

O'REILLY®

Wydanie II

Laravel w działaniu

Budowa nowoczesnych
aplikacji w PHP



Helion 

Matt Stauffer

Tytuł oryginału: Laravel Up & Running A Framework for Building Modern PHP Apps, 2nd Edition

Tłumaczenie: Piotr Rajca

ISBN: 978-83-283-7633-5

© 2021 Helion S.A.

Authorized Polish translation of the English edition Laravel: Up & Running 2E ISBN 9781492041214 © 2019 Matt Stauffer.

This translation is published and sold by permission of O'Reilly Media, Inc., which owns or controls all rights to publish and sell the same.

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

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

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

Autor oraz wydawca 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 wydawca nie ponoszą również żadnej odpowiedzialności za ewentualne szkody wynikłe z wykorzystania informacji zawartych w książce.

Helion S.A.

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/larav2>

Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

Printed in Poland.

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

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

Spis treści

Wstęp	17
1. Dlaczego Laravel?	23
Po co używać frameworku?	23
„Zrobię to po swojemu”	24
Konsekwencja i elastyczność	24
Krótka historia WWW i frameworków PHP	24
Ruby on Rails	25
Zalew frameworków PHP	25
Dobre i złe strony CodeIgnitera	25
Laravel 1, 2 i 3	26
Laravel 4	26
Laravel 5	26
Co jest tak wyjątkowe w Laravelu?	27
Filozofia Laravela	27
Jak Laravel zapewnia programistom szczęście?	28
Społeczność użytkowników Laravela	29
Jak działa Laravel?	29
Dlaczego Laravel?	31
2. Konfiguracja środowiska roboczego do korzystania z Laravela	33
Wymagania systemowe	33
Composer	34
Lokalne środowisko programistyczne	34
Laravel Valet	34
Laravel Homestead	35
Tworzenie nowego projektu Laravela	36
Instalowanie Laravela przy użyciu narzędzia instalacyjnego	36
Instalowanie Laravela przy użyciu opcji create-project Composera	36
Lambo: polecenie „laravel new” o supermocach	36

Struktura katalogów Laravela	37
Katalogi	38
Pliki	39
Konfiguracja	40
Plik .env	41
Wszystko gotowe	43
Testowanie	43
TL;DR	44
3. Trasowanie i kontrolery	45
Szybkie wprowadzenie do MVC, czasowników HTTP oraz architektury REST	45
Czym jest MVC?	45
Czasowniki HTTP	46
Czym jest REST?	47
Definiowanie tras	48
Czasowniki tras	50
Obsługa tras	50
Parametry tras	51
Nazwy tras	52
Grupy tras	54
Oprogramowanie pośrednie	55
Prefiksy ścieżek	57
Trasy awaryjne	58
Trasy do poddomen	58
Prefiksy przestrzeni nazw	59
Prefiksy nazw	59
Trasy podpisane	59
Podpisywanie trasy	60
Modyfikacja tras w celu umożliwienia ich podpisywania	61
Widoki	61
Bezpośrednie zwracanie prostych tras przy użyciu Route::view()	62
Stosowanie twórców widoków w celu udostępniania danych we wszystkich widokach	63
Kontrolery	63
Pobieranie danych wejściowych od użytkowników	66
Wstrzykiwanie zależności do kontrolerów	67
Kontrolery zasobów	68
Kontrolery zasobów API	69
Kontrolery jednej akcji	70
Wiązanie tras i modelu	70
Niejawne wiązanie tras i modelu	71
Niestandardowe wiązanie trasy i modelu	71

Przechowywanie tras w pamięci podręcznej	72
Oszukiwanie metod obsługi formularzy	73
Czasowniki HTTP w Laravelu	73
Oszukiwanie metody HTTP w formularzach HTML	73
Zabezpieczanie przed atakami CSRF	74
Przekierowania	75
Metoda redirect()->to()	76
Metoda redirect()->route()	76
Metoda redirect()->back()	77
Inne metody przekierowań	77
Metoda redirect()->with()	78
Przerywanie przekierowania	79
Niestandardowe odpowiedzi	80
Metoda response()->make()	80
Metody response()->json() oraz ->jsonp()	80
Metody response()->download(), ->streamDownload() oraz ->file	80
Testowanie	81
TL;DR	82
4. Szablony Blade	83
Wyświetlanie danych	84
Struktury sterujące	85
Konstrukcje warunkowe	85
Pętle	86
Dziedziczenie szablonów	88
Definiowanie sekcji przy użyciu @section/@show oraz @yield	88
Dołączanie fragmentów widoków	90
Stosowanie stosów	92
Stosowanie komponentów i gniazd	93
Twórcy widoków i wstrzykiwanie usług	95
Wiązanie danych z widokami przy użyciu twórców	96
Wstrzykiwanie usług w szablonach Blade	98
Dyrektywy niestandardowe	99
Parametry w niestandardowych dyrektywach	100
Przykład: Stosowanie niestandardowych dyrektyw w aplikacji wielodostępnej	101
Łatwiejsze niestandardowe dyrektywy reprezentujące instrukcje „if”	102
Testowanie	102
TL;DR	103

5. Bazy danych i Eloquent	105
Konfiguracja	105
Połączenia z bazami danych	106
Opcje konfiguracji baz danych	107
Definiowanie migracji	108
Wykonywanie migracji	115
Wypełnianie tabel danymi	116
Tworzenie klasy wypełniającej	116
Fabryki modeli	117
Budowniczy zapytań	122
Podstawowe sposoby użycia fasady DB	122
Wykonywanie kodu SQL	123
Sekwencje zapytań tworzone przy użyciu budowniczego	124
Transakcje	132
Wprowadzenie do mechanizmu Eloquent	134
Tworzenie i definiowanie modeli mechanizmu Eloquent	135
Pobieranie danych w modelach Eloquent	136
Wstawianie danych i ich aktualizacja w mechanizmie Eloquent	138
Usuwanie rekordów przy użyciu mechanizmu Eloquent	142
Zasięgi	144
Dostosowywanie interakcji z polami przy użyciu akcesorów, mutatorów i rzutowania atrybutów	147
Kolekcje mechanizmu Eloquent	150
Serializacja w mechanizmie Eloquent	153
Związki w mechanizmie Eloquent	155
Rekordy podrzędne aktualizujące znaczniki czasu rekordów nadrzędnych	166
Zdarzenia mechanizmu Eloquent	168
Testowanie	170
TL;DR	171
6. Komponenty frontendowe	173
System budowania Mix	173
Struktura katalogów systemu Mix	175
Uruchamianie systemu Mix	175
Co nam daje Mix?	176
Predefiniowane ustawienia frontendowe i generowanie kodu uwierzytelniającego	182
Predefiniowane ustawienia frontendowe	183
Generowanie kodu uwierzytelniającego	184
Podział na strony	184
Podział na strony wyników pobieranych z baz danych	184
Ręczna obsługa podziału na strony	185

Pojemniki komunikatów	186
Nazwane pojemniki komunikatów	188
Funkcje pomocnicze do obsługi łańcuchów, tworzenia liczb mnogich i lokalizowania	188
Funkcje pomocnicze do obsługi łańcuchów i tworzenia liczb mnogich	188
Lokalizowanie	189
Testowanie	193
Testowanie pojemników komunikatów i błędów	193
Tłumaczenie i lokalizowanie	193
TL;DR	193
7. Gromadzenie i obsługa danych od użytkowników	195
Wstrzykiwanie obiektu Request	195
Metoda \$request->all()	196
Metody \$request->except() oraz \$request->only()	196
Metoda \$request->has()	197
Metoda \$request->input()	197
Metody \$request->method() oraz \$request->isMethod()	198
Dane wejściowe w formie tablicy	198
Dane wejściowe w formacie JSON (i metoda \$request->json())	198
Dane tras	200
Obiekt Request	200
Dane z parametrów trasy	200
Pliki przesyłane na serwer	200
Walidacja	203
Wywoływanie validate() na rzecz obiektu Request	203
Ręczna walidacja	205
Obiekty reguł niestandardowych	206
Wyświetlanie komunikatów o błędach walidacji	206
Żądania formularzy	207
Tworzenie żądań formularzy	207
Korzystanie z żądań formularzy	208
Masowe przypisanie w obiektach mechanizmu Eloquent	209
{{ a {!!	210
Testowanie	211
TL;DR	212
8. Artisan i Tinker	213
Prezentacja Artisana	213
Podstawowe polecenia Artisana	214
Opcje	215
Grupy poleceń	215

Pisanie niestandardowych poleceń Artisan	218
Przykładowe polecenie	220
Argumenty i opcje	221
Korzystanie z danych wejściowych	222
Wyświetlanie komunikatów z prośbą o podanie danych	224
Wyświetlanie wyników	225
Polecenia implementowane przy użyciu domknięć	226
Wywoływanie poleceń Artisana w normalnym kodzie	227
Tinker	228
Serwer zrzutów Laravela	228
Testowanie	229
TL;DR	230
9. Uwierzytelnianie i autoryzacja użytkowników	231
Model User i jego migracja	232
Stosowanie globalnej funkcji pomocniczej auth() i fasady Auth	235
Kontrolery związane z uwierzytelnianiem	235
Kontroler RegisterController	236
Kontroler LoginController	237
Kontroler ResetPasswordController	238
Kontroler ForgotPasswordController	239
Kontroler VerificationController	239
Metoda Auth::routes()	239
Generowany kod obsługujący uwierzytelnianie	240
„Zapamiętaj mnie”	241
Ręczne uwierzytelnianie użytkowników	242
Ręczne wylogowywanie użytkownika	243
Unieważnianie sesji na innych urządzeniach	243
Oprogramowanie pośrednie związane z uwierzytelnianiem	243
Weryfikacja adresu e-mail	244
Dyrektywy mechanizmu Blade związane z uwierzytelnianiem	245
Strażniki	246
Zmiana domyślnego strażnika	246
Stosowanie innych strażników bez zmieniania domyślnego	247
Dodawanie nowych strażników	247
Strażniki żądań obsługiwanych przy użyciu domknięć	247
Tworzenie niestandardowego dostawcy użytkowników	248
Niestandardowi dostawcy użytkowników dla nierelacyjnych baz danych	248
Zdarzenia związane z uwierzytelnianiem	249
Autoryzacja (ACL) i role	250
Definiowanie reguł autoryzacji	250
Fasada Gate (lub wstrzykiwanie obiektu Gate)	251

Kontrola dostępu do zasobów	252
Oprogramowanie pośrednie służące do autoryzacji	252
Autoryzacja w kontrolerach	253
Sprawdzanie instancji klasy User	254
Testy w szablonach Blade	255
Przechwytywanie testów	255
Polityki	256
Testowanie	258
TL;DR	260
10. Żądania, odpowiedzi i oprogramowanie pośrednie	263
Cykl życia żądania w Laravelu	263
Ładowanie aplikacji	264
Dostawcy usług	265
Obiekt Request	266
Pobieranie obiektu Request w Laravelu	267
Pobieranie podstawowych informacji na temat żądania	268
Obiekt Response	272
Tworzenie i stosowanie obiektu Response w kontrolerach	272
Wyspecjalizowane typy odpowiedzi	273
Laravel i oprogramowanie pośrednie	278
Wprowadzenie do oprogramowania pośredniego	278
Tworzenie niestandardowego oprogramowania pośredniego	279
Wiązanie oprogramowania pośredniego	281
Przekazywanie parametrów do oprogramowania pośredniego	284
Zaufane serwery pośredniczące	285
Testowanie	286
TL;DR	287
11. Kontener	289
Krótkie wprowadzenie do wstrzykiwania zależności	289
Wstrzykiwanie zależności w Laravelu	291
Globalna funkcja pomocnicza app()	291
W jaki sposób kontener określa zależności?	292
Dodawanie klas do kontenera	293
Określanie zależności z wykorzystaniem domknięć	293
Stosowanie singletonów, nazw zastępczych i instancji	294
Wiązanie konkretnej instancji z interfejsem	295
Wiązanie kontekstowe	296
Wstrzykiwanie do konstruktora w plikach Laravla	296
Wstrzykiwanie do metod	297

Fasady a kontener	298
Jak działają fasady?	299
Fasady czasu rzeczywistego	300
Dostawcy usług	301
Testowanie	301
TL;DR	302
12. Testowanie	303
Podstawy testowania	303
Nazewnictwo testów	308
Testowanie środowiska	308
Cechy używane podczas testowania	309
RefreshDatabase	309
WithoutMiddleware	310
DatabaseMigrations	310
DatabaseTransactions	310
Proste testy jednostkowe	310
Testy aplikacji — jak działają?	311
Klasa TestCase	311
Testy HTTP	312
Proste testy stron przy użyciu <code>\$this->get()</code> oraz innych wywołań HTTP	312
Testowanie API przy użyciu <code>\$this->getJSON()</code> i innych wywołań HTTP zwracających dane JSON	313
Stosowanie asercji z obiektem <code>\$response</code>	314
Uwierzytelnianie odpowiedzi	315
Kilka innych modyfikacji testów żądań HTTP	316
Obsługa wyjątków w testach aplikacji	317
Testy baz danych	317
Stosowanie fabryk modeli w testach	318
Wypełnianie tabel danymi w testach	318
Testowanie innych systemów Laravela	318
Imitacje zdarzeń	319
Imitacje magistrali i kolejki	320
Imitacje poczty elektronicznej	321
Imitacje powiadomień	322
Imitacje magazynów	323
Stosowanie atrap	323
Krótkie wprowadzenie do stosowania atrap	324
Krótkie wprowadzenie do Mockery	324
Imitowanie innych fasad	326

Testowanie komend Artisana	327
Sprawdzanie składni poleceń Artisana	328
Testy w przeglądarce	329
Wybór odpowiedniego narzędzia	329
Testowanie przy użyciu narzędzia Dusk	330
TL;DR	340
13. Tworzenie API	341
Podstawy tworzenia API typu REST-like dla zasobów JSON	341
Organizacja kontrolera i zwracanie danych w formacie JSON	342
Odczyt i wysyłanie nagłówków	346
Wysyłanie nagłówków odpowiedzi w Laravelu	347
Odczytywanie nagłówków żądania w Laravelu	347
Dzielenie wyników modeli Eloquent na strony	347
Sortowanie i filtrowanie	348
Sortowanie wyników zwracanych przez API	349
Filtrowanie wyników zwracanych przez API	350
Przekształcanie wyników	351
Pisanie własnych przekształceń	352
Obsługa zagnieżdżania i związków przy użyciu niestandardowych przekształceń	353
Zasoby API	355
Tworzenie klasy zasobu	355
Kolekcje zasobów	356
Zagnieżdżanie powiązanych zasobów	358
Stosowanie podziału na strony z zasobami API	358
Warunkowe stosowanie atrybutów	359
Dodatkowe możliwości dostosowywania zasobów API	360
Uwierzytelnianie API przy użyciu Laravel Passport	360
Krótkie wprowadzenie do OAuth 2.0	360
Instalowanie Passporta	361
API Passporta	362
Różne typy przyznań	363
Zarządzanie klientami i żetonami	
przy użyciu API Passporta oraz komponenty Vue	371
Zasięgi	373
Wdrażanie Passportu	375
Uwierzytelnianie przy użyciu żetonu API	375
Dostosowywanie odpowiedzi 404	376
Wyzwalanie trasy awaryjnej	377
Testowanie	377
Testowanie Passporta	378
TL;DR	378

14. Przechowywanie i pobieranie	379
Magazyny plików — lokalne oraz w chmurze	379
Konfiguracja dostępu do plików	379
Stosowanie fasady Storage	380
Dodawanie innych dostawców systemów plików	382
Podstawowe sposoby przesyłania plików na serwer i operacji na nich	382
Proste sposoby pobierania plików	384
Sesje	384
Dostęp do sesji	384
Dostępne metody instancji sesji	385
Mechanizm składowania „flash” sesji	387
Pamięć podręczna	387
Dostęp do pamięci podręcznej	388
Dostępne metody instancji pamięci podręcznej	388
Ciasteczka	390
Ciasteczka w Laravelu	390
Dostęp do narzędzi obsługi ciasteczek	390
Rejestracja	393
Kiedy i dlaczego używać dzienników?	394
Zapisywanie wpisów w dziennikach	394
Kanały dzienników	395
Wyszukiwanie pełnotekstowe przy użyciu Laravel Scout	397
Instalacja pakietu Scout	398
Oznaczanie modelu do indeksowania	398
Przeszukiwanie indeksu	398
Kolejki i Scout	399
Wykonywanie operacji bez indeksowania	399
Warunkowe indeksowanie modeli	399
Ręczne wyzwalanie indeksowania z poziomu kodu	400
Ręczne wyzwalanie indeksowania z poziomu wiersza poleceń	400
Testowanie	400
Przechowywanie plików	400
Pamięć podręczna	403
Ciasteczka	403
Rejestrowanie	404
Pakiet Scout	405
TL;DR	405
15. Poczta elektroniczna i powiadomienia	407
Poczta elektroniczna	407
Obsługa „klasycznej” poczty elektronicznej	408
Podstawowe sposoby stosowania składni „wysyłalnej”	408

Szablony e-maili	410
Metody dostępne wewnątrz metody build()	411
Załączniki oraz wstawiane obrazy	412
Wiadomości e-mail w formacie Markdown	413
Wyswietlanie wysyłanych wiadomości w przeglądarce	415
Kolejki	415
Praca lokalna	416
Powiadomienia	417
Definiowanie metody via() w klasie powiadomienia	420
Wysyłanie powiadomień	420
Umieszczanie powiadomień w kolejce	421
Domyślnie dostępne typy powiadomień	421
Testowanie	425
Poczta elektroniczna	425
Powiadomienia	426
TL;DR	426
16. Kolejki, zadania, zdarzenia, rozgłaszanie i mechanizm planowania	427
Kolejki	427
Dlaczego kolejki?	428
Podstawowa konfiguracja kolejek	428
Zadania umieszczane w kolejkach	428
Uruchamianie procesu roboczego	432
Obsługa błędów	432
Kontrola kolejki	435
Kolejki wspierające działanie innych elementów Laravela	435
Laravel Horizon	435
Zdarzenia	436
Zgłaszanie zdarzeń	437
Nasłuchiwanie zdarzeń	438
Rozgłaszanie zdarzeń przy użyciu WebSocket i Laravel Echo	441
Konfiguracja i ustawienia	442
Rozgłaszanie zdarzeń	442
Odbieranie wiadomości	445
Zaawansowane narzędzia do rozgłaszania	447
Laravel Echo (kliencki kod JavaScript)	451
Mechanizm planujący	455
Dostępne typy zadań	455
Dostępne ramki czasu	456
Definiowanie stref czasowych w zaplanowanych poleceniach	458
Blokowanie i nakładanie	458

Obsługa wyników generowanych przez zadania	458
Punkty zaczepienia zadań	459
Testowanie	459
TL;DR	461
17. Funkcje pomocnicze i kolekcje	463
Funkcje pomocnicze	463
Tablice	463
Łańcuchy znaków	465
Ścieżki aplikacji	467
Adresy URL	468
Różne funkcje	470
Kolekcje	472
Podstawy	472
Kilka metod	474
TL;DR	478
18. Ekosystem Laravela	479
Narzędzia opisane w niniejszej książce	479
Valet	479
Homestead	479
Instalator Laravela	480
Mix	480
Dusk	480
Passport	480
Horizon	480
Echo	481
Narzędzia nieopisane w tej książce	481
Forge	481
Envoyer	481
Cashier	482
Socialite	482
Nova	483
Spark	483
Lumen	483
Envoy	484
Telescope	484
Inne zasoby	484
Słowniczek	487

Dlaczego Laravel?

W początkowych latach dynamicznie rozwijającego się internetu pisanie aplikacji internetowych wyglądało całkowicie inaczej niż obecnie. W tamtych czasach programiści byli odpowiedzialni nie tylko za pisanie kodu stanowiącego unikalną logikę biznesową aplikacji, lecz także za implementację komponentów powszechnie występujących na wszystkich witrynach, takich jak: uwierzytelnianie, weryfikacja danych wprowadzanych przez użytkowników, dostęp do baz danych, obsługa szablonów itd.

Obecnie programiści mają do dyspozycji dziesiątki frameworków oraz tysiące komponentów i bibliotek. Bardzo często można usłyszeć, że w czasie potrzebnym do nauczenia się korzystania z jednego frameworku pojawiają się trzy nowe (i rzekomo lepsze), które mają go zastąpić.

Stwierdzenie: „Dlatego, że istnieje” może być wystarczającym wyjaśnieniem dla wspinania się na jakiś górski szczyt, jeśli jednak chodzi o wybór konkretnego frameworku do zastosowania albo o podjęcie decyzji, czy w ogóle jakiegoś używać, istnieją lepsze uzasadnienia. Czy warto nawet zadawać pytanie, dlaczego używać frameworków? Albo bardziej konkretnie, dlaczego używać Laravela?

Po co używać frameworku?

Bardzo łatwo można odpowiedzieć na pytanie, jakie korzyści wynikają ze stosowania poszczególnych komponentów lub pakietów dostępnych dla programistów używających języka PHP. W przypadku pakietów za stworzenie i utrzymanie tych izolowanych fragmentów kodu o ściśle zdefiniowanym przeznaczeniu odpowiada ktoś inny i teoretycznie rzecz biorąc, ta osoba znacznie lepiej rozumie działanie tego komponentu niż my, wzięwszy pod uwagę czas, jaki możemy na to poświęcić.

Frameworki podobne do Laravela, takie jak Symfony, Lumen czy Slim, stanowią kolekcje komponentów przygotowanych przez niezależnych programistów, powiązanych ze sobą charakterystycznym dla danego frameworku „spoiwem” — plikami konfiguracyjnymi, dostawcami usług, narzuconą strukturą katalogów czy skryptami uruchamiającymi aplikacje. A zatem, ogólnie rzecz ujmując, zaleta korzystania z frameworków polega na tym, że ktoś za nas podejmuje decyzje nie tylko o tym, jakich komponentów użyć, lecz także *jak te komponenty powinny ze sobą współdziałać*.

„Zrobię to po swojemu”

Założmy, że podjęliśmy decyzję o rozpoczęciu prac nad nową aplikacją internetową, którą zamierzamy stworzyć bez korzystania z jakiegokolwiek frameworku. Gdzie musimy zacząć? No cóż... taka aplikacja pewnie będzie musiała obsługiwać żądania HTTP, więc będziemy musieli przejrzeć wszystkie dostępne biblioteki obsługujące żądania i odpowiedzi HTTP i wybrać jedną z nich. Następnie będziemy musieli wybrać mechanizm odpowiedzialny za trasowanie żądań HTTP. A przy okazji będziemy musieli wybrać jakiś sposób zapisu plików z informacjami o konfiguracji tras. A jakiej składni w nich używać? Gdzie je umieszczać? A co z kontrolerami? Gdzie mają być zapisywane i jak je wczytywać? No cóż, zapewne będziemy potrzebowali jakiegoś kontenera wstrzykiwania zależności, który pozwoliłby nam wybierać kontenery oraz obsługiwać ich zależności. Ale jaki to ma być kontener?

Co więcej, jeśli poświęcimy czas, by odpowiedzieć na te wszystkie pytania, i pomyślnie stworzymy aplikację, to jaki będzie wpływ podjętych przez nas decyzji na kolejnych programistów, którzy będą się tą aplikacją zajmować? Co się stanie, jeśli będziemy mieli na głowie cztery albo kilkanaście takich aplikacji opartych na samodzielnie skonstruowanych frameworkach i będziemy musieli pamiętać, gdzie w każdej z nich umieszczać kontrolery i jaka jest składnia plików z informacjami o trasach?

Konsekwencja i elastyczność

Frameworki rozwiązują ten problem, udostępniając dogłębnie przemyślaną odpowiedź na pytanie: „Którego komponentu należy użyć w tym miejscu?” oraz zapewniając, że wszystkie wybrane komponenty będą ze sobą dobrze współdziałać. Co więcej, frameworki dostarczają konwencji pozwalających zmniejszyć ilość kodu, który musi zrozumieć programista dopiero zaczynający pracę nad danym projektem — jeśli będziemy rozumieć, jak działa określanie tras w jednym projekcie napisanym w Laravelu, będziemy rozumieli, jak ono działa także w innych projektach używających tego frameworku.

Kiedy ktoś nakazuje przygotowanie nowego frameworku dla każdego nowego projektu, tak naprawdę postuluje zapewnienie możliwości *kontrolowania* tego, co wejdzie w skład fundamentu, na którym będziemy budowali te aplikacje. To oznacza, że najlepsze frameworki będą nam zapewniać nie tylko solidną podstawę, lecz także swobodę w dostosowywaniu ich do własnych potrzeb. I jak pokażę w dalszej części tej książki, właśnie te cechy sprawiają, że Laravel jest tak unikalny.

Krótką historia WWW i frameworków PHP

Ważnym elementem udzielenia odpowiedzi na pytanie: „Dlaczego Laravel?” jest zrozumienie historii tego frameworka oraz zrozumienie historii jego poprzedników. Nim Laravel zyskał popularność, istniało wiele innych frameworków, i to nie tylko pisanych w języku PHP, lecz także w środowiskach internetowych.

Ruby on Rails

David Heinemeier Hansson udostępnił pierwszą wersję frameworku Ruby on Rail w roku 2004 i od tamtego czasu trudno jest znaleźć jakikolwiek framework do tworzenia aplikacji internetowych, który nie byłby w jakimś stopniu wzorowany na Ruby on Rails.

Framework ten spopularyzował zastosowanie wzorca MVC, API typu RESTful korzystających z formatu JSON, konwencji konfiguracyjnych, wzorca ActiveRecord, jak również wielu innych narzędzi i konwencji, które wywarły ogromny wpływ na sposób, w jaki programiści obecnie tworzą aplikacje internetowe — zwłaszcza pod względem szybkości ich tworzenia.

Zalew frameworków PHP

Dla większości programistów było oczywistym, że Rails oraz inne podobne frameworki do tworzenia aplikacji były powiewem przyszłości, a frameworki PHP, w tym takie, które jawnie naśladowały Rails, faktycznie zaczęły powstawać bardzo szybko.

Pierwszym z nich był CakePHP udostępniony w 2005 roku, szybko po nim pojawiły się frameworki Symfony, CodeIgniter, Zend Framework oraz Kohana (stanowiąca modyfikację CodeIgnitera). W roku 2008 pojawił się framework Yii, a w 2010 Aura i Slim. W roku 2011 pojawiły się FuelPHP i Laravel, które nie tyle były modyfikacjami CodeIgnitera, ile stanowiły dla niego alternatywy.

Niektóre z tych frameworków były bardziej podobne do Ruby on Rails, koncentrowały się na mechanizmach odwzorowań obiektowo-relacyjnych (ORM), strukturach MVC oraz innych rozwiązaniach mających zapewnić szybkość tworzenia aplikacji. Inne, takie jak Symfony i Zend, koncentrowały się bardziej na korporacyjnych wzorcach projektowych i zastosowaniach związanych z handlem elektronicznym.

Dobre i złe strony CodeIgnitera

CakePHP oraz CodeIgniter były dwoma wczesnymi frameworkami PHP, które najbardziej otwarcie podchodziły do tematu stopnia, w jakim były wzorowane na Ruby on Rails. CodeIgniter szybko zdobył popularność i w 2010 roku był prawdopodobnie najpopularniejszym z niezależnych frameworków PHP.

Był prosty, łatwy w użyciu, dysponował niesamowitą dokumentacją i prężną społecznością. Jednak dość wolno adaptował nowoczesne technologie i wzorce projektowe, a wraz z rozwojem frameworków do tworzenia aplikacji internetowych oraz wszelkiego rodzaju narzędzi wspomagających pisanie w języku PHP CodeIgniter powoli zaczął odstawać od konkurentów pod względem zaawansowania technologicznego i wbudowanych możliwości. W odróżnieniu od wielu innych frameworków rozwojem CodeIgnitera zajmowała się firma, której stosunkowo dużo czasu zajęło podjęcie decyzji o wykorzystaniu nowych możliwości języka PHP 5.3, takich jak przestrzenie nazw, przeniesienie bazy kodu do serwisu GitHub, a następnie zapewnienie możliwości korzystania z Composera. W 2010 roku Taylor Otwell, twórca Laravela, był już na tyle niezadowolony z CodeIgnitera, że zdecydował się na stworzenie własnego frameworku.

Laravel 1, 2 i 3

Pierwsza wersja beta Laravela 1 została udostępniona w czerwcu 2011 roku i w całości napisano ją od podstaw. Była wyposażona we własny mechanizm ORM (Eloquent), mechanizm trasowania oparty na domknięciach (wzorowany na Ruby Sinatra), nowoczesny system rozszerzeń i funkcje pomocnicze wspomagające tworzenie formularzy, weryfikację poprawności danych, uwierzytelnianie itd.

Prace nad rozwojem frameworka postępowały szybko i jego kolejne wersje, Laravel 2 i Laravel 3, pojawiły się odpowiednio w listopadzie 2011 roku i lutym 2012 roku. Wprowadzono w nich kontrolery, obsługę testów jednostkowych, narzędzia wiersza poleceń, kontroler odwrócenia sterowania (Ioc), obsługę związków oraz migracji w mechanizmie Eloquent.

Laravel 4

Tworząc wersję Laravel 4, Taylor od podstaw przepisał cały framework. W tamtym czasie Composer, obecnie wszechobecny menedżer pakietów PHP, był na najlepszej drodze, by stać się standardem, a Taylor zauważył zalety, jakie da przepisanie frameworka w postaci kolekcji komponentów dostarczanych i scalanych przy użyciu Composera.

Taylor stworzył więc kolekcję komponentów o nazwie kodowej *Illuminate* i w maju 2013 roku udostępnił framework Laravel 4 o całkowicie nowej strukturze. Kod tej nowej wersji Laravela nie był udostępniany w formie pakietu, który programista pobierał w całości, lecz raczej sam framework pobierał większość swych komponentów z Symfony (kolejnego frameworka udostępniającego swoje pakiety tak, że mogły być pobierane przez inne rozwiązania) i Illuminate, używając do tego celu Composera.

W Laravelu 4 zostały także wprowadzone kolejki, komponent poczty elektronicznej, fasady (*facades*) i wstępna inicjalizacja baz danych. A ponieważ od tego momentu Laravel był zależny od komponentów Symfony, ogłoszono, że także on będzie udostępniany w sześciomiesięcznym cyklu wydawniczym.

Laravel 5

W listopadzie 2014 roku miała zostać udostępniona wersja 4.3 Laravela, jednak wraz z postępem prac stało się oczywiste, że znaczenie wprowadzanych w nim zmian zasługiwało na nadanie tej wersji frameworka nowego numeru, i tak w lutym 2015 roku został udostępniony Laravel 5.

W Laravelu 5 wprowadzono nową strukturę katalogów, usunięto funkcje pomocnicze do tworzenia formularzy i kodu HTML, wprowadzono interfejsy kontraktu, nowe widoki, mechanizm Socialite do uwierzytelniania w mediach społecznościowych, mechanizm Elixir do kompilacji zasobów dodatkowych, mechanizm Scheduler do uproszczenia korzystania z systemowego narzędzia cron, dotenv do ułatwienia zarządzania środowiskiem, wprowadzono żądania formularzy i zupełnie nowe narzędzie REPL (pętla odczyt-przetworzenie-wyświetlenie, *read-evaluate-print loop*). Od tamtego czasu Laravel zyskał jeszcze nowe możliwości i dojrzał, jednak nie wprowadzono w nim już żadnych poważniejszych zmian.

Co jest tak wyjątkowe w Laravelu?

A zatem co takiego odróżnia Laravela od konkurentów? Dlaczego warto używać jednocześnie więcej niż jednego frameworka PHP? Przecież większość z nich i tak używa komponentów pochodzących z Symfony, nieprawdaż? Zastanówmy się więc przez chwilę, co sprawia, że Laravel jest taki wyjątkowy.

Filozofia Laravela

Wystarczy rzucić okiem na materiały reklamowe i pliki README Laravela, żeby zacząć doceniać jego wartość. Taylor używa słów związanych ze światłem, takich jak „Illuminate” (oświecać, rozjaśniać, być źródłem natchnienia) i „Spark” (iskra). Ale stosuje także takie określenia jak „Artisans” (rzemieślnicy), „Elegant” (elegancki) i dalej: „Breath of fresh air” (powiew świeżego powietrza), „Fresh start” (nowy początek). I w końcu: „Rapid” (szybki) i „Warp speed” (prędkość nadświetlna).

Dwoma wartościami frameworku, na które kładzie się największy nacisk, są: szybkość tworzenia aplikacji oraz zadowolenie programistów. Taylor opisał język „Artisan” jako celowo odróżniający się na tle innych praktycznych zalet. Genezę tego sposobu myślenia można zauważyć w pytaniu zadany przez niego w serwisie StackExchange (<https://softwareengineering.stackexchange.com/questions/90954/is-there-any-benefit-to-obsession-with-making-code-look-pretty>), w którym stwierdził: „Czasami spędzam absurdalne ilości czasu (godziny), starając się zapewnić, by «kod wyglądał ładnie» — stanowiącym próbę znalezienia uzasadnienia dla wkładania wysiłku w dążenie do uzyskania lepszych doznań zapewnianych przez sam wygląd kodu. Oprócz tego często wspominał o znaczeniu, jakie ma ułatwianie programistom realizowania swoich pomysłów oraz eliminacji niepotrzebnych barier na drodze do tworzenia wspaniałych produktów.

Laravel w swojej istocie jest frameworkiem, który daje programistom wielkie możliwości i potrzebne narzędzia. Jego celem jest dostarczanie przejrzystego, prostego i pięknego kodu oraz możliwości, które programiści będą mogli szybko poznać, opanować i zastosować do tworzenia własnego kodu — przejrzystego, prostego i trwałego.

Koncepcja tworzenia z myślą o programistach jest wyraźnie zauważalna w materiałach dotyczących Laravela. W dokumentacji frameworku napisano, że „najlepszy kod tworzą szczęśliwi programiści”. Przez jakiś czas nieoficjalnym sloganem frameworku było „szczęście programistów od momentu rozpoczęcia pisania aż do wdrożenia”. Oczywiście, twórcy każdego narzędzia i frameworku będą twierdzić, że zależy im na szczęściu programistów. Jednak nadawaniu temu zagadnieniu *pierwszoplanowego*, nie drugorzędnego znaczenia miało ogromny wpływ na styl i rozwój Laravela. O ile dla innych frameworków pierwszoplanowym celem może być czystość architektury lub zgodność z celami i wartościami korporacyjnych zespołów programistycznych, w przypadku Laravela podstawowym celem jest służyć pojedynczym programistom. Nie oznacza to wcale, że Laravel nie pozwala na tworzenie aplikacji o czystej architekturze lub gotowych do zastosowań korporacyjnych, jednak w jego przypadku uzyskanie tych celów nie będzie się odbywać kosztem czytelności i łatwości analizy bazy kodu.

Jak Laravel zapewnia programistom szczęście?

Stwierdzenie, że zależy nam na szczęściu programistów, to jedna sprawa, ale zupełnie czym innym jest faktycznie o nie zadbać — wymaga to bowiem znalezienia odpowiedzi na pytanie, jakie cechy frameworku w największym stopniu uprzykrzają programistom życie, a które będą ich uszczęśliwiać. Laravel stara się ułatwiać pracę programistom na kilka sposobów.

Przed wszystkim Laravel jest frameworkiem służącym do szybkiego tworzenia aplikacji. Oznacza to, że koncentruje się on na zapewnieniu łagodnej (płytkiej) krzywej trudności nauki oraz na minimalizacji kroków, jakie trzeba wykonać od rozpoczęcia prac nad nową aplikacją do jej opublikowania. Wszystkie czynności najczęściej wykonywane podczas tworzenia aplikacji internetowych, zaczynając od interakcji z bazami danych, przez uwierzytelnianie i obsługę poczty elektronicznej, a kończąc na wykorzystaniu pamięci podręcznej, są upraszczane przez komponenty udostępniane przez framework. Jednak komponenty Laravela nie są tak wspaniałe same z siebie; zapewniają one spójny API i przewidywalne struktury w obrębie całej aplikacji. Oznacza to, że kiedy będziemy próbowali zrobić w Laravelu coś nowego, najprawdopodobniej skończy się to stwierdzeniem „... i to po prostu działa”.

Jednak nie chodzi o sam framework. Laravel udostępnia cały ekosystem narzędzi służących do budowania i uruchamiania aplikacji. Do prac programistycznych służą Homestead i Valet, do zarządzania serwerem — Forge, a do zaawansowanego wdrażania — Envoyer. Do tego dochodzi zestaw dodatkowych pakietów: Cashier do obsługi płatności i subskrypcji, Echo do obsługi technologii WebSockets, Scout do obsługi wyszukiwania, Passport do obsługi API służących do uwierzytelniania, Dusk do testowania interfejsów użytkownika, Socialite do obsługi logowania w mediach społecznościowych, Horizon do monitorowania kolejek, Nova do budowania paneli administracyjnych oraz Spark do ułatwiania tworzenia rozwiązań typu SaaS¹. Laravel stara się umożliwić programistom uniknięcie wykonywania powtarzających się czynności, tak by mogli skoncentrować się na tym, co jest unikalne.

Oprócz tego Laravel stosuje podejście „konwencje ponad konfigurację”. Oznacza to, że jeśli będziemy chcieli ograniczyć się do domyślnego sposobu działania Laravela, będziemy musieli zrobić znacznie mniej niż w przypadku stosowania innych frameworków, które zmuszają do deklarowania wszystkich ustawień niezależnie od tego, czy programista chce użyć zalecanej konfiguracji. Stworzenie projektu korzystającego z Laravela zajmuje mniej czasu niż stworzenie analogicznego rozwiązania korzystającego z większości innych frameworków PHP.

Co więcej, Laravel w bardzo dużym stopniu koncentruje się na zapewnieniu prostoty. Jeśli chcemy, nic nie stoi na przeszkodzie, by korzystać ze wstrzykiwania zależności, atrapy, repozytoriów, wzorców Data Mapper, Command Query Responsibility Segregation oraz wszelkich innych bardziej zaawansowanych wzorców architektonicznych. Jednak w odróżnieniu od innych frameworków, które mogą sugerować stosowanie takich rozwiązań we wszystkich projektach, Laravel, jego dokumentacja i społeczność preferują rozpoczynanie od najprostszych możliwych implementacji — globalnej

¹ *Software as a Service* — oprogramowanie jako usługa — *przyp. tłum.*

funkcji tu, fasady tam, wzorca ActiveRecord gdzieś indziej. W ten sposób programiści mogą tworzyć możliwie najprostsze aplikacje zaspokajające ich potrzeby, bez ograniczania możliwości ich stosowania w bardziej złożonych środowiskach.

Interesującym zagadnieniem pokazującym, w jaki sposób Laravel różni się od innych frameworków PHP, jest to, że jego twórca i społeczność używających go programistów są bardziej związani i zainspirowani językiem Ruby, frameworkiem Ruby on Rails oraz językami programowania funkcyjnego, niż językiem Java. Obecnie w społeczności programistów PHP istnieje silny nurt dążenia do złożoności, tworzenia bardziej rozbudowanego kodu i wykorzystania tych aspektów języka, które przypominają Javę. Laravel wydaje się znajdować po przeciwnej stronie i stara się wykorzystywać bardziej ekspresyjne, dynamiczne i prostsze praktyki programistyczne i możliwości języka.

Spółeczność użytkowników Laravela

Jeśli niniejsza książka stanowi pierwszy kontakt Czytelnika ze społecznością użytkowników Laravela, to może oczekiwać czegoś naprawdę szczególnego. Jednym z czynników wyróżniających ten framework, który w wielkim stopniu przyczynił się do jego rozwoju i popularności, jest życzliwa i chętna do pomocy społeczność jego użytkowników. Zaczynając od wideoporadników *Laracast* Jeffreya Waya (<https://laracasts.com/>), przez serwis Laravel News (<https://laravel-news.com>), kanały Slack, IRC i Discord poświęcone Laravelowi, znajomych na Twitterze, blogerów, po podcasty na konferencjach Laracon — Laravel dysponuje bogatą i żywą społecznością, pełną osób, które używały go od samego początku jego istnienia, i takich, które dopiero zaczynają go poznawać. I bynajmniej nie jest to żaden przypadek:

Od samego początku istnienia Laravela przyświecała mi idea, że wszyscy chcą się czuć czegoś częścią. To naturalny ludzki instynkt, że chcemy należeć i być akceptowanymi przez jakąś grupę podobnie myślących osób. A zatem dzięki zapewnieniu frameworkowi pewnej osobowości i aktywnej działalności w ramach społeczności, takie poczucie może się przekształcić w społeczność.

— Taylor Otwell, wywiad dla Product and Support

Taylor od samego początku rozumiał, że projekt typu *open-source* do odniesienia sukcesu potrzebuje dwóch rzeczy: dobrej dokumentacji i przyjaznej społeczności. I te dwie rzeczy są cechami charakterystycznymi Laravela.

Jak działa Laravel?

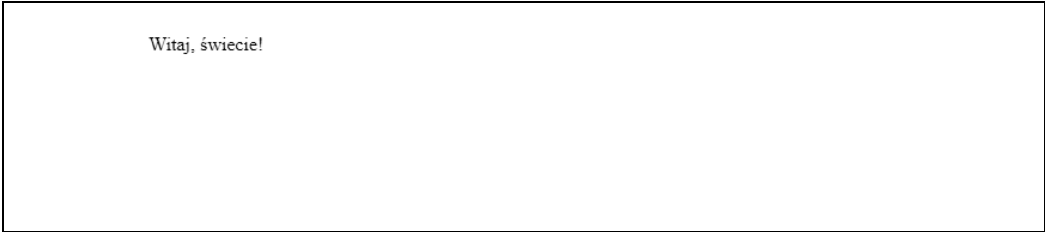
Jak na razie wszystko, co napisałem, było całkowicie abstrakcyjne. Można by się zapytać: „A co z kodem?”. Przyjrzyjmy się zatem prostej aplikacji (Listing 1.1.), tak byśmy mogli zobaczyć, jak w praktyce wygląda codzienne stosowanie frameworku Laravel.

Listing 1.1. „Witaj, świecie!” w pliku `routes/web.php`

```
<?php

Route::get('/', function () {
    return 'Witaj, świecie!';
});
```

Najprostszą rzeczą, jaką można zrobić w aplikacji Laravela, jest zdefiniowanie trasy i zwracanie wyniku za każdym razem, kiedy ktoś się do tej trasy odwoła. Jeśli zainicjujemy na komputerze zupełnie nową aplikację Laravela, zdefiniujemy trasę przedstawioną na listingu 1.1, a następnie uruchomimy aplikację z poziomu katalogu *public*, uzyskamy w pełni funkcjonalną aplikację typu „Witaj, świecie!” (patrz rysunek 1.1).



Witaj, świecie!

Rysunek 1.1. Zwracanie łańcucha „Witaj, świecie!” w aplikacji Laravela

W przypadku wykorzystania kontrolerów aplikacja wygląda całkiem podobnie, co pokazałem na listingu 1.2.

Listing 1.2. Aplikacja typu „Witaj, świecie!” napisana przy użyciu kontrolera

```
// Plik: routes/web.php
<?php
Route::get('/', 'WelcomeController@index');

// Plik app/Http/Controllers/WelcomeController.php
<?php

namespace App\Http\Controllers;

class WelcomeController extends Controller
{
    public function index()
    {
        return 'Witaj, świecie!';
    }
}
```

A jeśli powitanie będzie przechowywane w bazie danych, aplikacja także będzie wyglądać podobnie (jak pokazałem na listingu 1.3):

Listing 1.3. Aplikacja typu „Witaj, świecie!” z wieloma powitaniami pobieranymi z bazy danych

```
// Plik routes/web.php
<?php

use App\Greeting;

Route::get('create-greeting', function () {
    $greeting = new Greeting;
    $greeting->body = 'Witaj, świecie!';
    $greeting->save();
});
```

```

Route::get('first-greeting', function () {
    return Greeting::first()->body;
});

// Plik app/Greeting.php
<?php

namespace App;

use Illuminate\Database\Eloquent\Model;

class Greeting extends Model
{
    //
}

// Plik database/migrations/2020_07_19_010000_create_greeting_table.php
<?php

use Illuminate\Database\Schema\Blueprint;
use Illuminate\Database\Migrations\Migration;

class CreateGreetingsTable extends Migration
{
    public function up()
    {
        Schema::create('greetings', function (Blueprint $table) {
            $table->bigIncrements('id');
            $table->string('body');
            $table->timestamps();
        });
    }
    public function down()
    {
        Schema::dropIfExists('greetings');
    }
}

```

Przykład przedstawiony na listingu 1.3 może być nieco onieśmielający — jeśli faktycznie tak jest, po prostu go pomiń. O wszystkim, co dzieje się w tych przykładach, dowiesz się z kolejnych rozdziałów książki; jednak już na pierwszy rzut oka widać, że dzięki wykorzystaniu zaledwie kilku wierszy kodu możliwe jest przygotowanie migracji bazy danych, modelu oraz pobranie z bazy rekordów. To naprawdę jest aż tak proste.

Dlaczego Laravel?

A zatem dlaczego Laravel?

Gdyż pozwala nam urzeczywistniać pomysły bez pisania niepotrzebnego kodu, przy użyciu do tego celu nowoczesnych standardów, oddając nam do dyspozycji aktywną społeczność oraz zestaw narzędzi o ogromnych możliwościach.

Jak również dlatego, Drogi Programisto, że zasługujesz, by być szczęśliwym.

PROGRAM PARTNERSKI

— GRUPY 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
Helion 

Laravel: framework godny profesjonalisty!

Framework Laravel cieszy się rosnącą popularnością wśród programistów aplikacji w języku PHP. Jest szybki, potężny i elastyczny, można się go błyskawicznie nauczyć, a generowany przez niego kod cechują przejrzystość i czytelność. Co więcej, Laravel skupia wokół siebie zaangażowaną społeczność, która z jednej strony systematycznie tworzy nowe narzędzia i pakiety wzbogacające ten framework, z drugiej — chętnie śpieszy z pomocą w rozwiązywaniu problemów. Jest to więc świetny, dynamicznie rozwijający się zestaw narzędzi, który spodoba się każdemu profesjonalnemu programiście PHP, ceniącemu efektywność i wysoką jakość tworzonego kodu.

Ta książka stanowi praktyczne i kompletne wprowadzenie do Laravela (zawiera informacje dotyczące Laravela 5.8). Dzięki niej osoby dysponujące pewnym doświadczeniem w tworzeniu kodu PHP błyskawicznie rozpoczną tworzenie znakomitych aplikacji. Znalazły się tutaj zarówno ogólne informacje na temat korzystania z tego frameworka, jak i przykłady szczegółowych zastosowań. Omówiono także mnóstwo narzędzi i bibliotek wzbogacających funkcjonalność Laravela: Dusk, Horizon, Artisan, Mix czy Passport. Opisano też interfejsy pozwalające na dostęp do systemu plików, sesji, ciasteczek, pamięci podręcznej i wyszukiwania oraz narzędzia do korzystania z kolejek, implementacji zadań, zdarzeń i publikowania zdarzeń WebSocket.

W książce między innymi:

- tworzenie szablonów w Blade
- generowanie, walidacja, normalizacja oraz filtrowanie danych użytkownika
- praca z bazami danych za pomocą Eloquent
- testowanie kodu PHP: PHPUnit, Mockery oraz Dusk
- tworzenie API typu RESTful
- inne narzędzia i biblioteki Laravela

Matt Stauffer — programista i trener, uważany za jednego z najlepszych znawców Laravela. Pracuje w firmie konsultingowej Tighten, w której zajmuje stanowisko dyrektora technicznego. Uwielbia rozmawiać o programowaniu i blogować, jest też twórcą i gospodarzem *The Five-Minute Geek Show*. Często występuje jako prelegent na branżowych konferencjach.

 helion.pl	<i>Sprawdź nasze szkolenia!</i> SZKOLENIA  AKADEMIA IT & BUSINESS HELIONSZKOLENIA.PL	KOD KORZYŚCI Sięgnij po więcej! ▶  ISBN 978-83-283-7633-5  9 788328 376335
INFORMATYKA W NAJLEPSZYM WYDANIU		Cena: 99,00 zł