Vysoká škola ekonomická v Praze

Fakulta informatiky a statistiky

Katedra informačních technologií

Študijný program: Aplikovaná informatika

Obor: Informační systémy a technologie

Porovnanie PHP frameworkov Symfony 2 a Zend Framework 2

DIPLOMOVÁ PRÁCA

Diplomant : Bc. Ján Koščo

Vedúci : Ing. Jarmila Pavlíčková

Oponent : Ing. Tomáš Fesič

2013

Prehlásenie

Prehlasujem, že som diplomovú prácu spracoval samostatne a že som uviedol všetky použité pramene a literatúru, z ktorej som čerpal.

V Prahe dňa 11. Decembra 2013 .....

Ján Koščo

Poďakovanie

Týmto by som sa chcel poďakovať vedúcej diplomovej práce Ing. Jarmile Pavlíčkovej za spoluprácu, cenné rady a za usmerňovanie mojej činnosti.

Abstrakt

Diplomová práca sa zaoberá porovnaním dvoch webových aplikačných frameworkov jazyka PHP. Cieľom práce je poskytnúť komplexný pohľad na možnosti, rozdiely a výhody zvolených frameworkov. Hlavným zdrojom pre porovnávanie sú ukážkové webové aplikácie.

Pre komplexné porovnávanie boli zvolené rozličné kritéria, ktoré analyzujú ich vlastnosti. Tieto kritéria sú rozdelené do piatich celkov. Podpora technológií skúma kompatibilitu frameworku pre rozličné verzie platformy. Najdôležitejšou a najrozsiahlejšou kategóriou je vývoj aplikácií. Táto kategória rozoberá najčastejšie úlohy pri tvorbe webových aplikácií a spôsoby ako k nim pristupujú jednotlivé frameworky. Ďalšími kategóriami sú návrh a architektúra frameworku, testovanie a všeobecná podpora pre vývojárov. Posledné zo spomínaných kritérií analyzuje online a offline zdroje informácií, ktoré slúžia na naučenie sa základom frameworku, ale taktiež pokročilým technikám jeho využitia.

Výsledky porovnania vyššie uvedených kritérií, by mali pomôcť vývojárom pri výbere moderného webového frameworku pre vývoj aplikácií na platforme PHP.

Kľúčové slová

Zend Framework, Symfony, PHP, framework, porovnanie, webové aplikácie

Abstrakt

Diplomová práce se zabývá srovnáním dvou webových aplikačních frameworků jazyka PHP. Cílem práce je poskytnout komplexní pohled na možnosti, rozdíly a výhody zvolených frameworků. Hlavním zdrojem pro srovnáni jsou ukázkové webové aplikace.

Pro komplexní srovnáni byly zvoleny různá kritéria, které analyzují jejích vlastnosti. Tato kritéria jsou rozděleny do pěti celků. Podpora technologií, zkoumá kompatibilitu frameworku pro různé verze platformy. Nejdůležitější a nejrozsáhlejší kategorií je vývoj aplikací. Táto kategorie rozebírá nejčastější úlohy při tvorbě webových aplikací a způsoby jak k nim přistupují jednotlivé frameworky. Dalšími kategoriemi jsou návrh a architektura frameworku, testování a obecná podpora pro vývojáře. Poslední ze zmíněných kritérií analyzuje online i offline zdroje informací, které slouží k naučení se základům frameworku, ale také pokročilým technikám jeho využití.

Výsledky srovnáni výše uvedených kritérií, by měly pomoci vývojářům při výběru moderního webového frameworku pro vývoj aplikací na platformě PHP.

Klíčová slova

Zend Framework, Symfony, PHP, framework, srovnání, webové aplikace

Abstract

Master thesis deals with a comparison of two web application frameworks in PHP. The aim of this work is to provide a comprehensive view of the possibilities, differences and advantages of the selected frameworks. The main source for comparison is sample web applications that demonstrate these possibilities.

For a comprehensive comparison, various criteria were chosen to analyze the properties of frameworks. These criteria are divided into five categories. Technology support examines the compatibility of framework for different versions of the platform. The most important and most widespread category is the analysis of options for creation of web applications, which discusses the most common tasks in developing applications and how frameworks approaches it. Other categories are design and architecture of framework, testing and general support for developers, which analyzes both online and offline sources of information used to learn and solve problems using the selected framework .

Comparison results based on above mentioned criteria can help developers when choosing a modern web development framework for PHP platform.

Keywords

Zend Framework, Symfony, PHP, framework, comparison, web applications

Obsah

1. Úvod 1

1.1. Cieľ práce 1

1.2. Prínosy práce 1

1.3. Predpoklady práce 2

1.4. Rešerš prác s podobnou témou 2

2. Vymedzenie základných pojmov 3

2.1. Programovací jazyk PHP 3

2.2. Webové aplikácie 3

2.3. Návrhové vzory 4

2.3.1. Návrhové vzory dátových zdrojov 4

2.3.2. Návrhové vzory webovej prezentácie 5

3. Definícia porovnávacích kritérií 7

3.1. Architektúra frameworku a zavedené konvencie 7

3.2. Podporované technológie 8

3.3. Vývoj aplikácií 9

3.3.1. Databáze 9

3.3.2. Bezpečnosť 10

3.3.3. Internacionalizácia a lokalizácia 10

3.3.4. Radiče 11

3.3.5. Formuláre 12

3.3.6. Komponentový návrh a znova použiteľnosť 13

3.3.7. Pohľady 13

3.3.8. Nástroje príkazového riadku 14

3.4. Testovanie 14

3.5. Podpora a iné 15

4. Výber frameworkov 17

4.1. Kritéria výberu 17
4.2. Vyhodnotenie výberu 20
5. Ukážková aplikácia 21
5.1. Popis aplikácie 21
5.2. Požadovaná funkcionalita 21
5.2.1. Špecifikácia prípadov použitia 21
5.3. Fyzický dátový model 23
5.4. Technické požiadavky 23
5.5. Návrh užívateľského rozhrania 24
6. Vývoj ukážkovej aplikácie 26
6.1. Inštalácia, konfigurácia a spustenie aplikácie 26
6.2. Komponenty 29
6.3. Databáze a dátové modely 31
6.4. Radiče 35
6.4.1. Smerovanie v aplikácií 36
6.4.2. Rozširujúce funkcie radičov 38
6.4.3. Bleskové správy 38
6.5. Formuláre 39
6.6. Pohľady 43
6.7. Bezpečnosť 45
6.7.1. Autentizácia a autorizácia 46
6.7.2. Ochrana proti najčastejším útokom 47
6.8. Internacionalizácia a lokalizácia 51
6.9. Nástroje príkazového riadku 52
6.10. Testovanie 54
6.11. Hodnotenie autora 55
7. Vyhodnotenie kritérií 56
7.1. Architektúra frameworku a zavedené konvencie 56

7.2. Podporované technológie 57
7.3. Databáze 57
7.4. Pohľady 59
7.5. Radiče 59
7.6. Komponentový návrh a znova použiteľnosť 59
7.7. Formuláre 60
7.8. Bezpečnosť 61
7.9. Internacionalizácia a lokalizácia 61
7.10. Nástroje príkazového riadku 62
7.11. Testovanie 62
7.12. Podpora a iné 62
7.13. Finálne hodnotenie kritérií 64
Terminologický slovník 68
Zdroje 69
Zoznam obrázkov 72
Zoznam tabuliek 72
Zoznam výpisov 73

1. Úvod

V čase Facebooku, Twitteru a iných veľkých webových aplikácií, ktoré denné navštívia desiatky miliónov užívateľov, sú na tieto aplikácie kladené vysoká nároky. A to hlavne nároky na bezpečnosť, rýchlosť, rozšíriteľnosť, škálovateľnosť a všeobecnú udržateľnosť aplikácie. Zaručiť všetky tieto vlastnosti je komplikované. Jedným zo spôsobov je dodržanie všeobecných, už vyskúšaných postupov, ktoré k nim vedú. Tieto *„best practices“* prinášajú do vývoja webové aplikačné frameworky.

Výber kvalitného frameworku je rozhodnutie, ktoré netreba podceňovať. Aplikácie vytvorené pomocou jedného framewokru, nie je jednoduché previesť na iný. Preto je potrebné pred začatím každého projektu analyzovať súčasne používaný rámec a overiť, či spĺňa všetky požiadavky vývojárov.

Existuje veľké množstvo framework pre rozličné platformy. Vždy by ale mali prinášať pridanú hodnotu vývojárom a pomáhať im k tomu, aby vytvárali kvalitné, otestované a udržateľné aplikácie. Táto práca sa venuje webovým aplikačným frameworkom na platforme PHP.

1.1. Cieľ práce

Hlavným cieľom tejto práce je stanovanie kritérií na všeobecné zhodnotenie možností webových aplikačných frameworkov. Zvolené kritéria analyzujú vlastnosti a taktiež podporované funkcie týchto rámcov. Druhým cieľom je zhodnotenie dvoch rámcov pomocou týchto kritérií a určenie vhodnejšieho kandidáta na tvorbu webových aplikácií.

Vedľajším cieľom práce je ukázať čitateľovi postupy tvorby aplikácií pri použití webových aplikačných frameworkov. Jednotlivé postupy a kroky vývoja sú opísané v kapitole venujúcej sa implementácií ukážkovej aplikácie pomocou oboch rámcov.

1.2. Prínosy práce

Kritéria stanovené v tejto práci slúžia na porovnanie možností frameworkov. Vývojári ich teda môžu využiť pri vlastnej analýze iných open-source riešení na rozličných platformách.

Okrem iného by táto práca taktiež mala ponúknuť čitateľovi všeobecný pohľad na tvorbu webových aplikácií pomocou webových aplikačných frameworkov. Zameriava sa na možnosti, ktoré tieto rámce ponúkajú.

Osobným prínosom práce pre autora je prehĺbenie znalostí s dvoma analyzovanými frameworkmi, ktoré patria k najpoužívanejším vo svete.

1.3. Predpoklady práce

Pre plné pochopenie práce sa predpokladá, že čitatelia sú oboznámení minimálne so základnými konceptmi objektového programovania v jazyku PHP a problematikou webových aplikácií.

1.4. Rešerš prác s podobnou témou

Téma webových aplikačných frameworkov je vysoko diskutovaná. Bolo k nej napísaných viacero prác, ktoré skúmajú a porovnávajú frameworky rozličných veľkostí a na rozličných platformách. Webové aplikačné frameworky sa vyvíjajú veľmi rýchlo a za jeden rok môže byť vydaných aj niekoľko minoritných verzií. Tie neprinášajú len opravy, ale väčšinou aj novú funkcionalitu a možnosti. Knižné podklady aj akademické práce sa tak stávajú neaktuálnymi veľmi rýchlo.

Hoci bol Symfony 2 vydaný už pred viac ako 2 rokmi a patrí k najznámejším frameworkom, prác, ktoré by sa venovali tvorbe aplikácií pomocou tohto frameworku alebo ho porovnávali neexistuje mnoho. Jedná sa o bakalárske práce *Realizace internetového projektu s využitím PHP frameworků* (KORECKÝ, 2013) a *Webový informační systém pro správu přihlášek na kurzy* (PRCHAL, 2013). Obe rozoberajú základné koncepty tvorby aplikácií pomocou frameworkov. Taktiež v týchto prácach vytvárajú menšie webové aplikácie. Okrem týchto prác boli všetky informácie získavané z oficiálnej dokumentácie, referenčnej príručky a niekoľkých blogov základných vývojárov frameworku.

Zend framework taktiež patrí medzi známe frameworky vo svete a často práve preto býva zástupcom webových aplikačných frameworkov na platforme PHP v akademických prácach. Väčšina nájdených prác ale používa staršiu verziu, a to Zend Frameworku 1. Tento framework sa podstatne líši od skúmanej druhej verzie. Najrozsiahlejšou analýzou je diplomová práca Magdy Kutišovej (2013). Tá analyzuje rozdiely Zend framework medzi verziami 1 a 2. Táto práca porovnávala verziu 2.1.5, ktorá už je v súčasnosti neaktuálna. Druhou nájdenou prácou je porovnanie Zend Frameworku a Nette Frameworku od Petra Olišara (2013). Všetky ďalšie informácie boli získavané z oficiálnej dokumentácie a blogu.

Oba frameworky boli vydané už dávnejšie, no zatiaľ neexistuje akademická práca, ktorá by sa venovala ich podrobenému porovnaniu.

2. Vymedzenie základných pojmov

Pre plné porozumenie tejto práce je potrebné nadefinovať a vysvetliť určite pojmy, na ktoré nasledujúce kapitoly často odkazujú. V úvode bude v skratke predstavený jazyk PHP, následne budú objasnené pojmy *webové aplikácie* a *webový aplikačný framework*. Kapitola bude zakončená krátkym úvodom do návrhových vzorov.

2.1. Programovací jazyk PHP

PHP: Hypertext Preprocessor je open source multiplatformový skriptovací programovací jazyk. V súčasnosti je najpoužívanejším (Q-SUCCESS, 2013) programovacím jazykom na strane servera a najpopulárnejší (TIOBE Software, 2013) programovací jazyk vôbec. Prevažne sa využíva na tvorbu dynamických webových stránok.

Vznikol v roku 1995 ako súhrn niekoľkých skriptov jazyka Perl, ktoré umožňovali autorovi Rasmusovi Lerdorfovi zobrazovať počet návštev na jeho webovej stránke. O dané skripty sa začalo zaujímať veľké množstvo vývojárov. V roku 1996 ich prepísal do jazyka C a zverejnil pod názvom *Personal Home Page*. Neskôr bol tento projekt premenovaný a začal používať súčasný názov. Vďaka snahe vývojárov Zeeva Suraskiho a Andiho Gutmansa, ktorí prepísali syntaktický analyzátor, bolo možné vydať v roku 1998 tretiu verziu PHP. Títo vývojári taktiež založili spoločnosť Zend Technologies a začali pracovať na novom jadre Zend Engine. Až vydanie PHP 5 prinieslo plnú podporu objektové programovania.

Vyššie spomínaná spoločnosť ma na starosť vývoj PHP dodnes. Vývojový tím PHP aktuálne spravuje a podporuje tri posledné minoritné verzie jazyka, a to 5.3, 5.4 a 5.5. U verzie 5.3 už bol ohlásený koniec životného cyklu vývoja a pre danú verziu sú vydávané len bezpečnostné záplaty. Aktuálnou verziu je 5.5.6, ktorá vyšla 14. novembra 2013.

PHP sa stal rýchlo obľúbeným, pretože umožňuje bez nutnosti kompilácie tvoriť webové stránky, ponúka širokú škálu funkcií, podporuje práce s relačnými databázami a je ľahko rozšírený o nové funkcie.

2.2. Webové aplikácie

Jedná sa o software poskytovaný pomocou webového serveru cez verejnú počítačovú sieť *Internet* alebo v rámci podniku pomocou *Intranetu*. Ide o aplikáciu typu *klient-server*. Užívateľ pomocou webového prehliadača posiela požiadavky na webový server, ktorý mu v odpovedi posiela vygenerované webové stránky.

Webové aplikácie väčšinou využívajú trojvrstvovú architektúru (KŘIŽAN, 2010):

- *Prezentačná vrstva* má na starosť vykreslenie stránok vo webovom prehliadači.
- *Dátová vrstva* komunikuje a využíva databázové server na ukladanie dát.
- *Aplikačná vrstva,* ktorá ma na starosť spracovanie požiadavky a vygenerovanie stránky. *A*plikačná logika na strane servera komunikuje s dátovou vrstvou a posiela výstup na prezentačnú vrstvu.

Framework

Hoci je pojem framework často používaný, nemá presnú definíciu. Preto sú ďalej uvedené dve definície dostupné z literatúry. Válek (2008) definuje framework ako „*Sadu tried a rozhraní, ktorých kombináciou je možné riešiť špecifické problémy vývoju aplikácií*".

Podľa definície Hrubého (2010) je framework „*softwarová štruktúra, ktorá slúži ako podpora pri programovaní a vývoji a organizácií iných softwarových projektov. Môže obsahovať podporné programy, knižnicu API, návrhové vzory alebo doporučené postupy pri vývoji.*"

V tejto práci je pod pojmom framework myslený webový aplikačný framework. Ide o úzko zameraný typ frameworku, ktorý sa orientuje na vývoj webových aplikácií.

2.3. Návrhové vzory

Podľa Dvořáka (2003) predstavujú návrhové vzory všeobecný popis riešenia problému. Ich vznik a rozvoj je spojený s potrebou zachytiť možnosti riešení netriviálnych problémov, ktoré sa opakovane objavujú pri návrhu informačných systémov, predovšetkým v oblasti designu. Rozšírený pohľad na to prináša Martin Fowler (2002) vo svojej knihe *Patterns of Enterprise Application Architecture*, kde rozoberá architektonické vzory. Teda vzory, ktoré sa využívajú na návrh veľkých podnikových aplikácií.

Fowler (2002) delí architektonické návrhové vzory do desiatich kategórií. V nasledujúcej kapitole je objasnené využitie a účel návrhových vzorov, ktoré sú využívané u webových aplikačných frameworkov.

2.3.1. Návrhové vzory dátových zdrojov

V tejto kategórii definuje Fowler (2002) štyri návrhové vzory na prácu s dátovými zdrojmi:

- Table Data Gateway,
- Row Data Gateway,
- Active Record,
- Data Mapper.

Table Data Gateway

Implementácia návrhového vzoru *Table Data Gateway* vytvára a definuje objekt, ktorý reprezentuje celú tabuľku v databáze. Slúži ako brána na prácu s ňou a to tým, že zapuzdruje všetku interakciu z aplikácie. Pozostáva z niekoľkých metód, ktoré slúžia na získavanie, aktualizáciu, vytváranie a mazanie dát.

Row Data Gateway

Implementáciou tohto návrhového vzoru vznikajú objekty, ktoré reprezentujú jeden záznam z tabuľky. Jednotlivé stĺpce tabuľky sú reprezentované ako vlastnosti týchto objektov. Účel týchto objektov je držať informácie o zázname a umožniť k nim prístup. Objekty neobsahujú žiadnu doménovú logiku a preto musia existovať v aplikáciách iné objekty, ktoré budú túto funkcionalitu poskytovať.

Active Record

Tak ako návrhový vzor *Row Data Gateway*, aj *Active Record* vytvára objekty pre jednotlivé záznamy tabuľky. Rozdielom je, že tieto objekty taktiež obsahujú aj doménovú logiku. Poskytujú teda všeobecné funkcie slúžiace na prácu s celou tabuľkou, nielen s jedným záznamom.

Data Mapper

Jedná sa o mapovací objekt, ktorý vytvára vrstvu medzi doménovými objektmi a databázou. Slúži na komunikáciu a prenášanie dát medzi týmito dvoma časťami aplikácie. Týmto spôsobom je vývojár odtienený od databázovej schémy a samotnej práce databázového servera.

2.3.2. Návrhové vzory webovej prezentácie

Táto kapitola rozoberá návrhové vzory, ktoré definujú, akým spôsobom sú spracované požiadavky užívateľov na webové aplikácie a aké sú možnosti vykreslenia výsledných výstupov do prehliadača (FOWLER, 2002, s. 70).

Model View Controller

Jedná sa o veľmi často používaný návrhový vzor. Separuje dáta od spôsobu ich prezentácie a taktiež od spôsobu správania aplikácie. Skladá sa z troch častí: modelov, pohľadov a radičov. Každá časť má presne definovanú funkciu. Modely definujú doménové objekty a slúžia na komunikáciu s databázou. Pohľady vytvárajú užívateľské rozhranie na základe získaných dát z modelov. Radiče sú medzi vrstvou, medzi modelmi a pohľadmi. Prijímajú požiadavky užívateľov, získavajú požadované dáta z modelov a prenášajú ich do pohľadov.

Implementácia tohto návrhového vzoru prináša kompletnú separáciu jednotlivých častí od seba. Umožňuje to lepšiu testovateľnosť, nahraditeľnosť jednotlivých častí v aplikácii.

Front Controller

Front controller je typ radiča, ktorý spracúva všetky požiadavky na aplikáciu a distribuuje ich k príslušným príkazom, ktoré sa majú vykonať. Tento typ radiča sa využíva pri väčších aplikácií, kedy pre všetky vyžadované stránky a akcie je potrebné vykonať opakujúce sa funkcie, napr. overenie identity užívateľa v rámci bezpečnosti alebo nastavenia správnej jazykovej mutácie aplikácie.

Template View

Tento návrhový vzor sa využíva na vykresľovanie dynamických HTML stránok. Do staticky definovaných šablón sú pridané pred pripravené zástupné symboly, ktoré sú nahradzované dátami z modelov.

Two Step View

Jedná sa o dvojkrokové vytváranie HTML výstupov. Prvým krokom je vytvorenie logickej štruktúry jednotlivých častí stránky a druhým je samotné vykreslenie. Pri webových aplikácií je tento spôsob využívaný pri definovaní layoutu, teda opakujúcich sa časti stránok, ktoré sú naprieč aplikáciou rovnaké.

3. Definícia porovnávacích kritérií

V nasledujúcej kapitole budú stanovené kritéria, ktorými budú porovnávané dva zvolené PHP frameworky. Väčšina kritérií analyzuje, do akej miery daný framework podporuje určitú vlastnosť, napr. *lokalizáciu textov*. Všetky kritéria sú ohodnocované číselne v intervale $\langle 0,1 \rangle$, kde:

- Hodnota 0 znamená, že vlastnosť je nepodporovaná.
- Percentuálne hodnoty z intervalu $(0,1)$ indikujú čiastočnú podporu.
- Plná podpora je ohodnotená 1.

Pri kritériách, ktoré analyzujú časovú alebo konfiguračnú náročnosť, napr. *konfigurácia testovania,* evidujeme možnosti vysoká (hodnota 0) a nízka náročnosť (hodnota 1).

Všetky kritéria sú radené do kategórií, ktoré ich členia do väčších celkov podľa analyzovaných vlastností. Súčet váh jednotlivých kritérií v rámci kategórie je rovný 1. V práci som nadefinoval nasledujúcich päť kategórií:

- Architektúra frameworku a zavedené konvencie,
- podporované technológie,
- vývoj aplikácie,
- testovanie,
- podpora a iné.

Kategóriám je pridelená váha pomocou metódy párového porovnávania, tzv. Fullerovej metódy (FIALA, 1997). Výsledné porovnanie je teda úlohou viackriteriálneho hodnotenia variant.

3.1. Architektúra frameworku a zavedené konvencie

Návrhové vzory

V kapitole 2.3 bolo vysvetlené, čo sú to návrhové vzory a prečo sa pri tvorbe software používajú. Táto kapitola taktiež obsahovala zoznam a definíciu niekoľkých z nich. Kritérium hodnotí, či sú návrhové vzory používané a implementované u jednotlivých časti frameworkov.

Pravidlá pre písané kódu

Pri vývoji softwaru je dôležité dbať na to, aby bol kód dlhodobo udržiavateľný. Hoci väčšina programovacích jazykov (WIKIPEDIA, 2013) má definované aspoň určité doporučené konvencie písania kódu, nie všetci vývojári tieto pravidlá musia dodržiavať. Zavedením štandardov do projektov sa zvyšuje prehľadnosť zdrojového kódu a jeho udržateľnosť.

Kritérium hodnotí, či jednotlivé projekty majú definované vlastné konvencie písania kódu alebo dodržiavajú aspoň všeobecné konvencie. V súčasnosti medzi najrozšírenejšie konvencie v jazyku PHP patria definované štandardy skupiny *PHP Framework Interop Group*[1].

3.2. Podporované technológie

Nové verzie jazyka PHP prinášajú vždy novú funkcionalitu, ktorá nie je vždy spätne kompatibilná so staršími verziami. Frameworky musia byť ale vyvíjané tak, aby ich bolo možné používať na čo najväčšom počte webových serverov. Nasledujúce kritéria hodnotia minimálne požiadavky na verziu jazyka, podporu virtuálneho stroja a možnosti využitia nástroja na správu závislosti pri vývoji a nasadzovaní aplikácií.

Podporovaná verzia PHP

Tabuľka 1 obsahuje zoznam podporovaných verzií PHP u niekoľkých českých hostingových spoločností. Najpodporovanejšou verziou je v súčasnosti stále verzia PHP 5.3. Nasledujúce kritérium analyzuje minimálne požiadavky na podporovanú verziu PHP a teda, či je možné prevádzkovať aplikácie na českých hostingoch.

Tabuľka 1 - Podporované verzie PHP u českých hostigových spoločností

Hostingová spoločnosť	Podporované verzie PHP
Active 24	5.3
Banan	5.2 a 5.3
Czechia	5.4
Forpsi	5.2 a 5.3
Gigaserver	5.3
Hosting90	5.3
Ignum	5.3
ONEbit	5.3 a 5.4
Savana	5.4
Wedos	5.3 a 5.4

Podpora HHVM

PHP je skriptovací jazyk, čo znamená, že pri každom spustení skriptu sa musí celý zdrojový kód interpretovať. To so sebou nesie spotrebu ďalšieho strojového času, ktorý pri skompilovanom kódu už nie je potrebný. Najväčší portál na svete Facebook (PROTALINSKI, 2013) je taktiež naprogramovaný v jazyku PHP a preto sú na neho kladené vysoké požiadavky, čo sa týka výkonu. Preto vývojári z Facebooku navrhli a implementovali virtuálny stroj, ktorý prevádzal zdrojový kód PHP do jazyka C++. Ten už bolo možné skompilovať, čím navýšili požadovaný výkon.

[1] <http://www.php-fig.org/>

V súčasnej verzii tohto projektu funguje virtuálny stroj podobne ako na platforme Java, kedy kompilátor funguje na princípe *Just in time*. Vytvára teda spustiteľný kód až v okamihu potreby. Zdrojový kód virtuálneho stroja je dostupnýako open-source na serveri Github[2].

Výsledky z testovania Christiana Stockera (2013) poukazujú na to, že výkon aplikácií pri využití virtuálneho stroja je možné navýšiť aj o viac ako 300%, v porovnaní s PHP verzie 5.3. Výkon aplikácie je rýchlejší aj oproti optimalizovanej verzii 5.5 a to o viac ako 200%.

Kritérium hodnotí výsledky jednotkových testov pri použití HipHop virtuálneho stroja, ktoré indikujú celkovú podporu tohto frameworku.

Správa závislosti

V minulosti boli pri PHP projektoch inštalované knižnice tretích strán pomocou repozitára PEAR[3] (*PHP Extension and Application Repository*). Ten mal okrem iného na starosť aj správu ich závislosti. Jednotlivé balíčky boli inštalované globálne v systéme, čo zapríčiňovalo problémy. Vznikali hlavne v situáciách, ak rôzne projekty vyžadovali rozličné verzie jednej knižnice, napr. knižnice na testovanie *PHPUnit*.

V roku 2011 vznikol projekt Composer[4], ktorý bol inšpirovaný nástrojmi na správu závislostí u iných programovacích jazykov napr. *Gems* u jazyka Ruby. Tento nástroj uľahčuje inštaláciu a aktualizáciu balíčkov a taktiež má na starosť riešenie závislosti medzi samotnými baličkami. Nasledujúce kritérium hodnotí, či projekty využívajú na správu závislostí tento nástroj.

3.3. Vývoj aplikácií

Nasledujúca kategória analyzuje najčastejšie aspekty vývoja webových aplikácií. Keďže táto kategória obsahuje najväčší počet porovnávacích kritérií, budú jednotlivé kritéria rozdelené do pod kategórií.

3.3.1. Databázy

Podpora relačných databázových serverov

Štandardom pri tvorbe webových aplikácií je využitie databázy na ukladanie dát. Kritérium analyzuje podporu najznámejších databázových systémov.

[2] <https://github.com/facebook/hhvm>
[3] <http://pear.php.net/>
[4] <http://getcomposer.org/>

Podpora NoSQL databázových serverov

Veľkým súčasným trendom sú NoSQL databázy, ktoré umožňujú vysokú škálovateľnosť, jednoduchšiu distribuovateľnosť a vyšší výkon. Všetky tieto vlastnosti sú získané na úkor toho, že relačné databázy majú presne stanovenú schému aj s dátovými obmedzeniami a referenčnou integritou. Tieto databázy sa hodia na ukladanie neštruktúrovaných dát, napr. logov, či rozličných dokumentov. Kritérium hodnotí, či je možné využívať niektorú zo súčasných NoSQL databáz.

Spojenie s viacerými databázami

U väčších aplikácií môže nastať situácia, kedy je potrebné využívať viacero databázových systémov pre prácu s dátami. Jednoduchým príkladom môže byť využitie vyššie spomínaných NoSQL databáz, keďže umožňujú rýchly zápis aj čítanie dát. Kritérium hodnotí, či je možné sa v rámci aplikácií pripájať k viacerým databázovým serverom.

Migrácia databázových schém

Vývoj aplikácií je kontinuálny proces, ktorý má pravidelne dopad na databázovú schému aplikácie. Je pomerne riskantné spoliehať sa na to, že tieto zmeny budú aplikované na produkčnú databázu v správnom čase alebo poradí pri nasadzovaní novej verzie aplikácie. Taktiež do tohto zapadá aj ľudský faktor, kedy vývojár alebo administrátor zabudne aplikovať určité zmeny. Je dôležité preto tieto zmeny automatizovať. Kritérium skúma, či frameworky poskytujú nástroj na migráciu databázových schém.

3.3.2. Bezpečnosť

Hlavným cieľom bezpečnosti u webových aplikácií je neumožniť užívateľom prístup k zdrojom, na ktoré nemajú právo. V dnešnej dobe webové aplikácie ukladajú množstvá dát o užívateľoch. Od základných informácií, údajoch o platobných kartách, či lokalizačných údajoch. Pri prelomení aplikácie môže získať útočník veľkú časť užívateľovej identity. Preto je dôležité vytvárať bezpečné aplikácie odolné voči útokom.

Kategória hodnotí 3 kritéria a to podpora pre autentizáciu a autorizáciu užívateľov a dostupnosť metód na ochranu proti základným útokom na webové aplikácie.

3.3.3. Internacionalizácia a lokalizácia

Kvalitné webové aplikácie komunikujú s užívateľmi v jazyku, ktorému rozumejú. Rozhranie aplikácie je prispôsobené zvyklostiam ich národnosti, napr. zobrazenie formátu času. Každá krajina a jazyk má definované svoje označenie, ktoré pod sebou zhromažďuje tieto definované zvyklosti.

Framework umožňuje internacionalizáciu, ak poskytuje možnosti, ako tieto zvyklosti webovej aplikácie nastaviť. Lokalizácia je samotné nastavenie týchto zvyklostí do aplikácie pomocou príslušného označenia krajiny a jazyka. Jedná sa o preklad textov, nastavenie formátu jazyka, či meny.

V tejto kategórií sú hodnotené nasledujúce kritéria:

- Podpora internacionalizácie,
- lokalizácia textov,
- lokalizácia užívateľských textov,
- lokalizácia meny,
- lokalizácie dátumov a časov.

Kritériom lokalizácia užívateľských textov je myslená podpora lokalizácie pre dáta, ktoré sú uložené v databáze. Jedná sa o preklady užívateľských vstupov. Tie sú zobrazované podľa toho, aká je na stránke nastavená jazyková lokalizácia.

3.3.4. Radiče

Pomocné funkcie

Všeobecným pravidlom pri písaní kódu je „Neopakujú sa“ (Don't repeat yourself). Funkcie alebo časti kódu, ktoré sú potrebné na viacerých miestach aplikácie, majú byť napísané len raz. Kritérium hodnotí, či je možné vytvárať znova použiteľné funkcie, ktoré sú používané v radičoch.

Podpora bleskových správ

Jedná sa o krátke správy uložené v užívateľových sessions. Využívajú sa na jednorazové zobrazenie dodatočných informácií o užívateľovi. Napríklad po presmerovaní je užívateľovi zobrazená konfirmačná správa, že jeho objednávka bola úspešne odoslaná. V angličtine je tento typ správ nazývaný *Flash message*.

Mapovanie URL adries

V angličtine je táto funkcionalita nazývaná *Routing*, teda smerovanie požiadavku do príslušnej akcie v správnom radiči. Kritérium ohodnocuje podporu nasledujúcich typov mapovania URL adries:

- Statické smerovanie – pevne definovaná URL adresa, napr. */dashboard*.
- Dynamické smerovanie – parametrizácia príslušných akcií, napr. */employee/ <parameter>*.
- Smerovanie s využitím regulárneho výrazu – využívané napríklad pri multi-jazyčných aplikáciách.

3.3.5. Formuláre

Dôležitou súčasťou webových stránok a aplikácií sú formuláre, ktoré slúžia na odosielanie dát od užívateľov. Kritéria v nasledujúcej kapitole analyzujú, ako frameworky uľahčujú prácu vývojárom pri definovaní formulárov, ich kastomizácií, možnostiam kontroly vstupných dát od užívateľov a v neposlednej rade ako zvyšujú alebo znižujú zabezpečenie aplikácií.

Spôsob definície formulárov

Keďže sú formuláre tak častým prvkom webových aplikácií je dôležité, aby bola ich definícia čo najjednoduchšia. To ale nesmie znemožniť rozšíriteľnosť, ak je to potrebné. Kritérium rozoberá *komplexitu definície formulárov* a možnosti kastomizácie.

Preddefinované typy elementov

Kritérium hodnotí, či sú v rámci formulárovej komponenty implementované elementy, ktoré reprezentujú základné formulárové polia. Keďže v špecifikácií HTML 5[5] boli základné formulárové polia rozšírenie o nové typy polí, napr. *email*, overuje sa taktiež aj podpora týchto polí.

Možnosť vytvárať vlastne typy formulárových elementov

Keďže formulárové polia sú znova použiteľné, určite by mal pre nich existovať spôsob ako ich rozšíriť o určitú vlastnosť. Kritérium hodnotí podporu rozšíriteľnosti už existujúcich formulárových polí. Okrem toho je taktiež overená podpora definície vlastných formulárových typov, napr. pohlavia užívateľov.

Validácie

Formuláre po odoslaní posielajú užívateľský vstup do aplikácie. Predtým, než sú tieto dáta uložené do databázy je potrebné overiť, či sú platné. Okrem kontroly kompatibility dátových typov existuje veľké množstvo typov kontroly, napríklad to, že údaj je povinný alebo musí byť definovaný v stanovenom tvare a podobne. Kritérium overuje, či existuje vo frameworku podpora pre *validácie formulárových prvkov* a *možnosť vytvárať vlastné validácie*.

Ochrana proti Cross-Site Request Forgery

Kritérium hodnotí podporu ochrany proti útoku Cross-Site Request Forgery (v skratke predstavený v kapitole 6.7.2). Analyzuje, či formulárové komponenty majú implementované riešenie generovania autorizačného tokenu a jeho následnej kontroly pri odoslaní formuláru.

[5] <http://www.w3.org/html/wg/drafts/html/CR/forms.html#forms>

3.3.6. Komponentový návrh a znova použiteľnosť

Frameworky sú ukážkou komponentovej architektúry. Pomocou oddelených a samostatných komponent sú vytvorené komplexné riešenia, u ktorých existuje možnosť nahradiť určitú časť za inú. Desmonda D'souza (1999, s. 386) definuje komponentu ako: *„Ucelený softwarový balíček, ktorý môže byť nezávisle vyvinutý a poskytovaný a taktiež môže byť kombinovaný s ďalšími komponentmi bez nutnosti zásahu do samotných komponent.“.*

Väčšina súčasných frameworkov umožňuje vývojárom pristupovať k vývoju taktiež komponentovo. To z dôvodu, aby zdrojový kód mohol byť znova použiteľný. Kritérium overí, akým spôsobom je možné vytvárať vlastné komponenty, možnosti integrácie *komponent tretích strán* a počet dostupných komponent.

3.3.7. Pohľady

Poslednou, ešte nespomenutou časťou MVC architektúry, sú pohľady. Ako bolo vysvetlené v kapitole 2.3, pohľady majú na starosti vykreslenia finálnej stránky pre užívateľov. Kritéria v tejto kategórií hodnotia, akým spôsobom sú pohľady tvorené a či existujú pomocné nástroje a funkcie, ktoré to uľahčujú.

Vykresľovanie pohľadov

Niektoré časti stránky sú identické naprieč celou aplikáciou, napr. hlavička a pätička. Kritérium hodnotí podporu dedičnosti šablón.

V rámci kategórií o bezpečnosti bolo zámerne vynechané kritérium hodnotiace ochranu proti Cross Site Scriptingu, pretože je hodnotené v rámci tejto kategórie. Jedinou ochranou je escepovanie užívateľských vstupov pri výpise do stránky. Grudl (2009), definuje escpovanie ako „Prevod znakov majúcich v danom kontexte špeciálny význam na iné odpovedajúce sekvencie.“ Dané kritérium analyzuje podpora pre manuálne a automatické escapeovanie užívateľských vstupov v pohľadoch.

Správa zdrojov

Webové aplikácie pozostávajú z niekoľkých druhov zdrojov. Ide o súbory kaskádových štýlov, JavaScriptu a obrázkov. Všetky tieto súbory musia byť v koreňovom adresári webového serveru, aby boli prístupné verejnosti.

Ako bolo v predchádzajúcej časti vysvetlené, komponentový prístup vytvára samostatné celky, ktoré majú na starosť určitú funkcionalitu. V rámci týchto komponent musia byť zakomponované

všetky potrebné súbory, teda aj kaskádové štýly a JavaScriptové súbory. Ďalším kritériom je teda overenie, či existuje nástroj na m*anažment súborov* umiestnených mimo koreňový adresár.

Pomocné funkcie v pohľadoch

U väčších aplikácií sa stáva, že určitý prvok užívateľského rozhrania sa musí konzistentne vykresľovať a správať naprieč celou aplikáciou. Vo chvíli keď sa špecifikácia prvku pozmení, je potrebné ho aktualizovať naprieč celou aplikáciou. Jedným z riešenie tohto problému je vytvorenie pomocnej funkciu pre pohľady, ktorá sa o vykreslenie postará a aktualizácia prvku prebieha len na jednom mieste aplikácie. Toto kritérium skúma a hodnotí, či existuje spôsob ako rozšíriť pohľad o novú funkcionalitu.

3.3.8. Nástroje príkazového riadku

Niektoré súčasné frameworky umožňujú ovládať časť aplikácie pomocou príkazového riadku. Môže sa jednať o jednoduché príkazy typu vyčistenia cache, ale aj o komplexnejšie viac krokové príkazy, ktorých výsledkom je napr. vytvorenie užívateľa v aplikácií. Frameworky budú hodnotené v nasledujúcich kritériách:

- Dostupnosť vyššie spomínané nástroja,
- príkazy na prácu s databázou,
- generovanie časti aplikácií,
- vytváranie vlastných príkazov.

3.4. Testovanie

Kritéria sa zaoberajú testovateľnosťou aplikácií ale aj samotných frameworkov. Analyzuje sa všeobecná úroveň zložitosti nastavenia a spojazdnenia testov na lokálnom stroji a podpora pre rozličné typy testov. V kontexte tejto práce sú testami nazývané automatické testy. Jedná sa o testy, u ktorých nie je potrebný ručný zásah vývojárov alebo testerov.

Pokrytie zdrojového kódu testami

Väčšina moderných frameworkov má jednotkovými testami pokrytú väčšinu svojich komponent. Býva taktiež pravidlom, že existujú pravidlá pre prispievanie do projektu, ktoré striktne trvajú na testovaní implementácie nových funkcionalít alebo opráv chýb. Pri tomto nedostatku sa môže stať, že zmeny nebudú schválené a pridané do zdrojového kódu.

Kritériom je analyzované pokrytie zdrojového kódu frameworku testami. Počet percent reprezentuje získanú hodnotu z intervalu $(0,1)$, napr. pokrytie 70% znamená získanie 0,7 bodu.

Konfigurácia testovania

Jedná sa o subjektívne hodnotené kritérium, ktoré bude analyzovať časovú a konfiguračnú zložitosť pre jednotlivé frameworky. Patrí sem základná a pokročila konfigurácia testov, inštalácia potrebných testovacích knižníc a pred pripravenosť aplikácií na to, aby bola testované.

Podporované úrovne testovania

Testovanie webových aplikácií môže prebiehať na viacerých úrovniach. Buď podľa úrovne granularity alebo veľkosti najmenšej jednotky, ktorá je testovaná (ŠPLÍCHALOVÁ, 2008, s. 10).

Autorka definuje nasledujúce 4 základné typy testov:

- *Unit testy* – testované sú malé jednotky.
- *Integračné testy* – integrácia viacerých jednotiek a overenie vzájomného fungovania, u webových aplikácií je najbežnejším príkladom prepojenie s databázou.
- *Systémové testy* – overujú software ako celok, nazývané taktiež funkčné testy.
- *Akceptačné testy*.

Keďže sa v tomto kritériu zaoberáme len automatickými testami, posledný typ môžeme z analýzy vylúčiť. Hodnotí sa teda možnosť testovania aplikácie na vyššie definovaných úrovniach.

3.5. Podpora a iné

Posledná kategória sa zaoberá podporou pre vývojárov, kóderov alebo ľudí, ktorí uvažujú nad tým, že začnú jeden z analyzovaných framework používať.

Kvalita oficiálnej dokumentácie

Pre vývojárov, ktorí chcú začať používať nový framework je dôležité aby mali miesto, kam sa pozrú, ak riešia problém. Týmto miestom je oficiálna dokumentácia. Tá musí byť vždy aktuálna a mala by obsahovať aj dodatočné materiály, napríklad:

- Ukážkovú aplikáciu,
- príklady problémov a riešení,
- vysvetlené odporúčaných praktík,
- ozrejmenú a pokrytú čo najväčšiu časť funkcionalít daného frameworku.

Aktivita v diskusných fórach

Komunikácia o frameworku je dôležitým aspektom aktívnej komunity. Užívatelia a vývojári môžu vyjadriť svoj názor, ale taktiež si môžu pomôcť navzájom pri riešení problémov. Keďže na

internete existuje veľké množstvo diskusných fór a bolo by ťažké ich všetky zmapovať, kritérium sa zameria len na nasledujúce 2:

- Počet diskusií v najväčšej skupine na portály groups.google.com.
- Počet položených otázok na portály Stack Overflow.

Dostupnosť kvalitných zdrojov

Za kvalitné zdroje sú považované knihy. Nie je v silách autora práce sčítať počet dostupných článkov alebo návodov, ktoré sa venujú danej tematike. Kritérium teda hodnotí dostupnosť knižiek napísaných výhradne o danom frameworku.

Framework s vyšším počtom kníh bude ohodnotený hodnotou 1. Druhá hodnota bude vypočítaná úmerne k prvej.

Podpora vo vývojových prostrediach

Medzi dôležité aspekty patrí podpora frameworkov vo vývojových prostrediach. Tá napr. uľahčuje prácu pri auto dopĺňaní názvu tried, automatického aliasingu (importu) tried alebo špecifického zobrazenia štruktúry súborov v projekte.

Skúmaná bude podpora frameworkov v nasledujúcich dvoch vývojových prostrediach:

- Netbeans 7,
- JetBrains PHPStorm 7.

Krivka učenia

Ide o najsubjektívnejšie kritérium, ktoré analyzuje aké komplikované je pochopiť základné princípy frameworku. Berie do úvahy ako komplikované bolo vytvorenie ukážkovej aplikácie a aké problémy pri tejto implementácií bolo potrebné riešiť. Čím komplikovanejšie bolo učenie sa práce s frameworkom, tým nižšie je hodnotenie.

4. Výber frameworkov

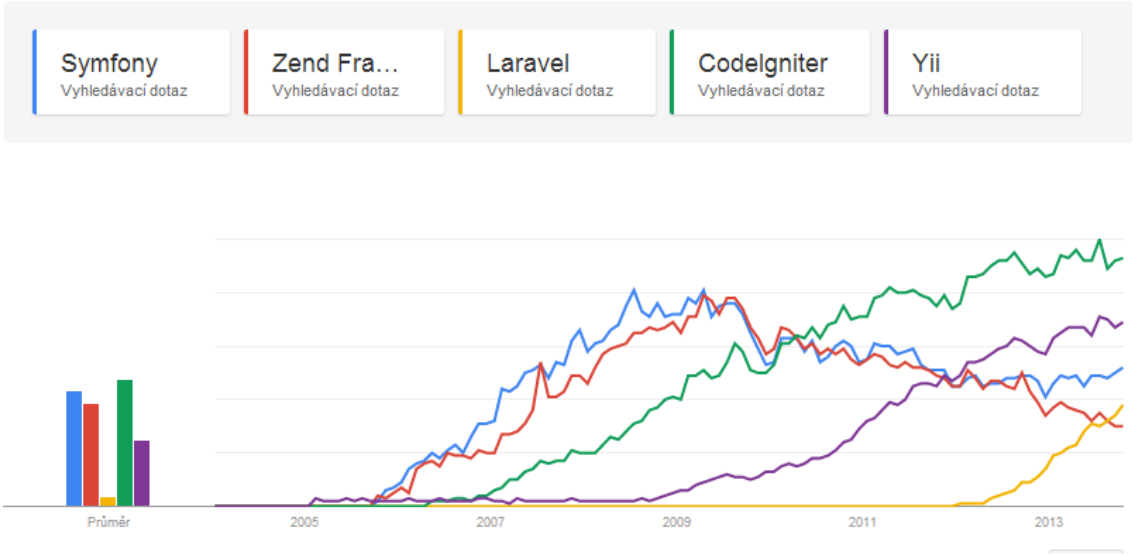
Cieľom kapitoly je výber dvoch PHP frameworkov, ktoré budú výsledne porovnané podľa stanovených kritérií z prechádzajúcej kapitoly. Výber je postavený na jednoduchej analýze popularity jednotlivých projektov z viacerých perspektív:

- Všeobecný záujem o framework,
- záujem vývojárov,
- aktivita komunity.

4.1. Kritéria výberu

Popularita vo vyhľadávačoch

Kritérium sa radí pod kategóriu *všeobecný* záujem o framework. Ide o množstvo vyhľadávaní kľúčových slov vo vyhľadávači Google. Pomocou nástroja Google Trends je možné vidieť trendy vo vyhľadávaní v priebehu času od roku 2004. Vyhľadávané boli názvy jednotlivých frameworkov. Výsledok môže nekorektne vyjadrovať reálny stav, pretože sa neberie do úvahy verzia daného frameworku. U oboch boli vydané dve hlavné verzie, ktoré sú nezávislé a spätne nekompatibilné.



Graf 1 - Počty vyhľadaní definovaných kľúčových slov v Google Trends. [Zdroj: GOOGLE INC, 2013b]

Výsledný graf poukazuje, že najhľadanejším framworkov je *CodeIgniter,* po ktorom nasleduje *Yii* a *Symfony*. Zaujímavý je prepad hľadania frázy *Zend Framwork*. Hoci bola v roku 2012 vydaná nová hlavná verzia frameworku. Laravel od svojho vzniku získava na popularite.

Popularita na serveri Stack Overflow

Stack Overflow je webová stránka, ktorá umožňuje svojím užívateľom klásť a zároveň zodpovedať otázky týkajúce sa programovania. Ide teda o platformu, kde si môžu vývojári navzájom pomáhať.

Kritérium meria popularitu na základe počtu položených otázok so štítkom určeným pre daný framework. Jednotlivé hlavné verzie majú vlastný štítok, napr. *symfony2* či *zend-framework2* a teda oproti predchádzajúcemu kritériu už nevzniká žiadna nejednoznačnosť vo výsledkoch.

Tabuľka 2 - Počet položených otázok na servery Stack Overflow [Zdroj: STACK EXCHANGE, Inc., 2013]

	Počet otázok	Vznik štítku
Zend Framework 2	2,981	August 2011
Symfony 2	14,134	Apríl 2011
Laravel	3,967	December 2011
CodeIgniter	23,179	Október 2010
Yii	6,758	Február 2011

Najviac položených otázok sa týka frameworku *CodeIgniter*, ktorý nasleduje *Symfony 2*. Je dôležité taktiež zohľadniť dátum vzniku jednotlivých štítkov. Najstarším je taktiež *CodeIgniter*, takže počet otázok sa rozloží do väčšieho časového úseku.

Počet sledujúcich užívateľov

Nasledujúce kritérium radíme do kategórie *záujem vývojárov*. Jedná sa o počet užívateľov webovej služby GitHub, ktorí označili projektové repozitáre pomocou funkcie *Star.* Jedná sa o spôsob, akým vývojári sledujú zmeny v danom projekte. Okrem iného sa daná funkcia používa aj na označovanie zaujímavých projektov.

Tabuľka 3 - Počet užívateľov služby GitHub.com sledujúcich repozitáre frameworkov [Zdroj: GITHUB, Inc., 2013]

	Počet užívateľov
Zend Framework 2	4,366
Symfony 2	7,271
Laravel	6,585
CodeIgniter	6,598
Yii	3,161

Z Tabuľka 3 je vidieť, že najsledovanejším PHP frameworkom na portále GitHub je *Symfony 2*. Prekvapivo, na druhom mieste je *Laravel*, hoci ide o najmladší projekt.

Všetky nasledujúce kritéria radíme pod aktivitu komunity, teda vývojárov, ktorí daný framework vyvíjajú. V týchto kritériách nie je braná do úvahy aktivita týkajúca sa dokumentácie a iných dostupných zdrojov.

Počet prispievateľov do zdrojového kódu

Všetky projekty sú vyvíjané ako open-source a teda každý sa môže zaradiť medzi prispievateľov do zdrojového kódu. Vývoj všetkých projektov je umiestnený na portále GitHub, ktorý je v súčasnosti najpopulárnejšou hostingovou službou pre vývoj open-source softwaru (FINLEY, 2011).

V danom kritériu hodnotíme celkový počet prispievateľov do zdrojového kódu frameworkov ku dňu 26. 10. 2013. Do tohto čísla sa nezarátavajú prispievatelia, ktorí majú na starosť oficiálnu dokumentáciu jednotlivých projektov.

Tabuľka 4 - Celkový počet prispievateľov do zdrojového kódu [Zdroj: GITHUB, Inc., 2013]

	Počet prispievateľov
Zend Framework 2	481
Symfony 2	775
Laravel	203
CodeIgniter	261
Yii	185

Z výsledných čísel je možné usúdiť, že *Symfony 2* a *Zend Framework 2* majú najväčší počet prispievateľov do projektu.

Mesačná aktivita prispievateľov

Mimo počtu aktívnych prispievateľov, je potrebné sledovať aj ich aktivitu na projekte. Toto kritérium sleduje počet zmien tzv. commitov v projektovom repozitári za 30 dní v časovom úseku 26.9.2013 až 26.10.2013.

Tabuľka 5 - Počet zmien za 30 dní v projektovom repozitáry [Zdroj: GITHUB, Inc., 2013]

	Počet zmien
Zend Framework 2	388
Symfony 2	360
Laravel	22
CodeIgniter	72
Yii	53

Medzi najaktívnejšie projekty môžeme zaradiť *Zend Framework 2* a *Symfony 2*. Najmenej aktívnym projektom je *Laravel*.

Obecná aktivita vývoja frameworku

Dôležitým faktorom je aj pravidelnosť aktualizácie. Ide o dátum vydania poslednej verzie frameworku. Do úvahy je brané aj to, že *Zend Framework* a *Symfony* majú definovaný pravidelný release cyklus, ktorý je striktne dodržiavaný.

Tabuľka 6 - Dátumy vydania posledných verzií frameworkov [Zdroj: GITHUB, Inc., 2013]

	Dátum vydania poslednej verzie	Verzia
Zend Framework 2	26. 8. 2013	2.2.4
Symfony 2	10.10.2013	2.3.6
Laravel	13. 10. 2013	4.0.9
CodeIgniter	29.7.2013	2.1.4
Yii	11.8.2013	1.1.14

Posledné verzie boli vydané frameworkmi *Laravel* a *Symfony 2*.

4.2. Vyhodnotenie výberu

Predchádzajúca kapitola rozobrala jednotlivé kritéria voľby PHP frameworkov, ktoré budú v tejto práci porovnávané.

Prvým zvoleným je *Symfony 2*, ktorý dominoval vo všetkých kritériách s výnimkou vyhľadávania v Googli. Ide o framework s najväčšou a veľmi aktívnou vývojárskou komunitou.

Druhým zvoleným je *Zend Framework 2*. Hoci v kritériách všeobecného záujmu veľmi neuspel, exceloval v aktivite komunity. Je vidieť, že za týmto projektom stojí množstvo vývojárov a nejde len o dočasný projekt.

Veľmi dôležitým aspektom, ktorý taktiež ovplyvnil výber, je projektová orientácia. Teda na aké typy projektov sa dané frameworky orientujú. Vývojové tímy oboch zvolených projektov brali už od začiatku vývoja do zreteľa to, že budú používané aj pre vývoj veľkých podnikových aplikácií.

5. Ukážková aplikácia

Spôsob ktorým budú overené kritéria nadefinované v kapitole 3, je vytvorenie webovej aplikácie pomocou oboch frameworkov. Pri implementácií budú demonštrované vývojové postupy pre jednotlivé projekty. Aplikácie bude mať identickú funkcionalitu, čo znamená, že užívateľ by nemal spoznať rozdiel. Samotná implementácia môže byť u jednotlivých riešení odlišná, pretože pri vývoji budú použité všetky prostriedky daného frameworku, ktoré prácu urýchľujú.

5.1. Popis aplikácie

Vyvíjaná aplikácia bude webová služba na zber spätnej väzby od registrovaných aj neregistrovaných užívateľov. Hlavným účelom je poskytnutie priestoru ľudom vyjadriť svoje pocity, teda pochváliť alebo posťažovať sa na služby vo svojom okolí. S tým súvisí aj poskytnutie týchto informácií firmám, ktoré na základe tejto spätnej väzby môžu zlepšiť svoje nedostatky.

Keďže firmy v budúcnosti môžu byť vytvárané aj užívateľmi, nie je možné zaručiť, že pôjde len o firmy z určitého segmentu (napr. reštaurácie) a preto je hodnotiaci formulár pomerne všeobecný. Užívatelia budú hodnotiť aktuálny pocit k danej spoločnosti pomocou emotikony. Pre objasnenie ich hodnotenia budú mať možnosť pridať aj dodatočný komentár.

5.2. Požadovaná funkcionalita

Z popisu v prechádzajúcej kapitole je zrejmá základná požadovaná funkcionalita ukážkovej aplikácie. Obrázok 1 zobrazuje diagram prípadov použitia tejto aplikácie. Mimo spomínané funkcie registrácie, prihlásenia a hodnotenia firiem, bude aplikácia umožňovať pre registrovaného užívateľa aj sledovanie vlastných hodnotení a editácie profilu. V aplikácií budú taktiež existovať tzv. firemné účty. Pôjde o užívateľov spojených s určitou firmou. Títo užívatelia budú mať možnosť sledovať jednotlivé komentáre udelené ich spoločnosti.

Posledným typom užívateľov budú správcovia aplikácie, ktorí majú na starosť kompletnú administráciu všetky zdrojov, teda hodnotení, firiem a užívateľov.

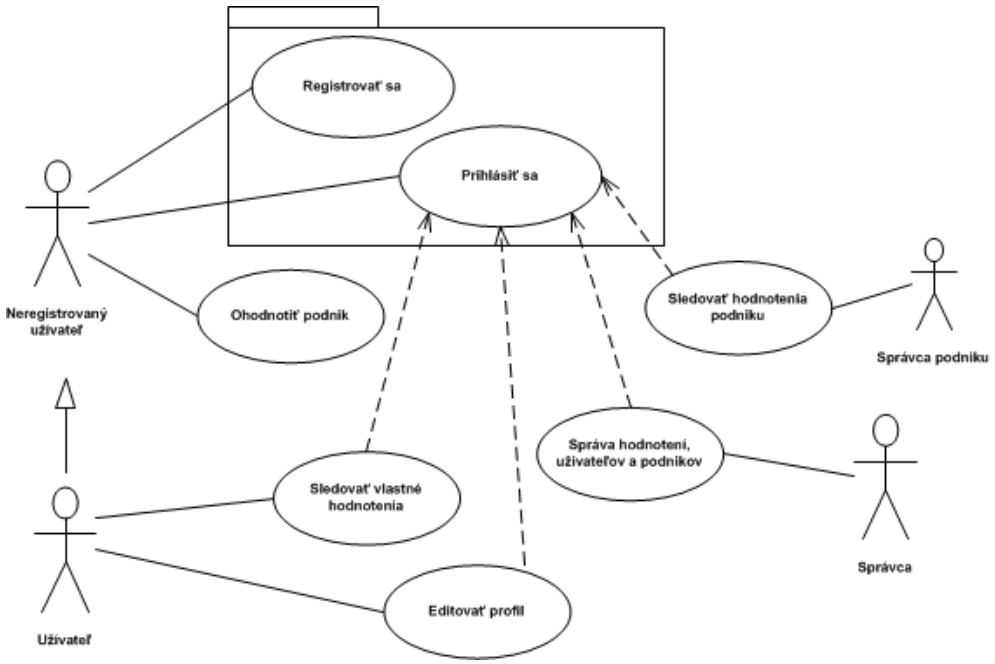
5.2.1. Špecifikácia prípadov použitia

Registrácia

Na stránke registrácie bude mať neregistrovaný neprihlásený užívateľ možnosť si vytvoriť nový účet. Po úspešnom vyplnení a odoslaní registračného formulára sa bude môcť užívateľ prihlásiť do aplikácie.

Hodnotenie podniku

Pridávanie hodnotení je dostupné len pre neregistrovaných užívateľov a registrovaných užívateľov, ktorí nie sú správcami žiadneho podniku v systéme. Prihlásenie a registrácia nie je pre túto akciu povinná a to hlavne z toho dôvodu, aby každý užívateľ mohol vyjadriť spätnú väzbu.



Obrázok 1 - Diagram prípadov použitia ukážkovej aplikácie

Sledovanie vlastných hodnotení

Po úspešnom prihlásení má možnosť registrovaný užívateľ (s výnimkou správcov podnikov) sledovať ním pridané hodnotenia. V súčasnom riešení nebude možné už odoslané hodnotenie upravovať.

Úprava profilu

Prihlásený užívateľ bude mať možnosť editovať svoje osobné údaje, ktoré sú zobrazované v jeho profile.

Sledovanie hodnotení podniku

Po úspešnom prihlásení bude môcť správca podniku, teda užívateľ asociovaný s určitým podnikom v systéme, sledovať anonymné hodnotenia od užívateľov.

Správa hodnotení, užívateľov a podnikov

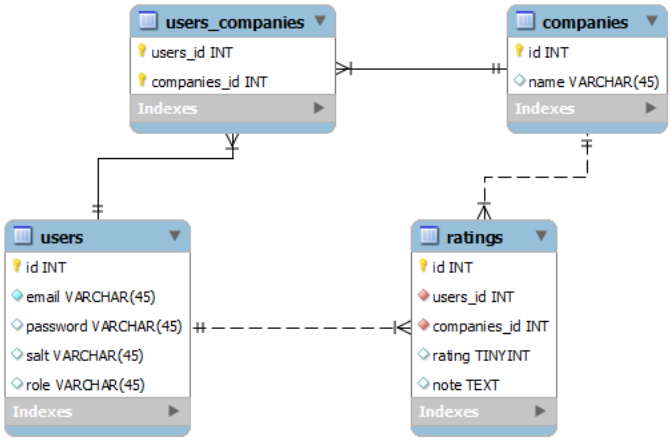
Správcovia aplikácie bude mať ako jediní možnosť editovať všetky dostupné zdroje v aplikácií. Teda možnosť vytvárať, editovať a mazať hodnotenie, užívateľov aj podniky. Taktiež im to umožní asociovať užívateľov k podnikom alebo mazať neslušné hodnotenia.

Správcovia aplikácie majú automatické právo na všetky dostupné akcie v systéme, či už pre regulárnych užívateľov alebo správcov podnikov.

5.3. Fyzický dátový model

Obrázok nižšie znázorňuje fyzický dátový model ukážkovej aplikácie. Jedná sa o návrh MySQL databázy. Základnou entitou v aplikácií je *užívateľ*. Evidujú sa u neho prihlasovacie údaje a rola v systéme. Druhou entitou je *spoločnosť*, ktorú užívatelia v aplikácií hodnotia. Užívateľ s rolou *správca firmy* je asociovaný so spoločnosťou pomocou väzobnej tabuľky. Poslednou entitou je *hodnotenie*, u ktorého je zoznámený užívateľ, ktorý dané hodnotenie zadal, hodnotenie a firma, ktorá je hodnotená.

Tento dátový model obsahuje len všeobecný návrh štruktúry pre aplikáciu. Zaznamenáva hlavné dáta, ktoré sú potrebné pre funkčnosť aplikácie. Jednotlivé riešenia môžu dané entity rozšíriť o dodatočné informácie, napr. čas vytvorenia či aktualizácie záznamu.



Obrázok 2 - Fyzický dátový model ukážkovej aplikácie

5.4. Technické požiadavky

K funkčným požiadavkám definovaným v kapitole 5.2 je potrebné doplniť technickú špecifikáciu pre aplikáciu. Obe aplikácie musia mať implementovanú:

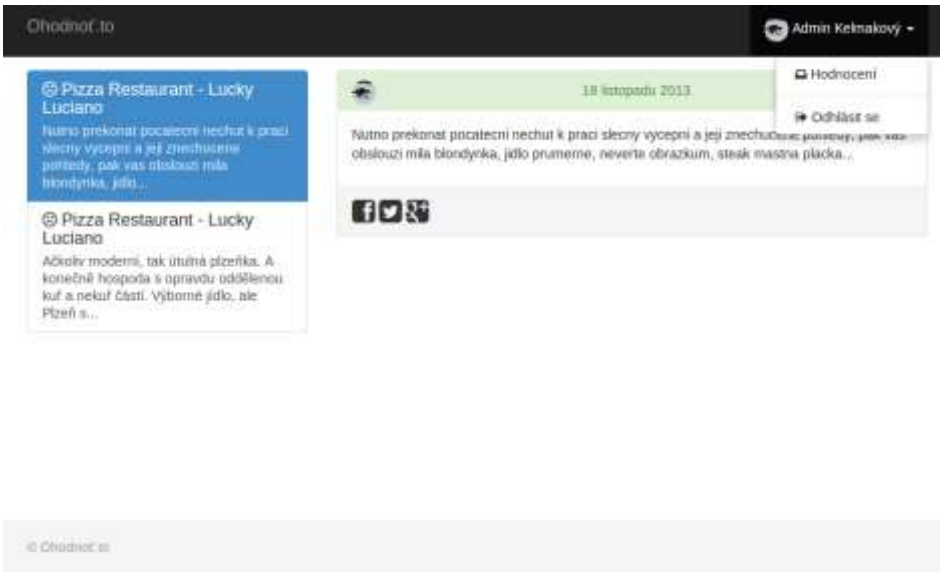
- autentizáciu s využitím databázy MySQL,
- bezpečné vytváranie a ukladanie hesiel,
- autorizáciu.

Mimo vyššie spomínané podmienky, musí byť aplikácia taktiež zabezpečená proti základným spôsobom útokov ozrejmených v kapitole 6.7.2.

Hoci sa jedná sa o pomerne jednoduchú webovú aplikáciu, jej rozsah by mal byť dostačujúci na overenie možností jednotlivých frameworkov.

5.5. Návrh užívateľského rozhrania

Ukážková aplikácia slúži na pridávanie hodnotení a teda návrh rozhrania bol zameraný prevažne na podstatnú funkcionalitu aplikácie. Z tohto dôvodu neboli navrhnuté všetky obrazovky, ktoré boli špecifikované v kapitole 5.2. Rozhranie bolo navrhované v programe Axure RP a jednoduchý HTML prototyp bol vytvorený pomocou knižnice Bootstrap 3.

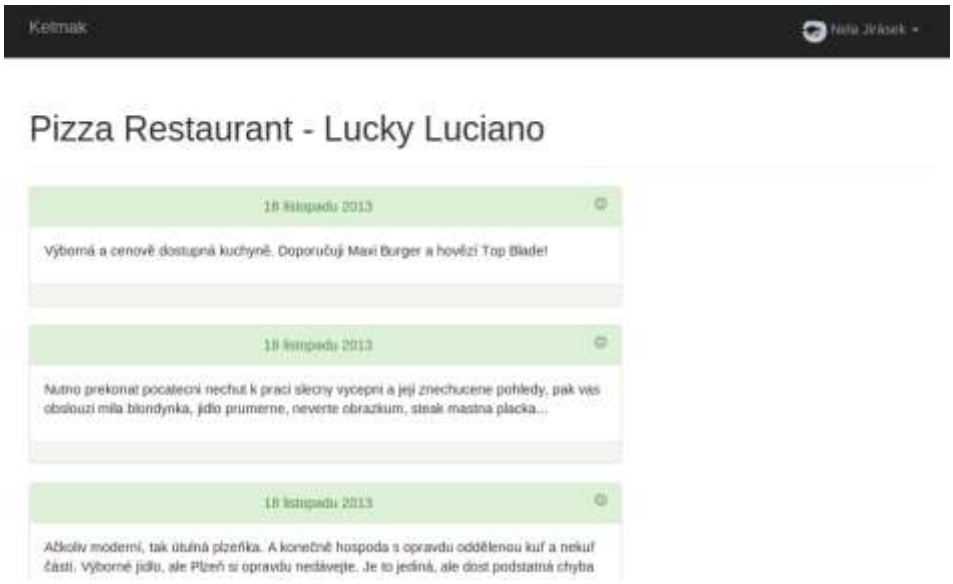


Obrázok 3 - Detail hodnotenia a bočný panel s posledným užívateľovými hodnoteniami

Keďže tvorba grafických podkladov a kaskádových štýlov neovplyvňujú hodnotenie frameworkov, bude rozhranie po grafickej stránke veľmi jednoduché. Stránky budú upravené tak, aby nepôsobili nedokončene a boli použiteľné. Rozhranie aplikácie sa skladá z troch základných komponent. Ide o hore ukotvenú hlavnú navigáciu s rozbaľujúcou sa ponukou, dole ukotvenou pätičkou a hlavným obsahom stránky.

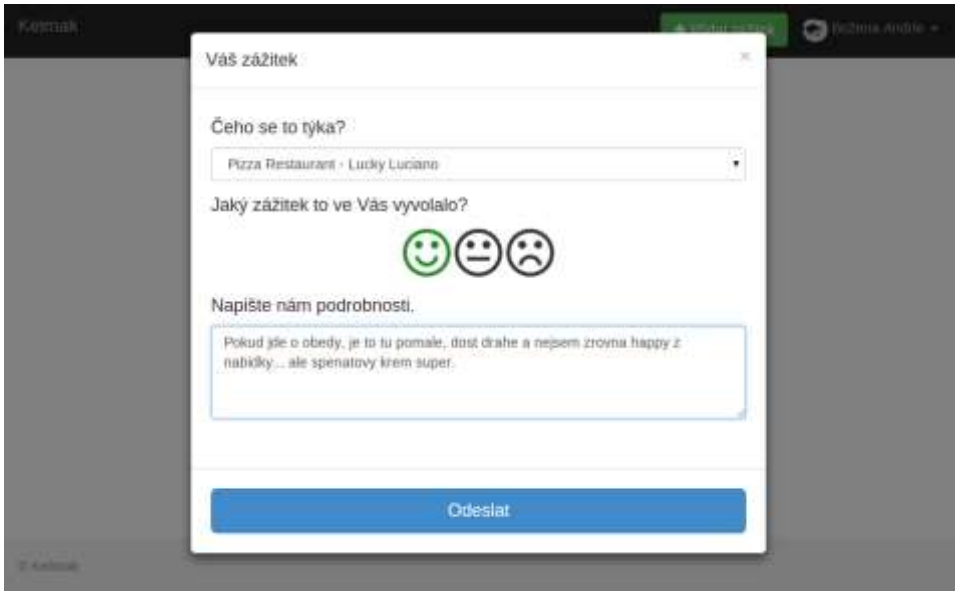
Na Obrázok 3 je zobrazené rozhranie pre regulárneho užívateľa aplikácie. Základná obrazovka ukazuje v bočnom paneli naposledy pridané hodnotenia. Tieto ovládacie prvky vedú na stránku detailu hodnotenia. V obsahovej časti stránky je načítané posledné užívateľovo hodnotenie.

Na obrázku nižšie je vidieť rozhranie pre užívateľov, ktorí sú asociovaní s určitou firmou v systéme. Obsahová časť stránky pozostáva z anonymného zoznamu všetkých hodnotení, ktoré boli pre daný podnik zadané. V navigácií už nie je dostupný ovládací prvok na vytvorenie hodnotenia, pretože títo užívatelia na to nemajú právo.



Obrázok 4 - Zoznam hodnotení podniku

Poslednou navrhnutou obrazovkou je dialóg na pridanie hodnotenia podniku. Ja navrhnutý ako veľmi jednoduchý formulár, ktorý má minimalizovať počet potrebných úkonov potrebných na ohodnotenie. Už pri návrhu bolo zámerom prispôsobiť aplikáciu tak, aby bola ľahko použiteľná aj na mobilných zariadeniach.



Obrázok 5 - Dialóg na pridávanie hodnotení

6. Vývoj ukážkovej aplikácie

Nasledujúca kapitola ukazuje na praktickom príklade ako bola vytvorená ukážková aplikácia pomocou oboch aplikačných frameworkov. Zameriava sa prevažne na časti vývoja, ktoré sú spojené s definovanými kritériami z kapitoly 3.

Ukážkové aplikácie boli vyvíjané na operačnom systéme Xubuntu 13.10 s nainštalovaným PHP verzie 5.5.5 a MySQL serverom verzie 5.5.34. Aplikácie boli distribuované buď pomocou webového serveru Apache alebo pomocou vstaveného webového serveru, ktorý je dostupný v PHP od verzie 5.4.

6.1. Inštalácia, konfigurácia a spustenie aplikácie

Inštalácia frameworku

Zdrojový kód oboch frameworkov je umiestnený na serveri GitHub.com a pomocou verzovacieho systému Git a jeho štítkov je možné získať framework v požadovanej verzii veľmi jednoducho. Samotné frameworky obaľujú a integrujú viacero komponent, ktoré sú na sebe nezávislé a sú vyvíjané vo vlastných repozitároch. Pri vývoji je odporúčané vždy používať stabilné verzie frameworkov, modulov, či knižníc tretích strán, pretože bývajú otestované a obsahujú menší počet chýb.

Komunity oboch frameworkov vytvorili tzv. štartovacie alebo štandardné repozitáre, ktoré obsahujú základnú kostru aplikácie. Využívajú vyššie spomínané repozitáre frameworkov, ale aj doporučené knižnice tretích strán. Tieto balíčky sú preferovaným spôsobom inštalácie, pretože v sebe majú zakomponované dobré návyky a odporúčania vývojárov.

V kapitole 3.2 bol v skratke predstavený nástroj *composer*, ktorý je v súčasnosti de facto štandardom na riešenie závislostí u PHP aplikácií. Obe štartovacie projekty obsahujú súbor `composer.json`, ktorý definuje závislosti projektu. Mimo tohto súboru obsahujú tieto repozitáre taktiež:

- Určitú preddefinovanú adresárovú štruktúru,
- základné nastavenia aplikácie.

Tieto distribúcie majú rozličné názvy, pre Symfony to je Symfony Standard Edition[6] a pre Zend Framework je to ZendSkeletonApplication[7].

[6] <https://github.com/symfony/symfony-standard>
[7] <https://github.com/zendframework/ZendSkeletonApplication>

Composer je možné používať aj na vytvorenie nových projektov. Pomocou príkazu `create-project` je stiahnutá požadovaná distribúcia na predefinované miesto. Pri tejto akcií sú taktiež stiahnuté všetky požadované knižnice tretích strán, ktoré sú nastavené ako závislosti tohto projektu. Vďaka tomu, že je composer napísaný v PHP (rozšírenie PHAR) je tento skript multiplatformový a priebeh inštalácie a správa závislostí je na všetkých operačných systémoch identická.

Výpis 1 - Vytvorenie nového projektu pomocou nástroja Composer

```
php composer.phar create-project symfony/framework-standard-edition
/var/www/symfony-project 2.3.7
```

Konfigurácia

Väčšina konfigurácie u Symfony aplikácie sa vykonávajú v priečinku `app/cofig` v jednotlivých príslušných yaml súboroch. Štandardná distribúcia počíta s tým, že aplikácia bude využívať databázu. Tým že sú konfiguračné súbory parametrizované, je možné využiť súbor `parametrs.yml`, do ktorého stačí pridať prístupové údaje k MySQL serveru a zvoliť databázu, s ktorou má aplikácia pracovať.

Výpis 2 - Nastavenie pripojenia aplikácie k MySQL serveru v Symfony 2

```
parameters:
    database_driver: pdo_mysql
    database_host: 127.0.0.1
    database_port: null
    database_name: test
    database_user: root
    database_password: root
```

U Zend Frameworku je potrebný ešte manuálny zásah do konfigurácie aplikácie. Správnym nastavením `ServiceManagera` bude načítaný požadovaný databázový adaptér, ktorý umožní komunikáciu s databázou. Toto nastavenia sa definuje v konfiguračnom súbore `global.php`, ktorý sa nachádza v priečinku `config/autoload`.

Výpis 3 - Nastavenie pripojenia aplikácie k MySQL serveru v Zend Frameworku

```
return array(
    'db' => array(
        'driver'=> 'Pdo',
        'dsn' => 'mysql:dbname=test;host=localhost',
        'driver_options' => array(
            PDO::MYSQL_ATTR_INIT_COMMAND => 'SET NAMES \'UTF8\''
        ),
    ),
    'service_manager' => array(
        'factories' => array(
            'Zend\Db\Adapter\Adapter' =>
                    'Zend\Db\Adapter\AdapterServiceFactory',
        ),
    ),
);
```

Citlivé údaje ako užívateľské meno a heslo je potrebné umiestniť do separátneho súboru `local.php`.

Výpis 4 - Definícia prístupových údajov k databázovému serveru v Zend Frameworku

```
return array(
    'db' => array(
        'username' => 'root',
        'password' => 'root',
    ),
);
```

Spustenie aplikácie

Vydanie PHP verzie 5.4 prinieslo do vývoja aplikácií malú revolúciu. Touto novinkou bol vstavaný webový server, ktorý sa dá pustiť z príkazového riadku. Už teda nie je potrebné na lokálnom stroji inštalovať robustný webový server ako je napr. Apache, pretože výkonovo úplne dostačuje ako vývojové prostredie. V produkčnom prostredí sa tento server neodporúča používať.

Pre rýchle overenie, že boli frameworky úspešne nainštalované, základná webová aplikácia beží a je správne nakonfigurovaná, stačí spustiť spomínaný webový server.

Spustenie webovej aplikácie vytvorenej pomocou Zend Frameworku:

```
$ php -S 0.0.0.0:8080 -t public/ public/index.php
```

Spustenie webovej aplikácie vytvorenej pomocou Symfony:

```
$ php app/console server:run
```

Po zadaní týchto príkazov sú webové aplikácie dostupné na adresách http://localhost:8000/ a http://localhost:8080/.

Využitím vstaveného serveru je Symfony webová aplikácia spustená vo vývojárskom prostredí. Pri využití webového serveru ako je Apache, je toto prostredie dostupné na URL adrese <adresa servera>/app_dev.php/. Symfony ponúka taktiež rýchle overenie, či lokálne prostredie spĺňa minimálne požiadavky pre vývoj aplikácií s týmto framework. Stačí navštíviť URL adresu http://localhost:8000/config.php.

6.2. Komponenty

Ako bolo v kapitole 3.3.6 definované, komponenty by mali byť navrhované zo zreteľom na jednoduchosť a znova použiteľnosť. Obe frameworky vsádzajú na tento koncept. Samé sú tvorené veľkým počtom menších komponent. Vyvíjané aplikácie by mali taktiež nasledovať tento princíp a mali by byť rozčlenené do menších celkov, ktoré riešia jednu určitú funkcionalitu. Každý framework má vlastné pomenovanie pre komponenty, ale funkciu v rámci aplikácií spĺňajú identickú.

V Symfony 2 sú nazývané komponenty ako *Bundles*. Dokumentácia definuje bundle ako „štruktúrovanú sadu súborov v jednom priečinku, ktoré implementujú jednu vlastnosť, či funkcionalitu aplikácie. Každý bundle obsahuje všetky súbory súvisiace z danou funkciou. To znamená PHP súbory, šablóny, súbory kaskádových štýlov, JavaScriptové súbory, testy a iné“ (SENSIOLABS, 2013).

Vlastné komponenty vytvorené pre aplikáciu sú umiestnené v priečinku `src`. Všetky komponenty tretích strán sú načítavané z externých úložísk, ako napríklad zo serveru Githubu, pomocou nástroja Composer. Inštalované sú do priečinka `/vendor`. Pri tvorbe vlastný komponent je potrebné dodržať konvenciu pomenovania. Názov priečinku aj hlavného súboru musí obsahovať suffix Bundle. Napr. `TestPostBundle/TestPostBundle.php`. Posledným krokom je už len registrácia novej komponenty v konfiguračnom súbore `AppKernel.php`.

Tým, že Symfony vsádza na znova použiteľnosť bundles, je dobré dodržiavať taktiež konvencie[8] adresátovej štruktúry. V rámci znova použiteľnosti je dôležité riešiť aj konfiguráciu modulov. Komponenty vystavujú konfiguračné možnosť pomocou služieb. Vývojári si vďaka tomu môžu prispôsobiť správanie podľa svojich požiadaviek.

[8] <http://symfony.com/doc/current/cookbook/bundles/best_practices.html>

Výpis 5 - Nastavenie komponenty FOSUserBundle

```
fos_user:
    db_driver: orm
    firewall_name: main
    user_class: Kelmak\CoreBundle\Entity\User
```

Ústredné miesto, kde komunita poskytuje svoje komponenty, je už viac krát spomínaný server Github. Agregátor KNPbundles[9] ponúka zoznam dostupných komponent a vytváraná platformu na ich hľadanie a hodnotenie. V súčasnosti ponúka 2142 registrovaných komponent (KNPLABS, 2012). Výsledný počet určite nie je finálny, pretože na tomto portáli určite nie sú zaregistrované všetky existujúce komponenty. Komunita, ktorá sa vytvorila okolo tohto frameworku, si snaží pomáhať tým, že vytvára kvalitné komponenty. Veľmi dobrým príkladom je *FOSUserBundle*[10], ktorý integruje do aplikácií funkcionalitu užívateľov. Pozostáva zo všetkých základných funkcií ako prihlásenia, registrácie, editácie profilu a iných.

V Zend Frameworku sú komponenty nazývané *moduly*. Všeobecne fungujú na rovnakom princípe ako bundles v Symfony. Každý modul je umiestnený v separátnom priečinku a má na starosť vytvorenie určitej funkcionality. Aby boli moduly funkčné, je ich v konfiguračnom súbore aplikácie `application.config.php` potrené zaregistrovať.

Výpis 6 - Registrácia modulu v Zend Frameworku

```
'modules' => array(
    'Application',
    'Rating',
),
```

Najväčší rozdiel medzi modulmi a bundles je v ich v konfigurácií. Symfony uľahčuje prácu vývojárom tým, že viacero časti bundlu je možné používať bez nutnosti ďalšej konfigurácie. To ale za predpokladu dodržania konvencií. Príkladom môžu byť súbory s prekladmi. Zend Framework oproti tomu razí cestu manuálnej konfigurácii. Každý modul obsahuje konfiguračný súbor `module.config.php`. V ňom je potrebné konfigurovať napríklad: pohľady, radiče, trasy, služby, preklady a podobne. Tieto nastavenia sú predávané do manažéra služieb (*Service Manager*), aby s nimi bolo možné pracovať odkiaľkoľvek z aplikácie a bolo ich možné dodatočne prekonfigurovať.

[9] <http://knpbundles.com/>
[10] <https://github.com/FriendsOfSymfony/FOSUserBundle>

Výpis 7 - Konfigurácia radičov v základnom module Application

```
'controllers' => array(
    'invokables' => array(
        'Application\Controller\Index' =>
            'Application\Controller\IndexController',
        'Application\Controller\Rating' =>
            'Application\Controller\RatingController'
    ),
),
```

Aj komunita okolo Zend Frameworku má portál s agregátorom modulov. Ide o pod sekciu modules[11] na oficiálnych stránkach frameworku. Tá v súčasnosti ponúka 387 modulov (ZEND TECHNOLOGIES LTD, 2006 – 2012).

6.3. Databázy a dátové modely

Na ukážkach zdrojového kódu je vidieť, že obe aplikácie sa pripájali k databázovému serveru MySQL. Frameworky ale poskytujú podporu pre väčšie množstvo databázových systémov ako znázorňuje Tabuľka 7.

Tabuľka 7 - Podporované databázové systémy

	Symfony 2	Zend Framework 2
IBM DB2	Áno	Nie
Microsoft SQL Server	Áno	Áno
MySQL	Áno	Áno
Oracle	Áno	Nie
PostgreSQL	Áno	Áno
SQLite	Áno	Áno

Pre lepšie pochopenie práce s databázami u jednotlivých frameworkov, je potrebné vysvetliť všeobecný princíp pre prácu s databázami v PHP. Pre prácu s určitým typom databázového servera existujú špecifické funkcie napr. `mysql_query`. Využívanie týchto funkcií ale nezapadá do kontextu objektového programovania. Taktiež to môže priniesť problémy v budúcnosti, pri migrácií aplikácie na iný databázový systém. Riešením je využitie rozšírenia *PHP object model* (PDO), ktoré definuje konzistentné rozhranie pre prístup k databázam. PDO poskytuje abstraktnú vrstvu pristupujúcu k dátam. To znamená, že nezávisí na type používaného databázového systému v aplikácií, pretože všetky funkcie na vytvorenie požiadaviek alebo získanie dát sú rovnaké (PHP DOCUMENTATION GROUP, 2013).

[11] <http://modules.zendframework.com/>

Symfony 2

Štandardná distribúcia Symfony využíva pre prácu s databázami knižnicu Doctrine. Samotný framework nie je závislý na tejto komponente a môže ju nahradiť za inú napr. knižnicu Propel.

Doctrine je knižnicou vybudovanou nad PDO. Hlavnou časťou je abstraktná a prístupová databázová vrstva (*DBAL* alebo *Doctrine database abstraction & access layer*), ktorá je medzivrstvou medzi ďalšími komponentmi Doctrine a PDO. Ponúka objektovo orientované rozhranie na manipuláciu s databázami. Tabuľka 7 obsahuje zoznam podporovaných databázových systémov, pre ktoré existuje implementovaný ovládač, ktorý využíva dostupné funkcie rozšírenia PDO. Poslednou časťou používanej knižnice je Doctrine ORM. Zaisťuje mapovanie objektov v aplikácií na relačnú databázu. *ORM* alebo *Object-Relational Mapper* je implementáciou návrhové vzoru Data Mapper, ktorý bol predstavený v kapitole 2.3.

Výpis 8 - Definícia entity hodnotenia v Symfony 2

```
/**
 * @ORM\Entity
 * @ORM\Entity(
 *   repositoryClass="Test\CoreBundle\Repository\RatingRepository"
 * )
 */
class Rating
{
    /**
     * @ORM\Id
     * @ORM\Column(type="integer")
     * @ORM\GeneratedValue(strategy="AUTO")
     */
    protected $id;

    /**
     * @ORM\ManyToOne(targetEntity="Test\CoreBundle\Entity\Place")
     * @ORM\JoinColumn(name="place_id", referencedColumnName="id")
     */
    protected $placeId;

    /**
     * @ORM\Column(type="smallint")
     */
    protected $rating;

    /**
     * @ORM\Column(type="text")
     */
    protected $note;
}
```

Jednotlivé objekty sú definované pomocou tzv. entít. Jedná sa o PHP triedy, ktorú sú podľa Tichého (2010) „základným kameňom v *Doctrine 2 a vôbec celej aplikácie. Každá entita reprezentuje objekt reálneho sveta, takzvaný doménový objekt*“.

Mapovanie prebieha pomocou definície metadát jednotlivých entít. Tento zápis sa deklaruje buď v XML súbore, YAML súbore alebo pomocou anotácií v PHP triede, ktorá reprezentuje entitu. Doctrine poskytuje viacero typov metadát, ktoré je možné definovať. Napríklad definícia typu alebo väzby medzi entitami. Všetky možnosti nastavení sú dostupné v referenčnej príručke[12].

Samotné požiadavky do databázy sú ale vykonávané pomocou tzv. entity repositories. Týmto je oddelená databázová logika od definície modelu. Každá vytvorená entita automaticky využíva štandardnú repository, ktorú spravuje manažér entít (*Entity manager*). Táto repository ponúka množstvo pomocných funkcií na prácu s databázou napr. funkciu *findBy* a jej pokročilé rozšírenia. Tie uľahčujú prácu vývojárom, tým že umožňujú klásť základné požiadavky na server bez nutnosti ďalšej práce.

Výpis 9 - Používanie štandardných funkcií Entity Repository

```
// Získanie repository z Entity managera
$repository = $doctrine->getRepository('TestCoreBundle:Rating');
// Nájdenie všetky hodnotení podľa užívateľa
$userRatings = $repository->findByUserId($this->getUser());
```

Na výpise nižšie je vidieť, že pre entitu hodnotení bola pomocou anotácie definovaná vlastná repository. Tá obsahuje špecifické požiadavky, nadefinované len pre túto entitu.

Výpis 10 - Vytvorenie vlastného dotazu v repository

```
class RatingRepository extends EntityRepository
{
    public function findLatestByUser($user, $limit = 10)
    {
        $queryBuilder = $this->createQueryBuilder('r')
            ->select('r')
            ->where('r.userId = :user_id')
            ->setParameter('user_id', $user)
            ->orderBy('r.createdAt', 'ASC')
            ->setMaxResults($limit);

        return $queryBuilder->getQuery()->getResult();
    }
}
```

[12] <http://docs.doctrine-project.org/en/2.0.x/reference/annotations-reference.html>

Zend Framework

Pre prácu s databázou využíva Zend Framework vlastnú komponentu *Zend Db*. Ako Doctrine, taktiež sa skladá z niekoľkých vrstiev. Najnižšou vrstvou sú *Adaptéry*, ktoré spĺňajú identickú funkciou ako abstraktná vrstva DBAL u Doctrine. Mapovanie relačných databáz je ale u Zend_Db navrhnuté a riešené pomocou návrhových vzorov Row Data Gateway a Table Data Gateway, ktoré boli spomínané v kapitole 2.3. Implementáciou týchto vzorov sú triedy *TableGateway* a *RowGateway*. Pre každú databázovú tabuľku musia teda existovať dve triedy a to trieda, ktorá definuje entitu a trieda, ktorá reprezentuje celú tabuľku.

Modelové triedy sú definované pomocou verejných atribútov triedy. Tieto atribúty reprezentujú stĺpce v tabuľke. Tým, že sú tieto atribúty verejné, nie je potrebné pre nich definovať žiadne funkcie na získavanie a úpravy, tzv. gettery a settery.

Výpis 11 zobrazuje definície tabuľkovej triedy. Tá slúži na tvorbu dotazov nad celou tabuľkou. Funkcia *fetchAllUser* ukazuje spôsob, akým sú vytvárané vlastné databázové dotazy.

Výpis 11 - Definícia objektu reprezentujúceho databázovú tabuľku v Zend Frameworku

```
use Zend\Db\TableGateway\TableGateway;
class RatingTable
{
    protected $tableGateway;

    public function __construct(TableGateway $tableGateway)
    {
        $this->tableGateway = $tableGateway;
    }

    public function fetchAllUser(User $user, $limit = 10)
    {
        $select = $this->tableGateway->getSql()->select();
        $select->where(array('user_id = ?' => $user->id))
            ->order('created_at ASC')
            ->limit($limit);

        return $this->tableGateway->selectWith($select);
    }
}
```

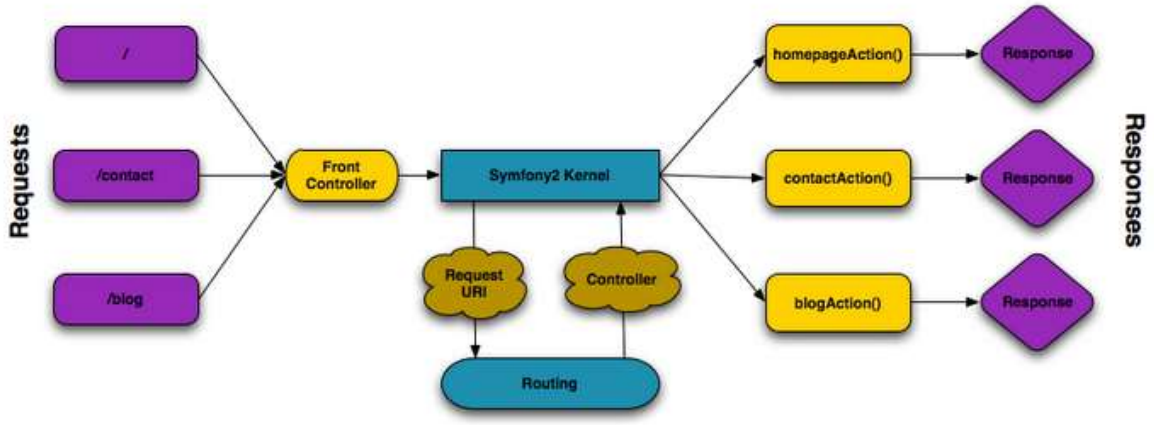
Posledným krokom je nastavenie spôsobu vytvárania inštancií modelových tried, tak ako to znázorňuje Výpis 12. V konfigurácií modulu je nadefinovaná služba, ktorá vytvára novú inštanciu vo chvíli, keď je modelová trieda vyžiadaná prvý krát.

Výpis 12 - Nastavenie spôsobu tvorby inštancií tried

```
public function getServiceConfig()
{
    return array(
        'factories' => array(
            'Application\Model\RatingTable' =>  function($sm) {
                $tableGateway = $sm->get('RatingTableGateway');
                $table = new RatingTable($tableGateway);
                return $table;
            },
            'RatingTableGateway' => function ($sm) {
                $dbAdapter = $sm->get('Zend\Db\Adapter\Adapter');
                $resultSetPrototype = new ResultSet();
                $resultSetPrototype->setArrayObjectPrototype(new Rating());
                return new TableGateway('ratings',
                    $dbAdapter, null, $resultSetPrototype
                );
            },
        ),
    );
}
```

6.4. Radiče

Ako bolo v kapitole 2.3 vysvetlené, úlohou radičov je vytvoriť HTTP odpoveď na základe HTTP požiadavky užívateľa. Webové aplikácie mávajú veľké množstvo akcií a jednotlivé HTTP požiadavky musia byť nejakým spôsobom distribuované do správnych radičov. Na tento účel slúži v oboch frameworkov *Front Controller*, ktorý to vykonáva na základe nadefinovaných trás (*routes*). Každá vytvorená akcia, ak má byť prístupná užívateľom, musí mať nadefinovanú trasu. Na obrázku nižšie je zobrazený priebeh spracovanie http požiadavku vo frameworku Symfony 2. Obdobné riešenie je implementované aj u Zend Frameworku 2.



Obrázok 6 - Priebeh spracovania HTTP požiadavky a routovania v Symfony 2. Zdroj: SENSIOLABS[13]

6.4.1. Smerovanie v aplikácií

Ako takmer u každej konfigurácie v Symfony 2, aj smerovania je možné konfigurovať pomocou XML, YML a PHP súboru. Smerovacia komponeneta *Router* umožňuje pomerne rozsiahle možnosti nastavenie trás. Definícia nižšie poukazuje, akým spôsobom sa vytvára dynamická trasa. Uvedený príklad pošle požiadavku do radiča hodnotení. Okrem toho má nepovinný parameter `page`, ktorý v tomto prípade definuje číslo strany. Pomocou regulárneho výrazu je kontrolované, že tento parameter je číselný.

Výpis 13 - Nastavenie trasy v Symfony 2

```
ratings:
    path:     /ratings/{page}
    defaults: { _controller: TestCoreBundle:Rating:index, page: 1 }
    requirements:
        page:  \d+
```

Trasy v Symfony môžu byť definované na rozličných miestach. V hlavnom konfiguračnou súbore, ktorý sa nachádza v `app/config/routing.yml` alebo v rámci vlastného modulu, napr. `src/Test/CoreBundle/Resources/config/routing.yml`. Najjednoduchším spôsobom je definícia pomocou anotácií, ktorá je dostupná v radičoch až po nainštalovaní modulu *SensioFrameworkExtraBundle*[14].

[13] <http://symfony.com/doc/current/book/routing.html>
[14] <https://github.com/sensiolabs/SensioFrameworkExtraBundle>

Výpis 14 - Definícia trasy pomocou anotácie v radiči

```
/**
 * @Route("/rating")
 */
class RatingController extends Controller
{
    /**
     * @Route("/create", name="rating_create")
     */
    public function createAction()
    {
        // Definícia akcie
    }
}
```

Zend Framework umožňuje jediný spôsob definície tras a to v hlavnom konfiguračnom súbore modulu – `config/module.config.php`. Trasy sú definované pomocou asociatívnych polí.

Výpis 15 - Definícia trasy v Zend Frameworku

```
'router' => array(
    'routes' => array(
        'ratings' => array(
            'type' => 'Segment',
            'options' => array(
                'route' => '/ratings[/:page]',
                'constraints' => array(
                    'page' => '[0-9]+',
                ),
                'defaults' => array(
                    'controller' => 'Application\Controller\Rating',
                    'action'     => 'index',
                )
            ),
        ),
    ),
),
```

V časti `routes` sa definuje formát URL adresy. Pomocou zápisu `[/:page]` je definovaný nepovinný parameter. V časti `constraints` sú pomocou regulárneho výrazu kontrolované vstupné parametre. V sekcií `defaults` je nastavované, kam sa má daná požiadavka nasmerovať, ak URL adresa odpovedá danému predpisu. Typ *segment* sa používa u definície trás s parametrami. Zend Framework ponúka ďalšie typy definícií a to napríklad *Scheme*, *Regex*, *Literal* a iné. Zoznam všetkých typov a ich možnosti sa nachádza v referenčnej príručke[15].

[15] <http://framework.zend.com/manual/2.2/en/modules/zend.mvc.routing.html>

6.4.2. Rozširujúce funkcie radičov

Tvorba funkcií, ktoré sú dostupné vo všetky radičoch je v Zend Frameworku riešená pomocou rozšírení (*plugin*s). Štandardne radiče využívajú niektoré pred vytvorené pluginy, napr. Bleskové správy, či presmerovávač.

Symfony nemá implementované riešenie pre rozšírenie funkcionality len pre radiče. Novú funkcionalitu, ktorá má byť dostupná z viacerých miest, je ale možné nadefinovať ako službu (*service*). Novo vytvorenú službu stačí zaregistrovať do service containeru, odkiaľ si ju už radič môže jednoducho získať.

Obe tieto riešenia sú v súlade z princípom DRY. Symfony ale umožňuje využitie týchto funkcií odkiaľkoľvek, nie len z radičov.

6.4.3. Bleskové správy

Ako bolo vyššie spomínané, bleskové správy sú v Zend Frameworku dostupné pomocou pluginu. Základná funkcia `addMessage` uloží správu do sessions a tým sa stane jednorázovo dostupná pri nasledujúcej HTTP požiadavke užívateľa. Tento plugin poskytuje vstavenú podporu pre 3 typy bleskových správ: informačné, chybné a úspešné.

Výpis 16 - Vytvorenie bleskovej správy v Zend Frameworku

```
$this->flashMessenger()->addSuccessMessage(Váš zažitek byl uložen.');
```

Symfony ma implementovanú funkcionalitu bleskových správ v komponente Sessions. Tá zlučuje súvisiace skupiny sessions do tzv. „*Bags*“. Pre bleskové správy existuje už vytvorená vlastná `FlashBag`, ktorá umožňuje ich tvorbu a správu. Ako prvý parameter funkcie je predávaný typ správy a teda vývojári si môžu definovať vlastné typy správ.

Výpis 17 - Vytvorenie bleskovej správy v Symfony

```
$this->get('session')->getFlashBag()->add(
    'success',
    'Váš zažitek byl uložen'
);
```

6.5. Formuláre

Na príklade formulára na tvorbu hodnotenia budú demonštrovaný spôsob definície a práce s formulármi u oboch frameworkov.

Symfony 2

Tvorba formulárov prebieha pomocou tzv. Form buildera. Ten má na starosti prijať definíciu formulára a na jej základe ho vytvoriť. Formuláre, ktoré sa majú v aplikácií vyskytovať iba raz, je najlepšie definovať v príslušnom radiči. Pre všetky ostatné, ktoré je potrebné znova použiť v aplikácií, je overenou technikou vytváranie nových typov formulárových polí. Tieto zhlukujúce typy polí sú väčšinou používané pri práci s entitami. Výpis 18 zobrazuje tento spôsob definície, kedy pre formulár na tvorbu hodnotenia bol definovaný typ RatingType.

Ako je vidieť z výpise nižšie, **Chyba! Nenalezen zdroj odkazů.** jednotlivé formulárové polia sú efinované a pridávané do staviteľa. Sú definované názvom, typom a určitými vlastnosťami , ktoré sú vo formáte asociatívneho poľa. Niektoré vlastnosti sú všeobecné pre všetky polia, napríklad vlastnosť popis (*label*). Jednotlivé typy polí ale môžu mať určené špecifické vlastnosti, napr. pre typ dátum je to nastavenie formátu.

Výpis 18 - Spôsob definovanie formulárov v Symfony 2

```
class RatingType extends AbstractType
{
    public function buildForm(FormBuilderInterface $builder, array $options)
    {
        $builder
            ->add('rating', 'choice', array(
                'label' => 'Hodnocení',
                'choices' => array(
                    2 => 'Dobrý', 1 => 'Ok', 0 => 'Zlý'
                ),
                'empty_value' => false
            ))
            ->add('note', 'textarea', array(
                'label' => 'Poznámka',
                'trim' => true,
                'required' => false
            ))
            ->add('submit', 'submit', array('label' => 'Odeslat'))
        ;
    }
}
```

Symfony má implementované všetky základné formulárové polia definované v HTML. Taktiež ale ponúka podporu pre nové typy špecifikované v HTML 5 ako napríklad email, URL,

či vyhľadávanie (*search*). V referenčnej príručke[16] v časti o formulároch je zoznam všetkých podporovaných typov aj ich vlastností.

V aplikácií je občas potrebné použiť jeden typ formulárového poľa na viacerých miestach, napr. pohlavie užívateľa. Jedná sa o klasické riešenie pomocou radio poľa s rozdielom, že tento typ má prednastavené určité štandardné vlastnosti - hodnoty môžu byť len muž a žena. Takýto typ rozšírenia je dostupný pomocou vytvorenie nového typu.

Výpis 19 - Asociovanie entity s formulárom v Symfony 2

```
$rating = new Rating();
$form = $this->createForm(new RatingType(), $rating, array(
    'action' => $this->generateUrl('rating_create_submit'),
    'method' => 'POST',
));
```

Veľmi užitočnou funkciou formulárovej komponenty je mapovania entít v aplikácií na jednotlivé formuláre. Ako je vidieť vo výpise vyššie. Týmto jednoduchým spôsobom je možné automaticky načítať dáta z databáze a umiestniť ich do poľa s výberom.

Keďže validácia je častým úkonom pri tvorbe webových aplikácií, je v Symfony implementovaná samostatná komponenta, ktorá ich má na starosti. Používa sa taktiež u validácie formulárov. Formulár obsahuje metódu isValid(), ktorá na základe predefinovaných pravidiel skontroluje užívateľské vstupy.

Ako už bolo viac krát spomenuté, tak aj validácie je možné definovať rozličnými spôsobmi. Okrem základného spôsobu definície pomocou PHP je taktiež možné nadefinovať validáciu konfiguračným yaml a xml súborom, či pomocou anotácie. Posledné tri spôsoby sú dostupné len pre formuláre, ktoré sú používané na prácu s entitou. Asi najjednoduchším riešením je pridanie anotácie pri definícií entity.

Výpis 20 - Definícia validácie v entite užívateľ

```
/**
 * @Assert\NotBlank()
 * @Assert\Length(min = "3")
 */
private $firstName;
```

Zápis pomocou PHP je používaný u formulárov, ktoré nemajú asociovanú žiadnu entitu, pretože pre ne neexistuje iný spôsob definície validácie.

[16] <http://symfony.com/doc/current/reference/forms/types.html>

Výpis 21 - Definícia validácie formulárového elementu

```
$builder->add('lastName', 'text', array(
    'constraints' => array(
        new NotBlank(),
        new Length(array('min' => 3)),
    ),
));
```

Zend Framework

Definícia formulárov je v Zend Frameworku možná viacerými spôsobmi. Prvou metódou je tvorba pomocou továrnej metódy. Formulár je skonštruovaný na základe konfiguračnej definície. Rozsiahle asociatívne pole definuje všetky elementy, obaľujúce skupiny aj nastavenia formulára. Jedná sa o metódu veľmi podobnú tvorbe formulárov u Symfony.

Výpis 22 - Definícia formulára pomocou továrnej metódy

```
$factory = new Factory();
$form = $factory->createForm(array(
    'hydrator' => 'Zend\Stdlib\Hydrator\ArraySerializable',
    'elements' => array(
        array(
            'spec' => array(
                'name' => 'name',
                'options' => array(
                    'label' => 'Poznámka',
                ),
                'attributes' => array(
                    'type'  => 'text'
                ),
            )
        ),
   )
));
```

Oba ďalšie spôsoby už pristupujú k tvorbe formulárov tak, že jednotlivé formulárové elementy sú definované oddelene. Buď pomocou asociatívnych polí alebo vytvorením inštancie formulárového elementu a manuálneho nastavenia jeho vlastností. Výpis 23 demonštruje oba spôsoby tvorby.

Pre definované polia vo výpisoch boli použité základy typy ako text, či element rádio. Tak ako Symfony, aj Zend Frameworku poskytuje pre všetky štandardné formulárové elementy už pred pripravené triedy. Pre vytvorenie vlastných prvkov alebo vlastností je využívaná dedičnosť PHP tried, tak ako napríklad element na výber mesiaca *MonthSelect* je rozšírením všeobecného dátovému elementu.

Výpis 23 - Tvorba formulára v Zend Frameworku

```
namespace Application\Form;

use Zend\Form\Form;
use Zend\Form\Element\Radio;

class RatingForm extends Form
{
    public function __construct($name = null)
    {
        parent::__construct('rating');
        $this->setAttribute('method', 'post');

        $ratingField = new Radio('rating');
        $ratingField->setLabel('Hodnocení')
            ->setValueOptions(array(
                2' => 'Dobrý', '1' => 'Ok', '0' => 'Zlý')
            );
        $this->add($ratingField);

        $this->add(array(
            'name' => 'note',
            'type' => 'Text',
            'options' => array(
                'label' => 'Poznámka',
            ),
        ));
    }
}
```

Validácie a filtrovanie formulárových polí sú riešené pomocou všeobecnej komponenty na filtrovanie vstupov od užívateľov, triedy `InputFilter`. Validácia je riešená na úrovni celých formulárov a nie pri zápise formulárového elementu. Na definíciu sa využíva metóda `getInputFilter`, ako je vidieť na Výpis 24. Pre formuláre, ktoré sú určené na prácu s entitami, je odporúčané definovať tieto validačné pravidlá v príslušnej modelovej triede.

U odoslaných formulárov je pomocou metódy `isValid` možné skontrolovať, či boli splnené všetky požadované kritéria pre jednotlivé formulárové elementy.

Výpis 24 - Definícia validácií pre formuláre v Zend Frameworku

```
public function getInputFilter()
{
    $inputFilter = new InputFilter();
    $inputFilter->add(array(
        'name' => 'first_name',
        'required' => true,
        'validators' => array(
            array(
                'name'=> 'StringLength',
                'options' => array(
                    'min'=> 3,
                ),
            ),
        ),
    ));
}
```

6.6. Pohľady

Poslednou analyzovanou časťou v rámci architektúry MVC sú pohľady. Tie ponúkajú oddelenie aplikačnej vrstvy aplikácie od prezentačnej. Základným spôsobom tvorby dynamického obsahu, je kombinácia HTML kódu pre štruktúru stránky a PHP kódu, ktorý do nej vkladá dáta. Takéto riešenie je použité ako základné v Zend Frameworku. Nadstavbou tohto riešenia sú šablónovacie systémy, ktoré dopĺňajú súbory pohľadov o špeciálne značky. Tie dodávajú šablónam možnosť komplexnejšej práce s dátami a nie len obyčajnému výpisu. Týmto značkami je možné vytvárať riadiace štruktúry, transformovať dáta a podobne. Predvoleným riešením pre Symfony je šablónovací systém Twig.

V kapitole 2.3 boli definované základné návrhové vzory pre pohľady. Obe frameworky implementujú riešenie pohľadov pomocou návrhového vzoru *Two Step View*. Obsahujú globálnu šablónu, tzv. layout. Do layoutu sú vkladané sub-šablóny vygenerované pre jednotlivé akcie v aplikácii. U Zend Frameworku má každý modul nakonfigurovaný predvolený layout, ktorý sa využíva pre všetky radiče a ich pohľady. Základnou šablónou je súbor /view/layout/layout.phtml. Vykreslený obsah jednotlivých akcií je prenesený do layoutu pomocou premennej `$content` ako je vidieť na Výpis 25. Ak má niektorá akcia využívať iný layout pre svoje zobrazenie, je ho potrebné v príslušnom radiči predefinovať.

Výpis 25 - Definovanie globálnej šablóny

```
<div class="container">
      <?php echo $this->content; ?>
</div> <!-- /container -->
```

Šablónovací systém Twig ponúka obdobné, ale rozšírené riešenie tvorby šablón. V globálnej šablóne sa totiž definujú zástupné bloky, do ktorých je vkladaný obsah sub-šablón. Jedna akcia teda môže vygenerovať a dosadiť dáta do rozličných častí stránky. Jednotlivé bloky sú definované značkou block a jeho názvom.

Výpis 26 - Definícia zástupného bloku v Layoute

```
<main role="main" class="container main-content">
    {% block content %}{% endblock %}
</main>
```

Aký layout má byť načítaný pre jednotlivé akcie, si definujú šablóny akcií samé pomocou funkcie `extends`. Napr. `{% extends 'TestCoreBundle::layout.html.twig' %}`.

V šablónach sú dostupné pomocné funkcie, ktoré uľahčujú opakujúce sa činnosti alebo obaľujú logiku výpisu potrebnú na rozličných miestach v aplikácií. Šablónovací systém Twig umožňuje definovať 2 druhy pomocných funkcií a to buď rozšírenia (*Twig Extensions*) alebo makrá. U Zend Frameworku sa vytvárajú pomocné triedy pre pohľady (*View Helpers*). Frameworky obsahujú pár základných a často používaných funkcií, napr. funkciu na vykreslenie statickej šablóny zo súboru alebo funkciu na generovanie URL adries podľa zadefinovaných trás a podobne. Niekoľkými dôležitými funkciami sa zaoberá kapitola 6.7.2, ktorá rozoberá spôsoby ochrany proti útoku *Cross Site Scriptingu*.

Tvorba vlastných funkcií je u oboch frameworkov veľmi podobná. Jedná sa o obyčajnú PHP triedu, rozširujúcu buď triedy *Twig_Extension* alebo *AbstractHelper*. U Zend Frameworku je potrebné novú pomocnú triedu registrovať v konfiguračnom súbore modulu. U Symfony ide o definíciu rozšírenia ako služby. Funkcie, ktoré sú komplexnejšie alebo je potrebné, aby boli dostupné vo všetkých pohľadoch, sú definované pomocou spomínaných rozšírení. Pre jednoduchšie funkcie sa vytvára makro, ktoré musí byť pred použitím v šablóne na importované. Príklad vlastného makra je vidieť na výpise nižšie.

Výpis 27 - Nové pomocné makro a jeho import v rámci šablóny

```
// macro.twig
{% macro getRatingType(value) %}
{% if value == 0 %}frown{% elseif value == 1 %}meh{% else %}smile{% endif %}
{% endmacro %}

// index.twig
{% from 'TestCoreBundle::macros.html.twig' import getRatingType %}
{{ getRatingType(rating.rating) }}
```

Dôležité sú aj pomocné funkcie, ktoré majú na starosť pridávanie kaskádových štýlov a JavaScriptových do šablón. U Zend Framework tieto funkcie umožňuje pridanie súborov dostupných len v koreňovom adresári aplikácie napr. */public/css* a *publis/js*. Ak sa vychádza u komponentového vývoja z toho, že každá komponenta má obsahovať všetky dostupné súbory pre jednu funkcionalitu, tak takéto riešenie vytvára problém pri tvorbe aplikácií. Symfony ponúka na jeho riešenie vlastnú komponentu *Assetic*. Tá umožňuje pri vývoji pomocnými funkciami smerovať na zdroje dostupné v jednotlivých moduloch. Pre produkčnú prevádzku všetky zdroje automaticky prekopíruje do koreňového priečinku tak, aby boli dostupné pre užívateľov. Okrem základného pridávania súborov poskytuje možnosť ich modifikácie pred tým, než je vytvorená produkčná kopia súborov. Nižšie je napríklad nožné vidieť spôsob zlúčenia dostupných súborov do jedného.

Výpis 28 - Konfigurácia JavaScriptových súborov

```
{% javascripts
'@TestCoreBundle/Resources/public/js/definitions.js'
'@TestCoreBundle/Resources/public/js/app.js'
output='application.js'
%}
<script src='{{ asset_url }}'></script>
{% endjavascripts %}
```

Pre Zend Frameworku zatiaľ podobné oficiálne riešenie neexistuje.

6.7. Bezpečnosť

Ako bolo v definícii bezpečnostných kritérií stanovené, hlavným cieľom bezpečnosti je neumožniť užívateľom prístup k zdrojom, na ktoré nemajú právo. Prístup by mal byť umožnený len overeným návštevníkom. Overenie pozostáva z dvoch krokov a to autentizácie a autorizácie. Prvý krok sa zameriava na overenia identity užívateľa, napr. na základe prístupových údajov. Po úspešnom overení nastáva druhý krok procesu a to skúmanie práv tohto užívateľa.

V prvej časti kapitoly bude rozoberané, akým spôsobom sú spomínané dve fázy implementované u frameworkov. Neskôr budú spomenuté spôsoby ochrany proti častým typom útokov u webových aplikácií.

6.7.1. Autentizácia a autorizácia

V Symfony má tieto akcie na starosti modul Security. Ten je nastaviteľný pomocou hlavného konfiguračného súboru aplikácie. Hlavnou konfiguračnou direktívou v súvislosti s autentizáciou je poskytovanie informácie o užívateľoch tzv. *provider*. V prípade ukážkovej aplikácie sa jedná o entitu užívateľ, ktorá je mapovaná na databázovú tabuľku. Spôsob šifrovania užívateľského hesla je definovaný pomocou direktívy typ kódovača.

Autorizácia je riešená pomocou užívateľských rolí v systéme. Roliam môže byť nadefinovaná určitá hierarchia, ktorá umožní dediť definované pravidlá. Každý užívateľ musí mať priradenú minimálne jednu rolu, napr. redaktor či administrátor. Priebeh autorizácie závisí od spôsobu definície prístupových pravidiel. V súčasnosti sú podporované 3 typy definície pravidiel autorizácie.

Prvým spôsobom je definícia vzorov URL adries a k nim prislúchajúce požadované role. Napríklad pre URL adresu, ktorej časť spĺňuje požiadavku pre vzor admin, je potrebná rola administrátora. Iným spôsobom je manuálna kontrola roly užívateľa pri prístupe do akcie v príslušnom radiči, ktorý má byť zabezpečený. Posledným spôsobom je vytvorenie zoznamu pre riadenie prístupov tzv. *Access control list*. Využíva sa hlavne u komplexných aplikácií, jeho princíp je vysvetlený nižšie v časti venujúcej sa Zend Frameworku.

Výpis 29 - Konfigurácia autentizácie a autorizácie v Symfony 2

```
security:
    providers:
        database:
            entity: { class: TestCoreBundle:User, property: email}

    encoders:
        Test\CoreBundle\Model\User:
            algorithm: sha512

    role_hierarchy:
        ROLE_ADMIN: [ROLE_USER, ROLE_COMPANY_USER]

    access_control:
        - { path: ^/users, roles: ROLE_ADMIN }
```

Zend Framework používa pre každú fázu overovania vlastnú oddelenú komponentu. Ide o *Zend Authentication* a *Zend ACL*. Autentizácia aplikácie je taktiež podporovaná a jednoducho implementovaná. Užívateľské údaje sú pomocou vybraného autentizačného adaptéru overené a užívateľovi je vytvorené príslušná session. Vo frameworku existujú pred pripravené autentizačné adaptéry pre základné spôsoby prihlasovania pomocou databáz, štandardnou HTTP autentizáciu alebo pomocou LDAPu.

K autorizácií je pristupované identickým spôsobom ako to bolo u Symfony. Práva sú teda určené na základe rolí užívateľov, kde jednotlivé roly môžu taktiež vytvárať hierarchiu a dediť definované pravidlá. Hlavným rozdielom je, že jediný podporovaný spôsob definície pravidiel je pomocou zoznamu pre riadenie prístupov. Ten v aplikácií definuje dostupné zdroje a dostupné akcie vykonávané s týmito zdrojmi a definuje, ktoré roly majú možnosť dané zdroje používať.

Na výpise nižšie je vytvorená hierarchia rolí v aplikácií a určený jediný zdroj - hodnotenie. Neprihlásený užívateľ má možnosť vytvárať nové hodnotenia. Prihlásený užívateľ môže taktiež vytvárať nové hodnotenia, keďže táto rola dedí pravidlá definované pre neprihlásených užívateľov. Okrem toho sú jeho práva ale rozšírené o ďalšie dostupné akcie. Administrátor má povolené vykonávať všetky akcie nad zdrojom.

Výpis 30 - Definícia zoznamu pre riadenie prístupov

```
use Zend\Permissions\Acl\Acl;
use Zend\Permissions\Acl\Role\GenericRole as Role;

$acl = new Acl();

$acl->addRole(new Role('guest'));
$acl->addRole(new Role('user'), 'guest');
$acl->addRole(new Role('administrator'));

$acl->addResource(new Resource('ratings'));

$acl->allow('guest', 'ratings', 'create');
$acl->allow(user', 'ratings', 'index);
$acl->allow('administrator');
```

6.7.2. Ochrana proti najčastejším útokom

Lapáček (2011, s. 26) vo svojej práci poukazuje na najčastejšie spôsoby útokov na webové aplikácie. Jedná sa o tieto typy útokov:

- Cross Site Scripting,
- SQL injection,

- Crose-site Request Forgery,
- Session hijakcking a Session Fixation.

Nasledujú krátke vysvetlenia ako jednotlivé útoky prebiehajú a aké nedostatky a zraniteľnosti v zdrojovom kóde webových aplikácií útočníci využívajú. V rámci jednotlivých vysvetlení bude načrtnuté aj jedno z dostupných riešení a analýza riešenia týchto problémov u oboch frameworkov.

Cross Site Scripting

Tento typ útoku spočíva v tom, že neošetrené a nedôveryhodné dáta sú pri vykresľovaní vo webovej aplikácií interpretované ako HTML kód. Útočník má takto možnosť vložiť do stránky kód JavaScriptu, ktorý môže napríklad odoslať dáta o sedení na vlastný server a tým pádom získať prístup do aplikácie. Riešenie tohto problému bolo už v tejto práci rozoberané v kapitole 3.3.7.

Ošetrovaním vstupov sa používa v štyroch rozdielnych situáciách: ošetrenie HTML kódu, JavaScriptové kódu, kódu kaskádových štýlov a HTML atribútov.

Zend Framework má definované pomocné funkcie pre jednotlivé typy ošetrení. Jedná sa o funkcie `escapeHtml`, `escapeJs`, `escapeCss` a `escapeHtmlAttr`. Pri písaní kódu teda musí programátor manuálne zvoliť, o ktorý typ ide. Šablónovací systém Twig u Symfony ponúka filter *escape*, ktorý taktiež umožňuje ošetrovať všetky typy výstupov. Pri definícií pohľadu musí taktiež programátor zvoliť, o aký kontext má záujem, napr. `{{ user.username|escape('js') }}`.

Obe frameworky teda ponúkajú spôsob ako jednotlivé typy ošetrovať.

SQL injection

Jedná sa o techniku, kedy útočník vytvára alebo upravuje existujúce SQL príkazy za účelom získať skryté dáta, manipulovať a prepísať cenné dáta alebo dokonca vykonať nebezpečné akcie na úrovni databázového servera. Je to dosiahnuté využitím vstupov z aplikácie v kombinácií s parametrami staticky vybudovaných SQL dotazov v aplikácií (THE PHP GROUP, 2013).

Obrana proti tomuto útoku je pomerne jednoduchá. Stačí nevytvárať v aplikácií SQL požiadavky pomocou skladania reťazcov, ale využiť parametre požiadaviek.

Pri tvorbe dotazu v Symfony je použitý tzv. query builder, ktorý umožňuje použitie parametrov. Pri vykonaní požiadavky je vstup užívateľa ošetrený a nahradený za zástupné symboly. Tie sú v pôvodnom zápise označené pomocou dvojbodky, napr. *:place_id*.

Výpis 31 - Tvorba parametrizovaného dotazu v Symfony 2

```
$queryBuilder = $this->createQueryBuilder('r')
    ->select('r')
    ->where('r.place_id = :place_id')
    ->setParameter('place_id', $placeId);
```

Obdobné riešenie je použité aj v pri tvorbe požiadaviek Zend Frameworku. Užívateľské vstupy sú ošetrené a nahradené za otáznik.

Výpis 32 - Tvorba parametrizovaného dotazu v Zend Frameworku

```
$placesTable = new Places();
$placesTable->select()
    ->where('place_id = ?', $placeId);
```

Cross-Site Request Forgery

Ide o útok, kedy sa útočník maskuje za iného užívateľa a ktorý vykoná neplánovanú akciu bez toho, aby o tom vôbec vedel. Pri tomto type útoku musí útočník poznať URL adresy slúžiace na vykonávanie akcií v aplikácií. Príkladom môže byť zmazanie užívateľa dostupné na adrese *users/delete/1*. Autorizovanému užívateľovi je podstrčený zdroj (napr. obrázok), ktorý vykoná HTTP požiadavku na aplikáciu.

Prvým bezpečnostným krokom je to, že všetky akcie v aplikácií, ktoré vedú k úprave či mazaniu citlivých dát, by mali odosielať dáta pomocou http metódy *POST*. Týmto sa zaručí, že sa jednoduchá http požiadavka nevykoná. Druhou úrovňou riešenia tohto problému je pridanie autorizačného tokenu do aplikácie. Volaná akcia sa vykoná, len ak je autorizačný token dostupný a jeho autenticita potvrdená.

Symfony aj Zend Framework implementovali možnosť overenie pomocou autorizačného tokenu. U všetkých formulárov v rámci aplikácie vytvorenej pomocou Symfony je táto ochrana automatická. U Zend frameworku je potrebné ju u každého formulára manuálne pridať ako je vidieť **Chyba! Nenalezen zdroj odkazů.**.

Výpis 33 - Vytvorenie autorizačného tokenu pre formulár

```
$form = new Form('form');
$form->add(array(
    'type' => 'Zend\Form\Element\Csrf',
    'name' => 'csrf'
)));
```

Session hijakcking a Session Fixation

U webových aplikácií je dôležité si udržiavať informácie o stave aplikácie a užívateľovej práce s ňou. Keďže je http bez stavový protokol, sú požiadavky jedného užívateľa na webový server vždy novým spojením. Riešením je každému užívateľovi vygenerovať identifkátor sedenia, t.j. náhodne vygenerovaný token, podľa ktorého si server páruje dokopy požiadavky konkrétneho návštevníka (Tichý, 2008a).

Obe útoky súvisia s určitou manipuláciou s týmto identifikátorom. Jeho odcudzením získava útočník možnosť pristúpiť do aplikácie bez nutnosti zadania prístupových údajov.

Podľa Tichého (2008b) existujú 3 možnosti predávania identifikátoru v aplikácií a to pomocou

- Parametrov v URL,
- skrytých formulárových polí
- cookies.

Najlepším spôsobom ochrany proti ukradnutiu týchto dát (*Session hijakcking*) je využitie zápisu údajov o sedení výhradne do cookie prehliadača. Keďže pri použití nešifrovaného spojenia a prenášaní identifikátora pomocou URL adries je možné pomocou odchytávania paketov možné tento údaj získať. Jedinou ochranu, ktorú môžu frameworky ponúknuť v tomto prípade, je ochrana proti Cross Site Scriptingu, ktorá znemožní útočníkovi pomocou JavaScriptu získať a odoslať identifikátor. Správnym nastavením sessions direktív[17] v PHP je možné úroveň zabezpečenia webovej aplikácie výrazne vylepšiť.

Druhým útokom je podvrhnutie identifikátora útočníkom, kedy je užívateľov súčasný kľúč nahradený vlastným. K danému užívateľovi sa po prihlásení automaticky uloží tento útočníkom používaný identifikátor. Týmto spôsobom získava útočník prístup do aplikácie. Riešením je generovať novú hodnotu pre daný identifikátor v určitých intervaloch alebo pri určitých akciách.

Zend Framework interne využíva natívne funkciu na správu sedenia a teda pomocou funkcie `session_regenerate_id` generuje nový identifikátor session. Programátor má možnosť využiť funkciu `regenerateId` v triede `SessionManager`, ktorý túto prácu zaobaľuje.

Symfony taktiež využíva komponentu, ktorá obaľuje Sessions. Je potrebné sa vyhnúť manuálnym volaniam natívnych funkcií PHP, pretože niektoré funkcie sú reimplementované a nahradzujú tie natívne. Ochranou je zavolanie funkcie `regenerate.`

[17] Direktívy use_only_cookies a cookie_httponly.

6.8. Internacionalizácia a lokalizácia

Hoci vo funkčných požiadavkách pre ukážkovú aplikáciu nie je definovaná viacjazyčnosť, nasledujúca kapitola sa venuje podpore internacionalizácie a lokalizácie v oboch skúmaných aplikačných frameworkoch.

Ak majú aplikácie správne komunikovať v jazyku užívateľa, je potrebné nastaviť tzv. *locale*. Ten definuje aké lokalizačné súbory majú byť používané. Zápis je vo formáte *názov jazyka* podľa štandardu ISO639-1, podčiarkovacie znamienko nasledované názvom krajiny podľa štandardu ISO3166 napríklad *sk_SK*. Nastavenie *locale* sa v Symfony definuje globálne pre celú aplikáciu v, zatiaľ čo u Zend Frameworku je to na úrovni jednotlivých modulov.

Všetok obsah, ktorý musí byť prispôsobený užívateľom, je potrebné určitým spôsobom označiť. Tým sa pri generovaní nahradí obsahom v správnom tvare podľa aktuálneho *locale*. Oba frameworky ponúkajú viacero rovnakých možností ako takýto obsah označovať ako je vidieť na výpisoch nižšie.

Výpis 34 - Preklad statických textov v Zend Frameworku

```
// Radič
$translator = $this->getServiceManager()->get('translator');
$translator->translate(''Vitajte!');
// Pohľad
echo $this->translate("'Vitajte");
```

Výpis 35 - Preklad statických textov pomocou kľúčom definovaných textov v Symfony

```
// Radič
$translator = $this->get('translator');
$translator->trans('home.welcome');
// Pohľad
{% trans "home.welcome" %}
```

Takto označené texty je potrebné manuálne vyextraktovať pomocou nástroju xgettext. Vygenerované texty alebo kľúče sú ukladané do katalógov pre príslušný jazyk. Zend Framework podporuje v súčasnosti katalógy definované pomocou PHP polí a súborov typu Gettext, Tmx a Xliff. Symfony navyše podporuje Ini a CSV súbory.

Okrem základných prekladov statických textov je dôležité, aby boli lokalizované aj časy, dátumy a meny. Pohľady obsahujú pomocné funkcie, ktoré na základe aktuálnej konfigurácie vypíšu tieto dáta v správnom formáte. V Zend Frameworku sú to funkcie `currencyFormat()`, `dateFormat()` a `numberFormat()`. Šablónovací systém Twig ponúka taktiež podporu pre rovnaké funkcie

s výnimkou výpisu meny. Tieto funkcie ale nie sú dostupné bez dodatočného zaregistrovania rozšírenie tohto systému.

U dynamických textov, teda dát načítavaných z databáz, je lokalizácia taktiež možná. Umožňuje to kvalitné rozšírenie *Translatable*[18] pre Doctrine. To automaticky podľa konfigurácie aplikácie vypisuje dáta entít zo stĺpcov, ktoré obsahujú definovaný sufix, napr. u entity hodnotenia by to bol stĺpec `note_sk` a `note_en`.

Prepínanie jazyka a zvyklosti môže byť v aplikáciách implementované rozličným spôsobom. Napríklad auto detekciou podľa požadovanej URL adresy, ktorá využíva určitý prefix alebo definovaním jazyka u jednotlivých užívateľov a podobne.

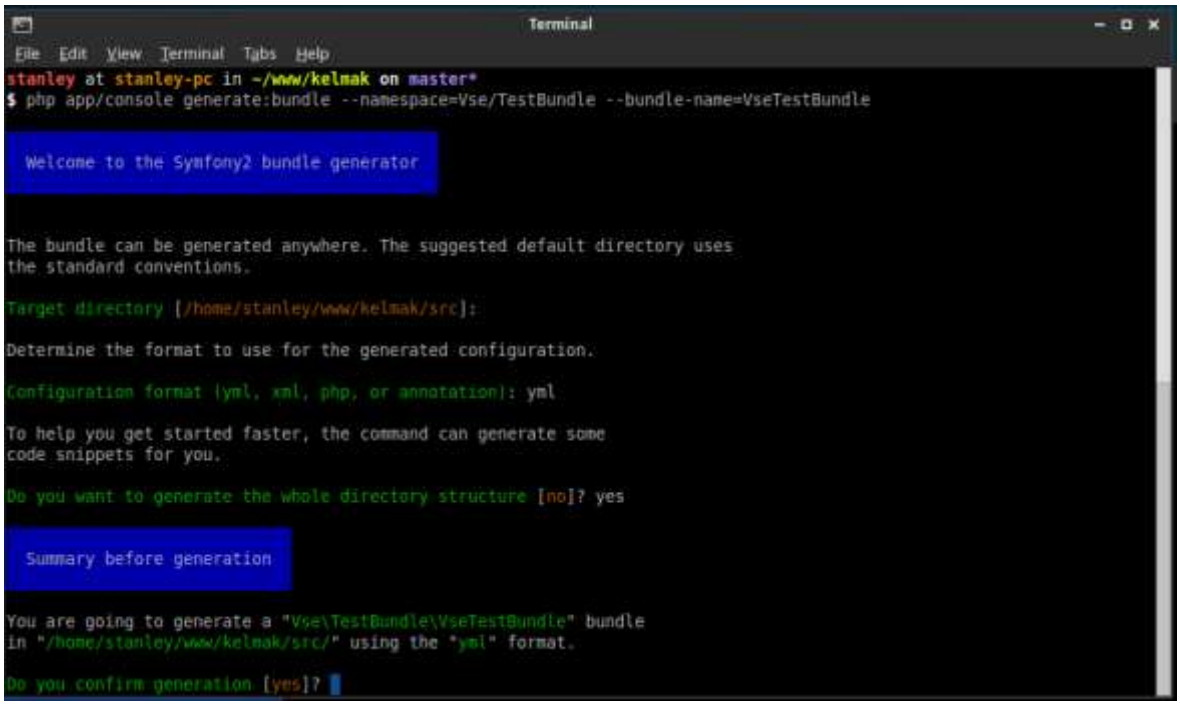
6.9. Nástroje príkazového riadku

Automatizácia často opakujúcich sa činností dokáže znížiť chybovosť a urýchliť vývoj. Prevažne pre činnosti, ktoré pozostávajú z viacerých krokov. Nástroje príkazového riadku sú navrhnuté práve pre tento účel automatizácie. Môžu byť implementované napríklad ako shell či php skripty. Obe frameworky takýto nástroj implementovaný majú. U Symfony sa jedná o konzolu a u Zend Frameworku o ZFTool.

Konzola

Jedná sa o komponentu, ktorá je zaradená do štandardnej distribúcie Symfony. Konzola bola navrhnutá veľmi flexibilne a každý vývojár ma možnosť ju rozšíriť o vlastné príkazy, ktoré urýchlia prácu s jeho modulom. Samotný nástroj je pomerne malá Symfony aplikácia, ktorá ale pristupuje ku všetkým modulom daného projektu. Automaticky načítava a využíva nadefinované príkazy, ktoré spĺňajú určité konvencie.

[18] <https://github.com/l3pp4rd/DoctrineExtensions/blob/master/doc/translatable.md>



Obrázok 7 - Generovanie nového bundlu pomocou konzoly

Dostupné príkazy sa môžu líšiť podľa toho, aké moduly sú v aplikácií zaregistrované. Medzi najpoužívanejšie príkazy patrí napríklad generátor radičov, modulov, entít, či príkazy na vyčistenie súborov vyrovnávacej pamäte (cache súborov).

Jednotlivé príkazy môžu vyžadovať ďalší vstup užívateľa, ktorým je požiadavky viac špecifikovaná. Napríklad príkaz na tvorbu nového modulu vyžaduje od užívateľa ešte spresnenie názvu, umiestnenia a používaného typu súborov pre konfiguráciu modulu.

Medzi základné príkazy, ktoré integruje do konzole modul *Doctrine* je generovanie entít, ich formulárov alebo vytvorenie základných akcií nad týmito entitami. Jedná sa o tzv. metódu Scaffoldingu. Ondřej Válek (2008) ju definuje ako metódu na „*generování zdrojového kódu uživatelského rozhraní aplikace, které pak umožní provádět základní operace nad databázovými tabulkam (INSERT, SELECT, UPDATE, DELETE). Na základě databázového schématu je vygenerována funkční kostra aplikace, jakési jednoduché administrační rozhraní, které zjednodušuje vývoj zbývající aplikace.*“.

ZFTool

Je taktiež nástroj tvorený aplikáciou samotného frameworku, ale je vyvíjaný samostatne. Ponúka len obmedzené možnosti pre vývojárov. Jedná sa o príkazy na tvorbu modulov, radičov alebo vytvorenia nového Zend Framework projektu.

V súčasnosti neexistuje žiaden spôsob ako rozšíriť funkcionalitu tohto nástroja bez nutnosti zásahu do jeho zdrojového kódu. Vývojári týmto krokom ale strácajú možnosť jednoduchej aktualizácie tohto nástroja na novšie verzie.

6.10. Testovanie

Ak má byť vyvíjaný software kvalitný, je dôležité aby bol otestovaný a neobsahoval chyby. Na základe špecifikácie funkcionality je vytváraná nie len samotná aplikácia ale aj testy, ktoré overujú že všetko funguje ako má. Ako definuje Šplíchalová (2008, str.6) „testovanie overuje že predmet testu spĺňuje požiadavky na neho kladené". Pri zmenách je tým zaručené, že súčasné funkcie neboli zmenené a aplikácia je funkčná.

V kapitole 3.4 boli definované, že existujú testy jednotiek, integračné testy a funkčné testy. Na testovanie PHP aplikácií je de facto štandardom knižnica PHPUnit, ktorá ponúka možnosti testovania všetkých spomínaných typov. Tak ako iné komponenty, tak aj PHPUnit je možné nahradiť inou testovacou knižnicou napr. *Simple Test*[19] alebo *PHPSpec*[20].

Oba frameworky využívajú základné triedy z knižnice PHPUnit na jednotkové aj integračné testovanie. Špecifickým u oboch frameworkom je ale funkčné testovanie. To overuje, že postup určitej akcií v aplikácií funguje ako má. Tieto testy môžu vytvárať požiadavky na aplikáciu, príjmať odpovede, poprípade vykonať určitú akciu napríklad vyplnenie formulára a jeho následné odoslanie. Tieto testy overujú správnosť radičov a ich nadefinovaného správania. V Symfony sa pre tieto testy využíva trieda `WebTestCase` a v Zend Frameworku 2 je to trieda `AbstractHttpControllerTestCase`. Výpise nižšie ukazujú základný princíp fungovania fungkčných testov v oboch frameworkoch. Výpis 37 ukazuje spôsob funkčného testovania v Symfony 2.

Výpis 36 - Funkčný test s Zend Frameworku 2

```
public function testShouldAccessUserRatings()
{
    $this->dispatch('/ratings);
    $this->assertResponseStatusCode(200);

    $this->assertModuleName('Application`);
    $this->assertControllerName('Application\Controller\Rating);
    $this->assertControllerClass(RatingController);
    $this->assertMatchedRouteName('ratings');
}
```

[19] <http://www.simpletest.org/>
[20] <http://www.phpspec.net/>

Výpis 37 - Funkčný test prihlásenia do aplikácie v Symfony

```
public function testShouldLogInUser()
{
    $client = static::createClient();

    $crawler = $client->request('GET', '/login');
    $this->assertEquals(200, $client->getResponse()->getStatusCode());

    $form = $crawler->selectButton('Přihlásit')->form(array(
        '_username'  => 'admin',
        '_password'  => 'nbusr123',
    ));

    $client->submit($form);
    $client->followRedirect();
    $this->assertEquals(200, $client->getResponse()->getStatusCode());
}
```

6.11. Hodnotenie autora

Vývoj ukážkovej aplikácie pomocou Zend Frameworku 2 a Symfony 2 bol mojím prvým použitím týchto frameworkov v praxi, hoci som v minulosti sledoval, akým smerom sa všeobecne tieto frameworky uberajú. Rozsiahla dokumentácia a články na blogoch boli v oboch prípadoch hlavnými zdrojmi informácií a pri prvom stretnutí s frameworkmi boli veľmi nápomocné.

V jednotlivých častiach tejto kapitoly boli ukázané základné spôsoby tvorby aplikácie pomocou týchto frameworkov. Ako je vidieť, oba hodnotené frameworky sa snažia pokryť čím väčšiu časť potrebnej funkcionality na tvorbu webových aplikácií. Ponúkajú funkcie a spôsoby na riešenie častých úkonov a problémov pri vývoji. Pri tom všetkom ale dbajú na to, aby neobmedzovali a neznemožňovali vývojárom implementáciu vlastných riešení.

Ako vývojára používajúceho pri práci prevažne Zend Framework 1, ma zaujali hlavne zmeny týkajúce sa architektúry nových frameworkov a noviniek týkajúcich sa zmien v PHP. Najväčšími zmenami oproti starším frameworkm je využívanie menných priestorov, ktoré boli pridané v PHP 5.3. Poskytujú taktiež spôsob na definovanie a predávanie závislosti medzi objektmi a to pomocou servisnej triedy. Frameworky využívajú *Event managera* alebo *Event dispectecher* a sú teda vedené udalosťami. To znamená, že v príprade nejakej udalosti v aplikácií je možné vykonať určitú nadväzujúcu činnosť.

Vytvorenie ukážkových aplikácií bolo pre mňa veľmi prínosné. Najmä, čo sa týka prehĺbenia znalostí nových frameworkov, návrhových vzorov a zmien, ktoré priniesli nové verzie jazyka PHP.

7. Vyhodnotenie kritérií

Nasledujúca kapitola obsahuje porovnanie oboch frameworkov podľa kritérií definovaných v kapitole 3. Ako bolo načrtnuté už v úvode tejto kapitoly, jednotlivým kategóriám bude pridelená váha pomocou metódy párového porovnávania. Keďže kategória vývoj obsahuje veľké množstvo kritérií, bola rozdelená do ôsmich pod kategórií.

7.1. Architektúra frameworku a zavedené konvencie

Návrhové vzory

U oboch frameworkov je kladený vysoký doraz na kvalitu návrhu. Oba frameworky pri návrhu a implementácií využívajú známe návrhové vzory. Spracovanie užívateľských požiadaviek na aplikáciu je riadené pomocou *Front Controller*. Následne sa využíva *Model View Controller*. V kapitole 6.3 a 6.6 boli spomenuté niektoré ďalšie používané návrhové vzory. Okrem spomínaných architektonických vzorov sú využívané aj všeobecne známe návrhové vzory ako:

- Abstract Factory,
- Builder,
- Dependency Injection
- Facade,
- Mediator,
- Chain Of Responsibility,
- a iné.

Pravidlá pre písané kódu

Projekty majú definované odporúčané pravidlá pre písanie zdrojového kódu. Symfony dodržiava štandardy[21] definované v dokumentoch PSR-0, PSR-1 and PSR-2. Zend framework má definované vlastné a pomerne rozsiahle pravidlá[22].

Tabuľka 8 - Vyhodnotenie kritérií v kategórií architektúra frameworku

	Váha	Symfony 2	Zend Framework 2
Návrhové vzory	0,7	Áno	Áno
Pravidlá pre písané kódu	0,3	Áno	Áno
	1	**1**	**1**

[21] <http://symfony.com/doc/current/contributing/code/standards.html>
[22] <http://framework.zend.com/wiki/display/ZFDEV2/Coding+Standards>

7.2. Podporované technológie

Podporované verzie PHP

Obe frameworky majú minimálne požiadavky na verziu PHP 5.3.3. Bez žiadnych komplikácií je teda možné vytvárať a prevádzkovať webové aplikácie pri súčasnom stave hostingových spoločností v Českej republike.

Podpora HHVM

Podpora virtuálneho stroja je najdôležitejším kritériom, pretože prináša revolúciu vo svete PHP. Jedná sa o prvý neoficiálny interpretátor jazyka PHP, ktorý svojím výkonom posunie hranice webových aplikácií na platforme PHP ďaleko dopredu. Vývojový tím stojací za virtuálnym strojom pre PHP pravidelne publikuje súčasný stav podpory webových aplikačných frameworkov a systémov na správu obsahu (*CMS*). Kritérium hodnotí na základe oficiálnych výsledkov jednotkových testov od vývojového tímu (MARCEY, 2013). Výsledky sú 87% fungujúcich testov frameworku Symfony a 92% frameworku Zend Framework.

Správa závislosti

V kapitole 6.1 bolo poukázané, že obe projekty využívajú nástroj Composer na riešenie závislosti.

Tabuľka 9 - Vyhodnotenie kritérií v kategórií podporované technológie

	Váha	Symfony 2	Zend Framework 2
Podporované verzie PHP	0,1	Áno	Áno
Podpora HHVM	0,6	0,87	0,92
Správa závislosti	0,3	Áno	Áno
	1	**0,92**	**0,95**

7.3. Databáze

Kapitola hodnotí kritéria, ktoré sa týkajú spojenia aplikácií s rozličnými databázovými servermi a pokročilými technikami synchronizácie databázových schém. Je dôležité podotknúť, že kritéria sú hodnotené pri využití knižníc v štandardných distribúciách frameworkov. Teda využitia knižnice Doctrine pre Symfony a komponenty Zend Database pre Zend Framework 2. Výsledné hodnotenie kritérií obsahuje Tabuľka 10.

Podpora relačných databázových serverov

Tabuľka v úvode kapitoly venujúcej sa databázam a dátovým modelom ukážkovej aplikácie, obsahuje zoznam oficiálne podporovaných databázových systémov pre jednotlivé frameworky. Symfony vďaka tomu, že využíva Doctrine, poskytuje podporu pre všetky základné databázové servery. Zend Framework v súčasnosti zatiaľ oficiálne nepodporuje IBM DB2 a Oracle, hoci v oficiálnom repozitáry existujú vytvorené databázové adaptéry pre tieto servery.

Podpora NoSQL databázových serverov

Doctrine v súčasnosti podporuje dva NoSQL databázové servery. Ide o dokumentové databázy *CouchDB* a *MongoDB*. Ich integrácia do Symfony prebieha pomocou inštalácie oficiálnych komponent DoctrineCouchDBBundle[23] a DoctrineMongoODMModule[24]. Tieto moduly spravuje vývojový tím samotnej Doctrine. Sú teda pravidelne aktualizované tak aby podporovali všetku požadovanú funkcionalitu. Zend Framework v súčasnosti oficiálne nepodporuje pripojenie s NoSQL databázami. Pre obe spomínané databázové servery už existujú návrhy na vytvorenie samostatných komponent, ale zatiaľ neboli v žiadnej verzii implementované.

Spojenie s viacerými databázami

Ako bolo v práci spomínané, Doctrine využíva na načítanie, perzistenciu a mazanie dát tzv. entity manager. V aplikáciách vytvorených pomocou Symfony je možné využívať viacero inštancií tohto managera pre komunikáciu s rozličnými databázovými servermi.

Podobná možnosť do Zend Frameworku pribudla od verzie 2.2 (O'PHINNEY, 2013), ktorá umožnila definovanie viacerých aktívnych databázových adaptérov.

Migrácia databázových schém

Doctrine ponúka možnosť migrácie databázových schém. Pre Symfony aplikácie je potrebné využiť vytvorenú komponentu DoctrineMigrationsBundle[25], ktorú taktiež ako predchádzajúce, spravuje rovnaký vývojový tím.

V súčasnosti žiadna z komponent Zend Frameworku neprináša funkcie pre databázové migrácie. Pre túto činnosť je potrebné využiť knižnice alebo nástroje tretích strán.

Tabuľka 10 - Vyhodnotenie kritérií v kategórií databáze

	Váha	Symfony 2	Zend Framework 2
Podpora relačných databázových serverov	0,5	1	0,66
Podpora NoSQL databázových serverov	0,2	Áno	Nie
Spojenie s viacerými databázami	0,1	Áno	Áno
Migrácia databázových schém	0,2	Áno	Nie
	1	**1**	**0,43**

[23] <https://github.com/doctrine/DoctrineCouchDBBundle>
[24] <https://github.com/doctrine/DoctrineMongoODMModule>
[25] <https://github.com/doctrine/DoctrineMigrationsBundle>

7.4. Pohľady

Všetky kritéria s výnimkou ochrany proti Cross Site Scriptingu, sú analyzované v kapitole 6.6. Objasnenie implementácie ochrany proti útokom je objasnené v časti Ochrana proti najčastejším útokom. Výsledné hodnotenie tejto kapitoly sa nachádza v Tabuľka 11.

Rozšírenou variantnou dedičnosti šablón je definícia zástupných blokov v šablónach Twigu. Základné riešenie u Zend Frameworku túto možnosť nemá. Z tohto dôvodu získava menšie ohodnotenie.

Najväčším nedostatkom riešenia pohľadov u Zend Frameworku z hľadiska vývojárov je absencia manažmentu zdrojov. V základnom riešenie nie je navrhnuté žiadne riešenie ako majú byť súbory kaskádových štýlov a JavaScriptu načítavané z jednotlivých modulov ani spôsob, ktorým je možné tieto súbory pripraviť pre produkčné použitie.

Tabuľka 11 - Vyhodnotenie kritérií v kategórií pohľady

	Váha	Symfony 2	Zend Framework 2
Dedičnosť šablón	0,3	Áno	0,8
Ochrana proti Cross Site Scriptingu	0,4	Áno	Áno
Manažment zdrojov	0,1	Áno	Nie
Pomocné funkcie	0,2	Áno	Áno
	1	**1**	**0,84**

7.5. Radiče

Keďže všetky potrebné informácie súvisiace s radičmi boli objasnené v kapitole 6.4., táto kapitola už ponúka len výslednú tabuľku hodnotenia kritérií.

Tabuľka 12 - Vyhodnotenie kritérií v kategórií radiče

	Váha	Symfony 2	Zend Framework 2
Pomocné funkcie	0,3	Áno	Áno
Podpora bleskových správ	0,1	Áno	Áno
Podpora mapovanie URL adries	0,6	Áno	Áno
	1	**1**	**1**

7.6. Komponentový návrh a znova použiteľnosť

V kapitole 6.2 bol na príklade demonštrovaný spôsob tvorby vlastných komponent. V tejto kapitole budú k jednotlivým kritériám doplnené krátke objasnenia k ohodnoteniu oboch frameworkov. Výsledné vyhodnotenie sa nachádza v Tabuľka 13.

Komplikovanosť definície komponent

Hoci moduly aj bundles slúžia na rovnaký účel, veľkým rozdielom je ich samotná konfigurácia a tvorba. Konfigurácia modulov sa nachádza iba v jednom súbore a samotný zápis prebieha pomocou neprehľadných asociatívnych polí. Taktiež je u každého modulu potrebná opakovaná konfigurácia všetkých služieb, aj keď ide o základné funkcie ako pohľady či preklady. Symfony na druhú stranu poskytuje širokú škálu spôsobov konfigurácie a vďaka dodržaným konvenciám je možné konfiguráciu zjednodušiť.

Znova použiteľnosť a integrácia

Oba frameworky poskytujú možnosť rýchlej integrácie komponent tretích strán s možnosťou konfigurácie pomocou definícií služieb. Úplná samostatnosť konfigurácie u modulov Zend Frameworku ale vedie k tomu, že komponenty nie sú závislé na konfigurácií aplikácie a to umožňuje ich lepšiu znova použiteľnosť.

Počet komponent tretích strán

Symfony s väčším počtom komponent získava ohodnotenie 1. Zend Framework získa ohodnotenie úmerne k jeho evidovanému počtu komponent.

Tabuľka 13 - Vyhodnotenie kritérií v kategórií komponentový návrh

	Váha	Symfony 2	Zend Framework 2
Komplikovanosť definície komponent	0,4	1	0,4
Znova použiteľnosť a integrácia	0,3	0,8	1
Počet komponent tretích strán	0,3	1	0,18
	1	**0,94**	**0,51**

7.7. Formuláre

Tvorbe formulárov a ich definícií bola venovaná kapitola 6.5. Téme *Cross –Site Request Forgery* je venovaná časť kapitoly 6.7.2.

V Tabuľka 14 sú zaznamená výsledky jednotlivých kritérií. Okrem nižšie preberaného kritéria venujúceho sa spôsobu definície formulárov, oba frameworky splňujú všetky kritéria. Ponúkajú teda základnú potrebnú funkcionalitu na prácu s formulármi.

Spôsob definície formulárov a ich polí

Vďaka tomu, že Zend Framework umožňuje viacero spôsobov definície formulárov a nenúti vývojárov využívať len zápisy pomocou asociatívnych polí, je jeho ohodnotenie v rámci prvého kritéria vyššie ako u Symfony. Možnosť definovať validačné pravidlá v modeloch tried reprezentujúcich entity, je príklad separácie správneho doménovej logiky od pohľadu ale pre

všeobecné formuláre, ktoré nepracujú so žiadnymi entitami, môže byť tento zápis zbytočne komplikovaný.

Tabuľka 14 - Vyhodnotenie kritérií v kategórií formulárov

	Váha	Symfony 2	Zend Framework 2
Spôsob definície formulárov a ich polí	0,4	0,6	0,9
Podpora základných formulárových polí	0,1	Áno	Áno
Podpora nových typov formulárových polí	0,05	Áno	Áno
Rozšírenie vlastností formulárových polí	0,05	Áno	Áno
Definícia vlastných formulárových polí	0,1	Áno	Áno
Validácia	0,15	Áno	Áno
Ochrana proti Cross-Site Request Forgery	0,15	Áno	Áno
	1	**0,84**	**0,96**

7.8. Bezpečnosť

Oba frameworky spĺňajú všetky kritéria v tejto kategórií. Ich vysvetlenie sa nachádza v kapitole 6.7 - Bezpečnosť.

Tabuľka 15 - Vyhodnotenie kritérií v kategórií bezpečnosť

	Váha	Symfony 2	Zend Framework 2
Podpora autentizácie	0,3	Áno	Áno
Podpora autorizáciu	0,3	Áno	Áno
Metódy na ochranu aplikácie	0,4	Áno	Áno
	1	**1**	**1**

7.9. Internacionalizácia a lokalizácia

Ukážka integrácie funkcionality internacionalizácie a lokalizácie do aplikácie je preberaná v kapitole 6.8.

Keďže prvé dve analyzované kritéria tvoria nevyhnutné minimum pre tvorbu viac jazyčných aplikácií, je ich ohodnotenie najvyššie. Váha kritéria, ktoré hodnotí lokalizáciu vstupov je najnižšia, pretože sa jedná o nadštandardnú funkcionalitu.

Tabuľka 16 - Vyhodnotenie kritérií v kategórií internacionalizácia a lokalizácia

	Váha	Symfony 2	Zend Framework 2
Podpora internacionalizácie	0,4	Áno	Áno
Lokalizácia statických textov	0,3	Áno	Áno
Lokalizácia užívateľských textov	0,05	Čiastočná	Nie
Lokalizácia meny	0,1	Nie	Áno
Lokalizácie dátumov a časov	0,15	Áno	Áno
	1	**0,875**	**0,95**

7.10. Nástroje príkazového riadku

Účel a možnosti nástrojov pre jednotlivé frameworky boli podrobnejšie analyzované v kapitole 6.8. Nástroj konzola pre Symfony ponúka veľké možnosti pre vývojárov a to hlavne jednoduchú rozšíriteľnosť a množstvo už dostupných príkazov. Tie dokážu vývoj urýchliť a uľahčiť. Nástroj ZFTool na druhej strane ponúka vývojárom pomoc len s podporou pre základné generovanie a ničím iným.

Tabuľka 17 - Vyhodnotenie kritérií v kategórií nástroje príkazového riadku

	Váha	Symfony 2	Zend Framework 2
Dostupnosť nástroja príkazového riadku	0,3	Áno	Áno
Príkazy na prácu s databázou	0,1	Áno	Nie
Generovanie časti aplikácií	0,1	Áno	Áno
Vytváranie vlastných príkazov	0,5	Áno	Nie
	1	**1**	**0,4**

7.11. Testovanie

Tabuľka 18 obsahuje výsledné hodnotenia kategórie testovanie. Jednotlivé kritéria boli analyzované v kapitole 6.10. Dôležité je dodať, že oba frameworky výrazne propagujú testovanie aplikácií a sami dbajú na to, aby bol pridávaný zdrojový kód vždy otestovaný.

Tabuľka 18 - Vyhodnotenie kritérií v kategórií testovanie

	Váha	Symfony 2	Zend Framework 2
Konfigurácia testovania	0,4	1	1
Podpora jednotkových testov	0,2	Áno	Áno
Podpora integračných testov	0,2	Áno	Áno
Podpora funkčných testov	0,2	Áno	Áno
	1	**1**	**1**

7.12. Podpora a iné

V nasledujúcej kapitole budú zhodnotené kritéria pre základnú podporu frameworku pre vývojárov. Každé kritérium je doplnené o vysvetlenie pre jednotlivé frameworky, pretože v tejto práci nebolo k daným témam dosiaľ nič napísané.

Kvalita dokumentácia

V dokumentácií oboch framework je návod pre jednoduchú aplikáciu, na ktorej sú vysvetlené základné princípy fungovania frameworku. Napriek tomu, že je tento návod obsiahnutý v oboch prípadoch, Symfony ponúka pomerne rozsiahlejšiu variantu. Rozšírením tohto návodu je tzv. kuchárka (*Cookbook*), ktorá rozoberá pokročilejšie techniky a možnosti jednotlivých komponent a celého frameworku.

Zend Framework rozoberá v referenčnej príručke všetky svoje dostupné komponenty a ich vlastnosti. Tento spôsob nie je pre začiatočníkov najlepší, pretože jednotlivé komponenty sú popisované mimo kontext celého frameworku a nemusí byť jasné, ako sú medzi sebou zaintegrované.

Aktivita v diskusných fórach

U kritérií zaoberajúcich sa počtom diskusií a otázok nebolo možné jednoznačne určiť výsledné počty. A to z dôvodu, že skúmané frameworky majú pomerne úspešných predchodcov. Kľúčovým slovom na vyhľadávanie skupín, diskusií a otázok bol názov frameworku vždy doplnený o aktuálnu verziu. Je možné, že týmto spôsobom boli vyfiltrované aj relevantné diskusie.

Počty nájdených diskusií na portály groups.google.com (GOOGLE, Inc., 2013a):

- Symfony 2 – 6,273 (26 skupín),
- Zend Framework 2 – 123 (9 skupín).

Počet nájdených otázok na portály Stack Overflow (STACK EXCHANGE, Inc., 2013):

- Symfony 2 – 14,134,
- Zend Framework 2 – 2,981.

Výpočet ohodnotenie týchto kritérií bude pomerové, teda framework s vyšším počtom získa hodnotenie 1. Ohodnotenie druhého frameworku je vypočítané úmerne k prvému počtu.

Dostupnosť knižiek

Okrem toho, že oficiálna dokumentácia oboch frameworkov je na vysokej úrovni, existuje už aj niekoľko kníh, ktoré sa venujú striktne tejto tematike. V súčasnosti existuje 5 kníh venujúcich sa základom Zend Frameworku. Jedinou dostupnou knihou pre Symfony je *A Year With Symfony*, ktorá sa ale zaoberá už pokročilými technikami pri vývoji s týmto frameworkom.

Podpora vo vývojových prostrediach

Ani jedno testované vývojové prostredie nemá vstavanú podporu pre analyzované frameworky. Jedinou možnosťou je využitie rozšírení tretích strán. Plným hodnotením je ohodnotená pokročilá integrácia do vývojového prostredia.

Pre prostredie Netbeans existujú rozšírenie len s veľmi základnou funkcionalitou a to možnosťou vytvárať nové projekty. Rozšírenie pre Symfony poskytuje naviac integráciu nástroja príkazového riadku. Pre Zend Framework ponúka PhpStorm len obmedzené rozšírenie, zahrňujúce identickú funkcionalitu spomínanú vyššie. Rozšírenie pre Symfony je ale na druhej strane príkladom

pokročilej integrácie s vývojovým prostredím. Medzi hlavné funkcie, ktoré je potrebné vyzdvihnúť, patrí automatické dopĺňanie veľkého množstva častí, napr. anotácií u entít a radičov, definovaných smerovaní, názvy súborov z pohľadmi, služieb registrovaných v Dependency Injection kontajnery a iných.

Krivka učenia

Dôsledkom toho, že oba frameworky ponúkajú veľké množstvo funkcionality, je začiatok vývoja pomerne komplikovaný. Hoci sa komunita snaží pomôcť začiatočníkom kvalitnou dokumentáciou, je krivka učenia pomerne strmá. Až po určitom čase, kedy vývojár objaví spojitosti a možnosti jednotlivých komponent, je možný efektívny vývoj aplikácií.

Vďaka rozsiahlej dokumentácií, existujúcej štandardnej distribúcie a veľkého množstva uľahčení pri konfigurácií (napr. pomocou anotácií) ponúka Symfony jednoduchší odrazový bod pre vývojárov.

Tabuľka 19 - Vyhodnotenie kritérií v kategórií podpora frameworku

	Váha	Symfony 2	Zend Framework 2
Kvalita dokumentácie	0,4	1	0,7
Počet diskusií v Google Groups	0,05	1	0
Počet otázok na Stack Overflow	0,1	1	0,21
Dostupnosť knižiek	0,05	0,2	1
Podpora vo vývojových prostrediach	0,2	0,7	0,25
Krivka učenia	0,2	0,6	0,3
	1	**0,82**	**0,46**

7.13. Finálne hodnotenie kritérií

V práci bolo nadefinovaných štyridsať osem kritérií, ktoré boli rozdelené do dvanástich kategórií. Tieto kategórie budú porovnané pomocou Fullerovej metódy. Pri aplikácií tejto metódy sa zostavuje tzv. Fullerov trojuholník, ktorý je tvorený dvoj riadkami, v ktorých sa každá dvojica kategórií vyskytuje práve raz. U jednotlivých dvojíc sa určí, ktorá kategória je dôležitejšia. Ak sú porovnávané kategórie rovnako dôležité, nie je zvolená žiadna preferencia. Každej kategórii je týmto spôsobom určená jej výsledná preferencia.

Tabuľka 20 - Kategórie kritérií

	Názov kategórie kritérií
K1	Architektúra frameworku a zavedené konvencie
K2	Podporované technológie
K3	Databázy
K4	Pohľady
K5	Radiče
K6	Komponentový návrh a znova použiteľnosť

K7	Formuláre
K8	Bezpečnosť
K9	Internacionalizácia a lokalizácia
K10	Nástroje príkazového riadku
K11	Testovanie
K12	Podpora a iné

Tabuľka vyššie obsahuje zoznam kategórií a ich označenia, ktoré je použité pri tvorbe Fullerového trojuholníka.

Tabuľka 21 - Fullerov trojuholník

K1	K1	**K1**	K1	K1	**K1**	K1	**K1**	**K1**	**K1**	**K1**
K2	**K3**	K4	**K5**	K6	K7	**K8**	K9	K10	K11	K12
	K2	K2	K2	K2	K2	K2	**K2**	**K2**	K2	K2
	K3	**K4**	**K5**	**K6**	**K7**	**K8**	K9	K10	**K11**	K12
		K3	K3	**K3**	K3	K3	**K3**	**K3**	**K3**	**K3**
		K4	K5	K6	K7	K8	K9	K10	K11	K12
			K4	**K4**	K4	K4	**K4**	**K4**	K4	**K4**
			K5	K6	**K7**	**K8**	K9	K10	**K11**	K12
				K5	K5	K5	**K5**	**K5**	K5	**K5**
				K6	**K7**	**K8**	K9	K10	K11	K12
					K6	K6	**K6**	**K6**	K6	**K6**
					K7	**K8**	K9	K10	**K11**	K12
						K7	**K7**	**K7**	K7	**K7**
						K8	K9	K10	**K11**	K12
							K8	**K8**	**K8**	**K8**
							K9	K10	K11	K12
								K9	K9	**K9**
								K10	**K11**	K12
									K10	K10
									K11	**K12**
										K11
										K12

Počet porovnaní N je kombináciou dvoch prvkov z počtu definovaných kritérií, teda je rovný kombinačnému číslu. V prípade tejto práce je to $N = \binom{12}{2} = 66$ a teda pri určovaní preferencií sa vykoná 66 porovnaní. Normovaná váha kritéria Kj je vypočítaná ako $w_j = \frac{f_j}{N}$, kde f_j je celkový počet preferencií kategórie a N je celkový počet určených preferencií pri porovnaní. Výpočty normovaných váh sú v Tabuľka 22.

Tabuľka 22 - Výpočet normovanej váhy kategórií

	Názov kategórie kritérií	**Počet preferencií**	**Váha**
K1	Architektúra frameworku a zavedené konvencie	7	$7/60$
K2	Podporované technológie	2	$1/30$

K3	Databázy	8	2/15
K4	Pohľady	5	1/12
K5	Radiče	7	7/60
K6	Komponentový návrh a znova použiteľnosť	4	1/15
K7	Formuláre	7	7/60
K8	Bezpečnosť	10	1/6
K9	Internacionalizácia a lokalizácia	2	1/30
K10	Nástroje príkazového riadku	0	0
K11	Testovanie	6	1/10
K12	Podpora a iné	2	1/30
	Celkom	**60**	**1**

Vypočítané normované váhy sú použité na výsledný výpočet hodnotenia kritérií oboch frameworkov.

Tabuľka 23 - Celkové vyhodnotenie kritérií

	Váha	**Symfony 2**		**Zend Framework 2**	
		Hodnotenie v kategórií	Výsledné hodnotenie	Hodnotenie v kategórií	Výsledné hodnotenie
K1	7/60	1	0,12	1	0,12
K2	1/30	0,92	0,03	0,95	0,03
K3	2/15	1	0,13	0,43	0,06
K4	1/12	1	0,08	0,84	0,07
K5	7/60	1	0,12	1	0,12
K6	1/15	0,94	0,06	0,51	0,03
K7	7/60	0,84	0,10	0,96	0,11
K8	1/6	1	0,17	1	0,17
K9	1/30	0,87	0,03	0,95	0,03
K10	0	1	0,00	0,4	0,00
K11	1/10	1	0,10	1	0,10
K12	1/30	0,82	0,03	0,46	0,02
	1	**11,395**	**0,96**	**9,5**	**0,85**

Tabuľka 23 obsahuje finálne vyhodnotenie kritérií s využitím normovaných váh pre jednotlivé kategórie. Tým, že kritéria skúmajú základné a všeobecné vlastnosti frameworkov je možné usúdiť, že vhodnejším kandidátom na tvorbu webových aplikácií je framework Symfony 2. Výsledné hodnotenie sa môže líšiť, ak by stanovené kritéria boli definované pre špecifický typ aplikácií.

Mimo vyššie uvedený záver, je z čiastkových aj finálnych výsledkov možné taktiež usúdiť, že porovnávané frameworky sú si veľmi podobné. Hoci podpora niektorých funkcií a možností je u Zend Framework 2 menšia, všeobecné princípy na tvorbu aplikácií sú rovnaké.

8. Záver

Práca sa venuje porovnaniu dvoch webových aplikačných frameworkov, na základe širokej škály hodnotiacich kritérií. Informácie získané na takéto zhodnotenie boli čerpané z dokumentácií, z článkov na Internete ale hlavne zo skúseností, ktoré autor načerpal pri tvorbe ukážkových aplikácií.

V úvodnej časti práce boli stanovené hodnotiace kritéria, ktoré umožňujú všeobecné porovnanie frameworkov bez obmedzenia na ich veľkosť, či platformu. Tieto kritéria skúmajú frameworky na základne ich navrhnutej architektúry, podporovaných technológií, všeobecnej podpory ale hlavne z hľadiska poskytovanej funkcionality pre vývojárov.

Práca sa následne venuje porovnaniu dvoch vybraných frameworkov a to Symfony 2 a Zend Framework 2. V kapitolách venujúcich sa vývoju ukážkovej aplikácie, boli demonštrované základné koncepty tvorby webových aplikácií a akým spôsobom k nim jednotlivé frameworky pristupujú.

Oba hodnotené frameworky ponúkajú základnú kostru pre vývoj veľkých webových aplikácií a pridávajú podporu pre riešenie všeobecných úloh pri vývoji. Napriek tomu, že ponúkajú komplexné riešenie jednotlivých úloh, neobmedzujú vývojárov pri vlastnej implementácií. Umožnené je to vďaka kvalitnému návrhu a možnosti jednoduchej rozšíriteľnosti pomocou vlastných komponent. Frameworky sa líšia spôsobmi konfigurácie jednotlivých komponent a rozsahom podpory určitej funkcionality.

Na základe získaných informácií pri tvorbe ukážkovej aplikácie boli následné oba frameworky porovnané podľa stanovených kritérií. Z výsledkov je možné usúdiť, že framework Symfony 2 dáva vývojárom lepšie možnosti a výhody pri tvorbe webových aplikácií.

Terminologický slovník

Termín	Skratka	Popis
Access Control List	ACL	Metóda riadenie prístupov v aplikácií. (Autor)
Cascading Stylesheets	CSS	Kaskádové štýly sú jazyk používaný na definovanie vzhľadu HTML dokumentov.
Structured Query Language	SQL	Neprocedurálne jazyk, ktorý slúžia na prácu s dátami v relačných databázach. (Autor)
Scaffolding	Scaffolding	Metóda generovania zdrojového kódu zdrojového kódu užívateľského rozhrania aplikácie, ktorý umožňuje vykonávať základné operácie s databázovou tabuľkou. (VÁLEK, 2008)
Object-Relational Mapping	ORM	Mapovanie doménových objektov a vzťahov medzi nimi na tabuľky relačných databáz. (TICHÝ, 2010)
Object-oriented programming	OOP	Spôsob programovania softwaru, ktorý vychádza z myšlienky, že všetky programy ktoré vytvárame sú simuláciou skutočného alebo virtuálneho sveta. (PECINOVSKÝ, 2009) Jednotlivé časti sú reprezentované ako objekty, ktoré medzi sebou komunikujú.
HyperText Markup Language	HTML	Značkovací jazyk využívaný na tvorbu webových stránok. (Autor)
Cache	Cache	Vyrovnávacia pamäť využívaná vo webových aplikáciách na uloženie dočasných údajov . (Autor)
eXtensible Markup Language	XML	Značkovací jazyk, ktorý slúži k popisu dát (Autor)
YAML Ain't Markup Language	YAML	Formát a štandard pre ukladanie serializovaných dát v rozličných programovacích jazykoch. (YAML, 2013)
Content management system	CMS	Ide o skupina nástrojov alebo technológií, ktoré nám to umožňujú efektívne štruktúrované ukladanie, zobrazovanie a opätovné použitie obsahu. (KOŠČO, 2011)
PHP Data Objects	PDO	Konzistentné rozhranie pre prístup k databázam. Ide o abstraktnú vrstvu pristupujúcu k databázam. (PHP DOCUMENTATION GROUP, 2013)
Hypertext Transfer Protocol	HTTP	Protokol používaný při komunikaci v rámci služby World Wide Web (KUTIŠOVÁ, 2013)
PHP: Hypertext Preprocessor	PHP	Skriptovací programovací jazyk. (Autor)

Zdroje

BÖHMER, Marian. *Návrhové vzory v PHP*. 1. vyd. V Brně: Computer Press, 2012, 320 s. ISBN 978-80-251-3338-5.

D'SOUZA, Desmond Francis a Alan Cameron WILLS. *Objects, components, and frameworks with UML*. Reading, Mass.: Addison-Wesley, 1999 [cit. 2013-11-16.], xxv, 785 p. ISBN 02-013-1012-0. Dostupné z: http://www.catalysis.org/books/ocf/index.htm

DVOŘÁK, Miloš. *Design patterns a jejich využití při objektovém návrhu aplikací v prostředí Internetu*. Dostupné z: http://objekty.vse.cz/Objekty/Vzory. Praha, 2003. Diplomová práca. Vysoká škola ekonomická v Praze, Fakulta informatiky a statistiky, Katedra informačních technologií. Vedúci práce Helena Jilková.

FIALA, Petr a Josef JABLONSKÝ. *Vícekriteriální rozhodování*. Praha: Vysoká škola ekonomická, 1997, 316 s. ISBN 80-707-9748-7.

FINLEY, Klint. Github Has Surpassed Sourceforge and Google Code in Popularity. In: *ReadWrite* [online]. 2011-06-02 [cit. 2013-10-27]. Dostupné z: http://readwrite.com/2011/06/02/github-has-passed-sourceforge

FOWLER, Martin. *Patterns of Enterprise Application Architecture*. Boston: Pearson Education, Inc., 2002. ISBN 978-0321127426.

FRIEBELOVÁ, Jana. *Vícekriteriální hodnocení variant*. České Budějovice, 2009. Dostupné z: http://www2.ef.jcu.cz/~jfrieb/tspp/data/teorie/Vicekritko.pdf

GILMORE, Jason W. *Velká kniha PHP 5 MySQL: kompendium znalostí pro začátečníky i profesionály*. Vyd. 1. Brno: Zoner Press, 2005, 711 s. ISBN 80-868-1520-X.

GRUDL, David. *Escapováni: Definitivní příručka*. [online]. 19.5.2009 [cit. 2013-11-03]. Dostupné z: http://phpfashion.com/escapovani-definitivni-prirucka

GITHUB, Inc. *GitHub* [online]. 2013 [cit. 2013-10-26]. Dostupné z: https://www.github.com

GOOGLE, Inc. *Skupiny* [online]. 2013a [cit. 2013-10-26]. Dostupné z: https://groups.google.com/

GOOGLE, Inc. *Google Trends* [online]. 2013b [cit. 2013-10-26]. Dostupné z: http://www.google.cz/trends/explore?q=symfony#q=Symfony%2C%20Zend%20Framework%2C%20Laravel%2C%20CodeIgniter%2C%20Yii&cmpt=q

HRUBÝ, Jan. *Využití frameworku Django pro tvorbu informačních systémů*. Praha, 2010. Diplomová práca. Fakulta informatiky a statistiky, Katedra informačních technologií. Vedúci práce Jarmila Pavlíčková.

KOŠČO, Ján. *Prípadová štúdia webovej aplikácie s využitím open source prostriedkov*. Praha, 2011. Bakalárska práca. Fakulta informatiky a statistiky, Katedra informačních technologií. Vedúci práce Vilém Sklenák.

KNPLABS. *KnpBundles: Discover bundles for Symfony2* [online]. 2012 [cit. 2013-11-26]. Dostupné z: http://knpbundles.com/

KŘIŽAN, Ondřej. *Vybrané frameworky pro vývoj aplikací v PHP a jejich srovnání*. Praha, 2010. Diplomová práca. Vysoká škola ekonomická v Praze, Fakulta informatiky a statistiky, Katedra informačních technologií. Vedúci práce Luboš Pavlíček.

LAPÁČEK, Vladimír. *Bezpečnost při vývoji aplikací*. Praha, 2011. Diplomová práca. Vysoká škola ekonomická v Praze, Fakulta informatiky a statistiky, Katedra informačních technologií. Vedúci práce Alena Buchalcevová.

NOBACK, Matthias. *A Year With Symfony: Writing healthy, reusable Symfony2 code* [online]. Vancouver: Ruboss Technology Corporation, 2013, 172 s. Dostupné z: https://leanpub.com/a-year-with-symfony

MARCEY, Joel. Locking Down for Performance and Parity. In: *HipHop Virtual Machine for PHP* [online]. 7.11.2013 [cit. 2013-11-23]. Dostupné z: http://www.hhvm.com/blog/1499/locking-down-for-performance-and-parity

O'PHINNEY, Matthew Weier. Zend Framework 2.2.0 Stable Released!. In: *ZF Blog* [online]. 15.5.2013 [cit. 2013-11-24]. Dostupné z: http://framework.zend.com/blog/zend-framework-2-2-0-stable-released.html

PECINOVSKÝ, Rudolf. *Myslíme objektově v jazyku Java: kompletní učebnice pro začátečníky*. 2., aktualiz. a rozš. vyd. Praha: Grada, 2009, 570 s. Myslíme v--. ISBN 978-80-247-2653-3.

PHP DOCUMENTATION GROUP. *PHP Manual* [online]. © 1997-2013 [cit. 2013-11-25]. Dostupné z: http://www.php.net/manual/en/

PROTALINSKI, Emil. Facebook Now Has 1.19 Billion Monthly Active Users. In: The Next Web [online]. 2013-10-30 [cit. 2013-11-30]. Dostupné z: http://thenextweb.com/facebook/2013/10/30/facebook-passes-1-19-billion-monthly-active-users-874-million-mobile-users-728-million-daily-users/

ROMER, Michael. *Web Development with Zend Framework 2: Concepts, Techniques and Practical Solutions* [online]. Vancouver: Ruboss Technology Corporation, 2012, 452 s. Dostupné z: https://leanpub.com/zendframework2-en

SENSIOLABS. *High Performance PHP Framework for Web Development – Symfony* [online]. © 2005 - 2013. Dostupné z: http://symfony.com

ŠPLÍCHALOVÁ, Marcela. *Metodika testování webových aplikací*. Praha, 2008. Diplomová práca. Vysoká škola ekonomická v Praze, Fakulta informatiky a statistiky, Katedra informačních technologií. Vedúci práce Alena Buchalcevová.

STACK EXCHANGE, Inc. *Stack Overflow* [online]. 2013 [cit. 2013-10-26]. Dostupné z: http://stackoverflow.com/

STOCKER, Christian. *HHVM with Symfony 2 looks amazing*. [online]. 29.10.2013 [cit. 2013-11-03]. Dostupné z: http://blog.liip.ch/archive/2013/10/29/hhvm-and-symfony2.html

ZEND TECHNOLOGIES LTD. *Zend Framework* [online]. © 2006 - 2013. Dostupné z: http://framework.zend.com

Q-SUCCESS. *Usage of server-side programming languages for websites*. *World Wide Web Technology Surveys* [online]. 2013 [cit. 2013-11-22]. Dostupné z: http://w3techs.com/technologies/overview/programming_language/all

THE PHP GROUP. *SQL Injection*. [online]. 2001-2013 [cit. 2013-11-23]. Dostupné z: http://php.net/manual/en/security.database.sql-injection.php

TICHÝ, Ján. *Seriál o bezpečnosti sessions: Nenechte si uhodnout Session ID*. [online]. 2008a [cit. 2013-11-23]. Dostupné z: http://www.phpguru.cz/clanky/nenechte-uhodnout-sid

TICHÝ, Ján. *Seriál o bezpečnosti sessions: Session hijacking aneb ukradení session ID*. [online]. 2008b [cit. 2013-11-23]. Dostupné z: http://www.phpguru.cz/clanky/session-hijacking

TICHÝ, Jan. Doctrine 2. In: *Zdroják* [online]. 21.7.2010 [cit. 2013-11-24]. Dostupné z: http://www.zdrojak.cz/serialy/doctrine-2/

TIOBE Programming Community Index for November 2013. TIOBE Software [online]. 2013 [cit. 2013-11-22]. Dostupné z: http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html

VÁLEK, Ondřej. *Webové aplikační frameworky pro rychlý vývoj aplikací*. Praha, 2008. Diplomová práca. Vysoká škola ekonomická v Praze, Fakulta informatiky a statistiky, Katedra informačních technologií. Vedúci práce Luboš Pavlíček.

WIKIPEDIA. Coding conventions. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001-, 2013-11-26 [cit. 2013-11-30]. Dostupné z: http://en.wikipedia.org/wiki/Coding_conventions

EVANS, Clark C. *YAML Ain't Markup Language*. [online]. 2001 [cit. 2013-12-10]. Dostupné z: http://www.yaml.org/

ZEND TECHNOLOGIES LTD. *ZF2 Modules: Browse modules for Zend Framework 2* [online]. © 2005 - 2012 [cit. 2013-11-26]. Dostupné z: http://modules.zendframework.com/

ZEND TECHNOLOGIES LTD. Programmer's Reference Guide of Zend Framework 2 [online]. © 2006 - 2013 [cit. 2013-11-26]. Dostupné z: http://framework.zend.com/manual/2.2/en/index.html

Zoznam obrázkov

Obrázok 1 - Diagram prípadov použitia ukážkovej aplikácie 22
Obrázok 2 - Fyzický dátový model ukážkovej aplikácie 23
Obrázok 3 - Detail hodnotenia a bočný panel s posledným užívateľovými hodnoteniami 24
Obrázok 4 - Zoznam hodnotení podniku 25
Obrázok 5 - Dialóg na pridávanie hodnotení 25
Obrázok 6 - Priebeh spracovania HTTP požiadavky a routovania v Symfony 2 36
Obrázok 7 - Generovanie nového bundlu pomocou konzoly 53

Zoznam tabuliek

Tabuľka 1 - Podporované verzie PHP u českých hostigových spoločností 8
Tabuľka 2 - Počet položených otázok na servery Stack Overflow 18
Tabuľka 3 - Počet užívateľov služby GitHub.com sledujúcich repozitáre frameworkov 18
Tabuľka 4 - Celkový počet prispievateľov do zdrojového kódu 19
Tabuľka 5 - Počet zmien za 30 dní v projektovom repozitáry 19
Tabuľka 6 - Dátumy vydania posledných verzií frameworkov 20
Tabuľka 7 - Podporované databázové systémy 31
Tabuľka 8 - Vyhodnotenie kritérií v kategórií architektúra frameworku 56
Tabuľka 9 - Vyhodnotenie kritérií v kategórií podporované technológie 57
Tabuľka 10 - Vyhodnotenie kritérií v kategórií databáze 58
Tabuľka 11 - Vyhodnotenie kritérií v kategórií pohľady 59
Tabuľka 12 - Vyhodnotenie kritérií v kategórií radiče 59
Tabuľka 13 - Vyhodnotenie kritérií v kategórií komponentový návrh 60
Tabuľka 14 - Vyhodnotenie kritérií v kategórií formulárov 61
Tabuľka 15 - Vyhodnotenie kritérií v kategórií bezpečnosť 61
Tabuľka 16 - Vyhodnotenie kritérií v kategórií internacionalizácia a lokalizácia 61
Tabuľka 17 - Vyhodnotenie kritérií v kategórií nástroje príkazového riadku 62
Tabuľka 18 - Vyhodnotenie kritérií v kategórií testovanie 62
Tabuľka 19 - Vyhodnotenie kritérií v kategórií podpora frameworku 64
Tabuľka 20 - Kategórie kritérií 64
Tabuľka 21 - Fullerov trojuholník 65
Tabuľka 22 - Výpočet normovanej váhy kategórií 65
Tabuľka 23 - Celkové vyhodnotenie kritérií 66

Zoznam výpisov

Výpis 1 - Vytvorenie nového projektu pomocou nástroja Composer 27
Výpis 2 - Nastavenie pripojenia aplikácie k MySQL serveru v Symfony 2 27
Výpis 3 - Nastavenie pripojenia aplikácie k MySQL serveru v Zend Frameworku 28
Výpis 4 - Definícia prístupových údajov k databázovému serveru v Zend Frameworku 28
Výpis 5 - Nastavenie komponenty FOSUserBundle 30
Výpis 6 - Registrácia modulu v Zend Frameworku 30
Výpis 7 - Konfigurácia radičov v základnom module Application 31
Výpis 8 - Definícia entity hodnotenia v Symfony 2 32
Výpis 9 - Používanie štandardných funkcií Entity Repository 33
Výpis 10 - Vytvorenie vlastného dotazu v repository 33
Výpis 11 - Definícia objektu reprezentujúceho databázovú tabuľku v Zend Frameworku 34
Výpis 12 - Nastavenie spôsobu tvorby inštancií tried 35
Výpis 13 - Nastavenie trasy v Symfony 2 36
Výpis 14 - Definícia trasy pomocou anotácie v radiči 37
Výpis 15 - Definícia trasy v Zend Frameworku 37
Výpis 16 - Vytvorenie bleskovej správy v Zend Frameworku 38
Výpis 17 - Vytvorenie bleskovej správy v Symfony 38
Výpis 18 - Spôsob definovanie formulárov v Symfony 2 39
Výpis 19 - Asociovanie entity s formulárom v Symfony 2 40
Výpis 20 - Definícia validácie v entite užívateľ 40
Výpis 21 - Definícia validácie formulárového elementu 41
Výpis 22 - Definícia formulára pomocou továrnej metódy 41
Výpis 23 - Tvorba formulára v Zend Frameworku 42
Výpis 24 - Definícia validácií pre formuláre v Zend Frameworku 43
Výpis 25 - Definovanie globálnej šablóny 44
Výpis 26 - Definícia zástupného bloku v Layoute 44
Výpis 27 - Nové pomocné makro a jeho import v rámci šablóny 45
Výpis 28 - Konfigurácia JavaScriptových súborov 45
Výpis 29 - Konfigurácia autentizácie a autorizácie v Symfony 2 46
Výpis 30 - Definícia zoznamu pre riadenie prístupov 47
Výpis 31 - Tvorba parametrizovaného dotazu v Symfony 2 49
Výpis 32 - Tvorba parametrizovaného dotazu v Zend Frameworku 49
Výpis 33 - Vytvorenie autorizačného tokenu pre formulár 49

Výpis 34 - Preklad statických textov v Zend Frameworku 51
Výpis 35 - Preklad statických textov pomocou kľúčom definovaných textov v Symfony 51
Výpis 36 - Funkčný test s Zend Frameworku 2 54
Výpis 37 - Funkčný test prihlásenia do aplikácie v Symfony 55