

Paweł Kamiński

Laravel

Wstęp do programowania
aplikacji internetowych



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: Małgorzata Kulik

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

Grafika na okładce została wykorzystana za zgodą Shutterstock.com

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/larwpa>

Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

ISBN: 978-83-283-5130-1

Copyright © Helion 2019

Printed in Poland.

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

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

Spis treści

0 autorze	7
Konwencja książki	9
Rozdział 1. Wstęp do Laravela	13
1.1. Dlaczego Laravel?	13
1.2. Czym jest więc ten wspaniały, mityczny wręcz Laravel?	14
Najważniejsze funkcjonalności i cechy Laravela	15
Rozdział 2. Przygotowanie środowiska	17
2.1. Instalacja XAMPP dla Windows	18
2.2. Instalacja XAMPP w systemie Ubuntu	25
2.3. Instalacja XAMPP w systemie OSX	31
2.4. Composer	38
Instalacja Composera w systemie Windows	39
Instalacja Composera w systemie OSX i Ubuntu	42
Zasada działania	44
2.5. IDE	46
NetBeans	46
Eclipse	48
Atom	50
Sublime Text	50
2.6. GIT	52
Instalacja GIT w Windows	53
Instalacja GIT w Ubuntu	56
Instalacja GIT w OSX	57
Przygotowanie repozytorium	58
2.7. Podsumowanie przygotowań	61

Rozdział 3. Laravel — pierwsze kroki	63
3.1. Proces instalacji Laravela	63
3.2. Konfiguracja Virtual Hosts	67
Windows	67
Ubuntu i OSX	68
3.3. Pierwszy program	69
Konfiguracja aplikacji	72
Laravel Artisan	76
3.4. Architektura MVC	77
3.5. Routing w Laravelu	79
Routing — rodzaje żądań	81
Żądania sparametryzowane	82
Nazwy tras	83
Prefiksy	84
Co uległo zmianie?	84
Przekierowania w routingu	84
Powiązanie modelu z parametrem	85
Resources	86
Projekt	86
3.6. Tworzenie kontrolerów	87
Service Container	87
Dependency injection	90
Cykl życia żądania	91
Middleware	92
Fasada	95
Projekt	96
3.7. System szablonów Blade	98
Projekt	99
Komponenty i gniazda	100
Instrukcje Blade	100
Dołączanie zewnętrznego kodu	103
Projekt	104
Rozdział 4. Baza danych i model	109
4.1. Podstawy baz danych	110
Konfiguracja bazy w Laravelu	111
Tworzenie i wywoływanie migracji	113
Projekt	117
Table Seeders	119

4.2. Eloquent ORM	121
Tworzenie modelu	123
Odwołanie do modelu z kontrolera	126
Pobieranie pojedynczego rekordu	128
Projekt	129
Dodawanie nowych rekordów	132
Aktualizacja rekordów	134
Usuwanie rekordów	136
Przeszukiwanie tabel	137
Kolekcje — dostępne operacje	139
Projekt	141
4.3. Relacje	141
Dodawanie relacji jeden do jednego	142
Dodawanie relacji jeden do wielu	146
Relacja wiele do wielu	152
Relacje typu has-many-through	160
Eager loading	162
4.4. Query Builder	163
Projekt	166
Pobieranie danych z wielu tabel	169
4.5. Wzorzec Repository	171
Budowa wzorca	171
Przykład użycia	174
Rozdział 5. Formularze	179
5.1. Dodawanie danych za pomocą formularzy	179
5.2. Dodawanie danych powiązanych relacją	185
5.3. Formularze edycji danych	191
5.4. Walidacja formularzy	196
Form Request	199
5.5. Internacjonalizacja	202
Rozdział 6. Rozszerzone możliwości Laravela	209
6.1. Laravel Mix	209
6.2. Usługi — services	214
6.3. Events	221
6.4. Commands	225
6.5. Klasy Helpers	228

6.6. Obsługa poczty e-mail	232
Konfiguracja Laravela	237
Wysyłka wiadomości e-mail	238
Rozdział 7. Autentykacja	243
Rozdział 8. Wstęp do budowy API	255
8.1. API Resources	256
Testowanie API — wstęp do testów manualnych	259
Rozdział 9. Publikowanie aplikacji	265
Zakończenie	269
Skorowidz	270

Rozdział 5.

Formularze

Wyjątkowo tytuł rozpoczynającego się rozdziału nie jest nazwą technologii czy przykładem terminu zarezerwowanego tylko i wyłącznie dla branży informatycznej. Wręcz przeciwnie, formularze są elementem otaczającym człowieka z każdej strony, towarzyszą zwykłej wizycie u lekarza, uzupełnianiu dziennika lekcyjnego, podpisywaniu listy obecności w pracy, o formularzach możemy nawet mówić w momencie tworzenia trywialnej listy zakupów — można więc powiedzieć, że są codziennością człowieka.

Wszystkie przypadki tworzenia tego typu struktur są przykładem uzupełniania, podawania porcji danych, które są następnie obsługiwane w różny sposób — lista zakupów jest odczytywana w supermarkecie, lista obecności w pracy na pewno przyda się kierownikowi, natomiast dziennik lekcyjny jest niezbędny do podsumowania wyników ucznia. Każde podawanie danych, uzupełnianie formularzy ma konkretny cel i znaczenie.

Nie inaczej jest w przypadku formularzy na stronach internetowych — służą one do pobierania danych bezpośrednio od użytkownika, są elementem interfejsu graficznego, umożliwiają interakcję pomiędzy użytkownikiem a projektowanym systemem.

Mimo że rozpoczynający się rozdział nie jest jednym z najdłuższych, to samo jego istnienie świadczy o randze formularzy — elementy tego typu są powszechnie używane we współczesnych aplikacjach internetowych, a ich obsługa (przygotowanie, zapisanie pobranych danych, walidacja) jest zagadnieniem ważnym i wartym omówienia.

W poszczególnych podrozdziałach spróbujemy stworzyć i obsłużyć kilka formularzy różnego typu — od formularzy pobierających dane w celu stworzenia nowych rekordów po formularze edycji. Przedstawimy również sposoby walidowania pobranych danych.

5.1. Dodawanie danych za pomocą formularzy

Jedną z elementarnych funkcjonalności formularzy jest możliwość obsługi danych, które przeznaczone są do bezpośredniego zapisania w bazie. W naszym systemie przykładem takich danych mogą być obiekty książek i autorów. Są to proste obiekty, niewymagające skomplikowanej logiki, i świetnie nadadzą się jako pierwszy przykład formularza.

Prace rozpoczynamy od stworzenia nowego widoku, oczywiście umieścimy w nim kod HTML formularza, a jako nazwy użyjemy standardowej etykiety operacji, którą implementujemy. Cała ścieżka do pliku widoku będzie więc prezentowała się następująco: `/resources/views/books/create.blade.php`, a zawartość będzie zgodna z poniższym listingiem:

```
@extends('template')
@section('title')
    Lista książek
@endsection

@section('content')
<div class="container">

    <h2>Dodawanie książki</h2>
    <form action="{{ action('BookController@store') }}" method="POST" role="form">
    <input type="hidden" name="_token" value="{{ csrf_token() }}" />
    <div class="form-group">
        <label for="name">Tytuł książki</label>
        <input type="text" class="form-control" name="name" />
    </div>
    <div class="form-group">
        <label for="name">Rok publikacji</label>
        <input type="text" class="form-control" name="year" />
    </div>
    <div class="form-group">
        <label for="name">Miejsce wydania</label>
        <input type="text" class="form-control" name="publication_place" />
    </div>
    <div class="form-group">
        <label for="name">Liczba stron</label>
        <input type="text" class="form-control" name="pages" />
    </div>
    <div class="form-group">
        <label for="name">Cena</label>
        <input type="text" class="form-control" name="price" />
    </div>

    <input type="submit" value="Dodaj" class="btn btn-primary"/>
    </form>
</div>
@endsection
```

Analizując powyższy kod, a właściwie jego centralną część, można wywnioskować, że:

- W celu stworzenia nowego formularza użyto standardowego znacznika języka HTML o nazwie `form` — w znaczniku tym użyto atrybutu `action`, w którym zadeklarowano, który kontroler (`BookController`) i metoda (`store`) ma obsługiwać zapis. Dalej w atrybucie `method` zdefiniowano typ żądania na `POST`.
- Jako pierwszego elementu formularza użyto ukrytego pola (zapis `type=hidden`), którego wartość definiowana jest przez metodę `csrf_token()` — jest to metoda Laravela, która dodaje do formularza token. Dzięki niemu w momencie obsługi żądania Laravel może sprawdzić, czy żądanie faktycznie ma źródło w formularzu. Cała ta funkcjonalność to ochrona przed atakami i próbami wysyłania zmodyfikowanych żądań HTTP.

- Kolejne pola formularzy to standardowe formatki typu `input`. Co interesujące, nazwy pól są zgodne z ich odpowiednikami z bazy danych, co przyda nam się w kodzie obsługi formularza.
- Formularz zatwierdzany jest przyciskiem stworzonym za pomocą znacznika `input` o atrybucie `type` równym `submit`.

Wszystko to pokazuje, że praktycznie jedyną różnicą przy budowie formularzy obsługiwanych przez Laravela jest zdefiniowanie kodu, który będzie go obsługiwał, a także dodanie kodu CSRF, który zabezpieczy nasz kod obsługi przed przesyłaniem żądań spoza formularza — oczywiście w przypadku braku dodania tego ukrytego pola będzie to oznaczało przerwanie operacji i zakończenie pracy skryptu.

Przejdźmy do kodu obsługi, czyli kontrolera `BookController`. Jeszcze niedawno, gdy zajmowaliśmy się modelem i tworzeniem nowych książek, w metodzie `create` tymczasowo dodaliśmy kod, który tworzył nowy obiekt książki. Obiekt ten miał na twardo zapisane parametry dodawanej pozycji. Teraz, gdy mamy już do dyspozycji formularze, możemy zmodyfikować nasz kod i sprawić, by metoda `create` nie robiła nic innego jak tylko wyświetlała widok, który niedawno stworzyliśmy:

```
class BookController extends Controller
{
  (...)
  public function create(BookRepository $bookRepo)
  {
    return view('books/create');
  }
}
```

Obsługę samego zapisywania danych musimy za to przenieść do metody `store`, której kod prezentuje się następująco:

```
public function store(Request $request, BookRepository $bookRepo)
{
  $data = [
    "name" => $request->input('name'),
    "year" => $request->input('year'),
    "publication_place" => $request->input('publication_place'),
    "pages" => $request->input('publication_place'),
    "price" => $request->input('publication_place'),
  ];
  $booksList = $bookRepo->create($data);

  return redirect('books');
}
```

Metoda ta za argumenty przyjmuje dwa obiekty — `Request` i `BookRepository`. Oba są wstrzykiwane bezpośrednio z kontenera, nie musimy ich jawnie tworzyć. Ciało opisywanej metody to przede wszystkim zmienna tablicowa. Tak jak przed modyfikacjami przechowuje ona wartości poszczególnych pól tworzonego obiektu — w naszym przypadku oczywiście książki. Każdemu z pól przypisywane są wartości pobierane z obiektu żądania (w tym celu wykorzystywana jest metoda `input`).

Obiekt jest zapisywany w bazie za pomocą metody `create` repozytorium — tu już nic nie ulega zmianie. Warto również przypomnieć, że w momencie tworzenia widoku formularza zwrócono baczność uwagę na nadawanie polom nazw identycznych jak ich odpowiednikom w bazie danych. Dzięki temu można teraz delikatnie przebudować kod metody `store` i zamiast ręcznie przypisywać kolejnym polom obiektu ich wartości z formatek, użyć metody `all`, która zwróci całą, gotową tablicę danych:

```
public function store(Request $request, BookRepository $bookRepo)
{
    $data = $request->all();
    $booksList = $bookRepo->create($data);

    return redirect('books');
}
```

Powyższy kod zrealizuje dokładnie tę samą funkcjonalność, co poprzedni — jest jednak zdecydowanie krótszy i bardziej zwięzły. W celu przetestowania nowego kodu należy otworzyć stronę:

`http://biblioteka.local/books/create`

Na ekranie powinien wyświetlić się formularz nowej książki, uzupełniony danymi. Przykład widoczny jest na rysunku 5.1.

Książki ▾ Wypożyczenia Autorzy

Dodawanie książki

Tytuł książki

Rok publikacji

Miejsce wydania

Liczba stron

Cena

RYСУNEK 5.1. Widok formularza nowej książki

Zatwierdzenie formularza powinno zaowocować dodaniem wpisu do bazy i przekierowaniem na listę wszystkich pozycji, która już powinna zawierać nowy rekord (rysunek 5.2).

Książki ▾	Wypożyczenia	Autorzy						
Hobbit	2001	Warszawa	310	29.99	Podgląd	Edycja	Usuń	
Kolor magli	2005	Katowice	205	24.99	Podgląd	Edycja	Usuń	
Quo vadis	2001	Warszawa	650	59.99	Podgląd	Edycja	Usuń	
Pan Tadeusz	1999	Kraków	450	39.99	Podgląd	Edycja	Usuń	
Czarny Dom	2010	Warszawa	648	59.99	Podgląd	Edycja	Usuń	
Folwark zwierzęcy	1990	Warszawa	200	19.99	Podgląd	Edycja	Usuń	

RYSUNEK 5.2. Lista książek z widoczną nową pozycją

Tę samą czynność — czyli przebudowę dodawania nowego rekordu — spróbujemy wykonać dla tabeli z listą autorów. Analogicznie czynności rozpoczynamy od stworzenia pliku widoku `/resources/views/authors/create.blade.php` z zawartością:

```
@extends('template')
@section('title')
    Lista książek
@endsection

@section('content')
<div class="container">
    <h2>Dodawanie autora</h2>
    <form action="{{ action('AuthorController@store')}}" method="POST" role="form">
    <input type="hidden" name="_token" value="{{ csrf_token() }}" />
    <div class="form-group">
        <label for="name">Nazwisko</label>
        <input type="text" class="form-control" name="lastname" />
    </div>
    <div class="form-group">
        <label for="name">Imię</label>
        <input type="text" class="form-control" name="firstname" />
    </div>
    <div class="form-group">
        <label for="name">Data urodzenia</label>
        <input type="text" class="form-control" name="birthday" />
    </div>
    <div class="form-group">
        <label for="name">Gatunki</label>
        <input type="text" class="form-control" name="genres" />
    </div>

    <input type="submit" value="Dodaj" class="btn btn-primary"/>
    </form>
</div>
@endsection
```

Zmian jest niewiele — dodane zostały odpowiednie pola (zgodne z bazą danych), a także odpowiedni nagłówek i oczywiście zadeklarowano kod, który będzie służył do obsługi. Podobnie jak w przypadku tabeli książek, modyfikujemy metodę `create` kontrolera `AuthorController`, tak by jej jedyną funkcją było zwracanie widoku formularza.

Zmianie ulega oczywiście również metoda `store`. Ponieważ w tym kontrolerze i modelu autora nie zaimplementowaliśmy jeszcze wzorca repozytorium, odwołujemy się bezpośrednio do Eloquent ORM — tworzymy nowy obiekt, za pomocą metody `fill` uzupełniamy jego pola danymi pobranymi bezpośrednio z formularza i zapisujemy go w bazie, używając przy tym metody `save`:

```
class AuthorController extends Controller
{
    (...)
    public function create()
    {
        return view('authors/create');
    }
    public function store(Request $request)
    {
        $data = $request->all();
        $author = new Author();
        $author->fill($data);
        $author->save();

        return redirect('authors');
    }
}
```

Efekt działania powyższego kodu można łatwo przetestować. Wywołanie poniższego adresu URL:

`http://biblioteka.local/authors/create`

wyświetli formularz nowego autora (rysunek 5.3), którego uzupełnienie i zatwierdzenie powinno zapisać zmiany w bazie.

Książki > Wypożyczenia > Autorzy

Dodawanie autora

Nazwisko
Orwell

Imię
George

Data urodzenia
1903-06-25

Gatunki
satyra

Dodaj

RYСУNEK 5.3. Widok formularza nowego autora

Potwierdzeniem zapisania autora w bazie będzie umiejscowienie go na liście (rysunek 5.4).

Jak widać, dodawanie nowych rekordów nie jest wyjątkowo trudne, gdyż w praktyce używamy lekko zmodyfikowanych klasycznych formularzy zbudowanych w kodzie HTML. Możemy

Nazwisko autora	Imię autora	Data urodzin	Gatunki	Książki
Straub	Peter	1943-03-02	horror, thriller	Czarny Dom
King	Stephen	1947-09-21	horror, thriller	Czarny Dom
Orwell	George	1903-06-25	satyra	

RYСУNEK 5.4. Lista autorów z widocznym nowym wpisem

więc korzystać z wszystkich funkcjonalności i technik tego języka, co powoduje, że ewentualna migracja już istniejącego kodu do kodu, który spełniałby swoją rolę we współpracy z Laravelem, nie jest czynnością trudną.

Na tym etapie ostatnią modyfikacją projektu będzie dodanie nowych pozycji do paska nawigacji. W tym celu należy dodać do pliku `/resources/views/template.blade.php` nowe odnośniki, dla książek:

```
<li class="nav-item dropdown">
  <a class="nav-link dropdown-toggle" data-toggle="dropdown"
    ↪href="{{ URL::to('books') }}">Książki</a>
  <div class="dropdown-menu">
    <a class="dropdown-item" href="{{ URL::to('books/cheapest') }}">Top 3 najtańszych</a>
    <a class="dropdown-item" href="{{ URL::to('books/longest') }}">Top 3 najdłuższych</a>
    <a class="dropdown-item" href="{{ URL::to('books') }}">Wszystkie</a>
    <a class="dropdown-item" href="{{ URL::to('books/create') }}">Dodaj nową</a>
  </div>
</li>
```

a także dla autorów:

```
<li class="nav-item dropdown">
  <a class="nav-link dropdown-toggle" data-toggle="dropdown"
    ↪href="{{ URL::to('authors') }}">Autorzy</a>
  <div class="dropdown-menu">
    <a class="dropdown-item" href="{{ URL::to('authors') }}">Wszyscy</a>
    <a class="dropdown-item" href="{{ URL::to('authors/create') }}">Dodaj nowego</a>
  </div>
</li>
```

Spróbujmy teraz przyjrzeć się, jak można stworzyć bardziej rozbudowane formularze, w tym z danymi, które są połączone relacją.

5.2. Dodawanie danych powiązanych relacją

Jakkolwiek dodawanie danych okazało się niezbyt trudnym zadaniem, w celu dodania nowych rekordów, które są w relacji z innymi obiektami, należy poczynić pewne przygotowania i nieco dopasować kod napisany w poprzednim rozdziale. Spróbujmy stworzyć formularz, który doda do systemu funkcjonalność tworzenia nowych wypożyczeń.

Gdy zerkniemy do bazy danych projektowanego systemu, a szczególnie do tabeli loans, to szybko przypomnimy sobie, że w jej rekordach widnieje klucz obcy o nazwie book_id. Jest to oczywiście klucz wykorzystywany do stworzenia wiązania z tabelą books, czyli obiektem książki. Tworząc formularz nowego wypożyczenia, musimy pamiętać, że użytkownik nie jest w stanie operować na kluczach — nasz interfejs powinien pozwolić na wybór odpowiedniej książki, na przykład za pomocą tytułu, wykorzystując wybraną formatkę kodu HTML — zapewne w przypadku wyboru książki lista rozwijana będzie najtrafniejszym wyborem. Idąc dalej tym tropem, dochodzimy do wniosku, że widok formularza powinien zawierać listę rozwijaną z wszystkimi dostępnymi w systemie książkami, które oczywiście muszą zostać pobrane również z bazy danych. Zajrzyjmy więc do kontrolera LoanController:

```
class LoanController extends Controller
{
    (...)
    public function create()
    {
        $books = Book::all();
        return view('loans/create', ['books' => $books]);
    }
}
```

Jak widać na załączonym listingu, modyfikacji uległa metoda create, która teraz, podobnie jak w poprzednich przypadkach książek i autorów, jako główne zadanie ma wywołanie widoku loans/create. Dodatkowo, w celu realizacji listy rozwijanej z książkami w opisywanej metodzie pobierana jest z bazy lista wszystkich dostępnych pozycji, a następnie przekazywana bezpośrednio do widoku.

Dzięki temu możliwe jest stworzenie samego pliku widoku (*/resources/views/loans/create.blade.php*), którego zawartość wygląda następująco:

```
@extends('template')
@section('title')
    Lista książek
@endsection
@section('content')
<div class="container">

<h2>Dodawanie wypożyczenia</h2>
<form action="{{ action('LoanController@store') }}" method="POST" role="form">
<input type="hidden" name="_token" value="{{ csrf_token() }}" />
<div class="form-group">
    <label for="name">Tytuł książki</label>
    <select type="text" class="form-control" name="book_id">
        @foreach ($books as $book)
            <option value="{{ $book->id }}">{{ $book->name }}</option>
        @endforeach
    </select>
</div>
<div class="form-group">
    <label for="name">Dane wypożyczającego</label>
    <input type="text" class="form-control" name="client" />
</div>
<div class="form-group">
    <label for="name">Data wypożyczenia</label>
```

```

        <input type="text" class="form-control" name="loaned_on" />
    </div>
    <div class="form-group">
        <label for="name">Przewidywany zwrot</label>
        <input type="text" class="form-control" name="estimated_on" />
    </div>
    <input type="submit" value="Dodaj" class="btn btn-primary"/>
</form>
</div>
@endsection('content')

```

Zmiany dotyczą oczywiście dodania nowego pola, stworzonego za pomocą znacznika `select`, któremu nadano nazwę `book_id` (podobnie jak klucz obcy tabeli `loans`). W celu wyświetlenia listy użyto pętli `foreach`, która w każdym swoim przebiegu tworzy jeden element `option` o atrybucie `value` równym identyfikatorowi książki, a także etykietę równej jej nazwie. Dzięki temu wyświetlona lista będzie uzupełniona pełnymi danymi w postaci nazwy książki, natomiast serwer otrzyma wartości klucza.

Kolejny etap to implementacja metody `store` kontrolera książek:

```

public function store(Request $request)
{
    $book = Book::find($request->input('book_id'));
    $data = $request->all();

    $loan = new Loan();
    $loan->fill($data);
    $book->loans()->save($loan);
    $loan->save();

    return redirect('loans');
}

```

Jak widać, w pierwszej kolejności na podstawie identyfikatora pobranego z listy rozwijanej wyszukiwany jest obiekt książki, natomiast reszta kodu jest już analogiczna do wersji standardowej — tworzony jest nowy obiekt wypożyczenia (co warte uwagi, nie skorzystano tu z wzorca repozytorium), do którego pól przypisywane są wartości pobrane z formularza. Następnie za pomocą metod `loans` i `save` zapisujemy dane relacyjne, a następnie sam obiekt wypożyczenia.

W celu potwierdzenia skuteczności napisanego kodu warto teraz przejść do adresu:

```
http://biblioteka.local/loans/create
```

i wpisać przykładowe dane wypożyczenia. Na rysunku 5.5 umieszczono widok uzupełnionego formularza, gotowego do wysyłki.

Jego zatwierdzenie powinno uruchomić proces zapisywania danych w bazie, czego dowodem będzie zaktualizowana lista, widoczna na rysunku 5.6.

W podobny sposób należy zaktualizować również formularz dodawania nowej książki. Zgodnie z logiką, przy jej dodawaniu chcielibyśmy wybrać autorów — co ważne, może ich być kilku.

Książki ▾ Wypożyczenia Autorzy ▾

Dodawanie wypożyczenia

Tytuł książki
Hobbit

Dane wypożyczającego
Adam Kowalski, Storczykowa 1A, Białystok

Data wypożyczenia
2019-11-01

Przewidywany zwrot
2019-11-14

Dodaj

RYСУNEK 5.5. *Uzupełniony formularz dodawania nowego wypożyczenia*

Książki ▾ Wypożyczenia Autorzy ▾

Nazwa książki	Data wypożyczenia	Data planowanego zwrotu	Data zwrotu	Dane klienta
Hobbit	2019-04-10	2019-04-24		Tadeusz Jakacki, Jaworowa 13, 00-900 Warszawa, 600 111 222
Pan Tadeusz	2019-11-01	2019-11-14		Adam Kowalski, Storczykowa 1A, Białystok

RYСУNEK 5.6. *Lista wypożyczeń z widocznym nowym wypożyczeniem*

W tym celu musimy dobrać odpowiedni element interfejsu, który umożliwiłby wybór kilku pozycji — może to być lista checkboxów czy też lista wyboru z dodanym atrybutem `multiple`.

W pierwszej kolejności modyfikacji ulega sam kontroler, a dokładniej metoda `create`:

```
use App\Models\Author;
(...)
class BookController extends Controller
{
    (...)

    public function create(BookRepository $bookRepo)
    {
        $authors = Author::all();
        return view('books/create', ['authors' => $authors]);
    }
}
```

Podobnie jak w przypadku wypożyczeń, tak i tutaj należy pamiętać, że przy tworzeniu formularza niezbędna będzie lista — tym razem autorów. Dane pobieramy oczywiście z bazy i przekazujemy do odpowiedniego widoku formularza. Pamiętajmy o dodaniu przestrzeni nazw modelu autora — nie był on jeszcze wykorzystywany w tym kontrolerze.

Kolejne zmiany to modyfikacja widoku nowej książki (plik `/resources/views/books/create.blade.php`):

```
(...)
<h2>Dodawanie książki</h2>
  <form action="{{ action('BookController@store') }}" method="POST" role="form">
    <input type="hidden" name="_token" value="{{ csrf_token() }}" />

<div class="form-group">
  <label for="name">Autor</label>
  <select type="text" class="form-control" name="author_id[]" multiple>
    @foreach ($authors as $author)
      <option value="{{ $author->id }}">{{ $author->lastname }} {{ $author-
>firstname }}</option>
    @endforeach
  </select>
</div>
(...)
```

Podobnie jak w przypadku wypożyczeń, także i tu zastosowano:

- Listę rozwijaną `select`, którą zatytułowano `author_id[]`. Użyte nawiasy kwadratowe świadczą o tym, że aplikacja oczekuje tablicy wyników, a nie jednej konkretnej wartości.
- Pętlę typu `foreach`, która dodaje kolejne znaczniki `option` reprezentujące jednego autora.
- Użycie znacznika `multiple` w znaczniku `select` umożliwia wybór więcej niż jednego autora.
- Każde z pól listy rozwijanej ma wartość równą identyfikatorowi autora, a także etykiety w postaci nazwiska i imienia.

Ostatni element układanki to oczywiście zmiany w sposobie obsługi zapisywania formularza. Szybki podgląd kontrolera `BookController` i metody `store` uświadamia, że modyfikacji należy dokonać w repozytorium — wszak właśnie użyliśmy tego wzorca w celu oddzielenia działań na bazie od samego kodu kontrolera.

Zaktualizowany kod repozytorium książki prezentuje się następująco:

```
(...)
class BookRepository extends BaseRepository{

public function __construct(Book $model){
  $this->model = $model;
}

public function create(array $data) {
  $book = Book::create($data);

  if(isset($data['author_id'])) {
    $book->authors()->sync($data['author_id']);
  }
  return $book;
}
(...)
```

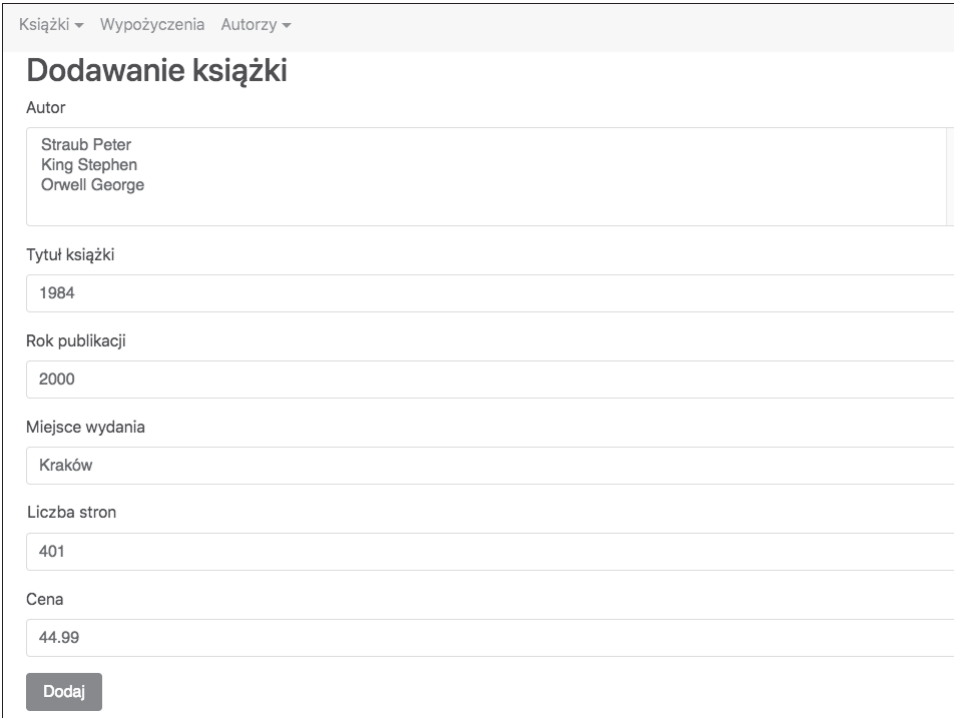
Jak widać, metoda `create` zyskała nowy kod, w którym sprawdzamy, czy dane otrzymane z formularza (czyli żądania HTTP) zawierają informacje o autorach w postaci tablicy identyfikatorów. W tym celu użyta została instrukcja `isset` interpretera. W przypadku istnienia takiej tablicy następuje odwołanie obiektu książki do relacji z autorami, na której wykonywana jest metoda `sync`. Przypomnijmy, że metoda ta przyjmuje tablicę identyfikatorów, które mają zostać dopisane do wybranego obiektu. W naszym przypadku oznacza to, że do wybranej książki zostaną przypisani autorzy o podanych identyfikatorach.

W tym miejscu warto również przypomnieć, że gdyby była potrzeba dodania do relacji tylko jednego rekordu, to możliwe byłoby użycie metody `attach`.

Niemniej jednak powyższe modyfikacje powinny być już gotowe do przetestowania. W tym celu należy otworzyć formularz nowej książki:

<http://biblioteka.local/books/create>

a także uzupełnić go przykładowymi danymi (warto skorzystać z opcji kilku autorów). Na rysunku 5.7 przedstawiono formularz z wypełnionymi polami.



Książki ▾ Wypożyczenia Autorzy ▾

Dodawanie książki

Autor

Straub Peter
King Stephen
Orwell George

Tytuł książki

1984

Rok publikacji

2000

Miejsce wydania

Kraków

Liczba stron

401

Cena

44.99

Dodaj

RYСУNEK 5.7. Formularz dodawania nowej książki z możliwością wyboru kilku autorów

Potwierdzenie poprawnego zapisania nowej pozycji można uzyskać poprzez przejście na listę książek i podgląd nowej. W przypadku wyboru dwóch lub więcej autorów wszyscy oni powinni pojawić się na samym dole panelu (rysunek 5.8).

Książki ▾ Wypożyczenia Autorzy ▾	
Nazwa książki	1984
Rok wydania	2000
Miejsce wydania	Kraków
Liczba stron	401
Cena	44.99
Autorzy	<ul style="list-style-type: none"> • King Stephen • Orwell George

RYСУNEK 5.8. Panel podglądu książki z widoczną listą autorów

Pozostaje już tylko dopisanie w panelu nawigacji opcji dodania nowego wypożyczenia (plik `/resources/views/template.blade.php`):

```
<li class="nav-item dropdown">
  <a class="nav-link dropdown-toggle" data-toggle="dropdown" href="{{ URL::to('loans') }}">Wypożyczenia</a>
  <div class="dropdown-menu">
    <a class="dropdown-item" href="{{ URL::to('loans') }}">Wszystkie</a>
    <a class="dropdown-item" href="{{ URL::to('loans/create') }}">Dodaj nową</a>
  </div>
</li>
```

W tym momencie warto również zapisać zmiany w repozytorium GIT:

```
git add .
git commit -m "Formularze dodawania nowych rekordow"
git push origin master
```

Po poprawnym zapisaniu danych na zdalnym serwerze można przejść do kolejnego rozdziału, czyli próby stworzenia formularzy edycji danych.

5.3. Formularze edycji danych

Oprócz standardowego dodawania nowych rekordów nierzadko istnieje potrzeba aktualizacji już istniejących danych. W niniejszym rozdziale spróbujemy stworzyć nowy formularz, który będzie służył do modyfikacji zapisanych wcześniej rekordów.

W rozdziale opisującym warstwę modelu stworzyliśmy już kod, który odpowiada za realizację samego zapisu — wtedy też na twardo zapisaliśmy dane zaktualizowanego obiektu. Teraz przyszła pora na implementację interfejsu, który zapewni nam dużo większą swobodę, gdyż nowe wartości pól będą pobierane od użytkownika.

Prace rozpoczynają się od modyfikacji kontrolera `BookController` i metody `edit` — w przypadku tworzenia formularza edytującego musimy pamiętać, że w odróżnieniu od tworzenia nowego rekordu, musi być on wypełniony danymi pobranymi z bazy. W tym celu na podstawie identyfikatora uzyskanego ze sparametryzowanego odnośnika listy książek wywołujemy metodę `find` i odnajdujemy rekord, który ma zostać zmodyfikowany. Ponieważ formularz będzie również zawierał listę autorów, ich też pobieramy z bazy i oba stworzone obiekty przekazujemy do widoku edycji. Cała metoda `edit` widoczna jest na poniższym listingu:

```
public function edit(BookRepository $bookRepo, $id)
{
    $book = $bookRepo->find($id);
    $authors = Author::all();
    return view('books/edit', ['book' => $book,
                              'authors' => $authors]);
}
```

Przechodząc do pliku widoku edycji (`/resources/views/template.blade.php`), mamy więc do dyspozycji dwa obiekty: książkę, która ma zostać wyedytowana, a także listę wszystkich dostępnych w systemie autorów:

```
@extends('template')
@section('title')
    Edycja książki
@endsection
@section('content')
<div class="container">

<h2>Edycja książki</h2>
<form action="{{ action('BookController@update', [$book->id]) }}" method="POST" role="form">
<input type="hidden" name="_token" value="{{ csrf_token() }}" />

<div class="form-group">
    <label for="name">Autor</label>
    <select type="text" class="form-control" name="author_id[]" multiple>
        @foreach ($authors as $author)
            @if(in_array($author->id,$book->authors->pluck('id')->toArray()))
                <option value="{{ $author->id }}" selected="{{ $author->lastname }}" {{
$author->firstname }}</option>
            @else
                <option value="{{ $author->id }}">{{ $author->lastname }} {{ $author-
>firstname }}</option>
            @endif
        @endforeach
    </select>
</div>
    <input type="hidden" name="book_id" value="{{ $book->id }}" />
<div class="form-group">
    <label for="name">Tytuł książki</label>
    <input type="text" class="form-control" name="name" value="{{ $book->name }}" />
</div>
<div class="form-group">
    <label for="name">Rok publikacji</label>
    <input type="text" class="form-control" name="year" value="{{ $book->year }}" />
</div>
```

```

</div>
<div class="form-group">
  <label for="name">Miejsce wydania</label>
  <input type="text" class="form-control" name="publication_place" value="
    {{ $book->publication_place }}" />
</div>
<div class="form-group">
  <label for="name">Liczba stron</label>
  <input type="text" class="form-control" name="pages" value="{{ $book->pages }}" />
</div>
<div class="form-group">
  <label for="name">Cena</label>
  <input type="text" class="form-control" name="price" value="{{ $book->price }}" />
</div>
<input type="submit" value="Aktualizuj" class="btn btn-primary" />
</form>
</div>
@endsection('content')

```

Widok edycji formularza jest bardzo zbliżony do wcześniejszej wersji służącej do tworzenia rekordów. Wprowadzone zmiany to:

- Nagłówek informacyjny.
- Zmodyfikowany atrybut `action` znacznika formularza — podana metoda i kontroler obsługujący, a także w parametrze żądania przekazany identyfikator książki (tak by kod obsługujący wiedział, którą z książek należy zaktualizować).
- Dodanie instrukcji warunkowej w znaczniku `select`, który tworzy listę rozwijaną wszystkich autorów dostępnych w systemie. Instrukcja warunkowa sprawdza, czy identyfikator kolejnego pobranego z listy autora znajduje się na liście autorów przypisanych do relacji z książką. Jeśli identyfikatory są zgodne, oznacza to, że autor z danej iteracji pętli powinien zostać zaznaczony (jako znajdujący się w relacji z książką). Zostanie to zrealizowane poprzez dodanie atrybutu `selected` w znaczniku `option`. W przeciwnym wypadku system ma oczywiście stworzyć zwykłą pozycję listy rozwijanej — niezaznaczoną.
- Kolejne pola formularza posiadają zdefiniowany i przypisany atrybut `value`. Wartości pochodzą z pól obiektu książki, która została przekazana z kontrolera do widoku. Zabieg ten oznacza, że wyświetlone na stronie formatki będą posiadały predefiniowane dane.
- Zmiana etykiety przycisku zatwierdzającego przesłanie formularza do obsługi.

Kolejny etap prac to delikatna modyfikacja tablicy routingu. Co prawda na skutek użycia `resources` mamy domyślnie utworzoną trasę do metody `update` z żądaniem typu `PATCH`, ale zdefiniowany przez nas formularz przesyła dane za pomocą żądania typu `POST`, więc nad istniejącą już trasę usuwania książki dopisujemy nowy routing do metody `update`:

```

Route::post('/books/{id}/update', 'BookController@update');
Route::get('/books/{id}/delete', 'BookController@destroy');

```

Tak stworzoną trasę warto teraz oprogramować — w tym celu edytujemy kontroler `BookController` i metodę `update`:

```
public function update(Request $request, BookRepository $bookRepo, $id)
{
    $data = $request->all();
    $booksList = $bookRepo->update($data, $id);

    return redirect('books');
}
```

Ciało opisywanej metody składa się z pobrania wszystkich pól formularza do tablicy `$data`, a następnie przekazania jej do metody `update` repozytorium `BookRepository`, którą spróbujemy zaimplementować:

```
public function update(array $data, $id) {
    $book = Book::find($id);
    $book->fill($data);
    $book->save();

    if(isset($data['author_id'])) {
        $book->authors()->sync($data['author_id']);
    }
    return $book;
}
```

Modyfikacje to przede wszystkim odnalezienie odpowiedniej książki (z użyciem identyfikatora uzyskanego z żądania), a także wypełnienie jej pól danymi otrzymanymi z kontrolera (metoda `fill`). Tak zaktualizowany obiekt jest następnie zapisywany, a także wykonywane są czynności związane z przypisywaniem relacji — użycie metod `authors` i `sync` w celu zdefiniowania danych relacyjnych. Cała grupa powyższych czynności jest bliźniaczo podobna do kodu realizującego dodawanie nowego rekordu — należy jedynie pamiętać, że operujemy na już istniejących danych, a nie całkowicie nowych.

W celu przetestowania nowej funkcjonalności można teraz wybrać opcję edycji wybranej książki, która jest dostępna na stronie listy. Formularz edycyjny powinien zawierać wszystkie dane zgodne z podanymi podczas tworzenia obiektu (przykład na rysunku 5.9).

Wybranie przycisku aktualizacji powinno uruchomić odpowiednią logikę, a po przejściu do podglądu edytowanego rekordu powinny wyświetlić się nowe, nadpisane wartości (rysunek 5.10).

Na zakończenie klasycznie zapisujemy zmiany w repozytorium GIT:

```
git add .
git commit -m "Formularz edycji książki"
git push origin master
```

Podsumowując rozważania na temat edycji, można dojść do wniosku, że najważniejszym jej elementem jest pamięć o tym, by razem z edytowanymi danymi przesyłać również identyfikator obiektu, który podlega edycji. Jest to kluczowe w kwestii zrozumienia idei tej funkcjonalności. Cała reszta, łącznie z odnoszeniem do relacji i samym zapisem, jest bliźniaczo podobna do wcześniej tworzonych formularzy dodających nowe rekordy tabel.

Książki ▾ Wypożyczenia ▾ Autorzy ▾

Edycja książki

Autor

Straub Peter
King Stephen
Orwell George

Tytuł książki

1984

Rok publikacji

2000

Miejsce wydania

Kraków

Liczba stron

501

Cena

44.99

Aktualizuj

RYSUNEK 5.9. Formularz edycji książki

Książki ▾ Wypożyczenia ▾ Autorzy ▾

Nazwa książki	1984
Rok wydania	2000
Miejsce wydania	Kraków
Liczba stron	501
Cena	44.99
Autorzy	<ul style="list-style-type: none"> • King Stephen • Orwell George • Straub Peter

RYSUNEK 5.10. Podgląd książki z widocznym nowym autorem

5.4. Walidacja formularzy

W wszystkich testach związanych z dodawaniem i edycją danych zakładaliśmy, że użytkownik świadomie lub nie poprawnie wypełnił wszystkie pola — żadnego nie zostawił pustego, a format danych zgadzał się zakładanym przez nas typem. W rzeczywistych aplikacjach działających produkcyjnie programista nie ma takiego komfortu — wręcz przeciwnie, żelazną zasadą tworzenia kodu obsługujące formularze jest obsługa wszystkich sytuacji wyjątkowych: braku danych, problemów z formatem czy zależności pomiędzy polami (jeśli jedno z pól jest uzupełnione, to inne jest obowiązkowe).

Laravel oczywiście posiada zaimplementowane techniki realizujące funkcjonalność walidacji, które od razu wykorzystamy. W tym celu w pierwszym etapie prac należy wybrać miejsce, w którym umieścimy samą logikę testującą dane. W naszym systemie biblioteki dane otrzymywane i obsługiwane są w kontrolerach, a konkretnie w przypadku książek w metodach `store` i `update`. Spróbujmy wyedytować pierwszą z nich:

```
public function store(Request $request, BookRepository $bookRepo)
{
    $validatedData = $request->validate([
        'name' => 'required|max:255',
        'year' => 'required|integer',
        'publication_place' => 'required|string',
        'pages' => 'required|integer',
        'price' => 'required|numeric',
    ]);
    $data = $request->all();
    $book = $bookRepo->create($data);

    return redirect('books');
}
```

Dokonane zmiany to wywołanie metody `validate` na obiekcie żądania (`$request`), które dotarło do serwera. Opisywana metoda jako argument przyjmuje tablicę. Jej kluczami są poszczególne elementy formularza, które mają być testowane pod kątem poprawności. Wartościami są natomiast etykiety, nazwy reguł, które konkretne pola muszą spełnić. Jedno pole może posiadać kilka różnych reguł, odseparowanych od siebie za pomocą symbolu „|”. Co więcej, reguły mogą mieć parametry, tak jak reguła `max`, która za parametr przyjmuje liczbę 255, co oznacza, że długość pola nie może być większa niż ta wartość.

Wśród reguł zaimplementowanych w kodzie powyżej są:

- pole `name` — wymagane, maksymalnie 255 znaków,
- pole `year` — wymagane, liczba całkowita,
- pole `publication_place` — wymagane, ciąg znaków,
- pole `pages` — wymagane, liczba całkowita,
- pole `price` — wymagane, wartość liczbowa.

Przedstawione reguły to oczywiście przykłady pasujące do sytuacji. Tabela 5.1 zawiera najpopularniejsze z zasad walidacyjnych.

TABELA 5.1. *Lista możliwych do użycia reguł*

Nazwa reguły	Testowane pole musi
accepted	Posiadać jedną z wartości: yes, on, 1 lub true.
alpha	Być ciągiem cyfr i liter.
after:date	Być datą po dacie date.
different:field	Posiadać inną wartość niż pole field.
email	Być poprawnym adresem poczty e-mail.
exists:table, column	Posiadać wartość, która znajduje się wśród danych kolumny column tabeli table, np. 'miasto' => 'exists:miasta,nazwa' oznacza, że pole formularza miasto musi posiadać wartość, która znajduje się w tabeli miasta w kolumnie nazwa.
file	Posiadać odwołanie do poprawnie wgranego pliku.
gt:field	Posiadać wartość większą od wartości pola field.
gte:field	Posiadać wartość równą wartości pola field lub większą.
image	Posiadać odwołanie do poprawnie wgranego pliku graficznego (jpg, png, bmp, gif lub svg).
in:value1, value2,...	Posiadać wartość należącą do listy (value1, value2 itd.).
integer	Być typu całkowitego.
ip	Posiadać wartość będącą poprawnym adresem IP.
json	Posiadać wartość będącą poprawnym tekstem reprezentującym dane typu JSON.
lt:field	Posiadać wartość mniejszą od wartości pola field.
lte:field	Posiadać wartość równą wartości pola field lub mniejszą.
max:field	Posiadać wartość równą polu field lub mniejszą.
min:field	Posiadać wartość równą polu field lub większą.
numeric	Być wartością numeryczną.
required	Być wypełnione, nie może być puste (włączając pusty ciąg znaków).
required_if:field,value	Być wypełnione, jeśli pole field jest równe wartości value.
required_with:field1, field2	Być wypełnione, jeśli któryś z pól field1, field2... jest wypełnione.

TABELA 5.1. Lista możliwych do użycia reguł (ciąg dalszy)

Nazwa reguły	Testowane pole musi
required_with:field1, field2	Być wypełnione, jeśli wszystkie z pól field1, field2... są wypełnione.
string	Być ciągiem znaków.
unique:table,column,except,idColumn	Posiadać wartość unikalną pośród danych z tabeli table i kolumny column; za pomocą except możliwe jest wyłączenie konkretnych danych z tego warunku.
url	Posiadać wartość będącą poprawnym adresem URL.

Tak stworzony kod, posiadający wybrane reguły walidacyjne, możemy teraz przetestować poprzez celowe pominięcie kilku pól i próbę dodania nowej książki. W teorii książka nie powinna zostać dodana — tak też się dzieje, jednak w obecnej chwili po wysłaniu formularza do obsługi jesteśmy przekierowywani ponownie do jego widoku. Zdecydowanie brakuje komentarza czy jakiegokolwiek informacji, gdzie został popełniony błąd i co użytkownik musi zrobić, by system zaakceptował jego żądanie.

Laravel zapisuje ewentualne błędy walidacyjne w obiekcie sesji przeglądarki. W rezultacie samo przekierowanie, czyli powrót na stronę formularza, nie jest wystarczające — musimy dodać kod, który wyświetli odpowiednie komunikaty. Najbardziej trywialnym miejscem umieszczenia tego typu informacji może być widok formularza — my jednak zastosujemy zdecydowanie bardziej globalne podejście i umieścimy ją bezpośrednio w szablonie strony (plik `/resources/views/template.blade.php`, pod znacznikiem zamknięcia nawigacji `nav` i przed poleceniem `@yield('content')`):

```
</nav>

@if ($errors->any())
    <div class="alert alert-danger">
        <ul>
            @foreach ($errors->all() as $error)
                <li>{{ $error }}</li>
            @endforeach
        </ul>
    </div>
@endif

@yield('content')
```

Tak dodany kod powinien wygenerować odpowiednie komunikaty — oczywiście tylko wtedy, gdy faktycznie poszczególne reguły nie będą spełnione. Przykładowy komunikat walidacji widoczny jest na rysunku 5.11.

Książki ▾
Wypożyczenia ▾
Autorzy ▾

- The name field is required.
- The year field is required.
- The publication place field is required.
- The pages field is required.
- The price field is required.

Dodawanie książki

Autor

Straub Peter
 King Stephen
 Orwell George

Tytuł książki

Rok publikacji

Miejsce wydania

Liczba stron

Cena

RYSUNEK 5.11. Przykład błędu walidacji w formularzu dodawania nowej książki

Dobrym podsumowaniem funkcjonalności walidacji może być pewne przemyślenie: mimo że technicznie jest to nietrudna i szybka funkcjonalność, to spełnia niebagatelne funkcje — powoduje, że logika biznesowa aplikacji, w tym wszelkie odwołania do bazy danych, posiada poprawne i zwalidowane dane. Jest to niezwykle ważne i kluczowe w każdym nowoczesnym systemie aplikacyjnym.

Form Request

Mimo że walidacja umieszczona w kontrolerze w stu procentach spełni swoje funkcje, to Laravel pokusił się o dostarczenie bardziej rozbudowanej techniki kontroli danych, której idea polega na odseparowaniu warstwy walidacji do osobnej klasy i utworzeniu tzw. Form Request. Nierzadko każda z metod kontrolera będzie posiadała osobny zestaw reguł, a co za tym idzie, osobną klasę żądania służącą do walidacji.

I tak kontroler książek w naszym systemie mógłby posiadać osobne żądania dla metody `store` i `updated` — obie one służą do zapisywania danych pobranych z formularza, a co za tym idzie, wymagana jest w nich walidacja poprawności otrzymanych danych. Z pewnością rozważyć można potrzebę tworzenia dwóch oddzielnych klas dla obu tych metod. Być może reguły poprawności danych jednej i drugiej funkcji będą takie same — wtedy można użyć jednej klasy.

Podstawowy obiekt `Form Request` tworzony jest za pomocą linii poleceń `Artisan`, co spróbujemy przetestować, tworząc obiekt żądania dla kontrolera książek:

```
php artisan make:request StoreBook
```

Efektom wywołania polecenia będzie utworzenie pliku `StoreBook.php` w katalogu `app/Http/Requests/`, który będzie zawierał klasę `StoreBook` dziedziczącą po klasie `FormRequest`. W środku odnaleźć można dwie metody, `authorize` i `rules`. Pierwsza z nich to sprawdzenie, czy dane żądanie w ogóle może się wykonać, czyli czy wywołujący je użytkownik ma możliwość jego wykonania.

Przykładem może być sprawdzenie, czy użytkownik edytuje książkę, do której jest przypisany. Jeśli użytkownik nie ma prawa do wykonania żądania, to funkcja zwróci wartość `false`; w praktyce każda inna odpowiedź da frameworkowi sygnał, że żądanie może zostać dokończony. Warto nadmienić, że w tym przypadku rzeczywistość pokrywa się z teorią — metoda `authorize` to świetne miejsce do dodania specjalnego warunku, który musi zostać spełniony dla konkretnego użytkownika. Zazwyczaj dotyczy jego praw dostępu (edycji nie swoich danych relacyjnych — postów, książek, list itp.).

Druga z metod to `rules`, która — jak sama nazwa wskazuje — przechowuje reguły dotyczące walidacji danego żądania. Możemy w niej używać wszystkich reguł przedstawionych wcześniej. Od razu spróbujmy przebudować nasz ostatni kod walidacji danych z metody `store` kontrolera `BookController` tak, by korzystać z `Form Request`. W tym celu edytujemy plik `StoreBook.php`, którego zawartość będzie prezentowała się następująco:

```
<?php
namespace App\Http\Requests;
use Illuminate\Foundation\Http\FormRequest;

class StoreBook extends FormRequest
{
    /**
     * Determine if the user is authorized to make this request.
     *
     * @return bool
     */
    public function authorize()
    {
        return true;
    }

    /**
     * Get the validation rules that apply to the request.
     *
     * @return array
     */
}
```

```

public function rules()
{
    return [
        'name' => 'required|max:255',
        'year' => 'required|integer',
        'publication_place' => 'required|string',
        'pages' => 'required|integer',
        'price' => 'required|numeric',
    ];
}
}

```

Jakie wnioski można wysnuć po analizie kodu?

- Metoda `authorize` zwróci w każdym przypadku logiczną prawdę. Oznacza to, że nie będzie działało żadne ograniczenie dotyczące użytkowników — wszyscy będą mogli wywołać wybrane żądanie.
- Metoda `rules` zdefiniuje szereg reguł i zasad dotyczących pól — jest to dokładna kopia zasad przeniesiona z kontrolera książek.
- Zdefiniowane reguły zwracane są w formie tablicy jako wynik działania metody `rules`.

W celu użycia tak stworzonego żądania musimy zmodyfikować kontroler książek — a dokładniej wskazać miejsce, w którym chcemy go użyć. W tym celu dokonujemy kosmetycznych zmian w metodzie `store` (usunięcie walidacji dodanej w poprzednim rozdziale, dodanie użycia przestrzeni nazw żądań HTTP, usunięcie `Illuminate\Http\Request`, a przede wszystkim podmiana żądania typu `Request` na bardziej szczegółowe, dodane przez nas `StoreBook`), której kod po zmianach będzie zgodny z poniższym:

```

(...)
use Illuminate\Http\Request;
use App\Http\Requests\StoreBook;
(...)
public function store(StoreBook $request, BookRepository $bookRepo)
{
    $data = $request->all();
    $book = $bookRepo->create($data);

    return redirect('books');
}

```

Tak dokonane zmiany nie powinny w tej chwili wpłynąć w jakikolwiek sposób na samo działanie aplikacji. Walidacja formularza dodawania nowej książki powinna działać tak jak do tej pory; zmiany dotyczą tylko samej struktury kodu.

Odseparowanie walidacji od kontrolera, mimo odrobinę większego wysiłku ze strony programisty, niesie też ze sobą dużo korzyści:

- możliwość wielokrotnego użycia tych samych reguł i zasad,
- umiejscowienie tych samych reguł w jednym miejscu, przez co ich podmiana czy aktualizacja są łatwiejsze,

- możliwość definiowania dodatkowych reguł dotyczących sprawdzania uprawnień konkretnego użytkownika (metoda `authorize`),
- większa elastyczność, odseparowanie kontrolera od logiki walidacji.

Powyższe zmiany dodajemy do repozytorium:

```
git add .
git commit -m "Form Request"
git push origin master
```

Dobrym ćwiczeniem podsumowującym pracę z Form Requests jest dodanie walidacji do formularzy tabeli wypożyczeń i autorów, do czego oczywiście gorąco zachęcam.

5.5. Internacjonalizacja

Ostatnim tematem poruszonym w rozdziale poświęconym formularzom będzie internacjonalizacja, czyli umiędzynarodowienie projektu. Nie jest to być może temat całościowo i wyłącznie powiązany z formularzami, ale stanowi ogromnie ważną część każdej współczesnej aplikacji internetowej.

Do tematu obsługi wielu języków Laravel podszedł w klasyczny sposób — każdy z języków posiada swój osobny plik, w którym przechowywane są tłumaczenia poszczególnych sentencji, umieszczone w jednej dużej tablicy. Pliki z tłumaczeniami przechowywane są w katalogu zgodnym z formatem: `/resources/lang/{nazwa_jezyka}/{nazwa_pliku}.php`

Przykładowy plik z tłumaczeniem może wyglądać następująco (plik `translations.php`):

```
<?php
return [
    'etykieta' => 'Tłumaczenie',
    'powitanie' => 'Witaj, :name',
];
```

Pierwszy element tablicy to standardowe tłumaczenie, gdzie pod etykietą o nazwie `etykieta` ukryta jest wartość w postaci napisu `Tłumaczenie`. Drugi wpis to etykieta `powitanie` wraz ze sparametryzowaną wartością.

Warto zauważyć, że domyślnie wraz z zainstalowanym frameworkiem programista otrzyma cztery pliki tłumaczeń:

- `auth.php` — tłumaczenia związane z autentykacją użytkownika,
- `pagination.php` — tłumaczenia związane ze stronicowaniem,
- `passwords.php` — tłumaczenia związane z hasłami,
- `validation.php` — tłumaczenia związane z walidacją.

Oczywiście, w aplikacji jednorazowo ustawiony może być tylko jeden aktualny język. Laravel dokonuje tego wyboru na podstawie ustawień systemowych i samych zmian w kodzie. Domyślny język aplikacji ustawiany jest w 83. linii pliku konfiguracyjnego `/app/config/app.php`:

```
'locale' => 'en',
```

Dynamiczna podmiana języka na inny odbywa się za pomocą metody `setLocale()` fasady App:

```
App::setLocale("pl"); //ustawi język na pl
```

Powyższe polecenie ustawi język systemu na pl i takiego też pliku będzie wyszukiwać w katalogu tłumaczeń. Ciekawostką jest fakt, że w przypadku niezalezienia pliku z tłumaczeniami dla wybranego języka wybierany jest język awaryjny. Definiowany jest on w tym samym pliku konfiguracyjnym, co język domyślny, w linii 96.:

```
'fallback_locale' => 'en',
```

Nierzadko programiście przyda się również opcja podglądu, który z języków jest aktualnie używany. W tym celu używana jest funkcja `getLocale()`, również z fasady App:

```
$lang = App::getLocale(); //zwróci aktualny język aplikacji
```

W tym miejscu warto zaznaczyć wcale nieoczywisty fakt, że instrukcje ustawiające język działają tylko i wyłącznie w obrębie jednego żądania. Oznacza to, że w celu ustawienia tego samego języka dla całej aplikacji należy użyć dodatkowego kodu, który zapisze aktualnie wybrany język w sesji lub ciasteczkach.

Pozostaje jeszcze jedno pytanie: jak wywołać same tłumaczenia? Jeśli będziemy kontynuować pracę z przykładowym tłumaczeniem przedstawionym kilka linii wyżej, to możemy je wywołać następująco:

```
echo __('translations.etykieta'); //wyświetli napis Tłumaczenie
echo __('translations.powitanie', ['name' => "Adam"]); //wyświetli napis Witaj, Adam
```

W przypadku widoków Blade, użyjemy oczywiście:

```
{{ __('translations.etykieta') }} //wyświetli napis Tłumaczenie
{{ __('translations.powitanie', ['name' => "Adam"]) }} //wyświetli napis Witaj, Adam
```

W celu przeciwiczenia nabytej wiedzy spróbujmy teraz zaimplementować internacjonalizację w aplikacji biblioteki, a także przetłumaczyć formularz dodawania nowej książki.

Pierwsza czynność to stworzenie nowej trasy, która będzie odpowiadała za faktyczną zmianę języka. W tym celu w pliku `web.php` dodajemy na samym końcu nowy wpis:

```
Route::get('language/{locale}', function ($locale) {
    session(['locale' => $locale]);
    return redirect()->action('BookController@index');
});
```

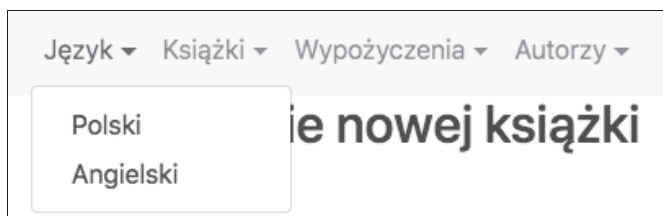
Wpis ten nie jest przypisany do żadnego z kontrolerów, ale od razu wykonuje pewną logikę — przede wszystkim w sesji, pod kluczem `locale`, zapisuje wartość języka pobraną z parametru żądania. I tak żądanie:

- `http://biblioteka.local/language/pl` — ustawi język polski,
- `http://biblioteka.local/language/pl` — ustawi język angielski.

Posiadając zdefiniowane trasy, możemy je teraz dodać do pliku szablonu strony (*resources/views/template.blade.php*) jako oddzielne linki do zmiany języka:

```
(...)
<ul class="nav navbar-nav navbar-right">
  <li class="nav-item dropdown">
    <a class="nav-link dropdown-toggle" data-toggle="dropdown" href="#">Język</a>
    <div class="dropdown-menu">
      <a class="dropdown-item" href="{{ URL::to('language/pl') }}">Polski</a>
      <a class="dropdown-item" href="{{ URL::to('language/en') }}">Angielski</a>
    </div>
  </li>
</ul>
```

Zmiany powinny być już widoczne jako część panelu nawigacji (rysunek 5.12).



RYСУNEK 5.12. Menu zmiany języka

Kolejne kroki to oczywiście stworzenie plików języka. W tym celu tworzymy podwójną strukturę plików i katalogów:

- katalog `/resources/lang/en` z plikami: *auth.php*, *forms.php*, *pagination.php*, *passwords.php* i *validation.php*,
- katalog `/resources/lang/pl` z plikami: *auth.php*, *forms.php*, *pagination.php*, *passwords.php* i *validation.php*.

Jak łatwo zauważyć, nasze nowe tłumaczenia będą znajdowały się w pliku *forms.php*. Warto jednak przekopiować zawartości pozostałych plików do katalogu z polskimi tłumaczeniami i je przetłumaczyć.

Wracając do samego pliku *forms.php*, możemy teraz uzupełnić go zgodnie z tym, czego faktycznie potrzebujemy w widoku formularza nowej książki. Poniżej znajduje się listing pliku z polskim tłumaczeniem (`/resources/lang/pl/forms.php`):

```
<?php
return [
    'author' => 'Autor',
    'book_title' => 'Tytuł książki',
    'publication_year' => 'Rok wydania',
```



```

    'publication_place' => 'Miejsce wydania',
    'pages' => 'Liczba stron',
    'price' => 'Cena',
    'add' => 'Dodaj',
    'new_book' => 'Dodawanie nowej książki'
];

```

Druga, angielska wersja, trafia oczywiście do *resources/lang/en/forms.php*:

```

<?php

return [
    'author' => 'Author',
    'book_title' => 'Book Title',
    'publication_year' => 'Publication Year',
    'publication_place' => 'Publication Place',
    'pages' => 'Pages',
    'price' => 'Price',
    'add' => 'Add',
    'new_book' => 'Adding a new book'
];

```

Kolejne zmiany to odczytanie zapisanego w sesji języka bezpośrednio w metodzie kontrolera — w naszym wypadku w metodzie *create*, gdyż to ona tworzy formularz dodawania nowej książki. Odczytany z sesji język jest następnie ustawiany jako aktywny przy użyciu metody *setLocale*. Warto przypomnieć, że czynność tę należy wykonać w każdej metodzie, która ma obsługiwać wiele języków. W przyszłości poznamy bardziej ogólne miejsce do jednokrotnego odczytania kodu języka, na razie jednak musi nam wystarczyć takie podejście:

```

public function create(BookRepository $bookRepo)
{
    \App::setLocale(session('locale'));
    $authors = Author::all();
    return view('books/create', ['authors' => $authors]);
}

```

Ostatnie zmiany to użycie zadeklarowanych etykiet bezpośrednio w formularzu nowej książki (plik */resources/views/books/create.blade.php*):

```

@extends('template')

@section('title')
    Lista książek
@endsection

@section('content')
<div class="container">

    <h2>{{ __('forms.new_book') }}</h2>
    <form action="{{ action('BookController@store') }}" method="POST" role="form">
    <input type="hidden" name="_token" value="{{ csrf_token() }}" />

        <div class="form-group">
            <label for="name">{{ __('forms.author') }}</label>

```

```

                                <select type="text" class="form-control"
name="author_id[]" multiple>
                                @foreach ($authors as $author)
                                    <option value="{{ $author->id }}">{{
$author->lastname }} {{ $author->firstname }}</option>
                                @endforeach
                                </select>
                            </div>

<div class="form-group">
    <label for="name">{{ __( 'forms.book_title' ) }}</label>
    <input type="text" class="form-control" name="name" />
</div>

<div class="form-group">
    <label for="name">{{ __( 'forms.publication_year' ) }}</label>
    <input type="text" class="form-control" name="year" />
</div>

<div class="form-group">
    <label for="name">{{ __( 'forms.publication_place' ) }}</label>
    <input type="text" class="form-control" name="publication_place" />
</div>

<div class="form-group">
    <label for="name">{{ __( 'forms.pages' ) }}</label>
    <input type="text" class="form-control" name="pages" />
</div>

<div class="form-group">
    <label for="name">{{ __( 'forms.price' ) }}</label>
    <input type="text" class="form-control" name="price" />
</div>

    <input type="submit" value="{{ __( 'forms.add' ) }}" class="btn btn-primary"/>
</form>
</div>
@endsection('content')

```

Tak dokonane zmiany powinny zaowocować możliwością przełączenia języka angielskiego na polski — oczywiście na razie tylko w zakresie jednego formularza. W celu modyfikacji reszty kodu należy stworzyć odpowiednie etykiety i przypisać im wartości, dla każdego języka z osobna. Nie można również zapomnieć o ręcznym wywołaniu języka w każdej metodzie kontrolera, lecz ten problem zostanie rozwiązany w jednym z nadchodzących rozdziałów.

Tymczasem zapisujemy zmiany w repozytorium:

```

git add .
git commit -m "Internacjonalizacja"
git push origin master

```

Wysłanie kodu do zewnętrznego repozytorium zakończy nie tylko pracę z umiędzynarodowieniem aplikacji, ale także pracę z formularzami. Co prawda w następnych rozdziałach z pewnością będziemy się z nimi spotykać i z nich korzystać, lecz będą dla nas już tylko narzędziem do

wykonania określonych zadań. Formularze i ich obsługa to jeden z najważniejszych elementów współczesnych aplikacji. Wykorzystujemy je na każdym kroku i nie wygląda na to, by wkrótce miało się to zmienić.

Warto jednak pamiętać o kilku podstawowych zasadach ich tworzenia. Istotne jest pamiętanie o tym, że dane powinny być walidowane i sprawdzane. Ciekawym zagadnieniem jest także badanie UX, czyli tego, jak budować formularze, by były przyjazne, nie męczyły użytkownika, a przy tym wykonywały swoje zadania. Ten temat wykracza już jednak poza ramy niniejszej książki.

Skorowidz

A

aktualizacja rekordów, 134
Apache, 23
API, Application Programming Interface, 255
 Resources, 256
 testowanie, 259
aplikacja
 Composer, 17
 XAMPP, 18
architektura MVC, 77
Atom, 50
atrybut action, 193
autentykacja, 243
 funkcjonalności, 244

B

baza danych, 17, 18, 109
 aktualizacja rekordów, 134
 dodawanie nowych rekordów, 132
 konfiguracja, 111
 pobieranie pojedynczego rekordu, 128
 przeszukiwanie tabel, 137
 tworzenie struktury, 119
 usuwanie rekordów, 136
biblioteki Packagist, 45
Bitnami, 26
Blade, 98
 gniazda, 100
 instrukcje, 100
 komponenty, 100
 szablony, 98

C

commands, 225
Composer, 17, 38
 instalacja w systemie OSX i Ubuntu, 42
 instalacja w systemie Windows, 39
 zasada działania, 44
cykl życia żądania, 91

D

dependency injection, 90
dodawanie nowych rekordów, 132

E

eager loading, 162
Eclipse, 48
edycja danych, 191
Eloquent ORM, 121, 162
e-mail, 232
 konfiguracja Laravela, 237
 wysyłka wiadomości, 238
events, 221

F

fasada, 95
Form Request, 199
formularz, 179
 dodawanie danych, 179, 185
 edycja danych, 191
 logowania, 248

 rejestracji, 248
 walidacja, 196
funkcje tablicowe, 229
funkcjonalności, 15
 autentykacji, 244

G

GIT, 52
 instalacja w OSX, 57
 instalacja w Ubuntu, 56
 instalacja w Windows, 53
gniazda, 100

I

IDE, Integrated Development Environment, 17, 46
instalacja
 Bitnami, 26
 Composer
 w systemie OSX i Ubuntu, 42
 w systemie Windows, 39
GIT
 w OSX, 57
 w Ubuntu, 56
 w Windows, 53
Laravela, 63
XAMPP
 dla Windows, 18
 w systemie OSX, 31
 w systemie Ubuntu, 25
instrukcje
 Blade, 100
 warunkowe, 193

internacjonalizacja, 202
 interpreter PHP, 17, 18

K

klasy pomocnicze, Helpers, 228
 kolekcje
 operacje, 139
 komponenty, 100
 konfiguracja
 bazy danych, 111
 Virtual Hosts, 67
 konsola Artisan, 76
 kontroler, 78, 87, 126

L

Laravel, 13
 Artisan, 76
 baza danych, 109
 cechy, 15
 dołączanie bibliotek, 103
 formularze, 179
 funkcjonalności, 15
 instalacja, 63
 klasy Helpers, 228
 konfiguracja aplikacji, 72
 konfiguracja bazy, 111
 kontrolery, 87
 Mix, 209
 obsługa poczty e-mail, 232
 obsługa zdarzeń, 221
 odwołanie do modelu, 126
 pierwszy program, 69
 polecenia, 225
 routing, 79
 szablony Blade, 98
 tworzenie modelu, 123
 usługi, 214
 lista rozwijana, 189

M

mapowanie
 obiektowo-relacyjne, 122
 metody
 kontrolera, 97
 Query Builder, 164
 middleware, 92

migracje, 113
 model, 78, 109
 modyfikacje kolumn, 116
 MVC, Model View Controller, 78
 MVP, Minimum Viable Product, 265
 MySQL, 23

N

nazwy tras, 83
 NetBeans, 46
 Node.js, 211
 NPM, Node Package Manager, 211

O

obsługa poczty e-mail, 232
 odwołanie do modelu, 126
 opcje uruchomienia NPM, 212
 operacje na kolekcjach, 139

P

pętla foreach, 189
 pliki
 .env, 72, 267
 migracji, 114
 pobieranie pojedynczego rekordu, 128
 poczta e-mail, 232
 polecenia, commands, 225
 dotyczące
 migracji, 116
 modyfikacji ciągów znaków, 230
 ścieżek systemowych, 230
 Laravel Artisan, 77
 pomocnicze, 231
 polecenie
 git add, 65
 git init, 65
 prefiksy, 84
 program
 Atom, 51
 Eclipse, 49

NetBeans, 47, 48
 Sublime Text, 52
 przeszukiwanie tabel, 137
 publikowanie aplikacji, 265

Q

Query Builder, 162
 metody, 164
 pobieranie danych, 169

R

reguły, 197
 rekordy
 aktualizacja, 134
 dodawanie, 132
 pobieranie, 128
 usuwanie, 136
 relacja, 141, 185
 jeden do jednego, 142
 jeden do wielu, 146
 typu has-many-through, 160
 wiele do wielu, 152
 relacyjne bazy danych, 109
 repozytorium, 58
 zdalne, 108
 resources, 86
 routing, 79
 nazwy tras, 83
 powiązanie modelu z parametrem, 85
 prefiksy, 84
 przekierowania, 84
 rodzaje żądań, 81
 zasoby, resources, 86
 żądania sparametryzowane, 82

S

Service Container, 87
 services, 214
 serwer
 aplikacyjny, 17, 18
 MySQL, 110
 serwis Gitlab, 59
 SQL, Structured Query Language, 109

struktura bazy danych, 119
Sublime Text, 50
SVN, 53
szablony Blade, 98

Ś

ścieżki
 bezwzględne, 213
 względne, 213
środowisko programistyczne,
 IDE, 17

T

tabele
 przeszukiwanie, 137
Table Seeders, 119
testowanie API, 259
tworzenie
 kontrolerów, 87
 migracji, 113
 modelu, 123
typy kolumn, 115

U

usługa Mailgun, 237
usługi, services, 214
usuwanie rekordów, 136

V

Virtual Hosts
 konfiguracja, 67

W

walidacja formularzy, 196
widok, 78
wydajność, 162
wysyłka wiadomości e-mail,
 238
wywoływanie migracji, 113
wzorzec Repository, 171

X

XAMPP, 18
 instalacja dla Windows, 18
 instalacja w systemie OSX,
 31
 instalacja w systemie
 Ubuntu, 25
 okno programu, 24

Z

zasoby, Resources, 86
zdarzenia, events, 221
znacznik
 <iframe>, 255
 multiple, 189
 select, 189

Ż

żądania
 cykl życia, 91
 sparametryzowane, 82

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

Odkryj możliwości Laravela

- Poznaj nowoczesny framework do tworzenia aplikacji WWW
- Naucz się wykorzystywać jego możliwości w praktyce
- Twórz kod, testuj go i sprawdzaj, czy działa zgodnie z założeniami

Laravel to wydajny, nowoczesny i dostępny za darmo framework do tworzenia aplikacji internetowych, napisany w języku PHP i bazujący na wzorcu architektonicznym Model-View-Controller. Zalety tego rozwiązania ceni coraz więcej programistów tworzących różne aplikacje webowe. Swoją popularność framework zawdzięcza stałemu rozwojowi, spójnemu i przystępnemu kodowi, świetnej dokumentacji oraz aktywnej społeczności, zapewniającej wsparcie również mniej doświadczonym programistom.

Laravel. Wstęp do programowania aplikacji internetowych to świetna książka dla osób, które pragną poznać framework od podstaw. Autor podręcznika postawił sobie za cel przekazanie wiedzy w sposób przyjazny i przystępny. Skoncentrował się na aspekcie praktycznego zastosowania prezentowanych informacji. Dzięki temu czytelnik krok po kroku zagłębia się w kolejne zagadnienia. Tworzy działającą i użyteczną aplikację webową, nie tracąc przy tym czasu na zbędną teorię, którą bez trudu można znaleźć w dokumentacji.

- Podstawowe informacje o Laravelu
- Instalacja i konfiguracja środowiska pracy
- Routing, kontrolery i szablony Blade
- Konfiguracja i używanie bazy danych
- Korzystanie z formularzy i walidacja danych
- Zaawansowane możliwości frameworka
- Uwierzytelnianie użytkowników i tworzenie API
- Publikowanie aplikacji w sieci

Twórz nowoczesne aplikacje przy użyciu doskonałego frameworka!

	<p><i>Sprawdź nasze szkolenia!</i></p>  <p>SZKOLENIA</p> <p>AKADEMIA IT & BUSINESS</p> <p>WWW.SZKOLENIA.HELION.PL</p>	<p>KOD KORZYŚCI <i>Sięgnij po więcej!</i> ►</p> 
 helion.pl		<p>ISBN 978-83-283-5130-1</p>  <p>9 788328 351301</p>
INFORMATYKA W NAJLEPSZYM WYDANIU		Cena: 49,00 zł