisterModel;

        if (filterContext.HttpContext.Session["Captcha"] == null ||
            ↪filterContext.HttpContext.Session["Captcha"].ToString() != model.Captcha)
        {
            filterContext.ActionParameters["captchaValid"] = false;
        }
        else
        {
            filterContext.ActionParameters["captchaValid"] = true;
        }

        base.OnActionExecuting(filterContext);
    }
}
```

Listing 3.14. Wykorzystanie atrybutu `CaptchaValidator`

```
[CaptchaValidator]
[AllowAnonymous]
[HttpPost]
public ActionResult Register(RegisterModel model, bool captchaValid)
{
    if (captchaValid)
    {
        ...
    }
}
```

Podczas wytwarzania prostych aplikacji nie trzeba definiować własnych filtrów. Framework ASP.NET MVC 4 dostarcza cztery podstawowe filtry zaimplementowane w postaci atrybutów. Filtry te mogą być stosowane na poziomie akcji kontrolerów, kontrolerów lub na poziomie aplikacji:

- ♦ `AuthorizeAttribute` — ogranicza dostęp.
- ♦ `HandleErrorAttribute` — atrybut określający, jak obsługiwać wyjątek, który wystąpi podczas przetwarzania (filtr wymaga, aby w pliku konfiguracyjnym *Web.config* element `CustomErrors` miał wartość `true`).
- ♦ `OutputCacheAttribute` — zapewnia buforowanie wyjścia.
- ♦ `RequireHttpsAttribute` — wymusza ponowne przesłanie żądania przez HTTPS, gdy żądanie przyszło z HTTP.

3.4. Wbudowane atrybuty

Framework ASP.NET MVC udostępnia wbudowane atrybuty, takie jak `NonActionAttribute`, `ActionNameAttribute` oraz `AcceptVerbsAttribute`.

3.4.1. NonAction

Wszystkie publiczne metody definiowane w kontrolerach to akcje, które użytkownik może wywołać, wpisując odpowiedni adres URL w przeglądarce. Czasami jednak zachodzi potrzeba zdefiniowania publicznej metody, która nie będzie dostępna dla użytkownika, a widzialność tej metody nie może zostać zmieniona. Aby ograniczyć bezpośredni dostęp do metody z przeglądarki, wystarczy opatrzyć ją atrybutem `NonActionAttribute` — listing 3.15.

Listing 3.15. Zastosowanie atrybutu `NonActionAttribute`

```
[NonAction]
public void Test()
{
    // logika metody
}
```

Podczas próby wywołania z przeglądarki metody opatrzonej atrybutem `NonAction` framework nie uruchomi metody, tylko w odpowiedzi zwróci kod błędu 404.

3.4.2. Atrybut `ActionNameAttribute`

Zastosowanie atrybutu `ActionNameAttribute` pozwala zmienić nazwę akcji. W przykładzie przedstawionym na listingu 3.16 zdefiniowane są dwie metody: `Delete` oraz `DeleteConfirmed`. Poprzez opatrzenie atrybutem `ActionName("Delete")` metody `Delete` ➔ `Confirmed` zostanie ona wywołana przy żądaniu POST akcji `Delete`.

Listing 3.16. *Zastosowanie atrybutu `ActionNameAttribute`*

```
[HttpGet]
public ActionResult Delete(int id)
{
    var model = repository.Find(id);

    // Wyświetla widok potwierdzający usunięcie rekordu.
    return View(model);
}

[HttpPost, ActionName("Delete")]
public ActionResult DeleteConfirmed(int id)
{
    repository.Delete(id);

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

3.4.3. Atrybut `AcceptVerbsAttribute`

Atrybut `AcceptVerbsAttribute` służy do związania akcji z metodą protokołu HTTP. Metody protokołu HTTP, które można związać tym atrybutem z akcjami, to GET, POST, PUT i DELETE. Przykładowo aby akcja `Test()` przekazywała wynik metodą POST, należy opatrzyć ją atrybutem jak na listingu 3.17.

Listing 3.17. *Zastosowanie atrybutu `AcceptVerbsAttribute`*

```
[AcceptVerbs(HttpVerbs.Post)]
public ActionResult Test()
{
    return ViewA()
}
```

Framework dostarcza również cztery skrócone formy tego atrybutu w postaci identyfikatorów: `HttpDeleteAttribute`, `HttpGetAttribute`, `HttpPostAttribute`, `HttpPutAttribute`.

3.5. Podsumowanie

We frameworku ASP.NET MVC kontroler to klasa, która w odpowiedzi na żądanie użytkownika zwraca odpowiedź. Kontroler może wydawać dowolny zbiór poleceń do modelu, a użytkownikowi (przeglądarce) zwrócić dowolny widok. Każdy kontroler zawiera jedną lub więcej akcji, które w istocie są specjalnymi metodami służącymi do przekazywania różnego rodzaju wyników. Predefiniowane typy `ActionResult` zostały zestawione w tabeli 3.1.

Z kontrolerami związany jest *model binding*, czyli mechanizm odwzorowania danych przychodzących w żądaniu HTTP bezpośrednio na parametry akcji i obiekty *.NET* (model). Każde przychodzące żądanie jest rozpatrywane pod kątem kontrolera i akcji, której

rezultat jest odpowiedzią na żądanie. Framework najpierw poszukuje odpowiedniego kontrolera, później akcji. Jeśli oba elementy zostaną odnalezione, to odszukiwane są dane oczekiwane przez akcję, a zdefiniowane jako parametry tej akcji.

Akcje są metodami publicznymi klasy kontrolera. Jeśli zachodzi konieczność wykluczenia publicznej metody kontrolera jako akcji, to należy ją opatrzyć atrybutem `NonActionAttribute`, co powoduje skutki opisane w punkcie 3.4. To jest jeden z trzech wbudowanych atrybutów, nazywanych również filtrami akcji. Dwa pozostałe to `ActionNameAttribute` i `AcceptVerbsAttribute`. `ActionNameAttribute` służy do prostej zmiany nazwy akcji. Atrybut drugi służy do związania akcji z metodą protokołu HTTP.

Filtry mogą być stosowane do indywidualnych akcji lub do całego kontrolera. Służą do rozszerzania logiki, która jest stosowana w aplikacji podczas przetwarzania żądania, ale bez konieczności umieszczania tej logiki w sposobie działania. W efekcie zapis działań jest elegancki i czytelny.

Skorowidz

A

- adnotacja danych, 69
 - DataAnnotations, 121
- AJAX, 80
 - Helpers, 80
 - Ajax.ActionLink, 81
 - Ajax.BeginForm, 80
 - Ajax.BeginRouteForm, 82
 - Ajax.RouteLink, 82
- akcje, 39, 41, 51
 - About, 31
 - CaptchaImage, 190
 - Change, 229, 239
 - Confirm, 179
 - Create, 44, 45, 248, 276, 310
 - Delete, 230, 242, 265, 280, 313, 360, 362
 - testowanie, 365
 - DeleteConfirmed, 313
 - Details, 271, 283, 314
 - Edit, 226, 239, 260, 280, 311
 - Execute, 40
 - filtry, 47
 - Get, 360, 361
 - testowanie, 362
 - Index, 31, 60, 105, 217, 221, 235, 256, 290
 - KupZdjecie, 316
 - List, 106
 - Login, 131, 177
 - LogOff, 131
 - Manage, 180
 - MojeZdjecia, 314
 - PobierzZdjecie, 315
 - Post, 360, 361
 - testowanie, 363
 - Redirect, 32
 - RedirectToLocal, 131
 - Register, 172, 191
 - Send, 123, 205
 - Subscribe, 106, 108
 - Test(), 50
 - Unsubscribe, 107, 108
 - uruchamianie, 41
 - Wyszukaj, 328
- arkusz stylów CSS, 212
- ASP.NET, 23, 24
 - funkcjonalności, 24
 - MVC, 7, 25, 37
 - AcceptVerbsAttribute, 50
 - ActionNameAttribute, 49
 - ASP.NET MVC 4, 7
 - cykl życia żądania, 54
 - deklaracja tras, 76
 - kod źródłowy, 28
 - konfiguracja, 26
 - kontroler, 39, 50
 - konwencja, 26
 - mechanizm routingu, 27
 - metody helperów HTML, 26
 - NonActionAttribute, 49
 - routing, 75
 - rozszerzalność, 26
 - TempData, 57
 - testowalność, 27
 - testowanie, 30
 - ViewBag, 56
 - ViewData, 55
 - założenia projektowe, 25
- MVC 1, 28
- MVC 2, 28
- MVC 3, 28, 55
 - ViewBag, 55
- MVC 4, 7, 23, 27, 29, 379
 - /App_Browsers, 37
 - /App_Data, 33
 - /App_GlobalResources, 37

ASP.NET

MVC 4

- /App_LocalResources, 37
- /App_Start, 33
- /App_Themes, 37
- /Areas, 37
- /bin, 35
- /Content, 35
- /Controllers, 35
- /Filters, 35
- /Global.asax, 36
- /Images, 35
- /Models, 35
- /packages.config, 36
- /Scripts, 36
- /Views, 36
- /Web.config, 36
- AJAX, 80
- Bundling, 90
- ControllerBase, 39
- dodanie roli, 305
- Entity Framework, 66
- filtry, 46
- foldery, 33
- G&M, 90
- główne widoki, 61
- metody pomocnicze HTML, 62
- metody pomocnicze Razor, 63
- Minification, 91
- Moje Zdjęcia, 8, 301
- Planner, 8
- pliki, 33
- Portal Usług, 8, 139
- routing URL, 29
- schemat URL, 29
- strona startowa aplikacji, 99
- struktura aplikacji, 33, 34
- Subskrypcja, 8
- szablon widoku, 60
- szablony, 33
- Web API, 349
- widoki częściowe, 61
- wybór rodzaju projektu, 305
- wybór szablonu projektu, 98, 352
- wybór typu projektu, 98

Web API, 349

- definicja routingu, 355
- dodane zadanie, 375
- planner, 351
- strona główna, 374
- strona startowa, 353
- struktura projektu, 353, 354
- własności, 349

- wybór daty i godziny, 375
- wyszukiwanie zadania, 376
- zastosowanie, 351

- Web Forms, 7, 23, 24, 37
- cechy, 24
- objawy starzenia, 24

- ASPX, 54, 57, 60
- instrukcja warunkowa, 58
- kontrolka logowania, 58
- pętla foreach, 59

atrybut

- AcceptVerbsAttribute, 50
- ActionNameAttribute, 49
- Bind, 45
- NonActionAttribute, 49
- autouzupełnianie, 332
- implementacja, 332

B

biblioteki

- Entity Framework, 66
- jQuery, 209
- MvcContrib, 219
- SimpleMembership, 97, 127
- Bundling, 90
- widok, 93

C

CAPTCHA, 189

- akcja generująca obrazek, 189
- CaptchaValidator, 191
- dodanie pola, 189
- wynik działania, 193
- checkbox, 185
- Code First, 67, 68, 84

D

- Database First, 66

E

EDM, 66

- Entity Framework, 83, 306
- atrybut, 66
- Code First, 66, 68, 84
- dodanie nowych rekordów, 86
- klasy modelu, 84
- modyfikacja rekordów, 86
- usuwanie rekordów, 86
- Database First, 66

Database Migrations, 87
 aktualizacja bazy, 89
 klasa konfiguracji, 87
 migracja, 88
EDM, 66
encja, 66
EntityFramework.SqlServerCompact, 103
ERM, 66
 konfiguracja, 103
 konwencje, 67
 Model First, 66
 ORM, 66, 83
 POCO, 85
 związki, 66, 70
 dziedziczenie, 72
 generalizacja, 72
 jeden-do-jednego, 70
 jeden-do-wielu, 71
 wiele-do-wielu, 71, 72
ERM, 66

F

filtry, 39, 46, 51
 akcji, 47
 implementacja, 48
 AuthorizeAttribute, 49
 CaptchaValidatorAttribute, 48
 definiowanie, 48
 HandleErrorAttribute, 49
 OutputCacheAttribute, 49
 RequireHttpsAttribute, 49
 rezultatu, 47
 rodzaje, 47
 wyjątku, 48
 zezwoleń, 47
funkcja
 @RenderBody(), 116
 CreateAdministrator(), 208
 Fill(), 254
 find(), 373
 GetAllCustomers(), 215
 GetAllServiceProviders(), 233
 GetCommentByServiceId, 270
 GetNewServices, 289
 GetServiceViewModelById, 269
 SeedMembership, 129, 171

G

G&M, 90
 plik konfiguracyjny, 92
 włączenie, 94
 wpływ, 91

H

helper
 Html.ActionLink, 79
 Html.RouteLink, 80
 MailHelper, 123
 Html.TextAreaFor, 125
 Razor, 63
 TextBoxFor, 125

I

interfejs
 IController, 39
 ITaskRepository, 357

J

jQuery, 369

K

klasa
 AccountHelper, 176
 ActionFilterAttribute, 48
 ActionResult, 42, 43
 ApiController, 77, 361
 AuthorizeAttribute, 47
 BundleConfig, 185
 CaptchaValidatorAttribute, 192
 Category, 144, 145
 CategoryRepository, 161
 Comment, 144
 CommentCategory, 144
 CommentRepository, 163
 Controller, 39, 77
 ControllerBase, 39
 Customer, 144
 CustomerRepository, 164
 DbContext, 68
 HandleErrorAttribute, 48
 HttpConfiguration, 355
 InitializeMembership, 128, 170
 LocalPasswordModel, 144
 LoginModel, 130, 144
 MailHelper, 175
 Newsletter, 121, 144
 PortalUslugContext, 153
 RegisterModel, 144
 RequireHttpsAttribute, 47
 RouteConfig, 29
 Service, 144, 145, 148
 ServiceMetaData, 148
 ServiceProvider, 144

klasa

- ServiceProviderRepository, 166
- ServiceRepository, 168
- Subscriber, 101, 102
- SubscriptionContext, 102
- Task, 355
- TaskContext, 356
- TaskRepository, 358
- WebApiConfig, 354
- Zdjecie, 303, 307
- ZdjecieDbContext, 307

kontroler, 39, 50

- Account, 127, 171
 - CAPTCHA, 189
 - zmiana widoków, 182
- ActionResult, 42, 43
- Administrator, 130, 131
 - dodanie widoków, 132
- akcje, 40, 41, 51
- Category, 194
 - akcje, 195
 - modyfikacja, 197
- Comment, 275
 - dodawanie widoków, 277
- CommentCategory, 202
- Controller, 39
- ControllerBase, 39
- Customer, 216
 - dodanie widoku, 217
- filtry, 39, 46, 51
- Home, 289
- IController, 39
- model binding, 44, 50
- Newsletter, 122, 205
 - Authorize, 135, 209
 - dodanie widoków, 125, 206
- przykładowy kontroler, 40
- Service, 246
 - dodanie widoków, 249, 257
- ServiceProvider, 234
- Subscriber, 104
 - akcje, 106
 - Authorize, 135
 - dodanie kontrolera, 105
 - dodanie widoków, 109
 - strona Index, 111
- Tasks, 359, 360
 - akcje, 360
 - implementacja, 359
- tworzenie, 41
- uruchamianie akcji, 41
- ValuesController, 353
- wyniki akcji, 40
- Zdjecia, 308

akcje, 309

modyfikacja, 309

ziarnistość, 40

kontrolka

DataPicker, 375

Html.Grid, 223

Html.Pager, 224

L

LINQ, 68

Mmapowanie obiektowo-relacyjne, *Patrz* ORM

mechanizm grupowania i minimalizacji,

Patrz G&M

mechanizm refleksji, 45

metody

ErrorCodeToString, 180

Execute, 39

GetActiveServicesByUserId, 284

GetEditCategories, 262

Hide(), 185

Index, 77, 79

MapHttpRequest, 77

MapRoute, 30

OnActionExecuted, 47

OnActionExecuting, 47, 48

OnResultExecuted, 47

OnResultExecuting, 47

PobierzKlucze, 333

pomocnicze

HTML, 62

Razor, 63

PrzyjmijPotwierdzenie, 318

Response.Write, 42

Show(), 40, 185

WyslijPotwierdzenie, 317

WyslijZdjecie, 319

Minification, 91

Model First, 67

model związków encji, *Patrz* ERMModel-Widok-Kontroler, *Patrz* MVC

modele, 65

AccountModels, 154

Administrator, 130

adnotacja danych, 69

binding, 44

Bind, 45

DefaultModelBinder, 45

refleksja, 45

ValueProviderDictionary, 45

Category, 149

- Comment, 149
 - CommentCategory, 150, 202
 - Customer, 150
 - CustomerFilterViewModel, 220
 - CustomerListContainerViewModel, 220
 - CustomerViewModel, 215, 216
 - danych encji, *Patrz* EDM
 - domenowy, 65, 68
 - Portal Usług, 144, 147
 - ServiceProvider, 282
 - Subskrypcja, 101
 - klasy, 68
 - domenowa, 68, 70
 - typy asocjacji, 70
 - Newsletter, 153
 - RegisterModel, 182, 185
 - CAPTCHA, 189
 - rodzaje, 66
 - Service, 147
 - ServiceListContainerViewModel, 282
 - ServiceProvider, 151, 233
 - Subscriber, 114
 - Task, 355
 - walidacja danych, 69
 - wejściowy, 65
 - Newsletter, 121
 - widokowy, 65, 214
 - AddServiceModel, 246
 - CommentsFilterViewModel, 268
 - CommentViewModel, 268
 - KupZdjecieViewModel, 317
 - ServiceCommentsViewModel, 269
 - ServiceFilterViewModel, 252
 - ServiceListContainerViewModel, 252
 - ServiceViewModel, 252
 - Zdjecie, 303, 306
 - Moje Zdjęcia, 8, 301, 380
 - autouzupełnianie, 332
 - implementacja, 332
 - biznesowa wizja, 302
 - diagram przypadków użycia, 303
 - dodanie modelu, 306
 - dodawanie kontrolera, 308
 - ekran niezalogowanego użytkownika, 337
 - model danych, 303
 - modyfikacja kontrolera, 309
 - modyfikacje widoków, 320
 - podstawowy widok aplikacji, 309
 - projektowa wizja, 302
 - przeszukiwanie zdjęć, 328
 - Razor, 320
 - szablon strony, 320
 - użytkownicy, 302
 - anonimowy, 303
 - niezalogowany, 337
 - zarejestrowany, 303
 - wersja w chmurze, 337
 - Azure Storage, 341
 - Cloud Service, 341
 - modyfikacja plików konfiguracyjnych, 343
 - wdrożenie serwisu, 344
 - wybieranie listy serwerów, 342
 - zakładanie konta rozliczeniowego, 338
 - zarządzanie kluczami dostępu, 343
 - wybór rodzaju projektu, 305
 - MS SQL Server Compact, 99, 146
 - MVC, 11
 - aplikacje webowe, 17
 - obsługa żądań HTTP, 17
 - schemat działania, 18
 - ASP.NET MVC 4, 23
 - Django, 20
 - interakcja warstw, 12
 - klasyczna postać wzorca, 12
 - Kompozyt, 16
 - Kontroler, 12, 20
 - model, 65
 - domenowy, 65
 - wejściowy, 65
 - widokowy, 65
 - Model, 12, 20
 - MVP, 18
 - MVVM, 19
 - Obserwator, 15
 - PAC, 19
 - Ruby on Rails, 19
 - CoC, 19
 - DRY, 20
 - Spring, 11, 20
 - Strategia, 15, 16
 - Struts, 11, 20
 - szczegółowa komunikacja, 14
 - usytuowanie użytkownika, 13
 - widok, 53
 - Widok, 12, 20
 - wzorec projektowy, 14
 - wzorec złożony, 14
 - Zend Framework, 20
- ## N
- narzędzia
 - @helper, 63
 - Fiddler, 362
 - Microsoft Web Platform Installer, 97
 - NuGet, 103

O

operacje CRUD, 88, 156
ORM, 66, 83

P

PHP, 20

 Zend Framework, 20

Planner, 8, 351, 380

 akcje kontrolera, 360

 arkusz styli CSS, 369

 biznesowa wizja, 351

 definicja wywołania akcji, 371

 dodane zadanie, 375

 dodanie bazy danych, 356

 dodanie nowego kontrolera, 360

 domenowy model aplikacji, 355

 instalacja na zewnętrznym serwerze, 377

 interfejs użytkownika, 366

 jQuery, 369

 przygotowanie projektu, 351

 repozytorium, 357

 interfejs, 357

 klasa, 358

 strona główna, 374

 szablon listy zadań, 372

 testowanie akcji kontrolera, 362

 wybór daty i godziny, 375

 wybór szablonu projektu, 352

 wyszukanie zadania, 376

platforma .NET, 23

 ASP.NET, 23

 MVC 4, 23

 frameworki, 23

POCO, 85

Portal Usług, 8, 139, 379

 akcje, 172

 biznesowa wizja, 139

 byty biznesowe, 142

 CAPTCHA, 189

 checkbox, 185

 diagram przypadków użycia, 143

 dodawanie

 bazy danych, 147

 serwera, 147

 arkusza stylów CSS, 211

 domenowy model aplikacji, 144

 instalacja aplikacji na zewnętrznym

 serwerze, 297

 kategoria komentarzy, 144 150 201

 generowanie kontrolera, 202

 kategoria usług, 194

 dodawanie kategorii, 201

 dodawanie kontrolera, 194

 lista kategorii, 201

 kategoria usługi, 142, 149

 komentarz, 144, 149, 275

 dodawanie komentarzy, 275 279

 dodawanie kontrolera, 275

 edycja komentarzy, 279

 usuwanie komentarzy, 279

 komunikaty, 193

 menu, 209

 administratora, 214

 implementacja, 209

 usługobiorcy, 214

 model domenowy, 144, 147

 newsletter, 144, 153

 dodanie kontrolera, 204

 wysyłanie, 208

 połączenie z lokalną bazą danych, 170

 projekt aplikacji, 142

 projektowa wizja, 140

 projektowanie kontrolerów, 145

 przygotowywanie projektu, 146

 przypadki użycia, 141

 rejestrowanie użytkowników, 169

 repozytorium, 156

 ICategoryRepository, 157

 ICommentRepository, 158

 ICustomerRepository, 158

 interfejs, 156, 169

 IServiceProviderRepository, 159

 IServiceRepository, 160

 strona główna, 289, 294

 modyfikacja, 289

 usługa, 142, 148, 245

 dodanie widoku, 257

 dodawanie usług, 246, 251

 edycja usług, 260, 265

 implementacja kontrolera, 246

 lista usług, 260

 szczegóły usługi, 279

 usuwanie usług, 264

 wyświetlanie listy, 251

 wyświetlanie szczegółów, 267

 usługobiorca, 142, 150, 214

 edycja danych, 225

 formularz rejestracyjny, 186

 implementacji kontrolera, 216

 model widokowy, 214

 usuwanie, 229

 wyświetlanie listy, 216, 225

 usługodawca, 142, 151, 232

 dane szczegółowe, 282, 288

 dodawanie widoku, 236, 240

 edycja danych, 239

- formularz rejestracyjny, 186
- implementacja kontrolera, 234
- lista aktywnych usług, 282, 288
- model widokowy, 232
- usuwanie, 242
- wyświetlanie listy, 235
- użytkownicy, 139
 - Administrator, 141 208
 - niezarejestrowany, 140
 - rejestracja, 139, 169
 - usługobiorca, 140
 - usługodawca, 141
 - zarejestrowany, 141
- użytkownik, 142
- widoki mobilne, 291
 - lista usług, 296
 - newsletter, 296
 - strona główna, 293
 - strona wzorcowa, 292
 - szczegóły usługi, 296
 - tryby wyświetlania, 291
- Python, 20
 - Django, 20

R

- RAD, 23
- Razor, 54, 57, 60
 - instrukcja warunkowa, 58
 - kontrolka logowania, 58
 - metody pomocnicze, 63
 - pętla foreach, 59
- routing, 75
 - modyfikacja, 119
 - trasowanie, 76
 - domyślna trasa przekierowań, 76
 - generowanie linków, 79
 - trasa użytkownika, 78
- URL, 29
 - Application_Start(), 29
 - domyślna konfiguracja, 30
 - MapRoute, 30
 - RouteConfig, 29
- Ruby on Rails, 19
 - CoC, 19
 - DRY, 20

S

- strona wzorcowa, 116
 - dodanie obsługi komunikatów, 193
 - urządzenia mobilne, 292

- studium przypadku, 8
 - Moje Zdjęcia, 8
 - Planner, 8
 - Portal Usług, 8
 - Subskrypcja, 8
- Subskrypcja, 8, 95, 379
 - biznesowa wizja, 95
 - diagram przypadków użycia, 96
 - dodawanie
 - bazy danych, 100
 - modelu domenowego, 101
 - modelu wejściowego, 121
 - nowego kontrolera, 105
 - serwera, 99
 - nowego widoku, 110
- Entity Framework, 102
 - konfiguracja, 103
- formularz
 - dodawania subskrypcji, 114
 - logowania użytkownika, 134
- instalacja aplikacji, 137
- komunikat błędu
 - przy usuwaniu subskrypcji, 117
 - walidacji, 115
- konfiguracja serwera pocztowego, 124
- lista subskrybentów, 118
- modyfikacja routingu, 119
- oferowane działania, 95
- połączenie z bazą danych, 128
- projektowa wizja, 96
- przygotowanie projektu, 97
- strona główna aplikacji, 135
- strona wzorcowa, 116, 119
 - dodanie obsługi komunikatów, 116
 - modyfikacja kodu, 120
 - modyfikacja stopki, 120
 - modyfikacja tytułu, 120
- Subscriber, 102, 105
- SubscriptionContext, 102
- użytkownicy systemu, 96
 - Administrator, 96, 127
 - identyfikacja, 96
 - Subskrybent, 96
- wysyłanie newslettera, 123, 126

T

- tryby wyświetlania, 291
 - definiowanie nowego, 291

V

- Visual Studio 2012, 97
 - uruchamianie, 97

W

widok, 53

 _LoginPartial, 188

 Bundling, 93

 Create, 199, 250, 278, 323

 częściowy, 61

 _Filter, 222, 238, 259

 _Menu, 210

 _Menu.Mobile, 293

 _ServiceFilter, 287

 Delete, 231, 244, 267, 327

 Details, 273, 285, 326

 Edit, 200, 227, 240, 263, 281, 324

 generowanie linków, 79

 główny, 61

 Index, 111, 199, 218, 237, 257, 290, 321, 328, 366

 modyfikacja, 223

 KupZdjecie, 331

 List, 117

 Login, 133, 186

 Manage, 187

 mobilny

 Details.Mobile, 295

 Index.Mobile, 294

 Send.Mobile, 296

 nietypizowany, 110

 PotwierdzenieWyslane, 331

 przekazywanie danych, 54, 55

 Register, 182, 190

 CAPTCHA, 191

 Send, 125, 206

 Sent, 126, 127, 208

 silnik widoku, 53

 ASPX, 54, 57

 Razor, 54, 57

 SprzedazZakonczone, 331

 Subscribe, 112

 szablony, 60

 TempData, 57

 Unsubscribe, 115

 ViewBag, 56

 ViewData, 55

 ViewResult, 54, 55

Windows Azure, 301

 Activity Log, 346

 Cloud Service, 304, 337, 340

 zakładanie usługi, 341

 pliki konfiguracyjne, 335

 pobranie pliku poświadczeń, 345

 ServiceConfiguration.Cloud, 336

 ServiceConfiguration.Local, 335

 ServiceDefinition, 336

 SQL Database, 337, 340

 connection string, 342

 dodawanie nowego serwera, 342

 Storage, 301, 337, 340

 Blobs, 319

 connection string, 343

 zakładanie konta, 341

 zarządzanie kluczami dostępu, 343

 Tools, 301

 wybieranie listy serwerów, 342

 zakładanie konta rozliczeniowego, 338

wyniki akcji, 39

wzorce projektowe, 11

 Kompozyt, 16

 MVC, 11

 MVP, 18

 MVVM, 19

 Obserwator, 15

 PAC, 19

 Repozytorium, 156

 Strategia, 15, 16

Z

zdarzenie

 Application_Start(), 29, 128, 170, 208, 291

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 to technologia, której poznanie pozwala na szybkie tworzenie eleganckich, prostych w utrzymaniu i wyrafinowanych w działaniu średnich i dużych aplikacji webowych. Współpraca z najważniejszymi standardami internetowymi, takimi jak HTML5, CSS, jQuery czy chmura Windows Azure, możliwość projektowania aplikacji dla urządzeń mobilnych i uproszczenie budowy modelu domenowego to tylko próbka najnowszych funkcjonalności, jakie zapewnia wersja 4 tego znakomitego frameworka.

Jeśli chcesz zostać rozchwytywanym specjalistą pracującym z użyciem ASP.NET MVC 4, nie znajdziesz lepszej książki. W części pierwszej opisano powstanie i warianty wzorca MVC, a także warstwy kontrolerów, widoków i modeli. Jej lektura pozwoli Ci poznać zasady działania frameworka, zorientować się w jego strukturze i opanować korzystanie z komponentów zewnętrznych, takich jak jQuery czy AJAX. Natomiast część druga zawiera cztery studia przypadku, czyli szczegółowe omówienie procesu budowania czterech różnych aplikacji korzystających z różnych technologii pomocniczych i osadzonych w różnych środowiskach. Dzięki temu uda Ci się sprawdzić, jak to wszystko działa w praktyce.

ASP.NET MVC 4 + ta książka = sukces w programowaniu!

Model-Widok-Kontroler
Framework ASP.NET MVC 4
Kontrolery
Widoki
Modele
Routing, czyli przekierowania
AJAX
Entity Framework
+ Database Migrations
Studium przypadku:
serwis Subskrypcja
Studium przypadku:
serwis Portal Usług
Studium przypadku:
serwis Moje-Zdjęcia
w Windows Azure
Studium przypadku:
serwis Planner
Bibliografia

helion.pl
księgarnia
internetowa

Nr katalogowy: 13405



Księgarnia internetowa:
<http://helion.pl>



Zamówienia telefoniczne:
0 801 339900



0 601 339900



Helion

Sprawdź najnowsze promocje:

- <http://helion.pl/promocje>
- Książki najchętniej czytane:
• <http://helion.pl/bestsellery>
- Zamów informacje o nowościach:
• <http://helion.pl/nawosci>

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

sięgnij po **WIĘCEJ**



KOD KORZYŚCI

ISBN 978-83-246-6534-1



Cena: 59,00 zł

Informatyka w najlepszym wydaniu

9 788324 665341