Katedra informatiky
Přírodovědecká fakulta
Univerzita Palackého v Olomouci

BAKALÁŘSKÁ PRÁCE

Framework Laravel

2015

Lukáš Fiala

Vedoucí práce: Mgr. Petr Krajča, Ph.D.

Studijní obor: Aplikovaná informatika, kombinovaná forma

Bibliografické údaje

Autor:	Lukáš Fiala
Název práce:	Framework Laravel
Typ práce:	bakalářská práce
Pracoviště:	Katedra informatiky, Přírodovědecká fakulta, Univerzita Palackého v Olomouci
Rok obhajoby:	2015
Studijní obor:	Aplikovaná informatika, kombinovaná forma
Vedoucí práce:	Mgr. Petr Krajča, Ph.D.
Počet stran:	60
Přílohy:	1 CD
Jazyk práce:	český

Bibliograhic info

Author:	Lukáš Fiala
Title:	Laravel Framework
Thesis type:	bachelor thesis
Department:	Department of Computer Science, Faculty of Science, Palacký University Olomouc
Year of defense:	2015
Study field:	Applied Computer Science, combined form
Supervisor:	Mgr. Petr Krajča, Ph.D.
Page count:	60
Supplements:	1 CD
Thesis language:	Czech

Anotace

Práce popisuje PHP framework Laravel. Nejdříve z obecného hlediska a hlavních komponent, následně se zaměřuje na bezpečnost a jednotkové testování webových aplikací. Pro srovnání používá PHP frameworky Nette a Symfony.

Synopsis

This work describes PHP framework Laravel. First, in general terms and main components, then focuses on security and unit testing of web applications. For comparison uses PHP frameworks Nette and Symfony.

Klíčová slova: php framework; laravel; nette; symfony

Keywords: php framework; laravel; nette; symfony

Děkuji Mgr. Petru Krajčovi, Ph.D. za cenné rady a připomínky. Dále děkuji všem, kteří měli se mnou trpělivost při tvorbě této práce.

datum odevzdání práce podpis autora

Obsah

1 Úvod **8**
- 1.1 Předpoklady . . . 8
- 1.2 Co je to framework? . . . 8
- 1.3 Důvod výběru tématu práce . . . 8
- 1.4 Struktura práce . . . 8
- 1.5 Zdroje informací . . . 9
- 1.6 Další webové frameworky . . . 10
  - 1.6.1 Nette framework . . . 10
  - 1.6.2 Symfony . . . 10

2 Představení frameworku Laravel **11**
- 2.1 Základní informace . . . 11
- 2.2 Stručná historie . . . 12
- 2.3 Licence . . . 12
- 2.4 Oficiální webové stránky a dokumentace . . . 13
- 2.5 Styly pro psaní kódu . . . 13
- 2.6 Minimální systémové požadavky . . . 14
- 2.7 Instalace frameworku . . . 14
- 2.8 Konfigurace . . . 14
  - 2.8.1 Konfigurace prostředí . . . 15
- 2.9 Velikost instalace . . . 15
- 2.10 Adresářová struktura . . . 16

3 Základní vlastnosti **17**
- 3.1 Artisan . . . 17
- 3.2 Facades . . . 18
- 3.3 Routování . . . 19
- 3.4 Controllers . . . 20
- 3.5 Databáze . . . 20
  - 3.5.1 Podporované databáze . . . 20
  - 3.5.2 Konfigurace databáze . . . 21
  - 3.5.3 Schema Builder . . . 21
  - 3.5.4 Migrace . . . 22
  - 3.5.5 Třída DB . . . 22
  - 3.5.6 Query Builder . . . 23
  - 3.5.7 Eloquent ORM . . . 23
  - 3.5.8 Plnění databáze daty . . . 25
- 3.6 Šablonovací systém . . . 26
- 3.7 Formuláře . . . 26
- 3.8 Posílání e-mailů . . . 28
- 3.9 Ukládání hesel . . . 28
- 3.10 Autentizace uživatelů . . . 29
- 3.11 Validace dat . . . 30

3.12 Logování a ladění chyb . . . 31
3.13 IoC container . . . 31
3.14 Další komponenty . . . 33

4 Bezpečnost **35**
4.1 CSRF (Cross-Site Request Forgery) . . . 35
4.2 XSS (Cross-Site Scripting) . . . 36
4.3 SQL Injection . . . 37
4.4 Sessions . . . 37
4.4.1 Session hijacking . . . 37
4.4.2 Session fixation . . . 38
4.5 Mass Assignment (hromadné přiřazení) . . . 38

5 Jednotkové testování **40**
5.1 Laravel a jednotkové testování . . . 40
5.2 PHPUnit . . . 40

6 Zhodnocení **44**
6.1 Silné stránky . . . 44
6.1.1 Moderní způsob vývoje . . . 44
6.1.2 Přehledná a jednoduchá syntax . . . 44
6.1.3 Dokumentace . . . 44
6.1.4 Komunita, vzdělávání, knihy . . . 44
6.2 Slabé stránky . . . 45
6.2.1 Bezpečnost . . . 45
6.2.2 Bouřlivý nástup a vývoj frameworku . . . 45
6.2.3 Absence očekávaných komponent . . . 46
6.2.4 Uplatnění na trhu práce . . . 46

7 Tvorba API - praktická část **47**
7.1 Technické požadavky na API . . . 47
7.2 URI prefix . . . 47
7.3 Autentizace uživatelů . . . 47
7.4 JSON . . . 48
7.5 Nezávislost API na struktuře databáze . . . 48
7.6 Stránkování . . . 49
7.7 Jednotkové testování . . . 50

Závěr **53**

Conclusions **54**

A Příloha - Schema Builder a typy sloupců **55**

B Obsah přiloženého CD **56**

Literatura **57**

Seznam obrázků

1 Struktura databáze ukázkové aplikace 9
2 Logo frameworku Laravel 11
3 Příkazový řádek Artisan 18

Seznam tabulek

1 Frameworky a počet instalací ze serveru Packagist 9
2 Aktuální verze frameworků 12
3 Historie frameworku Laravel 12
4 Frameworky a typ licence 13
5 Oficiální webové stránky a dokumentace frameworků 13
6 Styly pro psaní kódu a standardy PSR 13
7 Minimální požadavky pro provoz frameworků 14
8 Doporučený způsob instalace frameworků 15
9 Velikost frameworků po instalaci 15
10 Frameworky a příkazový řádek 18
11 RESTful routování 20
12 Podporované databáze 21
13 Podpora migrací u frameworků 22
14 ORM 25
15 Nástroje pro plnění databáze daty 25
16 PHP frameworky a šablonovací systémy 26
17 Formuláře 27
18 Výchozí ovladač pro odesílání e-mailů 28
19 Způsob ukládání hesel 29
20 Autentizace a uživatelské role 30
21 Validace dat 31
22 Nástroje pro logování a ladění chyb 31
23 Ochrana proti útoku CSRF 36
24 Ochrana proti útoku XSS 36
25 Ochrana proti SQL injection 37
26 Ochrana před session fixation 38
27 Ochrana před Mass Assignment 39
28 Frameworky a testovací nástroje 43
29 Schema Builder - typy sloupců 55

1 Úvod

V posledních letech se na internetu objevilo mnoho webových PHP[1] frameworků. Lehce lze dohledat již více jak 50 různých variant a stále vznikají varianty nové. Některé jsou populární a známé více, jiné méně. Framework Laravel patří v současné době v zahraničí k těm nejpopulárnějším.

Tato práce postupně popisuje hlavní komponenty frameworku Laravel. Dále se zabývá bezpečností, jednotkovým testováním a tvorbou API[2] webové aplikace.

1.1 Předpoklady

Pro porozumění práci se předpokládá čtenářova znalost objektového programování v jazyku PHP a základní znalost dotazovacího jazyka SQL[3] pro práci s relačními databázemi. Vhodným předpokladem pro porozumění je také alespoň základní orientace v některém MVC[4] frameworku.

1.2 Co je to framework?

V práci velice často používám slovo framework. Význam tohoto slova hezky vysvětluje Jeff Croft. Framework je sada nástrojů, knihoven, konvencí a osvědčených postupů, které vytváří abstrakci nad rutinními úkoly do obecných modulů, které mohou být lehce znovu využity [1].

1.3 Důvod výběru tématu práce

Důvodem výběru tohoto tématu je má zkušenost s frameworkem Laravel a jeho stále větší popularita u PHP vývojářů na celém světě. V žebříčcích oblíbenosti PHP frameworků pro rok 2014 se Laravel umisťuje na předních pozicích, např. *Best PHP Frameworks for 2014* [2] nebo *13 PHP Frameworks to Help Build Agile Applications* [3]. Jeho popularitu lze vyčíst i z počtu instalací ze serveru Packagist[5], viz tabulka 1. Údaje jsou platné k datu 13. 4. 2015.

1.4 Struktura práce

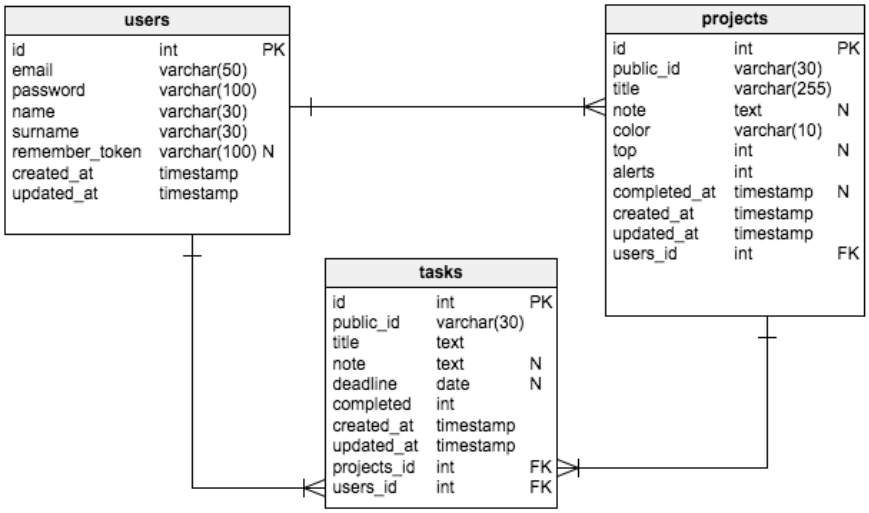
Jednotlivé kapitoly, respektive podkapitoly, obsahují teoretickou a praktickou část. V teoretické části představuji framework Laravel, následně v praktické části jej přibližuji ukázkou zdrojového kódu. V závěru kapitoly je obvykle tabulka, která obsahuje porovnání s ostatními frameworky. Sedmá kapitola obsahuje větší praktickou ukázku, zaměřenou na vytvoření API aplikace pro evidenci projektů a úkolů.

[1] PHP: Hypertext Preprocessor
[2] Application Programming Interface
[3] Structured Query Language
[4] Model-View-Controller
[5] Úložiště pro PHP balíčky - https://packagist.org

Tabulka 1: Frameworky a počet instalací ze serveru Packagist

Název frameworku	Počet instalací za posledních 30 dní
Laravel	352 711
Symfony	362 303
Nette	16 846
CakePHP	34 606
Zend Framework 2	78 091

Aplikace používá databázi o třech tabulkách (`users`, `projects` a `tasks`), kde uživatel vlastní mnoho projektů a projekt má mnoho úkolů. Strukturu tabulek a vazby mezi nimi popisuje obrázek 1.



Obrázek 1: Struktura databáze ukázkové aplikace

Ukázková aplikace je k dispozici na přiloženém CD.

1.5 Zdroje informací

Při zpracování práce jsem vycházel z oficiální dokumentace [4]. Užitečným zdrojem mi bylo také několik knih, které se tématem Laravel zabývají. Konkrétně jde o tyto:

1. Taylor Otwell. *Laravel: From Apprentice To Artisan*. Leanpub 2013.
2. Dayle Rees. *Laravel: Code Bright*. Leanpub 2014.
3. Jeffrey Way. *Laravel Testing Decoded*. Leanpub 2013.

Dále jsem čerpal z vlastních zkušeností s frameworkem Laravel. Zdůraznil bych i další důležitý zdroj informací - web Laracasts [5], který obsahuje video

tutoriály k frameworku Laravel a k programování v jazyce PHP. Autorem webu je populární lektor Jeffrey Way, který dříve pracoval pro firmu Envato a jejich vzdělávací portály Tuts+.

1.6 Další webové frameworky

V práci srovnávám Laravel také s dalšími PHP frameworky. Především jde o Nette framework verze 2.2 a Symfony verze 2.5.

1.6.1 Nette framework

Nette framework je webový framework pro jazyk PHP. Původním autorem Nette je David Grudl. Nyní se o další vývoj stará komunita Nette Foundation [6], jejíž hlavní postavou zůstává stále David Grudl. V České republice je tento framework u webových vývojářů velmi populární.

1.6.2 Symfony

Symfony je framework určený pro vývoj webových stránek a aplikací. Je napsán v jazyce PHP. Jde o svobodný software, šířený pod licenci MIT. První verze vyšla v říjnu 2005. Jejím autorem byl Fabien Potencier z francouzské agentury Sensio [7]. Tato agentura i nadále zůstává hlavním sponzorem frameworku Symfony.

Poznámka: Framework Laravel některé komponenty frameworku Symfony využívá.

2 Představení frameworku Laravel

2.1 Základní informace

Autorem frameworku Laravel je softwarový inženýr Taylor Otwell, který se svojí rodinou žije v americkém Arkansasu [8]. Do konce roku 2014 byl vývoj frameworku Laravel Otwellovou vedlejší činností. Od ledna 2015 je práce na frameworku naopak již jeho činností hlavní [9].

Obrázek 2: Logo frameworku Laravel

Na začátek práce přidávám i oficiální text, kterým se framework Laravel prezentuje [10].

Laravel je webový aplikační framework s výstižnou a elegantní syntaxí. Věříme, že vývoj aplikací může být zábavná a tvůrčí činnost, která vývojáře opravdu baví. Laravel se snaží obtížnosti u vývoje ulehčit zjednodušením běžných funkčností většiny webových projektů, jako je např. autentizace, routování, práce se sessions nebo kešování.

Cílem frameworku Laravel je, aby byl proces vývoje aplikací příjemný pro vývojáře, aniž by to ale negativně ovlivnilo jejich funkčnost. Šťastní vývojáři tvoří ten nejlepší kód. Pro tento účel jsme se snažili zkombinovat to nejlepší, co jsme se naučili u ostatních frameworků, a to bez ohledu na programovací jazyk (např. Ruby on Rails, ASP.NET MVC a Sinatra).

Laravel je dostupný, ale výkonný framework, poskytuje nástroje pro velké a robustní aplikace. Vynikající IoC container, přehledný migrační systém, pevně integrovaná podpora pro jednotkové testování. To jsou nástroje pro aplikace, které vytváříme.

2.2 Stručná historie

První verze frameworku Laravel vyšla v roce 2011. Aktuální stabilní vydání je verze 5.0.27 (19.04.2015).

Poznámka: V práci popisuji tyto verze frameworků: Laravel 4.2.11, Symfony 2.5.7 a Nette 2.2.6. Jsou to verze, které byly aktuální, když jsem začal práci v listopadu 2014 psát.

Tabulka 2: Aktuální verze frameworků (údaje platné k 19.04.2015)

Laravel	Nette	Symfony
5.0.27	2.3.1	2.6.6

Oproti svým konkurentům je Laravel poměrně mladý framework. První veřejná verze Nette frameworku vyšla v roce 2006. Symfony byl představen ještě o rok dříve, tzn. v roce 2005.

Tabulka 3: Historie frameworku Laravel [11]

verze	měsíc a rok vydání
Laravel 1	červen 2011
Laravel 2	listopad 2011
Laravel 3	únor 2012
Laravel 4	květen 2013
Laravel 5	leden 2015

2.3 Licence

Framework Laravel je svobodný software[6], distribuovaný pod licencí MIT, která umožňuje software používat téměř bez omezení. Lze jej kopírovat, modifikovat, slučovat, publikovat, distribuovat či prodávat. Jedinou podmínkou je zahrnutí textu licence do všech kopií softwaru [12].

U frameworku Nette si můžeme vybrat ze dvou typů licencí - BSD nebo GNU General Public Licence. Licence BSD nemá v podstatě žádné omezení, framework lze používat i pro komerční projekty. Taktéž licence GPL, jen je potřeba zachovat původní autorská práva [13].

Typy licencí jednotlivých frameworků popisuje tabulka 4.

[6]Software, který zaručuje uživatelům svobodu jej spouštět, kopírovat, distribuovat, studovat, měnit a zlepšovat

Tabulka 4: Frameworky a typ licence

Laravel	Nette	Symfony
MIT	New BSD nebo GNU GPL	MIT

2.4 Oficiální webové stránky a dokumentace

Oficiální webové stránky frameworku Laravel jsou na adrese http://laravel.com. Adresy oficiálních webových stránek a dokumentací ostatních frameworků obsahuje tabulka 5.

Tabulka 5: Oficiální webové stránky a dokumentace frameworků

Framework	Oficiální webové stránky	Oficiální dokumentace
Laravel	http://laravel.com	http://laravel.com/docs
Nette	http://nette.org	http://doc.nette.org/cs/
Symfony	http://symfony.com	http://symfony.com/doc/current/index.html

2.5 Styly pro psaní kódu

Laravel dodržuje standardy PSR-0 [14] a PSR-1 [15]. Kromě těchto standardů se předpokládá i dodržování následujících pravidel:

- Deklarace jmenného prostoru třídy je na stejném řádku jako `<?php`
- Znak `{` je na stejném řádku jako název třídy
- Metody a řídící struktury používají Allmanův způsob zápisu [16]
- Odsazuje se tabulátorem, zarovnává mezerami

Tabulka 6: Styly pro psaní kódu a standardy PSR

Laravel	Nette	Symfony
PSR-0, PSR-1	nedodržuje standardy PSR [17]	PSR-0, PSR-1, PSR-2

Tabulka 7: Minimální požadavky pro provoz frameworků

Položka	Laravel	Nette	Symfony
PHP	>=5.4	>=5.3.1	>=5.3.3
Další požadavky	PHP rozšíření MCrypt	viz nástroj Requirements Checker [18]	povolen JSON, ctype a v `php.ini` nastaven `date.timezone`
Nástroj pro otestování dalších minimálních požadavků	neobsahuje nástroj pro otestování	Requirements Checker	příkaz `php app/check.php`

2.6 Minimální systémové požadavky

Laravel vyžaduje pro svůj provoz verzi PHP 5.4 nebo vyšší a PHP rozšíření MCrypt. Blíže minimální požadavky popisuje tabulka 7.

2.7 Instalace frameworku

Doporučená instalace je přes Composer[7]. Existují tři způsoby, jak Laravel Composerem instalovat:

1. **instalátorem Laravel**
   Příkaz `composer global require "laravel/installer=~1.1"` instalátor nainstaluje. Poté stačí příkazem `laravel new` framework nainstalovat. Tento způsob instalace je následně rychlejší než příkaz `composer create-project`.

2. **příkazem composer create-project**
   `composer create-project laravel/laravel --prefer-dist`

3. **stažením ze serveru Github**
   Stáhneme poslední verzi frameworku Laravel a rozbalíme ji. Následně příkazem `composer install` doinstalujeme potřebné závislosti. Tento způsob instalace vyžaduje mít nainstalovaný i Git[8].

2.8 Konfigurace

Po instalaci framework Laravel v podstatě nepotřebuje žádnou další konfiguraci. Jen složka `app/storage` musí mít povolen zápis. Pokud provozujeme Laravel na serveru Apache, tak je nutné mít aktivní modul `mod_rewrite`.

[7] Nástroj pro správu závislostí v jazyku PHP - http://getcomposer.org
[8] Distribuovaný systém správy verzí - http://git-scm.com

Tabulka 8: Doporučený způsob instalace frameworků

Laravel	Nette	Symfony
Composer	Composer	Composer

V případě, že budeme v aplikaci např. využívat databázi nebo rozesílat e-maily, tak nastavení těchto služeb se definuje v konfiguračních souborech umístěných ve složce `app/config`.

2.8.1 Konfigurace prostředí

U aplikace je výhodné mít rozdílnou konfiguraci pro vývojové, testovací a produkční prostředí. Rozdílnou konfiguraci pro různá prostředí definujeme ve složce `app/config` vytvořením nové složky, jejíž název odpovídá názvu prostředí, ve kterém aplikace aktuálně běží.

Ukázka:

Pro definici odlišných připojovacích údajů k databázi na lokálním prostředí vytvoříme složku `app/config/local`. Do ní zkopírujeme soubor `app/config/database.php` (je součástí standardní instalace frameworku). Není nutné kopírovat kompletní obsah souboru, ale jen tu část, která bude odlišná. Zbytek konfigurace se kaskádově převezme z nadřazené složky `app/config`.

Aby framework Laravel poznal v jakém prostředí aktuálně běží, tak v souboru `bootstrap/start.php` přiřadíme k názvu prostřední název počítače.

```
1 // bootstrap/start.php
2 $env = $app->detectEnvironment(array(
3   'local' => array('nazev-pocitace'),
4 ));
```

Poznámka: Jako výchozí prostředí se uvažuje prostředí produkční.

2.9 Velikost instalace

Framework Laravel po instalaci na disku zabírá přibližně 25 MB.

Tabulka 9: Velikost frameworků po instalaci

Laravel	Nette	Symfony
25.6 MB	4.4 MB	54.3 MB

2.10 Adresářová struktura

Adresářová struktura frameworku Laravel je po instalaci následující:

```
|-app                - adresář s aplikací Laravel
|---commands         - CLI příkazy
|---config           - konfigurační soubory
|-----local
|-----packages
|-----testing
|---controllers      - třídy controllers
|---database         - soubory pro práci s databází
|-----migrations     - migrace
|-----seeds          - třídy pro plnění databáze
|---lang             - soubory s jazykovou lokalizací
|-----en
|---models           - třídy modelové vrstvy
|---start            - spouštěcí skripty
|---storage          - odkládací složka pro logy a kešování
|-----cache
|-----logs
|-----meta
|-----sessions
|-----views
|---tests            - jednotkové testy
|---views            - šablony
|-----emails
|-------auth
|-bootstrap          - spouštěcí soubor aplikace
|-public             - veřejná složka aplikace
|---packages
|-vendor             - adresář pro knihovny
|-artisan            - CLI
|-composer.json      - závislosti projektu
|-composer.lock
|-CONTRIBUTING.md
|-phpunit.xml        - konfigurační soubor pro PHPUnit
|-readme.md
|-server.php         - emulace lokálního serveru
```

3 Základní vlastnosti

Laravel využívá architekturu Model-View-Controller (MVC), která rozděluje zdrojový kód do tří nezávislých vrstev. Toto rozdělení na tři části má mnoho výhod. Aplikace je přehlednější, lze ji v budoucnu jednodušeji vyvíjet a můžeme jednotlivé části testovat zvlášť [19].

- **Model** - vrstva obsahující aplikační logiku a data
- **View** - vrstva starající se o zobrazení výstupu, u webových frameworků obvykle s využitím šablonovacího systému
- **Controller** - řídící vrstva, zpracovává požadavky, mění stav modelu

MVC architekturu představil Trygve Reenskaug již v sedmdesátých letech 20. století [20]. Mnoho webových frameworků tento typ architektury využívá, Symfony a Nette nejsou výjimkou. Respektive Nette využívá architekturu MVP, což je v podstatě pokročilá varianta architektury MVC. Písmeno P v názvu MVP označuje presenter, který má obdobnou roli jako controller. Rozdíl obou architektur popisuje David Grudl např. na webu zdrojak.cz [21].

3.1 Artisan

Artisan je příkazový řádek (CLI[9]), který je součástí frameworku Laravel. Obsahuje příkazy pro zrychlení a ulehčení opakujících se činností prováděných při tvorbě webových aplikací. Artisan umí generovat kostry jednotlivých částí aplikace, pracovat s migracemi a databází, přepínat stav aplikace, pracovat s frontou úloh, atd.

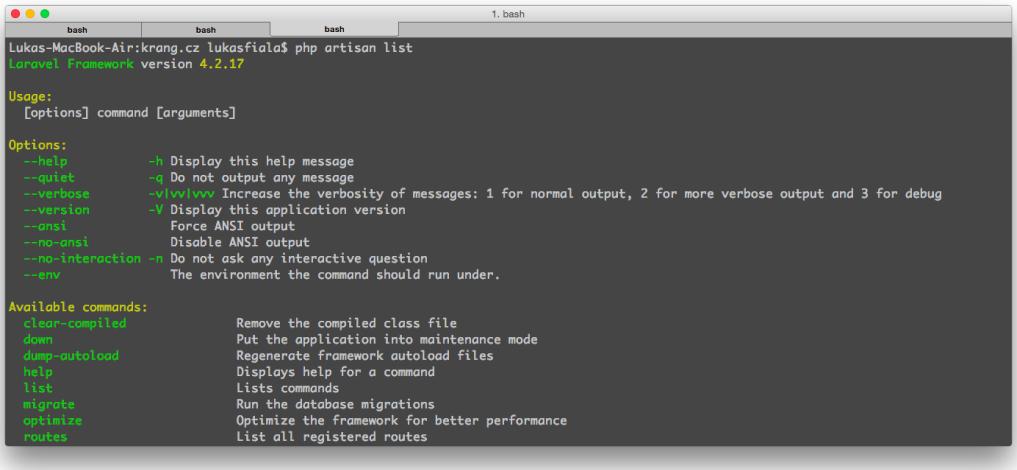
Poznámka: Artisan je postaven na komponentě Symfony Console.

Ukázka:

Výpis všech příkazů Artisanu zobrazíme příkazem `list`, respektive `php artisan list` (viz obr. 3). Každý příkaz obsahuje také nápovědu. Zobrazit ji můžeme příkazem `help` a názvem požadovaného příkazu. Například nápovědu k příkazu `routes`, který vypisuje URL routování aplikace, zobrazíme příkazem `php artisan help routes`.

Příkazy lze i volat jen pro zvolené prostředí. K tomuto účelu slouží přepínač `--env`. Migraci databáze na lokálním prostředí provedeme příkazem `php artisan migrate --env=local`. V tomto případě se použijí odpovídající konfigurační soubory pro lokální prostředí, aniž by to ovlivnilo produkční, případně jiné prostředí.

[9] Command Line Interface



Obrázek 3: Příkazový řádek Artisan

Tabulka 10: Frameworky a příkazový řádek

Laravel	Nette	Symfony
Artisan	neobsahuje	Console Component

3.2 Facades

Na internetu jsou články, které kritizují přílišné používání statických metod ve frameworku Laravel [22]. Tento problém se týkal převážně frameworku Laravel verze 3. Většina komponent této verze byla dostupná právě přes statické metody. Např. kód pro vytvoření routeru vypadal takto:

```
1 Route::get('/', function() {
2   return 'Homepage';
3 });
```

Statické metody se používaly především pro přehlednější syntaxi. Nevýhodou statických metod je ale mj. obtížnější testování [22]. Ve frameworku Laravel verze 4 přišel Taylor Otwell s řešením, kdy zachoval přehlednou syntaxi, ale bez negativního vedlejšího efektu obtížného testování. Použil tzv. třídy Facades, které vycházejí z návrhového vzoru Facade. Jsou to v podstatě „aliasy“, které při svém zavolání vykonají odpovídající metodu objektu, který je uložen v containeru.

Pokud se vrátím k předchozímu příkladu, tak metoda `Route::get()` by ve skutečnosti vykonala tento kód:

```
1 $app['router']->get('/',function() {
2   return 'Homepage';
3 });
```

Facades se nastavují v souboru `app/config/app.php`, kde určujeme název a odpovídající třídu.

```
'Artisan' => 'Illuminate\Support\Facades\Artisan'
```

Využití Facades nám také přináší ohebnost a možnost budoucích úprav aplikace. Ještě jednou se vrátím k předchozímu příkladu. V případě, že nám z nějakého důvodu nevyhovuje implementace routeru, který používá Laravel a chceme, třeba jen v určité části aplikace použít router vlastní, tak můžeme původní router předefinovat na nový takto:

```
$app['router'] = new MyAppRouter();
```

Zbytek kódu aplikace při využití Facades není třeba měnit.

Poznámka: Při použití Facades se některým programátorům může jevit kód méně přehledný (nejsou jasné závislosti). Používání Facades ve frameworku Laravel není povinné, aplikace lze psát kompletně bez nich. Facades berme jako syntaktický cukr, který má zpřehlednit kód a ulehčit práci.

3.3 Routování

U většiny webových aplikací se bez routování neobejdeme. Je to velice důležitá část aplikace, která zpracovává HTTP požadavky a dále je směruje do „těla“ aplikace. Ale i obráceně jsme schopni vygenerovat dané akci odpovídající URL adresu [23].

Webové frameworky jsou v tomto směru vybaveny spoustou nástrojů pro práci s routováním aplikace. Laravel obsahuje mj. tyto nástroje: podpora HTTPS, subdomény, filtrování, pojmenování routerů, atd.

Ukázka:

Ukázkovou aplikaci a její část pro zobrazení detailu vybraného projektu můžeme provozovat např. na adrese `http://domena.cz/app/projects/23`. Pokud si uvedenou adresu rozebereme podrobněji, zjistíme, že využíváme HTTP protokol pro přístup k aplikaci umístěné na doméně `domena.cz`. Část adresy `app/projects/23` slouží ke správnému routování (nasměrování) požadavku na aplikační logiku. V tomto případě chceme zobrazit projekt, který má identifikátor s číslem 23.

Routování nastavujeme v souboru `app/routes.php`. Výše uvedený příklad pro zobrazení detailu projektu může vypadat v souboru `app/routes.php` následovně:

```
Route::get('app/projects/{id}', ['uses' =>
'ProjectsController@show]);
```

Používáme třídu `Route`, následně určujeme zvolenou metodou typ HTTP požadavku (v tomto případě `GET`), na kterém bude router naslouchat. V argumentech metody `get` určujeme podobu URL adresy a také controller a metodu, která s požadavkem bude pracovat.

Poznámka: V routeru by se mělo řešit jen routování, samotná aplikační logika by měla být řešena v jiných částech aplikace, např. v controllerech.

3.4 Controllers

Controller (řadič) zpracovává požadavky z routeru a následně řeší aplikační logiku. Ve frameworku Laravel se controllery ukládají do složky `app/controllers`. Můžeme je vytvářet ručně nebo použít příkaz z příkazového řádku Artisan.

Ukázka:

Controller pro práci s projekty vytvoříme příkazem `php artisan controller:make ProjectsController`. Tento příkaz vytvoří soubor `app/controllers/ProjectsController.php`, který obsahuje základní kostru controlleru. Ke controlleru přiřadíme router kódem:

```
1 // app/routes.php
2 Route::resource('products', 'ProductsController');
```

Metoda `resource` zaregistruje router a odpovídající REST[10] akce viz tabulka 11.

Tabulka 11: RESTful routování

Typ požadavku	URI	Akce	Název routeru
GET	products	index	products.index
GET	products/create	create	products.create
POST	products	store	products.store
GET	products/{id}	show	products.show
GET	products/{id}/edit	edit	products.edit
PUT/PATCH	products/{id}	update	products.update
DELETE	products/{id}	destroy	products.destroy

Poznámka: Pro zobrazení struktury routování aplikace lze použít příkaz `php artisan routes`.

3.5 Databáze

Laravel má pro práci s databázemi hned několik nástrojů. Je to Schema Builder, třída DB, Query Builder a Eloquent ORM. V následujících kapitolách tyto nástroje popisuji.

3.5.1 Podporované databáze

Laravel podporuje tyto databázové systémy: MySQL, Postgres, SQLite a SQL Server. Podporované databáze u ostatních frameworků popisuje tabulka 12.

[10] Representational state transfer - způsob, jak vytvořit, číst, editovat nebo smazat data

[11] Symfony používá ORM Doctrine, který je od frameworku oddělený a jeho použití je volitelné

Tabulka 12: Podporované databáze

Databáze	Laravel	Nette	Symfony[11]
MySQL	ano	ano	ano
Postgres	ano	ano	ano
SQLite	ano	ano	ano
SQL Server	ano	ano	ano
Oracle	ne	ano	ano
ODBC	ne	ano	ano

3.5.2 Konfigurace databáze

Databáze se konfiguruje v souboru `app/config/database.php`, kde zadáváme typ použité databáze, přihlašovací údaje, výchozí připojení, atd.

Poznámka: V konfiguračním souboru lze také nastavit, které připojení se použije pro příkazy typu `SELECT` a naopak které připojení pro příkazy typu `INSERT`, `UPDATE` a `DELETE`.

3.5.3 Schema Builder

Schema Builder (SB) je třída obsahující metody pro definici a úpravy struktur databázových tabulek. SB pracuje nad všemi podporovanými databázemi.

Poznámka: Schema Builder využívají i migrace, které jsou popsány v kapitole 3.5.4

Ukázka:

Kód pro vytvoření tabulky `projects` a požadovaných sloupců v této tabulce může vypadat takto:

```
1  Schema::create('projects', function(Blueprint $table)
2  {
3  $table->increments('id');
4        $table->string('public_id',30);
5        $table->string('title',50);
6        $table->text('note')->nullable();
7        $table->string('color',10);
8        $table->boolean('top')->nullable();
9        $table->boolean('alerts')->default(0);
10       $table->timestamp('completed_at')->nullable();
11       $table->integer('user_id')->unsigned();
12       $table->foreign('user_id')->references('id')->on('users')->
             onDelete('cascade');
13       $table->timestamps();
14 });
```

Poznámka: SB umožňuje definovat typy sloupců viz tabulka 29 v příloze A.

3.5.4 Migrace

Migrace si představme jako verzovací systém nad databází. Umožňují týmu spolupracovníků upravovat databázovou strukturu, aniž by to ohrozilo nejen chod aplikace, ale i samotnou databázi. Migrace jsou třídy, které se ukládají do složky `app/database/migrations`. Obsahují dvě metody - `up` a `down`. V těchto metodách se obvykle využívá Schema Builder.

Ukázka:

Ukázková aplikace obsahuje tabulku `projects`. Nejjednodušší způsob, jak pro ni vytvořit migraci, je použít následující Artisan příkaz:

```
1 php artisan migrate:make create_projects_table --table=projects
```

Tento příkaz vytvoří do složky `app/database/migrations` třídu, která obsahuje dvě metody. Metodu `up`, která je provedena při zavolání migrace (`php artisan migrate`) a metodu `down`, která je provedena při odvolání migrace (`php artisan migrate:rollback`).

Do metody `up` doplníme kód Schema Builderu pro definici jednotlivých sloupců tabulky `projects`. Kód je stejný jako v předchozí kapitole. Do metody `down` doplníme naopak kód pro odstranění tabulky `projects`.

```
1 Schema::drop('projects');
```

Následně příkazem `php artisan migrate` dojde k vytvoření tabulky `projects` v databázi. Pokud by nám tabulka nebo její struktura nevyhovovala, tak můžeme migraci odvolat příkazem `php artisan migrate:rollback`, upravit migraci a opět zavolat `php artisan migrate`.

Tabulka 13: Podpora migrací u frameworků

Laravel	Nette	Symfony
ano	ne	ano[12]

3.5.5 Třída DB

Pokud máme správně nastavené údaje k databázi, tak můžeme využívat třídu `DB` a její metody pro práci s databází.

Ukázka:

Dotaz typu `SELECT`, který z tabulky `projects` vybere projekt mající `id = 12`, vypadá následovně:

```
1 DB::select('select * from projects where id = ?', array(12));
```

[12] U Symfony je třeba využít DoctrineMigrationsBundle

Metoda `select` vrací pole položek z databáze. Ve třídě `DB` máme k dispozici taktéž metody `insert`, `update` a `delete`. Metody `update` a `delete` po vykonání vracejí počet ovlivněných řádků.

Metoda `transaction` je určena pro provedení více příkazů v transakci. Pokud dojde k výjimce, tak se automaticky zavolá příkaz `rollback`.

```
1 DB::transaction(function()
2 {
3    DB::table('projects')->update(array('title' => 'Wohoo'));
4    DB::table('projects')->delete();
5 });
```

3.5.6 Query Builder

Query Builder je další rozhraní pro práci s databází. Je abstraktnější než třída `DB`, navíc automaticky chrání před SQL injection.

Ukázka:

Následující kód je určen pro výběr projektů, které mají příznak `top = 1`, neboli jde o důležité projekty.

```
1 DB::table('projects')->where('top', 1)->get();
```

Další ukázka kódu je určena pro výběr nedokončených projektů.

```
1 DB::table('projects')->whereNull('completed_at')->get();
```

3.5.7 Eloquent ORM

Eloquent ORM[13] poskytuje jednoduchou implementaci návrhového vzoru Active Record[14]. Každá databázová tabulka má vlastní model, který se stará o interakci s touto tabulkou.

Ukázka:

V ukázkové aplikaci máme tabulku `projects`. Ve složce `app/models` k ní vytvoříme odpovídající model. Můžeme ho vytvořit dvěma způsoby:

1. příkazem v příkazovém řádku Artisan
   `php artisan generate:model Project`
   Tento příkaz vytvoří do složky `app/models` soubor se třídou `Project`.

2. vytvořením souboru `app/models/Project.php` s tímto obsahem:

```
1 class Project extends \Eloquent {
2       protected $table = 'projects';
3 }
```

[13] Object-relational mapping
[14] Architektonický návrhový vzor pro práci s datovými zdroji

Poznámka: Model by měl mít název jako tabulka, ale v jednotném čísle. My máme tabulku `projects` a k ní tedy model `Project`.

Po vytvoření modelu již můžeme využívat Eloquent ORM, viz následující ukázkový kód pro práci s projekty.

```
1  // nacteni vsech projektu
2  $projects = Project::all();
3
4  // nacteni projektu s identifikatorem 1
5  $project = Project::find(1);
6
7  // editace projektu
8  $project = Project::find(1);
9  $project->title = 'A Brief History of Time';
10 $project->save();
11
12 // vytvoreni noveho projektu
13 Project::create([
14    'public_id' => '811942',
15    'title' => 'The Nature of Space and Time',
16    'color' => 'red',
17    'top' => 1,
18    'user_id' => 1
19 ]);
20
21 // smazani projektu s identifikatorem 2
22 Project::find(2)->delete();
```

Z výše uvedeného kódu je patrné, že práce s databází je při využití Eloquent ORM přehledná a jednoduchá. Pracovat lze i s tabulkami, které jsou ve vzájemném vztahu. Například tabulka `users` je ve vztahu 1 ku N s tabulkou `projects`, neboli uživatel může mít mnoho projektů a projekt má právě jednoho uživatele. Pro tento případ do modelu `app/models/User.php` doplníme metodu `projects`:

```
1 public function projects()
2 {
3       return $this->hasMany('Project');
4 }
```

A do modelu `app/models/Project.php` metodu `user`:

```
1 public function user()
2 {
3       return $this->belongsTo('User');
4 }
```

Následně lze pracovat i se vztahy mezi tabulkami, např. viz následující ukázka.

```
1 // nacteni vsech projektu od uzivatele s identifikatorem 1
2 $projects = User::find(1)->projects;
3
4 // nacteni jmena vlastnika projektu s identifikatorem 2
5 $owner = Project::find(2)->user()->name;
```

Nejčastěji používané ORM u ostatních frameworků popisuje tabulka 14.

Tabulka 14: ORM

Laravel	Nette	Symfony
Eloquent ORM	Dibi*	Doctrine*

* V základní instalaci nejsou součástí frameworku

3.5.8 Plnění databáze daty

Laravel poskytuje způsob, jak naplnit databázi daty, a to nejen testovacími. Používá pro tento účel třídy, které se ukládají do složky `app/database/seeds`.

Ukázka:

Následující ukázka je určena pro vytvoření nového uživatele v tabulce `users`.

```
1  //app/database/seeds/UserTableSeeder.php
2  class UserTableSeeder extends Seeder {
3
4     public function run()
5     {
6        DB::table('users')->delete();
7
8        User::create(array(
9           'email' => 'foo@bar.com',
10          'password' => Hash::make('secret'),
11          'name' => 'Jane',
12          'surname' => 'Oliver',
13       ));
14    }
15 }
```

V prvním kroku se odstraní obsah tabulky `users`, poté se vytvoří jeden nový záznam. Samotné naplnění databáze provedeme Artisan příkazem `php artisan db:seed`.
Obdobné nástroje ostatní frameworky v základní instalaci neobsahují, viz tabulka 15.

Tabulka 15: Nástroje pro plnění databáze daty

Laravel	Nette	Symfony
Seeder	neobsahuje	DoctrineFixturesBundle*

* V základní instalaci není součástí frameworku

3.6 Šablonovací systém

Webové frameworky obvykle obsahují nástroje pro práci se šablonami. Účelem těchto nástrojů je oddělení HTML od kódu programovacího jazyka.

Ve frameworku Laravel máme možnost tvořit šablony dvěma způsoby. Prvním způsobem je tvorba šablon v čistém PHP. Druhým způsobem je využití šablonovacího systému Blade [24], kdy se soubory ukládají s příponou `.blade.php` do složky `app/views`.

Ukázka:

Následující ukázka šablony Blade vypisuje projekty příkazem `@foreach` v sekci `content`, která je součástí šablony `master` umístěné ve složce `app`.

```
1  @extends('app.master')
2
3  @section('content')
4
5     <h2>Projekty</h2>
6
7     @foreach ($projects as $project)
8     <p>
9        {{ $project->title }}
10
11       @if($project->top === 1)
12          <span>Top projekt</span>
13       @endif
14
15    </p>
16    @endforeach
17
18 @stop
```

Používané šablonovací systémy u ostatních frameworků popisuje tabulka 16.

Tabulka 16: PHP frameworky a šablonovací systémy

Laravel	Nette	Symfony
Blade	Latte	Twig

3.7 Formuláře

Mnoho webových aplikací pracuje s formuláři. Laravel obsahuje třídu `Form`, která návrh a zpracování formulářů značně ulehčuje. Obsahuje metody pro práci s jednotlivými prvky webových formulářů.

Ukázka:

Následující ukázka použití třídy `Form` v šablonovacím systému Blade je určena pro vytvoření formuláře pro zadávání nového projektu.

```
1  @extends('app.master')
2
3  @section('content')
4
5    {{ Form::open(['route' => 'app.projects.index']) }}
6
7    <div class="form-group">
8      {{ Form::label('title','Nazev projektu') }}
9      {{ Form::text('title',null, ['class' => 'form-control', 'required
           ']) }}
10     <p class="text-danger">{{ $errors->first('title') }}</p>
11   </div>
12
13   <div class="form-group">
14     {{ Form::label('color','Barva projektu') }}
15     {{ Form::color('color',null, ['class' => 'form-control', '
           required']) }}
16     <p class="text-danger">{{ $errors->first('color') }}</p>
17   </div>
18
19   <div class="form-group">
20     {{ Form::label('note','Popis projektu') }}
21     {{ Form::textarea('note',null, ['class' => 'form-control', 'rows'
           => '2']) }}
22   </div>
23
24   <div class="form-group">
25     {{ Form::checkbox('alerts', '1', null, ['id' => 'alerts']); }}
26     {{ Form::label('alerts','Zasilat upozorneni e-mailem') }}
27   </div>
28
29   <div class="form-group">
30     {{ Form::submit('Vytvorit projekt', ['class' => 'btn btn-success
           btn-embossed center-block mt']) }}
31   </div>
32
33   {{ Form::close() }}
34
35 @stop
```

Webové frameworky běžně nástroje pro tvorbu a práci s formuláři obsahují, viz tabulka 17.

Tabulka 17: Formuláře

Laravel	Nette	Symfony
třída Form	Nette\Forms	komponenta Form

Poznámka: Nette vytváří formuláře i s JavaScriptovou validací na straně uživatele aplikace.

3.8 Posílání e-mailů

Rozesílání e-mailů je častým úkolem webových aplikací. Ve frameworku Laravel se způsob odesílání e-mailů nastavuje v konfiguračním souboru `app/config/mail.php`. Laravel ve výchozím nastavení využívá pro práci s e-maily knihovnu Swift Mailer[15]. Pokud by nám nevyhovovala, tak lze ve výše uvedeném souboru nastavit ovladač jiný.

Ukázka:

Ukázka zdrojového kódu pro odeslání e-mailu, který pro tělo zprávy využije šablonu Blade.

```
1 Mail::send('emails.welcome', array('key' => 'value'), function(
      $message)
2 {
3     $message->to('john@doe.com', 'John Doe')->subject('Welcome!');
4     $message->attach($pathToFile);
5
6 });
```

Prvním argumentem metody `send` je název šablony, která bude použita jako tělo odesílaného e-mailu. Druhým argumentem jsou data, která budeme mít k dispozici v uvedené šabloně zkrze proměnou `$key`. Třetím argumentem je Closure[16], ve kterém definujeme další údaje o e-mailu (např. předmět, adresáta, přidání přílohy, ...).

Výchozí ovladače pro odesílání e-mailů u ostatních frameworků popisuje tabulka 18.

Tabulka 18: Výchozí ovladač pro odesílání e-mailů

Laravel	Nette	Symfony
Swift Mailer	PHP funkce `mail`	Swift Mailer

3.9 Ukládání hesel

Pokud ukládáme v aplikaci hesla uživatelů, je nutné je hašovat vhodným algoritmem [25]. Laravel obsahuje třídu `Hash`, která poskytuje metody pro bezpečné ukládání hesel. Využívá k tomu hašovací funkci Bcrypt.

Ukázka:

Heslo lze zahašovat následovně: `$password = Hash::make('tajneheslo');` Když následně potřebujeme ověřit správnost hesla, lze použít:

[15] http://swiftmailer.org
[16] Uzávěr

```
1 if(Hash::check('tajneheslo', $heslo))
2 {
3    // Bylo zadano spravne heslo
4 }
```

I ostatní frameworky používají funkci Bcrypt, viz tabulka 19.

Tabulka 19: Způsob ukládání hesel

Laravel	Nette	Symfony
Bcrypt	Bcrypt	Bcrypt

3.10 Autentizace uživatelů

U webových aplikací často řešíme autentizaci (česky ověření) uživatelů. Část aplikace je přístupná všem a naopak část aplikace je přístupná jen registrovaným a přihlášeným uživatelům.

Laravel má několik nástrojů pro autentizaci uživatelů. Autentizace se nastavuje v souboru `app/config/auth.php`. Konkrétně v tomto souboru nastavujeme ovladač, který bude autentizaci provádět, a dále také název databázové tabulky a modelu, vůči kterému se budou uživatelé autentizovat. Ve výchozím nastavení Laravel používá ovladač `eloquent`, model `User` a databázovou tabulku `users`.

Ukázka:

Velká část ukázkové aplikace je dostupná jen přihlášeným uživatelům. Jde o URL adresy, které začínají `/app`. Tohoto omezení docílíme použitím filtru `auth` v routeru.

```
1 // app/routes.php
2 Route::group(['prefix' => 'app', 'before' => 'auth'], function()
3 {
4    ...
5 });
```

Pokud do prohlížeče zadáme jakoukoliv adresu, která bude začínat `/app`, tak budeme přesměrováni na přihlašovací formulář. Filtr `auth` je definován v souboru `app/filters.php`. Laravel má i filtr `auth.basic` pro jednoduché ověření přístupu (HTTP Basic Authentication). U tohoto typu autentizace není nutné vytvářet stránku s formulářem pro zadání přihlašovacích údajů.

Po odeslání formuláře s přihlašovacími údaji lze ověřit jejich správnost metodou `attemp` ze třídy `Auth`. Metodě `attemp` předáme dva argumenty - `email` a `heslo`.

```
1 // app/controllers/AuthController.php
2 $userdata = array(
3       'email' => Input::get('email'),
4       'password' => Input::get('password')
5 );
```

```
6
7  if(Auth::attempt($userdata))
8  {
9     // udaje byly spravne, presmerovani do aplikace
10       return Redirect::intended('/app');
11 }
```

Pokud byly zadány platné přihlašovací údaje, tak je uživatel metodou `intended` ze třídy `Redirect` přesměrován na URL adresu, ze které přišel původní požadavek na autentizaci. V aplikaci lze přihlášeného uživatele ověřit metodou `check` ze třídy `Auth`.

```
1  if(Auth::check())
2  {
3        // Uzivatel je prihlasen ...
4  }
```

Laravel má i nástroje pro resetování zapomenutého hesla. Naopak čím Laravel od svých konkurentů nedisponuje, je podpora uživatelských rolí. Pokud se podíváme např. na framework Symfony, tak ten má podporu uživatelských rolí velice rozsáhlou [26].

Tabulka 20: Autentizace a uživatelské role

	Laravel	Nette	Symfony
Nástroje pro autentizaci	ano	ano	ano
Uživatelské role	ne	ano	ano

3.11 Validace dat

Pokud webová aplikace je určena i pro zadávání uživatelských dat, tak před uložením do databáze bychom měli ověřit, zda jsou v požadovaném formátu. Laravel má k tomuto účelu třídu `Validator`. Ta obsahuje metody, které lze při a po validaci využít.

Ukázka:

V ukázkové aplikaci si na údaje o projektu aplikujeme několik omezení. Název a barva projektu jsou povinné položky. Dále název musí mít minimálně 2 znaky a maximálně 50 znaků. Barva může mít maximálně znaků 10. Tyto pravidla přiřadíme jako pole do proměnné `$rules`.

```
1  // app/controllers/ProjectsController.php
2  $rules = [
3     'title' => 'required|between:2,50',
4     'color' => 'required|max:10'
5  ];
```

Pro vytvoření validátoru je určena metoda `make` ze třídy `Validator`.

```
1 $validator = Validator::make(Input:all(), $rules);
```

Prvním argumentem metody `make` jsou data, která budeme validovat. Ve druhém argumentu jsou pravidla, která budou na validaci dat aplikována. Zda validace dat byla úspěšná ověříme metodou `fails`. Validátor nám také vrací zprávy s informacemi, kde u validace nastal případný problém.

```
1 if ($validator->fails())
2 {
3     return Redirect::route('app.projects.create')->withErrors(
          $validator)->withInput();
4 }
```

Laravel obsahuje přibližně 40 předdefinovaných validačních pravidel, ale lze vytvářet i pravidla vlastní.

Poznámka: Laravel validuje data jen na straně serveru. Naopak framework Nette disponuje i validací formulářů na uživatelské straně JavaScriptem.

Tabulka 21: Validace dat

	Laravel	Nette	Symfony
Validace dat	ano	ano	ano
JavaScriptová validace	ne	ano	ne

3.12 Logování a ladění chyb

Pří vývoji je důležité být přehledně a jasně informován o chybách v aplikaci. Laravel, stejně jako framework Symfony, používá logovací knihovnu Monolog, která poskytuje spoustu užitečných nástrojů pro logování a ladění nejen chybových stavů. Framework Nette využívá nástroj Tracy (znám také pod názvem Laděnka).

Tabulka 22: Nástroje pro logování a ladění chyb

Laravel	Nette	Symfony
Monolog	Tracy	Monolog

3.13 IoC container

Inversion of Control (IoC) container je nástroj pro správu závislostí tříd. Framework Laravel je složen z různých částí (např. databázová vrstva, routovací vrstva, validační nástroje, šablonovací systém, atd.). A právě v containeru do sebe tyto části zapadají a tvoří tak celý framework Laravel. Jak zmiňuje Dayle Rees,

container slouží pro ukládání věcí [27]. V tomto případě věcmi myslí právě jednotlivé součásti frameworku Laravel.

Při spouštění aplikace vytvořené ve frameworku Laravel se IoC container vytváří jako první. Je to objekt, který vznikne ze třídy `Illuminate\Foundation\Application` a je přiřazen do proměnné `$app`. Pro přístup k této proměnné kdekoliv v aplikaci lze použít pomocnou funkci `app()`.

Poznámka: Všechny komponenty frameworku Laravel jsou ve jmenném prostoru `Illuminate`. Tento název vznikl v začátcích frameworku a přetrval i do současné verze. Pro práci s frameworkem Laravel je dobré si tento název zapamatovat.

IoC container je nejdůležitější částí frameworku, která drží vše pohromadě. I to je jeden z důvodů, proč pro práci s frameworkem Laravel je dobré IoC containeru dobře porozumět. IoC container umožňuje psát následující kód:

```
1  class Baz {}
2
3  class Bar {
4     public $baz;
5
6     public function __construct(Baz $baz)
7     {
8        $this->baz = $baz;
9     }
10 }
11
12 Route::get('bar', function(Bar $bar) {
13    var_dump($bar);
14 });
```

Výše uvedený kód je určen k vytvoření routeru, který využívá třídu `Bar`. Díky IoC containeru stačí napsat `Bar $bar`, což je velice výhodné mj. pro případné budoucí úpravy kódu aplikace, jelikož se nemusíme zajímat o závislosti jednotlivých tříd. Laravel sám pozná, že potřebuje instanci třídy `Bar` a ta instanci třídy `Baz`. Za povšimnutí stojí fakt, že v uvedeném kódu jsme nikde nemuseli psát `$bar = new Bar(new Baz);`.

Ukázka:

V ukázkové aplikaci máme `ProjectsController`, který obsahuje metody pro výpis všech projektů (`index`), zobrazení detailu projektu (`show`), smazání projektu (`destroy`), atd. Data o projektech máme uložena v databázi, tzn. chceme je načíst z databáze a následně je předat do `views` a prezentovat uživateli. Cílem je ale také možnost jednoduchých budoucích úprav aplikace. Např. změníme úložiště, kdy data o projektech nebudou uložena v databázi, ale na pevném disku. Zde výhodně využijeme IoC container.

Pro ukládání tříd a rozhraní pro práci s daty si vytvoříme novou složku, např. `app/repositories`. Poté ve složce vytvoříme rozhraní `ProjectRepositoryInterface`, které má dvě metody - `all` a `store`. Metoda `all` vypisuje všechny projekty, metoda `store` ukládá nový projekt.

```
1  // app/repositories/ProjectRepositoryInterface.php
2  interface ProjectRepositoryInterface {
3     public function all();
4     public function store($data);
5  }
```

Taktéž vytvoříme implementaci výše uvedeného rozhraní `ProjectRepositoryInterface`. Může vypadat následovně:

```
1  // app/repositories/ProjectRepository.php
2  class ProjectRepository implements ProjectRepositoryInterface {
3
4  public function all() {
5        return Project::all();
6  }
7
8  public function store($request) {
9        return Project::create($request);
10 }
```

Následně frameworku Laravel musíme říct, kterou implementaci rozhraní má používat. To uděláme např. v souboru `app/routes.php` metodou `bind`.

```
1  // app/routes.php
2  App::bind('ProjectRepositoryInterface','ProjectRepository');
```

Výhodou tohoto řešení je jednoduchý způsob, jak změnit implementaci rozhraní za jiné. Když bychom chtěli data ukládat na disk a ne do databáze, tak vytvoříme třídu `ProjectRepositoryFilesystem` pro ukládání dat na pevný disk a v `App::bind` ji nastavíme místo třídy `ProjectRepository`. Zbytek kódu aplikace nebude nutné měnit.

V `ProjectsControlleru` nyní můžeme využít uvedené rozhraní a jeho nastavenou implementaci.

```
1  // app/controllers/ProjectsController.php
2  protected $projects;
3  function __construct(ProjectRepositoryInterface $projects) {
4        $this->projects = $projects;
5  }
6
7  public function index()
8  {
9     $projects = $this->projects->all();
10       ...
11 }
```

Poznámka: Další výhodou tohoto řešení je i jednodušší jednotkové testování.

3.14 Další komponenty

Framework Laravel disponuje i dalšími nástroji, které se často při vývoji webových aplikací využívají. Jde mj. o nástroje pro práci s různými kešovacími (caching) systémy, jazykové lokalizace, nástroje pro práci s frontou úkolů, strán-

kování vypisovaných dat, atd. Všechny tyto komponenty jsou dobře popsány v oficiální dokumentaci.

4 Bezpečnost

Důvodem, proč při vývoji webových stránek a aplikací využívat některý framework, je snaha vyhnout se bezpečnostním problémům a dírám ve zdrojovém kódu aplikace. Většina moderních frameworků tyto bezpečnostní problémy zná a snaží se jejich výskyt omezit. Postupně si projdeme zabezpečení frameworku před nejčastějšími útoky.

4.1 CSRF (Cross-Site Request Forgery)

Podstata útoku CSRF spočívá v tom, že uživatele přimějeme navštívit stránku napadané aplikace, která provádí nějakou akci, aniž by o tom uživatel věděl. Útok tím pádem může být snadno veden proti aplikacím, do kterých se útočník může sám přihlásit a tím zjistit jejich strukturu nebo které mají přístupný zdrojový kód [28].

Například aplikace může používat pro odstranění projektu adresu ve tvaru `http://domain.com/projects/delete/12`. Útočník v tomto případě na své stránky umístí nějaký prvek (např. obrázek), který bude na výše uvedenou adresu odkazovat. Pokud návštěvník tuto stránku navštíví, tak dojde k odstranění projektu, jelikož jde o oprávněný požadavek na odstranění projektu.

Poznámka: Je nepodstatné, zda se projekt odstraňuje metodou GET nebo POST. V obou případech útok CSRF hrozí. Pro mazání položek by se měla ale používat metoda POST.

Obranou před útokem CSRF je ověření, zda uživatel provedl operaci dobrovolně. To zajistíme vložením a následnou kontrolou autorizačního tokenu [29]. Pokud používáme šablonovací systém Blade a třídu `Form`, tak framework Laravel automaticky doplní do formuláře autorizační token, např. `<input name="_token" type="hidden" value="X2bHsEUZczZh5BoqtGdalyHmPwGrvnhM6ymacHH7">`. Ten lze následně ověřit filtrem `csrf`.

Ukázka:

Následující kód je určen pro vytvoření formuláře s CSRF tokenem, který se automaticky vygeneruje do šablony Blade.

```
1 {{ Form::model($user, ['url' => '/app/account','method' => 'patch']) }}
2  ...
3 {{ Form::close() }}
```

Také je nutné aplikovat CSRF filtr na router.

```
1 // app/routes.php
2 Route::patch('account', ['before' => 'csrf', 'uses' => 'UsersController@update']);
```

Pokud při zaslání požadavku na router token nesouhlasí, tak dojde k vyjímce `TokenMismatchException`.

Tabulka 23: Ochrana proti útoku CSRF

Laravel	Nette	Symfony
ano	ano	ano

Nástroje na obranu před útokem CSRF poskytují všechny testované frameworky, viz tabulka 23.

Poznámka: Je nutné pamatovat, že Laravel verze 4 filtr CSRF neaplikuje na router automaticky.

4.2 XSS (Cross-Site Scripting)

XSS je způsob narušení webových stránek, kdy je útočník modifikuje tak, že se v jejich kontextu provede podstrčený JavaScriptový kód. Tomuto typu útoku se dá poměrně jednoduše bránit ošetřováním vypisovaných dat, u kterých si nemůžeme být jisti jejich hodnotou – ručně nebo automaticky využitím šablonovacího systému [30]. Druhou možností je ošetřovat data na vstupu, ale dle [31] nejde o nejvhodnější řešení.

V šablonovacím systému Blade se útoku XSS můžeme bránit použitím trojice složených závorek, která vypisovaná data, mající v HTML speciální význam (např. `<` nebo `&`), zobrazí v escapovaném tvaru. Pokud data neošetříme, tak hrozí XSS útok, který může provádět v aplikaci nebezpečnou činnost.

Ukázka:

Pokud uživatel vyplní do názvu projektu např. `<b onmouseover=alert('Booom')>Název projektu</b>` a neošetřili bychom výstup před XSS útokem, tak by se po najetí myší na název projektu zobrazilo JavaScriptové varovné okno s textem `Booom`. V šabloně Blade výstup ošetříme před XSS útokem trojicí složených závorek.

```
1 // app/views/projects/index.blade.php
2 {{{ $project->title }}}
```

Všechna nebezpečná data (např. zadaná uživatelem) bychom měli ošetřit před XSS útokem.

Tabulka 24: Ochrana proti útoku XSS

Laravel	Nette	Symfony
ano	ano	ano

Poznámka: Nette Framework používá technologii Context-Aware Escaping, která výstupy ošetřuje automaticky [32].

4.3 SQL Injection

SQL Injection je útok, kdy uživatel modifikuje SQL dotaz pomocí předávaných dat. Obranou před útokem je oddělení SQL dotazů a uživatelských dat nebo ošetření všech vstupních dat [33]. Query Builder i Eloquent ORM automaticky chrání před SQL injection.

Ukázka:

Mějme například kód pro vyhledání uživatele dle jeho e-mailové adresy a hesla.

```
1 $email = $_POST['email'];
2 $password = sha1($_POST['password']);
3 $q = "SELECT * FROM users WHERE (email = '$email' AND password = '
    $password')";
```

Tento kód je nebezpečný a není chráněn před útokem SQL injection. Pokud jako email zadáme hodnotu `ozz@domain.com' or true)--`, tak výsledek bude platný bez ohledu na zadanou hodnotu hesla. Problém je u znaků `--`, které zakomentují zbytek SQL dotazu. Stejný dotaz v Eloquent ORM ošetřený proti SQL injection může vypadat takto:

```
User::where('email',$email)->where('password',$password)->get();
```

Útok SQL injection je již dlouhou dobu znám, proto také webové frameworky obsahují nástroje, jak tento bezpečnostní problém omezit, viz tabulka 25.

Tabulka 25: Ochrana proti SQL injection

Laravel	Nette	Symfony
ano	ano	ano

4.4 Sessions

Při práci se sessions (relacemi) hrozí hned několik typů útoků.

4.4.1 Session hijacking

Session hijacking je útok, kdy se útočníkovi podaří získat session identifikátor uživatele. Tento identifikátor slouží pro jeho jednoznačnou identifikaci, tzn. poté se útočník může za uživatele začít vydávat. Obrana proti tomuto typu útoku je téměř nemožná. Důležité je zamezit vyzrazení identifikátoru session [34].

Omezení útoku session hijacking:

1. Odebrat právo čtení adresáři, do kterého se identifikátory ukládají. Důvod je ten, že se sessions ve výchozím nastavení ukládají do souborů pojmenovaných dle identifikátoru session.

2. Používat poslední verze internetových prohlížečů. Např. Google Chrome testuje možnost podepisování cookies, což zamezuje útoku session hijacking [35].

3. Chránit před útokem XSS, jelikož může být veden k získání idetifikátoru session.

4. Pokud se session identifikátor přenáší v URL, tak je uložen v logu serveru, v historii prohlížeče, na vytištěných stránkách, ...

5. Vhodná konfigurace PHP [36]

4.4.2 Session fixation

Session fixation je útok, kdy útočník podvrhne oběti svůj session identifikátor. Následně počká, až se oběť přihlásí a může se začít za oběť útoku vydávat [34]. Obrana proti session fixation je jednoduchá. Po přihlášení uživateli přiřadíme nový session identifikátor.

Laravel automaticky po autentizaci uživatele regeneruje session identifikátor.

Tabulka 26: Ochrana před session fixation

Laravel	Nette	Symfony
ano	ano	ano

4.5 Mass Assignment (hromadné přiřazení)

Vzor Mass assignment (česky hromadné přiřazení) je způsob, jak naplnit objekt daty odeslanými z formuláře. Uveďme si jednoduchý příklad pro vytvoření nového uživatele:

```
1 $user = User::create(Input::all());
```

Třída `Input` a její metoda `all` automaticky zpracuje formulářové prvky dle jejich jména a přiřadí je sloupcům totožného jména v tabulce `users`.

Pokud bychom nebyli chráněni před hromadným přiřazením, tak by v tomto případě mohl útočník celkem jednoduše ovlivnit i citlivá data. V tabulce `users` by např. sloupci `admin` přiřadil hodnotu 1. Stačilo by formulář pro vytvoření uživatele modifikovat přidáním formulářového prvku `input` s názvem `admin` a hodnotou 1.

Eloquent modely ve frameworku Laravel chrání před hromadným přiřazením automaticky. Lze nastavit atributy, které je možné hromadně přiřadit a které ne. Přesněji, jsme schopni omezit pole atributů, které mohou být hromadně přiřazeny. K tomuto je určena proměnná `$fillable`.

```
1 // app/models/User.php
2 class User extends Eloquent {
3    protected $fillable = array('email', 'password', 'name', 'surname
         ');
4 }
```

Nebo naopak můžeme nastavit pole atributů, které nemohou být hromadně přiřazeny. V tomto případě použijeme proměnnou `$guarded`.

```
1 class User extends Eloquent {
2    protected $guarded = array('id', 'admin');
3 }
```

Zde mohou být všechny atributy, vyjma `id` a `admin`, hromadně přiřazeny. Na tuto vlastnost je třeba pamatovat!

Tabulka 27: Ochrana před Mass Assignment

Laravel	Nette	Symfony
ano	ano	ano

5 Jednotkové testování

V určitém momentu vývoje aplikace je obtížné, respektive nereálné ji testovat ručně. Proto se používají nástroje pro automatické testování. Testy píšeme, abychom si ověřili, že zdrojový kód aplikace pracuje správně a dle našeho očekávání [37]. Testování má i vedlejší efekt, kdy zdrojový kód aplikace bývá následně přehlednější, jelikož musíme dodržovat určité standardy pro psaní testovatelného zdrojového kódu [38].

Jednotkový test obvykle testuje pouze vybranou konkrétní jednotku, která je izolovaná od ostatních částí programu. Z pohledu procedurálního programování je jednotkou program, funkce, procedura, atd. Z pohledu objektově orientovaného programování je jednotkou obvykle třída, či konkrétní metoda [39]. Pokud test selže, tak přesně víme, kam se podívat.

5.1 Laravel a jednotkové testování

Framework Laravel je po instalaci nastaven pro testování testovacím frameworkem PHPUnit[17]. Jen do souboru `composer.json` musíme přidat framework PHPUnit a verzi, kterou chceme v aplikaci používat. Poté zavoláme `composer update`.

Kromě frameworku PHPUnit využívá Laravel také komponenty Symfony HttpKernel, DomCrawler a BrowserKit. Tyto komponenty umožňují při testování prohledávat a obsluhovat `views` aplikace. Provádějí to mj. simulací webového prohlížeče.

Jednotlivé testy se ukládají do složky `app/tests`. Testy jsou třídy, které dědí od třídy `TestCase`. Po instalaci frameworku je ve složce `app/tests` jeden ukázkový test. Tento test testuje, zda je dostupná kořenová URL adresa aplikace.

```
1 // app/tests/ExampleTest.php
2 class ExampleTest extends TestCase {
3
4       public function testBasicExample()
5       {
6             $crawler = $this->client->request('GET', '/');
7             $this->assertTrue($this->client->getResponse()->isOk());
8       }
9 }
```

Aplikaci otestujeme použitím příkazu `phpunit`, respektive `vendor/bin/phpunit` v příkazovém řádku.

5.2 PHPUnit

PHPUnit je defacto standard pro testování v PHP. Byl vytvořen pro tvorbu jednotkových testů, ale lze jej použít i pro tvorbu funkčních testů [40]. Funkční

[17] https://phpunit.de

testy jsou oproti jednotkovým testům širší a mohou testovat více částí (funkcí) aplikace současně [41].

PHPUnit pro ověření, zda se kód chová dle očekávání, využívá aserce neboli tvrzení (anglicky assertions). Například když potřebujeme otestovat, zda určitá hodnota odpovídá hodnotě `true`, lze použít aserci `assertTrue`.

```
1 class TrueTest extends TestCase {
2
3    public function testSomethingIsTrue()
4    {
5       $day = 'Monday';
6       $this->assertTrue($day === 'Monday');
7    }
8 }
```

Pokud test proběhne bez chyby, tak PHPUnit vypíše tyto informace:

```
1 PHPUnit 4.5.0 by Sebastian Bergmann and contributors.
2
3 .
4
5 Time: 103 ms, Memory: 7.75Mb
6
7 OK (1 test, 1 assertion)
```

Všimněte si tečky, která se objevila ve výsledku. Tato tečka představuje jeden test. Pokud bychom měli dva testy, byly by ve výsledku dvě tečky, atd. Naopak pokud při testu dojde k chybě, tak výsledek testu vypadá následovně:

```
1  PHPUnit 4.5.0 by Sebastian Bergmann and contributors.
2
3  F
4
5  Time: 132 ms, Memory: 8.00Mb
6
7  There was 1 failure:
8
9  1) TrueTest::testSomethingIsTrue
10 Failed asserting that false is true.
11
12 /Users/lukasfiala/Sites/laravel/app/tests/TrueTest.php:8
13
14 FAILURES!
15 Tests: 1, Assertions: 1, Failures: 1.
```

Tentokrát ve výsledku není tečka, ale písmeno F z anglického slova failure, česky selhání. Ve výsledku vidíme i detailnější informaci o chybě, mj. název testu, respektive metody.

PHPunit má k dispozici mnoho typů asercí, níže uvádím několik dalších příkladů. Kompletní seznam asercí je popsán v oficiální dokumentaci frameworku PHPUnit [42].

assertEquals

Metoda `assertEquals` je určena pro testování shody bez typové kontroly. Odpovídá operátoru ==.

```
$day = 'Monday';
$this->assertEquals('Monday', $day);
```

Metoda může mít až tři argumenty - očekávanou hodnotu, skutečnou hodnotu a volitelný text, který se zobrazí při chybě.

assertSame

Metoda `assertSame` kontroluje shodu, a to i včetně typové. V případě polí i pořadí prvků.

assertInstanceOf

Metoda `assertInstanceOf` ověřuje, zda je proměnná instancí dané třídy.

Framework Laravel obsahuje i několik vlastních asercí, které jsou určeny pro lehčí testování aplikací v něm vytvořených. Níže uvádím opět několik příkladů. Kompletní výčet lze dohledat v oficiální dokumentaci [43].

assertResponseStatus

Kontrola, zda je HTTP odpověď serveru zvoleného typu (404 - nebyl nalezen, 200 OK, ...).

```
public function testPageNotFound()
{
    $this->call('GET', '/neexistujici-stranka');
    $this->assertResponseStatus(404);
}
```

assertRedirectedToRoute

Kontrola, zda dojde k přesměrování na zvolený router.

```
$this->assertRedirectedToRoute('route.name');
```

assertViewHas

Kontrola, zda pohled (šablona) obsahuje požadovaná data.

```
public function testHomepageContainsNews()
{
    $this->call('GET', '/');
    $this->assertViewHas('news');
}
```

assertSessionHas

Kontrola, zda session obsahuje požadovaná data.

```
public function testSessions()
{
    $this->call('GET', '/');

    $this->assertSessionHas('name');

}
```

Poznámka: Laravel při testování automaticky používá testovací prostředí, jehož konfiguraci můžeme upravit ve složce `app/config/testing`.

Framework Symfony taktéž používá testovací framework PHPUnit. Naopak framework Nette používá pro jednotkové testování vlastní nástroj Nette Tester.

Tabulka 28: Frameworky a testovací nástroje

Laravel	Nette	Symfony
PHPUnit	Nette Tester	PHPUnit

6 Zhodnocení

6.1 Silné stránky

Silné stránky frameworku Laravel jsem rozdělil do několika následujících podkapitol.

6.1.1 Moderní způsob vývoje

Framework Laravel aplikuje moderní postupy pro vývoj webových aplikací. Využívá Composer pro správu závislostí, elegantní ORM, lze efektivně použít migrace pro práci s databází. Mnoho činností urychluje využitím příkazového řádku Artisan. Obsahuje velice užitečné komponenty pro práci s frontou událostí, e-maily, kešovacími systémy, routováním aplikace, atd. Po instalaci je připraven pro jednotkové testování.

Poznámka: Zde zmíním, že např. Jeffrey Way zvažoval odchod od PHP k Ruby on Rails, jelikož ho PHP už nebavilo. Až framework Laravel ho přesvědčil zůstat. Dle něj díky frameworku Laravel zažívá nejpoužívanější programovací jazyk pro webové stránky a aplikace novou renesanci [44].

6.1.2 Přehledná a jednoduchá syntax

Framework Laravel i jeho autor Taylor Otwell kladou velký důraz na přehlednou, výstižnou a čitelnou syntaxi.

6.1.3 Dokumentace

Dokumentace frameworku Laravel je velice dobře a přehledně zpracovaná. Taylor Otwell považuje kvalitní dokumentaci za velice důležitou, proto její tvorbě věnuje množství času a snaží se ji mít vždy aktuální.

6.1.4 Komunita, vzdělávání, knihy

Framework Laravel je velmi populární mezi vývojáři, což dokazují mj. počty fanoušků na různých serverech a sociálních sítích. Na serveru Github je Laravel nejpopulárnější a nejsledovanější PHP projekt [45]. Framework Symfony je na druhém místě, Nette se pohybuje někde kolem 180 místa.

Na sociální síti Twitter má Laravel 33 tisíc sledujících, Symfony 18 tisíc a Nette přes 2 tisíce sledujících.

K frameworku Laravel vyšlo mnoho knih. Na serveru Leanpub [46] jich nalezneme již 11. Na stejném serveru jsou čtyři knihy týkající se frameworku Symfony. Naopak k frameworku Nette nevyšla doposud žádná kniha. Podobná čísla k počtu vydaných knih nalezneme i na serveru Amazon. Pro framework Symfony jsou čísla na tomto serveru ještě příznivější, jelikož lze dohledat více jak 15 knih.

Dalším užitečným zdrojem informací je vzdělávací web Laracasts. Obsahuje přes 438 video tutoriálů, což je celkem více jak 43 hodin učebního materiálu [47].

Vždy v srpnu se koná oficiální konference k frameworku Laravel. Jmenuje se Laracon [48] a pořádá se na dvou místech. Jedno je vždy v Americe, druhé v Amsterdamu v Nizozemí.

Z výše uvedeného vyplývá, že framework Laravel je opravdu mezi vývojáři oblíbený. Nejen pro začátečníky je k dispozici mnoho vzdělávacích materiálů a knih, které jsou při vývoji užitečným pomocníkem. Je zřejmé, že i vývojáři o frameworku Laravel rádi píší a publikují různé články. Díky tomu vzniká kolem frameworku Laravel obrovská komunita nejen lidí, ale i učebních materiálů.

6.2 Slabé stránky

V této kapitole jsem se zaměřil na slabé stránky frameworku Laravel. Naopak čemu se nevěnuji, je výkonností porovnání, a to ať už k samotnému jazyku PHP, tak výkonností porovnání mezi frameworky. Důvodem, proč jsem se výkonnostním porovnáním frameworků nezabýval, je dle mého názoru především fakt, že pro vývoj většiny běžných webových aplikací jde o nepodstatné kritérium. A to především z pohledu, kdy využití frameworku oproti čistému PHP přináší spoustu výhod.

U hodně zatížených webových stránek a aplikací se stejně neobejdeme bez využití kešovacích systémů (např. Memcached[18] nebo Redis[19]), díky kterým je můžeme bezproblémově provozovat.

Poznámka: Nejnavštěvovanější instalace frameworku Laravel má kolem 16 miliónů návštěv denně [49].

6.2.1 Bezpečnost

Ačkoliv framework Laravel disponuje bezpečnostními nástroji chránícími před útoky XSS a CSRF, je nutné je volat ručně, tzn. dříve či později na to programátor aplikace může zapomenout.

V tomto směru má framework Nette výhodu, že všechny výstupy ošetřuje automaticky proti XSS, takže riziko, že programátor na něco zapomene je omezeno [29]. Symfony při použití šablonovacího systému Twig taktéž automaticky chrání před útokem XSS [50].

Proti CSRF se u všech tří frameworků musíme chránit ručně, u Symfony nastavením konfiguračního souboru [51], u Nette přidáním metody do formuláře [29], a u frameworku Laravel aplikací filtru `csrf` [52].

6.2.2 Bouřlivý nástup a vývoj frameworku

Verze frameworku Laravel obvykle nejsou zpětně kompatibilní. Přechod na vyšší verzi je různě obtížný dle zvolené verze [53]. Tento problém je do jisté míry určitě

[18] http://memcached.org
[19] http://redis.io

dán tím, že jde o nejmladšího zástupce porovnávaných frameworků. Do budoucna lze předpokládat, že se tento problém ustálí.

Za vývojem frameworku Laravel stojí především Taylor Otwell. Určitě to můžeme považovat za nevýhodu, že značná část vývoje frameworku stojí na bedrech jednoho člověka.

6.2.3 Absence očekávaných komponent

Framework Laravel má spoustu užitečných a zajímavých nástrojů a komponent. Ale některé, dle mého názoru podstatné, komponenty neobsahuje. Především jde o implementaci Access Control Listu (ACL)[20]. Nette i Symfony implementaci ACL mají.

Poznámka: K dispozici je mnoho uživatelských balíčků, které ACL do frameworku Laravel doplňují. Zmíním například Entrust [54].

Dále u frameworku Laravel chybí validace JavaScriptem na straně uživatele. Zde má Nette oproti konkurentům výhodu, jelikož JavaScriptovou validaci na rozdíl od frameworků Laravel a Symfony používá. K frameworkům Laravel a Symfony existují uživatelské balíčky, které tuto funkčnost taktéž doplňují.

Poznámka: Zde může někdo namítat, že JavaScript do PHP frameworku nepatří.

Taktéž Laravel neobsahuje žádnou komponentu pro práci s obrázky. Naopak Nette má užitečnou třídu `Image` pro zpracování obrázků.

6.2.4 Uplatnění na trhu práce

Ve výhodách jsem zmiňoval velkou komunitu, která kolem frameworku Laravel vzniká. V České republice tomu tak zatím ale není. Zde je stále, přihlédnutím i k počtu pracovních nabídek [55], nejpopulárnější framework Nette.

[20] seznam pro řízení přístupu a oprávnění k nějakému objektu

7 Tvorba API - praktická část

V této kapitole jsem se více zaměřil na praktickou ukázku. Popisuji zde tvorbu jednoduchého API pro práci s úkoly v ukázkové aplikaci. Vycházel jsem ze seriálu *Incremental APIs* uveřejněného na webu Laracasts [56]. Tato kapitola může také posloužit jako návod pro vytvoření API aplikace.

7.1 Technické požadavky na API

Před samotným vytvořením API si definujeme technické požadavky, které nám v budoucnu ulehčí případný další vývoj aplikace.

1. API bude mít URI prefix `api/v1`
2. nutnost autentizace uživatelů před prováděnými operacemi
3. výstup bude uživateli prezentován ve formátu JSON[21]
4. výstup bude doplněn o stavové kódy
5. výstup bude nezávislý na struktuře databáze
6. při vyšším počtu výsledků se použije stránkování

7.2 URI prefix

Jednotlivé části API budou volány různými URL adresami. Vždy ale bude adresa začínat prefixem `api/v1`. Tzn. například URL pro výpis všech úkolů z vybraného projektu bude vypadat následovně: `http://domena.cz/api/v1/projects/{id-projektu}/tasks`. Tohoto požadavku docílíme nastavením routeru následujícím způsobem:

```
1 // app/routes.php
2 Route::group(['prefix' => 'api/v1'], function ()
3 {
4   Route::get('projects/{projects}/tasks', 'ApiTasksController@show')
        ;
5 });
```

7.3 Autentizace uživatelů

Důležitým požadavkem na API je nutnost autentizace uživatelů před vykonáním volané operace. Pokud uživatel nebude mít oprávnění k dané operaci, tak dojde k jejímu ukončení. Zde lze pro jednoduchost využít HTTP Basic Authentication filtr.

[21] JavaScript Object Notation

Poznámka: Je nutné si ale pamatovat, že při tomto způsobu autentizace je jméno a heslo přenášeno nešifrovaně a může tak být lehce odposlechnuto. Z tohoto důvodu by měla být komunikace zabezpečena HTTPS[22] protokolem.

Filtr `auth.basic` můžeme aplikovat přímo ve výše uvedeném routeru.

```
1 // app/routes.php
2 Route::group(['prefix' => 'api/v1', 'before' => 'auth.basic'],
     function ()
3 {
4   Route::get('projects/{projects}/tasks', 'ApiTasksController@show')
       ;
5 });
```

Druhou možností je přidat `auth.basic` filtr do konstruktoru třídy `ApiTasksControlleru`.

```
1 // app/controllers/ApiTasksController.php
2 function __construct(TaskTransformer $taskTransformer)
3 {
4     $this->beforeFilter('auth.basic');
5 }
```

7.4 JSON

Pro výpis dat využijeme formát JSON. Druhou možností by bylo využít formát XML, ale JSON je stále populárnější při práci s daty v API aplikacích [57].

Pro prezentaci dat ve formátu JSON lze využít facade `Response` a její metoda `json`.

```
1 // app/controllers/ApiController.php
2 public function respond($data)
3 {
4     return Response::json($data, $this->getStatusCode());
5 }
```

Jako argumenty předáváme vypisovaná data a stavový kód.

7.5 Nezávislost API na struktuře databáze

Důležitým cílem API je jeho nezávislost na struktuře databáze. Např. pokud bychom v databázi, respektive tabulce `tasks`, změnili název sloupce `deadline` na `end`, tak by se to ve struktuře a výpisu API nemělo projevit. V opačném případě by hrozilo, že by aplikace využívající API přestaly fungovat. Byla by jim předána jiná položka, než očekávaly (např. `end` namísto `deadline`).

Tento problém vyřešíme vytvořením třídy `Transformer`, která bude data z databáze měnit do struktury API.

```
1 // app/transformers/Transformer.php
2 abstract class Transformer {
```

[22] Hypertext Transfer Protocol Secure

```
3
4   public function transformCollection(array $items)
5   {
6     return array_map([$this, 'transform'], $items);
7   }
8
9   public abstract function transform($item);
10
11 }
12
13 // app/transformers/TaskTransformer.php
14 class TaskTransformer extends Transformer {
15
16   public function transform($task)
17   {
18     return [
19       'title'    => $task['title'],
20       'note'     => $task['note'],
21       'deadline' => $task['deadline'],
22       'completed' => $task['completed'],
23       'amount'   => $task['amount']
24     ];
25   }
26 }
```

7.6 Stránkování

Jelikož výsledků při některých voláních může být velké množství, je nepraktické je ihned všechny vypisovat. Lepším řešením bude využít stránkování. Laravel disponuje metodou `paginate`, které předáváme jako argument počet vypisovaných položek.

```
1
2 // app/controllers/ApiTasksController.php
3
4 $tasks = $project->tasks()->paginate($limit);
```

JSON výpis úkolů z projektu má při využití stránkování následující strukturu:

```
1 {"data":[
2    {"title":"Nakoupit na sobotu",
3     "note":null,
4     "deadline":"19.04.2015",
5     "completed":false,
6     "amount":null},
7    {"title":"Objednat pojisteni",
8     "note":null,
9     "deadline":"27.04.2015",
10    "completed":false,
11    "amount":null},
12   {"title":"Zajit na kontrolu",
13    "note":null,
14    "deadline":"28.04.2015",
```

```
15      "completed":false,
16      "amount":null},
17     {"title":"Zajet do servisu",
18      "note":null,
19      "deadline":"21.04.2015",
20      "completed":false,
21      "amount":null}
22     ],
23 "paginator":{
24     "total_count":4,
25     "current_page":1,
26     "total_pages":1,
27     "limit":10
28     }
29 }
```

7.7 Jednotkové testování

Vytvoříme si test, který bude testovat, zda po zavolání URL pro výpis úkolů z projektu bude výsledek obsahovat JSON odpověď s očekávanými položkami (`title`, `note`, `deadline`, `completed` a `amount`). Dále také test pro kontrolu, zda při zadání neexistujícího projektu je výsledkem chyba 404.

Poznámka: Pro testování nastavíme jinou databázi, jelikož nechceme ohrozit databázi produkční. Konkrétně použijeme SQLite[23] v paměti. Toto nastavení lze jednoduše provést v souboru `app/config/testing/database.php`.

```
1  // app/config/testing/database.php
2  'default' => 'sqlite',
3
4    'connections' => array(
5
6      'sqlite' => array(
7        'driver' => 'sqlite',
8        'database' => ':memory:',
9        'prefix' => '',
10     ),
11 )
12
13
14 // app/tests/ApiTasksTest.php
15 class ApiTasksTest extends TestCase {
16
17   public function setUp()
18   {
19       parent::setUp();
20       Artisan::call('migrate');
21   }
22
23   /** @test */
```

[23] relační databázový systém

```
24  public function it_fetches_api_tasks()
25  {
26
27    $project = $this->makeProject();
28
29    $this->makeTask($project);
30
31    $task = $this->getJson('api/v1/projects/'.$project->public_id.'/
          tasks')->data[0];
32
33    $this->assertResponseOk();
34
35    $this->assertObjectHasAttribute('title',$task);
36    $this->assertObjectHasAttribute('note',$task);
37    $this->assertObjectHasAttribute('deadline',$task);
38    $this->assertObjectHasAttribute('completed',$task);
39    $this->assertObjectHasAttribute('amount',$task);
40
41  }
42
43  /** @test */
44  public function it_404s_if_a_project_is_not_found()
45  {
46      $this->getJson('api/v1/projects/xxx/tasks');
47      $this->assertResponseStatus(404);
48  }
49
50  private function makeTask($project)
51  {
52      $task = new Task();
53      $task->public_id = 'kdsjhdf876869guzgqduzgFxv';
54      $task->title = 'Testing task';
55      $task->note = 'Note ...';
56      $task->deadline = '2015-12-31';
57      $task->completed = 0;
58      $task->amount = null;
59      $task->user_id = 1;
60      $task->project_id = 1;
61      $task->save();
62  }
63
64
65  private function makeProject()
66  {
67      $project = new Project();
68      $project->public_id = 'jlnsdldjnsk76sjdhkbcGSSlkj';
69      $project->title = 'Testing project';
70      $project->color = '#000000';
71      $project->user_id = 1;
72      $project->save();
73      return $project;
74  }
75
```

```
76    public function getJson($uri)
77    {
78        return json_decode($this->call('GET', $uri)->getContent());
79    }
80
81 }
```

Závěr

Práce postupně popisuje hlavní části PHP frameworku Laravel. Začíná obecnými informacemi, následně představuje nástroje pro práci s routováním, databází, šablonovacím systémem, autentizací uživatelů, atd. Dále se zaměřuje na bezpečnost a jednotkové testování webových aplikací. Pro srovnání využívá frameworky Nette a Symfony.

Mezi silné stránky frameworku Laravel patří elegantní syntax, aktuální dokumentace, velká komunita uživatelů a učebních materiálů. Dále také moderní nástroje a postupy pro vývoj webových aplikací, mj. kvalitní ORM, systém databázových migrací, příkazový řádek Artisan, Composer pro správu závislostí.

Naopak mezi slabé stránky frameworku Laravel patří ochrana před útoky CSRF a XSS, přesněji Laravel obsahuje nástroje, které před těmito útoky chrání, ale neaplikuje je automaticky a spoléhá na programátora, že je nezapomene použít. Taktéž Laravel je poměrně mladý framework a velice rychle se vyvíjí. S tím je spojená určitá ne 100% kompatibilita verzí, proto přechod aplikace na vyšší verzi nemusí být vždy úplně jednoduchý. Dále v České republice není framework Laravel zatím příliš populární, z tohoto pohledu stále převažuje framework Nette. Proto i většina pracovních nabídek preferuje znalost právě frameworku Nette.

Z mého pohledu Laravel vnímám jako moderní framework, který se v současné době velice rychle vyvíjí dobrým směrem a práce s ním je skutečně zábavná, čímž v podstatě potvrzuji úvodní slova práce, kterými Taylor Otwell framework Laravel představuje. Jistě bude zajímavé sledovat jeho budoucí vývoj.

Conclusions

This bachelor thesis describes the main parts of the PHP framework Laravel. It begins with some general information, then presents tools for working with routing, databases, templating system, authenticating users, etc. It also focuses on security and unit testing of web applications. For comparison uses frameworks Nette and Symfony.

Among the strengths of the framework Laravel include an elegant syntax, quality documentation, a huge community of users and learning materials. As well as modern tools and techniques for developing web applications - good ORM, database migrations, command line interface Artisan, Composer for managing dependencies.

The weaknesses of the framework Laravel include protection against CSRF and XSS attacks, more Laravel contains tools that protect against these attacks, but it does not apply automatically, and relies on a programmer to remember to use it. Also Laravel is a relatively young framework and rapidly evolving. This is associated with certain not 100% compatible versions, so upgrade to a higher version of a application may not always be quite simple. Furthermore, Laravel framework isn't too popular in the Czech Republic, from this perspective still prevails Nette framework. Therefore the majority of job offers prefers knowledge about Nette.

In my opinion Laravel is a modern framework, which is currently at rapidly developing in the right direction and work with it is really fun, so I basically accepted the introductory words of this work which present framework Laravel. It will be interesting to follow the future development of Laravel framework.

A Příloha - Schema Builder a typy sloupců

Tabulka 29: Schema Builder - typy sloupců

Příkaz	Datový typ
`$table->bigIncrements('id');`	BIGINT, AUTO_INCREMENT
`$table->bigInteger('votes');`	BIGINT
`$table->binary('data');`	BLOB
`$table->boolean('confirmed');`	TINYINT(1)
`$table->char('name', 4);`	CHAR
`$table->date('created_at');`	DATE
`$table->dateTime('created_at');`	DATETIME
`$table->decimal('amount', 5, 2);`	DECIMAL
`$table->double('column', 15, 8);`	DOUBLE
`$table->enum('choices', array('foo', 'bar'));`	ENUM
`$table->float('amount');`	FLOAT
`$table->increments('id');`	INT, AUTO_INCREMENT
`$table->integer('votes');`	INT
`$table->longText('description');`	LONGTEXT
`$table->mediumInteger('numbers');`	MEDIUMINT
`$table->mediumText('description');`	MEDIUMTEXT
`$table->morphs('taggable');`	Přidá INT taggable_id a STRING taggable_type
`$table->nullableTimestamps();`	Stejné jako timestamps(), ale povoluje hodnotu null
`$table->smallInteger('votes');`	SMALLINT
`$table->tinyInteger('numbers');`	TINYINT
`$table->softDeletes();`	Přidá sloupec deleted_at pro obnovitelné odstranění[24]
`$table->string('email');`	VARCHAR
`$table->text('description');`	TEXT
`$table->time('sunrise');`	TIME
`$table->timestamp('added_on');`	TIMESTAMP
`$table->timestamps();`	Přidá sloupec created_at a updated_at
`$table->rememberToken();`	Přidá sloupec remember_token (VARCHAR(100) NULL)
`->nullable()`	Povolení hodnoty null ve sloupci
`->default($value)`	Výchozí hodnota sloupce
`->unsigned()`	Číselným typům přiřazuje atribut UNSIGNED

[24] Laravel umí využívat tzv. obnovitelné odstranění (anglicky soft deleting), kdy záznam v databázi není smazán, ale jen vybranému sloupci (`deleted_at`) je nastaven příznak o odstranění

B Obsah přiloženého CD

`src/`
: Kompletní adresářová struktura a zdrojové kódy ukázkové aplikace (v ZIP archivu) pro zkopírování na webový server.

`doc/`
: Text práce ve formátu PDF, vytvořený s použitím závazného stylu KI PřF UP v Olomouci pro závěrečné práce, včetně všech příloh, a všechny soubory potřebné pro bezproblémové vygenerování PDF dokumentu textu (v ZIP archivu), tj. zdrojový text textu, vložené obrázky, apod.

`readme.txt`
: Instrukce pro nasazení ukázkové aplikace na webový server, včetně požadavků pro její bezproblémový provoz.

Literatura

[1] CROFT, Jeff. *Frameworks for Designers* [online]. 2007 [cit. 2015-03-14]. Dostupný z: ⟨http://alistapart.com/article/frameworksfordesigners⟩.

[2] SKVORC, Bruno. *Best PHP Frameworks for 2014* [online]. 2013 [cit. 2014-11-18]. Dostupný z: ⟨http://www.sitepoint.com/best-php-frameworks-2014/⟩.

[3] SMITH, Grace. *13 PHP Frameworks to Help Build Agile Applications* [online]. 2014 [cit. 2014-11-18]. Dostupný z: ⟨http://mashable.com/2014/04/04/php-frameworks-build-applications/⟩.

[4] OTWELL, Taylor. *Oficiální dokumentace frameworku Laravel* [online]. [cit. 2015-03-14]. Dostupný z: ⟨http://laravel.com/docs⟩.

[5] WAY, Jeffrey. *Webový portál Laracasts* [online]. [cit. 2015-03-14]. Dostupný z: ⟨https://laracasts.com⟩.

[6] *Nette Foundation.* [online]. [cit. 2015-03-14]. Dostupný z: ⟨http://nettefoundation.com⟩.

[7] *Oficiální webové stránky agentury Sensio.* [online]. [cit. 2015-03-14]. Dostupný z: ⟨http://www.sensio.com⟩.

[8] *Twitter účet Taylora Otwella.* [online]. [cit. 2015-03-14]. Dostupný z: ⟨https://twitter.com/taylorotwell⟩.

[9] OTWELL, Taylor. *On Laravel's Future: Part 2* [online]. 2014 [cit. 2014-11-18]. Dostupný z: ⟨http://blog.laravel.com/on-laravels-future-part-2⟩.

[10] *Laravel Introduction.* [online]. [cit. 2014-11-20]. Dostupný z: ⟨http://laravel.com/docs/4.2/introduction⟩.

[11] SURGUY, Maks. *History of Laravel PHP framework, Eloquence emerging* [online]. 2013 [cit. 2014-11-18]. Dostupný z: ⟨http://maxoffsky.com/code-blog/history-of-laravel-php-framework-eloquence-emerging/⟩.

[12] *MIT Licence.* [online]. [cit. 2015-03-12]. Dostupný z: ⟨http://opensource.org/licenses/mit-license.html⟩.

[13] *Licenční politika.* [online]. [cit. 2015-03-12]. Dostupný z: ⟨http://nette.org/cs/license⟩.

[14] *Autoloading Standard.* [online]. [cit. 2015-03-14]. Dostupný z: ⟨http://www.php-fig.org/psr/psr-0/⟩.

[15] *Basic Coding Standard.* [online]. [cit. 2015-03-14]. Dostupný z: ⟨http://www.php-fig.org/psr/psr-1/⟩.

[16] *Artistic Style 2.05.* [online]. [cit. 2015-03-14]. Dostupný z: ⟨http://astyle.sourceforge.net/astyle.html#_style=allman⟩.

[17] GRUDL, David. *Proč Nette nedodržuje standardy PHP-FIG / PSR?* [online]. 2014 [cit. 2015-03-14]. Dostupný z: ⟨http://phpfashion.com/proc-nette-nedodrzuje-standardy-php-fig-psr⟩.

[18] *Požadavky Nette Framework*. [online]. [cit. 2015-03-14]. Dostupný z: ⟨http://doc.nette.org/cs/2.2/requirements⟩.

[19] *MVC aplikace & presentery*. [online]. [cit. 2015-03-23]. Dostupný z: ⟨http://doc.nette.org/cs/2.2/presenters⟩.

[20] REENSKAUG, Trygve. *Models-Views-Controllers* [online]. 1979 [cit. 2014-11-18]. Dostupný z: ⟨https://heim.ifi.uio.no/~trygver/2007/MVC_Originals.pdf⟩.

[21] GRUDL, David. *Nette Framework: MVC & MVP* [online]. 2009 [cit. 2015-03-14]. Dostupný z: ⟨http://www.zdrojak.cz/clanky/nette-framework-mvc-mvp/⟩.

[22] REES, Dayle. *Laravel: Code Bright*. Vancouver: Leanpub, 2014. 449 s.

[23] GRUDL, David. *Routování URL* [online]. [cit. 2015-03-22]. Dostupný z: ⟨http://doc.nette.org/cs/2.2/routing⟩.

[24] *Laravel Templates*. [online]. [cit. 2015-03-22]. Dostupný z: ⟨http://laravel.com/docs/4.2/templates⟩.

[25] VRÁNA, Jakub. *Ukládání hesel bezpečně* [online]. 2013 [cit. 2015-03-14]. Dostupný z: ⟨http://php.vrana.cz/ukladani-hesel-bezpecne.php⟩.

[26] *Symfony Security*. [online]. [cit. 2015-03-12]. Dostupný z: ⟨http://symfony.com/doc/current/book/security.html#roles⟩.

[27] REES, Dayle. *Laravel: Code Bright*. 2014. 449 s.

[28] VRÁNA, Jakub. *Cross-Site Request Forgery* [online]. 2006 [cit. 2015-03-14]. Dostupný z: ⟨http://php.vrana.cz/cross-site-request-forgery.php⟩.

[29] GRUDL, David. *Zabezpečení před zranitelnostmi* [online]. [cit. 2015-03-14]. Dostupný z: ⟨http://doc.nette.org/cs/2.2/vulnerability-protection⟩.

[30] VRÁNA, Jakub. *Cross Site Scripting* [online]. 2005 [cit. 2015-03-14]. Dostupný z: ⟨http://php.vrana.cz/cross-site-scripting.php⟩.

[31] TICHÝ, Jan. *Aplikační data čistá jako lilie* [online]. 2008 [cit. 2015-03-18]. Dostupný z: ⟨http://www.phpguru.cz/clanky/aplikacni-data⟩.

[32] *Nette: Context-Aware Escaping*. [online]. [cit. 2015-04-12]. Dostupný z: ⟨http://doc.nette.org/cs/2.3/templating#toc-context-aware-escaping⟩.

[33] VRÁNA, Jakub. *Obrana proti SQL Injection* [online]. 2005 [cit. 2015-03-14]. Dostupný z: ⟨http://php.vrana.cz/obrana-proti-sql-injection.php⟩.

[34] VRÁNA, Jakub. *1001 Tipů a triků pro PHP*. Praha: Computer Press, 2012. 456 s. ISBN 978-80-251-2940-1.

[35] VRÁNA, Jakub. *Vývoj obrany proti Session Hijackingu* [online]. 2013 [cit. 2015-02-14]. Dostupný z: ⟨http://php.vrana.cz/vyvoj-obrany-proti-session-hijackingu.php⟩.

[36] *PHP Session Fixation / Hijacking*. [online]. [cit. 2015-01-13]. Dostupný z: ⟨http://stackoverflow.com/questions/5081025/php-session-fixation-hijacking⟩.

[37] WAY, Jeffrey. *Laravel Testing Decoded*. Vancouver: Leanpub, 2013. 257 s.

[38] ZAMRZLA, Josef. *Testování v PHP: tvorba testovatelného kódu* [online]. 2013 [cit. 2015-03-14]. Dostupný z: ⟨http://www.zdrojak.cz/clanky/testovani-v-php-tvorba-testovatelneho-kodu/⟩.

[39] *Unit testing*. [online]. [cit. 2015-03-22]. Dostupný z: ⟨http://cs.wikipedia.org/wiki/Unit_testing⟩.

[40] *Testing*. [online]. [cit. 2015-03-22]. Dostupný z: ⟨http://silex.sensiolabs.org/doc/testing.html⟩.

[41] WAY, Jeffrey. *Laravel Testing Decoded*. Vancouver: Leanpub, 2013. 257 s.

[42] *Appendix A. Assertions*. [online]. [cit. 2015-03-22]. Dostupný z: ⟨https://phpunit.de/manual/current/en/appendixes.assertions.html⟩.

[43] *Laravel Testing*. [online]. [cit. 2015-03-22]. Dostupný z: ⟨http://laravel.com/docs/4.2/testing⟩.

[44] WAY, Jeffrey. *Why Laravel is Taking the PHP Community by Storm* [online]. [cit. 2015-03-22]. Dostupný z: ⟨http://code.tutsplus.com/tutorials/why-laravel-is-taking-the-php-community-by-storm–pre-52639⟩.

[45] *Github*. [online]. [cit. 2015-03-22]. Dostupný z: ⟨http://bit.ly/githubphp⟩.

[46] *Leanpub*. [online]. [cit. 2015-03-22]. Dostupný z: ⟨https://leanpub.com⟩.

[47] *Laracasts Stats*. [online]. [cit. 2015-03-22]. Dostupný z: ⟨https://laracasts.com/stats⟩.

[48] *Laracon*. [online]. [cit. 2015-03-22]. Dostupný z: ⟨http://laracon.us⟩.

[49] OTWELL, Taylor. *Twitter* [online]. 2015 [cit. 2015-03-22]. Dostupný z: ⟨https://twitter.com/taylorotwell/status/564245808925528064⟩.

[50] *Creating and Using Templates*. [online]. [cit. 2015-03-22]. Dostupný z: ⟨http://symfony.com/doc/current/book/templating.html#output-escaping⟩.

[51] *Using CSRF Protection in the Login Form*. [online]. [cit. 2015-03-22]. Dostupný z: ⟨http://symfony.com/doc/current/cookbook/security/csrf_in_login_form.html⟩.

[52] *Laravel Security*. [online]. [cit. 2015-03-22]. Dostupný z: ⟨http://laravel.com/docs/4.2/security⟩.

[53] *Upgrade Guide*. [online]. [cit. 2015-03-22]. Dostupný z: ⟨http://laravel.com/docs/4.2/upgrade⟩.

[54] *Entrust - Role-based Permissions for Laravel 4*. [online]. [cit. 2015-03-22]. Dostupný z: ⟨https://github.com/Zizaco/entrust⟩.

[55] *Nabídky práce ve startupech - PHP programátor*. [online]. [cit. 2015-03-22]. Dostupný z: ⟨https://www.startupjobs.cz/nabidky/2/php-programmer⟩.

[56] WAY, Jeffrey. *Incremental APIs* [online]. 2014 [cit. 2015-01-18]. Dostupný z: ⟨https://laracasts.com/series/incremental-api-development⟩.

[57] WYSE, Josh. *Why JSON Is Better Than XML* [online]. 2014 [cit. 2015-01-23]. Dostupný z: ⟨http://cloud-elements.com/json-better-xml/⟩.

[58] WAY, Jeffrey. *Laravel Testing Decoded*. Vancouver: Leanpub, 2013. 257 s.