Univerzita Pardubice
Fakulta elektrotechniky a informatiky

Využití návrhových vzorů při vývoji webové aplikace v PHP

Bc. Richard Fíla

Diplomová práce

2019

Univerzita Pardubice
Fakulta elektrotechniky a informatiky
Akademický rok: **2017/2018**

ZADÁNÍ DIPLOMOVÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení:	**Bc. Richard Fíla**
Osobní číslo:	**I16212**
Studijní program:	**N2646 Informační technologie**
Studijní obor:	**Informační technologie**
Název tématu:	**Využití návrhových vzorů při vývoji webové aplikace v PHP**
Zadávající katedra:	**Katedra softwarových technologií**

Zásady pro vypracování:

Cílem diplomové práce je vytvořit mikro framework, který by sloužil k rychlému vývoji webových aplikací v jazyce PHP a který by byl postaven na aktuálních návrhových vzorech (Front-Controller, Observer, Dependency Injection, MVC, ?).
V úvodní části DP bude proveden úvod do problematiky návrhových vzorů. Dále bude provedena analýza současných PHP frameworků s důrazem kladeným na rozbor implementací návrhových vzorů v rámci frameworků. V práci bude provedena kritická komparace analyzovaných frameworků.
V aplikační části práce bude proveden návrh architektury frameworku a její vlastní implementace v jazyce PHP.

Rozsah grafických prací: **10**

Rozsah pracovní zprávy: **60**

Forma zpracování diplomové práce: **tištěná**

Seznam odborné literatury:

Erich Gamma, Richard Helm, Ralph Johnson, John M. Vlissides, Design Patterns: Elements of Reusable Object-Oriented Software,Addison-Wesley, 1994, ISBN: 078-5342633610.
Martin Fowler, Patterns of Enterprise Application Architecture, Addison-Wesley Professional, 2002, ISBN: 978-0321127426
Gutmans, Andi, Rethans, Derick a Bakken, Stig. 2007. Mistrovství v PHP 5. Praha : Computer press, 2007. ISBN: 978-80-251-1519-0.

Vedoucí diplomové práce: **Ing. Lukáš Čegan, Ph.D.**
Katedra informačních technologií

Datum zadání diplomové práce: **30. října 2017**

Termín odevzdání diplomové práce: **18. května 2018**

Ing. Zdeněk Němec, Ph.D.
děkan

L.S.

prof. Ing. Antonín Kavička, Ph.D.
vedoucí katedry

V Pardubicích dne 15. listopadu 2017

Prohlašuji:

Tuto práci jsem vypracoval samostatně. Veškeré literární prameny a informace, které jsem v práci využil, jsou uvedeny v seznamu použité literatury.

Byl jsem seznámen s tím, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorský zákon, zejména se skutečností, že Univerzita Pardubice má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 autorského zákona, a s tím, že pokud dojde k užití této práce mnou nebo bude poskytnuta licence o užití jinému subjektu, je Univerzita Pardubice oprávněna ode mne požadovat přiměřený příspěvek na úhradu nákladů, které na vytvoření díla vynaložila, a to podle okolností až do jejich skutečné výše.

Beru na vědomí, že v souladu s § 47b zákona č. 111/1998 Sb. o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších předpisů a směrnicí Univerzity Pardubice č. 9/2012, bude práce zveřejněna v Univerzitní knihovně a prostřednictvím Digitální knihovny Univerzity Pardubice.

V Pardubicích dne 22. 8. 2019 Bc. Richard Fíla

PODĚKOVÁNÍ

Děkuji vedoucímu, panu Ing. Lukáši Čeganovi, Ph.D. za kvalitní vedení, cenné rady a čas, který mi věnoval při tvorbě této diplomové práce. Dále bych rád poděkoval své rodině, blízkým přátelům a své ženě za podporu při studiu.

ANOTACE

V této diplomové práci je prezentována historie a současnost programovacího jazyka PHP. Dále jsou zde ukázány nejznámější PHP frameworky a mikro frameworky. Následně je v této diplomové práci obsaženo srovnání těchto frameworků jak mezi sebou, tak i mezi frameworky jiných programovacích jazyků. Další část se zabývá tématikou návrhových vzorů, týkajících se převážně tvroby webových stránek. Na závěr této diplomové práce je popsán podrobný postup při tvorbě vlastního mikro frameworku.

KLÍČOVÁ SLOVA

Návrhové vzory, MVC, Front Controller, Event Dispatcher, Template View, PHP, framework, mikro framework

TITLE

Using Design Patterns to Develop a Web Application in PHP

ANNOTATION

This thesis deals with history and present of programming language PHP. Furthermore, it shows the most famous PHP frameworks and micro frameworks. Comparsion of these frameworks among themselves and between frameworks of other programming languages is included in the thesis. The next part deals with the theme of design patterns. These patterns are mainly related to website creation. At the end of this thesis is described detailed procedure for creating your own micro framework.

KEYWORDS

Design Patterns, MVC, Front Controller, Event Dispatcher, Template View, PHP, Framework, Micro Framework

OBSAH

SEZNAM OBRÁZKŮ

SEZNAM TABULEK

SEZNAM ZKRATEK

ACL	Access Control List
API	Application Programming Interface
ASP.NET	Active Server Pages
BOM	Byte Order Mark
CLI	Command Line Interface
CSS	Cascading Style Sheets
DB	Database
DDD	Domain-Driven Design
DI	Dependency Injection
DOM	Document Object Model
EL	Expression Language
FSM	Finite State Machine
GUI	Graphical User Interface
HHVM	HipHop Virtual Machine
HMVC	Hiearchical Model View Controller
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
ICU	International Components for Unicode
JSON	JavaScript Object Notation
LDAP	Lightweight Directory Access Protocol
LTS	Long Term Support
MIME	Multipurpose Internet Mail Extensions
MIT	Massachusetts Institute of Technology
MMF	MVC Micro Framework
MVC	Model View Controller
MVVM	Model View ViewModel
NV	Návrhový vzor
ODM	Object Document Mapping
OOP	Objektově orientované programování
ORM	Objektově relační mapování
PDF	Portable Document Format
PHP	PHP (Personal Home Page): Hypertext Preprocessor
PHP/FI	PHP/Forms Interpreter
PHP FIG	PHP Framework Interop Group
PHP GTK	PHP Gimp Tool Kit
PHQL	Phalcon Query Language
PSR	PHP Standard Recommendation
REST	Representational State Transfer
RSS	RDF (Resource Description Framework) Site Summary
SQL	Standardized Query Language
UI	User Interface
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
VCS	Version Control System
WYSIWYG	What You See Is What You Get
XAMPP	Cross-Platform Apache, MariaDB (MySQL), PHP and Pearl
XML-RPC	eXtensible Markup Language – Remote Procedure Call
YAML	YAMP Ain´t Markup Language

ÚVOD

Hlavním cílem této diplomové práce je vytvořit vlastní mikro framework, který bude vývojářům usnadňovat tvorbu webových aplikací. Úvodní část se věnuje programovacímu jazyku PHP, jeho historii a aktuálnímu stavu.

Dále se tato diplomová práce zaměřuje na to, jak nám využití frameworků a mikro frameworků může jednoduše usnadnit práci. V této části je jejich stručné srovnání a je zde soupis návrhových vzorů, které využívají.

Následně je zde ukázáno, co vůbec jsou návrhové vzory a jak je můžeme sami využívat. Vysvětlíme si ty nejdůležitější, zejména návrhové vzory pro tvorbu webových stránek a podíváme se na jejich praktické využití. Je zde ukázáno ale i pár příkladů, kde se z oblíbených vzorů stanou zastaralé a nebezpečné programovací praktiky.

Poslední část této práce je zaměřena na vývoj vlastní aplikace. Bude se jednat o mikro framework, který umožní uživateli používat návrhový vzor MVC, Event Dispatcher, směrování atd. Na tomto jádru se poté postaví skeleton aplikace, který, stejně jako framework, bude poskytnut veřejně přes balíčkovací nástroj Composer. Jádro našeho frameworku obohatíme o balíčky třetích stran, které rozšíří jeho funkcionalitu. V závěru této práce se tedy pokusíme sjednotit nejnovější praktiky ve vývoji v PHP, využívání samostatných balíčků s návrhovými vzory, z nichž některé se využívají již desítky let.

1 PHP

Tato kapitola je jednoduchým úvodem do programovacího jazyka PHP. Nejprve se zabývá jeho stručnou charakteristikou a jeho historií. Dále jsou v této kapitole popsány poslední verze PHP s jejími nejnovějšími vlastnostmi. V závěru je seznámení s doporučenými standardy psaní kódu v PHP.

1.1 Charakteristika PHP

PHP je skriptovací programovací jazyk s podporou objektově orientovaného programování, určený pro vývoj webových aplikací. Oproti například JavaScriptu, PHP se využívá pro skriptování na straně serveru. Z tohoto důvodu není ve zdrojovém kódu webové stránky vidět žádný PHP kód. Prohlížeč odešle žádost na server a ten po vyhodnocení odešle adekvátní odpověď.

PHP stránky mají HTML strukturu, ale jsou obohacené o PHP kód, který je obalen speciálními tagy <?php a ?>. Mimo tvorby webových aplikací je například možné vytvářet skripty pro příkazový řádek (s využitím PHP parseru) nebo desktopovou aplikaci (s využitím PHP GTK). PHP nicméně není nejvhodnějším jazykem pro tvorbu desktopových aplikací (zejména ne s GUI). Pokud ho ale někdo velice dobře ovládá a nechce se učit nový programovací jazyk, může využít výhod PHP i v aplikacích na straně klienta.

Tento skriptovací jazyk je velmi doporučován pro začátečníky, jelikož už po krátké chvíli je uživatel schopný psát jednoduché kódy. PHP dále nabízí i spoustu prvků pro pokročilé a profesionální programátory. Obzvláště při využití některého z frameworků, nebo dokonce kombinací jejich komponent. [1]

1.2 Historie

PHP je programovací jazyk, vytvořený dánsko-kanadským programátorem Rasmusem Lerdorfem. První verze byla napsána v jazyce C a měla sloužit pro sledování návštěv na jeho osobních stránkách. Původní název byl “Personal Home Page Tools“, zkráceně PHP Tools. V současné době se význam zkratky vysvětluje jako PHP: Hypertext Preprocessor.

Časem se Rasmus rozhodl předělat PHP Tools na PHP/FI (Form Interpreter). Tuto novou verzi zpřístupnil veřejnosti v roce 1995. Umožňovala například využití databáze a obsahovala framework pro psaní jednoduchých webů.

Dalším důležitým milníkem v historii tohoto jazyka bylo vydání PHP verze 3. Zatímco do této doby pracoval Rasmus převážně sám, s vydáním této nové verze mu pomáhalo mnoho dobrovolníků, a to jak s psaním samotného kódu, tak i s dalšími aspekty souvisejícími s vydáním. Zejména dva izraelští programátoři Zeev Suraski a Andi Gutmans pomohli tím, že přepsali parsovací engine, který se stal základem pro PHP verzi 3.

V roce 2000 vyšla čtvrtá verze PHP. Suraski a Gutmans s dalšími vývojáři vydali další parser, tentokrát pojmenovaný jako Zend engine. Název Zend vznikl spojením křestních jmen obou vývojářů. V roce 2004 vyšlo PHP 5, které běželo převážně na Zend Engine 2.0.

3. prosince 2015 vyšla zatím poslední, sedmá verze PHP. Číslo 6 se přeskočilo z důvodu toho, že šestá verze byla používaná jako prototyp. Vývojáři se tímto chtěli vyhnout zaměňování těchto dvou verzí. S posledním dnem roku 2018 skončila podpora verze 5. Celých 14 let tedy byla hojně využívaná a velmi oblíbená. Aktuální číslo poslední verze je 7.3.7.

Podkapitola 1.2 byla zpracována za užití zdrojů [1][3][4][5].

1.3 PHP 7

V této podkapitole jsou ukázány nejnovější vlastnosti, se kterými PHP 7 v poslední verzi přišlo. Jedná se například o deklarování skalárních proměnných. To je možné ve dvou variantách, donucovacím a striktním. Dále PHP zlepšilo typovou kontrolu i co se týče návratových hodnot. Pomocí příkazu define je nyní možné definovat konstantní pole.

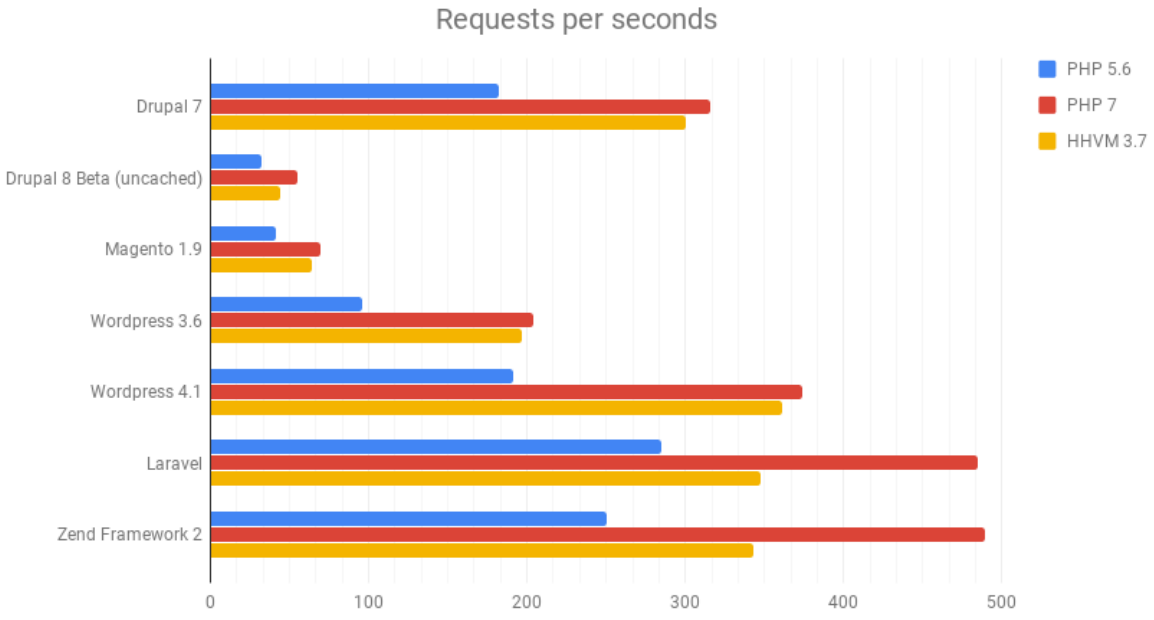
V nejnovějších verzích vývojáři přidali i dva nové operátory. Jedná se o operátor Null coalescing operator a takzvaný operátor vesmírné lodi. První operátor zlepšuje práci při kontrole objektů tím, že zohledňuje variantu hodnoty Null. Jeho syntaxe je podobná klasickému operátoru IF, pouze se dvěma otazníky místo jednoho. Spaceship operátor zase pomáhá porovnávat dvě rovnice. Podle poměru jejich výsledků vrátí klasické hodnoty -1, 0 a 1, ekvivalentní k menší než, rovno a větší než.

Jako další byla přidána i podpora anonymních tříd pomocí příkazu new class. Díky tomu je například možné vložit tento typ třídy jako vstupní parametr do metody, jako jsme zvyklí u vkládání funkcí. Dále se zlepšila možnost vytvoření dočasné vazby na objekt díky nové metodě Closure::call.

Nově přibyla také možnost filtrování unserialized tříd a větší podpora Unicode kódování. Rozšíření se také dočkala metoda assert, která je obohacena o funkcionalitu expectations. Dále

umožňuje PHP 7 skupinový import tříd pomocí deklarace use. Generátorům přibyla možnost používat klíčové slovo return pomocí metody getReturn a delegace generátoru pomocí výrazu yield from. Další nové funkce jsou intdiv, Session options, preg_replace_callback_array a CSPRNG.

Kromě nových funkcí se také rapidně zvýšila rychlost, jakou PHP disponuje. Na obrázku 1 je graf toho, jak si vede PHP 7 v porovnání s PHP 5,6, nebo HHVM 3,7 (HipHop Virtual Machine, což je rychlejší verze PHP 5 vytvořená firmou Facebook).



Obrázek 1 - Graf srovnání výkoností PHP 7 s předchozíma verzema [2]

Dále je vhodné zmínit, že v sedmé verzi je již dokončen proces přechodu k plně objektově orientovaným programovacím jazykům. Jakákoliv skepse vůči PHP je tedy naprosto zbytečná, neboť se teď může řadit mezi ostatní OOP jazyky jako je například Java nebo C#.

Podkapitola 1.3 byla zpracována za užití zdrojů [1][4][6].

1.4 PHP-FIG a PSR

Jako většina programovacích jazyků, i PHP disponuje vlastními standardy. Nejsou ovšem oficiální. Každopádně jsou využívány největšími PHP frameworky. Toto sdružení, pojmenované PHP Framework Interop Group, se rozhodlo vyřešit problém s rozdílným přístupem k psaní kódu v PHP.

Mezi největší představitele PHP-FIG patří: Zend, Yii framework, Phalcon, Drupal, CakePHP, Composer a další. V minulosti k nim patřili i například Symfony a Propel. Tyto velké firmy vždy zastupuje jeden hlavní představitel.

Při psaní malých, nezávislých aplikací, si vývojář může říct, že dodržování nějakých standardů, není to nejdůležitější. Ovšem ve chvíli, kdy se jedná o větší aplikaci, kde spolu komunikuje několik systémů, nástrojů třetích stran a využívá se kombinací několika frameworků, nastává odlišná situace. Díky využívání standardizace se kód zpřehlední a je lépe čitelný.

Každá část standardizace má číselné označení. V tuto chvíli jich je 19 a pouze 12 schválených. Jedná se například o základní kódové standardy, autoload standardy, standardy pro HTTP a další. Na oficiálních stránkách PHP-FIG je uvedená tabulka všech PSR. Její aktuální verze je znázorněna v tabulce 1.

Tabulka 1 - Tabulka všech PSR k datu 16.7. 2019 [7]

Číslo	Titul	Editoři	Status
0	Autoloading Standard	Matthew Weier O'Phinney	Zastaralé
1	Basic Coding Standard	Paul M. Jones	Schváleno
2	Coding Style Guide	Paul M. Jones	Schváleno
3	Logger Interface	Jordi Boggiano	Schváleno
4	Autoloading Standard	Paul M. Jones	Schváleno
5	PHPDoc Standard	Chuck Burgess	Navrhnuto
6	Caching Interface	Larry Garfield	Schváleno
7	HTTP Message Interface	Matthew Weier O'Phinney	Schváleno
8	Huggable Interface	Larry Garfield	Zamítnuto
9	Security Advisories	Michael Hess	Zamítnuto
10	Security Reporting Process	Michael Hess	Zamítnuto
11	Container Interface	Matthieu Napoli, David Négrier	Schváleno
12	Extended Coding Style Guide	Korvin Szanto	Posouzeno
13	Hypermedia Links	Larry Garfield	Schváleno
14	Event Dispatcher	Larry Garfield	Schváleno
15	HTTP Handlers	Woody Gilk	Schváleno
16	Simple Cache	Paul Dragoonis	Schváleno
17	HTTP Factories	Woody Gilk	Schváleno
18	HTTP Client	Tobias Nyholm	Schváleno
19	PHPDoc tags	Chuck Burgess	Navrhnuto

Každé PSR, má svůj status. Jsou zde obsaženy schválené, posuzované, teprve navržené, zamítnuté, nebo zastaralé standardy. Jednou za čas představitelé PHP-FIG hlasují o tom, které PSR, dostanou jaké statusy. Tyto standardy by neměly být využívány pouze frameworky a vývojáři aplikací, ale měly by najít podporu i ve vývojových prostředích. [7][8]

Existuje ale mnoho frameworků, které se těmito pravidly neřídí, nebo řídí, ale ne stoprocentně. Například Nette vydalo článek o tom, proč nedodržuje striktně tyto standardy. Jedná se o rozdíly mezi psaním čtyř mezer místo tabulátorů a podobně. PSR standard prosazuje psaní mezer nad tabulátory. Nicméně při hlasování celá jedna třetina PHP-FIG preferovala opak. U několika příkladů je dále ukázáno, že nějaké části PSR jsou dokonce v rozporu s použitou standardizací v oficiálním PHP manuálu. Sami zakladatelé PHP-FIG ale říkají, že se nesnaží vytvářet obecné standardy, ale pouze pravidla, u kterých by měl každý sám za sebe zvážit jejich využití. [9]

1.4.1 PSR-1 a PSR-2

Standardy PSR-1 a PSR-2 se zabývají základními postupy při psaní kódu. Jejich hlavním cílem je zpřehlednit zdrojové soubory s PHP kódem. V této části je ukázán pouze základní přehled těchto standardů. Podrobnější vysvětlení je na oficiálních stránkách PHP-FIG.
Základní pravidla PSR-1:

- Soubor MUSÍ používat pouze <?php a <?= tagy.
- Souboru MUSÍ používat pouze UTF-8 kódování bez BOM.
- Soubor BY MĚL buď deklarovat symboly (třídy, funkce, constanty atd.), nebo způsobovat vedlejší účinky (např. generovat výstup, měnit .ini nastavení, atd.), ale nikdy BY NEMĚL dělat oboje.
- Jmenné prostory a třídy MUSÍ splňovat pravidla pro autoloading (PSR-0, PSR-4).
- Třídy MUSÍ být deklarované pomocí StudlyCaps (velké písmeno každého slova).
- Třídní konstanty MUSÍ být deklarovány se všemy písmeny velkými a podtržítkem jako oddělovačem.
- Metody musí být deklarovány pomocí camelCase (první písmeno malé a všechny ostatní první písmena slov velká).

[7][8]

Základní pravidla PSR-2:

- Kód MUSÍ splňovat PSR-1.
- Kódu MUSÍ používat 4 mezery pro odsazení, ne tabulátor.

- NESMÍ být nastaven tvrý limit (když se překročí, vyvolá chybu) délky řádku. Měkký limit (když se překročí, pouze informuje) MUSÍ být 120 znaků. Řádek by měl být kratší nebo roven 80 znakům.
- Po deklarování jmenného prostoru MUSÍ být prázdný řádek. Stejně tak i po deklaraci use.
- Otevírací a uzavírací složené závorky při deklaraci tříd a metod MUSÍ být na vlastním řádku.
- Viditelnost (modifikátor přístupu) MUSÍ být deklarována u všech vlastností a metod. Klíčová slova abstract a final MUSÍ být deklarována před viditelností a klíčové slovo static MUSÍ být deklarované za viditelností.
- Za klíčovým slovem řídící struktury (if, for, atd) MUSÍ být mezera. Za názvem metody a funkce být NESMÍ.
- Otevírací složené závorky pro řídící struktury MUSÍ být na stejném řádku a uzavíarcí složené závorky MUSÍ být na samostatném řádku.
- Kulaté závorky pro řídící strukturu NESMÍ mít mezeru před ani za.

[7][10][11]

1.4.2 **PSR-3**

V každé aplikaci by měl být nějaký systém logování, tedy způsob zaznamenávání toho, co se při běhu aplikace děje, a to ať už se jedná o skutečnosti informačního charakteru nebo významné zprávy o fatálních chybách.

Přesná definice toho, jak by měl správný logger fungovat, je popsána v PSR-3. Tento standard zavádí také několik úrovní zaznamenávaných informací podle jejich důležitosti, nebo podle základního charakteru. Tyto úrovně jsou Emergency, Alert, Critical, Error, Warning, Notice, Info a Debug. [7]

1.4.3 **PSR-4**

Standard PSR-4 se zabývá autoloadingem. Tato funkcionalita umožňuje automatické načítání potřebných tříd. Předchůdcem pro PSR-4, byl standard PSR-0, který se také zabýval automatickým načítáním tříd. Postupem času byl označena jako zastaralý a vývojáři by se měli jejímu využívání vyvarovat a přejít plně na PSR-4.

K využíti PSR-4 je zapotřebí utilita Composer, která je podrobněji prezentována v následujících kapitolách. Když všechny balíčky obsažené v projektu využívají PSR-4, je možné je využívat v aplikaci vedle sebe, bez obav vzniku konfliktů. Nejdůležitějším prvkem při využívání autoloaderu, je správné užití jmenných prostorů. PSR-4 přesně specifikuje, jakou by měl mít jmenný prostor strukturu:

\<Jmenný prostor>(\<Podsložka jmenného prostoru>)\<Název třídy>.

První položce se také říká vendor namespace. Další je podsložka jmenného prostoru, která je ovšem nepovinná. Může jich být ale naopak i několik. Jako poslední se uvádí název třídy. Ukázka užití jmenného prostoru je v praktické části této diplomové práce.

Na začátek aplikace poté stačí přidat příkaz pro načtení autoloderu a problém s načítáním stránek bude pro celou aplikaci vyřešený. Abychom nemuseli při každém využívání načítané třídy psát celý jmenný prostor před její název, je možné použít operátor use. Většina vývojových prostředí (jako například PhpStorm) doplňuje potřebné direktivy use sama. [7][12]

1.4.4 **PSR-14**

PSR-14 je dalším ze skupiny standardů od společnosti PHP-FIG. Jedná se o jejich vlastní implementaci návrhového vzoru Event Dispatcher (upravená verze Obserevru a Mediatoru). Skládá se z pěti hlavních aktérů:

- **Event** (událost) je zpráva, kterou odesílá Emmiter. Může to být jakýkoliv PHP objekt.
- **Listener** (posluchač) může být jakákoliv část PHP kódu, která se dá zavolat (např. metoda). Jeden Event může mít neomezený počet posluchačů, ale i žádného. Listener může dále vytvářet další asynchronní chování.
- **Emitter** (volající kód) je součást PHP kódu, ve které si přejeme, aby se událost spustila. Tento prvek se neimplementuje jako speciální součást Event Dispatcheru. V praxi by se to dalo ukázat třeba v příkladu s loggerem. Kdyby se odchytila část kódu pomocí try a catch, tak v catch části by se dala vyvolat HandleError událost. Ta by se postarala o zaznamenání informace o výjimce do logu. Část kódu catch, která volá naši událost je v tomto případě Emmiter.
- **Dispatcher** (dispečer) je nejdůležitější součást v PSR-14. Dispatcher dostane od Emmiteru aktivovanou událost a má na starosti, aby byla předána všem relevantním posluchačům. Ty mu ale musí předat Listener Provider.

- **Listener Provider** (poskytovatel posluchačů) je zodpovědný za vyhledání všech relevantních posluchačů a předává jejich seznam Dispatcheru. Sám ale nesmí nijak Listenery kontaktovat.

Dále PSR-14 poskytuje tři rozhraní, které je možné implementovat. Jedná se o EventDispatcherInterface, ListenerProviderInterface a StoppableEventInterface. Poslední zmíněné rozhraní definuje specifickou vlastnost události. Stoppable Event je druh události, která se dá zastavit (dá se zastavit její šíření). Tím je možné například zajistit, aby se po zavolání konkrétního posluchače událost již dále nešířila a nevolala další posluchače. Všechny rozhraní budou více popsány v kapitole o praktické části této diplomové práce. [7]

2 PHP FRAMEWORKY

Tato kapitola se věnuje nejčastějším frameworkům programovacího jazyka PHP a užitečnému nástroji zvaném Composer. Dále je zde ukázáno, jaké návrhové vzory jsou v těchto frameworcích obsaženy. V poslední části této kapitoly je znázorněn význam implementace návrhových vzorů přímo ve frameworcích.

2.1 Framework všeobecně

Softwarový framework je soubor nástrojů, který ulehčuje tvorbu aplikací. Užíváním frameworků se snižuje time-to-market a celkově zvýšuje produktivita vytvářené aplikace, a to převážně díky správnému designu a možnosti opakovaného použití stejného kódu.

Vybráním předem připravených modulů frameworku si také zajistíme lepší práci v kolektivu. Frameworky totiž často bývají velmi dobře dokumentované. Dále se omezí riziko vzniku chyb, jelikož je uživateli umožněno psát menší množství kódu. [14][15]

PHP framework Symfony uvádí na svých stránkách velice výstižnou definici frameworku. Když si představíme průběh vytváření aplikace jako horolezeckou stěnu, znamená vrchol úspěšné dokončení aplikace a proces lezení je její vývoj. Kdyby framework nebyl využit, bylo by to jako přijít k holé stěně a muset si celou cestu připravit sám. Hledání trasy, riziko vracení se zpět či například vkládání skob. Naproti tomu s frameworkem máme předem vytyčené možnosti kudy se vydat a připravené záchytné body. [13]

Softwarový framework se dá rozdělit na následující části:

- **Toolbox** je sada předdefinovaných komponent, která se dá využívat při vývoji a zajistit uživateli méně psaní kódu.
- **Metodologie** je návod, jak psát aplikace. Pomáhá sjednotit styl psaní kódu vývojářů a využívání best practices.

[14][15]

2.2 Composer

Composer je nástroj PHP, který nám umožňuje spravovat závislosti na jednotlivých knihovnách v PHP aplikacích. Spustit tento nástroj je možné na operačních systémech Linux, Unix, macOS a Windows a pro správný chod vyžaduje PHP verzi 5.3.2 a vyšší.

V projektu stačí vytvořit soubor composer.json, do kterého se budou vypisovat závislosti na balíčcích. Je možné využívat oficiální balíčky konkrétních frameworků, samostatných vývojářů či vývojářských skupin, anebo vlastní balíčky. Práce s composerem bude popsána důkladněji ve čtvrté kapitole.

Hlavní výhodou využívání composeru je, že nám umožňuje využívat pouze ty knihovny, které doopravdy potřebujeme. Není tedy potřeba instalovat celý Symfony framework, když z něj zamýšlíme využívat pouze BrowserKit (nástroj pro simulaci funkce prohlížeče). Díky tomu můžeme kombinovat vlastnosti několika frameworků. Komposer dále také umožňuje využívat jiná verze frameworku pro každý balíček zvlášť. Někdy dokonce vyjde nová verze balíčku dříve než celý framework.

Teď, když jsme si vysvětlili pojem Composer, je vhodné ukázat přístup frameworků k balíčkování. Pro tyto účely byly vybrány frameworky Zend a Symfony. V následujících dvou podkapitolách si ukážeme jejich rozdíly a podobnosti při implementaci důležitých komponent v balíčcích.

Podkapitola 2.2 byla zpracována pomocí zdrojů [16][17][18][19].

2.3 Zend

Zend framework je založen na PHP verzi 5.6 a je optimalizovaný pro PHP 7. Jak již název napovídá, je z velké části pod záštitou firmy Zend, která stála za vývojem PHP verze 3 a vyšších. V současné době je nejaktuálnější verze 3, která nabízí stoprocentně objektově orientovaný kód. Jako ostatní, i Zend využívá komponenty od jiných frameworků, nicméně sám nabízí spousty vlastních.
Nejdůležitější komponenty Zend frameworku:

- **Zend_Acl** je jednoduchý a flexibilní Access control list pro správu práv. Jasně definuje zdroje (objekt ke kterému kontrolujeme přístup) a roli objektu, který žádá o přístup ke zdroji.
- **Zend_Auth** je API, které nabízí možnost autentifikace (kontrola, jestli objekt je skutečně tím, za koho se vydává). Neplést s autorizací. Ta je poskytována výše zmíněným ACL.
- **Zend_Config** je modul, který zjednodušuje přístup a práci s konfigurací dat v rámci aplikace.

- **Zend_Controller** je základ MVC systému obsaženém v tomto frameworku. Součástí tohoto modulu je Zend_Front_Controller, který implementuje návrhový vzor Front Controller.
- **Zend_Db** je jednoduché SQL rozhraní pro práci s databází. Jeho podtřída Zend_Db_Adapter zprostředkovává propojení PHP aplikace s RDBMS. Pro každé RDBMS je jiný adaptér.
- **Zend_Debug** je debugovací nástroj používaný ve frameworku Zend. Hlavní metoda je dump, která vypisuje nebo navrací informace o prováděném příkazu.
- **Zend_Exceptions** je třída, která umožňuje používání výjimek v rámci Zend Frameworku.
- **Zend_Gdata** je API, které se stará o využívání služeb Google Data. Mezi ty patří například kalendář, tabulky, YouTube, blogger, notebook a další. Přístup k těmto službám je zprostředkován pomocí Atom Publishing Protocolu.
- **Zend_Layout** je umožňuje obalení obsahu aplikace v rámci jiného view. Tato třída implementuje Two Step View vzor, který je přiblížen ve 3. kapitole.
- **Zend_Loader** je třída, která napomáhá s dynamickým nahráváním souborů.
- **Zend_Locale** je modul, který řeší problémy s lokalizací aplikace.
- **Tend_Log** je komponenta jejíž hlavním úkolem je logování neboli monitorování toho, co se v aplikaci děje.
- **Zend_Search_Lucene** je engine pro textové vyhledávání, napsaný kompletně v PHP 5. Ukládá si vlastní indexy k filesystému a nevyžaduje databázový server.
- **Zend_Server** poskytuje podporu různých serverových tříd (REST server, JSON server atd.).
- **Zend_Session** pomáhá spravovat a uchovávat session data.

Dále Zend framwork nabízí následující možnosti:

- cashování,
- správy RSS feedů,
- tvorby formulářů,
- správy HTTP,
- překladu PHP to JSON a naopak,
- LDAP,
- e-mailových služeb (i s podporou MIME),
- práce s měřícími jednotkami,

- správy paměti,
- OpenId,
- PDF,
- ukládání hodnot do registrů,
- REST,
- validátoru vstupních hodnot,
- XML-RPC,
- a podporu služeb z třetí strany jako Akismet, Amazon, Audioscrobbler, Delicious, Flickr, Nirvanix, Simpy, SlideShare, StrikeIron, Technorati a Yahoo, Uri.

[21]

Výše zmíněné komponenty uvádějí využívání návrhových vzorů MVC, Front Controller a Two Step View. V celé implementaci frameworku ale využívají i další návrhové vzory, jako například: DDD, Singleton, Strategy, Adapter a Factory. Všechny budou podrobněji popsány v následující kapitole. [20][21]

2.4 Symfony

Symfony framework byl poprvé uveden v roce 2005 pod záštitou firmy SensioLabs. Ta vytvořila Symfony jako pomocníka při vývoji webových stránek pro jejich klienty. Současná stabilní verze je 4,2 a 3,4 LTS verze. V dnešní době patří Symfony mezi vedoucí PHP frameworky na trhu.

Nejdůležitější komponenty Symfony frameworku:

- **Asset** spravuje generování URL a verzování webových prvků jako jsou CSS, JavaScriptové soubory a obrázky.
- **BrowserKit** simuluje chování webového prohlížeče. Tato komponenta ovšem umožnuje práci s konkrétní aplikací, ve které je použita. V případě že potřebujeme otestovat funkčnost využívání externích stránek nám BrowserKit nepomůže.
- **Config** je pomocník při práci s konfiguračními soubory.
- **Console** umožňuje vytvářet vlastní CLI příkazy.
- **CssSelector** je komponenta, díky které se dá převádět CSS selektory na jejich XPath ekvivalenty.
- **Debug** stejně jako v Zend Frameworku, i v Symfony nabízí možnost ladění chyb. Využívá pro to třídy ErrorHandler a ExceptionHandler.

- **EventDispatcher** umožňuje komunikaci mezi komponentama jedné aplikace. Využívá k tomu eventy a listenery.
- **Finder** vyhledává soubory a cesty v souborovém systému.
- **Guard** je nástroj pro autentifikaci.
- **OptionsResolver** pomáhá při vkládání vícero vstupů do metod. Jedná se o vylepšení metody array_replace.
- **Polyfill** je nástroj který poskytuje možnost rozšíření o PHP prvky, jako například APCu, Iconv, anebo konkrétní verze PHP.
- **PropertyInfo** pracuje se zdroji tříd a využívá metadata.
- **Security** je ochranný prvek, který zajišťuje bezpečnost aplikace. Stará se například o autorizaci uživatelů.
- **Stopwatch** umožňuje měření času provádění určitých částí kódu.

Dále Symfony nabízí následující možnosti:

- cachování,
- vkládání závislostí,
- zamykání sdílených zdrojů,
- práci s DOM,
- dotenv,
- EL,
- práci se souborovými systémy,
- formuláře,
- správu HTTP,
- ICU knihovnu,
- LDAP,
- posílání správ mezi aplikacemi,
- PHPUnit testování,
- serializaci,
- šablonování,
- tvorbu překladů,
- workflow a
- FSM.

Hlavními návrhovými vzory, které Symfony používá, jsou: MVC, decorator, mediator, observer, factory, composite, builder, service locator, dependency injection, front controller a unit of work. Většina z nich bude popsána ve 3. kapitole. [13]

2.5 Laravel

Laravel patří mezi nejoblíbenější PHP frameworky. Za jeho vytvořením stojí americký vývojář Taylor Otwell, který vydal první verzi v červnu 2011. Byla to jeho odpověď na špatné odchytávání problémů v PHP programování, který se v té době dal přirovnat k divokému západu programovacích jazyků. Například kvůli dynamickému typování proměnných se naskýtá velká šance že vzniknou chyby, které se projeví až po spuštění kódu. Otwell se inspiroval .net frameworkem od Microsoftu a vytvořil jednoduchý nástroj pro kvalitní psaní aplikací v PHP, podpořený rozsáhlou dokumentací.

Počátky Laravelu měly nezvykle vysokou frekvenci vydávání nových verzí. Verze 2 vyšla v listopadu 2011 a verze 3 v únoru 2012 (tedy 3 verze v méně než jednom roce). Framework obsahoval podporu MVC vývoje, ORM databázi, lokalizaci, routování a další. Dále byl přidán šablonovací engine Blade. Díky své jednoduchosti a rozsáhlé dokumentaci patří Laravel mezi frameworky s velice krátkou dobou pro naučení. Taylor Otwell v readme k druhé verzi laravelu uvedl [22]: Freeing you from spaghetti code, Laravel helps you create wonderful applications using simple, expressive syntax. Development should be a creative experience that you enjoy, not something that is painful. Enjoy the fresh air. Volně přeloženo: Laravel vám ulehčí psaní aplikací a osvoboví vás od špagetového kódu tím za použití jednoduché a výmluvné syntaxe. Vývoj by měl být kterativní zkušenost kterou si užijete, a ne něco bolestivého. Užijte si volného vzduchu.

Koncem května roku 2013 vyšla 4. verze Laravelu přezdívaná Illuminate. Jednalo se o naprosto nový produkt, který Otwell od základu předělal. Nový byl i přístup k pojetí frameworku, jelikož se už nejednalo o komplexní nástroj, ale o jednotlivé balíčky. Laravel byl jedním z prvních Frameworků, který pomohl při rozšíření používání Coposeru. Byla přidána i veliká podpora unit testování nebo nové funkcionality pro ORM, jako například měkké mazání záznamů. [23][24]

2.6 Yii

Yii je PHP framework, vytvořený čínským vývojářem, Qiangem Xuem. Název Yii znamená v čínštině „jednoduché a evolucionářsné“. Vychází z frameworku PRADO, který byl silně inspirován například frameworky ASP.NET nebo Borland Delphi.

Nejnovější verze je 2.0.22 a obsahuje mimo jiné implementaci vlastního MVC modelu, velikou podporu ORM, vlastní generátor kódu pro zpracovávání například databázových požadavků, nebo http request a response objekty. Jeho nejlepší vlastností je ale to, že nabízí vysokou úroveň ochrany. [25][26][27]

2.7 Phalcon

PHP Framework Phalcon se řadí mezi nejrychlejší aplikační frameworky. Jeho rychlost zajišťuje to, že je postaven jako rozšíření C jazyka. Mezi jeho základní vlastnosti patří:

- minimální spotřeba paměťových a procesních prostředků,
- podpora MVC a HMVC,
- podpora Dependency Injection,
- vlastní implementace směrovače požadavků,
- podpora ORM, ODM a cashování,
- PHQL (Phalcon Query Language), což je jeho vlastní obdoba SQL,
- možnost využívání transakcí,
- jednoduché nástroje pro frontend vývoj, jako například šablonovací nástroj Volt,
- částečně česká lokalizace na webových stránkách, což je v porovnání s ostatními frameworky (vyjma Nette) velmi překvapivé.

V porovnání s ostatními frameworky je Phalcon skutečně velmi rychlý. Když se ale porovná s čistým PHP 7, je jeho výkon na podobné úrovni. Neduh robustních frameworků je takový, že přidáváním různých funkcionalit, které zlehčují vývoj, se bohužel může zpomalit výkon. Jak bylo ale popsáno v 1. kapitole, PHP 7 nabízí spousty věcí už v základu a celkově se zlepšila jeho rychlost. Tím pádem hlavní důvod proč si vybrat právě framework Phalcon již není tak aktuální, jako býval dříve. [28]

2.8 Nette

Dalším zajímavým frameworkem, převážně pro české vývojáře, je Nette. Jedná se o framework vyvinutý českým programátorem a nadšencem pro PHP, Davidem Grudlem. První open source verze vyšla v roce 2008. Díky rozsáhlé české dokumentaci a široké komunitě je tento framework jedním z nejvhodnějších pro začínající české programátory.

Nette si implementuje vlastní šablonovací nástroj, nazívaný Latte, který využívá stejné syntaxe jako čisté PHP. Dále nabízí Tester (nástroj pro unit testování), Tracy (nástroj pro debugging) nebo například Neon, což je jejich vlastní formát pro serializaci dat, podobný YAMLu.

Užitečnou součástí frameworku je také nástroj zvaný Componette. Jedná se o databázi různých modulů a doplňků pro Nette. Ne všechny moduly jsou ale vytvořeny ověřenými vývojáři. Je proto často potřeba zkontrolovat, co do našeho řešení implementujeme, abychom zajistili co nejvíce optimalizovaný výsledek. Z tohoto důvodu frameworky jako Symfony atd. od tohoto principu dávno odešli. [19]

2.9 Srovnání webových frameworků

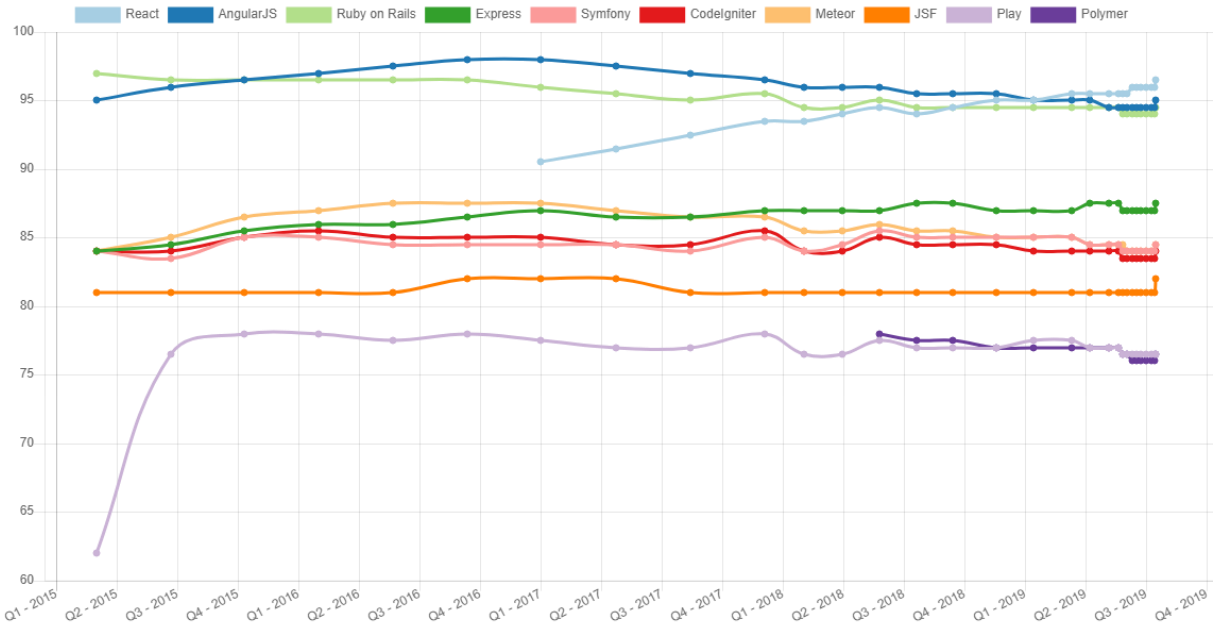
V této podkapitole je ukázáno, jak si vedou PHP frameworky v porovnání s frameworky jiných programovacích jazyků, zaměřených na tvorbu webových aplikací. Nejprve je zde nastíněno, jak je to s jejich popularitou.

Portál hotframeworks.com uvádí srovnání nejvyužívanějších webových frameworků současnosti. Jako metriku využívá GitHub Score, což je počet založených GitHub repozitářů s příznakem daného frameworku. Stejně to je i se Stack Overflow Score, kde se jedná o počet příspěvků týkajících se frameworku. Pro jednodušší zobrazení byly všechny hodnoty převedeny na stupnici od 0 do 100. Poté byl vypočítán průměr z obou těchto indexů. 10 frameworků s nejlepšími výsledky je uvedeno na grafu. Stack Overflow Score ale není tak významný jako GitHub Score, protože daný framework může být lehce pochopitelný a intuitivní. Nebude proto vyžadovat tolik vláken s dotazy na řešení problémů.

V tomto žebříčku se bohužel PHP frameworky umístily až na pátém a šestém místě. Jedná se o Symfony a CodeIgniter, a jsou to jediní zástupci PHP na špici tohoto srovnání. Nejlépe je na tom React a poté AngularJS, oba z rodiny JavaScriptových jazyků.

Graf nejlepších 10 frameworků je znázorněn na obrázku 2. V tabulce 2 je soupis nejlepších PHP frameworků podle jejich pořadí. Hodnoty v tabulce a grafu jsou lehce odchýlené. Nejspíše

z důvodu toho, že buď tabulka nebo graf je aktuálnější. Hodnoty na této stránce se mění skutečně každým okamžikem. V druhém kvartálu tohoto roku byl v první desítce pouze Laravel a Symfony ani CodeIgniter tam vůbec nebyly. [29]



Obrázek 2 - Graf 10 nejoblíbenějších webových frameworků k datu 16.7. 2019 [29]

Tabulka 2 - Nejoblibenější PHP frameworky [29]

Pořadí	Framework	Github body	Stack Owerflow body	Body celkem
6	CodeIgniter	81	87	84
7	Symfony	82	87	84
12	Zend	71	78	74
14	Yii	69	76	72
20	Slim	76	61	68
38	Kohana	59	60	59
46	SilverStripe Saphire	51	60	55
50	FuelPHP	58	49	53

Dále se podíváme na výkon webových back-end frameworků. K tomuto účelu jsme využili projekt skupiny the-benchmarker. Jedná se o nástroj kontrolující výkonnost nejpoužívanějších frameworků pro webové aplikace. Tento nástroj je dostupný jako repozitář na GitHub.com. Je to spustitelná aplikace, která umožňuje otestovat jaký framework bude nejlepší pro naše zařízení.

Tento nástroj umožňuje testovat frameworky, které jsou převážně z jazyků C, C++, C#, go, java, node, php, python, ruby a rust.

Dále je v jejich readme souboru ukázán aktuální stav testování. Vždy, když vyjde nová verze nějakého frameworku, provede tato skupina nové testování. V **Chyba! Chybný odkaz na záložku.** jsou znázorněny hodnoty čtyř nejlepších frameworků z rodiny PHP. Pro srovnání jsou v ní znázorněny i tři nejlepší a tři nejhorší frameworky vůbec.

Tabulka 3 - Srovnání výkonosti webových frameworků [30]

Pořadí	Jazyk	Framework	Požadavky/vteřinu	Propustnost
1.	c (11)	agoo-c (0.5)	199670.00	115.49 MB
2.	python (3.7)	japronto (0.1)	177634.00	212.57 MB
3.	java (8)	rapidoid (5.5)	153167.00	275.56 MB
49	php (7.3)	slim (3.12)	43847,33	217.11 MB
50	php (7.3)	zend-expressive (3.2)	42281	209.34 MB
51.	php (7.3)	symfony (4.3)	42019,67	208.50 MB
54.	php (7.3)	zend-framework (3.1)	39650	196.61 MB
84.	crystal (0.29)	athena (0.7)	6 247.67	7.81 MB
85.	ruby (2.6)	rails (5.2)	3 680.33	11.28 MB
86.	python (3.7)	cyclone (0.0)	2 889.33	7.85 MB

Hlavní metrika u této tabulky je počet požadavků za vteřinu. Podle té se PHP místilo nekde ve středu tabulky. Výkonnostně doshauje Slim framework, nejrychlejší z rodiny PHP, méně než čtvrtinové hodnoty, v porovnání s nejvýkonějším agoo-c. Nicméně hodnotami propustnosti se vyrovnají těm nejlepším. [30]

2.10 Micro Frameworky

Využívání velkých PHP frameworků (Laravel, Symfony, Zend) má tu výhodu, že uživateli pomůžou při celkovém nasazení aplikace. Obsahují funkcionality, které pokryjí celý postup při vývoji aplikace. Jakékoliv frameworky které toto pravidlo nesplňují, spadají do kategorie Micro Frameworků. Díky tomu, že micro frameworky mohou vynechat spousty komponent které klasické frameworky poskytují, bývají rapidně rychlejší.

V případě menších projektů, kde si jsme jisti, že využijeme pouze některé složitější funkcionality, je vhodné využít jeden nebo několik mikro frameworků. Dalším odvětvím, kde najdou mikro frameworky využití je, když využíváme velký framework a chceme ho podpořit nějakou

vlastní službou. V případě že se nám ale projekt časem rozroste, je možné že využijeme zbytečně moc velké množství mikro framworků. A ty spolu nemusí správně spolupracovat. [31]

2.10.1 **Slim**

Jedním z nejlépe hodnocených micro frameworků je Slim. Mezi jeho hlavní funkcionality patří HTTP Routování, Middleware, PSR-7 (HTTP zprávy) a Dependency Injection. S frameworkem Slim je sice možné vytvářet plnohodnotné a rozsáhlé aplikace, nicméně největší využití najde v jednoduchých aplikacích.

Jeho jádro je dispatcher, který přijme HTTP request, zavolá příslušnou rutinu a navrátí HTTP response. Dále nabízí vlastní rozšíření Slim-Csrf, Slim-HttpCache a Slim-Flash nebo rozšíření o balíčky třetích stran, dostupných na Packagist. [32]

2.10.2 **Silex**

Silex je postavený na základech frameworku Symfony a DI kontejneru Pimple. Je jednoduchým frameworkem pro základní aplikace o jednom souboru. Díky Pimplu jej ale lze lehce rozšířit o komponenty třetích stran. Nabízí stručné API, rozšiřitelnost a testovatelnost. Jeho hlavní funkcí je mapování Controllerů k routám (cestám). Vývoj zaštiťují Fabien Potencier (vývojář Symfoony) a Igor Wiedler. [33]

2.10.3 **Lumen**

Vývojáři frameworku Laravel přišli s vlastním odlehčeným Micro Frameworkem. Nazvali ho Lumen. Slibují lepší výsledky a větší výkon než Silex či Slim. I když umožňuje tvorbu samostatných aplikací, nejvíc jeho vlastností využijete, když budete chtít rozšířit vaši aplikaci psanou v Laravelu o nějakou vedlejší službu.

Když ale programátor využívá pro tvorbu své aplikace čistě Lumen a po čase zjistí, že se aplikace rozrostla do takové míry, že micro framework již nebude stačit, je velice jednoduché převést celou aplikaci pod Laravel. Na oficiálních stránkách Lumenu jsou o tom podrobné informace. [34]

2.10.4 Flight

Flight, stejně jako všechny výše zmiňované micro frameworky, poskytuje jednoduchý systém routování, který tvoří základ aplikací. Kromě toho nabízí také vlastní přístup ke statickým proměnným a třídám a možnost filtrování metod ještě předtím, než se spustí. Za jeho vývojem stojí vývojář ze San Franciska, Mike Cao. [35][36]

2.10.5 Srovnání PHP Micro frameworků

Jak z této podkapitoly vyplývá, většina micro frameworků poskytuje vlastní jednoduché řešení routování a mvc struktury. To ve výsledku funguje jako srdce celé aplikace. Dále každý mikro framework nabízí omezené množství dalších funkcionalit, které se u každého mohou lišit. Je také velice důležité, aby měl vývojář šanci rozšířit vlastní jádro frameworku o balíčky poskytované jinými vývojáři.

Při vývoji vlastního frameworku se tedy pokusíme přiblížit spíše k tomuto kompaktnímu řešení, než abychom vytvářeli komplexní složitý framework. Ten by mohl obsahovat veškeré funkcionality, které uživatel mnohdy ani nepotřebuje.

Když se podíváme na oblíbenost těchto mikro frameworků, můžeme si všimnout, že nejvyšší příčky řebříčků zabírají produkty zaštítěné většími frameworky. V námi vybraném seznamu je jedinou výjimkou micro framework flight. [37]

2.11 Implementace návrhových vzorů ve frameworkách

V této podkapitole je shrnutí, s jakými návrhovými vzory jsme se nejčastěji setkali při rozboru frameworků. Na základě toho bylo rozhodnuto, jaké návrhové vzory jsou implementované v našem vlastním mikro frameworku. V tabulce 4, 5 a 6 je znázorněno, jestli je návrhový vzor obsažen ve vybraném frameworku.

Tabulka 4 - Obsažení návrhových vzorů ve frameworcích, část 1.

Framework/ Návrhový vzor	MVC	Page Controller	Front Controller
Zend	Ano	Ne	Ano
Symfony	Ano	Ne	Ano
Laravel	Ano	Ne	Ano
Yii	Ano	Ne	Ano
Phalcon	Ano	Ne	Ano, formou dispatcheru
Nette	Ano	Ne	Ano, formou rozšíření
Slim	Ano	Ne	Ano
Silex	Ano	Ne	Ano
Lumen	Odlehčená verze	Ne	Ne
Flight	Ano	Ne	Ne

Tabulka 5 - Obsažení návrhových vzorů ve frameworcích, část 2.

Framework/ Návrhový vzor	Two step view	Template View	Composite
Zend	Ne	Ano	Ano
Symfony	Ne	Ano	Ano
Laravel	Ne	Ano	Ano
Yii	Ne	Možnost využít Twig nebo Smarty	Ano
Phalcon	Ne	Ne	Ano
Nette	Ne	Ano	Ne
Slim	Ne	Podpora Twigu	Ne
Silex	Ne	Podpora Twigu	Ne
Lumen	Ne	Podpora Blade	Ne
Flight	Ne	Částečně	Ne

Tabulka 6 - Obsažení návrhových vzorů ve frameworcích, část 3.

Framework/ Návrhový vzor	Observer	Dependency Injection
Zend	Ano	Ano
Symfony	Ano	Ano
Laravel	Ano	Ano
Yii	Ano	Ano
Phalcon	Ano	Ano
Nette	Ano	Ano
Slim	Ano	Ano
Silex	Ne	Ano
Lumen	Ano	Ano
Flight	Ne	Ne

Při rozhodování, které návrhové vzory bude náš mikro framework poskytovat jsme museli přihlédnout k následujícím faktorům. Ne všechny návrhové vzory jsou vhodné pro předem připravené řešení. Některé jsou svojí povahou určené k tomu, aby je implementoval sám programátor. Jedná se převážně o základní vzory spojené s OOP. Dalším kritériem pro výběr bylo rozdělení, které návrhové vzory jsou neužitečnější při vývoji webové aplikace.
Na základě výše zmíněných poznatků jsme vybrali následující návrhové vzory:

- MVC
- Front Controller
- Observer/Event Dispatcher
- Template View
- Dependency Injection

Dále se nabízí otázka, jaké jsou výhody začlenění návrhových vzorů přímo do frameworku oproti ponechání volných rukou uživateli při jejich implementaci. Největší výhodou je hlavní kladná vlastnost frameworků samotných, to je předpřipravení nástrojů, aby se uživatel nemusel zabývat znovuobjevením kola. Navíc je využívání předpřipravených návrhových vzorů od osvědčených vývojářů lepší i kvůli zmenšení pravděpodobnosti, že programátor udělá chybu. Od kódu poskytovaného velkým frameworkem se neočekává nesprávné chování.

Stinnou stránkou je ovšem to, že se ztrácí modularita. Když se rozhodneme využívat těchto předpřipravených řešení, je velice složité si je upravit na míru našemu problému. Z toho důvodu

se v několika případech stává výhodnější, abychom si s vytvořením dané funkcionality poradili sami.

3 NÁVRHOVÉ VZORY

Tato kapitola se zabývá návrhovými vzory. Nejprve je zde objasněn koncept návrhových vzorů všeobecně a poté je zaměřena na vybrané vzory nejčastěji využívané při vývoji webových aplikací. U každého vzoru je uveden jednoduchý praktický příklad s ukázkou jednoho nebo více diagramů, které nám pomohou lépe pochopit jeho fungování (kromě NV jedináček, který není potřeba dokládat s diagramem).

3.1 Návrhové vzory všeobecně

Při vývoji softwaru programátor často zjistí, že narazil na problém, s jehož obdobou se již potýkal. Aby předešel nutnosti řešit stejný problém znovu, využije znalostí z minulosti. Staré řešení samozřejmě nemusí přesně odpovídat novému problému. Vývojář tedy provede abstrakci předešlého vypracování, se kterou bude následně pracovat a formovat ji do nového řešení.

Rozdíl mezi zkušeným a nezkušeným programátorem je často v tom, že zkušený programátor má mnohem více předpřipravených řešení než nováček. Z tohoto důvodu byl vytvořen princip návrhových vzorů, tedy zobecnění často opakujících se problémů.

Princip návrhových vzorů vychází z normálního světa. Například ve stavitelství se dají části projektů také zobecnit. Dveře mohou být řešeny rozdílně, ale vesměs všechny využívají stejných nebo podobných principů.

Návrhové vzory zřídka ukazují konkrétní implementaci. Jedná se spíše o jednoduché vysvětlení problému a následné možné řešení. Dále by měly být popsány výhody a nevýhody daného řešení. Když bude mít programátor na výběr mezi více návrhovými vzory, měl by vědět, jak se jejich použitím ovlivní běh programu. [38][39][40]

3.2 Návrhové vzory pro webové aplikace

Většina návrhových vzorů v této kapitole, je využitelná i v jiném odvětví softwarového vývoje (desktopové aplikace, enterprise aplikace atd.), než jen webovém. Návrhové vzory pro tento výpis byly vybrány podle dvou faktorů. První je, jak často se objevují v komponentech webových frameworků. A druhý, jestli svým popisem jasně dávají najevo výhodu využití při tvorbě webové aplikace. [38][39][40]

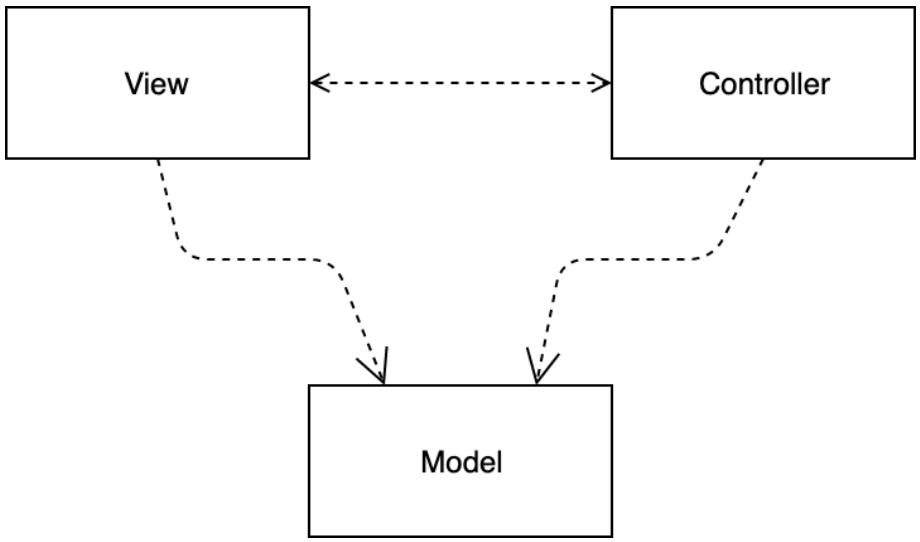
3.2.1 **MVC**

MVC (Model – View – Controller) je jedním z nejvyužívanějších návrhových vzorů. Umožňuje nám separovat pro uživatele viditelnou část od datové části aplikace. První výskyt tohoto návrhového vzoru byl jako framework pro programovací jazyk Smalltalk. V současné době je MVC chápáno spíše jako architektonický vzor než jako návrhový, jelikož se týká celkové architektury programu. [39]

Jedná se o základní myšlenku při tvorbě uživatelského rozhraní a mnoho frameworků využívá tento vzor (nebo jeho úpravu) pro jejich řešení UI. Například Microsoft vyvinul vlastní verzi toho vzoru s názvem MVVM (Model – View – Viewmodel).
Tento návrhový vzor rozděluje objekty na tři hlavní skupiny:

- **Model** je datová část aplikace.
- **View** je pro uživatele viditelná část.
- **Controller** je řídící logika.

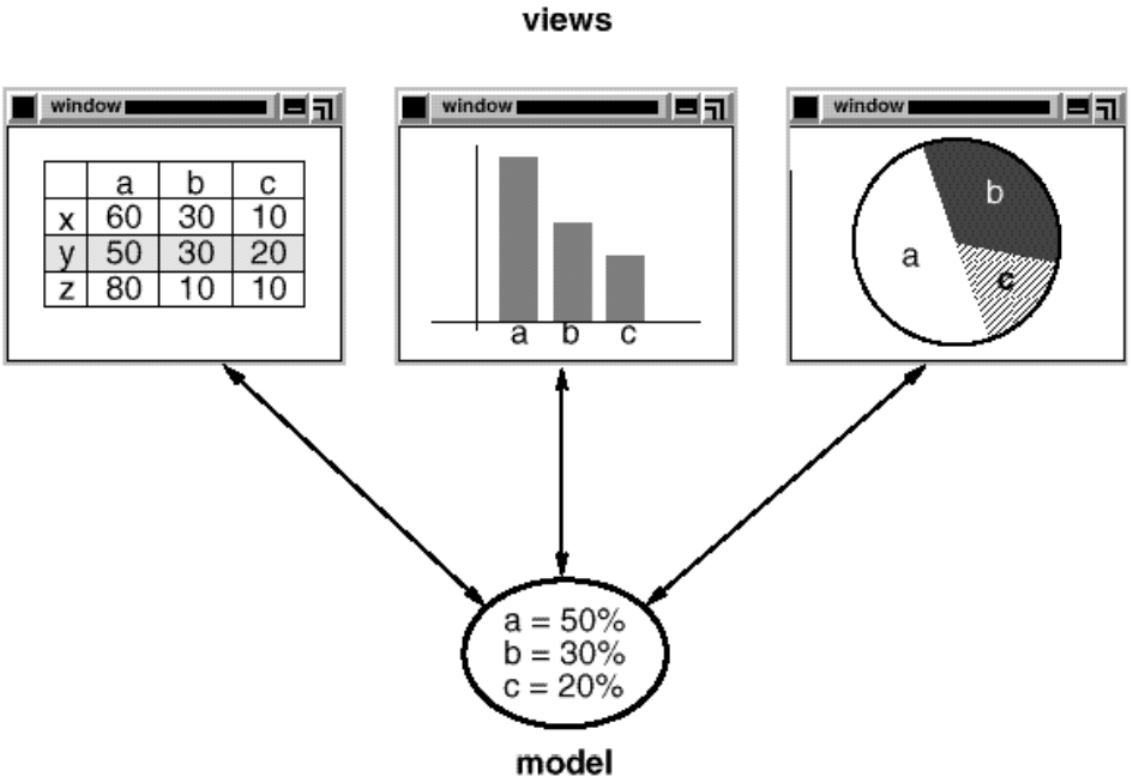
Na obrázku č. 3 je znázorněn vztah mezi touto trojicí. Model je pro uživatele neviditelný objekt, který uchovává všechna data a chování, které nejsou obsaženy v uživatelském rozhraní. Naproti tomu View zastává roli všech viditelných částí. Jakékoliv změny mezi těmito dvěma objekty jsou řešeny pomocí posledního člena této trojice, Controlleru.



Obrázek 3 - Vztah mezi objekty v MVC

Na základě vstupních hodnot od uživatele controller upraví model a následně se aktualizují data zobrazená ve view. Jelikož uživatel přijde do styku pouze s view a controllerem, jsou tyto dva objekty považovány jako součást uživatelského rozhraní.

Díky tomuto přístupu je také možné využívat vzniklé separace mezi view a modelem a mezi view a controllerem. První vzor separace nám umožňuje rychle měnit chování aplikace. Například při zobrazování dat na obrazovku se dá využívat více objektů ze skupiny View. Toto chování je znázorněno na obrázku 4.



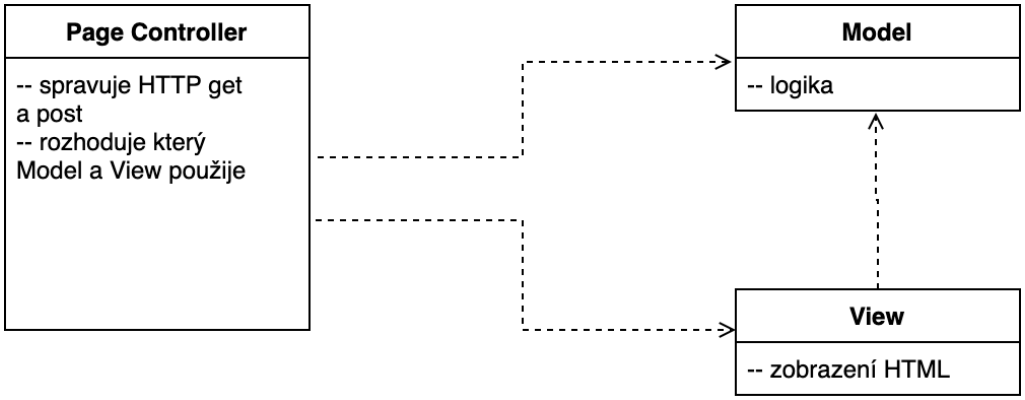
Obrázek 4 - Ukázka možnosti různého zobrazení stejných dat [38]

Při využívání tohoto přístupu je okamžitě viditelný jeden problém. Představme si, že v rámci aplikace máme otevřeno několik zobrazení stejných dat z modelu. Ve chvíli, kdy jako uživatel změníme něco v jedné části view, potřebujeme, aby se ostatní view rovněž aktualizovaly. K tomu je vhodné využít například návrhový vzor obesrver, který bude popsán níže.

Druhý typ separace nám přináší možnost zaměnit Controllery. Díky této vlastnosti je například možné přistupovat ke vstupům od uživatelů různými způsoby. Podle uživatelských rolí se určí příslušný controller a v něm bude umožněno pracovat pouze v souladu s právy daného uživatele. [38][39][40]

3.2.2 Page Controller

Page Controller je výstižný příklad návrhového vzoru určeného pouze pro webové aplikace. Jeho hlavním úkolem je spravovat HTTP požadavky v aplikaci. Controller sám určí jakou stránku zobrazit, nebo jakou akci provést. Jeho grafické znázornění je na obrázku 5.

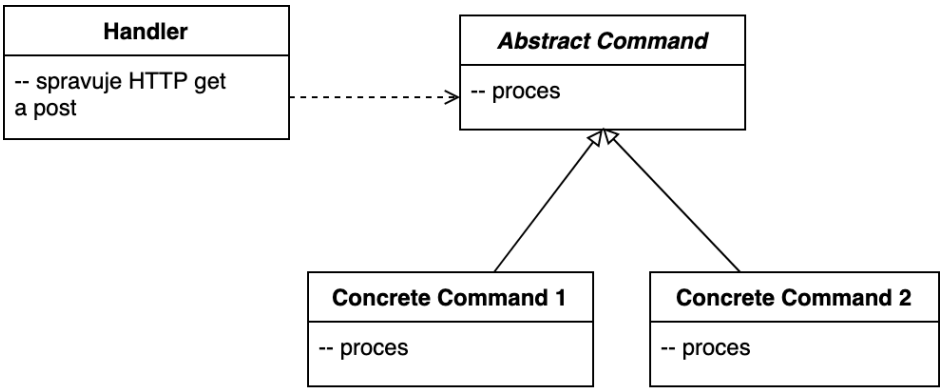


Obrázek 5 - Diagram návrhového vzoru Page Controller

Úkolem Page Controlleru je zachytit HTTP get a post požadavky, a na jejich základě vyhodnotit který model, nebo view použít. Základní myšlenkou je, že každé stránce či akci (v případě dynamických stránek) náleží příslušný controller. [40]

3.2.3 Front Controller

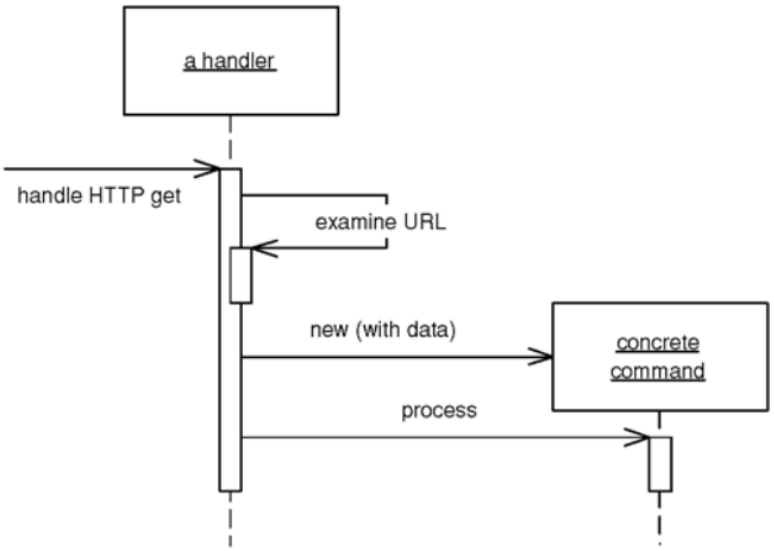
Front Controller zachycuje get a post požadavky a na základě jejich hodnot rozhoduje, jaká akce se provede. Je to tedy alternativa k Page Controlleru. Jeho hlavním cílem je snížit duplicitu kódu a možnost měnit vlastnosti controlleru za běhu. Na obrázku 6 je znázorněn model vzoru Front Controller.



Obrázek 6 - Diagram návrhového vzoru Front Controller

Mnoho podobných věcí se musí provést při řešení HTTP požadavků (bezpečnost, poskytnutí view uživateli na míru atd.). Zatímco Page Controller měl jeden ovladač pro každou akci zvlášť, Frontr Controller má jeden centrální handler a využívá hierarchického uspořádání příkazů. Abstraktní rodič příkazu implementuje společné chování a o konkrétní rozdílnosti se postará neabstraktní potomek.

Handler na základě informací z URI vyhodnotí, která akce se bude provádět. Dále je možné s použitím návrhového vzoru Decorator měnit vlastnosti handleru za běhu programu. Na obrázku 7 je příklad zpracování HTTP požadavku a následné vytvoření příkazu.



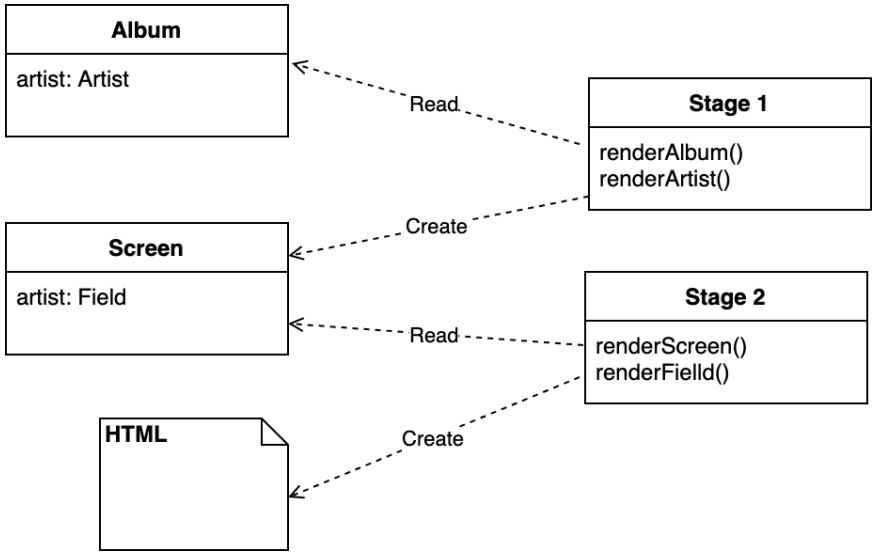
Obrázek 7 - Postup zpracování požadavku v návrhovém vzoru Front Controller [40]

Kvůli zaměnitelnosti Page Controlleru a Front Controlleru je možné se rozhodnout, který návrhový vzor chceme využívat na základě složitosti stránek. Pokud se jedná o jednoduché

stránky, ve kterých by komplexní přístup Front Controlleru přinášel spousty nevyužitých vlastností navíc, je vhodné použít Page Controller. V opačném případě využijeme spíše Front Controller. Většina PHP frameworků implementuje řešení Front Controlleru a Page Controller vynechává. [40]

3.2.4 Two step view

Návrhový vzor Two Step View se stará o zobrazení dat v HTML podobě. Z dat převzatých z databáze se vytvoří nejprve logická stránka, která neobsahuje žádné specifikace výsledného vzhledu. Z té se následně vytvoří konkrétní HTML stránka. Obrázek 8 zachycuje podstatu Two Step View vzoru.



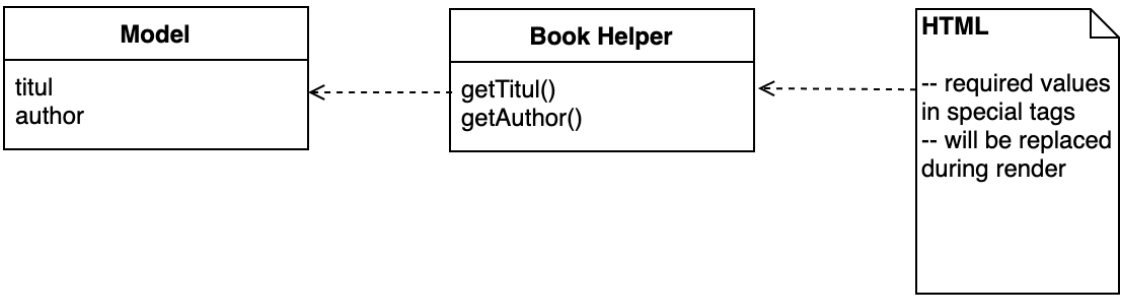
Obrázek 8 - Diagram návrhového vzoru Two Step View

Výhodou tohoto přístupu je například možnost jednoduchého zavedení jednotného vzhledu stránek. K tomu se dají využít i Template View a Transform view. Nicméně u těchto dvou příkladů je možné, že nastavování vzhledu stránek bude obsaženo v několika souborech duplicitně. Využívat sjednocený vzhled stránek je dobré například pro jednoduchou přehlednost, kvalitu designu nebo v případě potřeby globální úpravy všech stránek. [40]

3.2.5 Template View

Template View neboli šablonový pohled je další návrhový vzor určený převážně pro webové aplikace. Základní myšlenkou je nenechávat na základní HTML stránce žádnou logiku. Do

obyčejného HTML zápisu se přidají speciální tagy, které se před zobrazením stránky zamění za skutečné hodnoty. Jednoduchý diagram fungování Template View je na obrázku 9.



Obrázek 9 - Diagram návrhového vzoru Template View

Aby nemusela být HTML stránka přímo závislá na modelu, přidává se třída Helper. Je to něco jako mezikrok mezi databází a konečným zobrazením dat. V praxi je tento Helper praktikován jako součást Controlleru nebo jako mezikrok mezi Controllerem a pohledem. Celý vzor Template View bývá implementací části View v MVC.

Hlavní výhodou využívání Template View je, že pohled může vytvářet i designér sám, bez jakékoliv znalosti logické části kódu. Podobně fungují například WYSIWYG editory, které poté co designér poskládá grafické prvky na svá místa, podle nich vygenerují příslušný kód.

Další možnou funkcionalitou Template View je přidání podmíněných zobrazení a dalších jednoduchých logických řídících prvků. S takto rozsáhlým využíváním Template View ovšem přijdeme o jednoduchost a snadné používání i neprogramujícími designéry. [40]

3.2.6 **Singleton**

Dalším návrhovým vzorem, který zde bude přiblížen, je vzor singleton neboli jedináček. Jedná se o princip zajištění tvorby pouze jedné instance objektu v celé aplikaci. Ve výpočetní technice existuje několik případů, kdy je vhodné využít tento vzor. Například kontrola omezeného připojení do databáze (třeba z důvodu počtu licencí), schránka na kopírování, souborový systém, ovladač tiskárny a další.

Implementace jedináčka je velice jednoduchá. Stačí neuvádět veřejný konstruktor a místo něj nabídnout statickou metodu, která náš privátní konstruktor obalí. V těle této metody je zajištěna tvorba instance, která se následně uloží do statického atributu třídy.

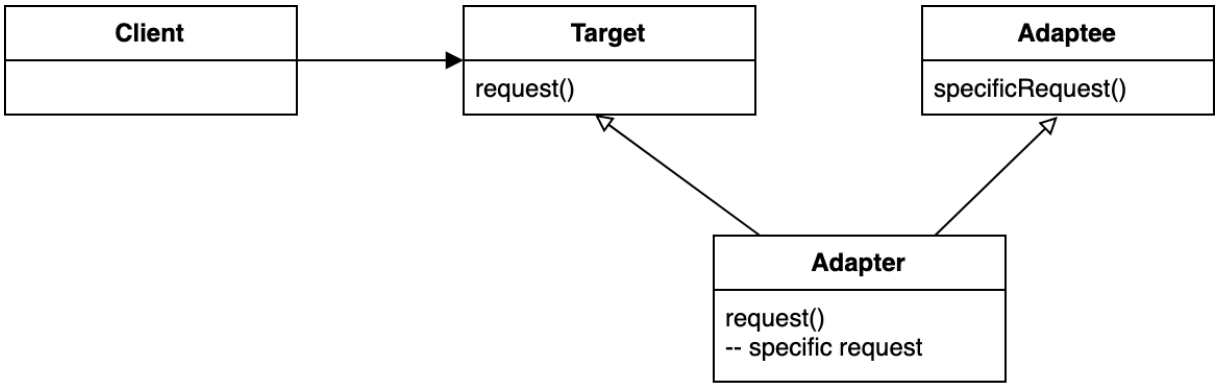
Dále se různé implementace jedináčka liší například způsobem předání odkazu na instanci nebo časem vzniku instance. V prvním případě je vhodné pro předání odkazu na vytvořenou instanci využít veřejnou metodu, která vše zařídí. Je ovšem možné využít globální proměnné. Od tohoto řešení se většina vývojářů snaží oprostit. V některých případech se ale může vyplatit. Například při častém odkazování na instanci.

Podle času vzniku instance se dá mluvit o časné a odložené inicializaci. V případě časné inicializace se vytváří instance již v deklaraci třídy. Naopak, odložená inicializace při požádání o instanci nejprve zjistí, jestli je už nějaká vytvořená. V případě že ne, ujme se úkoly jejího vytvoření. To může být vhodné například když je vytváření jedináčka z nějakého důvodu náročné (vysoké paměťové nároky, zabírání sdílených prostředků atd.).

Návrhový vzor singleton patří mezi opravdu jednoduché vzory a je jedním ze základních principů programování. Většinou se s jedináčkem setkáme již v úvodních kurzech programování. Někteří vývojáři ho ale nedoporučují, převážně z důvodu nutnosti užití statické metody k jeho vytvoření. Od statiky se snaží většina programátorů odpoutat a zastávají názor, že jakákoliv aplikace se dá napsat bez jediného použití statiky. [38][39][40]

3.2.7 Adapter

Návrhový vzor adaptér je možné využít ve chvíli kdy potřebujeme, aby námi vytvořená třída měla jiné rozhraní než to, kterým právě teď disponuje. V tom případě se mezi naši třídu a uživatele vloží třída adaptér. Na obrázku 10 je jednoduchý diagram adaptéru.



Obrázek 10 - Diagram návrhového vzoru Adapter

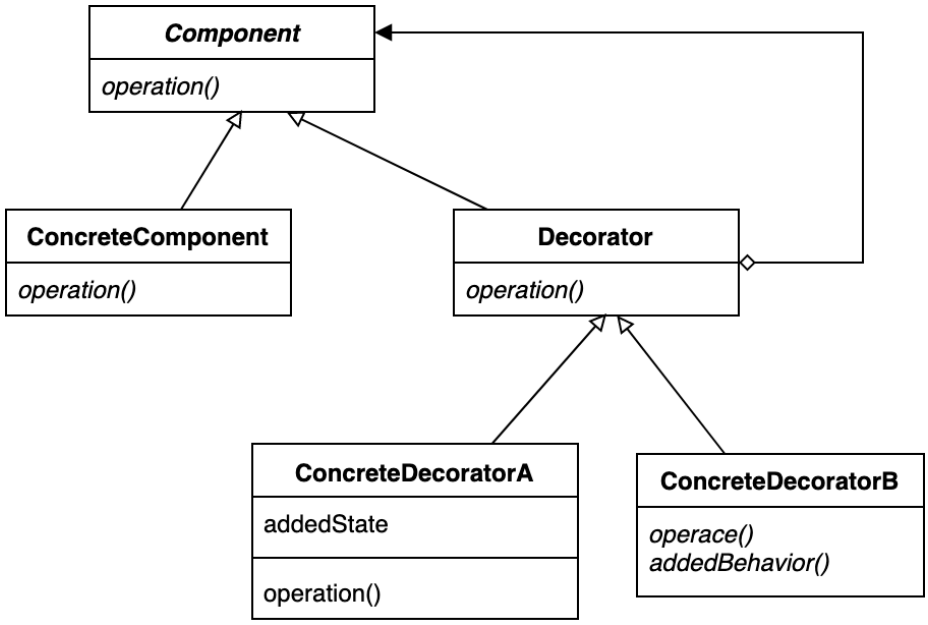
Příkladem užití může být převod jednoduchých datových typů na objekty. Například v případě potřeby předání těchto typů jako parametrů metod, které požadují na vstupu instance

tříd, typu objekt. V takovém případě se vezme jednoduchý datový typ a obalí se třídou obsahující rozhraní objektu. Z tohoto důvodu se tomuto návrhovému vzoru říká také wrapper (obalovač). [38][39][40]

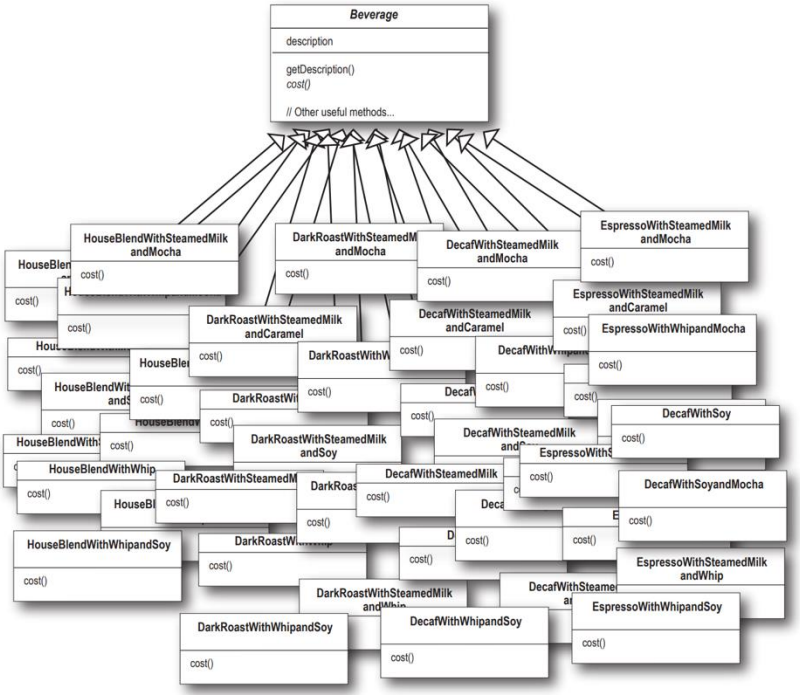
3.2.8 **Decorator**

Dekorátor, podobně jako adaptér, přidává objektu nějakou funkčnost navíc. Zkusme si význam a fungování dekorátoru nastínit následujícím příkladem. Mějme základní objekt, například černou kávu. Ta disponuje chutí a cenou. Když k ní přidáme smetanu, změní se jak chuť, tak cena. Nebo můžeme přidat cukr a šlehačku. V obou případech se základní objekt obalí další funkcionalitou. Ve druhém případě dokonce dojde ke dvojímu obalení. Na Obrázek 11 je diagram dekorátoru. Dále na obrázku 12 a obrázku 13 je ukázáno špatné řešení problému s kávou (objekt obsahuje všechny vlastnosti) a dobré řešení za pomoci dekorátoru.

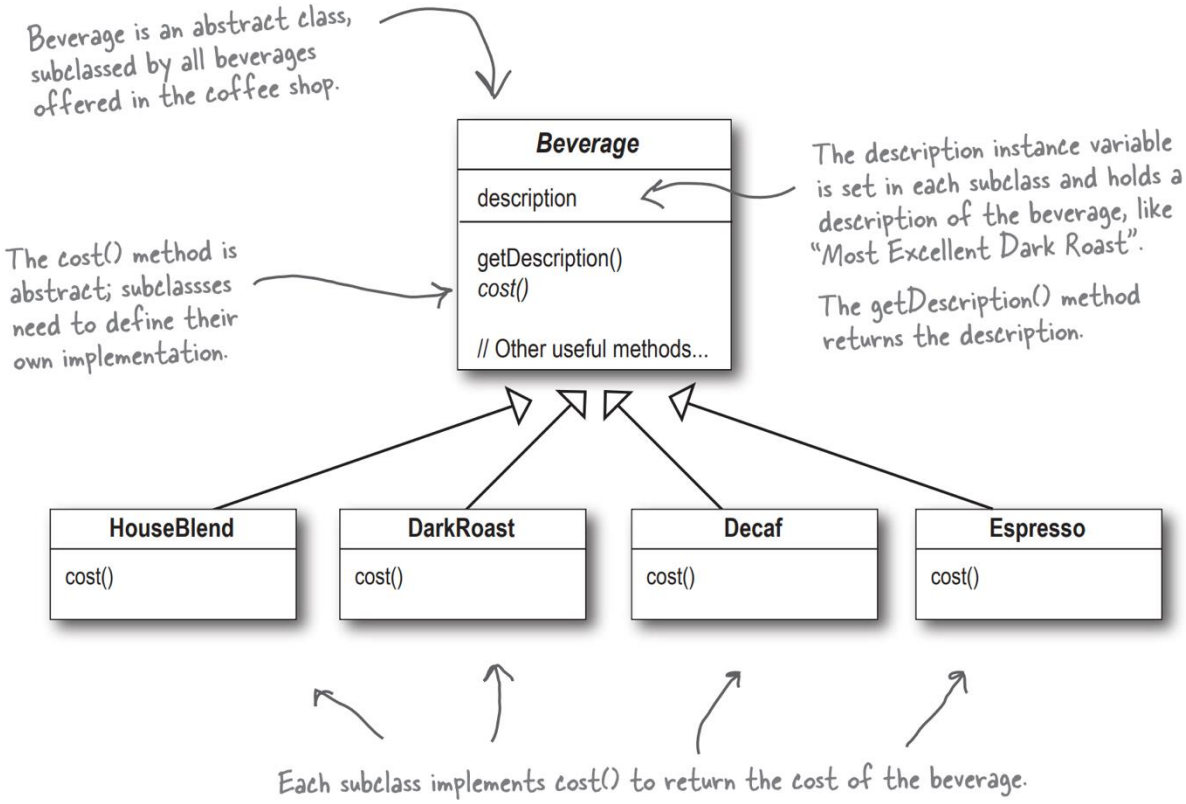
Tato podkapitola byla zpracována za použití zdrojů [38][39][40][41].



Obrázek 11 - Diagram návrhového vzoru Decorator



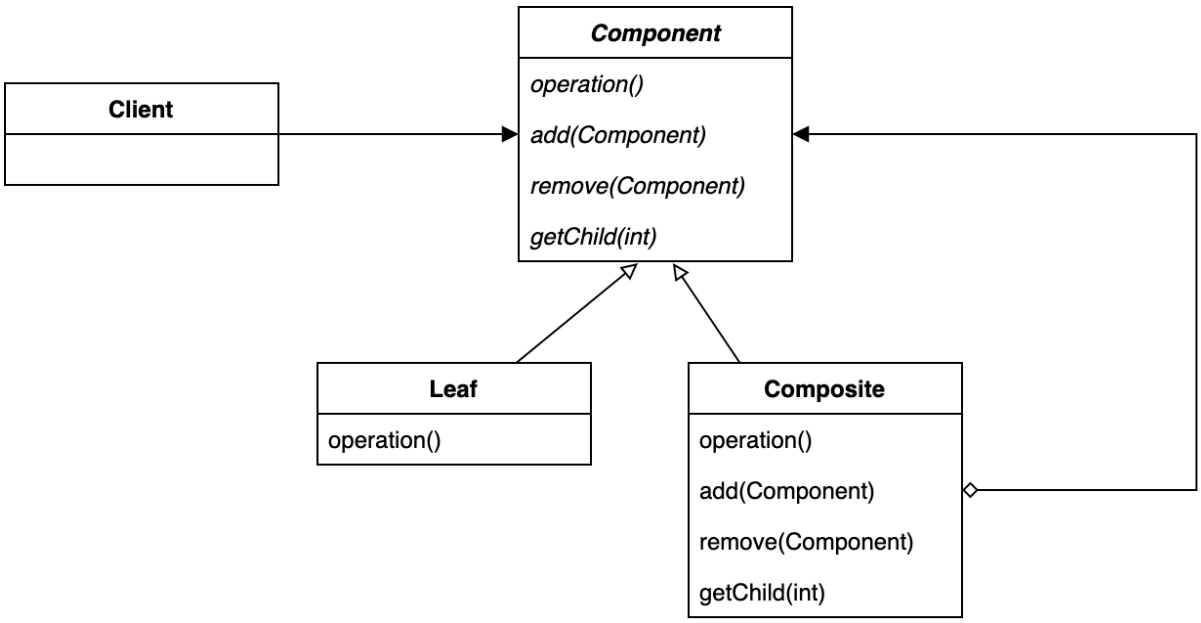
Obrázek 12 - Ukázka špatného řešení příkladu s kávou [41]



Obrázek 13 - Ukázka správného řešení příkladu s kávou pomocí návrhového vzoru Decorator [41]

3.2.9 **Composite**

Composite je návrhový vzor, který umožňuje pracovat s individuálními prvky stejně jako se skupinovými. Tento přístup je výhodný například při práci se stromy, což je jedno z českých synonym k tomuto návrhovému vzoru. Datová struktura strom definuje dva typy objektů, uzly a listy. Spojnicí mezi nimi je větev. Díky vzoru Composite se dá přistupovat k uzlům i listům stejně. Na obrázku 14 je jeho diagram.

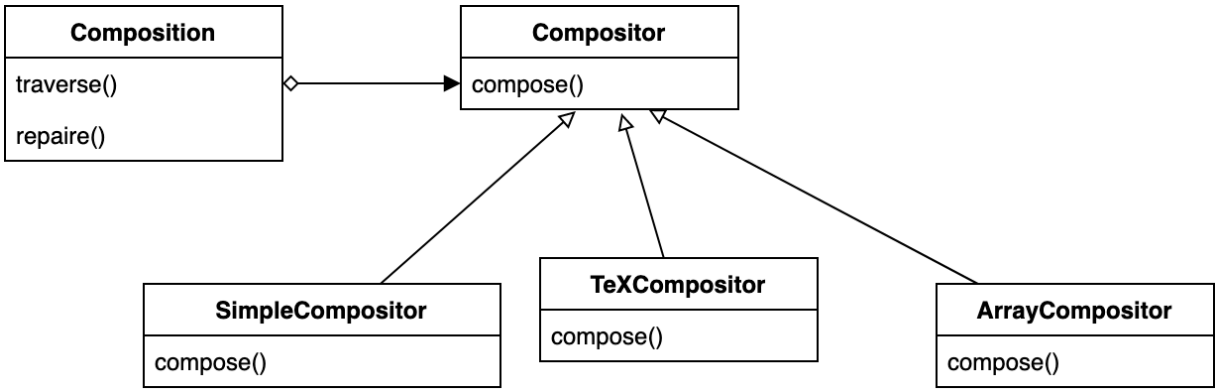


Obrázek 14 - Diagram návrhového vzoru Composite

Příklady užití v paxi jsou stromová struktura souborů a složek nebo strom matematických výrazů. U prvního příkladu sdílí složky (uzly) a soubory (listy) stejné metody. Druhý příklad určuje jednoduché výrazy, 3+5 jako listy a složené výrazy, ((3+5)*(2-3)) jako uzly. Je samozřejmě možné pro oba typy objektů přidávat specifické vlastnosti. V tom případě je nutné nejprve zjistit skutečný typ, a poté si zavolat některou z jím implementovaných metod. [38][39][40]

3.2.10 **Strategy**

Strategie je jedním z řešení dynamického přepínání mezi různým řešením stejného problému. Skládá se ze zaměnitelných komponent, které je možné za běhu programu střídat. Na obrázku 15 je návrh vzoru Strategy.



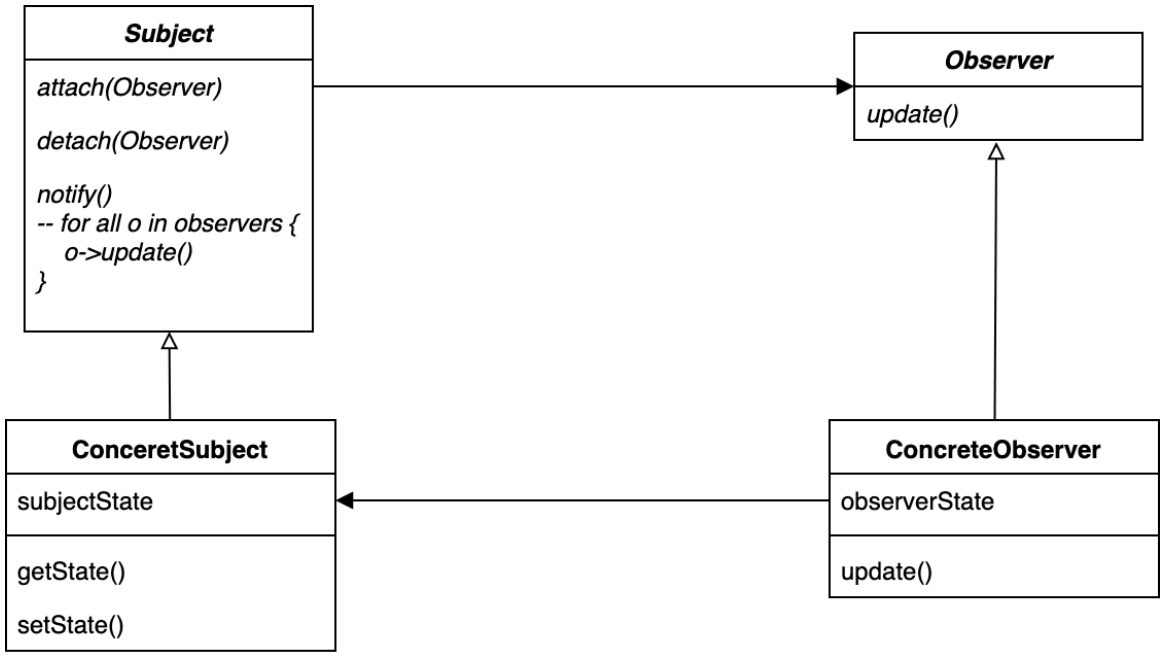
Obrázek 15 - Diagram návrhového vzoru Strategy

Tento návrhový vzor nám přináší mnoho výhod. Připravme si například komponentu, která nám umožňuje počítat obsah čtverce za pomoci klasického vzorce: $S = ((a + c)/2) \cdot v$.
K této komponentě můžeme vytvořit podobnou s tím, že bude využívat jiný vzorec. V našem případě třeba Pickův vzorec, $S = i + \frac{h}{2} - 1$, který pro výpočet používá pomocnou mřížku.
Když naše řešení postavíme tímto způsobem, je jednoduché rychle korigovat funkčnost programu přepínám těchto komponent.

Další výhodou tohoto vzoru je, že je připravený na budoucí úpravy. Pakliže se rozhodneme že je potřeba přidat jiný algoritmus pro výpočet, stačí napsat novou komponentu, kterou do našeho řešení zahrneme. [38][39][40]

3.2.11 **Observer**

Observer neboli pozorovatel je elegantní způsob, jak vyřešit odchytávání změn chování u objektů. Pozorovatelé se jednoduše přihlásí u pozorovaného. Ten je o každé své změně bude informovat a předá jim potřebná data. Na obrázku 16 je jednoduchý diagram tříd pro vytvoření návrhového vzoru observer.



Obrázek 16 - Diagram návrhového vzoru Observer

Nejlepší bude, ukážeme-li si jeho fungování na nějakém příkladu. Představme si portál s novinkami. Ten v našem případě bude zastupovat roli pozorovaného. Náš portál jednou za čas vydá nějakou zprávu formou html kódu s obrázky. Ve chvíli kdy je připravený danou zprávu zveřejnit, podívá se do svého seznamu pozorovatelů a všem odešle stejná data.

Teď se na stejný příklad podívejme z pohledu pozorovatele. Může to být například mobilní aplikace, webová stránka či emailová schránka. Všechny tři objekty mají zájem o novinky na našem portálu. Ve stejnou chvíli tedy přijdou nová data všem na seznamu pozorovatelů. Nyní je pouze na pozorovateli, jak s daty naloží. Jaký vybere způsob pro zobrazení dat uživateli případně jestli je vůbec využije.

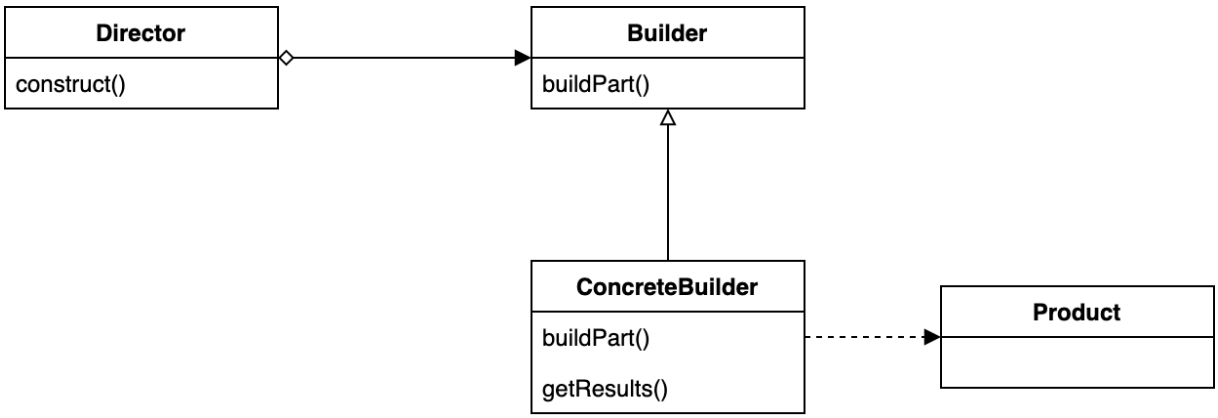
Možnost určit si, jestli poskytovaná data pozorovatel využije, je jedním z implementačních problémů, na které si musíme dát pozor. V klasickém případě může pozorovaný své informace předat okamžitě s upozorněním na změnu. Může ale nastat situace, kdy pozorovatel nemá o data zájem a dochází ke zbytečnému zahlcování prostředků. Tomuto přístupu se říká push (tlačit) mechanizmus.

Na druhou stranu je možné, aby pozorovaný všechny pouze upozornil a umožnil, například veřejnou metodou, přístup k datům. Pakliže se pozorovatel rozhodne pro přijetí dat, pouze zavolá příslušnou metodu. Tomuto se říká pull (tahat) mechanizmus. [38][39][40]

3.2.12 **Builder**

Návrhový vzor stavitel se stará o to, aby se dalo vytváření složitých objektů dělat co nejabstrktněji. Již samotný název vzoru nabádá k využití příklad z oblasti stavitelství. Mějme ředitele (director), který na základě zadaných kritérií volá příslušného stavitele (builder) a využívá jeho metod pro sestavení domu.

Stavitelů může být několik. Jeden na dřevostavby, další na cihlové domy a podobně. Všichni mají metody na postavení různých částí domu. Díky tomu se zajistí univerzálnost. Protože když je potřeba místo jednopatrového domu postavit dům dvoupatrový, zavolá ředitel metodu postav patro dvakrát. Na obrázku 17 je znázorněno fungování builderu.



Obrázek 17 - Diagram návrhového vzoru Builder

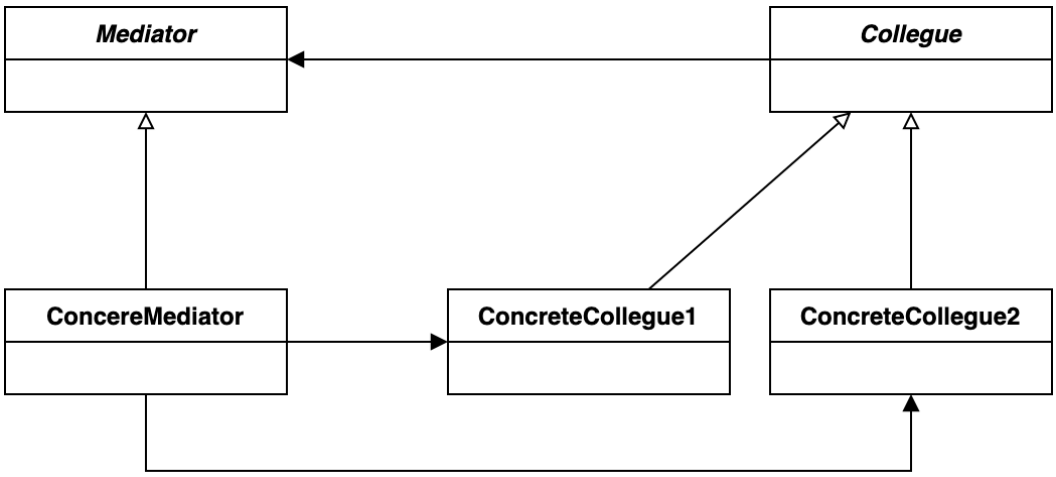
Dalším příkladem, tentokrát z oblasti IT, může být konvertor souborů. V tomto případě bude director, textový editor, například MS Word, kterému uživatel zadá příkaz export souboru. Na základě zvoleného typu souboru se vybere příslušný builder, který může mít například následující metody: vytvoř řádek, vytvoř odstavec, přejdi na novou stránku atd. Directorovi, je jedno jaký výstupní formát uživatel vybere (pdf, docx, txt, …), prostě si vybere příslušného buildera a začne mu předávat postupně příkazy pro sestavení dokumentu. [38][39][40][42]

3.2.13 **Mediator**

Mediator je návrhový vzor, jehož hlavním účelem je zrušení přímé vazby mezi provázanými objekty. Při návrhu aplikace je vhodné se těmto vazbám vyhnout. Hlavním důvodem je možnost upravovat kód komponenty, nebo je přímo zaměnit za jinou. To vše, aniž by o tom objekt, se kterým daná komponenta komunikuje, vůbec věděl. Tento přístup je vhodný zejména v případě,

kdy se na vývoji softwaru podílí více vývojářů nebo jestli naše komponenta komunikuje se samostatnou aplikací, jako například v servisně orientovaných designech projektech.

Tímto se zajistí univerzálnost a rozšiřitelnost našeho kódu. Další řešení k této problematice přináší i výše popsané návrhové vzory, například Factory, Builder, Decorator, nebo Observer. Na obrázku 18 je znázorněna struktura návrhového vzoru Mediator.



Obrázek 18 - Diagram návrhového vzoru Mediator

Doslovný význam slova mediátor je prostředník. Návrhový vzor zastává podobnou roli ve vzájemné komunikaci objektů. Opět si zkusíme nastínit využití mediátora na následujícím příkladu.

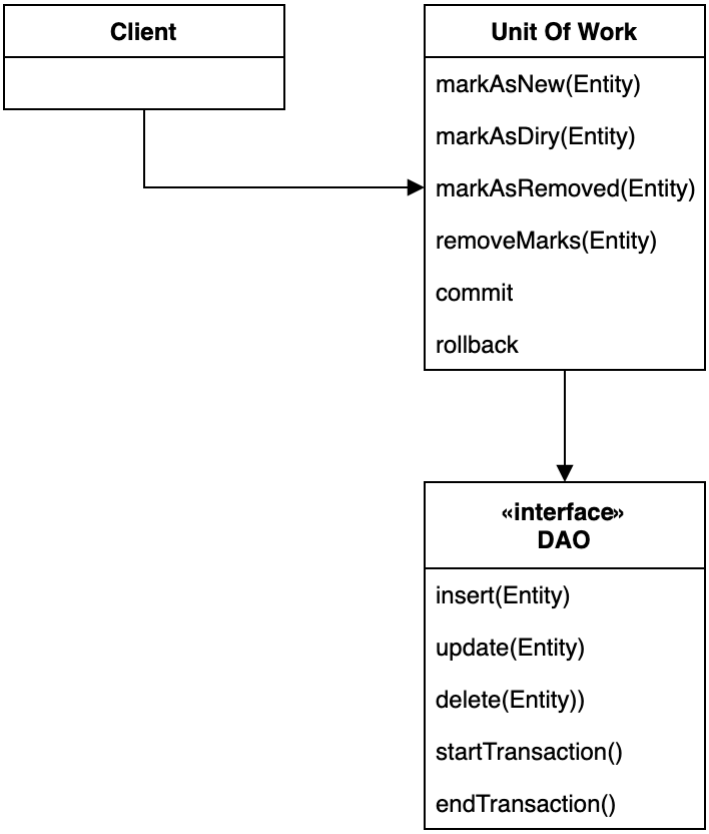
Nejčastěji bývá vysvětlován na logování do souboru. V programu máme více možností kam naše logy ukládat. Uživatel dá příkaz přidat záznam do logu. Nejprve tedy mediátoru nastaví, jestli chce logovat do konzole, nebo například do souboru. Poté si už jen volá metodu mediátoru loguj, které v parametru předá zprávu. O to, kterému objektu předat zprávu, se stará právě náš mediátor. Dokud uživatel nezmění parametr kam předávat výstup, bude mediátor zasílat stále stejným směrem.

Další výhodou je, že když se změní kód třídy, která je na mediátor napojená, není nutné přepisovat všechny ostatní třídy se kterými komunikuje. Stačí upravit pouze jednu, a to mediátor. Tím se zajistí rozšiřitelnost našeho kódu. [42]

3.2.14 **Unit of Work**

Návrhový vzor Unit of work řeší problematiku změny databázových dat uložených v paměti. Zpracovávaná data se načtou do paměti, nebo se tam přímo vytvoří či smažou. Jakékoliv změny

je nutné promítnout zpět do databáze, jinak o ně při smazání z paměti přijdeme. Další vlastností Unit of Work je, že nám umožňuje některé změny v databázi provádět v dávkách. Na obrázku 19 je znázorněn diagram Unit of Work.



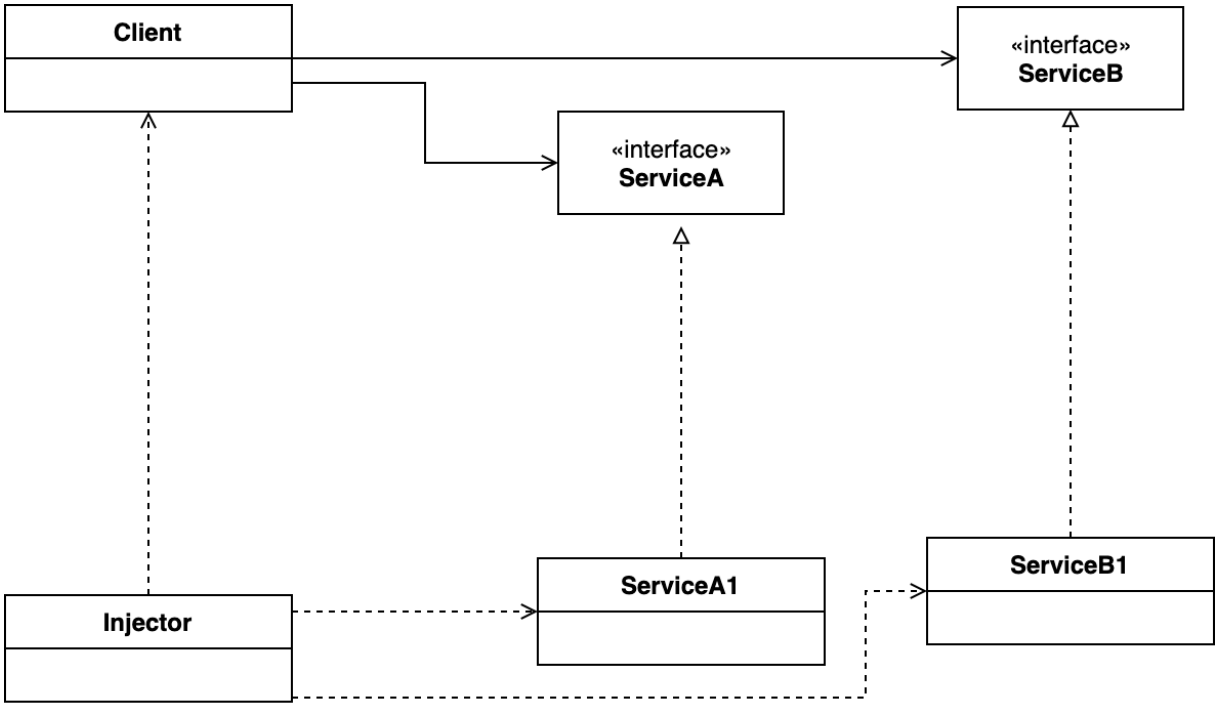
Obrázek 19 - Diagram návrhového vzoru Unit of Work

Třída Unit of Work má na starosti jakékoliv změny entit zapisovat a na příkaz commit, je promítnou do Třída Unit of Work má na starosti jakékoliv změny entit zapisovat a na příkaz commit je promítnou do databáze. Dále by měla nabízet možnost rollback, tedy vrácení zpět do stavu před jakoukoliv změnou. To je vhodné zejména když při úpravě entit objevíme problém, který by nám narušil funkčnost databáze.

Dále by měla nabízet možnost rollback tedy vrácení zpět do stavu před jakoukoliv změnou. To je vhodné zejména když při úpravě entit objevíme problém, který by nám narušil funkčnost databáze. [40]

3.2.15 **Dependency Injection**

Dependency Injection je návrhový vzor, který se, jak jeho název napovídá, stará o vkládání závislostí. Hlavní myšlenkou je nenechávat vytváření objektů na třídách, které je potřebují. Naopak se jim snažíme tyto objekty předávat v parametru. Na obrázku 20 je znázornění architektury tohoto vzoru.



Obrázek 20 - Diagram návrhového vzoru Dependency Injection

Bude opět vhodné si náš vzor vysvětlit na příkladu. Napíšeme metodu, která bude vytvářet kávu, podobně jako obsluha v kavárně. Daná obsluha umí vytvořit požadovaný nápoj, potřebuje k tomu ale základní suroviny a zařízení (kávovar, vodu, zrnkovou kávu atd.). Pro tento příklad se omezme pouze na zdroj zrnkové kávy. Když výběr tohoto zdroje necháme na obsluze, je dost pravděpodobné že dostaneme něco, co nám nebude chutnat. Co se ale stane, když obsluze při objednávce rovnou předáme vlastní zrnkovou kávu? Dopustíme se sice faux pas, ale budeme si stoprocentně jistí výsledkem.

Pro obsluhu je tento přístup rovněž jednodušší, protože nemusí vybírat z velkého výběru kávy (například globálního prostoru). Když si tento příklad představíme v programovém kódu, přináší tento návrhový vzor ještě jednu výhodu, a to jednoduchou čitelnost kódu. Když budeme provádět revizi kódu, okamžitě poznáme, s jakými zdroji volaná třída pracuje.

Jako takzvaný antivzor pro dependency injection bývá často označován Service Locator. Je to objekt obsahující všechny závislosti, které daná třída potřebuje. Když se ale bude předávat tento Service Locator v parametru metody, není hned z kódu patrné, o jaké závislosti se jedná. K tomu bychom museli důkladně prozkoumat implementaci volané metody.

Navíc při využívání Service Locatoru hrozí, že se bude takto předávat zbytečně moc informací. A to i takové, které nakonec nebudou vůbec potřeba. Tímto se může zpomalit běh programu a celkově zvýšit paměťová náročnost.

V praxi se dá princip Dependency Injetion shrnout na dvě pravidla. Při vytváření metody není potřeba vytvářet vlastní zdroje na kterých je metoda závislá. Metoda bude prostě počítat, že jí je někdo dodá při jejím volání. A druhé pravidlo je, že každá metoda by se měla hlásit jen k takovým závislostem, jaké ve skutečnosti potřebuje.

Podkapitola 3.2.15 byla zpracována za použití zdrojů [19][43][44][45].

4 MVC MICRO FRAMEWORK

Tato kapitola se věnuje tvorbě praktické části této diplomové práce. Nejprve se zaměřuje na PSR-4, které nám pomáhá při načítání použitých tříd. Poté je v ní ukázáno, jak implementovat návrhové vzory Front Controller, MVC, Template View a Observer. Dále kapitola probírá, jak při vývoji vlastního frameworku využívat samostatné nástroje třetích stran. V našem případě jde o komponentu Pimple, která je implementací návrhového vzoru Dependency Injection a Monolog, která dodává naší aplikaci logovací systém. Poté jsme implementovali balíčky obsažené ve velkých frameworcích. Využiji jsme komponentu HttpFoundation z frameworku Symfony a Eloquent ORM mapování z frameworku Laravel.

Na závěr je jednoduchý návod, jak vytvořit vlastní repositář ve VCS systému GitHub a jeho následné nahrání na Packagist. Díky tomu je možné si náš framework stáhnout pomocí Composeru. Buďto jako balíček obsažený ve složce vendor nebo jako vzorovou aplikaci s kompletním skeletem.

4.1 Nástroje

V této podkapitole jsou prezentovány nástroje a vývojářské prostředí použité při tvorbě praktické části této diplomové práce. Využit byl i nástroj Composer, který je ale podrobně popsán v kapitole 2.2, není zde proto uveden.

4.1.1 PhpStorm

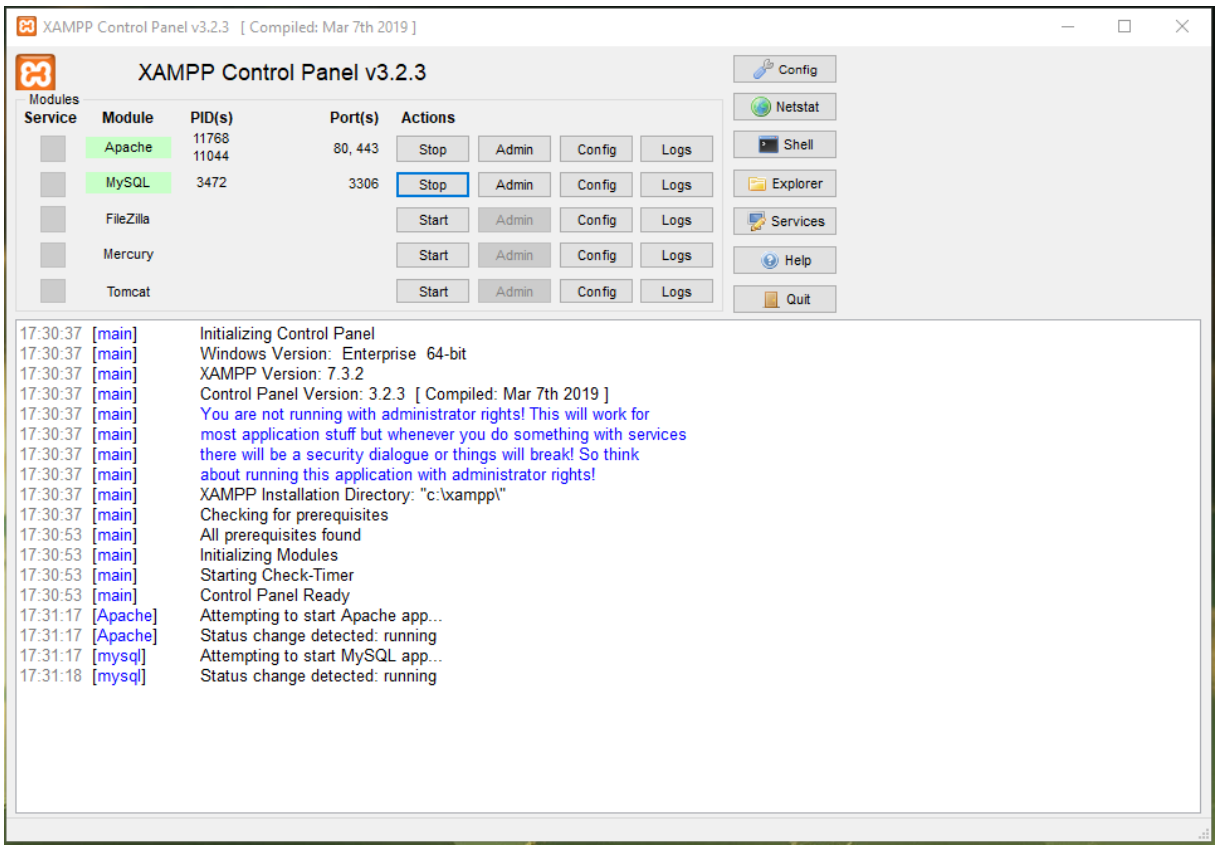
PhpStorm patří mezi nejoblíbenější vývojové prostředí pro tvorbu PHP aplikací. Obsahuje spoustu užitečných nástrojů (přímá integrace Composeru, VCS, atd.) a je uživatelsky velmi přívětivý. Tento nástroj vyvíjí mezinárodní firma JebBrains, jež sídlí v Praze. [46]

4.1.2 XAMPP

Dalším nástrojem je software XAMPP (X = crossplatform, multiplatformní, A = Apache server, M = Dříve MySql, nyní MariaDB, P = PHP, P - Pearl), který slouží převážně pro testování webových aplikací.

Hlavní výhodou je, že stačí nainstalovat pouze tento nástroj a o instalaci všech dalších služeb se postará za nás. My jsme z něj využili Apache server, PHP a MySql. Na obrázku 21 je hlavní

okno aplikace, ze kterého je mimo jiné možné spouštět, configurovat spravovat jednotlivé prvky XAMPPu. [47][48]



Obrázek 21 - Rozhraní nástroje XAMPP

4.1.3 GitHub

Jako poslední je zde služba GitHub. Jedná se nejrozšířenější a nejoblíbenější VCS současnosti. V tomto projektu je využívána pro správu verzí, ulehčení sdílení a založení komunity okolo frameworku. Na závěr si pomocí Gitu ještě vytvoříme balíčky v Composeru.

V repozitářích GitHubu se nachází projekty z celého spektra světa programování. Vedle velkých projektů, jako jsou například celé frameworky pro různé programovací jazyky zde nalezneme i spoustu menších, které se většinou zabívají elementárnímy problémy. Například implementací Binárně vyhledávacího stromu se dá na GitHubu nalézt nespočet, v široké škále programovacích jazyků. Dále slouží jako hlavní médium pro sdílení spousty pluginů do různých programů. Například textový editor Sublime Text se dá pomocí těchto doplňků povýšit na velice sofistikovaný vývojový nástroj.

4.2 Komponenty třetích stran

Jak již bylo zmíněno ve druhé kapitole, většina PHP frameworků využívá komponenty jiných frameworků (byly napsány kvalitně a nebyla potřeba vytvářet vlastní řešení). I mikro framework bude obsahovat části od jiných vývojářů.

4.2.1 Pimple – Dependency Injection Container

Pimple je velice jednoduchá PHP komponenta, která poskytuje Dependency Injection kontejner. Za jeho vývojem stojí Fabien Potencier, který je jedním z tvůrců frameworku Symfony. Patří mezi nejoblíbenější DI kontejnery a využívá ho například již zmiňovaný mikro framework Silex. [49]

4.2.2 Monolog

Mezi nejoblíbenější logovací systém patří Monolog, za jehož vývojem stojí Jordi Boggiano spolu s dalšími vývojáři. Ten plně implementuje rozhraní definované v PSR-3. Využívají ho například frameworky Symfony, Laravel, Lumen a FuelPHP. Frameworky Slim, Nette a Yii umožňují implementaci Monologu pomocí rozšíření.

Monolog umožňuje zaznamenávání informací do široké škály kanálů a médií. V naší aplikaci budeme využívat pouze výstup do konzole prohlížeče a do lokálního souboru, který bude součástí skeletu. [50]

4.2.3 HttpFoundation

Jako další se podíváme na komponentu z frameworku Symfony, a to konkrétně HttpFoundation. Ta nám umožňuje k HTTP komunikaci přistupovat objektově (čisté PHP také poskytuje její základní správu). Její hlavní bloky jsou Request a Response. Dále se díky této komponentě dají jednoduše spravovat soubory cookie. [13]

4.2.4 Eloquent

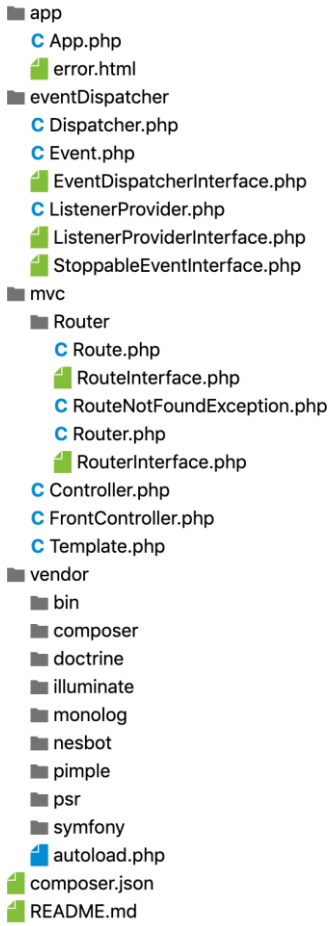
Jako poslední si ukážeme Eloquent ORM. Jedná se o komponentu z frameworku Laravel, díky které je možné mapování databázových tabulek na objekty. Díky tomu se zajistí jednoduší přístup k prvkům databáze. Navíc díky objektovému přístupu, je snazší tuto komponentu začlenit do jinak plně objektové aplikace. [51]

4.3 MVC Micro Framework

Jádrem celé naší aplikace je MVC Micro Framework, který uživateli poskytuje služby směrovače požadavků, Front Controlleru, MVC modelu, Event Dispatcheru a dále všech výše zmíněných služeb třetích stran. Tento mikro framework je nahrán jako součást skeletonu aplikace, na kterém je ukázáno jeho správné fungování.

4.3.1 Adresáře

Obrázek 22 znázorňuje, jakou adresářovou strukturu má náš mikro framework. Všechny tyto soubory jsou podrobněji popsány v následujících podkapitolách. Na obrázku 23 je vidět struktura, kterou pro správné fungování naše komponenta MVC vyžaduje.

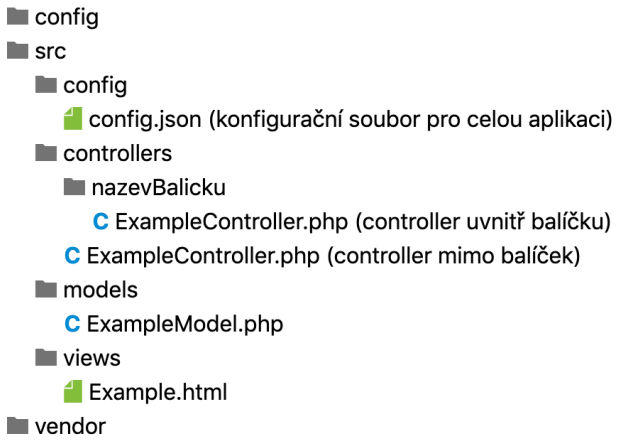


Obrázek 22 - Adresář mikro frameworku

Do složky config se vkládají konfigurační soubory. Dále je zde adresář src, který má čtyři složky: config, controllers, models a views. Do složky views se ukládají soubory s definicí

vzhledu, například .html. Složka models je připravená pro vytváření datových tříd a složka controllers obsahuje všechny uživatelem definované Controllery. Do složky config se poté ukládají všechny konfigurační soubory pro dané moduly.

Jako poslední je zde složka vendor, která obsahuje všechny nainstalované závislosti pomocí slžuby Composer. Je zde vidět více složek než jen od čtyř komponent, využívaných v našem frameworku. To je z toho důvodu, že i tyto komponenty využívají funkce od jiných vývojářů. Tudíš se při instalaci těchto balíčků se nainstalovaly i všechny jejich závislosti.



Obrázek 23 - Adresář požadovaný mikro frameworkem

4.3.2 Modul MVC

Náš framework obsahuje tři balíčky. Jedním z nich je balíček s názvem mvc. V něm jsou vytvořené třídy FrontController, Route, Router, Template a abstraktní třída Controller. Celý tento modul tvoří základ našeho frameworku.

Základní princip fungování je takový, že uživatel pošle na server HTTP požadavek a ten se převede na centrální Front Controller. V něm se pomocí Routeru rozhodne, jakou akci vyvolat. Akce budou definované v konkrétních ovladačích jako metody. V těchto metodách je možné například připravit prvky složky model (třídy které spravují data nebo přímo reprezentují data z databáze) a volat prvky složky view.

4.3.3 Front Controller

Při tvorbě našeho MVC frameworku jsme využili návrhový vzor Front Controller. Základní princip je popsán v kapitole 3.2.3. V našem případě jeho funkci zastupují tři prvky. Prvním je

.htacces soubor, dále soubor index.php (oba si popíšeme v kapitole zaměřené na testování našeho frameworku) a poté třída FrontController.

Naši třídu jsme si vložili do jmenného prostoru MVC. Dále jsme definovali privátní proměnné router, params a diContainer. Jeho konstruktor přijímá parametry router, request a volitelně diContainer. Router a diContainer jsou přiřazeny privátním vlastnostem a request je posílán jako parametr metodě route.

Jako další jsme si vytvořili právě metodu route. Ta z přijatého parametru request (očekává se, že bude instancí HttpFoundation\Request) zjistí obsah HTTP request požadavku. Z něj vyčte, kam přesměrovat a jaké jsou jeho parametry. Následně použije svojí instanci třídy router a použije metodu getMatch, která vrací název žádaného Controlleru.

Dále tato metoda zjišťuje, jaké parametry jsou potřeba pro zavolání konstruktoru. Pro to je zde využita vlastnost reflexe. To znamená, že PHP je schopno vytvořit jen jakýsi odraz třídy (nebo metody), který programu bude poskytovat informace. Tato reflexe je poslána do metody prepareParams, která zjistí, jaké parametry jsou potřeba a připraví je do pole. Parametry vyhledává nejprve v prvcích diContaineru a poté v parametrech poslaných v HTTP request.

Tímto náš mikro framework poskytuje nejjednodušší způsob, jak předávat závislosti. Uživateli stačí, když názvy vstupních parametrů u metod Controllerů budou stejné, jako názvy závislostí v DI kontejneru. Stejný princip funguje i u předávání proměnných pomocí HTTP requestu. Požadované parametry z DI kontejneru a z HTTP requestu lze střídat. Ve chvíli kdy metoda prepareParams zjistí, že požadovaný parametr není obsažen v DI kontejneru a ani v HTTP request, zkontroluje, jestli je parametru povinný. V případě že ano, vyvolá výjimku.

Poté metoda route zajistí vše potřebné se vytvoří instance Controlleru. Z té se zjistí, jestli existuje požadovaná metoda. Jestli existuje, zavolá se opět prepareParams, tentokrát pro tuto uživatelem žádanou metodu, která se zavolá, jestli proběhne vše v pořádku.

```
1.  <?php
2.  // urceni jmenneho prostoru
3.  namespace MVC;
4.  // import use
5.  use MVC\Router\RouterInterface;
6.  use Symfony\Component\HttpFoundation\Request;
7.  class FrontController
8.  {
9.      // deklarace promennych
10.     private $params = [];
11.     private $router;
```

```
12.     private $diContainer;
13.     // verejny konstruktor
14.     // vyzaduje na vstupu Router a Request
15.     // umoznuje vlozit vlastni DI kontejner
16.     public function __construct(RouterInterface $router, $request, $diConta-
    iner = null)
17.     {
18.         $this->router = $router;
19.         $this->diContainer = $diContainer;
20.         $this->route($request);
21.     }
22.     // metoda co vyhleda a zavola pozadovanou akci
23.     public function route(Request $request)
24.     {
25.         // zjisti nazev cesty podle informace z HTTP Request
26.         $name = $request->getPathInfo();
27.         // zjisti vsechny parametry predane v HTTP Request
28.         $this->params = $request->query->all();
29.         array_splice($this->params, 0, 1);
30.         // vyhleda v routeru spravnou cestu
31.         $route = $this->router->getMatch($name);
32.         // jestli pozadovana trida neexistuje, vyvola vyjimku
33.         $class = $route->getClass();
34.         if (!class_exists($class)) {
35.             throw new \LogicException("Class " . $class . " doesn't
    exists!");
36.         }
37.         // zajisti pozadovane parametry pro vytvoreni konstruktoru
38.         $reflection = new \ReflectionClass($class);
39.         $params = $this->prepareParams($reflection->getConstructor());
40.         // vytvori konstruktor
41.         $instance = $reflection->newInstanceArgs($params);
42.         // zjisti pozadovanou akci, nebo-li metodu
43.         // jestli metoda neexistuje, vyvola vyjimku
44.         $action = $route->getAction();
45.         if (!method_exists($instance, $action)) {
46.             throw new \BadFunctionCallException("Function " . $action . " do-
    esn't exists!");
47.         }
48.         // zajisti pozadovane parametry metody
49.         $methodReflector = new \ReflectionMethod($instance, $action);
50.         $params = $this->prepareParams($methodReflector);
51.         // zavola metodu prislusneho ovladace, se zadanymi parametry
52.         call_user_func_array(array($instance, $action), $params);
53.     }
54.     // metoda pro pripravu parametru
55.     private function prepareParams(\ReflectionMethod $method)
56.     {
```

```
57.         // zjisti pozadovane parametry a pocet pozadovanych parametru
58.         $parameters = $method->getParameters();
59.         $reqNum = $method->getNumberOfRequiredParameters();
60.         $retVal = [];
61.         // projde vsechny pozadovane parametry a zjisti jestli existuji v DI
    kontejneru nebo v parametrech predanych
62.         // v HTTP Request
63.         // jestli ne, vyhodi vyjimku
64.         for ($i = 0; $i < sizeof($parameters); $i++) {
65.             if (isset($this->diContainer[$parameters[$i]->getName()])) {
66.                 $retVal[] = $this->diContainer[$parameters[$i]->getName()];
67.             } elseif (isset($this->params[$parameters[$i]->getName()])) {
68.                 $retVal[] = $this->params[$parameters[$i]->getName()];
69.             } elseif ($i+1 <= $reqNum) {
70.                 throw new \InvalidArgumentException("Required parameters do-
    esn't match" .
71.                     " given parameters! Missing parameter: " . $parame-
    ters[$i]);
72.             }
73.         }
74.         return $retVal;
75.     }
76. }
```

4.3.4 Route

Jako další je ve jmenném prostoru MVC vložena třída Route. Ta nám reprezentuje definici cesty v našem frameworku. Má nastavené vlastnosti name, class a action, které se dají inicializovat v konstruktoru nebo v set metodách. Tyto vlastnosti poskytuje pomocí veřejných get metod.

4.3.5 Router

Třída router je také vložena ve jmenném prostoru MVC a jedinou proměnnou kterou spravuje, je privátní pole routes. Poskytuje metodu addRoute, která vstupní parametr typu Route nejprve odstraní z pole cest a poté opět vloží. Tímto se zajistí unikátnost hodnot. Podobně to bude řešeno v modulu Event Dispatcher.

Dále poskytuje metodu removeRoute, která podle jména vyhledá a vymaže příslušnou cestu. Jako poslední je zde metoda getMatch, která vyhledá ve všech cestách, jestli nějaká odpovídá vloženému parametru a jestli ano, vrátí jí. V případě že žádná cesta není vhodná, vyhodí Router výjimku, která informuje o tom, že cesta neexistuje.

Jelikož je naším záměrem, aby si uživatel mohl vytvořit vlastní implementace tříd Route a Router, je mu nabídnuto rozhraní, které musí jeho vlastní řešení implementovat. Tyto rozhraní implementují tedy i naše předdefinované třídy.

```php
<?php
// urceni jmenneho prostoru
namespace MVC\Router;
// rozsiruje rozhrani RouterInterface
class Router implements RouterInterface
{
    // deklarace promenne
    private $routes = [];
    // pridani nove cesty
    public function addRoute(RouteInterface $route)
    {
        // pred samotnym vlozenim nove cesty, zajisti aby v poli nebyla vicekrat, tim ze ji nejprve odstrani
        $this->removeRoute($route->getName());
        $this->routes[] = $route;
    }
    // odstraneni cesty
    public function removeRoute(String $name)
    {
        foreach ($this->routes as $index => $route) {
            if ($route->getName() === $name) {
                unset($this->routes[$index]);
            }
        }
    }
    // zjisti shodu a vrati, v pripade ze neexistuje, vyvola vyjimku
    public function getMatch(String $name): RouteInterface
    {
        foreach ($this->routes as $index => $route) {
            if ($route->getName() === $name) {
                return $route;
            }
        }
        throw new RouteNotFoundException("Route " . $name . " doesn't exists!");
    }
}
```

4.3.6 Abstraktní Controller

Každý uživatelem vytvořený Controller musí rozšiřovat abstraktní třídu Controller, která je v adresáři vendor/mvc/. Jedná se jednoduchou třídu, která umožňuje připravit proměnnou view a inicializuje ji jako instanci třídy Template.

```
1. <?php
2. // urceni jmenneho prostoru
3. namespace MVC;
4. use Models;
5. // abstraktni trida kontroler
6. abstract class Controller
7. {
8.     // deklarace promenne
9.     protected $view;
10.    // priprava view
11.    // zajisti aby view byla instance tridy Template
12.    // uzivatel ale i presto muze promennou view nastavit jako jakykoliv jiny
   soubor
13.    protected function setView()
14.    {
15.        $this->view = new Template();
16.    }
17. }
```

4.3.7 Template

Třída Template zastupuje v našem MVC frameworku prvek view. Její hlavní funkcí je možnost renderování souborů (.php, .html) definujících vzhled stránek, které se zobrazí uživateli. Nejdůležitější metodou v této třídě je tedy metoda render. Příjmá parametr viewName, který nám udává úplnou, nebo relativní cestu k souboru, branou z adresáře src/views/. Další parametry jsou status, potřebný pro sestavení HTTP response a booleanová hodnota fullPath, která určuje, zda se jedná o úplnou cestu či nikoliv.

Dále tato třída je implementací návrhového vzoru Template View. Uživatel může registrovat slovník hodnot nebo každou hodnotu samostatně, podle klíče a hodnoty. V html kódu se tyto hodnoty dají vyvolat tím, že se použijí tagy složených závorek, mezi které se vloží jejich název. V html kódu uvedeném níže je vidět využití speciálních tagů {} mezi které se napíše název proměnné. Ta se poté ve fázi renderování obsahu stránky nahradí za skutečné hodnoty. V tomto případě se do proměnných vypíše obsah připravených dat z databázové tabulky segment.

```
<h1>{title1}</h1>
<h2>{subtitle1}</h2>
<p>
  {content1}
</p>
```

Metoda render poté projde všechny výskyty těchto tagů a nahradí je skutečnými hodnotami (v případě že byly nejprve správně registrovány). Samotné nahrazení hodnot je provedeno metodou preg_replace, která je volána v cyklu foreach. Tento cyklus projde všechny zaregistrované hodnoty, vyhledá jejich výskyt v kódu a nahradí je skutečnými hodnotami.

```
<?php
// urceni jmenneho prostoru
namespace MVC;
// import use
use Symfony\Component\HttpFoundation\Response;
class Template
{
    // deklarace promenne
    private $vars = [];
    // funkce pro prirazeni vetsiho mnozstvi hodnot
    public function assignValues($vars)
    {
        array_push($this->vars, $vars);
    }
    // funkce pro prirazeni jedne hodnoty
    public function assignValue($key, $value)
    {
        $this->vars[$key] = $value;
    }
    // funkce, ktera vyhleda pozadovany soubor
    // pote v nem nahradi promenne ve specialnich znackach {} za prirazene hodnoty
    // vysledek vrati pomoci HTTP Response
    public function render($viewName, $status = 200, bool $fullPath = false)
    {
        // zjisti jestli je zvolena moznost plne cesty
        // potreba kdyz chce uzivatel vykreslovat pohledy z jine slozky nez src/views
        if ($fullPath) {
            $prefix = "";
        } else {
            $prefix = "src/views/";
        }
        // zjisti jestli soubor existuje, jestli ne, vyvola vyjimku
        if (file_exists($prefix . $viewName)) {
```

```
34.             // zajisti obsah stranky
35.             $content = file_get_contents($prefix . $viewName);
36.             // projde vsechny vyskyty hodnot ve specielnich znackach {} a na-
    hradi je za prirazene hodnoty
37.             // pomoci vyuziti regularnich vyrazu
38.             foreach ($this->vars as $key => $value) {
39.                 $content = preg_replace('/\{' . $key . '\}/', $value, $con-
    tent);
40.             }
41.             // pripravi a odesle HTTP Response
42.             $response = new Response(
43.                 'Content',
44.                 $status,
45.                 ['content-type' => 'text/html']
46.             );
47.             $response->setContent($content);
48.             $response->send();
49.         } else {
50.             throw new \LogicException("File not found!");
51.         }
52.     }
53. }
```

4.3.8 Modul Event Dispatcher

Další součástí jádra našeho frameworku je Event Dispatcher. Jedná se o jednoduchou implementaci toho nejdůležitějšího, definováno v PSR-14. Jak již bylo popsáno v podkapitole o PHP-FIG, Event Dispatcher je tvořen pěti částmi. V našem balíčku si vytvoříme pouze tři z nich. Část Emmiter je totiž jakýkoliv kód, ve kterém se budou naše události vyvolávat a Listener je ve zkutečnosti jakákoliv spustitelná část PHP kódu.

Jelikož PSR-14 definuje tři základní rozhraní, je nutné, abychom je implementovali. Vyřvořili jsme si tedy tyto rozhraní přesně podle poskytnutých šablon. Jedná se o StoppableEventInterface, EventDispatcherInterface a ListenerProviderInterface.

4.3.9 Event

Dále jsme si vytvořili třídu Event, která, jak již název napovídá, zastupuje část událost. Třídu jsme zařadili do jmenného prostoru EventDispatcher a přidali implementaci již vytvořeného rozhraní StoppableEventInterface. Dále jsme vytvořili tři privátní vlastnosti, name, parrams a propagationStopped, u které nastavíme výchozí hodnotu na false. Vlastnosti name a params umožníme nastavovat v konstruktoru a get a set metodách. Následně jsme vytvořili metodu

setPropagationStopped, ve které se nastavuje hodnota toho, jestli je šíření události zastavené. Dále je implementovaná metoda isPropagationStopped z interface, která pouze vrátí hodnotu s aktuálním nastavením šíření.

4.3.10 Listener Provider

Jako další jsme si vytvořili část Listener Provider. Jak je zmíněno v PSR-14, Event Dispatcher je zodpovědný za volání všech relevantních Listenerů, nicméně sám nemá rozhodovat o tom, kteří to jsou. K tomu je právě zapotřebí objekt Listener Provider. Vytvořili jsme si tedy jeho vlastní verzi, která implementuje rozhraní ListenerProviderInterface a spadá do jmenného prostoru EventDispatcher.

Dispatcher obsahuje jediný parametr, a to listeners neboli pole posluchačů. Tato třída poskytuje veřejné metody add, remove a getListenersForEvent. Metoda add přijímá parametry listener a event, které nejprve odstaní z pole posluchačů tím, že zavolá níže definovanou metodu remove (tím se zajistí unikátnost v poli). Následně se jednoduše vloží na konec nové asociativní pole, které nám spojuje právě tyto dva prvky. Metoda remove nám vyhledá v poli prvek, ve kterém souhlasí parametry event a listener a ty poté odstraní.

Jako poslední implementuje Dispatcher metodu getListenersForEvent, definovanou v rozhraní. Ta projde pole se všemi posluchači, zjistí, kteří jsou relevantní k danému objektu a vloží je všechny do nového pole retVal, které je návratovou hodnotou.

```
1.
2.  <?php
3.  // urceni jmenneho prostoru
4.  namespace EventDispatcher;
5.  // implementace rozhrani ListenerProviderInterface
6.  class ListenerProvider implements ListenerProviderInterface
7.  {
8.      // deklarace promenne
9.      private $listeners = [];
10.     // prida novou udalost a posluchace
11.     public function add($listener, Event $event)
12.     {
13.         // zajisti unikatnost v poli ti, ze nejdriv hodnotu odebere
14.         $this->remove($listener, $event);
15.         $this->listeners[] = [
16.             "event" => $event,
17.             "listener" => $listener
18.         ];
```

```
19.     }
20.     // odstrani udalost a posluchace
21.     public function remove($listener, Event $event)
22.     {
23.         foreach ($this->listeners as $index => $item) {
24.             if ($item["event"] == $event & $item["listener"] == $listener)
25.             {
26.                 unset($this->listeners[$index]);
27.             }
28.         }
29.     }
30.     // zjisti vsechny posluchace, kteri jsou k udalosti pripojeni
31.     public function getListenersForEvent(object $event) : iterable
32.     {
33.         $retVal = [];
34.         foreach ($this->listeners as $item) {
35.             if ($item["event"] == $event) {
36.                 array_push($retVal, $item["listener"]);
37.             }
38.         }
39.         return $retVal;
40.     }
41. }
```

4.3.11 Dispatcher

Na závěr jsme vytvořili samotnou třídu Dispatcher. Klasicky jsme jí zařadili do jmenného prostoru EventDispatcher a přidali implementaci příslušného rozhraní EventDispatcherInterface. Následně bylo nastaveno privátní pole listeners, které je datového typu ListenerProvider. Poté jsme si přidali metodu attach, která přijímá parametry event (datového typu Event) a listener (musí být callable). V té se volá metoda add z objektu listeners. Stejné to je i v metodě detach, pouze s tím rozdílem, že budeme volat metodu remove z objektu listeners.

Dále bylo nutné vytvořit metodu dispatch, definovanou v rozhraní. Ta přijímá jako vstupní parametr event, který musí být datového typu objekt. Zde se projdou všechny prvky kolekce vrácené v metodě getListenersForEvent, objektu listeners. Dále se zkontroluje, jestli není náhodou ukončena propagace události a jestli ne, zavolá se relevantní listener. Praktické užití modulu EventDispatcher si ukážeme v části věnované skeletonu aplikace.

```
1. <?php
2. // urceni jmenneho prostoru
3. namespace EventDispatcher;
4. // implementace rozhrani EventDispatcherInterface
```

```
5.  class Dispatcher implements EventDispatcherInterface
6.  {
7.      // deklarace promenne
8.      private $listeners;
9.      public function __construct()
10.     {
11.         $this->listeners = new ListenerProvider();
12.     }
13.     // prirazeni posluchace k udalosti
14.     public function attach(Event $event, callable $listener)
15.     {
16.         $this->listeners->add($listener, $event);
17.     }
18.     // odpoutani posluchace od udalosti
19.     public function detach(Event $event, callable $listener)
20.     {
21.         $this->listeners->remove($listener, $event);
22.     }
23.     // informovani posluchacu o spustene udalosti
24.     public function dispatch(object $event)
25.     {
26.         // kontrola jestli je zapnuta propagace
27.         foreach ($this->listeners->getListenersForEvent($event) as $listener)
    {
28.             if (!$event->isPropagationStopped()) {
29.                 call_user_func($listener);
30.             }
31.         }
32.     }
33. }
```

4.3.12 Modul App

Poslední částí našeho micro frameworku je modul App. Jeho hlavní funkcí je poskytnou možnost konfigurace MVC prvků v aplikaci vytvořené na našem frameworku. Obsahuje pouze dva soubory, App.php a error.html.

4.3.13 App

Třída App uživateli pouze zlehčuje používání komponenty MVC, která je ale plně využitelná i samostatně bez ní. Třídu jsme si vložili do jmenného prostoru App. Dále třída obsahuje privátní proměnné router, request a diContainer. Ve veřejném bezparametrickém konstruktoru si třída zkontroluje, jestli v adresáři config existuje soubor config.json. Užitvatel ho musí vložit přesně na toto místo.

V případě že soubor neexistuje, připraví konstruktor prázdný router a nový HttpFoundation\Request. Jestli ale naopak konfigurační soubor existuje, zavolá metodu prepare, která mu připraví všechny hodnoty. Ta převede všechny hodnoty v souboru config.json do asociativních polí, pomocí PHP příkazů file_get_contents a json_decode. Poté se podívá, jestli jsou v těchto polích obsažena klíčová slova, podle kterých se vyhledají hodnoty pro konfiguraci. K tomu jsou využity metody loadModules, loadRoutes, loadRequest a loadDiContainer.
V konfiguračním souboru je možné nastavovat následující hodnoty:

- **router** je cesta k uživatelem vytvořenému routeru, implementujícím RouterInterface.
- **routes**, kde jsou všechny cesty, které nejsou v žádných modulech.
- **modules**, která uvádí všechny moduly, které jsou v aplikaci obsažené.
- **diContainer**, cesta k uživatelem vytvořenému DI kontejneru.

Všechny cesty musí být ve formátu název cesty a poté asociativní pole s názvem controlleru (s plným jmenným prostorem) a názvem akce, kterou chceme volat. Stejným způsobem jsou definované cesty i v konfiguračních souborech balíčků. Ty musí být v adresáři src/config, musí se jmenovat nazevBalicku.config.json a příjmají jedinou hodnotu, a to routes.

Jestliže je zvolena možnost psaní cest přímo v konfiguračním souboru, projde metoda prepare všechny tyto hodnoty a postupně je vloží do nově vytvořené instance třídy Router. Ta se pak předá jako parametr při vytváření objektu FrontController. Ukázky konkrétního sestavení souboru config.json budou více upřesněny v kapitole o skeletu aplikace.

Dále je zde metoda run, ve které se vytvoří nová instance třídy FrontController. Do té se vloží všechny předem připravené hodnoty (router, request a diContainer). Celá tato operace, stejně jako volání metody prepare, je obaleno v bloku try a v případě jakékoliv výjimky se volá metoda handleException. V té je zajištěno, že se uživateli zobrazí zpráva a podrobné informace o výjimce na předem připravené stránce error.html.

Jak již bylo zmíněno, třída Router vyvolává námi definovanou výjimku RouteNotFoundException. Její výskyt se tedy ošetří jiným způsobem než výskyt jakékoliv jiné výjimky. Když Router nenajde žádnou cestu, je zřejmé, že uživatel zadal neplatný požadavek. Tudíž při sestavování HTTP response vložíme kód 404, značící Not Found.

```
1. <?php
2. // urceni jmenneho prostoru
3. namespace App;
4. // import use
5. use MVC\FrontController;
```

```
6.  use MVC\Router\Route;
7.  use MVC\Router\RouteNotFoundException;
8.  use MVC\Router\Router;
9.  use MVC\Template;
10. use Symfony\Component\HttpFoundation\Request;
11. class App
12. {
13.     // deklarace promennych
14.     private $router;
15.     private $request;
16.     private $diContainer;
17.     // konstruktor, ktery priradi zjisti jestli existuje soubor config a
    jestli ne, vytvori nove instance
18.     // router a request
19.     // dale kontroluje jestli pri se pri prirazeni promennych nevyvolaji vy-
    jimky
20.     public function __construct()
21.     {
22.         try {
23.             if (!file_exists("config/config.json")) {
24.                 $this->router = new Router();
25.                 $this->request = Request::createFromGlobals();
26.             } else {
27.                 $this->prepare();
28.             }
29.         }
30.         catch (\Exception $e) {
31.             $this->handleExceptions($e);
32.         }
33.     }
34.     // vytvori novou instanci front controlleru
35.     // zachyti vsechny vyjimky, ktere budou vyvolane pri behu front controll-
    eru
36.     public function run()
37.     {
38.         try {
39.             $frontController = new FrontController($this->router, $this-
    >request, $this->diContainer);
40.         }
41.         catch (RouteNotFoundException $e) {
42.             $this->handleExceptions($e, 404);
43.         }
44.         catch (\Exception $e) {
45.             $this->handleExceptions($e);
46.         }
47.     }
48.     // vsechny vyvolane vyjimky zachyti a zobrazi v instanci tridy Template
49.     private function handleExceptions(\Exception $e, $status = 500)
```

```
50.     {
51.         if ($this->diContainer != null) {
52.             if (isset($this->diContainer["logger"])) {
53.                 $this->diContainer["logger"]->critical($e);
54.             }
55.         }
56.         $template = new Template();
57.         $template->assignValue("exception", $e->getMessage());
58.         $template->assignValue("content", $e->getTraceAsString());
59.         $template->render("vendor/filarichard/mvc_micro_framework/app/er-
    ror.html", $status,true);
60.     }
61.     // priprava routeru, cest, requestu a DI kontejneru
62.     private function prepare()
63.     {
64.         $jsonFile = file_get_contents("config/config.json");
65.         $config = json_decode($jsonFile, true);
66.         if (array_key_exists("router", $config)) {
67.             $this->router = include $config["router"];
68.         } else {
69.             $this->router = new Router();
70.             $this->loadModules($config);
71.             $this->loadRoutes($config);
72.         }
73.         $this->loadRequest($config);
74.         $this->loadDiContainer($config);
75.     }
76.     // zjisti vsechny moduly definovane v konfiguracnim souboru
77.     // pote vyhleda konfiguracni soubor kazdeho modulu a vlozi vsechny cesty
    ktere jsou v nem ulozene
78.     private function loadModules($config)
79.     {
80.         if (array_key_exists("modules", $config)) {
81.             foreach ($config["modules"] as $module) {
82.                 if (file_exists("src/config/" . $module . ".config.json")) {
83.                     $jsonFile = file_get_contents("src/config/" . $module .
    ".config.json");
84.                     $moduleConfig = json_decode($jsonFile, true);
85.                     if (array_key_exists("routes", $moduleConfig)) {
86.                         foreach ($moduleConfig["routes"] as $name => $route)
    {
87.                             $this->router->addRoute(new Route($name,
    $route["controller"], $route["action"]));
88.                         }
89.                     }
90.                 } else {
91.                     throw new \LogicException("Cannot find config file!");
92.                 }
```

```
93.             }
94.         }
95.     }
96.     // nahraje vsechny cesty ulozene v zakladnim konfiguracnim souboru
97.     // nejsou v zadnem modulu
98.     private function loadRoutes($config)
99.     {
100.            if (array_key_exists("routes", $config)) {
101.                foreach ($config["routes"] as $name => $route) {
102.                    $this->router->addRoute(new Route($name, $route["con-
    troller"], $route["action"]));
103.                }
104.            }
105.        }
106.        // nahraje uzivatelem vytvoreny request, ktery neni v DI kontejneru
107.        private function loadRequest($config)
108.        {
109.            if (array_key_exists("request", $config)) {
110.                $this->request = include $config["request"];
111.            }
112.        }
113.        // nahraje DI kontejner a pripadne nahraje request z DI kontejneru
114.        private function loadDiContainer($config)
115.        {
116.            if (array_key_exists("diContainer", $config)) {
117.                $this->diContainer = include $config["diContainer"];
118.                if ($this->request === null && isset($this->diConta-
    iner["request"])) {
119.                    $this->request = $this->diContainer["request"];
120.                } else {
121.                    $this->request = Request::createFromGlobals();
122.                }
123.            }
124.        }
125.    }
```

4.3.14 Composer

Do konzole programu PhpStorm jsme napsali příkaz composer init, který nám pomohl při vytvoření souboru composer.json. V tuto chvíli se jedná o první výskyt composeru v naší praktické části. Inicializaci composeru jsme provedli s následujícími hodnotami.

```
Composer Init
C:\xampp\htdocs\mvc_micro_framework>composer init
Welcome to the Composer config generator
This command will guide you through creating your composer.json config.
Package name (<vendor>/<name>) [dell/mvc_micro_framework]: filarichard/mvc_micro_framework
Description []: Simple MVC Micro Framework
Author [Fila <fila.richard@gmail.com>, n to skip]: n
Minimum Stability []:
Package Type (e.g. library, project, metapackage, composer-plugin) []: project
License []: MIT
Define your dependencies.
Would you like to define your dependencies (require) interactively [yes]? no
Would you like to define your dev dependencies (require-dev) interactively [yes]? no
{
    "name": "filarichard/mvc_micro_framework",
    "description": "Simple MVC Micro Framework",
    "type": "project",
    "license": "MIT",
    "require": {}
}
Do you confirm generation [yes]? y
```

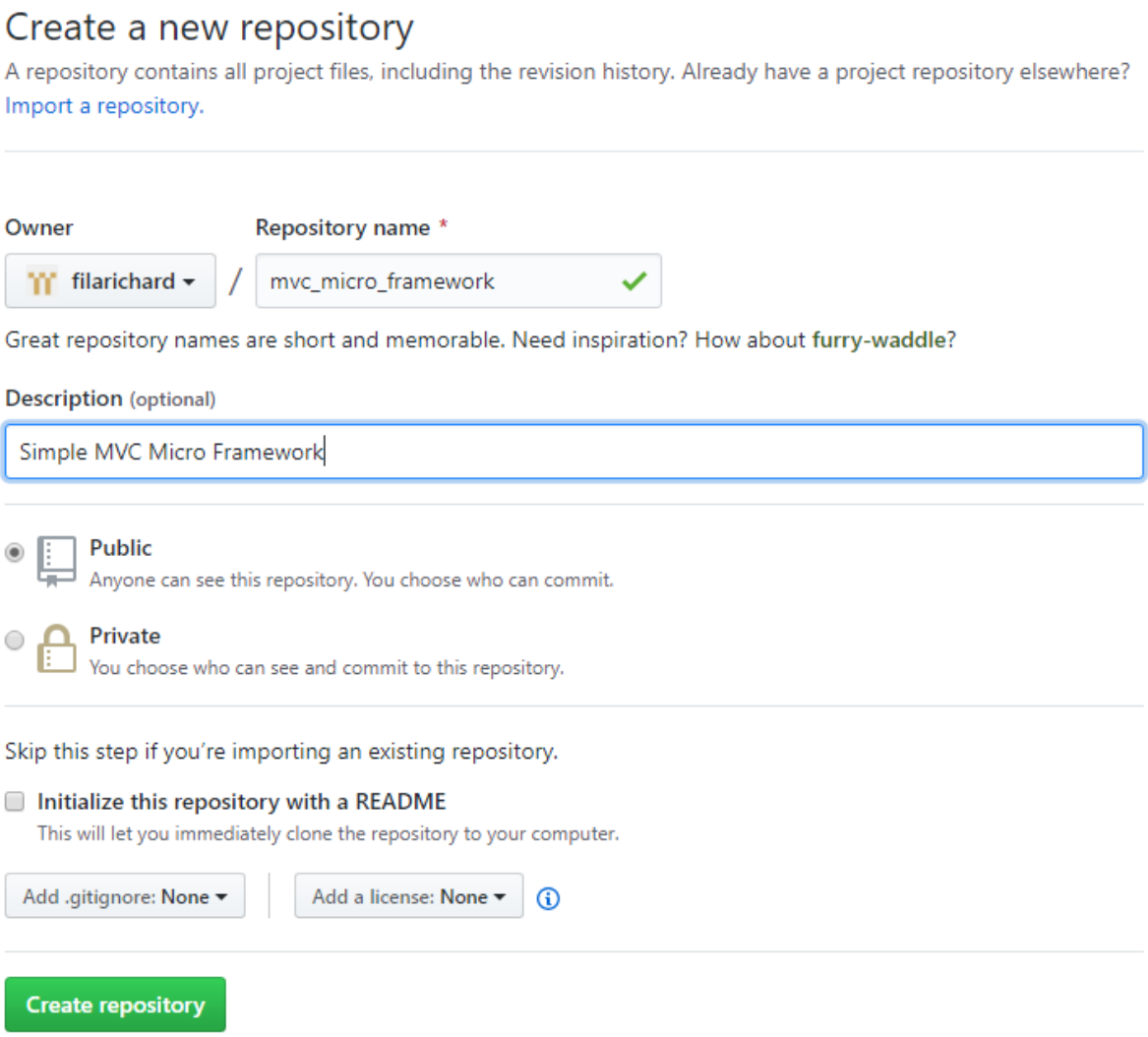
V následujícím bloku kódu je vidět finální podoba našeho souboru composer.json. Na konec jsme připsali hodnoty pro správný chod autoloadingu. Tuto vlastnost jsme si zajistili implementací psr-4, do které jsme vypsali jmenné prostory App, MVC a EventDispatcher. Dále jsme za položku autoload přidali require, kterým jsme si zajistili správné načítání komponent třetích stran. V našem případě využíváme následující komponenty:

- **illuminate/database** je databázovou komponentou frameworku Laravel.
- **pimple/pimple** je dependency injection kontejner.
- **monolog/monolog** je loggovací nástroj.
- **symfony/http-foundation** je komponentu z frameworku Symfony, která zjednodušuje http komunikaci.

```
{
    "name": "filarichard/mvc_micro_framework",
    "description": "Simple MVC Micro Framework",
    "type": "project",
    "license": "MIT",
    "autoload": {
        "psr-4": {
            "App\\":"app",
            "MVC\\": "mvc",
            "EventDispatcher\\": "eventDispatcher"
        }
    },
    "require": {
        "illuminate/database": "^5.8",
        "pimple/pimple": "^3.0",
        "monolog/monolog": "^1.24",
        "symfony/http-foundation": "^4.3"
    }
}
```

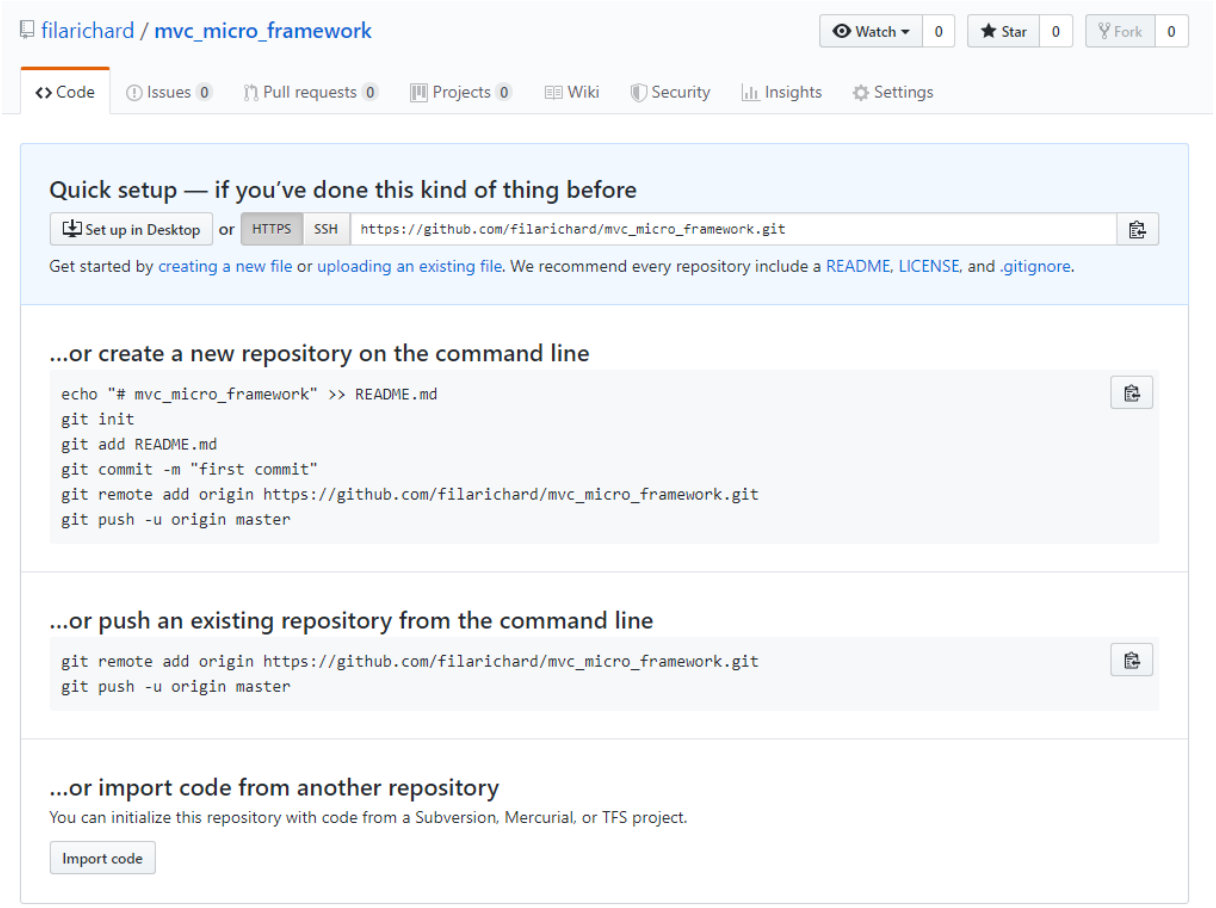
4.3.15 GitHub a Packagist

Tato podkapitola se zabývá vytvořením GitHub repozitáře a balíčku pro Composer, založeném na portále Packagist. Nejprve se podíváme na komunitu GitHub, kterou v tuto chvíli využijeme pouze jako nástroj pro zveřejnění našeho projektu a poskytnutí jeho kódu veřejnosti. Nicméně hlavní vlastností GitHubu je možnost správy verzí (VCS). Na úvodní stránce jsme si založili nový repozitář s názvem filarichard/mvc_micro_framework, který jsme vyplnili následujícím způsobem (Obrázek 24).



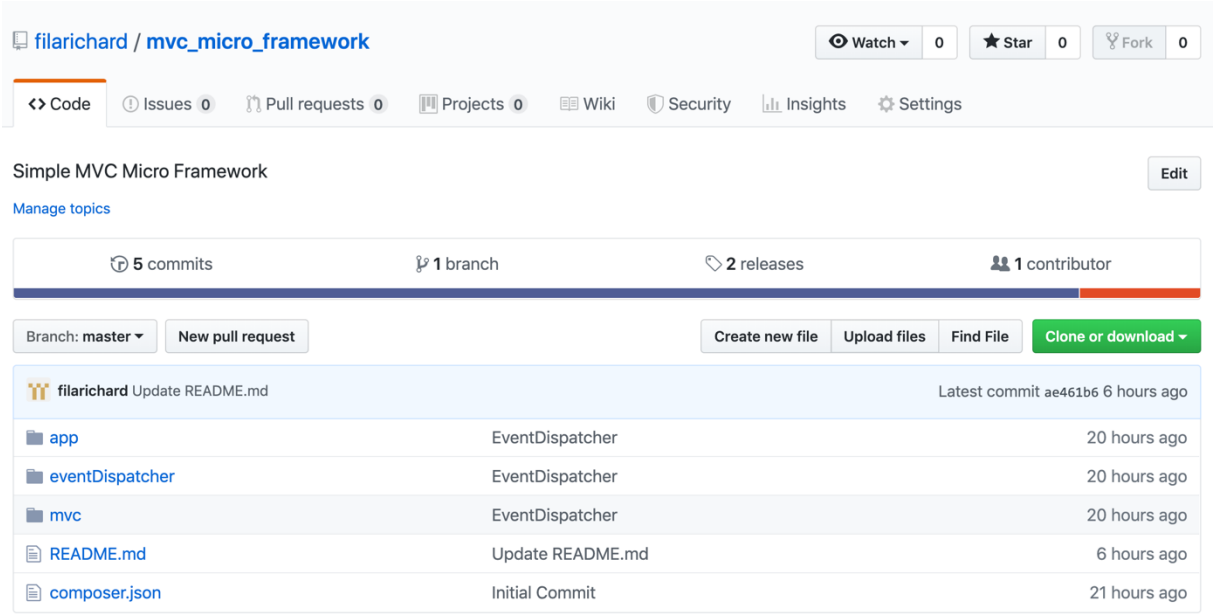
Obrázek 24 - Založení repozitáře na GitHub

Poté se nám zobrazila stránka, která potvrdila správné vytvoření repozitáře. Její snímek je na obrázku 25. Dále jsou na ní uvedené doporučené způsoby pro nahrání kódu.



Obrázek 25 - Úspěšné založení repozitáře na GitHub

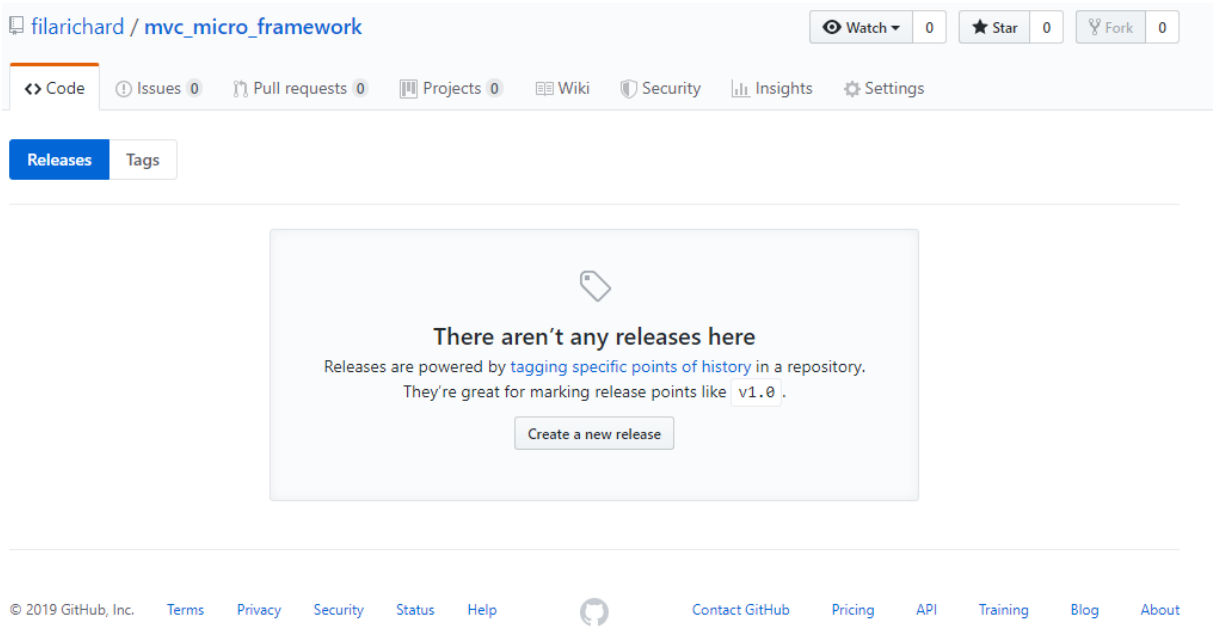
Když jsme provedli první commit and push (potvrzení a nahrání kódu na GitHub), objevila se nám nová stránka s podrobnými informacemi o aktuálním nahrání (Obrázek 26). Je zde kompletní kód, který je možné přímo v prohlížeči prohlížet, nikoliv však upravovat.



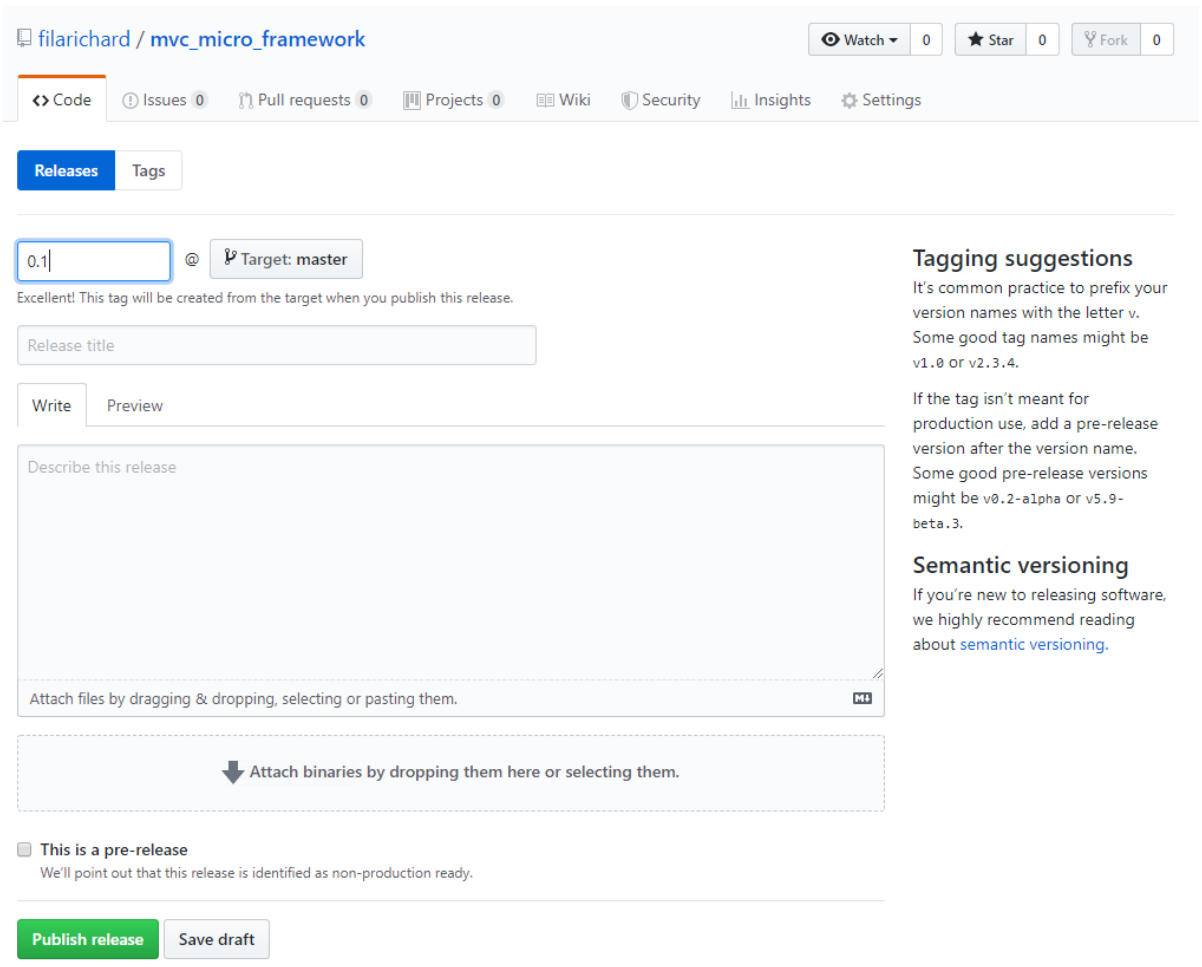
Obrázek 26 - Repozitář na GitHub po nahrátí zdrojových kódů

V horní části obrazovky jsou vidět záložky spojené s nově vytvořenou komunitou. Právě zde mají uživatelé možnost přidávat své názory a hlásit případné nedokonalosti či chyby na které narazili. Dále je zde možné vyčíst několik statistik o užívání našeho kódu.

Přímo nad kódem jsou čtyři záložky, commits, branch, releases a contributor. Nás v tuto chvíli zajímá pouze záložka releases neboli vydání. Právě takové vydání je totiž zapotřebí, aby bylo možní nahrát naši aplikaci na Packagist. Po kliknutí na záložku releases se nám zobrazila nová obrazovka, na které je možné vytvořit nové vydání našeho kódu. V tomto případě jsme si vystačili akorát s číslem vydání naší aplikace. Naše číslování jsme začali na 0.1. Celý tento postup je zobrazen na obrázku 27 a obrázku 28.



Obrázek 27 - Vytvoření vydání, část 1.

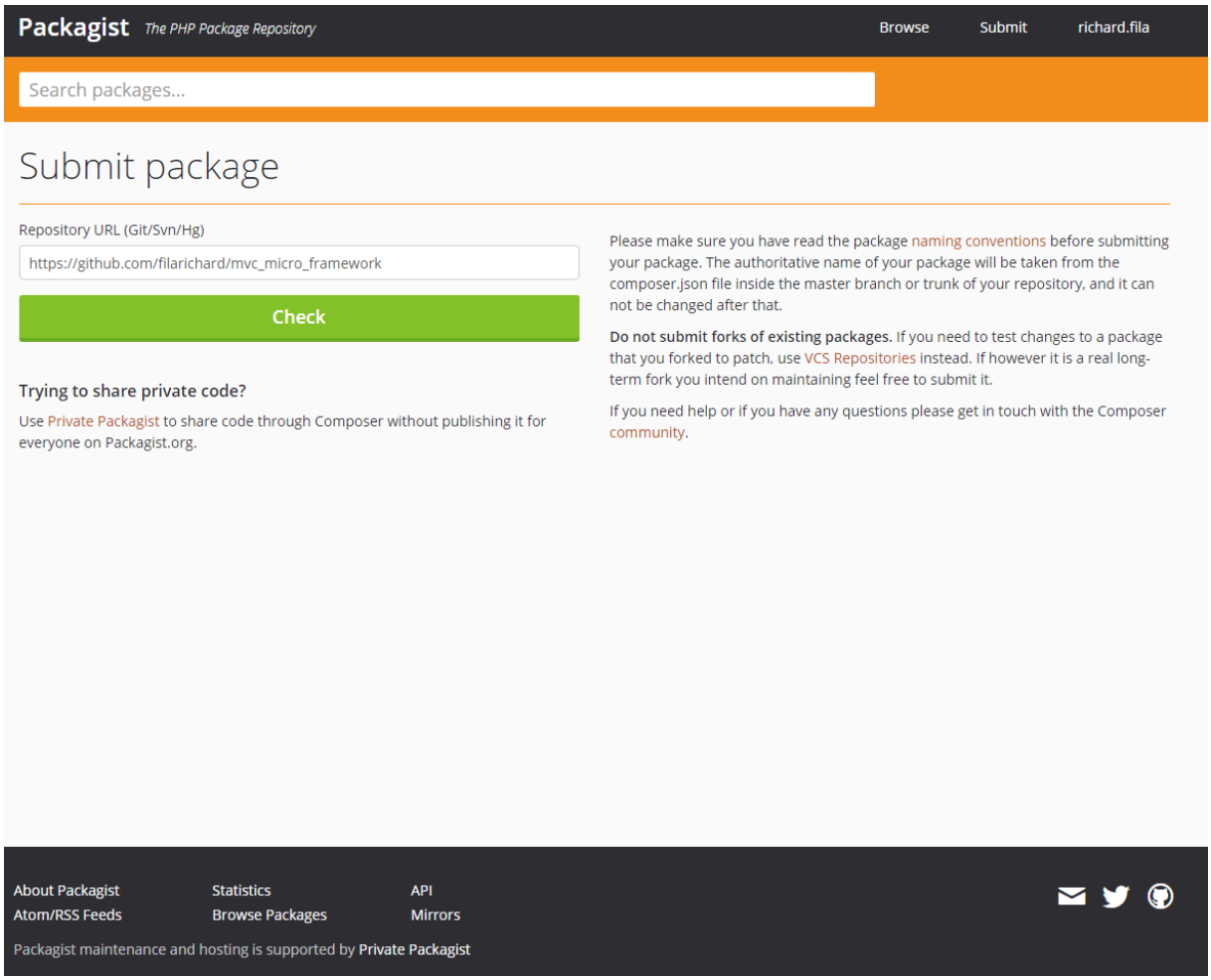


Obrázek 28 - Vytvoření vydání, část 2.

Na závěr jsme ještě přidali soubor README.md, což je soubor obsahující takzvané „čti mě“ informace. Pro jeho stestavení se využívá speciálního formátování Markup Language. V tomto souboru jsou obsaženy základní informace týkající se instalace našeho frameworku. Dále je zde uvedena základní sada příkazů, které jsou v něm umožněny.

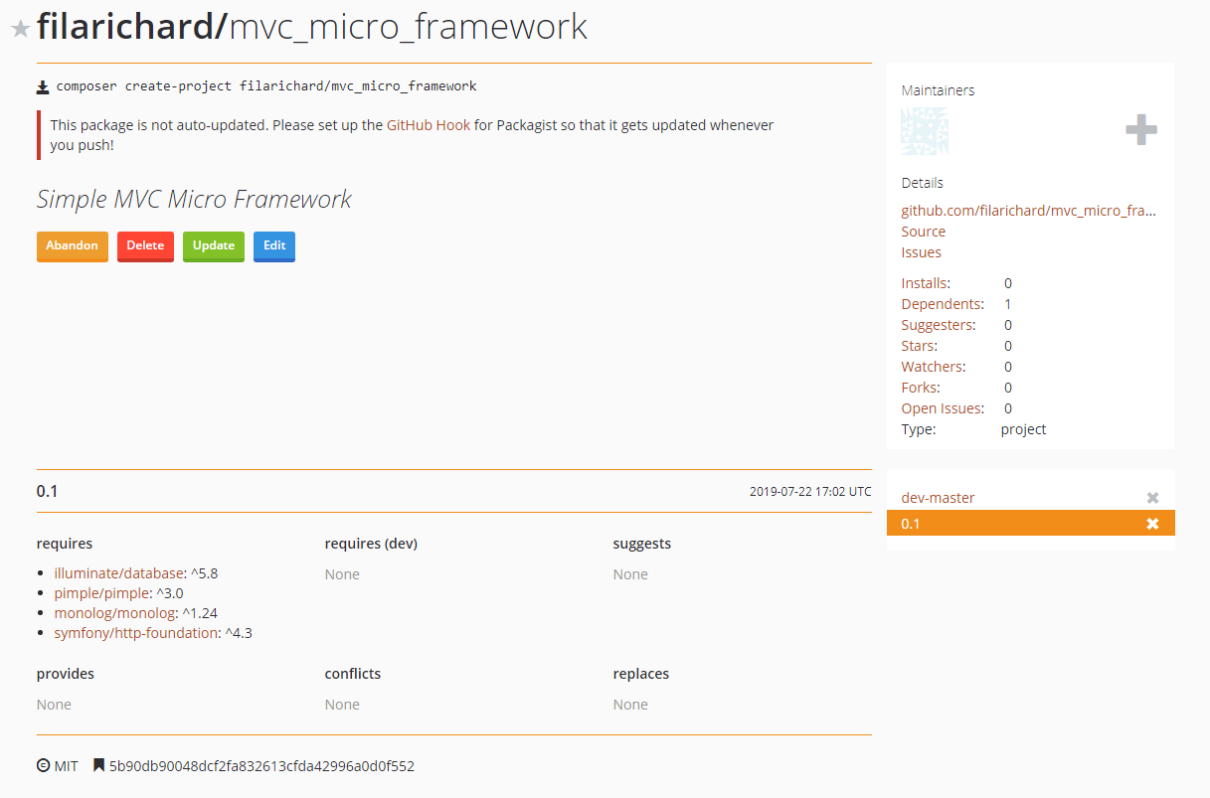
V tuto chvíli jsme s GitHubem hotovi a můžeme se podívat na Packagist. Jak již bylo zmíněno, jedná se o balíčkovací nástroj, díky kterému je možné jednoduše poskytnou vlastní kód široké veřejnosti jednoduchou formou. Uživatel pak pár jednoduchými příkazy dokáže zakomponovat jakýkoliv zveřejněný balíček do své vlastní aplikace. Stejného principu využívají PHP frameworky.

Po přihlášení na stránce jsme se přesunuli do podstránky submit. Zde jsme vložili odkaz na náš repozitář na GitHubu a stiskli tlačítko Check. Poté co Packagist zkontroloval jestli je daný repozitář v pořádku a zjistil, jestli obsahuje validní composer.json (z toho důvodu jsme použili příkaz composer init), nám nabídl vytvoření nového balíčku. Vše je znázorněno na obrázku 29.



Obrázek 29 - Nahrávání repozitáře na Packagist

Po úspěšném nahrání se nám zobrazila stránka s aktuálními detaily o našem balíčku (Obrázek 30). Na ní je ukázáno, jaké závislosti náš balíček obsahuje, aktuální vydání na GitHubu a v pravé části obrazovky je znázorněna statistika a základní informace o našem projektu. Jako například adresa GitHub repozitáře, počet instalací a další. Nejdůležitější informace se ovšem nachází v horní části stránky, kde je vypsán příkaz, díky kterému bude možné náš framework nainstalovat jako projekt.

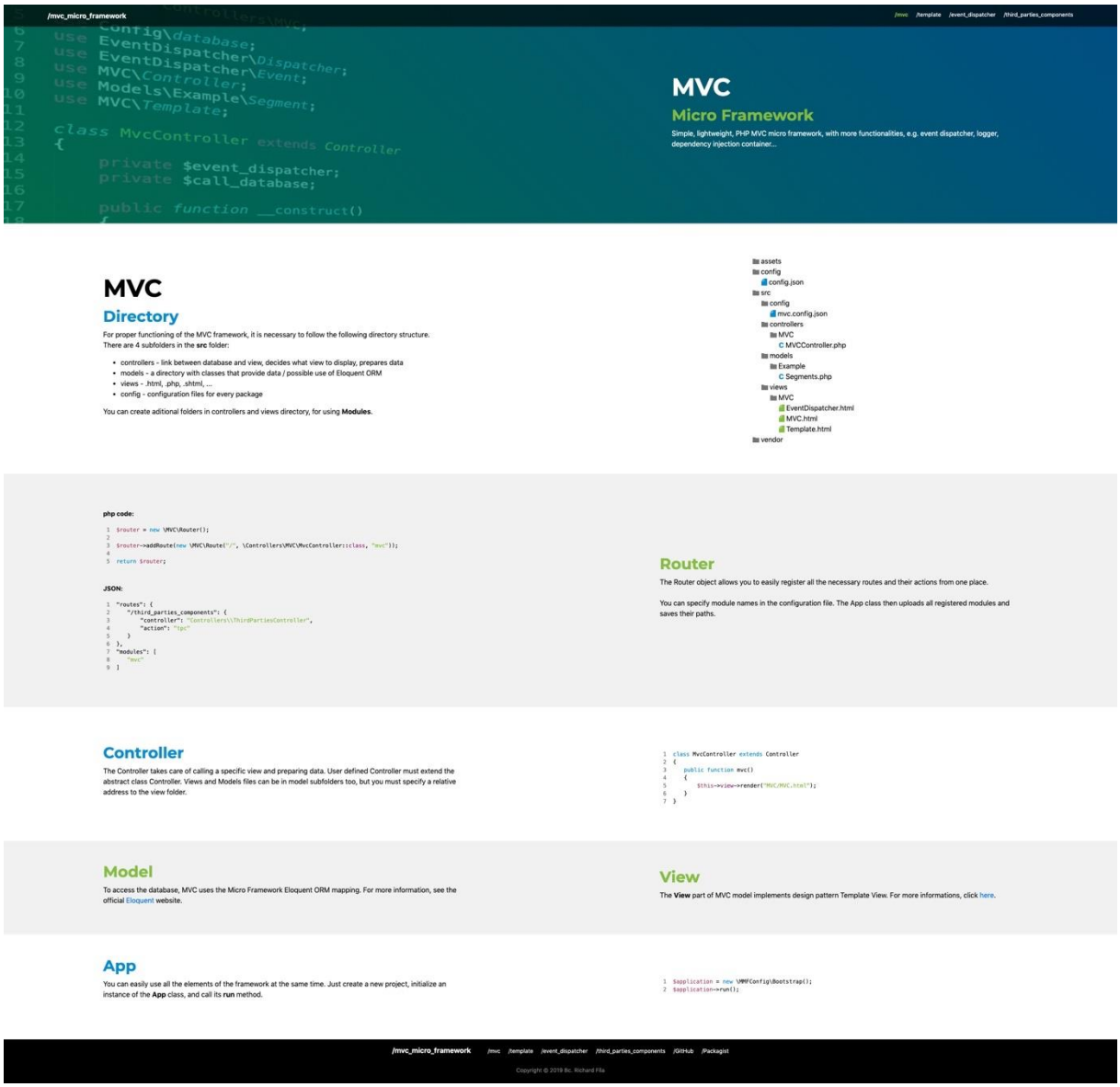


Obrázek 30 - Úspěšně vytvořený balíček na Packagist

4.4 Skeleton aplikace

Další nabízenou funkcí našeho frameworku je možnost stáhnutí předpřipravené aplikace s kompletní ukázkou funkčnosti. Všechny předem připravené třídy a soubory si může uživatel jednoduše smazat a nahradit vlastními soubory, případně přepsat pouze obsah těchto tříd.

Základní funkce tohoto skeletu je sestavení informačních stránek, které uživateli ukáží všechny poskytované funkce. Jsou zde tedy čtyři hlavní stránky mvc, template, event_dispatcher a thir_parties_components. Dále jsou zde odkazy na námi vytvořené komunity na Githubu a Packagistu. Náhled webové stránky je na obrázku 31.



Obrázek 31 - Vzhled úvodní stránky skeletonu aplikace

4.4.1 **composer.json**

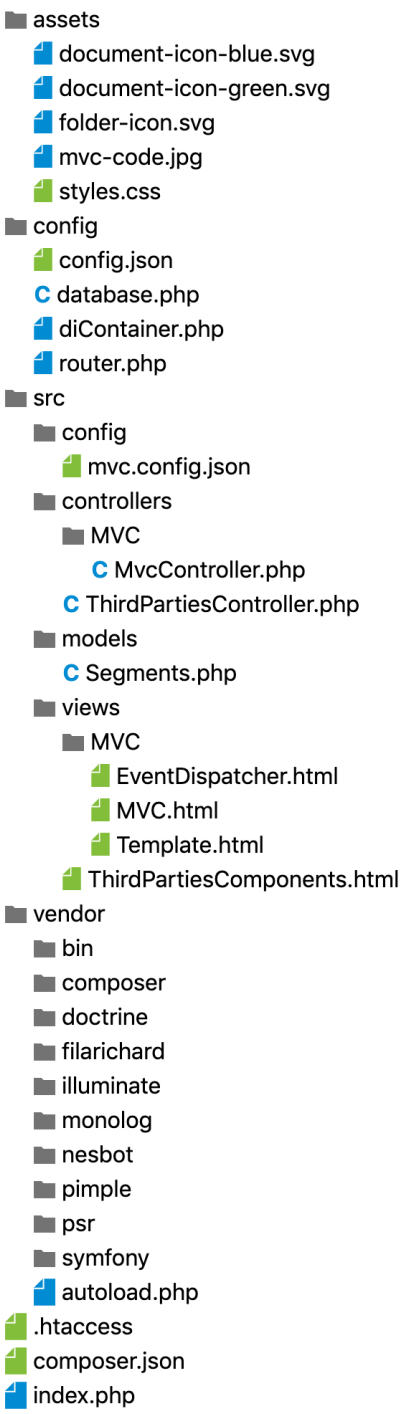
V tomto skeletu si stejným způsobem jako jsme postupovali při tvorbě frameworku vytvoříme nový soubor composer.json. Do něj vložíme pouze jedinou závislost, a to odkaz na námi vytvořený mikro framework.

Dále se pomocí psr-4 vytvoří jmenné prostory Controllers, Models a Config. Tímto se zajistí správné načítání struktury MVC a konfiguračních souborů. Jelikož soubory složky View nejsou PHP třídy a s velikou pravděpodobností se bude jednat o soubory typu html, není potřeba pro ně vytvářet vlastní jmenný prostor. Obsah souboru composer.json vypadá následovně.

```
{
    "name": "filarichard/mmf_skeleton",
    "description": "Full implementation of MVC Micro Framework",
    "type": "project",
    "license": "MIT",
    "autoload": {
        "psr-4": {
            "Controllers\\":"src/controllers",
            "Models\\": "src/models",
            "Config\\": "config"
        }
    },
    "require": {
        "filarichard/mvc_micro_framework": "^0.4"
    }
}
```

4.4.2 Adresář

Na obrázku 32 je znázorněn kompletní adresář všech souborů obsažených ve skelenotu našeho mikro frameworku. Ve složce assets jsou vloženy všechny soubory potřebné k správnému vykreslování .html stránek. Dále ve složce config je hlavní konfigurační soubor a poté soubory nastavující připojení k databáze, DI kontejner a vlastní router.



Obrázek 32 - Adresář skeletonu aplikace

V celém projektu je použit jeden ukázkový modul, s názvem MVC. Pro jeho správné nastavení byly vytvořeny složky MVC v adresářích controllers a views. Dále se ve složce config vytvořil soubor mvc.config.json, který obsahuje všechny cesty v tomto balíčku. V centrálním souboru config.json je vypsán v poli modules název modulu MVC. V následujících blocích jsou ukázky kódu z konfiguračních souborů, jak z centrálního, tak z balíčku.

Konfigurační soubor celé aplikace:

```
{
  "routes": {
    "/thirdPartiesComponents": {
      "controller": "Controllers\\ThirdPartiesController",
      "action": "tpc"
    }
  },
  "modules": [
    "mvc"
  ],
  "diContainer": "config/diContainer.php"
}
```

Konfigurační soubor balíčku:

```
{
  "routes": {
    "/": {
      "controller": "Controllers\\MVC\\MvcController",
      "action": "mvc"
    },
    "/mvc": {
      "controller": "Controllers\\MVC\\MvcController",
      "action": "mvc"
    },
    "/template": {
      "controller": "Controllers\\MVC\\MvcController",
      "action": "template"
    },
    "/eventDispatcher": {
      "controller": "Controllers\\MVC\\MvcController",
      "action": "dispatcher"
    }
  }
}
```

4.4.3 .htacces a index.php

Soubor .htaccess je přídavný konfigurační soubor, pomocí kterého si může uživatel upravovat některé vlastnosti serveru, aniž by měl přímý přístup k httpd.conf. Do něj by měl mýt správně přístup pouze správci serveru.

Do .htaccess napíšeme následující řádky:

```
1. RewriteEngine On
2.
3. RewriteBase /
4.
5. RewriteCond %{REQUEST_FILENAME} !-d
6. RewriteCond %{REQUEST_FILENAME} !-f
7.
8. RewriteRule ^(.+)$ index.php?url=$1 [QSA,L]
```

První příkaz slouží ke spuštění funkce Mod_rewrite. Jedná se o mocný nástroj, který umožňuje práci s URL adresami (přesměrování, atd.), se kterým je ale nutné zacházet opatrně. Při jeho používá se dá velice snadno dopustit chyby. Častým problémem je riziko zacyklení.

Druhý a třetí příkaz slouží k určení, jestli se náhodou nejedná o skutečné adresáře (!-d), nebo soubory (!-f). Pakliže podmínka je podmínka splněna, příkaz RewriteRule přesměruje požadavek (regulárním výrazem ^(.+)$ se vybere vše) do souboru index.php, jako parametr url. Příznak QSA zajišťuje že vše co bylo v původním požadavku za znakem ?, bude stejně předáno do přesměrovaného adresy. Dále příznak L určuje že se jedná o poslední pravidlo pro přepis a server tedy další pravidla bude ignorovat.

4.4.4 Views

Jako další jsme si připravili pohledovou složku našeho MVC návrhu. Jedná se o čtyři html stránky, které se renderují pomocí třídy Template. Jejich názvy odpovídají názvům metod v našich ovladačích. MVC.html, Template.html a EventDispatcher.html jsme uložili do adresáře src/views/MVC/ a ThirdPartiesComponents.html do adresáře src/views/ThirdParties/. Všechny jejich zdrojové kódy jsou obsaženy v přiložených souborech.

Při tvorbě těchto stránek byla použita sada nástrojů Bootstrap, převážně její Grid Systém. Aby se zajistila správná struktura webové aplikace, všechny soubory potřebné pro vykreslení stránek jsme uložili do adresáře assets.

4.4.5 Models

Složku Models v našem MVC struktuře zastupuje jakákoliv třída, která připravuje data a je využívána aktuálním ovladačem. V našem skeletu aplikace je využita pouze jediná třída, a to Segment. Jedná se o třídu sloužící k ORM mapování z databáze. Při tomto ORM mapování se využívá Eloquent z frameworku Laravel. Než si ukážeme, jak se taková třída píše, vytvoříme si její ekvivalent jako tabulku v databázi.

K tomuto účelu použijeme PhpMyAdmin, který je lehce dostupný a připravený v nástroji XAMPP. Na jeho úvodní stránce je možné si založit novou databázi. My si vytvoříme databázi s názvem test. Poté si vytvoříme jednoduchou tabulku, která obsahuje pouze pět parametrů. Automaticky se zvyšující primární klíč a poté parametry name, title, subtitle a content. Když tuto tabulku vytvoříme, naplníme si ji daty. PhpMyAdmin umožňuje také zadávat SQL příkazy. V přiložených souborech je soupis všech SQL příkazů potřebných k vytvoření této tabulky a jejímu následnému naplnění daty.

Aby bylo možné k této databázi přistupovat z PHP kódu, bylo potřeba vytvořit databázové připojení. K tomu jsme využili práve Eloquent a námi vytvořenou třídu dabase, kterou jsme uložili do adresáře config/. Celá tato třída implementuje návrhový vzor jedináček, který jsme zajistili tím, že jsme vytvořili pouze privátní konstruktor a k instanci se přistupuje statickou metodou.

V konstruktoru jsme si vytvořili proměnnou capsule, která je databázovým manažerem Eloquentu. Dále jsme jí nastavili konfiguraci pro připojení. Záměrně bylo vybráno tovární nastavení připojení.

Jako poslední jsme si vytvořili samotný soubor Segment.php. Uložili jsme si ho do jmenného prostoru Models\Example\ a do adresáře src/models/Example. Dále jsme přidali třídě dědičnost od třídy Illuminate\Database\Eloquent\Model. Následně jsme nastavili vyplnitelné položky, což byly v našem případě name, title, subtitle a content. Konkrétní užití volání této tabulky je ukázáno v následující podkapitole.

```
1.  <?php
2.
3.  namespace Models;
4.
5.  use Illuminate\Database\Eloquent\Model as Eloquent;
6.
7.  class Segment extends Eloquent
8.  {
9.      protected $fillable = [
10.         'name', 'title', 'subtitle', 'content'
11.     ];
12. }
```

4.4.6 Controllers

Dále jsme si připravili ovladače. V naší aplikaci byly využity pouze dva. Jeden na ukázku funkcí obsažených v našem frameworku (MvcController) a druhý na zobrazení balíčků třetích stran (ThirPartiesController).

4.4.7 MvcController

MvcController je ovladač, který připraví ukázku všech tří základních vlastností našeho MVC Micro Frameworku. Implementaci návrhových vzorů MVC s možností routování a Front Controlleru, Template Engine implementující návrhový vzor Template View a Event Dispatcher, který implementuje návrhový vzor Observer. Ovladač jsme si vložili do jmenného prostoru Controllers\MVC a do adresáře src/controllers/MVC/. Dále jsme našemu ovladači přidali dědičnost abstraktní třídy Controller.

```
1.  <?php
2.  // urceni jmenneho prostoru
3.  namespace Controllers\MVC;
4.  // import use
5.  use Config\Database;
6.  use EventDispatcher\EventDispatcherInterface;
7.  use EventDispatcher\StoppableEventInterface;
8.  use MVC\Controller;
9.  use Models\Segment;
10. // rozsireni abstraktni tridy Controller
11. class MvcController extends Controller
12. {
13.     // deklarace promennych
14.     private $eventDispatcher;
15.     private $callDatabase;
16.     // konstruktor, ve kterem se nastaví Event Dispatcher
17.     public function __construct(EventDispatcherInterface $eventDispatcher,
    StoppableEventInterface $event)
18.     {
19.         // pripraveni view, timto se da abstraktnimu controlleru vedet, ze
    chceme vyuzivat predpriparveneho
20.         // vzoru Template View
21.         $this->setView();
22.         $this->eventDispatcher = $eventDispatcher;
23.         $this->callDatabase = $event;
24.         $this->eventDispatcher->attach($this->callDatabase, array($this,
    'initializeDB'));
25.     }
```

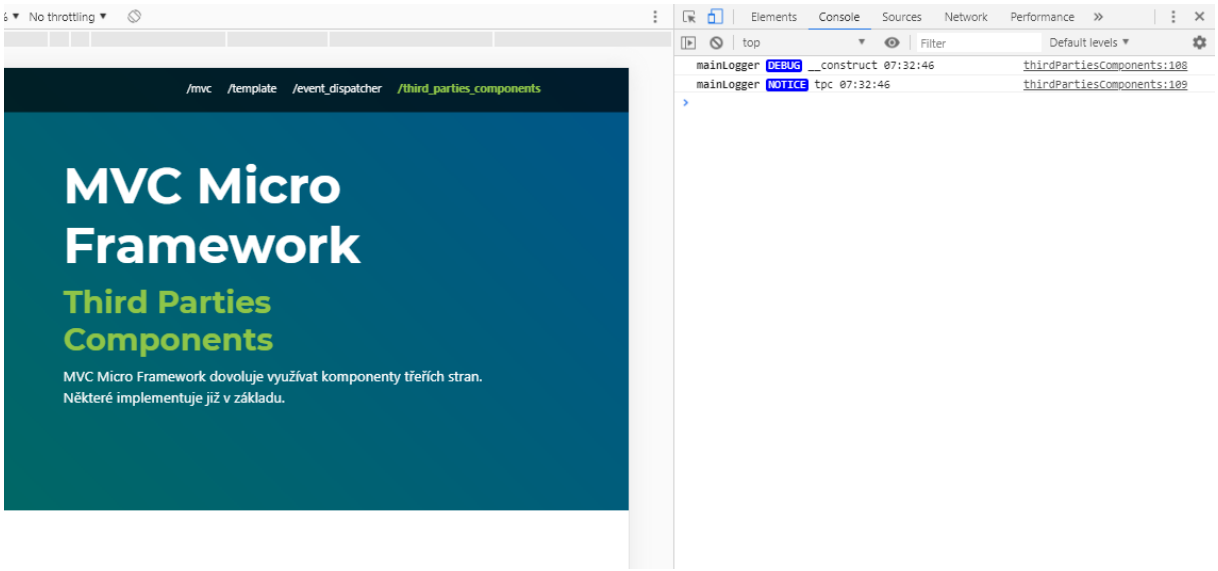
```
26.     // nasledujici funkce jsou volany jako akce routy
27.     public function mvc()
28.     {
29.         $this->view->render("MVC/MVC.html");
30.     }
31.     public function template()
32.     {
33.         // vyvolani akce, ktera pripravi databazi
34.         $this->eventDispatcher->dispatch($this->callDatabase);
35.         // odkomentovat ve chvily kdy je pripravena databaze
36.         // vyhledani dvou prvku v databazi
37. //          $first = Segment::where('name', 'template1')->first();
38. //          $second = Segment::where('name', 'template2')->first();
39.         // zakomentovat po pripraveni databaze
40.         $first = new class {
41.             public $title = "Template";
42.             public $subtitle = "MVC View";
43.             public $content = "View functionality in our MVC Micro Framework
    is backed by the Template class
44.             (Template View design pattern). When creating a view, the varia-
    ble name will be placed in curly brackets
45.             where we want to display values from the logical part of the ap-
    plication. For example {value}. These values
46.             must be correctly assigned in the controller.";
47.         };
48.         $second = new class {
49.             public $title = "";
50.             public $subtitle = "Inserting values";
51.             public $content = "Each segment on these example pages contains a
    paragraph or a first and second order
52.             heading. A template system is used on this page replacing the
    Template tags with associated values.";
53.         };
54.         // prirazeni hodnot prvku z databaze do NV Template View
55.         $this->view->assignValue("title1", $first->title);
56.         $this->view->assignValue("subtitle1", $first->subtitle);
57.         $this->view->assignValue("content1", $first->content);
58.         $this->view->assignValue("title2", $second->title);
59.         $this->view->assignValue("subtitle2", $second->subtitle);
60.         $this->view->assignValue("content2", $second->content);
61.         $this->view->render("MVC/Template.html");
62.     }
63.     public function dispatcher()
64.     {
65.         $this->view->render("MVC/EventDispatcher.html");
66.     }
67.     // metoda pro ziskani pripojeni k databazi
68.     public function initializeDB()
```

```
69.     {
70.         Database::GetInstance();
71.     }
72. }
73.
```

4.4.8 ThirdPartiesController

Další ovladač se stará o zobrazení využívaných komponent třetích stran. Stejně jako u MvcControlleru jsme přidali rozšíření abstraktní třídy Controller a uložili ho do jmenného prostoru. Tentokrát to byl Controllers\ a adresář src/controllers/.

Tento controller má pouze dvě metody. První je konstruktor, který nám připraví loggovací nástroj Monolog. Druhou metodou je tpc, která volá informační zprávu, zobrazí ji v konzoli prohlížeče a uloží do souboru log.txt. Ten jsme si předem vytvořili v kořenovém adresáři struktury. Ukázka záznamu z logu v prohlížeči je na obrázku 33 - Záznam Monologu do konzole prohlížeče.



Obrázek 33 - Záznam Monologu do konzole prohlížeče (Chrome)

```
1. <?php
2. // urceni jmenneho prostoru
3. namespace Controllers;
4. // import use
5. use MVC\Controller;
6. // rozsireni abstraktni tridy Controller
7. class ThirdPartiesController extends Controller
8. {
```

```
9.      // deklarace promenne
10.     private $logger;
11.     // konstruktor, ve kterem se vyzkousi komponenta logger
12.     public function __construct($logger)
13.     {
14.         $this->setView();
15.         $this->logger = $logger;
16.         $this->logger->debug("__construct " . date("h:i:s"));
17.     }
18.     // funkce, ve ktere se vyzkousi komponenta logger
19.     public function tpc()
20.     {
21.         $this->logger->notice("tpc " . date("h:i:s"));
22.         $this->view->render("ThirdPartiesComponents.html");
23.     }
24. }
```

4.4.9 Index.php

Jako poslední jsme si vytvořili soubor index.php, na který se nám převádí veškeré požadavky, které na server přijdou. Na začátek souboru jsme si přidali volání komponenty autoload. Tímto je zajištěno bezproblémové načítání tříd. Dále jsme si vytvořili novou instanci MVC Micro Frameworku a spustili na ní příkaz run.

```
1. <?php
2. require_once __dir__ . '/vendor/autoload.php';
3. $application = new App\App();
4. $application->run();
```

4.4.10 GitHub a Packagist

Poté, co jsme vytvořili všechny potřebné soubory, jsme všechno nahráli na GitHub a Packagist. Postupovali jsme stejně jako při nahrávání našeho mikro frameworku. Správné nahrání jsme si ověřili tím, že jsme si zkusili vytvořit vzorový projekt našeho skeletu. Pro instalaci jsme použili následující příkaz.

```
composer create-project filarichard/mmf_skeleton
```

Ten jsme spustili v příkazovém řádku PhpStorm a Composer nainstaloval všechny závislosti za nás. Nejprve nainstaloval mvc_micro_framework a poté komponenty třetích stran. Dále vytvořil všechny soubory a složky přesně podle očekávání.

4.5 Shrnutí praktické části

I když námi vytvořený framework nabízí spoustu možností, uživatele nijak nenutí je využívat. Je možné si náš framework stáhnout a využívat jeho komponenty samostatně. Nabízí například využití Dependency Injection kontejneru, uživatel se ale může rozhodnout pro jakýkoliv jiný způsob předání informací ovladačům. Dále nabízí poměrně velkou volnost při tvorbě cest.

Se stejnou politikou jsme postupovali i při vývoji skeletu aplikace. Uživateli je poskytnuto například logování do souboru. Program ale počítá i s tím, že soubor neexistuje a vytvoří ho sám. Případně si tuto možnost může uživatel sám vypnout před spuštěním. Dále je zde například možnost využít lehkého připojení do databáze a ORM. Nic se ale nestane, pokud uživatel nevytvoří správnou databázi a nenaplní ji daty před spuštěním aplikace.
Aplikace má pro správný běh následující požadavky:

- Webový server Apache
- PHP 7.1+
- MySQL
- Composer

Pro spuštění na serveru s jinou konfigurací je potřeba provést přídavné úpravy.

V příloze A, B a C jsou znázorněny UML diagramy tříd v tomto pořadí, modulu Event dispatcher, modulu MVC a skeletonu celé aplikace. Z důvodu zjednodušení vzhledu, zde nejsou obsaženy návaznosti na služby třetích stran.

ZÁVĚR

I v současné době se při vývoji aplikací obrací vývojář k programátorským praktikám, které se užívají již desítky let. Umění využívat návrhové vzory bývá často hlavním rozdílem mezi kvalitním programátorem a amatérem. Nicméně je nutné brát je s částečným nadhledem. Při jejich zkoumání se můžeme dozvědět, že spousta z těchto vzorů má mnoho možných interpretací. Často se také stává, že i u hojně opěvovaných vzorů se časem ukáže, že jiné řešení je mnohdy lepší a jsou posléze nahrazeny vzory novými. Stávají se z nich takzvané antivzory, jako například Service Locator. Rovněž je možné, že i tak základní a pro začínající programátory oblíbený návrhový vzor Jedináček se v určitých kruzích stává nepříjemností, bez které se dá obejít.

V první části této práce je stručné představení programovacího jazyka PHP. Je v ní ukázána jeho historie, současný stav a seznámili jsme se s velmi využívanými standardy PHP-FIG. Díky nim se stávají webové aplikace psané v PHP lépe čitelné a umožňují lepší spolupráci více autorů na jednom projektu.

V kapitole 2 jsou ukázány současné trendy v PHP programování, konkrétně využívání frameworků. Je v ní seznámení s nejdůležitějšími z velkých aplikačních frameworků, které vývojáři pomohou při sestavování rozsáhlých aplikací. Dále jsme se podívali na nejoblíbenější micro frameworky, které jsou vhodné pro malé aplikace či jako jádro, které se obalí dalšími balíčky. Ukázali jsme si jakou výhodu má používání jednotlivých komponent těchto frameworků a jaké návrhové vzory implementují.

Po ukázání nejdůležitějších návrhových vzorů, zejména vhodných pro tvorbu webových aplikací, se tato diplomová práce věnuje tvorbě vlastního frameworku a na něm postaveném skeletonu aplikace. Je zde ukázáno, jak se vytvořená aplikace publikuje a zakládá se okolo ní veřejná komunita.

Požadavky na praktickou část této diplomové práce byly splněny, je zde ovšem možnost rozšíření o další funkcionality. Díky využití balíčkovacího systému Composer je ovšem realizace takovéto rozšíření možná a velmi snadná.

POUŽITÁ LITERATURA

[1] *PHP* [online]. [cit. 2019-07-18]. Dostupné z: https://www.php.net/

[2] robsipek. Novinky ve verzi PHP 7. *Zooom.cz* [online]. 2018, 13 září 2018 [cit. 2019-07-25]. Dostupné z: https://zooom.cz/php-novinky-ve-verzi-php-7/

[3] TATROE, Kevin, Rasmus LERDORF a Peter MACINTYRE. *Programming PHP*. 3rd ed. Sebastopol, CA: O'Reilly Media, 2013. ISBN 978-144-9392-772.

[4] MACHAČ, Marek. PHP 7: deset věcí, které o něm potřebujete vědět. *Interval.cz* [online]. 2015, 26.10.2015 [cit. 2019-07-19]. Dostupné z: https://www.interval.cz/clanky/php-7-deset-veci-ktere-o-nem-potrebujete-vedet/

[5] KRČMÁŘ, Petr. PHP 5.x končí podpora ke konci roku, používá ho 62 % webů. *Root.cz* [online]. 2018, 15. 10. 2018 [cit. 2019-07-19]. Dostupné z: https://www.root.cz/zpravicky/php-5-x-konci-podpora-ke-konci-roku-pouziva-ho-62-webu/

[6] BODNÁR, Ján. Novinky jazyka PHP 7. *Root.cz* [online]. 2016, 21. 3. 2016 [cit. 2019-07-19]. Dostupné z: https://www.root.cz/clanky/novinky-jazyka-php-7/

[7] *PHP Standards Recommendations - PHP-FIG* [online]. [cit. 2019-07-18]. Dostupné z: https://www.php-fig.org/

[8] ČÁPKA, David. Standardy jazyka PHP - Úvod a PSR-1. *Itnetwork.cz* [online]. [cit. 2019-07-19]. Dostupné z: https://www.itnetwork.cz/php/psr/standardy-jazyka-php-uvod-a-psr-1

[9] GRUDL, David. Proč Nette nedodržuje standardy PHP-FIG / PSR?. p*hpFashion* [online]. 2014, 27.8. 2014 [cit. 2019-07-19]. Dostupné z: https://phpfashion.com/proc-nette-nedodrzuje-standardy-php-fig-psr

[10] ČÁPKA, David. Standardy jazyka PHP - PSR-2 část první. *Itnetwork.cz* [online]. [cit. 2019-07-19]. Dostupné z: https://www.itnetwork.cz/php/psr/standardy-jazyka-php-psr-2-cast-1

[11] ČÁPKA, David. Standardy jazyka PHP - PSR-2 část druhá. *Itnetwork.cz* [online]. [cit. 2019-07-19]. Dostupné z: https://www.itnetwork.cz/php/psr/standardy-jazyka-php-psr-2-cast-2

[12] KLÍMA, Tomáš. PSR-4. *Jak psát PHP* [online]. 2017, 10. 10. 2017 [cit. 2019-07-19]. Dostupné z: http://jakpsatphp.cz/PSR4/

[13] *Symfony* [online]. [cit. 2019-07-19]. Dostupné z: https://symfony.com/

[14] RIEHLE, Dirk. *Framework Design: A Role Modeling Approach*. Curych, Švýcarsko, 2000. Disertace. ETH Zürich.

[15] PEACOCK, Michael. *Programujeme vlastní e-shop v PHP 5*. Brno: Computer Press, 2011. ISBN 978-80-251-3181-7.

[16] MALÝ, Martin. Manifest miniaturního PHP. *Zdrojak.cz* [online]. 2012, 6.1.2012 [cit. 2019-07-19]. Dostupné z: https://www.zdrojak.cz/clanky/manifest-miniaturniho-php/

[17] GRUDL, David. Framework je přežitek. p*hpFashion* [online]. 2015, 8. 4. 2015 [cit. 2019-07-19]. Dostupné z: https://phpfashion.com/framework-je-prezitek

[18] *Composer* [online]. [cit. 2019-07-19]. Dostupné z: https://getcomposer.org

[19] *Nette - Comfortable and Safe Web Development in PHP* [online]. [cit. 2019-07-19]. Dostupné z: https://nette.org/en/

[20] DEEPAK, Kumar. Design patterns used in Zend framework (ZF2). *Deepak Kumar* [online]. 2014, 10. 10. 2014 [cit. 2019-07-19]. Dostupné z: http://deepakkumar.co.in/blog/technical/2014/10/10/design-patterns-used-in-zend-framework-zf2/

[21] *Zend Framework* [online]. [cit. 2019-07-19]. Dostupné z: https://framework.zend.com/

[22] SURGUY, Maksim. History of Laravel PHP framework, Eloquence emerging. *Maxoffsky* [online]. 2013, 27. 7. 2013 [cit. 2019-07-19]. Dostupné z: https://maxoffsky.com/code-blog/history-of-laravel-php-framework-eloquence-emerging/

[23] O'BRIEN, Jesse. A Brief History of Laravel. *Medium* [online]. 2016, 18. 7. 2016 [cit. 2019-07-19]. Dostupné z: https://medium.com/vehikl-news/a-brief-history-of-laravel-5d55970885bc

[24] BARNES, Eric L. Laravel Release Process. *Laravel News* [online]. 2019, 15. 4. 2019 [cit. 2019-07-19]. Dostupné z: https://laravel-news.com/laravel-release-process

[25] *Yii PHP Framework* [online]. [cit. 2019-07-19]. Dostupné z: https://www.yiiframework.com/

[26] MAKAROV, Alexander. The history of YII Framework. *Rmcreative* [online]. 2017, 4. 5. 2017 [cit. 2019-07-19]. Dostupné z: https://en.rmcreative.ru/blog/the-history-of-yii-framework/

[27] RAJPUT, Mehul. The Difference Between Laravel and Yii Framework – Identify The Best. *Mindinventory* [online]. 2017, 10. 4. 2017 [cit. 2019-07-19]. Dostupné z: https://www.mindinventory.com/blog/the-difference-between-laravel-and-yii-framework-identify-the-best/

[28] *PhalconPhp* [online]. [cit. 2019-07-19]. Dostupné z: https://phalconphp.com/cs/

[29] *HotFrameworks* [online]. [cit. 2019-07-19]. Dostupné z: https://hotframeworks.com/

[30] The Benchmarker. *The Benchmarker/web-frameworks* [online]. 2019 [cit. 2019-07-24]. Dostupné z: https://github.com/the-benchmarker

[31] HABIB, Omed. PHP Microframework vs. Full Stack Framework. *Appdynamics* [online]. 2015, 17. 11. 2015 [cit. 2019-07-19]. Dostupné z: https://www.appdynamics.com/blog/engineering/php-microframework-vs-full-stack-framework/

[32] *Slim Framework* [online]. [cit. 2019-07-19]. Dostupné z: http://www.slimframework.com/

[33] *Silex - The PHP micro-framework based on the Symfony Components* [online]. [cit. 2019-07-19]. Dostupné z: https://silex.symfony.com/

[34] *Lumen - PHP Micro Framework by Laravel* [online]. [cit. 2019-07-19]. Dostupné z: https://lumen.laravel.com/

[35] PATAKI, Daniel. Flight PHP – A Micro-Framework. *Daniel Pataki* [online]. 2016, 14. 6. 2016 [cit. 2019-07-19]. Dostupné z: https://danielpataki.com/flight-php/

[36] *Flight - An extensible micro-framework for PHP* [online]. [cit. 2019-07-19]. Dostupné z: http://flightphp.com/

[37] TARVAINEN, Jani. Symfony Benchmarks: Symfony Microkernel, Lumen, Silex, Slim... *Symfony Finland* [online]. 2016, 6. 1. 2016 [cit. 2019-07-19]. Dostupné z: https://symfony.fi/entry/symfony-benchmarks-microkernel-silex-lumen-and-slim

[38] GAMMA, Erich. *Návrh programů pomocí vzorů: stavební kameny objektově orientovaných programů*. Praha: Grada, 2003. Moderní programování. ISBN 80-247-0302-5.

[39] PECINOVSKÝ, Rudolf. *Návrhové vzory: [33 vzorových postupů pro objektové programování]*. Brno: Computer Press, 2007. ISBN 978-80-251-1582-4.

[40] FOWLER, Martin. *Patterns of enterprise application architecture*. Boston: Addison-Wesley, c2003. ISBN 03-211-2742-0.

[41] FREEMAN, Eric, Elisabeth ROBSON, Kathy SIERRA a Bert BATES. *Head First design patterns*. Sebastopol, CA: O'Reilly, c2004. ISBN 05-960-0712-4.

[42] *Source Making* [online]. [cit. 2019-07-24]. Dostupné z: https://sourcemaking.com/

[43] GRUDL, David. Co je Dependency Injection?. *PhpFashion* [online]. 2012, 12. 3. 2012 [cit. 2019-07-19]. Dostupné z: https://phpfashion.com/co-je-dependency-injection

[44] JANSSEN, Thorben. Design Patterns Explained – Dependency Injection with Code Examples. *Stackify* [online]. 2018, 19. 6. 2018 [cit. 2019-07-19]. Dostupné z: https://stackify.com/dependency-injection/

[45] GRUDL, David. DI versus Service Locator. *PhpFashion* [online]. 2012, 14. 3. 2012 [cit. 2019-07-19]. Dostupné z: https://phpfashion.com/di-versus-service-locator

[46] MACHAČ, Marek. Jak efektivně pracovat v PhpStorm?. *Interval.cz* [online]. 2014, 29.12.2014 [cit. 2019-07-19]. Dostupné z: https://www.interval.cz/clanky/jak-efektivne-pracovat-v-phpstorm/

[47] GANESAN, Prabhu. What is XAMPP? and How to Install XAMPP on Local Computer?. *WpBlogx* [online]. 2017, 16. 10. 2017 [cit. 2019-07-19]. Dostupné z: https://www.wpblogx.com/what-is-xampp/

[48] *Apache Friends* [online]. [cit. 2019-07-19]. Dostupné z: https://www.apachefriends.org/index.html

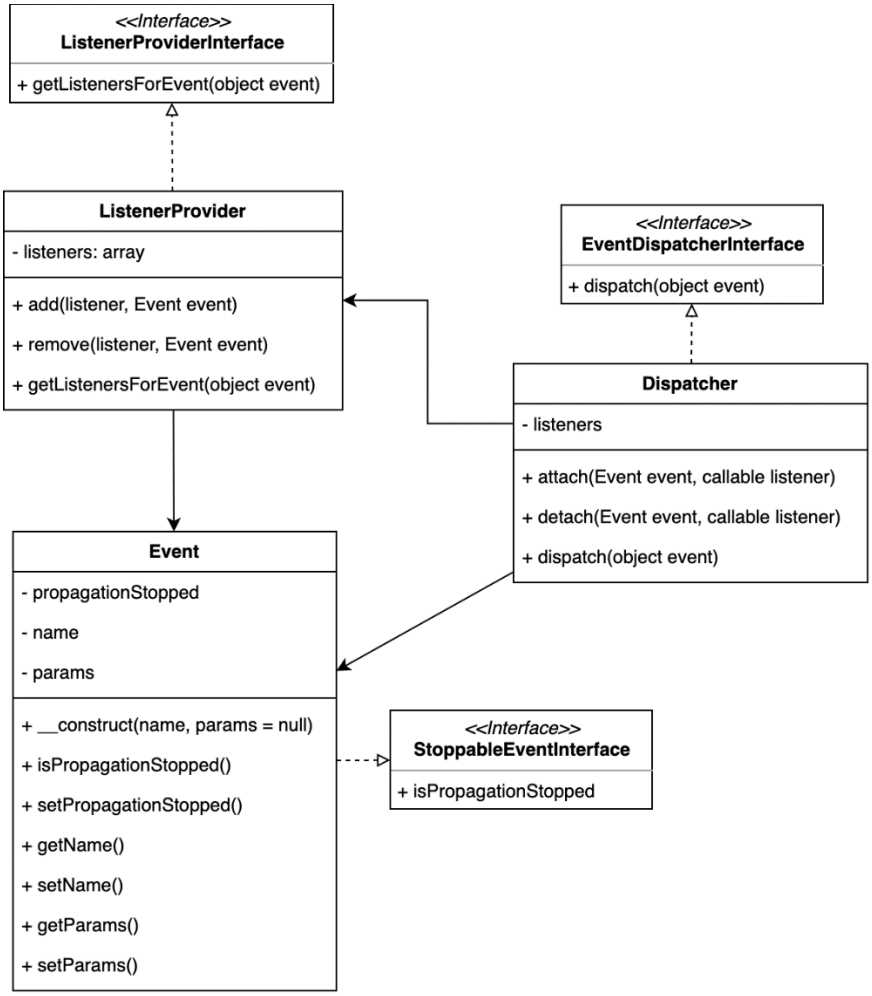
[49] *PIMPLE - A simple PHP Dependency Injection Container* [online]. [cit. 2019-07-24]. Dostupné z: https://pimple.symfony.com/

[50] MENDEZ, Juliet. PHP Monolog Tutorial: A Step by Step Guide. *Stackify* [online]. [cit. 2019-07-24]. Dostupné z: https://stackify.com

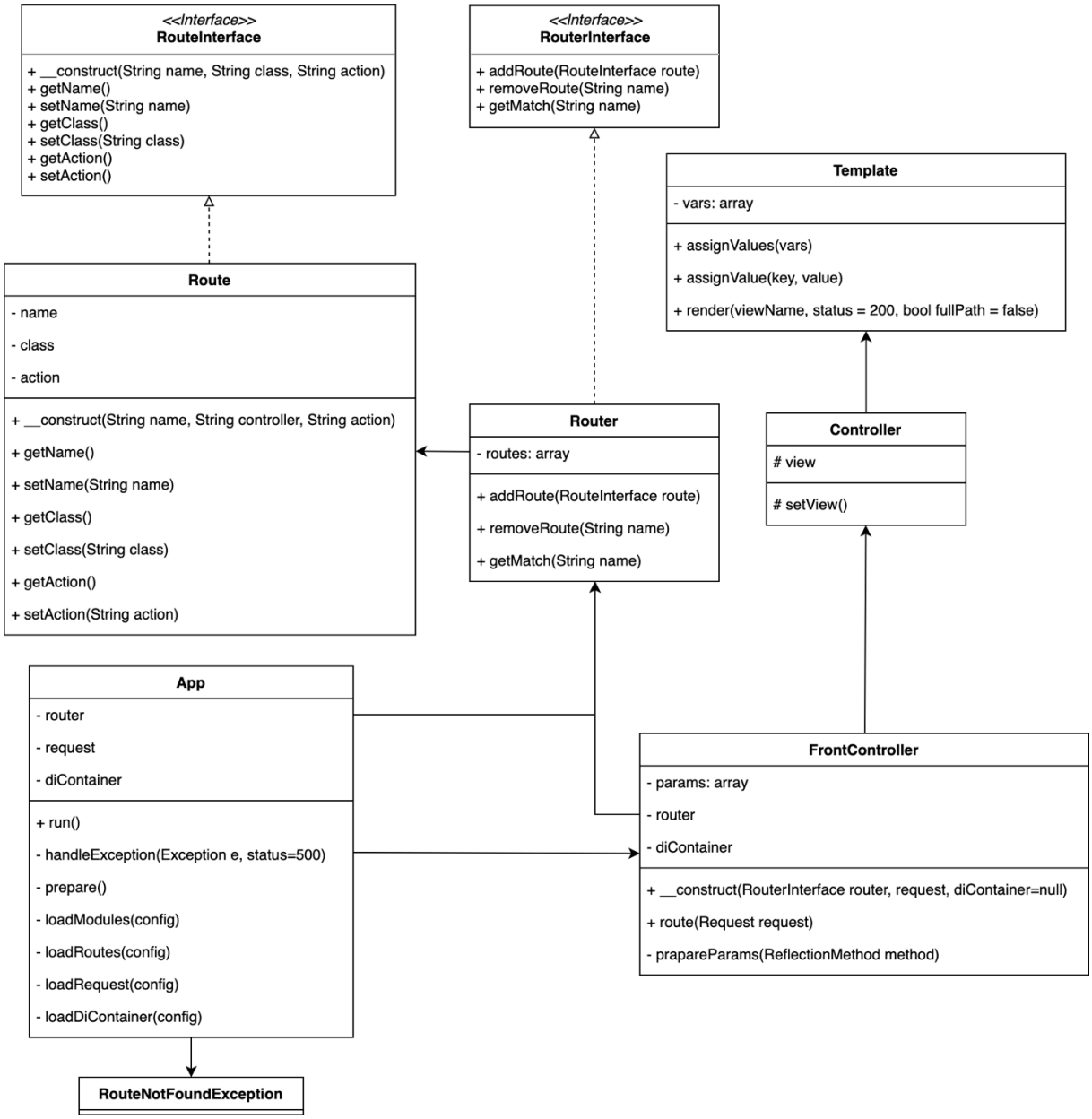
[51] *Laravel - The PHP Framework For Web Artisans* [online]. [cit. 2019-07-19]. Dostupné z: https://laravel.com/

PŘÍLOHY

PŘÍLOHA A – UML DIAGRAM TŘÍD MODULU EVENT DISPATCHER



PŘÍLOHA B – UML DIAGRAM TŘÍD MODULU MVC



PŘÍLOHA C – UML DIAGRAM TŘÍD SKELETU APLIKACE