

SQL Server 2014
SSL Visual Studio 2013 ORM
MVVM AJAX SEO WCF RWD
LINQ AGILE SCRUM TDD
REST service
EF 6 MVC 5
ASP.NET
Windows Server 2012
JSON
JSON
HTML5 HTTP
HTML5
CSS3
CSS3
WEB API 2.0 Windows AZURE
Google Facebook HTML DOM
NoSQL AJAX MVP
HOW

Krzysztof Żydzik, Tomasz Rak

C# 6.0 i MVC 5

Tworzenie nowoczesnych portali internetowych

Helion 

Autorzy:

Krzysztof Żydzik, Microsoft Certified Solutions Developer, Microsoft Certified Professional, Microsoft Specialist

Tomasz Rak, Politechnika Rzeszowska, Wydział Elektrotechniki i Informatyki, Katedra Informatyki i Automatyki, Rzeszów, Polska

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: Michał Mrowiec

Projekt okładki: Studio Gravite / Olsztyn
Obarek, Pokoński, Pazdrijowski, Zaprucki

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/c6mvc5>

Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

Kody źródłowe kolejnych kroków tworzenia przykładowej aplikacji dostępne są pod adresem:

<ftp://ftp.helion.pl/przyklady/c6mvc5.zip>

ISBN: 978-83-246-9496-9

Copyright © Helion 2015

Printed in Poland.

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

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

Spis treści

Wstęp	15
Rozdział 1. C# — teoria i praktyka	19
Wprowadzenie do języka C#	19
Kolejne wersje języka C#	20
C# 2.0 (.NET Framework 2.0, Visual Studio 2005)	20
C# 3.0 (.NET Framework 3.5, Visual Studio 2008)	20
C# 4.0 (.NET Framework 4.0, Visual Studio 2010)	21
C# 5.0 (.NET Framework 4.5, Visual Studio 2012 oraz 2013)	21
C# 6.0 (zapowiedź)	21
Konwencje	21
Cel stosowania konwencji	22
Pliki a klasy i interfejsy	22
Wcięcia	22
Komentarze	23
Deklaracje klas, interfejsów i metod	24
Puste linie	24
Nawiasy klamrowe	25
Konwencje nazewnictwa	25
Pozostałe dobre praktyki	26
Typy	26
Deklaracja zmiennej	27
Inicjalizacja zmiennej	28
Słowa kluczowe	29
Stałe i zmienne tylko do odczytu	29
Literały	30
Typ wyliczeniowy	31
Konwersje typów i rzutowanie	31
Opakowywanie (boxing) i rozpakowywanie (unboxing)	32
Wartości zerowe oraz typy dopuszczające wartości zerowe	33
Typy generyczne	34
Tablice, łańcuchy i kolekcje	34
Tablice	34
Łańcuchy	36
Kolekcje	37
Operatory	38
Operator trójargumentowy ?:	38
Operator ??	40

Instrukcje sterujące	40
Instrukcja if	40
Instrukcja switch	41
Instrukcje iteracyjne	43
Pętla while	43
Pętla do while	43
Pętla for	44
Pętla foreach	45
Instrukcje skoku	45
Klasy, obiekty, pola, metody i właściwości	46
Klasy	46
Obiekty	48
Pola	48
Metody	49
Właściwości	52
Podstawowe pojęcia związane z programowaniem obiektowym	53
Abstrakcja	53
Hermetyzacja	54
Dziedziczenie	54
Polimorfizm	55
Przeciążanie operatorów	56
Przeciążanie operatorów relacji	57
Metody Equals() i GetHashCode()	57
Przeciążanie operatorów konwersji	57
Przeciążanie operatorów logicznych	58
Przeciążanie operatorów arytmetycznych	60
Przeciążanie metod	61
Indeksatory	61
Klasa System.Object	63
Konstruktor i destruktor	64
Garbage Collector	66
Zasada działania GC	66
Podział na generacje (przechowywanie obiektów w pamięci)	66
Struktury	67
Interfejsy	68
Jawna implementacja interfejsu	69
Zwalnianie zasobów niezarządzanych	70
Interfejs IDisposable	70
Słowo kluczowe using	71
Delegaty, metody anonimowe, wyrażenia lambda i zdarzenia	72
Delegaty	72
Metody anonimowe	74
Wyrażenia lambda	74
Zdarzenia	75
Dyrektywy preprocesora	76
Wyjątki	77
Zgłaszanie wyjątków	77
Przepełnienia arytmetyczne	78
Instrukcje checked i unchecked	78
Przestrzenie nazw	79
Zagnieżdżanie przestrzeni nazw	79
Dyrektywa using	81
Alias	82
Zewnętrzne aliasy	83

Typy, metody, klasy i kolekcje uogólnione (generyczne)	83
Metody generyczne	84
Klasy generyczne	84
Kolekcje generyczne i interfejsy	86
Interfejs IDictionary<TKey, TValue> — słownik	86
Interfejs IEnumerable<T>	87
Interfejs ICollection<>	87
Interfejs IList<>	87
Interfejs IQueryable<>	88
Wyrażenia regularne	89
Data i czas	90
Operacje wejścia, wyjścia, foldery i pliki	92
Pozostałe elementy języka i nowości w wersji C# 5.0	93
Mechanizm refleksji i atrybuty	93
Atrybuty	94
IEnumerable a IEnumerator	96
Iteratory i słowo kluczowe yield return	97
Inicjalizatory obiektów i kolekcji	100
Drzewa wyrażeń	101
Metody rozszerzające	102
Metody i klasy częściowe	103
Metody częściowe	104
Zmienne domniemane	105
Typy anonimowe	105
Słowa kluczowe this i base	106
Typy dynamiczne	107
Argumenty nazwane — Named Arguments	110
Parametry opcjonalne	111
Obsługa kontra- i kowariancji oraz słowa kluczowe in i out	111
Słowa kluczowe is, as i typeof	114
Leniwa inicjalizacja — Lazy Initialization	114
Metody asynchroniczne — async i await	118
Atrybuty Caller Info	119
Nowości w C# 6.0	120
Konstruktory pierwotne — Primary Constructors	120
Automatyczna inicjalizacja właściwości — Initializers for Auto-properties	120
Dyrektywa using dla składowych statycznych — Using Static Members	121
Inicjalizatory słownikowe — Dictionary_INITIALIZER	121
Deklaracje inline dla parametrów out — Inline Declarations for Out Params	122
Wyrażenia dla właściwości — Property Expressions	122
Wyrażenia dla metod — Method Expressions	122
Modyfikator private protected	123
Kolekcje IEnumerable jako parametr — Params for Enumerables	123
Jednoargumentowe sprawdzanie wartości null — Monadic Null Checking	123
Słowo kluczowe await w blokach catch i finally	124
Filtry wyjątków — Exception Filters	124
Literały binarne i separatory cyfr — Binary Literals, Digit Separators	124

Rozdział 2. Wzorce architektoniczne **125**

Architektura wielowarstwowa	125
Architektura jednowarstwowa	126
Architektura dwuwarstwowa	126
Architektura trójwarstwowa	126
Architektura n-warstwowa	126

MVC	127
View	128
Controller	128
Model	128
Domain Model, MVC Model i ViewModel — porównanie	128
Model pasywny a model aktywny	129
MVP	129
Model	130
View	130
Presenter	130
MVVM	131
MVC, MVP i MVVM	131
DDD	132
SOA	132
EDA	133
Rozdział 3. Microsoft .NET Framework	135
Struktura .NET	135
CLI	136
CIL	137
CLR	137
DLR	137
Elementy .NET wykorzystywane w ASP.NET MVC	138
Implementacje .NET	138
Projekt Mono	139
WPF	139
WCF	140
Service Contract	140
Operation Contract	140
Data Contract	140
Data Member	140
WCF Endpoint = adres + binding + contract	142
Silverlight	142
Microsoft Azure	143
Windows Azure Storage	143
BLOB Storage	143
Table Storage	143
Queue Storage	143
Hostowanie aplikacji w Azure	143
Worker Role	144
Web Role	144
Web Site	144
Virtual Machine	144
Azure Service Bus	144
Service Bus Relay	144
Service Bus Queue	145
Service Bus Topic	145
ASP.NET Web Forms	145
ASP.NET Web Pages	146
ADO.NET	146
Obiekt DataSet	147
Obiekty DataTable i DataRow	147
Obiekt DataRelation	147
Obiekt DataView	147
.NET Framework Data Provider	147

LINQ	148
LINQ to XML	148
LINQ to Objects	149
LINQ to SQL	149
LINQ to DataSet	149
LINQ to Entities	149
Przykłady zapytań LINQ	150
Składnia metod — Method Syntax	150
Składnia zapytań — Query Syntax	151
PLINQ	151
Narzędzia ORM w .NET	153
Entity Framework	153
NHibernate	153
NHibernate 3 a Entity Framework 6	154
Alternatywa dla Entity Framework i NHibernate	154
Rozdział 4. Entity Framework 6	157
Podejście do pracy z modelem danych	157
Porównanie różnych podejść	157
Model dla podejścia Model First	158
Model dla podejścia Code First	158
Nowości wprowadzane w kolejnych wersjach EF	159
Nowości wprowadzone w EF 5	159
Nowości wprowadzone w EF 6	160
Relacyjne bazy danych i EF	160
Krótki opis baz relacyjnych	160
Relacja „jeden do wielu”	161
Relacja „jeden do jednego”	161
Relacja „wiele do wielu”	162
Relacje opcjonalne	165
Obiekty DbContext i DbSet	165
DbContext i DbSet	165
Metody Attach i Detach	165
Relacje poprzez klucz FK a relacje niezależne (obiektywne)	166
Relacje poprzez klucz obcy — FK Association	166
Relacje niezależne — Independent Association	167
Odpytywanie bazy danych za pomocą EF i LINQ	168
Wczytywanie zachłanne — Eager Loading	168
Wczytywanie leniwe — Lazy Loading	169
Jawne ładowanie — Explicit Loading	170
Problem N+1	170
Metoda AsNoTracking()	171
Odroczone i natychmiastowe wykonanie	171
Entity SQL	172
Bezpośrednie zapytania SQL do bazy (Direct/RAW SQL) i procedury składowane w EF	173
Transakcje w EF	174
Śledzenie zmian	175
Migawkowe śledzenie zmian — Snapshot Change Tracking	175
Dynamiczne śledzenie zmian — Dynamic Change Tracking (proxy)	175
Zarządzanie operacjami współbieżnymi	176
Kaskadowe usuwanie — Cascade Delete	177

Strategie dziedziczenia w bazie danych — TPT, TPH i TPC	178
TPH	178
TPT	178
TPC	179
SQL Logging	179
Code First Fluent API i Data Annotations	180
Migracje	182
Metoda Seed	183
Rozdział 5. ASP.NET MVC 5	185
Kolejne wersje ASP.NET MVC	185
ASP.NET MVC 1	185
ASP.NET MVC 2	185
ASP.NET MVC 3	186
ASP.NET MVC 4	186
ASP.NET MVC 5	186
ASP.NET MVC 6 (zapowiedź)	187
Konwencje w MVC	187
Struktura projektu	187
Konwencje a ASP.NET MVC	188
MVC Pipeline — ścieżka wywołań, handlers i moduły	189
Ścieżka wywołań	189
Pierwsze żądanie do aplikacji ASP.NET	189
Podstawowe obiekty tworzone dla każdego żądania	189
HttpApplication	190
Uchwyty i moduły HTTP	193
Uchwyty HTTP	193
Moduły HTTP	193
HttpHandler a HttpModule	193
Kontroler	194
Typy rezultatu	194
Parametry akcji	196
Żądanie GET	196
Żądanie POST	196
Filtry akcji	197
Widok	200
Zasady odnajdywania widoków	200
Folder Shared	201
Widoki częściowe	201
Razor	202
Dodatkowe właściwości silnika Razor	203
ViewBag, ViewData i TempData	204
Widoki typowane — Strongly Typed Views	205
HTML helpery	208
Paczki skryptów i minimalizacja — Script/CSS Bundling and Minification	209
Sekcje	211
Routing	214
Kolejność w routingu	214
Ignorowanie ścieżek	214
Ograniczenia	215
Routing na podstawie atrybutów	215
Prefiksy	216
Ograniczenia	217
Nazywanie ścieżek i generowanie linków po nazwie ścieżki	217
Obszary	217

Model	218
ViewModel	218
Walidacja	222
MVC Scaffolding	223
Generowanie kontrolerów	223
Generowanie widoków	227
Metody synchroniczne i asynchroniczne w MVC	228
Słowa kluczowe — Async, Await, Task	230
Cache	231
Cachowanie po stronie serwera — Server Side Caching	231
Atrybut OutputCache	232
Cachowanie częściowe	233
Cachowanie rozproszone	233
Cachowanie po stronie klienta — Client Side Caching	234
Cachowanie w HTML 5	234
HTML 5 Application Cache	234
HTML 5 WebStorage	234
Code First Data annotations	235
Bezpieczeństwo	235
SQL Injection	236
Cross-Site Request Forgery	236
Cross-Site Scripting	237
Over-Posting — parametr binding	237
Obsługa, śledzenie i logowanie wyjątków w MVC	238
Lokalne zarządzanie wyjątkami	238
Blok try-catch	238
Nadpisywanie metody OnException() w kontrolerze	238
Globalne zarządzanie wyjątkami	239
Klasa FilterConfig	239
HandleError na poziomie kontrolerów i akcji	239
Zwracanie widoków dostosowanych do konkretnych typów wyjątków	240
Logowanie globalne za pomocą osobnych narzędzi	240
Identyfikacja, uwierzytelnianie i autoryzacja w MVC 5	241
Identyfikacja	241
Uwierzytelnianie	241
Autoryzacja	242
Role w MVC	242
Stan aplikacji, sesje i ciasteczka	242
Stan aplikacji	242
Ciasteczka	243
Sesje	243
OWIN	244
ASP.NET Identity	244
WIF i uwierzytelnianie za pomocą claimów	245
Identity Provider, STS	246
Strona ufająca — Relying Party	246
Federated Authentication	247
Windows ACS	248
OpenId i OpenAuth	249
OpenId	249
OpenAuth	250

Rozdział 6. Web serwisy i ASP.NET Web API 2	251
Web API 2	251
Web API a ASP.NET MVC	252
Web serwis, REST, SOAP i OData	253
SOAP	253
REST	253
OData	254
CORS i JSONP	255
JSONP	255
CORS	255
Uruchamianie CORS w Web API	256
Routing w Web API	257
Mapowanie żądań na akcje bądź metody w kontrolerze Web API	257
Web API a Entity Framework i warstwa modelu	258
Typy rezultatu w Web API	258
Typ void	259
HttpResponseMessage	259
IActionResult	260
Inny dowolny typ z aplikacji	260
Pobieranie danych z Web API	261
Pobieranie danych po stronie serwera (.NET, C#)	261
Pobieranie danych po stronie klienta (JavaScript, jQuery, AJAX)	261
Wersjonowanie w Web API	262
Rozdział 7. Narzędzia, licencje i ceny	263
Serwer IIS	263
Kategorie dla modułów dostępnych w IIS	263
Pule aplikacji w IIS	264
Przetwarzanie żądań w IIS	264
Microsoft SQL Server 2014	264
Licencjonowanie SQL Server 2014	265
Ceny licencji SQL Server 2014	265
Nowości w SQL Server 2014	266
Windows Server 2012	267
Wersje Windows Server 2012	267
Licencjonowanie Windows Server 2012	267
Ceny Windows Server 2012	268
Microsoft Visual Studio 2013 Ultimate	268
Snippets	269
Page Inspector	269
Nowości w Visual Studio 2013	269
Poprawiony pasek przewijania	270
Podgląd definicji	270
Browser Link	270
JSON Editor i JavaScript	271
Powiązanie z Microsoft Azure	272
Wsparcie dla GIT	272
Najważniejsze skróty klawiszowe	272
Rozdział 8. Aplikacja i wdrożenie	277
Wzorce projektowe i architektoniczne wykorzystywane w .NET	277
Repozytorium	277
Wzorzec IoC	277
Repozytorium generyczne	278
Wzorzec UnitOfWork	278

Przykładowa aplikacja	278
Etap 1. Krok 1. Tworzenie nowego projektu i aktualizacja pakietów	279
Etap 1. Krok 2. Utworzenie modelu danych	283
Klasa <code>Kategoria</code>	286
Klasa <code>Ogloszenie_Kategoria</code>	287
Klasa <code>Uzytkownik</code>	287
Etap 1. Krok 3. Tworzenie klasy kontekstu	290
Etap 1. Krok 4. Przenoszenie warstwy modelu do osobnego projektu	294
Dodawanie referencji pomiędzy projektami	296
Ustawienie projektu startowego	297
Instalacja bibliotek dla nowego projektu	298
Przeniesienie plików z modelem do osobnej warstwy (projektu)	299
Etap 1. Krok 5. Migracje	300
Instalacja migracji	300
Konfiguracja migracji	301
Tworzenie migracji początkowej	302
Uruchomienie pierwszej migracji	305
Metoda <code>Seed()</code>	306
Zmiany w modelu i kolejna migracja	309
Praca z błędami i niespójnością w migracjach	310
Etap 1. Podsumowanie (warstwa modelu i migracje)	311
Etap 2. Krok 1. Dodawanie kontrolerów i widoków — akcja <code>Index</code>	311
Dodawanie kontrolera z widokami	311
Pierwsze uruchomienie aplikacji i routing	316
Lista ogłoszeń (akcja <code>Index</code>) — aktualizacja widoku/wyglądu strony	317
Lista ogłoszeń a pobieranie danych	321
Optymalizacja listy ogłoszeń	322
Etap 2. Krok 2. Debugowanie oraz metody <code>AsNoTracking()</code> i <code>ToList()</code>	324
Sprawdzanie wartości zmiennych	325
Metoda <code>ToList()</code> i odroczone wykonanie (<code>Deferred Execution</code>)	325
Metoda <code>AsNoTracking()</code>	326
Etap 2. Krok 3. Poprawa wyglądu i optymalizacja pod kątem SEO	329
Poprawa wyglądu strony za pomocą <code>Twitter Bootstrap</code>	329
Podświetlanie wierszy za pomocą <code>CSS</code>	330
Optymalizacja pod kątem pozycjonowania — SEO	331
Etap 2. Podsumowanie	333
Etap 3. Krok 1. Poprawa architektury aplikacji	334
Przeniesienie zapytania <code>LINQ</code> do osobnej metody	334
Przeniesienie metody do repozytorium	334
Etap 3. Krok 2. Zastosowanie kontenera <code>Unity</code> — <code>IoC</code>	336
Wstrzykiwanie repozytorium poprzez konstruktor w kontrolerze	336
Tworzenie interfejsu dla repozytorium	337
Instalacja kontenera <code>IoC Unity</code>	338
Wstrzykiwanie kontekstu do repozytorium	340
Cykl życia obiektu a kontener <code>IoC</code>	341
Etap 3. Podsumowanie	341
Etap 4. Krok 1. Akcje <code>Details</code> , <code>Create</code> , <code>Edit</code> , <code>Delete</code>	342
<code>Details</code>	342
Metoda <code>Details()</code> w repozytorium	342
Aktualizacja i optymalizacja SEO dla widoku <code>Details</code>	343
<code>Delete</code>	345
<code>Create</code>	353
<code>Edit</code>	359

Etap 4. Krok 2. Aktualizacja szablonu _Layout.cshtml	365
Etap 4. Krok 3. Widoki częściowe — PartialView	366
Etap 4. Podsumowanie	369
Etap 5. Bezpieczeństwo, uwierzytelnianie i autoryzacja dostępu	369
Uwierzytelnianie i logowanie przez portale	369
Autoryzacja — role	372
Zabezpieczanie akcji	373
Etap 5. Podsumowanie	380
Etap 6. Stronicowanie i sortowanie	381
Stronicowanie	381
Sortowanie	388
Etap 6. Podsumowanie	392
Etap 7. Ogłoszenia użytkownika, kategorie, cache i ViewModel	393
Zakładka Moje ogłoszenia	393
Cache	394
Kategorie	395
Zastosowanie HTML helpera — Html.Action	402
Zastosowanie ViewModel	403
Etap 7. Podsumowanie	406
Etap 8. Dane w JSON, zarządzanie relacją „wiele do wielu” i attribute routing	407
PartialView a dane w formacie JSON lub XML	407
Użycie attribute routing	407
Zarządzanie relacją „wiele do wielu” i autocomplete	409
Dodatek na ASP.NET MVC	409
Etap 8. Podsumowanie	410
Etap 9. Dodatek — tworzenie modelu dla podejścia Model First	410
Publikacja systemu na zewnętrznym serwerze hostingowym	415
Dodawanie domeny	416
Konfiguracja witryny	418
Tworzenie bazy danych	421
Tworzenie konta FTP	422
Połączenie z bazą danych poprzez SQL Server Management Studio	422
Wdrażanie aplikacji na serwer za pomocą Microsoft Visual Studio	423
Dodatek A Zasady i metodologie w programowaniu	427
Zasady	427
SOLID	427
Zasada pojedynczej odpowiedzialności (SRP)	427
Zasada otwarte-zamknięte (OCP)	428
Zasada podstawienia Liskov (LSP)	428
Zasada separacji interfejsów (ISP)	429
Zasada odwrócenia zależności (DIP)	431
GRASP	432
Creator	433
Information Expert	433
Controller	433
Low Coupling	433
High Cohesion	434
Polymorphism	434
Pure Fabrication	434
Indirection	435
Protected Variations	435
DRY	435
KISS	436

Rule of Three	436
Separation of Concern	436
YAGNI	437
MoSCoW	437
Metodologie	437
Manifest Agile	437
Scrum	439
eXtreme Programming	439
TDD	440
Dodatek B HTTP i SSL/TLS	443
HTTP	443
SSL/TLS	447
Rodzaje certyfikatów	449
Zakup certyfikatu SSL	450
Aktywacja, walidacja i instalacja certyfikatu SSL	450
Certyfikat w praktyce	450
Dodatek C HTML 5 i CSS 3	453
HTML 5	453
Sekcje	457
Nowe typy pól formularza	458
Atrybuty dla formularza	459
Znaczniki	459
Web Storage	461
Server Side Events	461
WebSockets	462
Drag and Drop	463
Geolokalizacja	465
Walidacja	466
CSS 3	466
Nowe selektory	471
Nowe własności	472
Twitter Bootstrap	473
CSS 4	473
Dodatek D HTML DOM i JavaScript	477
HTML DOM	477
Metody dostępne w DOM	477
Właściwości dostępne w DOM	479
Poziomy DOM	479
JavaScript	480
Składnia języka	480
Możliwości JavaScriptu	483
jQuery	484
Instalacja jQuery	485
Selektory i filtry	485
Zdarzenia	487
Efekty w postaci animacji	488
Metody	488
Przechodzenie po elementach HTML	489
jQuery UI	490
jQuery Mobile	490

AJAX	491
JSON	492
XMLHttpRequest	493
AJAX w jQuery	494
Dodatek E Bazy nierelacyjne	497
MongoDB	498
RavenDB	498
Dodatek F Podstawy pozycjonowania w Google	499
Metatagi	499
Znacznik <title>	500
Opis strony	500
Słowa kluczowe	500
Wartości noindex i nofollow	500
Znaczniki HTML	500
Linkowanie	501
Zaplecze, katalogi stron i precle	502
Skrypty katalogów	502
Skrypty blogowe	503
Schematy linkowania	503
Schemat koła	503
Schemat piramidy	504
Gwiazda	504
Schematy mieszane	505
Linkowanie wewnętrzne	505
„Długi ogon”	505
Przyjazne adresy URL — Friendly URL	506
Pliki związane z pozycjonowaniem	506
robots.txt	506
sitemap.xml	507
.htaccess	508
Filtry i kary	509
Ban	509
Sandbox	509
Zmiany algorytmu Google	510
Panda	510
Pingwin	510
EDM	511
Narzędzia związane z pozycjonowaniem	511
Google Analytics i Google Webmasters Tools	511
Narzędzia do pracy z tekstem	511
Systemy wymiany linków	512
Półautomaty, „dodawarki” i automaty do postowania	512
Inne narzędzia	513
Skorowidz	515

Rozdział 4.

Entity Framework 6

Entity Framework (EF) jest odpowiedzialny za kontakt aplikacji z bazą danych. Domyślne operacje wykonywane na bazie danych to operacje CRUD (ang. *Create, Read, Update, Delete*). Dzięki EF i Scaffolding (automatyczny generator kodu) można automatycznie wygenerować kod odpowiedzialny za operacje CRUD dla każdej tabeli z bazy danych.

Podejście do pracy z modelem danych

EF udostępnia trzy podejścia do pracy z warstwą dostępu do danych DAL (ang. *Data Access Layer*):

- ◆ *Model First* — tworzy się diagram ERD (ang. *Entity-Relationship Diagram*) w Visual Studio za pomocą Model Designera i na podstawie diagramu generowana jest baza danych (kod SQL) oraz wszystkie klasy z modelem danych, po jednej dla każdej tabeli.
- ◆ *Code First* — tworzy się samodzielnie klasy (odpowiadające tabelom w bazie danych) i na podstawie klas generowana jest baza danych (kod SQL); dodatkowo, aby uaktualnić bazę danych po zmianach wprowadzonych w plikach z klasami, wykorzystuje się migracje (aktualizacje bazy danych).
- ◆ *Database First* — klasy i diagram ERD są generowane lub uaktualniane na podstawie istniejącej bazy danych.

Porównanie różnych podejść

Model First pozwala na szybkie utworzenie bazy danych poprzez wstawianie elementów w edytorze graficznym, ale nie daje takiej kontroli nad kodem jak *Code First*. W przypadku korzystania z *Code First* trzeba ręcznie tworzyć pliki z klasami, co powoduje, że można się łatwo pomylić. *Database First* z kolei jest wykorzystywany w przypadku, gdy tworzy się aplikację na podstawie już istniejącej bazy danych. Dodatkowo po wybraniu bazy danych, na podstawie której chce się utworzyć projekt, wybiera się, czy w dalszej części zostanie użyte podejście *Code First*, czy *Model First*. W zależności od wyboru utworzone zostaną klasy (*Code First*) lub diagram (*Model First*).

Największą kontrolę nad kodem daje podejście *Code First*. Dla początkujących podejście *Model First* będzie bardziej intuicyjne, ponieważ baza jest prezentowana w formie diagramu ERD.

Model dla podejścia Model First

Model danych dla podejścia *Model First* jest zawarty w pliku *.edmx*. Domyślnie model jest prezentowany jako diagram ERD w Entity Designerze (narzędzie w Visual Studio do tworzenia diagramu ERD).

Na pełny model danych w podejściu *Model First* składają się trzy części:

- ◆ model konceptualny (ang. *Conceptual Schema Definition Language*),
- ◆ model źródła danych (ang. *Storage Schema Definition Language*),
- ◆ część odwzorowująca (ang. *Mapping Specification Language*).

Wszystkie części znajdują się w jednym pliku *.edmx*. Jest to plik XML. Na podstawie pliku *.edmx* generowany jest schemat bazy danych — diagram ERD.

Przykładowy schemat bazy danych utworzony w Entity Designerze prezentuje rysunek 4.1.

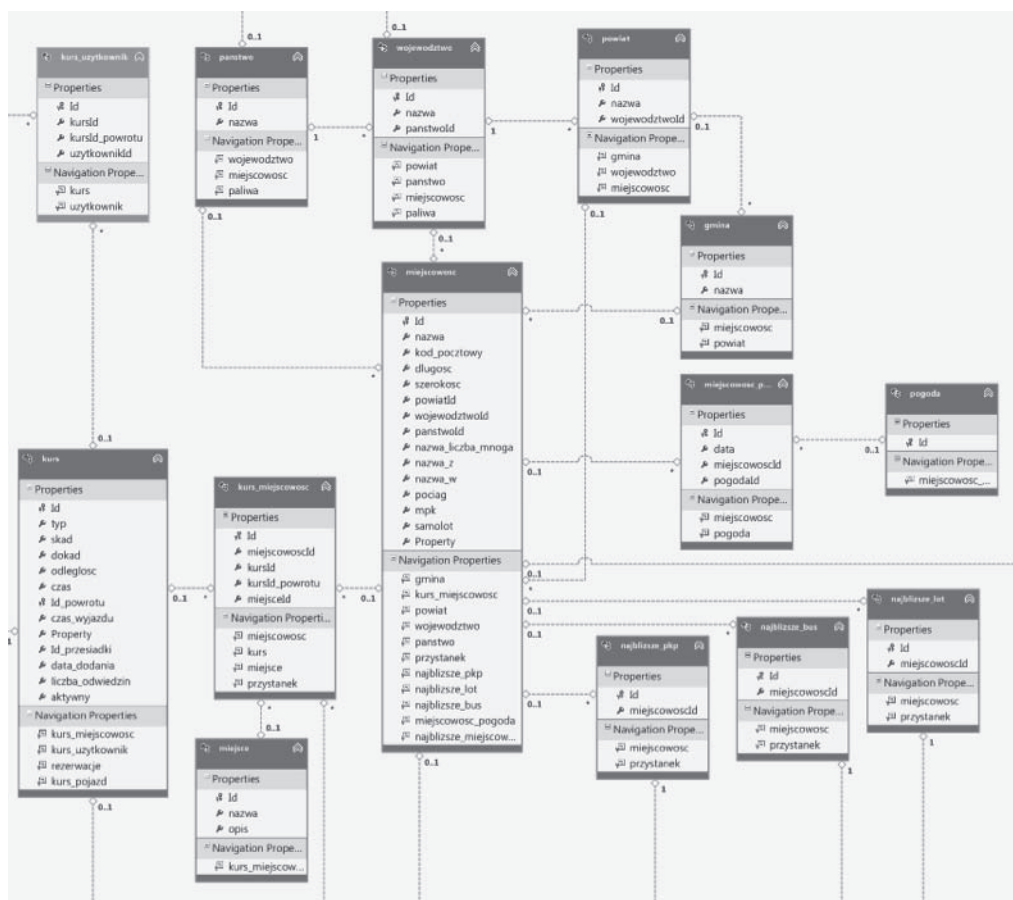
Model dla podejścia Code First

Przykładowa klasa z modelem dla podejścia *Code First* znajduje się na listingu 4.1. Jest to klasa POCO (ang. *plain-old CLR objects*), czyli klasa, która odwzorowuje strukturę tabeli z bazy danych na klasę w podejściu obiektowym.

Listing 4.1. Przykładowa klasa POCO

```
public partial class Kurs
{
    public Kurs()
    {
        this.KursUser = new HashSet<KursUser>();
    }
    public int Id { get; set; }
    public string Skad { get; set; }
    public string Dokad { get; set; }
    public System.DateTime DataWyjazdu { get; set; }
    public System.DateTime DataDodania { get; set; }

    public virtual ICollection<KursUser> KursUser { get; set; }
}
```

Rysunek 4.1. Diagram ERD z Entity Designera

Nowości wprowadzane w kolejnych wersjach EF

Nowości wprowadzone w EF 5

Najważniejsze nowości wprowadzone w EF 5 to:

- ♦ możliwość tworzenia wielu diagramów dla jednego modelu (*Model First*);
- ♦ wsparcie dla typu wyliczeniowego enum w *Code First* i *Model First*;
- ♦ wsparcie dla typów danych przestrzennych (ang. *spatial data types*) w *Code First* i *Model First*;

- ◆ TVF (ang. *Table-Valued Functions*) w *Model First* — podobne do procedur składowanych, z jedną małą różnicą — dane z TVF mogą być użyte w zapytaniach LINQ, natomiast dane z procedury składowanej nie.

Nowości wprowadzone w EF 6

Najważniejsze nowości wprowadzone w EF 6 to:

- ◆ własne konwencje w *Code First*;
- ◆ poprawiona wydajność dla dużych modeli w podejściu *Code First*;
- ◆ migracje dla wielu kontekstów do jednej bazy danych;
- ◆ automatyczne ponawianie transakcji (ang. *Connection Resiliency*), w przypadku gdy zakończy się ona niepowodzeniem;
- ◆ tworzenie nowych kontekstów dla otwartego połączenia;
- ◆ poprawione wsparcie dla transakcji;
- ◆ wstrzykiwanie implementacji w czasie działania programu (ang. *Dependency Resolution*) oraz wsparcie dla wzorców *Service Locator* i *IoC*;
- ◆ łatwe przechwytywanie logów (ang. *Interception/SQL Logging*), zapytań SQL generowanych przez EF.

Relacyjne bazy danych i EF

Krótki opis baz relacyjnych

Bazy relacyjne powstały w czasach, gdy przestrzeń dyskowa była dość dużym ograniczeniem dla programów. Dyski miały pojemności liczone w megabajtach, a nie — jak obecnie — w terabajtach. Aby oszczędzić miejsce na dysku, dane musiały być przechowywane w oszczędny sposób bez duplikacji wpisów. Relacyjne bazy danych składają się z tabel, pomiędzy którymi zachodzą relacje. Każda tabela składa się z wierszy i kolumn. Na każdy wiersz składają się jednakowo ułożone kolumny wypełnione wartościami, które z kolei w każdym wierszu mogą być inne. Każdy wiersz ma swój unikalny numer zwany kluczem podstawowym PK (ang. *Primary Key*), po którym można rozpoznać konkretny wiersz danych (np. konkretnego pracownika z tabeli *Pracownik*). Aby powiązać wiersz z jednej tabeli z wierszem znajdującym się w innej tabeli, należy umieścić w niej klucz obcy FK (ang. *Foreign Key*). Jest to ta sama wartość, która w pierwszej tabeli jest kluczem podstawowym (PK). Druga tabela może np. zawierać miejsce zamieszkania pracownika, które może być wspólne dla kilku pracowników. Dzięki takiemu rozwiązaniu zostanie utworzona relacja pomiędzy tabelami. Istnieje kilka rodzajów relacji.

Relacja „jeden do wielu”

Relacja „jeden do wielu” polega na tym, że do jednego wiersza w tabeli może być przypisanych kilka wierszy z innej tabeli. Przykładowo jeden klient może mieć kilka zamówień. Klucz obcy znajduje się w tabeli z zamówieniem i ma taką samą wartość jak klucz podstawowy z tabeli klienta. Po kluczu obcym można odczytać więcej danych na temat klienta, który złożył zamówienie, dzięki czemu nie trzeba do każdego zamówienia zapisywać tych samych danych pojedynczego klienta. Taka operacja odczytu dodatkowych danych z drugiej tabeli nazywana jest „joinem”. Na listingu 4.2 przedstawiono przykładową relację „jeden do wielu” dla podejścia *Code First*, a na rysunku 4.2 zaprezentowano tę samą relację w podejściu *Model First*.

Listing 4.2. Przykład relacji „jeden do wielu” w podejściu *Code First*

```
public class Ogloszenie
{
    public Ogloszenie()
    {
    }
    public int Id { get; set; }
    public string Tresc { get; set; }
    public System.DateTime DataDodania { get; set; }

    public string UzytkownikId { get; set; }

    public Uzytkownik Uzytkownik { get; set; }
}

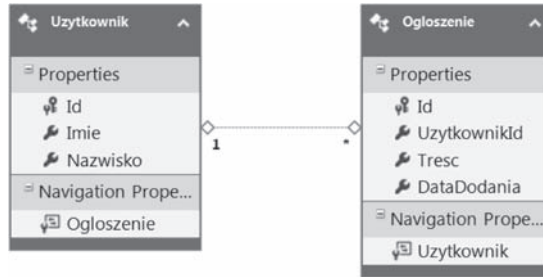
public class Uzytkownik
{
    public Uzytkownik()
    {
        this.Ogloszenie = new HashSet<Ogloszenie>();
    }
    public int Id { get; set; }
    public string Imie { get; set; }
    public string Nazwisko { get; set; }

    public virtual ICollection<Ogloszenie> Ogloszenie { get; private
        ↪set; }
}
```

Relacja „jeden do jednego”

Relacja „jeden do jednego” występuje, gdy jeden wiersz z pierwszej tabeli jest powiązany z tylko jednym rekordem z drugiej tabeli. Taka relacja może mieć sens np. wtedy, gdy chcesz przechowywać dodatkowe informacje na temat użytkownika, jednak nie chcesz, aby były one w tej samej tabeli, lub gdy użytkownik może pełnić tylko jedną funkcję w firmie. Na listingu 4.3 przedstawiono przykładową relację „jeden do jednego” dla podejścia *Code First*, a na rysunku 4.3 zaprezentowano tę samą relację w podejściu *Model First*.

Rysunek 4.2.
Przykład relacji
„jeden do wielu”
w podejściu
Model First



Listing 4.3. Przykład relacji „jeden do jednego” w podejściu Code First

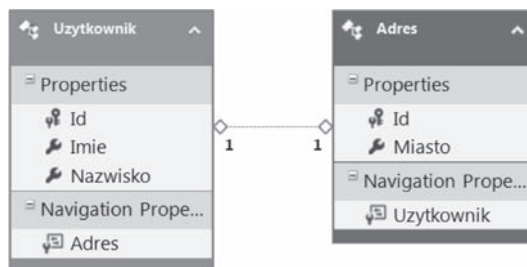
```
public class Uzytkownik
{
    public Uzytkownik()
    {
    }
    public int Id { get; set; }
    public string Imie { get; set; }
    public string Nazwisko { get; set; }

    [Required]
    public virtual Adres Adres { get; private set; }
}

public class Adres
{
    public Adres()
    {
    }
    [Key, ForeignKey(„Uzytkownik”)]
    public int Id { get; set; }
    public string Miasto { get; set; }

    public virtual Uzytkownik Uzytkownik { get; set; }
}
```

Rysunek 4.3.
Przykład relacji
„jeden do jednego”
w podejściu Model
First



Relacja „wiele do wielu”

Relacja „wiele do wielu” występuje, gdy jeden wiersz z pierwszej tabeli może być powiązany z wieloma wierszami z drugiej tabeli, a jeden wiersz z drugiej tabeli może być powiązany z kilkoma wierszami z pierwszej tabeli. Przykładowo pracownik może

mieć przypisanych wiele adresów, a jeden adres może być przypisany do wielu pracowników. Oczywiście nie chodzi tu o to, że w tabeli są zapisane te same dane, tylko o to, że klucz obcy pokazuje na ten sam wiersz w tabeli. Relacje „wiele do wielu” rozkłada się na dwie relacje „jeden do wielu” i dodaje trzecią tabelę pomiędzy tabelami, które mają relację „wiele do wielu”. Proces taki zabezpiecza przed redundancją danych (powtarzaniem tych samych danych). Nowa, środkowa tabela ma dwa klucze obce: jeden do tabeli pierwszej, a drugi do drugiej. W podejściu obiektowym w EF możliwe jest tworzenie relacji „wiele do wielu” bez dodatkowej klasy dla pośredniej tabeli. EF automatycznie przetłumaczy takie powiązanie i w bazie danych pojawi się trzecia tabela. Tabela pośrednia będzie niewidoczna z punktu widzenia kodu, jednak będzie się znajdowała w bazie danych. Na listingach przedstawiono przykładową relację „wiele do wielu” dla podejścia *Code First* bez dodatkowej tabeli (listing 4.4) i z dodatkową tabelą (listing 4.5), a na rysunkach zaprezentowano tę samą relację w podejściu *Model First* bez dodatkowej tabeli (rysunek 4.4) i z dodatkową tabelą (rysunek 4.5).

Listing 4.4. Przykład relacji „wiele do wielu” w podejściu *Code First* bez dodatkowej tabeli

```
public class Uzytkownik
{
    public Uzytkownik()
    {
        this.Wiadomosc = new HashSet<Wiadomosc>();
    }
    public int Id { get; set; }
    public string Imie { get; set; }

    public string Nazwisko { get; set; }

    public virtual ICollection<Wiadomosc> Wiadomosc { get; private
        ↪set; }
}

public class Wiadomosc
{
    public Wiadomosc ()
    {
        this.Uzytkownik = new HashSet<Uzytkownik>();
    }
    public int Id { get; set; }
    public string Tresc { get; set; }

    public virtual ICollection<Uzytkownik> Uzytkownik { get; private
        ↪set; }
}
```

Listing 4.5. Przykład relacji „wiele do wielu” w podejściu *Code First* z dodatkową tabelą

```
public class Uzytkownik
{
    public Uzytkownik()
    {
        this.Wiadomosc = new HashSet<Wiadomosc>();
    }
    public int Id { get; set; }
    public string Imie { get; set; }
```

```

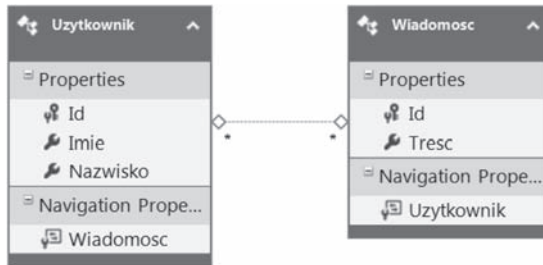
public string Nazwisko { get; set; }
public virtual ICollection<Uzytkownik_Wiadomosc>
    ↳Uzytkownik_Wiadomosc { get; private set; }
}

public class Uzytkownik_Wiadomosc
{
    public int Id { get; set; }
    public string UzytkownikId { get; set; }
    public int WiadomoscId { get; set; }
    public virtual Uzytkownik Uzytkownik { get; set; }
    public virtual Wiadomosc Wiadomosc { get; set; }
}

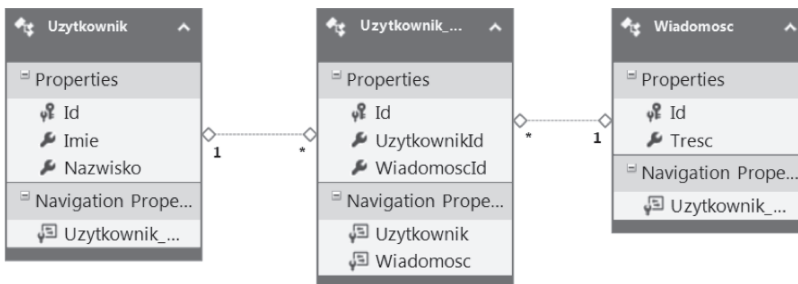
public class Wiadomosc
{
    public Wiadomosc()
    {
        this.Uzytkownik_Wiadomosc = new
            ↳HashSet<Uzytkownik_Wiadomosc>();
    }
    public int Id { get; set; }
    public string Tresc { get; set; }

    public virtual ICollection<Uzytkownik_Wiadomosc>
        ↳Uzytkownik_Wiadomosc { get; private set; }
}

```



Rysunek 4.4. Przykład relacji „wiele do wielu” w podejściu Model First bez dodatkowej tabeli



Rysunek 4.5. Przykład relacji „wiele do wielu” w podejściu Model First z dodatkową tabelą

Relacje opcjonalne

Relacje opcjonalne oznaczają, że dana relacja może, ale nie musi wystąpić. Duże znaczenie ma koniec, na którym znajduje się relacja opcjonalna. Przykładowo klient może, ale nie musi posiadać dodatkowych danych, dlatego relacja opcjonalna musi być po stronie tabeli klienta, ponieważ dane dodatkowe zawsze będą powiązane z klientem. Nie może być danych dodatkowych bez klienta, natomiast klient może być bez danych dodatkowych. Taka relacja to relacja 0..1 do 1.

Podobnie wygląda sytuacja z relacją 0..1 do wielu. W przypadku takiej relacji może istnieć wiele ogłoszeń, ale ogłoszenie niekoniecznie musi być powiązane z użytkownikiem.

Obiekty DbContext i DbSet

DbContext i DbSet

DbContext to obiekt odpowiedzialny za pracę z danymi. DbContext zajmuje się odczytem danych z bazy danych, śledzeniem stanu obiektów, zarządzaniem i powiązaniem pomiędzy danymi w pamięci, tłumaczeniem danych relacyjnych na obiektywne i zapisem danych do bazy. W klasie DbContext znajdują się właściwości DbSet, które reprezentują kolekcje określonych klas (tabel) znajdujących się w obiekcie kontekstu (listing 4.6).

Listing 4.6. Przykładowa klasa DbContext

```
public class ProduktContext : DbContext
{
    public DbSet<Kategoria> Kategorie { get; set; }
    public DbSet<Produkt> Produkty { get; set; }
}
```

Obiekt kontekstu jest przechowywany w pamięci do czasu, aż zostanie usunięty przez *Garbage Collector* lub zostanie wywołana metoda `Dispose()` na jego instancji. W aplikacji internetowej obiekt DbContext powinien być tworzony dla każdego żądania HTTP. Czas życia takiego obiektu powinien być jak najkrótszy, ponieważ nie zostanie on ponownie użyty.

Metody Attach i Detach

Dane pobierane z bazy są automatycznie przypisywane do kontekstu. Wyjątkiem są tutaj zapytania `AsNoTracking()`, których wynik nie jest zapisywany do kontekstu, więc dane nie są śledzone (możliwe jest przyłączenie takich danych za pomocą metod). Ponieważ DbContext przechowuje dane w postaci obiektowej, możliwe jest odłączenie obiektu od kontekstu lub przyłączenie obiektu pochodzącego z innego źródła, innego kontekstu bądź

obiektu wcześniej odłączonego od kontekstu (również z innej instancji kontekstu). Obiekty odłączone od kontekstu nie mają śledzonego stanu przez `DbContext` i nie mogą zostać zapisane do bazy danych.

Istnieją metody pozwalające na przyłączenie obiektu do kontekstu:

- ◆ `AddObject()` dodaje i ustawia stan obiektu na `Added` (dodany). Dla obiektów ze stanem `Added` klucz podstawowy (PK) nie musi mieć unikalnej wartości, ponieważ podczas zapisu i tak zostanie wygenerowany przez bazę danych.
- ◆ `Attach()` dodaje do kontekstu i ustawia stan na `Unchanged` (niezmieniony). Wartość PK musi być unikalna, w przeciwnym wypadku EF wywoła wyjątek.
- ◆ `AttachTo()` działa identycznie jak `Attach()`, jednak zamiast wywoływać ją dla obiektu kontekstu, to nazwę kontekstu podaje się jako parametr.

Za pomocą metody `Detach()` można odłączyć obiekt od kontekstu, jednak niesie to za sobą pewne konsekwencje. Jeśli obiekt posiadał relacje do innych obiektów, to powiązane obiekty nie zostaną usunięte z kontekstu. Stan obiektu nie jest śledzony, odłączenie nie wpływa na dane w bazie danych, a jeśli istnieje, nie zostaje wywołane kaskadowe usuwanie w trakcie wykonywania operacji `Detach()`:

```
context.Detach(zamowienie.Szczegoly.First());
```

Relacje poprzez klucz FK a relacje niezależne (obiektowe)

Relacje „jeden do jednego” zawsze są relacjami poprzez klucz (FK), natomiast relacje „wiele do wielu” zawsze są relacjami niezależnymi (ang. *Independent*). Dla relacji „jeden do wielu” można zastosować relację FK lub niezależną.

Relacje poprzez klucz obcy — FK Association

Relacja FK (ang. *FK Association*) polega na dodaniu klucza obcego do klasy odwzorowującej tabelę. Relacje pomiędzy klasami polegają na kluczu obcym, tak samo jak w bazie danych (listing 4.7). Takie rozwiązanie nie jest rozwiązaniem obiektowym, lecz relacyjnym. Natomiast klasy są częścią programowania obiektowego. Takie rozwiązanie ma swoje plusy i minusy.

Plusy relacji FK:

- ◆ znając wartość klucza, można dodać dane do sąsiedniej tabeli, nie trzeba posiadać wczytanego całego obiektu z bazy danych;
- ◆ podejście bardziej intuicyjne, identyczne jak w bazie danych i kodzie SQL.

Minusy relacji FK:

- ♦ podejście relacyjne, a nie obiektowe, w środowisku obiektowym;
- ♦ posiadanie dwóch relacji (zarówno FK, jak i relacji niezależnej) może powodować problem z synchronizacją pomiędzy tymi dwoma rodzajami relacji.

Listing 4.7. Przykład relacji FK w EF

```
public class Ogloszenie
{
    public Ogloszenie()
    {
    }
    public int Id { get; set; }
    public int UzytkownikId { get; set; }
    public Uzytkownik Uzytkownik { get; set; }
}

public class Uzytkownik
{
    public Uzytkownik()
    {
        this.Ogloszenie = new HashSet<Ogloszenie>();
    }
    public int Id { get; set; }
    public string Imie { get; set; }
    public string Nazwisko { get; set; }

    public virtual ICollection<Ogloszenie> Ogloszenie { get; private
        ↪set; }
}
```

Relacje niezależne — Independent Association

Relacja niezależna (ang. *Independent Association*) polega na tym, że nie dodaje się do klasy kluczy obcych, tylko obiekt klasy, z którą występuje relacja (listing 4.8). Konsekwencją takiego podejścia jest konieczność posiadania całego obiektu w pamięci — bez tego EF nie mógłby zamodelować relacji, ponieważ nie opiera się ona na FK.

Plusy relacji niezależnej:

- ♦ istnieje dostęp do wszystkich danych z sąsiedniej tabeli, a nie tylko do klucza, ponieważ w pamięci znajduje się cały obiekt;
- ♦ obiektowe podejście.

Minusy relacji niezależnej:

- ♦ skomplikowane tłumaczenie relacji pomiędzy obiektami na relacje w bazie danych;
- ♦ mniej intuicyjne niż relacje FK;
- ♦ jeśli tworzy się relacje, potrzebne są obiekty dla obu końców relacji (w FK wystarczyło podać wartość klucza obcego);

- ◆ nie ma stanu `modified` — za każdym razem relacja jest usuwana i dodawana jest nowa;
- ◆ mniej naturalne z punktu widzenia baz danych i języka SQL, trudniejsze do wychwycenia błędy.

Listing 4.8. Przykład relacji niezależnej w EF

```
public class Ogloszenie
{
    public Ogloszenie()
    {
    }
    public int Id { get; set; }

    public Uzytkownik Uzytkownik { get; set; }
}

public class Uzytkownik
{
    public Uzytkownik()
    {
        this.Ogloszenie = new HashSet<Ogloszenie>();
    }
    public int Id { get; set; }
    public string Imie { get; set; }
    public string Nazwisko { get; set; }
    public virtual ICollection<Ogloszenie> Ogloszenie { get; private
        ↪set; }
}
}
```

Odpytywanie bazy danych za pomocą EF i LINQ

Wczytywanie zachłanne — Eager Loading

Wczytywanie zachłanne (ang. *Eager Loading*) to proces, w którym zapytanie o jeden typ tabeli pobiera również dane z tabeli, które są z nią powiązane (posiadają relacje). Użytkownik posiada wiele ogłoszeń. Jeśli pobiera się użytkownika, to pobierane są również jego ogłoszenia. Aby użyć *Eager Loading*, gdy włączone jest *Lazy Loading*, należy wywołać metodę `Include()` w zapytaniu LINQ (listing 4.9).

Listing 4.9. Przykład zapytania *Eager Loading*

```
using (var context = new ApplicationDbContext())
{
    var uzytkownicy = context.Uzytkownik
        ↪.Include(b => b.Ogloszenia)
        ↪.ToList();
}
}
```

Wczytywanie leniwe — Lazy Loading

Wczytywanie leniwe (ang. *Lazy Loading*) pozwala na opóźniony odczyt danych. Nie ma konieczności pobierania danych ogłoszeń, jeśli chce się pobrać tylko dane użytkownika, pomimo że w klasie `Uzytkownik` znajduje się kolekcja ogłoszeń. Gdy używa się *Lazy Loading*, podczas pobierania danych do klasy użytkownika nie są ładowane jego ogłoszenia (listingi 4.10 i 4.11). *Lazy Loading* wymaga oznaczenia właściwości relacji jako wirtualnej (`virtual`). Aby było możliwe leniwe ładowanie, tworzone są dynamiczne puste obiekty proxy, które nadpisują pola wirtualne właściwości. Jeśli użytkownik próbuje się odwołać do obiektu proxy po raz pierwszy (chce wyświetlić również informacje o ogłoszeniach użytkownika), to tworzona jest instancja prawdziwego obiektu (kolekcji ogłoszeń) w miejsce pustego obiektu proxy.

Listing 4.10. Przykład klasy *Lazy Loading*

```
public class Uzytkownik
{
    public Uzytkownik()
    {
        this.Ogloszenia = new HashSet<Ogloszenie>();
    }

    public string Imie { get; set; }
    public string Nazwisko { get; set; }

    public virtual ICollection<Ogloszenie> Ogloszenia { get; private
        ↪set; }
}
```

Listing 4.11. Przykład zapytania *Lazy Loading*

```
using (var context = new ApplicationDbContext())
{
    var uzytkownicy = context.Uzytkownik.Take(10);
}
```

Podczas korzystania z Web API wymagana jest serializacja danych (zapis w formacie JSON lub XML). Aby serializacja była możliwa, konieczne jest wyłączenie *Lazy Loading*, ponieważ powoduje to ładowanie dużej ilości niepotrzebnych danych, a nawet zapętlenie. *Lazy Loading* można wyłączyć poprzez usunięcie modyfikatora `virtual` przed właściwością relacji lub globalnie w klasie z kontekstem, co zostało zaprezentowane w poniższym kodzie:

```
public ApplicationDbContext()
{
    this.Configuration.LazyLoadingEnabled = false;
}
```

Zalety *Lazy Loading*:

- ♦ szybsze ładowanie aplikacji,
- ♦ mniejsze zużycie pamięci poprzez wczytywanie danych na życzenie,

- ◆ zmniejszenie liczby zapytań do bazy danych.

Wady *Lazy Loading*:

- ◆ konieczność sprawdzania, czy *Lazy Loading* jest włączone, co skutkuje nieznacznie mniejszą wydajnością, gdyż dane i tak muszą zostać wczytane.

Jawne ładowanie — Explicit Loading

Jeśli chcemy, aby przy wyłączonym trybie *Lazy Loading* zapytanie zachowywało się tak jak w trybie *Lazy Loading*, trzeba skorzystać z zapytania *Explicit Loading*. Aby wykonać *Explicit Loading*, należy wywołać metodę `Load()` na powiązanej relacji tabeli (listing 4.12).

Listing 4.12. Przykład *Explicit Loading*

```
using (var context = new ApplicationDbContext())
{
    var uzytkownik = context.Uzytkownik.Find(2);
    context.Entry(uzytkownik).Collection(p => p.Ogloszenia).Load();
}
```

Problem N+1

Problem N+1 występuje, gdy nie korzysta się z *Lazy Loading* i chce się pobrać w jednym zapytaniu do bazy listę obiektów, które posiadają relacje z innymi obiektami (listing 4.13). Następnie dla każdego pobranego obiektu jest wykonywane osobne zapytanie pobierające dodatkowe dane z tabel, które są powiązane (posiadają relacje). Problem występuje w zapytaniach bez wykorzystania operacji `join`. Przykładowo każdy klient (wiersz w tabeli *klient*) ma przypisany jeden adres (wiersz z tabeli *adres*). Przy pobieraniu listy klientów, gdy każda klasa klienta ma pole adres, wywołane zostaje zapytanie zwracające listę klientów (bez adresów), a potem automatycznie wywoływane są dodatkowo osobne zapytania o adres. Każde zapytanie jest wywoływane osobno, a więc będzie tyle zapytań do bazy danych, ile zostanie zwróconych klientów. W *Lazy Loading Problem N+1* wystąpi dopiero wtedy, gdy znajdzie potrzeba pobrania adresów. Jeśli tabela posiada `index`, dane zostaną bardzo szybko zwrócone — i takie podejście jest prawidłowe (szybsze niż operacja `join`), jednak gdy danych jest dużo i trzeba wykonać bardzo dużo pojedynczych zapytań do bazy, operacja `join` powinna być szybsza (listing 4.14). Najlepszym wyjściem jest napisanie dwóch zapytań i sprawdzenie, które jest szybsze w danym przypadku i w jakim stopniu obciąża bazę danych.

Listing 4.13. Przykład kodu prezentującego *Problem N+1*

```
using (var context = new ApplicationDbContext())
{
    foreach (var uzytkownik in context.Uzytkownik)
    {
        foreach (var ogloszenie in uzytkownik.Ogloszenia)
        {
            Console.WriteLine("{0}: {1}", uzytkownik.Imie,
                ↳ogloszenie.Tytuł);
        }
    }
}
```

```

    }
}

```

Listing 4.14. Przykład kodu korzystającego z operacji *join* (metoda *Include*) bez Problem *N+1*

```

using (var context = new ApplicationDbContext())
{
    foreach (var uzytkownik in
        ↪context.Uzytkownik.Include("Ogloszenia"))
    {
        foreach (var ogloszenie in uzytkownik.Ogloszenia)
        {
            Console.WriteLine("{0}: {1}", uzytkownik.Imie,
                ↪ogloszenie.Tytul);
        }
    }
}

```

Metoda `AsNoTracking()`

Metoda rozszerzająca `AsNoTracking()` zwraca z zapytania dane, które nie będą śledzone przez EF ani nie będą przechowywane w pamięci obiektu `DbContext`. Zastosowanie `AsNoTracking()` skutkuje około dwukrotnie większą wydajnością przy odczycie. Jeśli dane mogą lub mają zostać zmodyfikowane, nie powinno się korzystać z tej metody. EF będzie wówczas śledził zmiany w obiekcie, co umożliwi edycję lub zapis danych do bazy (listing 4.15).

Listing 4.15. Przykładowy kod wykorzystujący metodę `AsNoTracking()`

```

using (var context = new WebApplicationContext())
{
    var dane = context.Tabela
        ↪.Where(d => d.Nazwa.Contains(".NET"))
        ↪.AsNoTracking()
        ↪.ToList();
}

```

Odroczone i natychmiastowe wykonanie

Natychmiastowe (ang. *Immediate*) wykonanie zapytania pobiera całość danych od razu podczas wykonywania zapytania. Odroczone (ang. *Deferred*) wykonanie zapytania korzysta z iteratorów (tabela 4.1) i `yield return`, a więc musi zwracać typ danych, który dziedziczy po `IEnumerable`, i zwraca dane pojedynczo w momencie, kiedy są potrzebne. Domyślnie używane jest odroczone wykonanie. Aby wywołać natychmiastowe wykonanie, należy użyć jednej z następujących metod: `ToList()`, `ToDictionary()` lub `ToArray()`. Wszystkie zapytania zwracające pojedynczą wartość są wykonywane natychmiastowo, np. `Average`, `Count`, `First` lub `Sum`, ponieważ nie ma możliwości użycia opóźnionego wykonania, jeśli jest tylko jeden element w kolekcji.

Tabela 4.1. Lista operatorów, które posiadają zaimplementowane iteratory

Metoda	Iterator
Cast	CastIterator
Concat	ConcatIterator
DefaultIfEmpty	DefaultIfEmptyIterator
Distinct	DistinctIterator
Except	ExceptIterator
GroupJoin	GroupJoinIterator
Intersect	IntersectIterator
Join	JoinIterator
OfType	OfTypeIterator
Range	RangeIterator
Repeat	RepeatIterator
Reverse	ReverseIterator
Select	SelectIterator
SelectMany	SelectManyIterator
Skip	SkipIterator
SkipWhile	SkipWhileIterator
Take	TakeIterator
TakeWhile	TakeWhileIterator
Union	UnionIterator
Where	WhereIterator
Zip	ZipIterator

Entity SQL

Entity SQL to język zapytań bazujący na SQL (alternatywa dla *LINQ to Entities*). Pozwala na pisanie zapytań w języku SQL, jednak w treści zapytania zamiast nazw tabel podaje się nazwy klas z aplikacji. *Entity SQL* powinien być używany tylko w momencie, gdy zapytanie pisane za pomocą LINQ jest zbyt skomplikowane. Zapytania *Entity SQL* nie są kompilowane, lecz konstruowane są w czasie działania programu. Zapytania LINQ są kompilowane i typowane (zwiększa się bezpieczeństwo i szybkość), dlatego powinny być używane, jeśli tylko jest to możliwe. Aby wykonać zapytanie *Entity SQL*, należy uzyskać `ObjectContext` z obiektu `DbContext` i skorzystać z generycznej metody `ObjectContext.Query<T>` (listing 4.16)¹.

Listing 4.16. Przykład zapytania *Entity SQL*

```
using(ApplicationDbContext context = new ApplicationDbContext())
{
    var adapter = (IObjContextAdapter)context;
    var objContext = adapter.ObjContext;
```

¹ <http://msdn.microsoft.com/en-us/library/bb399554.aspx>

```

string esqlQuery = @"SELECT VALUE Ogloszenie FROM
    ↳Models.Ogloszenia as Ogloszenie
    ↳WHERE Ogloszenie.Id > @parametrId";

ObjectQuery<Ogloszenie> query = new
    ↳ObjectQuery<Ogloszenie>(esqlQuery, objectContext,
    ↳MergeOption.NoTracking);

query.Parameters.Add(new ObjectParameter("parametrId", 1));

return query;
}

```

Bezpośrednie zapytania SQL do bazy (Direct/RAW SQL) i procedury składowane w EF

Zapytania *Direct/RAW SQL* powinny być stosowane tylko wtedy, gdy nie ma możliwości skorzystania z zapytań *LINQ to Entities* lub *Entity SQL*. W zapytaniu SQL podaje się nazwy tabel w bazie danych, a nie nazwy klas, jak to było w *Entity SQL*. W *Entity SQL* operuje się na typach pochodzących z aplikacji, co wpływa na bezpieczeństwo, ponieważ typy danych są znane i walidowane. Dla zapytań *RAW SQL*, czyli składających się z czystego kodu SQL, występuje niebezpieczeństwo ataku SQL Injection. Aby pobrać dane za pomocą języka SQL, należy użyć metody `SqlQuery()`, a w celu wysłania zapytania do bazy niezwracającego żadnych wartości korzysta się z `SqlCommand()`.

Przykład zapytania zwracającego typ klasy POCO:

```

using (var context = new ApplicationDbContext())
{
    var blogs = context.Ogloszenia.SqlQuery("SELECT * FROM dbo.Ogloszenia")
        .ToList();
}

```

Przykład wywołania procedury składowanej z parametrem:

```

using (var context = new ApplicationDbContext())
{
    var ogloszenieId = 1;
    var blogs = context.Ogloszenia.SqlQuery("dbo.GetOgloszenieById
        ↳@p0", ogloszenieId).Single();
}

```

Przykład zapytania zwracającego typ, który nie jest klasą POCO:

```

using (var context = new ApplicationDbContext())
{
    var tytuly = context.Database.SqlQuery<string>("SELECT Tytul FROM
        ↳ dbo.Ogloszenia").ToList();
}

```

Przykład zapytania SQL niezwracającego wartości:

```
using (var context = new ApplicationDbContext())
{
    context.Database.SqlCommand("UPDATE dbo.Ogloszenia SET Tytul =
        ↪'Jakis tytul' WHERE Id = 1");
}
```


Skorowidz

.NET Framework Data Provider, 147

A

- abstrakcja, 53
 - adnotacje do walidacji, 222
 - ADO.NET, 146
 - adresaty atrybutów, 94
 - Agile, 437
 - AJAX, 491, 495
 - AJAX Long-Polling, 492
 - AJAX Polling, 492
 - akcja
 - Create, 353, 379
 - Delete, 345, 350, 379
 - Details, 342, 376
 - Edit, 360, 363, 377
 - Index, 311, 317, 327, 385
 - aktualizacja
 - bibliotek, 281
 - Google
 - EDM, 511
 - Panda, 510
 - Pingwin, 510
 - kontrolera, 381, 387
 - pakietów, 279
 - szablonu, 365
 - widoku, 317, 345, 354, 360, 389
 - aliasy, 82
 - aliasy zewnętrzne, 83
 - animacje, 488
 - API
 - Drag and Drop, 463
 - Web Storage, 461
 - architektura
 - aplikacji, 334
 - dwuwarstwowa, 126
 - jednowarstwowa, 126
 - n-warstwowa, 126
 - trójwarstwowa, 126
 - argumenty
 - metod, 49
 - nazwane, 110
 - arkusz stylów CSS, 467
 - ASP.NET Identity, 244
 - ASP.NET MVC, 252
 - ASP.NET MVC 1, 185
 - ASP.NET MVC 2, 185
 - ASP.NET MVC 3, 186
 - ASP.NET MVC 4, 186
 - ASP.NET MVC 5, 185, 186
 - ASP.NET MVC 6, 187
 - ASP.NET Web API 2, 251
 - ASP.NET Web Forms, 145
 - ASP.NET Web Pages, 146
- atak
 - Cross-Site Request Forgery, 236
 - Cross-Site Scripting, 237
 - CSRF, 357
 - SQL Injection, 236
 - atrybut
 - Bind, 237
 - OutputCache, 232
 - atrybuty, 93
 - atrybuty Caller Info, 119
 - dla formularza, 459
 - globalne, 456
 - Attribute Routing, 216, 407
 - autocomplete, 408
 - automaty do postawiania, 512
 - automatyczna inicjalizacja właściwości, 120
 - autoryzacja, 242, 372
 - dostępu, 369
 - RSA, 448
 - Azure Service Bus, 144

B

- ban, 509
- baza danych, 305, 419
 - niespójność, 310
 - strategie dziedziczenia, 178

baza dokumentowa
 MongoDB, 498
 RavenDB, 498
 bazy niereelacyjne, 497
 bezpieczeństwo, 235, 369
 biblioteka
 Bootstrap, 329
 JQuery, 484
 jQuery Unobtrusive, 221
 PagedList, 382
 binding, 357
 BLOB Storage, 143
 blok try-catch, 238
 błąd, 310, 348
 404, 394
 dla podstrony, 397
 podczas edycji, 378
 w dostępie do danych, 395
 Browser Link, 270

C

C#, 19
 C# 2.0, 20
 C# 3.0, 20
 C# 4.0, 21
 C# 5.0, 21
 C# 6.0, 21
 cache, 231, 394
 cachowanie
 częściowe, 233
 po stronie klienta, 234
 po stronie serwera, 231
 rozproszone, 233
 w HTML 5, 234
 CAPTCHA, 513
 ceny
 licencji SQL Server, 265
 Windows Server 2012, 268
 certyfikat
 CSR, 450
 SSL, 449, 450
 aktywacja, 450
 instalacja, 450
 walidacja, 450
 ciasteczka, 243
 CIL, Common Intermediate Language, 19, 135
 claimy, 245
 CLI, Common Language Infrastructure, 135
 CLR, Common Language Runtime, 135, 137
 Code First, 157, 158
 Code First Data annotations, 235
 Code First Fluent API, 181
 controller, 128

CORS, 255, 256
 CRUD, 223
 CSS, 330
 CSS 3, 466
 nowe selektory, 471
 nowe własności, 472
 CSS 4, 473
 cykl życia obiektu, 341
 czas, 90

D

DAL, Data Access Layer, 157
 data, 90
 Data Annotations, 181, 235
 Data Contract, 140
 Data Member, 140
 Database First, 157
 DDD, 132
 debugowanie, 324
 aplikacji, 325, 327
 metody Seed, 308
 obiektu db, 327
 definiowanie własnego atrybutu, 256
 deklaracja zmiennej, 27
 deklaracje, 24
 deklaracje inline, 122
 delegaty, 72
 destruktor, 64
 diagram ERD, 159, 413
 DIP, 431
 DLR, Dynamic Language Runtime, 137
 długi ogon, Long Tail, 505
 dobre praktyki, 26
 dodawanie
 domeny, 415
 dyrektywy using, 285
 encji, 411
 kolumny, 413
 kontrolera, 224, 312
 kontrolerów, 311
 modelu, 410
 nowego projektu, 295
 nowej klasy, 283
 referencji, 296
 węzłów, 478
 widoków, 311
 dokument SOAP, 253
 DOM, 463, 469
 dostępne metody, 477
 dostępne właściwości, 479
 poziomy, 480
 schemat, 478
 domena, 415

- domena aplikacji, 189
- domyślny
 - layout, 313
 - widok, 402
- Drag and Drop, 463
- DRY, Don't Repeat Yourself, 435
- drzewa wyrażań, 101
- drzewo, 469
- DV, Domain Validation, 449
- dyrektywa using, 81, 121, 285
- dyrektywy preprocesora, 76
- dziedziczenie, 54
 - TPC, 179
 - TPH, 178
 - TPT, 178

E

- EDA, 133
- edycja ogłoszenia, 378
- edytowane pliki, 425
- EF 5, 159
- EF 6, 160
- elementy HTML 5, 456
- encje, 411
- Entity Framework, 153
- Entity Framework 6, 154, 157
- Entity SQL, 172
- EV, Extended Validation, 449
- eXtreme Programming, 439

F

- Federated Authentication, 247
- filtry, 485, 509
 - akcji, 197
 - wyjątków, 124
- folder
 - Models, 294
 - Shared, 201
 - z widokami, 316
- format
 - JSON, 406, 493
 - XML, 406
- formularz, 456
- framework Twitter Bootstrap, 473

G

- Garbage Collector
 - podział na generacje, 66
 - zasada działania, 66

- generowanie
 - kontrolerów, 223, 225
 - widoków, 226, 227
- geolokalizacja, 465
- Google Analytics, 511
- Google Webmasters Tools, 511
- GRASP, 432

H

- HandleError, 239
- Harvestery, 513
- hasło, 372
- hermetyzacja, 54
- hierarchia drzewa, 469
- hostowanie aplikacji, 143
- HTML 5, 453
 - atributy globalne, 456
 - nowe elementy, 456
 - sekcje, 457
 - SSE, 461
 - szablon, 454
 - typy pól formularza, 458
 - WebSockets, 462
 - znaczniki, 454, 459
- HTML 5 Application Cache, 234
- HTML 5 WebStorage, 234
- HTML DOM, 477
- HTML helper, 208, 401
- HTTP, 443

I

- Identity Provider, 246
- identyfikacja, 241
- IEnumerable, 96
- IHttpRequestResult, 260
- IIS, Internet Information Services, 263
 - moduły, 263
 - przetwarzanie żądań, 264
 - pule aplikacji, 264
- implementacja
 - .NET, 138
 - interfejsu, 69, 395
 - klasy repozytorium, 395
 - kontrolera, 396
 - metody POST, 362
- indeksatory, 61
- index, 373
- inicjalizacja
 - pól, 64
 - zmiennej, 28

inicjalizatory

- obiektów i kolekcji, 100
- słownikowe, 121

instalacja

- bibliotek, 298
- biblioteki PagedList, 382
- certyfikatu SSL, 450
- jQuery, 485
- kontenera IoC Unity, 338
- migracji, 300
- pakietów, 298

instrukcja

- checked, 78
- if, 40, 41
- switch, 41
- unchecked, 78

instrukcje

- iteracyjne, 43
- skoku, 45
- sterujące, 40

interfejs, 68, 86

- ICollection<>, 87
- IDictionary<TKey, TValue>, 86
- IEnumerable<T>, 87
- IHttpActionResult, 260
- IList<>, 87
- IOgloszenieRepo, 338
- IQueryable<>, 88

IoC, 397

ISAPI, 189

ISP, 429

iteratory, 97

J

JavaScript, 271, 352, 480

- jQuery, 484
- możliwości, 483
- składnia, 481

jawna implementacja interfejsu, 69

jawne ładowanie, 170

język C#, 19

jQuery, 352, 484, 495

- animacje, 488
- filtry, 485
- metody, 488
- selektory, 485
- zdarzenia, 487

jQuery Mobile, 491

jQuery UI, 490

JSON, 406, 493

- tablica, 493
- typy wartości, 493

JSON Editor, 271

JSONP, 255

K

kary, 509

kaskadowe usuwanie, 348

katalogi stron, 502

kategorie, 395

KISS, 436

klasa, 46

- ApplicationUser, 288
- FilterConfig, 239
- Kategoria, 286
- Ogloszenie_Kategoria, 287
- OgloszenieRepo, 335
- System.Array, 35
- System.Object, 63
- Uzytkownik, 287
- z kontekstem, 293

klasy

- częściowe, 103
- generyczne, 84

klucz

- FK, 166
- obcy, 166
- szyfrujący, 447

kolekcje, 37

- generyczne, 86
- IEnumerable, 123

komentarze, 23

komunikat o błędzie, 222, 351, 380

konfiguracja

- migracji, 301
- witryny, 417

konstruktor, 64

konstruktory pierwotne, 120

kontekst, 410

kontener

- IoC, 336, 341
- IoC Unity, 338

konto FTP, 421

kontrawariancja, 111, 112

kontroler, 194, 223

- Web API, 257
- z widokami, 311

konwencja ponad konfiguracją, 188

konwencje, 21

- nazewnicze, 25
- w MVC, 187

konwersja

- jawna, 58
- niejawna, 57

konwersje typów, 31

L

layout, 367
 leniwa inicjalizacja, 114
 licencjonowanie
 SQL Server, 265
 Windows Server, 267
 linki, 502
 linkowanie, 501
 linkowanie wewnętrzne, 505
 LINQ, 148, 168, 397, 400
 LINQ to DataSet, 149
 LINQ to Entities, 149
 LINQ to Objects, 149
 LINQ to SQL, 149
 LINQ to XML, 148
 lista ogłoszeń, 321, 395, 405
 literały, 30
 literały binarne, 124
 logowanie, 193, 369, 370, 373
 logowanie globalne, 240
 LSP, 428

Ł

łańcuchy, 36
 łączenie stylów CSS, 210

M

manifest Agile, 437
 mapowanie żądań na akcje, 257
 mechanizm
 refleksji, 93
 Scaffoldingu, 224
 metatagi, 499
 metoda, 49
 Aktualizuj, 365
 AsNoTracking, 171, 324, 327
 Attach, 165
 Create, 358
 Delete, 348
 Detach, 165
 Details, 342
 Dispose, 313
 Down, 304
 Edit, 359
 Equals, 57
 GET, 446
 GetHashCode, 57
 Index, 388
 OnException, 238
 POST, 362, 446
 POST dla akcji Create, 355

 POST dla akcji Delete, 347
 RegisterBundles, 209
 SaveChanges, 349
 Seed, 183, 305
 ToList, 325, 328
 Up, 302
 metody
 anonimowe, 74
 asynchroniczne, 118, 228, 230
 częściowe, 104
 generyczne, 84
 rozszerzające, 102
 synchroniczne, 228, 230
 zwrotne, 118
 metodyka Scrum, 439
 Microsoft Azure, 143
 Microsoft SQL Server 2014, 264
 Microsoft Visual Studio, 423
 Microsoft Visual Studio 2013 Ultimate, 268
 migracja startowa, 302
 migracje, 183, 300, 309
 minimalizacja, 209
 model, 128, 130, 218
 aktywny, 129
 danych, 157
 First, 157, 158
 pasywny, 129
 modele licencjonowania SQL Server, 265
 moduły
 Elmah, 241
 IIS, 263
 modyfikator private protected, 123
 MongoDB, 498
 MoSCoW, 437
 MVC, Model View Controller, 127, 131, 185
 autoryzacja, 242
 filtry, 198
 identyfikacja, 241
 konwencje, 188
 logowanie błędów, 241
 metody asynchroniczne, 228
 metody synchroniczne, 228
 role, 242
 struktura projektu, 187
 uwierzytelnianie, 241
 wyjątki, 238
 MVC Pipeline, 189
 moduły HTTP, 193
 ścieżka wywołań, 189
 uchwyty, 193
 MVC Scaffolding, 222
 MVP, Model View Presenter, 129, 131
 MVVM, Model View ViewModel, 131

N

narzędzia
 do pracy z tekstem, 511
 ORM, 153
 narzędzie NLOG, 240
 nawiasy klamrowe, 25
 nazwa domeny, 417
 nazwy
 kwalifikowane, 80
 niekwalifikowane, 80
 NHibernate, 153
 NHibernate 3, 154
 nowy projekt, 279

O

obiekt, 48
 DataRelation, 147
 DataRow, 147
 DataSet, 147
 DataTable, 147
 DataView, 147
 DbContext, 165
 DbSet, 165
 HttpApplication, 190
 obsługa
 błędów, 350, 363
 żądań, 191
 obszary, 217
 OCP, 428
 OData, 254
 odkomentowanie kodu, 370
 odnajdywanie widoków, 200
 odpytywanie
 bazy danych, 168
 cykliczne AJAX, 491
 odroczone wykonanie, 325
 ogłoszenia użytkownika, 393
 okno
 dodawania domeny, 416
 dodawania widoku, 207, 400
 NLOG, 240
 Output, 326
 potwierdzania wyboru, 352
 Properties, 414
 tworzenia bazy danych, 420
 tworzenia użytkownika, 421
 Watch, 326
 wyboru
 szablonu widoku, 228
 trybu działania witryny, 419

typu domeny, 417
 typu widoku, 228
 z błędami, 241
 zarządzania
 bazą danych, 420
 kontami FTP, 421
 witryną, 418
 opakowywanie, 32
 opcja
 Reference script libraries, 227
 Strongly Typed, 207
 OpenAuth, 250
 OpenId, 249
 operacja join, 400
 operacje
 CRUD, 223
 współbieżne, 176
 Operation Contract, 140
 operator, 38
 ??, 40
 trójargumentowy, 38
 opis strony, 500
 optymalizacja
 listy ogłoszeń, 322
 pod kątem pozycjonowania, 331
 pod kątem SEO, 329
 SEO, 343
 OV, Organization Verification, 449
 OWIN, 244

P

Page Inspector, 269
 pakiet PagedList.Mvc, 382
 pakiety, 282, 299
 panel użytkownika, 415
 parametr binding, 237
 parametry
 akcji, 196
 opcjonalne, 111
 pętla
 do while, 43
 for, 44
 foreach, 45
 while, 43
 piaskownica, 509
 plik
 .htaccess, 508
 _Layout.cshtml, 211
 BundlesConfig.cs, 209
 robots.txt, 506
 sitemap.xml, 507
 PLINQ, Parallel LINQ, 151

- pobieranie
 - danych, 220, 321, 326
 - po stronie klienta, 261
 - po stronie serwera, 261
 - tokena, 246
- podjęcie Model First, 409
- podpowiedzi składni, 319
- podświetlanie wierszy, 330
- podział wymagań klienta, 437
- pola, 48
- polimorfizm, 55
- połączenie
 - z bazą danych, 422
 - z serwerem, 494
- poprawa
 - architektury aplikacji, 334
 - wyglądu, 329
- postęp publikacji, 424
- potwierdzanie tożsamości, 447
- powiązania pomiędzy tabelami, 412, 413
- pozycjonowanie, 331, 499
 - .htaccess, 508
 - Friendly URL, 506
 - linkowanie, 501
 - linkowanie wewnętrzne, 505
 - metatagi, 499
 - narzędzia, 511
 - nofollow, 500
 - noindex, 500
 - robots.txt, 506
 - schematy linkowania, 503
 - sitemap.xml, 507
 - słowa kluczowe, 500
 - znaczniki HTML, 500
- prefiksy, 215
- presenter, 130
- problem N+1, 170
- procedury składowane w EF, 173
- programowanie
 - ekstremalne, 439
 - metodologie, 437
 - obiektywne, 53
 - zasady, 427
- projekt
 - Mono, 139
 - Repozytorium, 296
- protokół
 - SOAP, 253
 - SSL, 447
- przechodzenie po elementach HTML, 489
- przeciążanie
 - metod, 61
 - operatorów, 56
 - arytmetycznych, 60
 - konwersji, 57
 - logicznych, 58
 - relacji, 57

- przeniesienie
 - metody, 334
 - pliku, 299
 - zapytania LINQ, 334
- przestrzeń nazw, 79
- przestrzeń System.IO, 92
- przeszukiwanie sieci, 513
- przetwarzanie żądań, 264
- przyciski Bootstrap, 329
- przyjazne adresy URL, 506
- pseudoklasy
 - adresu, 475
 - czasu, 475
 - formularza, 475
 - logiczne, 473
 - struktury drzewa, 475
 - struktury siatki, 475
- pseudouwierzytelnianie, 250
- publikacja, 424
- publikacja systemu, 414
- pule aplikacji, 264
- punkt przerwania, 326
- puste linie, 24

Q

- Queue Storage, 143

R

- RavenDB, 498
- razor, 202
- refleksja, 93, 95
- regiony, 290
- reguła CSS, 468
- rejestracja
 - użytkownika, 369
 - z użyciem Google, 371
- relacja
 - FK, 166
 - jeden do jednego, 161
 - jeden do wielu, 161
 - wiele do wielu, 162, 408
- relacje
 - niezależne, 166, 167
 - opcjonalne, 165
 - poprzez klucz obcy, 166
- relacyjne bazy danych, 160
- repozytorium, 277, 335, 340, 410
- repozytorium generyczne, 278
- REST, 253

- rodzaje
 - arkuszy stylów:, 468
 - certyfi katów, 449
 - role, 242, 372
 - routing, 214, 316
 - ignorowanie ścieżek, 214
 - kolejność, 214
 - na podstawie atrybutów, 215, 252
 - obszary, 217
 - ograniczenia, 215
 - w Web API, 257
 - rozpakowywanie, 32
 - RP, Relying Party, 246
 - rzutowanie, 31
- S**
- sandbox, 509
 - Scaffolding, 224
 - schemat
 - dla Federated Authentication, 248
 - DOM, 478
 - formularza, 455
 - gwiazdy, 504
 - koła, 503
 - kontrolera, 225
 - piramidy, 504
 - wywołań, 192
 - schematy
 - linkowania, 503
 - mieszane, 505
 - Scrum, 439
 - sekcja featured, 213
 - sekcje, 211, 457
 - selektory, 471, 485
 - SEO, 329, 331, 343, 499
 - separacja zagadnień, 436
 - separatory cyfr, 124
 - serializacja, 495
 - Server Side Events, 461
 - Service Bus Queue, 145
 - Service Bus Relay, 144
 - Service Bus Topic, 145
 - Service Contract, 140
 - serwer
 - hostingowy, 414
 - IIS, 263
 - serwis z ogłoszeniami, 278
 - sesje, 243
 - silnik Razor, 203
 - Silverlight, 142
 - składnia
 - metod, 150
 - zapytań, 151
 - skróty klawiszowe, 272
 - skrypty, 209
 - blogowe, 503
 - katalogów, 502
 - słowo kluczowe, 29
 - as, 114
 - async, 118, 230
 - await, 118, 124, 230
 - base, 106
 - in, 111
 - is, 114
 - out, 111
 - task, 230
 - this, 106
 - typeof, 114
 - using, 71
 - yield return, 97
 - snippets, 269
 - SOA, 132
 - SOAP, 253
 - SoC, Separation of Concern, 436
 - SOLID, 427
 - sortowanie, 387
 - cyfr, 464
 - ogłoszeń, 392
 - sprawdzanie
 - danych, 308
 - wartości null, 123
 - SQL Logging, 179
 - SQL Server 2014, 266
 - SQL Server Management Studio, 422
 - SRP, 427
 - SSL/TLS, 447
 - stan aplikacji, 242
 - strona
 - edycji, 377
 - ufająca, 246
 - stronicowanie, 381
 - struktura, 67
 - .NET, 135
 - bazy danych, 305
 - dokumentu, 470
 - katalogów, 463
 - projektów, 280
 - projektu Repozytorium, 299
 - STS, 246
 - systemy wymiany linków, 512
 - szablon, 225
 - _Layout.cshtml, 365
 - dokumentu HTML 5, 454

Ś

śledzenie zmian, 175
dynamiczne, 176
migawkowe, 175

T

tabela
ograniczeń, 217
znaków szczególnych, 30
Table Storage, 143
tablica, 34
tablica typów, 28
TDD, Test Driven Development, 440
TempData, 204
testy, 440
TLS, Transport Layer Security, 448
token, 246
TPC, Table Per concrete Class, 179
TPH, Table Per Hierarchy, 178
TPT, Table Per Type, 178
transakcje w EF, 174
tryb Inspect Mode, 271
Twitter Bootstrap, 329, 473
tworzenie
bazy danych, 419, 420
interfejsu dla repozytorium, 337
klasy kontekstu, 290
konta FTP, 421
migracji początkowej, 302
modelu danych, 283
nowego projektu, 279
nowego projektu Repozytorium, 296
obiektu XMLHttpRequest, 493
paczek skryptów, 209
profilu, 232
referencji, 297
typ, 26
ActionResult, 260
HttpResponseMessage, 259
void, 259
wyliczeniowy, 31
z aplikacji, 260
typy
anonimowe, 105
danych MIME, 444
dopuszczające wartości zerowe, 33
dynamiczne, 107
pól formularza, 458
rezultatu, 194
walidacji, 449
wyjątków, 240
tytuł strony, 333

U

uchwyty HTTP, 193
UoW, 410
uruchamianie
aplikacji, 316
CORS, 256
pierwszej migracji, 304
Web Deploy, 419
uruchomiona aplikacja, 295
ustawienie
atrybutu, 463
projektu startowego, 297
usuwanie
kaskadowe, 177
węzłów, 478
uwierzytelnianie, 241, 369, 370
przez Windows ACS, 249
za pomocą claimów, 245

V

view, 128, 130
ViewBag, 204
ViewData, 204
ViewModel, 218, 402
Virtual Machine, 144
Visual Studio 2013, 269
Browser Link, 270
GIT, 272
JavaScript, 271
JSON Editor, 271
Microsoft Azure, 272
pasek przewijania, 270
podgląd definicji, 270
skróty klawiszowe, 272
tryb Inspect Mode, 271

W

walidacja, 221, 449
strony, 466
żądania POST, 357
walidator W3C, 468
warstwa modelu, 258, 294, 311
wartości
zerowe, 33
zmiennych, 325
WCF, Windows Communication Foundation, 140
WCF Endpoint, 142
wcięcia, 22
wczytywanie
leniwe, 169
zachłanne, 168

- wdrażanie aplikacji, 423
 - Web API, 251
 - ASP.NET MVC, 252
 - CORS, 256
 - Entity Framework, 258
 - pobieranie danych, 261
 - routing, 257
 - typy, 258
 - wersjonowanie, 262
 - Web API 2, 251
 - Web Deploy, 420
 - Web Role, 144
 - Web serwis, 251, 253
 - Web Site, 144
 - Web Storage, 461
 - WebSockets, 462
 - wersje
 - języka C#, 20
 - Windows Server 2012, 267
 - wersjonowanie w Web API, 262
 - weryfikacja położenia, 467
 - widok, 200, 390, 400, 404
 - Details, 344
 - dla konta Admin, 375
 - dla konta Pracownik, 375
 - dla niezalogowanych użytkowników, 376
 - dla zalogowanych użytkowników, 375
 - Index, 398
 - Partial, 368, 402
 - Partial View, 229, 386
 - z layoutem, 229
 - widoki
 - częściowe, 201, 366
 - typowane, 205
 - WIF, 245
 - Windows ACS, 248
 - Windows Azure Storage, 143
 - Windows Server 2012, 267
 - właściwości, 52
 - właściwości XMLHttpRequest, 494
 - włączenie NuGet, 281
 - Worker Role, 144
 - WPF, Windows Presentation Foundation, 139
 - wstrzykiwanie
 - implementacji, 397
 - kontekstu, 340
 - repozytorium, 336
 - wybieranie
 - klasy z modelem, 313
 - miejsca publikacji, 423
 - szablonu, 296
 - treści modelu, 411
 - typu aplikacji, 280
 - wygląd
 - aplikacji, 330–332
 - aplikacji z paginacją, 385
 - strony, 399
 - wyjątki, 77, 238
 - wykonanie
 - natychmiastowe, 171
 - odroczone, 171
 - wyrażenia
 - dla metod, 122
 - dla właściwości, 122
 - lambda, 74
 - regularne, 89
 - wyświetlanie ogłoszeń, 397
 - wywoływanie metod, 51
 - wzorce
 - architektoniczne, 125, 277
 - projektowe, 277
 - wzorzec
 - IoC, 277
 - MVC, 185
 - UnitOfWork, 278
- X**
- XMLHttpRequest, 493
- Z**
- zabezpieczanie akcji, 373
 - zagnieżdżanie przestrzeni nazw, 79
 - zainstalowane pakiety, 299
 - zakup certyfikatu SSL, 450
 - zapytania
 - bezpośrednie, 173
 - LINQ, 150
 - zarządzanie
 - kontami FTP, 421
 - wyjatkami
 - globalne, 239
 - lokalne, 238
 - zasada
 - Controller, 433
 - Creator, 433
 - DRY, 435
 - High Cohesion, 434
 - Indirection, 435
 - Information Expert, 433
 - KISS, 436
 - Low Coupling, 433
 - odwrócenia zależności, 431
 - otwarte-zamknięte, 428
 - podstawienia Liskov, 428

- pojedynczej odpowiedzialności, 427
- Polymorphism, 434
- Protected Variations, 435
- Pure Fabrication, 434
- separacji interfejsów, 429
- YAGNI, 437
- zasoby niezarządzone, 70
- zastosowanie
 - atrybutu Serializable, 94
 - ViewModel, 402
- zbiór zasad
 - GRASP, 432
 - SOLID, 427
- zdarzenia, 75, 464, 487
- zdobywanie
 - linków, 502
 - sztucznych linków, 509
- zgłaszanie wyjątków, 77
- zmienne
 - domniemane, 105
 - tylko do odczytu, 29
- znaczniki HTML, 454, 459, 500
- zwalnianie zasobów niezarządzanych, 70

Ż

- żądania
 - AJAX, 491
 - HTTP, 445
- żądanie
 - do aplikacji ASP.NET, 189
 - GET, 196, 445
 - POST, 196, 446

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>

C# 6.0 i MVC 5

Dynamiczny rozwój sieci, która wkracza w coraz to nowe obszary życia, pozwala nam korzystać z większości usług z dowolnego miejsca na świecie i za pośrednictwem każdego urządzenia z dostępem do internetu. Aby użytkownik mógł swobodnie poruszać się po sieci, konieczne jest rozwijanie coraz bardziej zaawansowanych, wygodniejszych i szybszych aplikacji internetowych działających po stronie serwera. Do niedawna większość aplikacji tego rodzaju była pisana w takich językach jak PHP, Python lub Java, obecnie ogromną popularność zdobywają w tej dziedzinie język C# i platforma MVC.

Jeśli dysponujesz ogólną wiedzą na temat programowania i baz danych, opanowałeś lub właśnie opanowujesz podstawy języka C# i chcesz poznać platformę MVC oraz możliwości, jakie oferuje w zakresie tworzenia aplikacji webowych, sięgnij po tę książkę. Nie stanowi ona klasycznego podręcznika do programowania, lecz raczej zbiór praktycznych wskazówek i objaśnień pozwalających w krótkim czasie rozpocząć przygodę z tworzeniem aplikacji internetowych, a następnie logicznie ją kontynuować. Dzięki lekturze krok po kroku poznasz proces powstawania kodu aplikacji i jego wdrażania przy użyciu różnych narzędzi. Poznaj świat aplikacji internetowych!

- Podstawy języka C# i jego możliwości
- Wzorce projektowe i architektoniczne oraz ich stosowanie
- Możliwości webowych platform firmy Microsoft
- Tworzenie aplikacji i serwisów internetowych oraz ich wdrażanie
- Optymalizowanie aplikacji pod kątem wymagań wyszukiwarek i serwisów społecznościowych
- Budowa własnego portalu

Buduj doskonałe serwisy internetowe w języku C# z platformą MVC!

Helion	
22947	numer katalogowy
księgarnia internetowa	
	http://helion.pl
zamówienia telefoniczne	
	0 801 339900
	0 601 339900
Informatyka w najlepszym wydaniu	

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

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

sięgnij po WIĘCEJ



KOD KORZYŚCI

ISBN 978-83-246-9496-9



9 788324 694969

cena: 89,00 zł