TECHNICKÁ UNIVERZITA V LIBERCI

Fakulta mechatroniky, informatiky a mezioborových studií

BAKALÁŘSKÁ PRÁCE

Liberec 2013 **Jaroslav Jakoubě**

Příloha A

TECHNICKÁ UNIVERZITA V LIBERCI
Fakulta mechatroniky, informatiky a mezioborových studií

Studijní program: B2646 – Informační technologie
Studijní obor: 1802R007 – Informační technologie

Srovnání databázových knihoven v PHP

Benchmark of database libraries for PHP

Bakalářská práce

Autor: **Jaroslav Jakoubě**
Vedoucí práce: Mgr. Jiří Vraný, Ph.D.

V Liberci 15. 5. 2013

Prohlášení

Byl(a) jsem seznámen(a) s tím, že na mou bakalářskou práci se plně vztahuje zákon č. 121/2000 Sb., o právu autorském, zejména § 60 – školní dílo.

Beru na vědomí, že Technická univerzita v Liberci (TUL) nezasahuje do mých autorských práv užitím mé bakalářské práce pro vnitřní potřebu TUL.

Užiji-li bakalářskou práci nebo poskytnu-li licenci k jejímu využití, jsem si vědom povinnosti informovat o této skutečnosti TUL; v tomto případě má TUL právo ode mne požadovat úhradu nákladů, které vynaložila na vytvoření díla, až do jejich skutečné výše.

Bakalářskou práci jsem vypracoval(a) samostatně s použitím uvedené literatury a na základě konzultací s vedoucím bakalářské práce a konzultantem.

Datum

Podpis

Abstrakt

Česká verze:

Tato bakalářská práce se zabývá srovnávacím testem webových aplikací psaných v programovacím skriptovacím jazyce PHP, které využívají různé knihovny pro komunikaci s databází. Hlavní důraz při hodnocení výsledků byl kladen na rychlost odezvy při zasílání jednotlivých požadavků. V rámci řešení byly zjišťovány dostupné metodiky určené na porovnávání těchto projektů. Byl také proveden průzkum zjišťující, které frameworky jsou nejvíce používané.

Klíčová slova:

Testování, PHP, webové aplikace, framework, knihovny

English version:

This bachelor's thesis is focused on benchmarking of the PHP frameworks and their database libraries used for creating web applications. Crucial function of each application is communication with the database and obtaining requested data. In each test was measured the request response time for every single function. Important part of the work was research of available methods used for benchmarking of the web applications. In this work, has been also defined which are the most favorite frameworks.

Keywords:

Benchmarking, PHP, web application, framework, libraries

Obsah

Seznam obrázků

Seznam grafů

Veškeré obrázky a grafy použité v této práci, jsou uloženy ve větším rozlišení na přiloženém CD.

Seznam zkratek

- PHP – „Hypertext Preprocessor“, programovací skriptovací jazyk určený pro tvorbu webových aplikací
- MVC – „Model View Controller“, návrhový vzor využívaný v programování definující strukturu aplikace
- URL – „Uniform Resource Locator“, řetězec dané struktury definující umístění na internetu
- JVM – „Java Virtual Machine“, virtuální stroj umožňující práci s java aplikacemi
- IRC – „Internet Relay Chat“, protokol umožňující komunikaci v reálném čase přes internet
- Active Record - návrhový vzor pro práci s datovými zdroji
- HTML – „HyperText Markup Language“, značkovací jazyk využívaný pro tvorbu webových aplikací
- RSS – „Rich Site Summary“, soubor upozorňující na změny dané stránky
- ZIP – souborový fromát pro kompresi dat
- PDF – „Portable Document Format“, souborový formát pro ukládání dokumentů nezávislých na softwaru kde byly vytvořeny
- RDF – „Resource Description Framework“, model metadat pro modelování informací v různých syntaxích.

- XUL – „XML User Interface Language“, formát pro tvorbu multiplatformního grafického rozhraní
- DAO – „Data access object“, objekt pro přístup k databázovým datům
- HMVC – „Hierarchical model–view–controller“ návrhový vzor odvozený od MVC s využitím přísnější hierarchie
- BSD – „Berkeley Software Distribution“, odvozená verze Unixu vytvořená Kalifornskou univerzitou v Berkeley poskytující licence pro volné šíření aplikací
- PDO – „PHP Data Objects“, rozšíření jazyka PHP podporující objektový přístup k datům
- SQL – „Structured Query Language“, dotazovací jazyk určený pro práci s databází
- ORM - Objektově relační mapován, technika převádějící databázové záznamy na objektový model
- NotORM – knihovna pro práci s databází
- XML – „Extensible Markup Language“, značkovací jazyk
- AJAX – „Asynchronous JavaScript and XML“, technika umožňující tvorbu interaktivních stránek bez nutnosti načítání
- CRUD – „Create Read Update Delete“, označení základních operací pro práci s databází
- LAMP – označení pro serverovou sestavu využívající Linuxový operační systém, Apache HTTP server, MySQL databázový server a jazyk PHP, Python nebo Perl
- HTTP – „Hypertext Transfer Protocol“, internetový protokol určený pro výměnu hypertextových dokumentů
- POST – dotazovací metoda HTTP protokolu umožňující předat určitá data

1 Úvod

Téměř každý současný programátor by chtěl vytvořit svoji aplikaci co nejvýkonnější a nejstabilnější, a to co v nejkratším čase. Tento názor se netýká jen tvůrců programů pro operační systémy osobních počítačů, ale zasahuje do všech programátorských oblastí. V dnešní době existuje obrovské množství zařízení, které jsou řízeny aplikacemi navrženými programátorem nebo celým týmem různých programátorů. A právě tyto projekty je nezbytné neustále vylepšovat a upravovat, protože technologie, kde jsou použity, se nezastavitelně vyvíjejí a potřebují, aby jejich programové vybavení drželo krok s nimi.

Tato stálá potřeba nového softwaru si právě žádá, aby byl vytvořen co nejdříve a nejefektivněji. Proto jsou k dispozici již naprogramované a odladěné funkce nebo celé aplikace, které mohou být přímo implementovány do daného projektu. Tyto soubory různých nástrojů, usnadňující a urychlující programátorovu práci, se nazývají frameworky.

Při tvorbě webových aplikací lze narazit na značné množství těchto frameworků, kde každý obsahuje rozdílnou funkcionalitu. Proto je zapotřebí tyto jednotlivé aplikace mezi sebou porovnat a zjistit, které jsou vhodné pro aktuálně požadovaný typ použití. Z těchto důvodů bylo cílem této práce provést srovnání většího množství volně dostupných frameworků určených pro tvorbu webových aplikací komunikujících s databází. U veškerých provedených testů byl kladen důraz na to, aby byla porovnávána funkcionalita, která se nejčastěji vyskytuje u reálných aplikací.

Pokud by měl někdo zkájem aplikovat neznámý framework, nebo by chtěl porovnat svůj stávající s jinými, nejspíše by se snažil vyhledat srovnávací test dostupný na internetu. Nedostatkem je, že takováto srovnání jsou sice k dispozici, ale zaměřují se výhradně na tvorbu primitivní aplikace vypisující jedinou větu nebo jednoduchou práci s databází. Téměř každá webová stránka je ale interaktivní a její obsah se neustále přizpůsobuje požadavkům různých uživatelů. Výsledky vycházející z těchto článků se tedy ani zdaleka neblíží porovnání při zátěži reálné aplikace. I takovéto testy jsou sice k dispozici, ale zaměřují se pouze na malé množství různých frameworků, nebo jsou již několik let staré a tím neobjektivní pro tvorbu nových stránek.

Proto by tato práce měla poskytovat objektivnější porovnání, pomocí kterého by šlo snáze určit chování aplikace využívající daný framework v reálném provozu.

2 Podrobnější představení problematiky

2.1 Úvod do problematiky

Pokud bychom chtěli vytvořit interaktivní webovou aplikaci, pak je zapotřebí, aby pracovala s proměnlivými daty a ta nabízela uživateli na základě jeho požadavků. K tomu se nejčastěji využívá databáze, kam jsou zasílány požadavky na výpis nebo úpravu jejich dat. Vývoj takovéto plně funkční aplikace, obsahující veškeré obslužné nástroje, je velmi časově náročný a vyžaduje, aby programátor znal téměř všechny vlastnosti každé použité funkce.

2.2 Vývoj vlastních knihoven

Jednou z možností, jak vytvořit požadovanou webovou aplikaci, je pokračovat v přidávání více nových funkcí, třeba i s podporou jiných programátorů. Následně je možné ji upravit do podoby, kterou lze po jednoduché změně lze použít na tvorbu odlišného webu využívajícího stejnou funkcionalitu. Tím by byl vytvořen projekt, pro který se používá označení framework.

Další možností, kterou by bylo možné uplatnit, je využití aplikací, které již vytvořili jiní, třeba zkušenější programátoři Proto jsou k dispozici různé frameworky, které vznikly například spoluprácí velikého množství programátorů. Výhodou takového projektu je jeho odladěná struktura a velké množství dostupných funkcí.

V současné době je k dispozici značné množství frameworků, na kterých pracují celé týmy zkušených programátorů. Malý přehled těchto nejoblíbenějších nástrojů je k dispozici například na stránkách memeburn [2], na kterých je uveden jen zlomek ze všech aktuálně dostupných nástrojů. Zdrojové kódy celé struktury jsou volně ke stažení a každý je může libovolně využít, upravit dle svého nebo může navrhnout nějaký typ zlepšení. U některých z nich proběhlo až tolik speciálních úprav, které měly za následek odtržení upravované verze od původního jádra, a tak daly základ naprosto novému frameworku. Z principu této tvorby vyplývá, že každý projekt může a často i má jinak navržené jádro a celkovou strukturu, a tím je ovlivněna jeho rychlost a využitelnost.

Právě tyto dvě vlastnosti jsou stěžejními faktory při výběru mezi jednotlivými frameworky. Hlavní problém nastává právě v situaci, kdy programátor neví, který si má pro svou aplikaci zvolit. Na oficiálních stránkách každého frameworku jsou sice

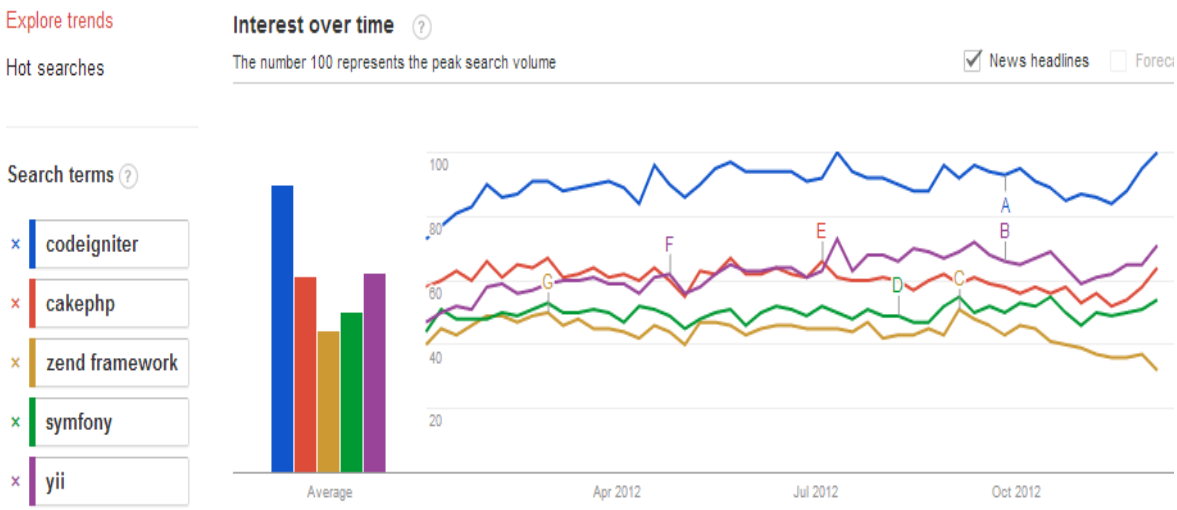
uvedeny jejich vlastnosti a výsledky z některých porovnávání, ale tyto údaje jsou zkreslující, protože většinou nejsou známy přesné konfigurace prováděného testu. Proto si uživatel buď vybere dle nějakého předchozího doporučení nebo se snaží najít nějaká objektivnější porovnání.

2.3 Druhy testování

Na internetu jsou k dispozici různé druhy testů, které mohou posloužit k drobné představě o výkonu či použitelnosti zkoumaného projektu, ale jen málo z nich se zabývá simulací chování aplikace v reálném provozu.

2.3.1 Abstraktní test

Nejabstraktnějším porovnáním jsou výsledky obsažené na webových stránkách WebcoderPRO [3], kde je určován nejlepší framework dle počtu jeho vyhledání přes vyhledávač Google. Srovnání je to velmi neobjektivní, protože z něj dokážeme vyčíst pouze počet vyhledání a nikoli jeho důvod. A tudíž výsledky mohou znamenat například nedostatečnou podporu na oficiálním webu a nutnost vyhledat podporu jinde. Na obrázku (Obr. 1) je znázorněn výsledek uveřejněný na těchto stránkách, který je vygenerován aplikací Google Trends.

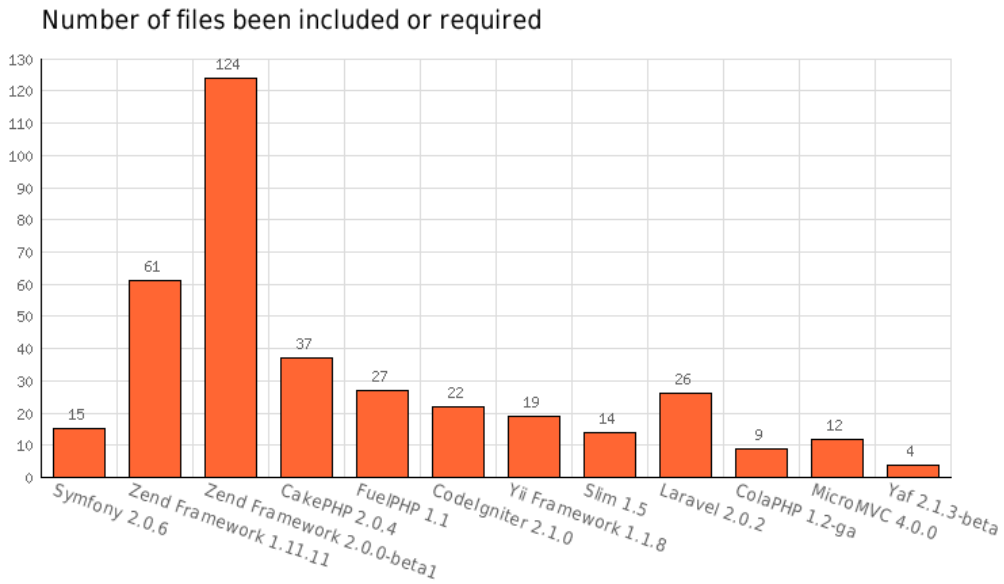


Obr. 1: Grafické znázornění počtu vyhledání

Zdroj: WebcoderPRO [3]

2.3.2 Jednoduchý test

Častou metodou porovnání je jednoduché vypsání libovolné věty, nejčastěji „Hello world!“, u které je měřena rychlost jejího vypsání. Tento způsob testování je využit například na stránkách Left Blank [4]. Jeden detailnější test tohoto typu je dostupný na blogu ruilog [5]. Zde se porovnávají frameworky využívající MVC strukturu a je u nich testována nejen rychlost odezvy, ale například i počet funkcí, které se při tomto výpisu volají, a přesná struktura jejich pořadí. Na obrázku (Obr. 2) je ukázka výsledků testu zaměřeného na měření počtu importovaných souborů pro jednoduchý výpis.



Obr. 2: Počty importovaných souborů při testu

Zdroj: ruilog [5]

Na tomto webu je testováno dvanáct různých aplikací, každá je napsána pomocí jiného frameworku. Výsledky jsou z roku 2011, což znamená, že od té doby byly určitě vydány nové verze.

Další test zaměřený na vypsání pouze jedné věty je k dispozici na stránkách Elefant CMS [6], které patří přímo frameworku Elefant zahrnutému ve srovnání. Tato měření jsou velice jednoduše pojata a k dispozici jsou pouze základní měřené údaje a grafická znázornění.

Podrobnější test a také obecný návod, jak tímto způsobem porovnávat různé aplikace, je obsažen na webových stránkách Dev Shed [7]. Na tomto portále je jednoduše vysvětlena charakteristika frameworku a jeho základní výhody. Také je zde uveden jednoduchý postup pro vlastní testování, ke kterému jsou v závěru zveřejněny

i výsledky. V tomto porovnávání je použit nástroj ApacheBench, který simuluje víceuživatelské zatížení dané aplikace. Na obrázku (Obr. 3) jsou znázorněny výsledky třetího testu z tohoto webu. Tabulka obsahuje data ze tří testů zaměřených na rychlost vyřízení požadavku.

PHP Framework	trial1	trial2	trial3	Average
	in milliseconds			
rainframework	439	442	477	452.67
Doophp	554	527	550	543.67
CodeIgniter	894	886	884	888.00
yii	1260	1277	1243	1260.00
Kohana	1313	1323	1336	1324.00
Symfony	2025	2039	2013	2025.67
cakephp	3654	3671	3667	3664.00

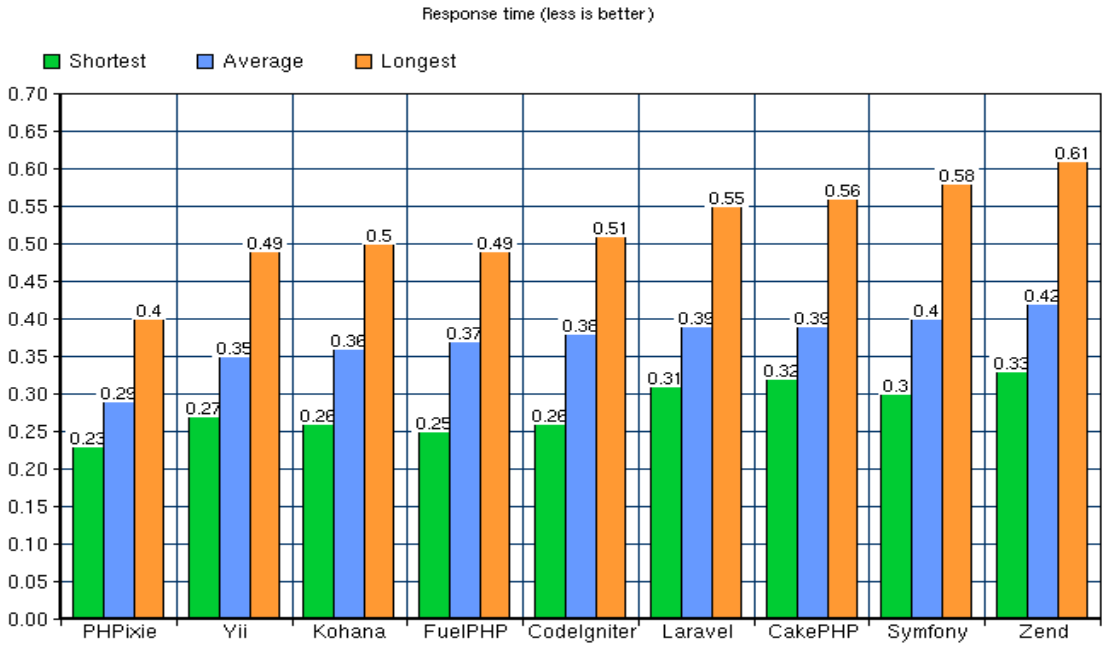
Obr. 3: Výsledky z ApacheBench

Zdroj: Dev Shed [7]

2.3.3 Testy reálnějších funkcí

Všechny výše uvedené testy pracují pouze s jednoduchým výpisem. Jak bylo zmíněno v úvodu, tak většího přiblížení reálné aplikaci dosáhneme testováním interaktivnějších stránek. Tento způsob testu je již méně obvyklý, a pokud byl někde proveden, tak pouze s malým počtem porovnávaných aplikací.

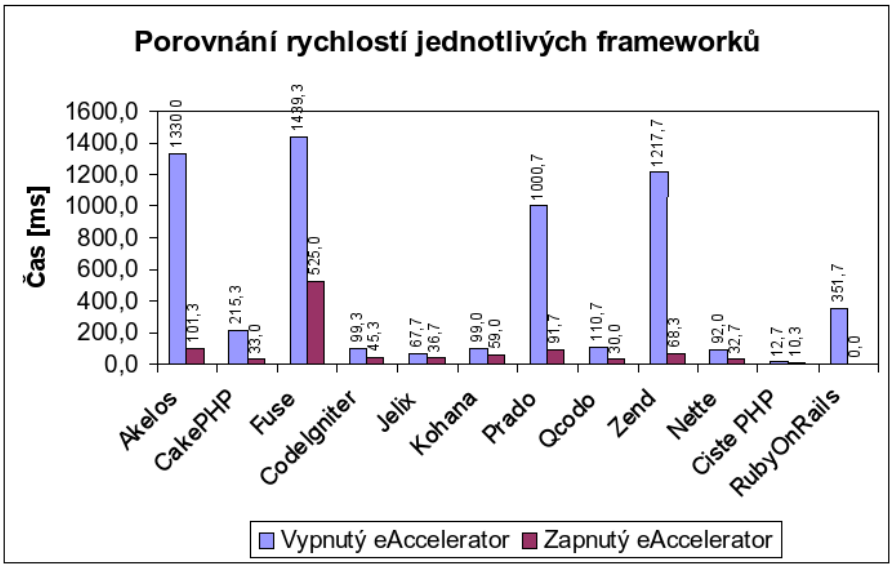
Jedním z dostupných porovnání tohoto typu se zabýval tým pracující na tvorbě frameworku PHPixie a je k dispozici na jeho oficiálních stránkách [8]. V průběhu testování byla měřena odezva aplikace obsahující funkce používané v reálném provozu. Program postupně získával data z URL, komunikoval s databází a vypisoval uživateli informace o požadovaných záznamech. Takováto měření byla provedena nejen u projektu využívajícího PHPixie, ale i u dalších šesti frameworků. Na obrázku (Obr. 4) lze vidět grafické znázornění výsledků tohoto testu v závislosti na rychlosti odezvy jednotlivých aplikací.



Obr. 4: Výsledky měření ze serveru PHPixie

Zdroj: PHPixie [8]

Podrobné srovnání bylo provedeno roku 2008 českým programátorem Petrem Daňkem a bylo publikováno na portále Root.cz [9]. Tento test byl určen právě k simulaci požadavků blížících se reálnému provozu. Pomocí programu Apache JMeter, který umožňuje funkci víceuživatelských požadavků, byly aplikacím zasílány příkazy, které nahrazovaly akce reálných uživatelů. Program si zobrazil údaje o jednotlivých uživatelích získané z databáze, následně si vypsal údaje o jednom konkrétním záznamu a v dalším kroku tyto informace upravil a zapsal zpět do původní databáze. Tím byla otestována jak funkcionalita výpisu formulářů a dat, tak rychlost práce s databází při různých operacích a paralelních požadavcích od různých uživatelů. V průběhu testu byly měřeny hodnoty týkající se rychlosti odezvy a paměťové náročnosti celého požadavku. Všechna měření byla prováděna několikrát a celkové výsledky byly v závěru zprůměrovány. Jednalo se o velice rozsáhlý test, protože obsahoval jedenáct nejznámějších frameworků. Na obrázku (Obr. 5) lze vidět závěrečné výsledky tohoto srovnání.



Obr. 5:Výsledky při měření rychlosti práce s databází

Zdroj: Root.cz[9]

2.4 Dostupné nástroje

Zaslání jednoduchého příkazu testovaným stránkám lze provádět ručně přímo ve spuštěné aplikaci, ale tímto způsobem nedochází k téměř žádnému zatížení a nelze se přiblížit reálnému provozu aplikace. Proto existují nástroje obsahující možnost zátěžového testu s daným počtem požadavků od specifikovaného počtu uživatelů.

2.4.1 ApacheBench

Jeden takovýto nástroj je k dispozici přímo v Apache webovém serveru a nazývá se ApacheBench. Je ovládán pouze pomocí příkazového řádku a slouží jen k jednoduchému měření rychlosti odezvy aplikace a zpracování požadavků. V příkazu lze zadat celkový počet požadavků, maximální množství paralelně připojených uživatelů, cílovou adresu a pár dalších užitečných atributů. Jedním z nich je cesta ke zdrojovému souboru, ve kterém jsou obsaženy hodnoty posílané přes POST metodu. Na obrázku (Obr. 6) je znázorněn výstup tohoto programu při testování rychlosti odezvy webu Yahoo!. Tyto výsledky byly čerpány z blogu Pete Freaitag [10], na kterém je jednoduše popsáno použití a vlastnosti tohoto nástroje.

```
Concurrency Level:      10
Time taken for tests:   1.889 seconds
Complete requests:      100
Failed requests:        0
Write errors:           0
Total transferred:      1003100 bytes
HTML transferred:       949000 bytes
Requests per second:    52.94 [#/sec] (mean)
Time per request:       188.883 [ms] (mean)
Time per request:       18.888 [ms] (mean, across all concurrent requests)
Transfer rate:          518.62 [Kbytes/sec] received

Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:       57   59   1.7     59      64
Processing:   117  126   7.5    124     162
Waiting:       57   62   7.0     60      98
Total:        175  186   8.0    184     224

Percentage of the requests served within a certain time (ms)
  50%    184
  66%    186
  75%    187
  80%    188
  90%    192
  95%    203
  98%    216
  99%    224
 100%    224 (longest request)
```

Obr. 6: Ukázka výstupu z ApacheBench

Zdroj: Pete Freitag's Blog [10]

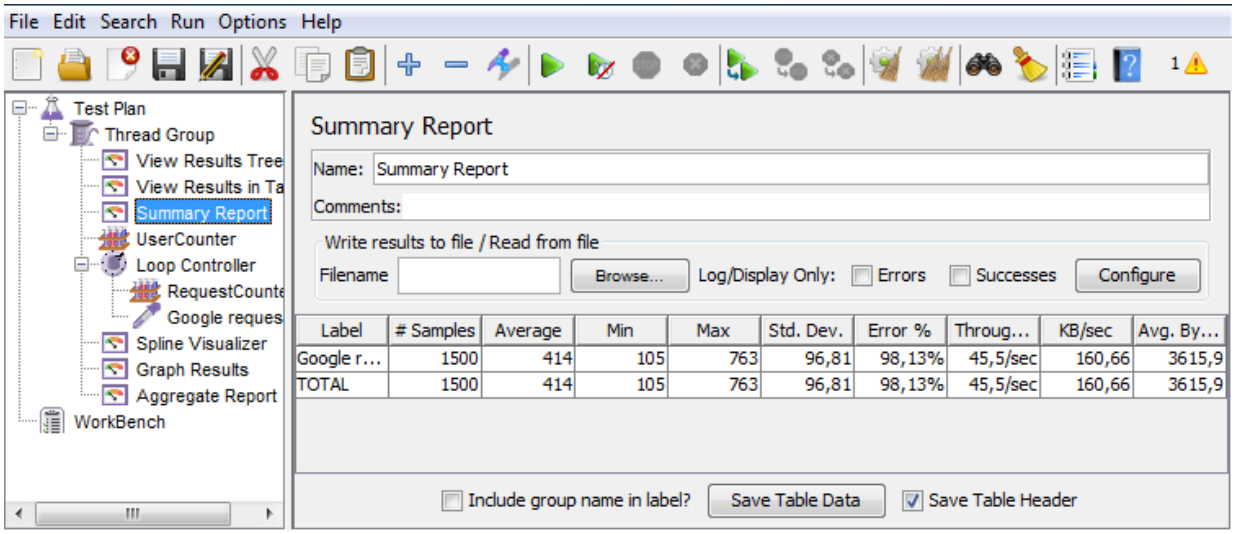
2.4.2 Apache JMeter

Propracovanějším nástrojem určeným nejen k zátěžovému testování webových aplikací je program Apache JMeter. Obsahuje dvě různá rozhraní - textové, které je pouze ve formě příkazového řádku, nebo grafické s velmi přehledným ovládáním. Je psán v programovacím jazyce Java, tím je i zaručena jeho kompatibilita a přenositelnost do všech druhů operačních systémů podporujících Java Virtual Machine (JVM).

Poskytuje veliké množství různých funkcí a také podporuje rozsáhlé možnosti rozšíření. Při správné konfiguraci se dá použít jako generátor jednotlivých příkazů nebo dokáže sloužit k monitorování provozu na daných stránkách. Veškeré výsledky mohou být zobrazeny pomocí různých druhů výstupu, mezi které patří několik typů grafů, tabulky se získanými daty nebo například zdrojové kódy vrácené aplikace. Jednotlivé příkazy se dají cyklit do různých smyček a lze jimi posílat libovolné parametry. Velmi snadné je i využití víceuživatelského zatížení, ve kterém lze nastavit maximální počet současně připojených uživatelů a interval, který řídí jejich postupné připojování.

V monitorovacím režimu lze zachytávat specifická data nebo jen soubory s definovaným formátem. Tento mód slouží například k tomu, aby si programátor ověřil, zda mu aplikace správně posílá požadavky nebo zda se mu zbytečně nevolají

nějaké funkce. Na obrázku (Obr. 7) je náhled výsledků této aplikace při měření rychlosti načtení domovské stránky vyhledávače Google pro 30 uživatelů, kde každý zadá 50 dotazů na zobrazení. Chybovost testu je způsobena ochranou serveru Google proti opětovnému načítání stejné stránky během krátkého okamžiku.



Obr. 7: Grafické prostředí programu Apache JMeter

2.5 Vybrané frameworky

Výběr frameworků využitých v této práci byl založen na rešerši zaměřené na jejich popularitu a funkcionalitu. V prvním kroku byl proveden průzkum různých webů obsahujících seznamy s „nejpoužívanějšími“ frameworky, ze kterých byl vybrán seznam projektů s nejčastějším výskytem. Údaje byly čerpány například z webu PHP Developer [11], z blogu Woork [12] a z velkého počtu dalších podobných stránek. Následně byl tento seznam konzultován s různými programátory webových aplikací. Komunikace byla prováděna přes mezinárodní a česká diskuzní fóra, například na portále Devel.cz [13]. Do domluvy byly zahrnuty i IRC kanály věnované některým frameworkům. Veškeré nasbírané údaje byly porovnány s projekty zahrnutými v různých dostupných testech a s údaji dostupnými na domovských stránkách každého z frameworků.

2.5.1 CakePHP

CakePHP patří mezi PHP frameworky používané pro tvorbu webových aplikací. Ve své základní struktuře využívá velké množství vlastností z Ruby on Rails. Je zaměřen na rychlý vývoj celé aplikace při využití návrhového vzoru MVC. Jeho struktura je srozumitelně uspořádána a tím je zajištěna i snadná orientace v celém kódu.

Poskytuje velké množství nástrojů používaných například pro jednoduché ověřování uživatelů nebo důkladnou validaci vstupních dat. Komunikuje s téměř všemi nejpoužívanějšími typy databází a při jejich obsluze využívá ActiveRecord. Díky jeho velké popularitě lze nalézt velké množství již hotových ukázkových aplikací nebo výukové kusy kódu. Na oficiálních stránkách [14] lze informace čerpat z velmi rozsáhlé dokumentace, která je rozdělena na část věnovanou jednotlivým funkcím a část zabývající se návody a postupy. Pro případnou podporu lze využít diskuzní fórum, IRC kanál nebo v této době nejvíce populární sociální sítě.

2.5.2 CodeIgniter

Tento framework je vyvíjen společností EllisLab, která na tvorbě spolupracuje s veřejnou komunitou programátorů. Jedná se o další známý projekt využívající veškeré moderní nástroje. Jádro je velice úsporně navrženo a lze ho rozšířit o velkou řadu doplňků. Díky tomu je celá struktura přehledná. Mezi jeho základní vlastnosti se řadí využívání návrhového vzoru MVC, zanechávání malého otisku v celé aplikaci a takzvaná čistá URL. V důsledku jeho velké popularity a veřejného kódu může každý schopný programátor vytvořit nějaký doplněk. Díky tomu existuje mnoho nástrojů na práci s e-maily, obrázky a hlavně dostupná komunikace s téměř každým používaným typem databáze. Rozdílem od jiných frameworků je, že nevyužívá šablony a tudíž se není potřeba učit nějaký jazyk určený právě k jejich tvorbě. Komunikace mezi jednotlivými příznivci probíhá přes oficiální IRC kanál nebo diskuzní fóra. Jako podpora velmi dobře poslouží i rozsáhlá dokumentace s mnoha ukázkovými příklady dostupnými na oficiálních stránkách [15].

2.5.3 DooPHP

DooPHP obsahuje jádro s menší paměťovou náročností, z čehož vyplývá, že v základě nepoužívá nadbytečné knihovny. Jedná se o framework s otevřeným kódem určeným pro rychlou tvorbu robustních aplikací. Je zaměřený výhradně na využívání objektového programování jazyka PHP. Vytvořené aplikace následně patří do skupiny s malým otiskem použitého frameworku. Na oficiálních stránkách [16] lze nalézt celkem rozsáhle zpracovanou dokumentaci, ovšem s trochu nepřehledným uspořádáním. Pro případné dotazy, nebo k řešení témat neobsažených v návodech, lze využít IRC kanál, diskuzní fórum nebo oficiální blog.

2.5.4 Jelix

Tento francouzský framework využívá programovací skriptovací jazyk PHP s doplněním o vlastní šablonovací systém s příponou *.jTpl. Obsahuje také podporu MVC návrhového vzoru a dalších moderních nástrojů. Jeho velká výhoda je podpora různých formátů výstupu obsahujících nejen standartní (X)HTML, ale také například RSS, ZIP, PDF, RDF, XUL… Od většiny ostatních se tento projekt liší tím, že pro vytvoření celého projektu nebo potřebných DAO objektů, je zapotřebí využít příkazový řádek. Velikou výhodou je možnost výběru z různých jednoduchých grafických rozhraní, která jsou již předdefinována a stačí je jen propojit s požadovanou funkcionalitou. K dispozici je i snadná rozšiřitelnost celého jádra o nové doplňkové nástroje. Komunita podporující Jelix není příliš rozsáhlá a to se projevuje i na dokumentaci. I přes její přehlednost zde chybí názorné návody a podrobnější příklady u novějších dostupných verzí. Sice na oficiálních stránkách [17] jsou k dispozici návody pro starší distribuce, ale ty někdy využívají funkce, které již nejsou dostupné a byly nahrazeny novějšími. Z těchto důvodů je zde velmi užitečná podpora ostatních příznivců Jelix frameworku, která je k dispozici v podobě oficiálního blogu, IRC kanálu, diskuzního fóra nebo stránek věnovaných rozšiřujícím doplňkům.

2.5.5 Kohana

Jedná se o velmi rozšířený, komunitou vytvořený aplikační framework, odvozený od původního CodeIgniter projektu. Ve své struktuře a návrhu aplikací využívá návrhový vzor MVC, přesněji HMVC. Jedná se o téměř stejný model, s rozdílem, že HMVC má pevně danou hierarchickou strukturu a přímou návaznost jednotlivých členů. Kohana nabízí možnost tvořit aplikace s BSD licencí, která umožňuje psát komerční aplikace. Jádro obsahuje velké množství užitečných ladících nástrojů, díky kterým lze jednoduše vyhledávat a odstraňovat problémy při tvorbě aplikace. Také je všechen „předepsaný“ kód velmi podrobně okomentován, díky tomu je v něm snadná orientace a není vždy nezbytné vyhledávat vysvětlení některých funkcí v dokumentaci. Hlavní podporované databáze jsou MySQL a PDO, ale lze najít rozšiřující balíčky umožňující komunikaci s jinými formáty. Dokumentace je trochu nepřehledná, ale velice rozsáhlá. Ostatní programátory lze kontaktovat přes IRC kanál nebo diskuzní fórum, vše je dostupné na oficiálních stránkách [18].

2.5.6 Laravel

Laravel patří mezi jedny z nejpodrobněji zdokumentované frameworky. Jeho rozsáhlá a velmi přehledná dokumentace je doplněna o velké množství názorných příkladů, ukázkových aplikací nebo dokonce výukových videí. Také lze zakoupit knihu věnovanou právě tomuto frameworku a tím i trochu přispět k jeho dalšímu vývoji. Struktura a návrh kódu je velmi intuitivní a pochopení jeho syntaxe je celkem jednoduché. Celé jádro obsahuje jen základní, i když rozsáhlou funkcionalitu, a lze ho rozšířit celou řadou přídavných balíčků, které jsou k dispozici přímo z domovské stránky [19]. Samozřejmostí je i využití objektového programování a MVC návrhového vzoru. Pro řešení témat, která nelze najít v dokumentaci, slouží velice frekventované diskuzní fórum a oficiální IRC kanál.

2.5.7 Nette

Českého programátora jistě zaujme značně populární framework Nette, který vznikl jako projekt českého tvůrce webových aplikací v PHP. Jedná se o framework využívající MVC strukturu a objektové programování s velkým důrazem na bezpečnost aplikace. Obsahuje mnoho ladících nástrojů, které pomáhají programátorovi tvořit bezchybný kód. Pro práci s databází využívá NotORM nebo Dibi knihovny, které podporují téměř všechny nejpoužívanější databázové formáty. Práce pomocí těchto knihoven je opravdu snadná a velmi rychlá. Pro návrh celé aplikace se využívá velmi srozumitelný a přehledný kód. Dokumentace je velice jasně a přehledně napsána a pro českého programátora je příjemná změna, že je kompletně v českém jazyce. Je v ní obsaženo veliké množství ukázek a příkladů, což velmi usnadní pochopení celkového programu. Díky velké popularitě je zde k dispozici i rozsáhlá komunita programátorů webových aplikací, kteří komunikují přes oficiální diskuzní fórum a IRC kanál dostupných na stránkách [20]. K dispozici je i stránka Planette [21], ve které lze nalézt mnoho ukázkových příkladů v podobě již hotových aplikací nebo podrobných návodů.

2.5.8 Prado

Název PRADO vznikl jako zkratka z anglického „PHP Rapid Application Development Object-oriented“, což vystihuje jeho základní vlastnosti, a to rychlý vývoj objektově orientovaných aplikací pomocí programovacího skriptovacího jazyka PHP. Rychlý vývoj samostatného projektu je podporován množstvím již implementovaných

funkcí a nástrojů. Jedním z nich je například zabudovaná možnost pro autorizaci nebo autentifikaci, která je velice stabilní a připravená k okamžitému použití nebo užitečné validační funkce kontrolující správnost získávaných dat. Aplikace jsou řízeny pomocí událostí a používají komponentový přístup. Také využívají asynchronní funkce JavaSriptu a XML, neboli AJAX, díky kterému je stránka schopná měnit svůj obsah bez nutnosti obnovování a tím zrychlit interaktivnost s uživatelem. Celý framework je velmi dobře zdokumentován a obsahuje různé podpůrné zdroje. V jedné části se dokumentace zabývá prvními kroky a podrobnou výukou vývoje aplikace a v druhé je zaměřena na detailnější vysvětlení jednotlivých funkcí a struktury. Na stránkách PradoSoft [22] jsou k dispozici i výuková videa a ukázkové aplikace. Samozřejmě je zde obsaženo diskuzní fórum sloužící pro komunikaci s ostatními programátory a řešení specifických problémů.

2.5.9 Qcodo

Qcodo je PHP framework, ve kterém lze jednoduše vytvořit jako malou tak i robustní webovou aplikaci. Při tvorbě se používá metoda přetváření existujícího datového modelu na model objektově relační, ke kterému jsou přidávány doplňkové funkce. Obsahuje snadné generátory kódu využívající výhradně vlastnosti PHP a AJAX. V jádru jsou zabudovány nástroje pro práci s databází využívající techniku objektově relačního mapování neboli ORM. Na domovských stránkách [23] jsou k dispozici ukázkové příklady a výuková videa. Jako podpůrný zdroj informací od ostatních programátorů slouží oficiální blog a různá diskuzní témata.

2.5.10 Recess

Tento framework využívá ke svému chodu jádro s velmi nízkou paměťovou náročností, které ve výsledné aplikaci zanechává jen malý otisk použitého projektu. Lze ho rozšířit o velké množství funkcí a nástrojů usnadňujících celý proces programování. V základní konfiguraci jsou zahrnuty velmi užitečné ladící a diagnostické nástroje, které pomáhají tvořit bezchybný a plně funkční kód. Návrh celé aplikace se řídí vlastnostmi objektového programování skriptovacího jazyka PHP, s využitím deklarací v poznámkách. Pro komunikaci s různými typy databází je použito objektově relační mapování, které automaticky propojuje výsledky se strukturou objektového programování. Využívá návrhový vzor MVC, díky kterému je velice přehledná i celá výsledná struktura aplikace. K bezchybnému přesměrovávání za běhu Recess

doporučuje mít povolen modul mod_rewrite, který usnadňuje práci s URL adresami. Veškeré návody a vysvětlení jednotlivých funkcí jsou zpracovány v podobě online dokumentace dostupné na domovských stránkách [24] nebo ve formě tištěné knihy. Případné dotazy, nebo nové nápady lze, sdílet s ostatními programátory webových aplikací prostřednictvím diskuzního fóra, blogu a oficiálního IRC kanálu.

2.5.11 Seagull

Seagull je framework využitelný k tvorbě nejen webových projektů, ale také ho lze využít při návrhu konzolových aplikací nebo grafického uživatelského rozhraní. Ve svém kódu se zaměřuje výhradně na objektově orientované programování jazyka PHP, propojené s návrhovým vzorem MVC. Jádro celého frameworku je velice úsporně navrženo a lze ho rozšířit o velké množství různých doplňků nebo podpůrných funkcí. Jednou z důležitých vlastností je možnost různých druhů výstupu aplikace nebo jednoduché propojení s kódem třetí strany. Na oficiálních stránkách [25] je dostupná nejen podrobná dokumentace v podobě projektu wiki, ale i velké množství ukázkových příkladů a výukových kusů kódu. Veškeré tyto informace lze stáhnout v off-line verzi. Podporu nebo informace od ostatních programátorů využívajících framework Seagull lze získat pouze přes dostupné sociální sítě nebo přes kontaktní formulář.

2.5.12 Symfony

Tento framework vznikl odvozením od projektu Mojavi3-DEV, který již využíval objektové programování. Současně se vyvíjí ve dvou různých verzích, kde každá z nich má troch jiné vlastnosti. Z části se nechal inspirovat jinými projekty, jako například Ruby on Rails, Django nebo Spring. Je velice oblíbený zejména díky snadnému vytváření jednotlivých aplikací nebo velkým možnostem rozšíření. Kompletní struktura má velmi stabilní jádro řízené programovacím skriptovacím jazykem PHP s využitím jeho objektového programování. Obsahuje různé užitečné nástroje, s jejichž pomocí lze například komunikovat s nejpoužívanějšími druhy dostupných databází. Vzhledem k jeho vysoké popularitě se na tvorbě a úpravách celého projektu podílí veliké množství různých programátorů. Celý oficiální web [26] je sepsán velmi zajímavou a čtivou formou. Tento způsob je dodržen i v celé přehledně strukturované a podrobné dokumentaci, kterou lze stáhnout i v podobě PDF knihy. Avšak chybí zde vetší množství ukázkových příkladů nebo již hotových aplikací, ale díky rozsáhlé komunitě lze najít spousta webů věnovaných právě frameworku Symfony. Komunikace

mezi programátory probíhá na běžně používaných portálech zahrnujících vše od sociálních sítí, přes IRC kanály a diskuzní fóra až po klasickou osobní domluvu pomocí emailu.

2.5.13 Yii

Pod tímto jednoduchým názvem se ukrývá zkratka z anglického „Yes It Is!“, neboli ve volném překladu „Ano je!“, která má znázorňovat přesnou odpověď na nejčastější otázky pokládané při zjišťování vlastností tohoto frameworku. Dle oficiálního webu [27] mezi ně patří například „Je Yii rychlý?“, „Je Yii spolehlivý?“ nebo „Je Yii bezpečný?“. Na všechny tyto dotazy by měl právě reagovat pouze název celého projektu. Mezi další klíčovou vlastnost celého jeho jádra patří obrovské využití objektového programování u základních komponent, které díky tomu lze využívat samostatně nebo velmi jednoduše k nim přidávat nové funkce. K nástrojům usnadňujících práci přímo při programování aplikace se řadí například generátor validního XHTML kódu nebo podpora grafických motivů a jednoduché změny celého vzhledu výsledného projektu. Aplikace napsaná pomocí Yii obsahuje ve svém základu výkonné zabezpečovací komponenty, které ji chrání před všemi různými typy útoků. Tento framework je velmi přehledně a podrobně zdokumentován a na oficiálních stránkách je i dostatečné množství návodů a ukázkových kódů. Dokumentace je publikována i v podobě tištěných knih. Kvůli rozsáhlému zájmu programátorů z různých zemí je stránka podpory rozdělena do různých jazykových kategorií.

2.5.14 Zend framework

Zend framework je další oblíbený nástroj určený pro rychlou tvorbu robustních webových aplikací s pomocí jazyka PHP. V jeho struktuře je využito objektově orientované programování a návrhový vzor MVC a celé jádro je děleno na jednotlivé moduly, které lze využít samostatně. Implementovány jsou i užitečné autentizační a autorizační nástroje, které zprostředkovávají chráněný uživatelský přístup k aplikaci. Zajišťuje možnost komunikace s téměř všemi druhy nejčastěji používaných databází a v případě nekompatibility lze vytvořit nebo sehnat modul doplňující chybějící funkcionalitu. Dokumentace je velmi podrobně a rozsáhle sepsána a lze se v ní snadno orientovat. K dispozici je velké množství jednoduchých návodů a ukázek kódu pro snadnější začátky a rychlé pochopení celé struktury. Náročnější programátor se

může přihlásit i do různých výukových kurzů. Na domovských stránkách tohoto projektu [28] lze s ostatními programátory komunikovat výhradně přes IRC kanál.

3 Cíle práce

Hlavním cílem této práce bylo porovnat chování nejpoužívanějších PHP frameworků a v nich obsažených knihoven určených pro práci s databází. Celý test byl řízen metodikou zvolenou na základě rešerše dostupných testů.

3.1 Upřesnění zadání

Do kompletního testu nebyly zahrnuty jen databázové knihovny, i když právě ty byly jedním z hlavních faktorů, který ovlivňoval výsledky testu. Byly testovány celé frameworky, které tyto knihovny obsahují a používají je ve svém jádru. Proto byla tato práce zaměřena především na otestování většího množství nejpoužívanějších a nejznámějších frameworků při práci s databází a tím využití právě těchto knihoven.

- Jednou z prvních částí, které muselo být dosaženo rozsáhlou rešerší, bylo zjistit, jaké frameworky jsou v dnešní době nejvíce používané a na které lze aplikovat srovnávací test.
- Dalším cílem bylo zjistit, jaké různé metodiky se používají přímo na porovnávání webových aplikací a vybrat z nich takové, které by splňovaly požadovanou funkcionalitu. S tím souvisí vyhledání a pochopení nástrojů používaných pro testování.
- Po výše zmíněné rešeršní práci byly provedeny části programátorské. V této fázi bylo cílem nainstalování a nakonfigurování vlastního webového serveru.
- Aby bylo možno ukládat data, musela být zvolena a nainstalována testovací databáze.
- Důležitým cílem bylo na server postupně instalovat jednotlivé frameworky a v nich vytvořit jednotlivé aplikace. U každého projektu bylo také zapotřebí pochopit jeho strukturu a funkcionalitu potřebnou pro bezchybný chod vlastní aplikace.
- Dalším cílem bylo vytvořit testovací procedury ve vybraném programu řízené navrženou testovací metodikou.
- Poslední bod se zabýval vyhodnocením naměřených údajů a jejich analýzou. Celé porovnání bylo následně publikováno v podobě článku na zvoleném webovém portálu.

4 Návrh řešení

4.1 Požadovaný výsledek testu

Porovnání je zaměřeno na rychlost práce knihovny používané při komunikaci s databází. Z toho důvodu je zapotřebí, aby každá aplikace vytvořená s pomocí jiného frameworku, a tím také využívající i jiné knihovny, byla testována za podobných nebo nejlépe stejných podmínek. Výsledky by neměly být ovlivňovány nežádoucími vlivy prostředí nebo použité technologie. Z těchto důvodů by měl test probíhat na lokální soukromé síti s možností regulace připojených uživatelů. Dále by průběh neměl být ovlivňován chodem programů nepotřebných přímo k testování. Při získávání informací z databáze by všechny výsledné aplikace měly pracovat se stejnými daty.

Z takto nastavených parametrů by měly vycházet porovnatelné údaje, ze kterých si bude možné udělat představu, který projekt je vhodný pro určitý typ zatížení. Aby bylo dosaženo takovýchto výsledků, je nezbytné, aby se tvorba, chování a testování jednotlivých aplikací řídila stejnými pravidly.

4.2 Vybraná metodika

Pro účely tohoto typu testování byla vybrána metodika zaměřující se na různé druhy zatížení jednotlivých aplikací. V každém projektu byly vytvořeny funkce, které otestovaly základní příkazy využívané při práci s databází. Tímto testováním byl framework zatížen vždy určitým typem požadavků, u kterých byla měřena rychlost jejich vykonání. Byly zde zahrnuty také kombinace jednoduchých příkazů, které se řídily strukturou využívanou v reálných aplikacích.

Takováto metodika se blíží chodu aplikace využívané v reálném provozu, přestože se stále nejedná o její úplně přesnou simulaci.

Celý proces byl složen z několika samostatných testů, které hodnotily různé vlastnosti dané knihovny. Veškeré jednotlivé kroky byly opakovaně provedeny a v jejich průběhu zaznamenávány průběžné výsledky, které byly v závěru zprůměrovány a vyhodnoceny.

4.2.1 Struktura příkazů

Práce reálné aplikace s databází se zakládá na čtyřech základních operacích a všechny ostatní příkazy jsou pouze rozšířením nebo zkombinováním těchto metod.

Jedná se o takzvané CRUD operace. Tato zkratka vznikla z anglického „Create, Read, Update, Delete“, což znamená, že aplikace něco vytváří, čte, upravuje a maže. V tomto souhrnném testu bylo porovnáváno chování všech těchto databázových příkazů v pěti oddělených testech. Každý test byl zaměřen na srovnání rychlosti odezvy jednotlivých aplikací s použitím některé z těchto funkcí.

Při vyhledávání v databázi za účelem zobrazení nějakého seznamu záznamů, je předpokládáno, že se ve skupině nemusí objevovat ani jeden řádek. Z tohoto důvodu byla při testování funkce Select, neboli při jednoduchém výpisu dat z databáze, využita pouze tato samotná funkce. V tomto porovnávání byla aplikací poskytována pouze data získaná z databáze, která byla vypsána v podobě seznamu. Takováto zátěž se dá porovnat s reálnou aplikací, která běžnému uživateli například pouze vypisuje televizní program, který nějaký správce uložil do databáze.

Naopak, pokud má aplikace upravit nebo odstranit nějaký údaj, který se nachází v databázi, musí nejdříve zkontrolovat, zda opravdu existuje. Z tohoto důvodu bylo nezbytné nejprve ověřit jeho existenci a až poté ho bylo možné upravit. Tento test simuloval zátěž jednotlivých aplikací při editaci a mazání údajů, např. o uživatelích nebo příspěvcích. Takovýto proces probíhá například v každé aplikaci, která obsahuje možnost vytvoření uživatelského účtu s nějakými podrobnostmi a jejich pozdější editaci.

Při vkládání nových záznamů do databáze bylo využito jedinečnosti generovaného identifikátoru, díky kterému mohlo být otestováno přidávání údajů bez vnitřní kontroly aplikací. Tento způsob byl vybrán z důvodu otestování vkládání údajů bez nezbytné kontroly, ale protože data byla využívána v dalších testech, musela obsahovat specifický identifikátor. S tímto typem zátěže se lze setkat například u vkládání jednotlivých komentářů k článku, kde není provedena kontrola duplicity příspěvku. Za takovýchto podmínek uživatel vytvoří nějaký záznam a pouze ho odešle do databáze. V tomto testu bylo využito přidávání nových položek do seznamu herců.

V posledním testu byly otestovány všechny funkce zároveň, přičemž byly zohledněny veškeré předchozí poznatky. Při takovéto konfiguraci docházelo k paralelnímu zadávání různých příkazů od jednotlivých uživatelů. Při tom byly měřeny časové úseky, za jaké dokázala aplikace reagovat na zasílané požadavky. Tento test se

svou strukturou blíží k funkcionalitě reálné aplikace, ve které se obvykle neřeší pouze jednoduché příkazy.

4.2.2 Definice zátěže

Webová stránka dostupná veřejnosti je velmi zřídka zatížena požadavky pouze jednoho návštěvníka. V reálném provozu veškeré stránky neustále procházejí nejen reální uživatelé, ale také roboti získávající informace například pro internetové vyhledávače. Z toho důvodu byly veškeré testy zaměřeny na víceuživatelskou zátěž každé aplikace. Avšak z důvodu omezeného výkonu dostupné výpočetní techniky byly podmínky nastaveny na 3000 požadavků v rámci jednoho testu. Rozložení této zátěže bylo rozděleno jako 100 jednotlivých příkazů pro 30 virtuálních uživatelů. Při závěrečném testu byly prováděny všechny čtyři operace zároveň, a tudíž výsledný počet získaných vzorků byl 12 000.

4.2.3 Omezení nežádoucích činitelů

Aby byly naměřené výsledky co možná nejpřesnější a daly se porovnávat, bylo nezbytné odstranit vliv vnějšího prostředí. Toho bylo dosaženo testováním na lokální síti, která byla izolována od veřejného provozu, a díky tomu nebyla ovlivňována činností ostatních uživatelů. Zároveň byl pro server vybrán operační systém Ubuntu server, který neobsahuje téměř žádné doplňkové služby, které by se samovolně spouštěly a měly by tak negativní dopad na jeho výkon. Na notebooku s testovacím programem byly vypnuty veškeré aplikace nepotřebné k chodu vlastního programu a byly zakázány automaticky spouštěné funkce systému.

Jednotlivé komponenty a projekty byly staženy téměř ve stejnou dobu a v průběhu jejich využívání nedocházelo k žádným aktualizacím. Po provedení celé série měření byly server i testovací notebook vypnuty, aby nedocházelo k ukládání nepotřebných dat zbytečně využívajících paměť počítače. Všechny testy byly provedeny vždy postupně za sebou, a to i v různém pořadí, aby bylo ověřeno, že jejich pořadí nemá žádný významný dopad na jejich výsledky.

4.3 Vlastnosti všech vytvářených aplikací

Veškeré testované aplikace byly upravovány tak, aby měly alespoň podobnou strukturu. Každý využívaný framework využívá jiné nástroje a pomocné funkce, které mohou mít negativní dopad na rychlost odezvy celé aplikace.

4.3.1 Struktura a vzhled

Pro získání srovnatelných výsledků bylo zapotřebí upravit každou aplikaci tak, aby nevyužívala příliš mnoho doplňkových funkcí. U některých projektů muselo být omezeno, nebo úplně odstraněno, automaticky generované grafické prostředí, či musely být odebrány funkce vyžadující přihlášení. Protože tento test byl zaměřen výhradně na rychlost odezvy při práci s databází, byly některé funkce upraveny tak, aby nevyužívaly nepotřebný kód a zároveň, aby se neporušila struktura a logika využívání databázových funkcí.

4.3.2 Konzistence dat

Aby nedocházelo k přepisování zdrojových souborů jednotlivých projektů, byl každý framework instalován do speciální složky, ve které byly upravovány a tvořeny zdrojové soubory pro výslednou aplikaci. Zároveň všechny aplikace pracovaly pouze se svými daty a nevyužívaly žádné sdílené složky dostupné na serveru. Aby bylo zamezeno kolizi dat v databázi, byl každý test proveden až do fáze mazání vytvořených záznamů a následně byla databáze zkontrolována, zda vše proběhlo úspěšně.

5 Realizace řešení

5.1 Výběr vhodných nástrojů

Po rešeršní práci popsané v kapitole 2.5, bylo pro porovnávání vybráno čtrnáct různých frameworků, které posloužily k naprogramování jednotlivých aplikací. Na závěr byl do testu zahrnut i projekt napsaný v jazyce PHP, který posloužil pro srovnání rychlosti odezvy bez využívání doplňkových knihoven nebo příliš komplikované struktury jádra.

5.1.1 Server

Jako server byl zvolen takzvaný LAMP server, což je zkratka definující nástroje využívané na serveru, přesněji Linuxový operační systém, Apache HTTP server, MySQL databázový systém a PHP, jakožto podporovaný skriptovací programovací jazyk. Lze využít i rozdílnou konfiguraci, například jakýkoliv jiný podporovaný jazyk, ale označení zůstává stejné.

Linuxovým operačním systémem byl zvolen Ubuntu server. Jeho jádro je navrženo tak, aby se největší množství výkonu počítače zabývalo výhradně obsluhou serverových částí. Celý tento systém lze rozšířit o různé doplňky, které mohou například usnadnit jeho správu.

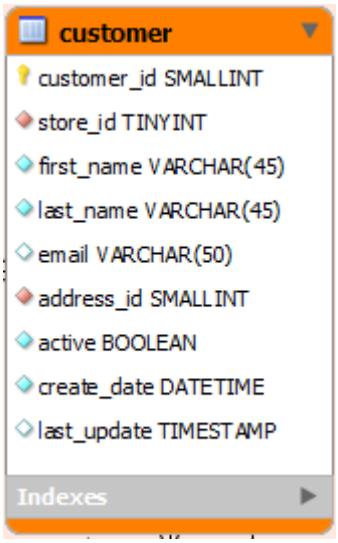
Apache HTTP server byl nainstalován ve verzi 2.2.22. Velká výhoda tohoto serveru je jeho multiplatformní rozšíření, díky kterému může být použit na všech nejčastěji používaných operačních systémech. Celá jeho funkcionalita je řízena rozsáhlými možnostmi konfigurace umožňujícími nadefinovat si server do požadované podoby.

MySQL databázový systém je víceuživatelský systém s podporou více vláken využívaný pro správu databázových záznamů, řízených pomocí SQL příkazů. Patří mezi volně dostupné nástroje, i když lze zakoupit i verzi s placenou licencí, a lze ho využít na většině, v současné době dostupných operačních systémech.

PHP je programovací skriptovací jazyk určený zejména pro tvorbu dynamických webových stránek. Ve své kompilované podobě jde i využít pro tvorbu desktopových aplikací. Využívá objektově orientované programování, což umožňuje návrh složitějších struktur celé aplikace. Velice užitečné je i dynamické typování jednotlivých proměnných a jejich automatická deklarace.

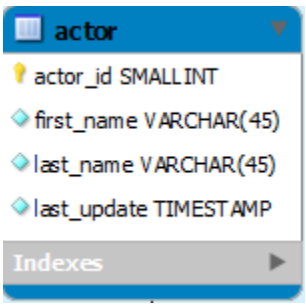
5.1.2 Databáze

Z důvodu možného pozdějšího rozšíření testů byla vybrána databáze Sakila, která obsahuje strukturu využívanou například v půjčovnách filmů. Tato databáze byla vytvořena programátorem pracujícím na dokumentaci k MySQL a je určená právě jako zdroj dat pro testování webových aplikací. Celé její schéma je uvedeno v příloze této práce, obrázek je čerpán z portálu Quanti [29] věnovanému databázím. V její struktuře jsou obsaženy téměř všechny funkce, které může databáze poskytovat. V tomto testu byly zahrnuty pouze dvě dostupné tabulky, na kterých byly prováděny požadované operace. Tabulka customer(Obr. 8), neboli záznam všech zákazníků, s pěti sty záznamy sloužila pro výpis. A tabulka actor(Obr. 9), neboli databáze dostupných herců, byla využita pro vkládání nových dat, jejich úpravu a mazání. Celkové schéma struktury databáze je zahrnuto v příloze (Příloha B).



Obr. 8: Schéma tabulky zákazníků

Zdroj:Quanti.cz [29] , výstřižek z celkového schématu



Obr. 9: Schéma tabulky herců

Zdroj: Quanti.cz[29] , výstřižek z celkového schématu

5.1.3 Testovací program

Mezi nejdůležitější nástroje používané při takovýchto testech je právě program generující jednotlivé příkazy posílané dané aplikaci. Pro potřeby tohoto porovnávání byl, na základě předešlé rešeršní práce popsané v kapitole 2.4, vybrán Apache JMeter. Tento užitečný nástroj, určený nejen na zátěžové testování webových aplikací, obsahuje široké spektrum funkcí využitelných pro zkoumání všech možných vlastností aplikace. V průběhu testu byl využit nejen jako generátor požadavků, ale v několika případech i jako monitorovací systém pro upřesnění chování určitých stránek.

5.2 Rozložení výpočetního výkonu

Pro účely tohoto testu byly využity dva počítače vzájemně propojené lokální bezdrátovou sítí, aby výkon potřebný pro práci testovacího programu neměl negativní dopad na rychlost serveru.

Stolní počítač se serverem

- Procesor: Intel(R) Core(TM)2 Duo E8400, 3.00 GHz
- Paměť: 8.0 GB DDR2
- Pevný disk: 5400 RMP, SATA
- Operační systém: Ubuntu Server 12.04.1 LTS x86

Notebook s testovacím programem

- Typ: ASUS UL80VT
- Procesor: Genuine Intel(R) CPU U7300, 1.3GHz
- Paměť: 4.0 GB DDR3
- Pevný disk: 5400 RMP, SATA
- Operační systém: Microsoft Windows 7 Professional x64, Service Pack 1

5.3 Jednotlivé frameworky

Na základě rešerše uvedené v kapitole 2.5 bylo pro testování vybráno 14 různých frameworků. Pro účely porovnání byly jednotlivé projekty stahovány v krátkém časovém intervalu, a až poté byly s jejich pomocí tvořeny jednotlivé aplikace. Každá aplikace byla vytvářena pomocí poslední dostupné stabilní verze frameworku. Tento postup byl použit z důvodu neustálého vývoje jednotlivých frameworků, který by mohl mít negativní vliv na výsledky aplikací vytvořených jako poslední.

5.3.1 Seznam verzí

- CakePHP 2.2.5, vydán 8. 1. 2013
- CodeIgniter 2.1.3, vydán 8. 10. 2012
- DooPHP 1.4.1, vydán 22. 2. 2011
- Jelix 1.5.0, vydán 20. 2. 2013
- Kohana 3.3.0, vydán 21. 10. 2012
- Laravel 3.2.13, vydán 11. 1. 2013
- Nette 2.0.8, vydán 19. 2. 2013
- Prado 3.2.1, vydán 19. 1. 2013
- Qcodo 0.4.22, vydán 15. 8. 2011
- Recess 0.20, vydán 7. 8. 2009
- Seagull 1.0.4, vydán 14. 1. 2013
- Symfony 2.2.0, vydán 24. 2. 2013
- Yii 1.1.13, vydán 30. 12. 2012
- Zend framework 2.1.4, vydán 13. 3. 2013

5.4 Podrobný průběh testu

Před tvorbou vlastních aplikací musel být nainstalován a správně nakonfigurován webový server, na kterém se ukládala veškerá data. Pro tyto účely byl vybrán linuxový Ubuntu server ve verzi 12.04.1 LTS x86, který je určen výhradně pro serverové účely. Díky tomuto zaměření se jeho jádro zabývá pouze zpracováváním údajů potřebných pro chod serveru a nejsou spouštěny zbytečné aplikace, které by ubíraly jeho výkon. Poskytuje výstup pouze ve formě příkazového řádku, který lze přes různé nástroje spravovat i z jiných počítačů. Velmi užitečnou vlastností použitého operačního systému je šifrování všech serverových souborů uložených na harddisku, které jsou k dispozici pouze po přihlášení do uživatelského účtu vytvořeného při instalaci.

5.4.1 Serverové doplňky

Při instalaci byla využita varianta LAMP serveru, která má v sobě obsažené balíčky pro jednoduchou instalaci doplňujících služeb. Tímto postupem se nakonfiguroval Apache HTTP server ve verzi 2.2.22 zajišťující zpracovávání požadavků jednotlivých uživatelů, podávaných prostřednictvím webových aplikací. Dalším důležitým prvkem bylo nastavení MySQL klienta ve verzi 5.5.24, který obstarává ukládání databázových

údajů. Pro podporu programovacího skriptovacího jazyka byl nainstalován modul PHP ve verzi 5.3.10.

Aby vše nemuselo být řízeno pouze z příkazového řádku, byla nainstalována i aplikace phpMyAdmin 3.4.10, která slouží jako grafické uživatelské rozhraní pro práci s databází. Pomocí tohoto nástroje byla implementována struktura databáze Sakila a následně do ní byly, pomocí oficiálního skriptu, vloženy generované údaje. K tvorbě jednotlivých aplikací byl použit textový editor PSPad, připojený z druhého počítače přes lokální síť.

5.4.2 Tvorba jednotlivých aplikací

Jakmile byl takto nainstalován a nakonfigurován celý server i s jednotlivými moduly, byla v jeho souborovém systému vytvořena složka „/www“, do které byly vkládány jednotlivé projekty. Každá aplikace byla vnořena do vlastní podsložky obsahující veškeré její kódy, aby nedocházelo k přepisování mezi jednotlivými projekty.

Tvorba každé aplikace začala stažením požadovaného frameworku v poslední dostupné stabilní verzi z oficiálních stránek. Tento archiv byl následně rozbalen do předem vytvořené složky a tím byla zpřístupněna jeho celá funkcionalita. U většiny projektů bylo nezbytné, aby byla provedena konfigurace jednotlivých doporučených částí zobrazených při připojení na lokální domovskou stránku aplikace, zpravidla „index.php“. V tomto zobrazení byly získány údaje, jaké nastavení a minimální požadavky framework vyžaduje nebo byla aplikace přímo přesměrována na nástroje, které po zadání klíčových údajů provedly vše automaticky. Pokud bylo všechno správně nastaveno, bylo možno přistoupit k tvorbě požadované aplikace.

V některých projektech byly k dispozici nástroje podporující okamžité generování již hotové aplikace komunikující se zadanou databází. Výsledné stránky byly sice vytvořeny během několika minut, ale o to složitější byla jejich úprava do požadované podoby. To bylo způsobeno příliš velkým počtem nepotřebných vlastností nebo složitého grafického prostředí zpomalujícího chod celé aplikace.

Ve všech projektech byl využíván zejména programovací skriptovací jazyk PHP. Pro pochopení některých funkcí, nebo jejich vlastností, které nebyly vytvořeny ve frameworku, ale týkaly se přímo jazyka PHP, byla využita online dokumentace [30] tohoto jazyka nebo příklady z knihy 1001 tipů a triků pro PHP [1].

- Funkce pro zobrazení

Pro první část testu musela každá aplikace obsahovat co nejjednodušší výpis všech záznamů, které byly získány z databáze Sakila z tabulky uživatelů (customer). Některé kódy byly samy generovány a obsahovaly například automatické stránkování při výpisech delších než 20 záznamů. Tyto nástroje musely být odstraněny, aby bylo možné porovnat výsledky s ostatními projekty, které tuto vlastnost neobsahují.

Na obrázku (Obr. 10) je vidět ukázka části kódu z frameworku Kohana, který plní funkci kontroleru. Tento kód byl použit na zavolání funkce jádra pro získání dat z databáze, která byla následně uložena do globální proměnné. Tato data byla předána další části vzoru MVC a to pohledu, který je vypsal dle definované struktury (Obr. 11).Takto zpracovaná data byla zobrazena jako jednoduchý seznam, ve kterém bylo obsaženo identifikační číslo, jméno a příjmení každého zákazníka.

```
public function action_list() {
    $customers = ORM::factory('Customer')->find_all();
    $view = new View('sakila/list');
    $view->set("customers", $customers);
    $this->response->body($view);
}
```

Obr. 10: Funkce kontroleru z frameworku Kohana

```
<?php foreach ($customers as $customer) : ?>
<div>
    <b><?php echo $customer->customer_id; ?></b>
    <?php echo $customer->first_name; ?>
    <?php echo $customer->last_name; ?>
</div>
<?php endforeach; ?>
```

Obr. 11: Kód pohledu z frameworku Kohana

- Funkce pro vkládání

Pro druhou část testu byly aplikaci zadány nové generované údaje, které byly vyplněny ve formuláři a následně zapsány do databáze. Jednalo se o záznamy v tabulce „actor“, kde byli vytvářeni noví herci s unikátními informacemi. Na obrázku (Obr. 12) je možné vidět část kódu využívanou ve frameworku Symfony 2, starající se o generování formuláře pomocí nástroje FormBuilder.

```
public function buildForm(FormBuilderInterface $builder, array $options)
{
    $builder
        ->add('actor_id')
        ->add('first_name')
        ->add('last_name')
    ;
}
```

Obr. 12: Funkce vytvářející formulář v Symfony 2

V následující ukázce (Obr. 13) byly zpracovány údaje zasílané po vyplnění a odeslání tohoto formuláře. Jedná se o kontroler frameworku Symfony 2, ve kterém je znázorněno využívání objektového programování, zasílání dat do databáze a následné přesměrování na stránku s hláškou, zda byl proces úspěšný. Metody POST jsou zde z důvodu kontroly vstupních údajů zasílané testovacím programem.

```
public function createAction()
{
    $actor  = new Actor();
    $request = $this->getRequest();
    $form    = $this->createForm(new ActorType(), $actor);
    $form->bindRequest($request);
    if(isset($_POST['actor_id'])){
      $actor->setActorId($_POST['actor_id']);
      $actor->setFirstName($_POST['first_name']);
      $actor->setLastName($_POST['last_name']);
    }
    $em = $this->getDoctrine()
               ->getEntityManager();
    $em->persist($actor);
    $em->flush();
    return $this->redirect($this->generateUrl('my_sakila_actor_success'));
}
```

Obr. 13: Funkce vytvářející nový záznam v databázi

- Funkce pro upravení

Pro třetí část testu byla využívána část aplikace podobná jako ve druhém kroku, ale měla za úkol upravovat záznamy v databázi. Byl vypsán formulář, do kterého byly zadány údaje specifikující záznam v databázi na základě identifikačního čísla, ten byl následně odeslán a předán funkci na uložení. V tomto zpracování byla obsažena kontrola, zda se v databázi nachází záznam definovaný tímto identifikátorem. Pokud ano, pak byla jeho data upravena dle údajů zasílaných programem a následně byla přesměrována na stránku s oznámením, zda vše proběhlo úspěšně. V následující

ukázce (Obr. 14) je znázorněna část kódu z kontroleru frameworku Zend. V tomto projektu bylo využito výjimky při kontrole, zda jsou data dostupná v databázi. Následně byl vypsán formulář, ve kterém byly údaje upravovány.

```
try {
    $actor = $this->getActorTable()->getActor($actor_id);
}
catch (\Exception $ex) {
    return $this->redirect()->toRoute('actor', array(
        'action' => 'index'
    ));
}
$form  = new ActorForm();
$form->bind($actor);
$form->get('submit')->setAttribute('value', 'Edit');
$request = $this->getRequest();
if ($request->isPost()) {
    $form->setInputFilter($actor->getInputFilter());
    $form->setData($request->getPost());
    if ($form->isValid()) {
        $this->getActorTable()->saveActor($form->getData());
        return $this->redirect()->toRoute('actor',
                        array('action' => 'success'));
    }
}
return array('actor_id' => $actor_id,'form' => $form,);
```

Obr. 14: Funkce pro kontrolu a úpravu údajů

- Funkce na odstranění

Pro čtvrtou část testu byla vytvořena poslední funkce aplikace, při které byl vypsán formulář, do něhož bylo zadáno identifikační číslo záznamu (herce), a tato data byla následně odstraněna z databáze. Byla zde znovu zahrnuta kontrola, zda se údaje v databázi nacházejí či nikoli, a po úspěšném nalezení byly odstraněny. Po vykonání příkazu byl uživatel přesměrován na stránku obsahující zprávu o úspěšném odstranění záznamu. Na ukázce (Obr. 15) je znázorněna část kódu vykonávající mazání záznamů ve frameworku CakePHP.

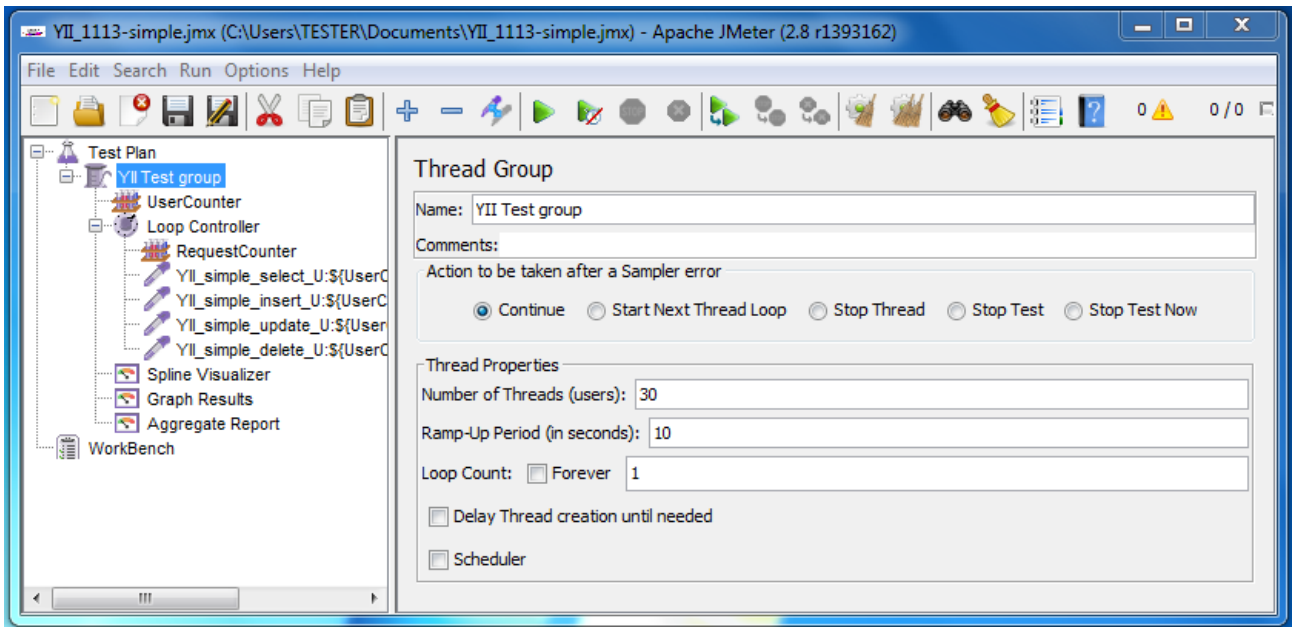
```
public function delete($actor_id) {
  if ($this->request->is('get')) {
      throw new MethodNotAllowedException();
  }

  $actor = $this->Actor->find('first', array('conditions' =>
                              array('actor_id' => $actor_id)));
  if (!$actor) {
    throw new NotFoundException(__('Invalid post'));
  }
  if ($this->Actor->delete($actor_id)) {
      $this->Session->setFlash('deleted');
      $this->redirect(array('action' => 'success'));
  }
}
```

Obr. 15: Funkce pro kontrolu a mazání záznamu

5.4.3 Testování vytvořených aplikací

Pro simulaci víceuživatelské zátěže byl použit program Apache JMeter 2.8 určený právě na testování webových aplikací. Tento užitečný nástroj disponuje velkým množstvím testovacích funkcí, jejichž výsledky lze zobrazovat pomocí zabudovaných nástrojů. Na obrázku (Obr. 16) je znázorněno prostředí Apache JMeter obsahující nastavení počtu uživatelů a takzvané „Ramp-Up Periody“, která definuje, za jak dlouho se jednotliví uživatelé připojí. Každá aplikace byla testována při zátěži tvořenou třiceti virtuálními uživateli, kde všichni zadávali 100 příkazů. Při zadané periodě přihlašování se každou sekundu připojili tři noví uživatelé. Toto nastavení bylo vybráno jako nejstabilnější varianta při optimální zátěži.



Obr. 16: Apache JMeter nastavení uživatelů

Na ukázce z grafického rozhraní testovacího programu (Obr. 16) lze vidět strukturu, dle které byl test řízen. V kmenové složce testovacího plánu (Test Plan) byly obsaženy jednotlivé nástroje využívané pro test daného frameworku. Byla zde zahrnuta počítadla, která znázorňovala číslo uživatele (UserCounter) a pořadí aktuálně prováděného příkazu (RequestCounter). Tyto hodnoty byly využity jak pro získání lepší orientace ve výsledcích, tak pro generování zasílaných dat. Na všech aplikacích bylo provedeno 5 testů. Každý byl vykonán třikrát po dokončení měření u všech projektů. Jednotlivé příkazy daného uživatele byly zahrnuty v cyklu (Loop Controller), který řídil jejich opakování v aktuálním testu. Tyto požadavky byly zasílány funkcí HTTP request definující testovanou adresu a případná zasílaná data. Veškeré naměřené hodnoty byly zobrazovány v tabulce (Aggregate Report) a v grafickém znázornění průběhu testu (Graph Results).

- Test výpisu údajů

V prvním testu byl zasílán požadavek na zobrazení kompletního seznamu zákazníků uložených v databázi, kde ke každému bylo vypsáno jeho identifikační číslo, jméno a příjmení. Tento příkaz byl zaslán stokrát od každého ze skupiny všech třiceti uživatelů, a tudíž tímto měřením bylo zasláno 3000 žádostí na vypsání.

- Test vkládání údajů

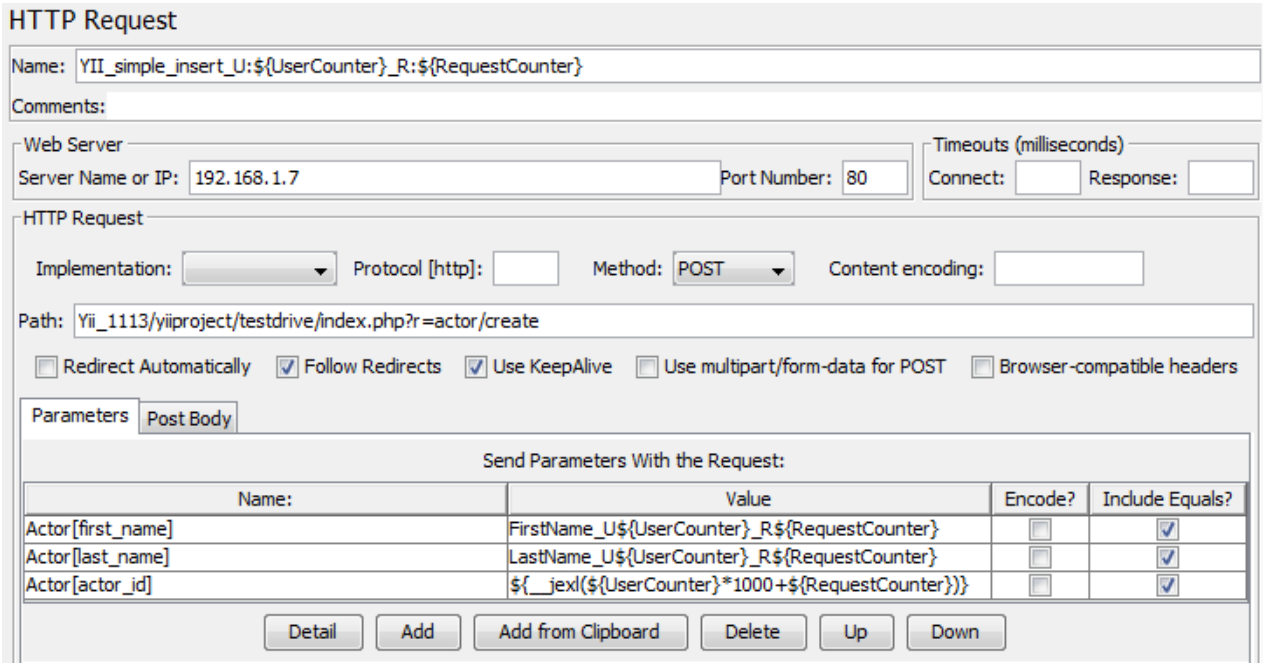
V druhém testu byla volána stránka obsahující formulář pro vložení nového záznamu do databáze herců. Programem Apache JMeter byla společně s příkazem zaslána i vygenerovaná data, která do něj byla automaticky vepsána, a celý formulář byl následně odeslán. Na obrázku (Obr. 17) je znázorněn výstřižek z funkce HTTP Request obsahující zasílané parametry. V položce „Name“ bylo zadáno identifikační jméno příkazu využívajícího počítadla uživatelů a příkazů. Následně byla vyplněna IP adresa lokálního serveru a URL odkazující přímo na stránku webové aplikace obsahující daný formulář.

Důležitou částí tohoto kroku byla zasílaná data pomocí metody POST obsahující vygenerované parametry. Aby nedocházelo k zadávání stejných údajů, byly vkládané údaje generovány na základě čísla uživatele definovaného počítadlem uživatelů (UserCounter) a pořadí aktuálně prováděného kroku (RequestCounter). Identifikační číslo záznamu v databázi(actor_id) bylo generováno pomocí knihovny pro matematické

funkce obsažené v programu Apache JMeter. Pro 30 uživatelů u každého se 100 vlastními cykly bylo tedy vygenerováné 3000 různých záznamů.

Příklad: 12. uživatel ve svém 25. požadavku vložil do tabulky údaje:

- Identifikační číslo (actor_id) = 12025.
- Jméno (first_name) = FirstName_U12_R25.
- Příjmení (last_name) = LastName_U12_R25.



Obr. 17: Apache JMeter nastavení parametrů požadavku

- Test úpravy údajů

Ve třetím testu byly zasílány požadavky na změnu údajů. Aplikace zkontrolovala, zda se v databázi nacházejí záznamy určené zasílaným identifikátorem. Pokud ano, pak byly uživateli zobrazeny ve formuláři, který byl automaticky přepsán novými hodnotami a následně odeslán zpět do databáze. Tento krok byl z pohledu programu Apache JMeter podobný jako test předcházející, jen se změnou adresy požadované stránky a zasílaných údajů. V tomto kroku bylo ke jménu a příjmení herce přidáno pouze klíčové slovo „New“, aby byly záznamy v databázi lehce rozeznatelné. Na konci tohoto testu bylo upraveno všech 100 položek u každého ze skupiny třiceti uživatelů.

Příklad: 10. uživatel ve své 65. žádosti upravil údaje nacházející se v tabulce herců pod identifikačním číslem (actor_id) 10065 na:

- Jméno (first_name) = FirstNameNew_U10_R65.
- Příjmení (last_name) = LastNameNew_U10_R65.

- Test odstranění údajů

Čtvrtý test byl zaměřen pouze na odstranění aplikací vytvořených a následně upravených položek. Do zobrazeného formuláře bylo předáno jen identifikační číslo záznamu určeného k odstranění. To bylo zkontrolováno aplikací, zda se takový záznam nachází v databázi a pokud byla kontrola úspěšná, došlo k odstranění celého řádku obsahujícího upravené údaje. Tato operace byla provedena u všech třiceti uživatelů v každém kroku jejich cyklu. Na konci tohoto testu bylo z databáze vymazáno všech 3000 herců vytvořených v průběhu testování jednoho frameworku.

Příklad: 29. uživatel ve svém 2. kroku odstranil údaje zapsané v tabulce herců na řádku s identifikačním číslem (actor_id) 29002.

- Test všech funkcí

Poslední test byl proveden jako simulace provozu aplikace, kde jeden uživatel může dělat jinou operaci než druhý. V tomto porovnání byly aktivní všechny čtyři předchozí testy a každý uživatel je prováděl vždy v jednom svém cyklu. Z důvodu postupného přidávání nových návštěvníků a různé reakční doby aplikace na jejich požadavky, bylo otestováno chování frameworku při 4krát větší zátěži než předtím a s paralelními požadavky různého typu. V tomto testu všech 30 virtuálních uživatelů v každém svém kroku postupně vypisovalo, vkládalo, upravovalo a mazalo údaje z databáze. Toto měření tedy obsahovalo celkem 12 000 požadavků.

Příklad: pro uživatele s pořadovým číslem 15, ve svém 26. kroku:

- Byl vypsán seznam zákazníků z tabulky „customer“.
- Byl vytvořen záznam v tabulce herců s identifikátorem (actor_id) „15026“, jménem (first_name) „FirstName_U15_R26“ a příjmením (last_name) „LastName_U15_R26“.

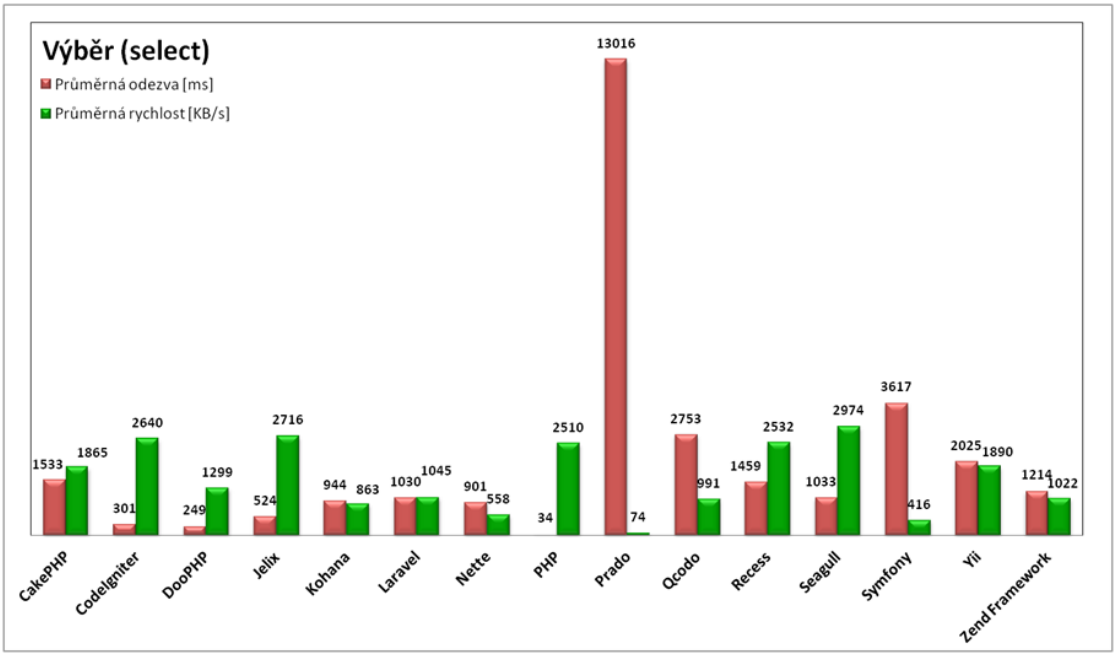
- Byla zkontrolována existence dat v tabulce, která byla následně upravena na: jméno (first_name) „FirstNameNew_U15_R26“ a příjmení (last_name) „LastNameNew_U15_R26“.
- Znovu bylo zkontrolováno, zda jsou k dispozici záznamy s daným identifikačním číslem, a pokud ano, pak byl celý řádek odstraněn.

6 Vyhodnocení výsledků

V každém testu byly pomocí programu Apache JMeter měřeny různé údaje získané monitorováním reakcí aplikace na zadávané příkazy. Porovnávání jednotlivých projektů bylo prováděno na základě získaných hodnot znázorňujících rychlost odezvy aplikace. Veškeré testy byly provedeny třikrát, a to vždy po dokončení celé série obsahující všech 15 projektů. Po dokončení všech tří skupin byly výsledky zprůměrovány a následně zavedeny do vlastních tabulek. Pro přehlednější orientaci byly zobrazeny v podobě grafů obsahujících jen nejdůležitější hodnoty. Na všech grafech byly využity naměřené hodnoty pro průměrnou rychlost odezvy (červené sloupce) v milisekundách a průměrnou rychlost komunikace s aplikací udávanou v KB/s (zelené sloupce).

6.1 Test výběru

V prvním testu byla měřena rychlost odezvy aplikace při jednoduchém výpisu seznamu uživatelů. Na grafu (Graf 1: Vyhodnocení testu výběru) jsou znázorněny výsledky tohoto měření. Lze zde pozorovat výrazný rozdíl při použití frameworku Prado, kde jeho reakční doba je několikanásobně větší než u ostatních. To může být způsobeno například velkým množstvím volaných funkcí nebo velkými objemy dat potřebných k jednoduchému výpisu.

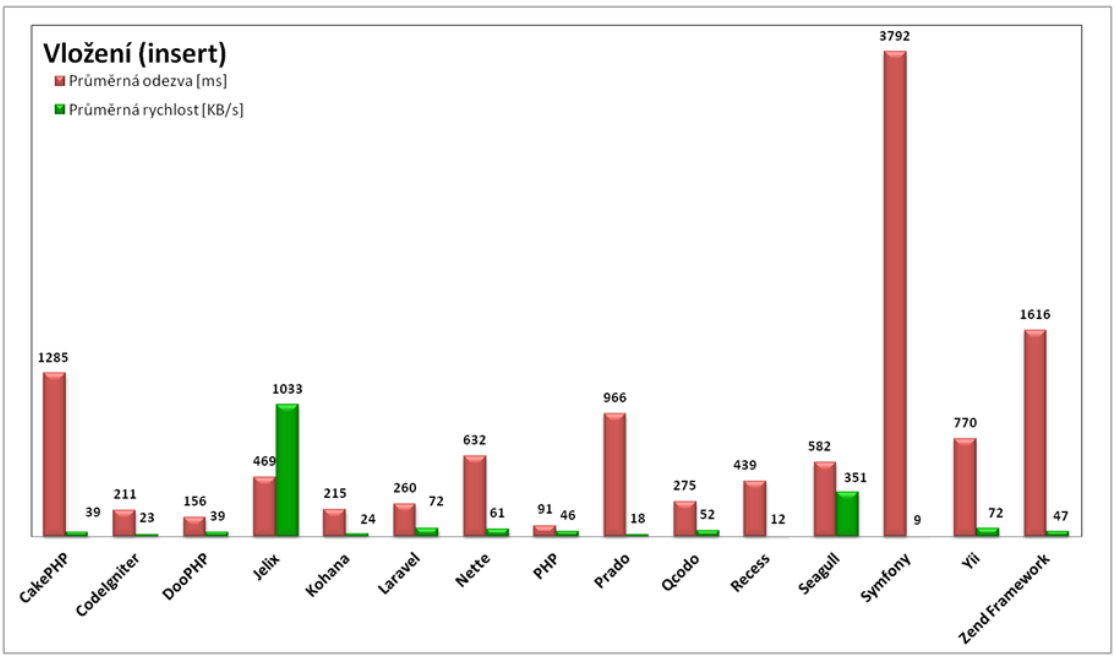


Graf 1: Vyhodnocení testu výběru

6.2 Test vložení

V druhém testu byla měřena reakční doba aplikací, které vykonávali funkci, přidávající nového herce. Při tomto příkazu byl vypisován pouze jednoduchý formulář, po jeho úspěšném odeslání a přidání dat do databáze byla zobrazena pouze zpráva, že byla data uložena. Díky takto omezené grafice a minimálnímu výpisu měla požadovaná webová stránka malou velikost a tím se snížila potřebná doba na její zobrazení.

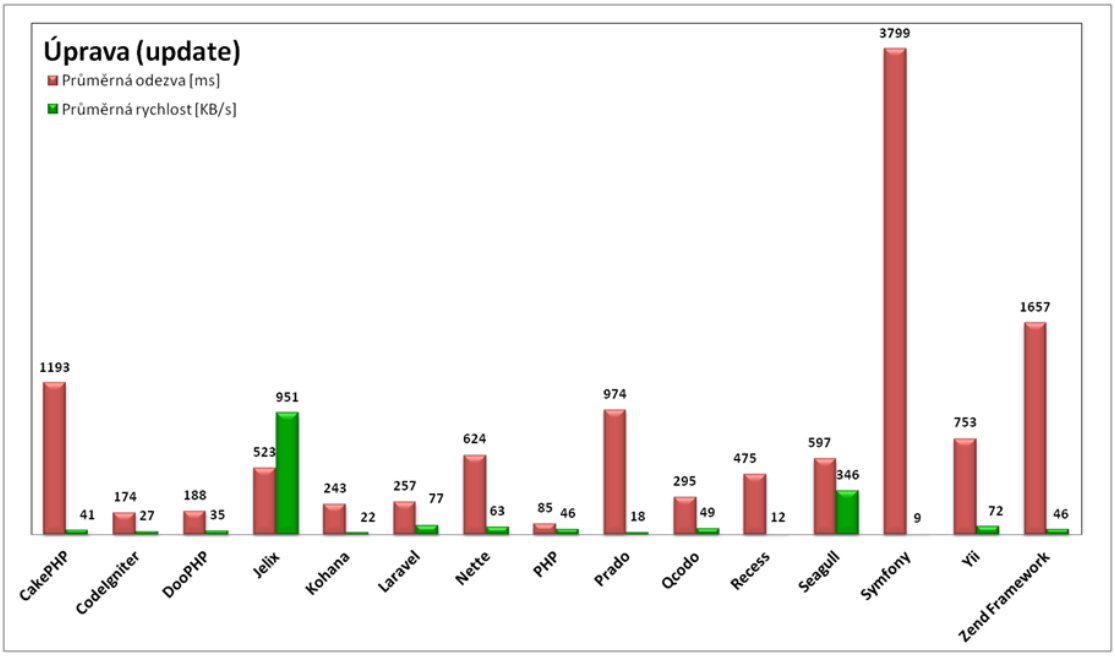
Na grafu (Graf 2) jsou znázorněny výsledky jednotlivých testovaných projektů, na kterých je zřejmé, že například při využití frameworku Symfony došlo jen k malé změně v rychlosti odezvy než třeba u aplikace s použitím frameworku Prado. To mohlo být způsobeno případně strukturou jádra, kde u Symfony je využíváno stejné množství funkcí jak pro výpis, tak pro zadávání údajů do databáze.



Graf 2: Vyhodnocení testu vložení

6.3 Test úpravy

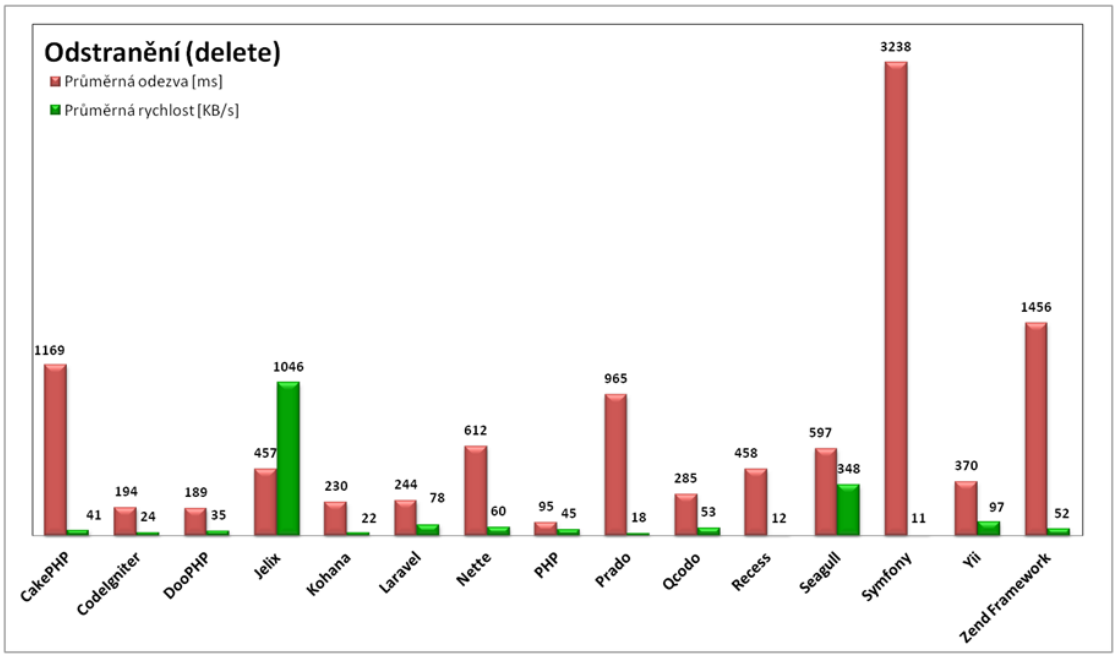
Ve třetím testu byla uživatelem vyžádána kontrola dat o záznamu herce v databázi dle vygenerovaného identifikačního čísla. Tyto údaje byly následně upraveny a odeslány zpět do databáze. Výpis nepotřebných údajů, nebo grafického prostředí, byl omezen jako v testu vkládání. Na grafu (Graf 3) jsou zobrazeny výsledky, na kterých je vidět jen nepatrný nárůst odezvy u některých projektů. To je způsobeno přidáním kontroly dostupnosti údajů.



Graf 3: Vyhodnocení testu úpravy

6.4 Test mazání

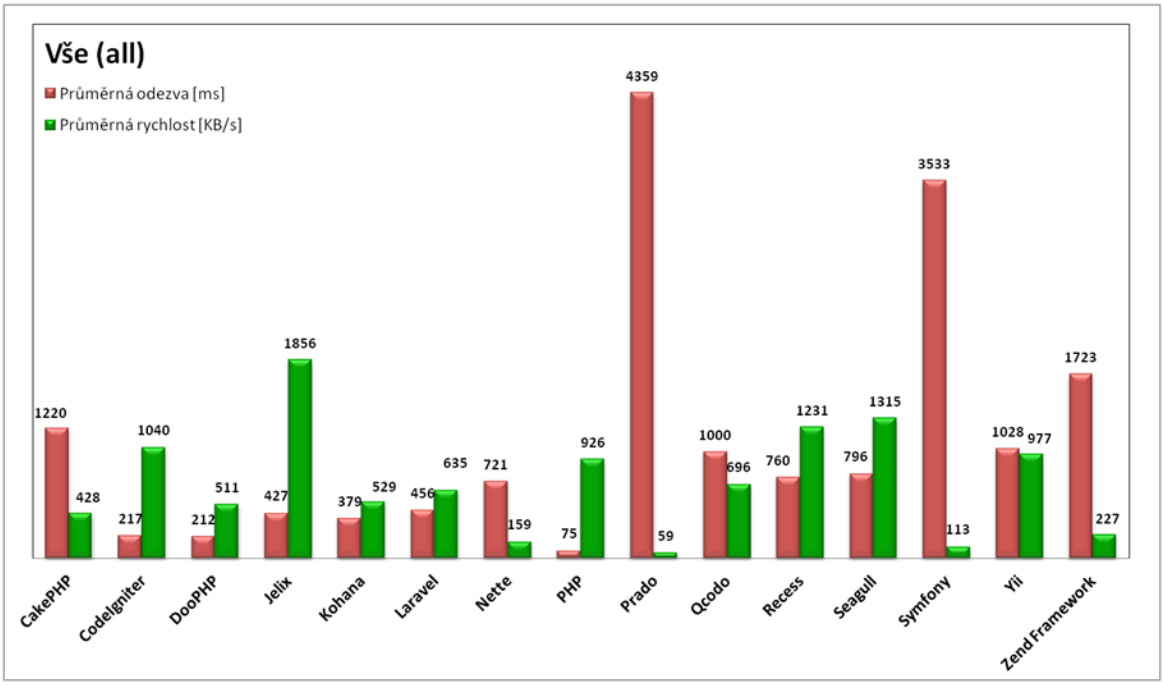
Po úspěšném upravení záznamů uložených v databázi byl proveden test rychlosti aplikace při vykonávání požadavku na odstranění určitého řádku. V každém projektu byla nejdříve provedena kontrola dostupných údajů dle identifikačního čísla zasílaného testovacím programem. Po jeho ověření byla zavolána funkce na odstranění celého záznamu. Výsledky tohoto měření byly zaznamenány do grafu (Graf 4), na kterém lze pozorovat, že rychlost při vymazávání je téměř stejná jako u úpravy záznamu.



Graf 4: Vyhodnocení testu odstranění

6.5 Souhrnný test

V posledním porovnání byly zahrnuty veškeré předcházející příkazy prováděné postupně u každého uživatele. V tomto testu byl vybranému uživateli v jednom z jeho kroků postupně vypsán seznam zákazníků, následně byla vytvořena nová položka v tabulce herců, která byla upravena a nakonec celá smazána. Tímto měřením, znázorněném v grafu (Graf 5), byly získány průměrné hodnoty rychlosti odezvy aplikace při zátěži složené z různých typů příkazů. V tomto porovnání je například patrný negativní dopad funkcí používaných pro výpis u frameworku Prado nebo velká reakční doba všech příkazů u projektu Symfony.



Graf 5: Vyhodnocení souhrnného testu

6.6 Shrnutí naměřených údajů

Z takto naměřených údajů lze vyčíst, že nejrychlejší projekt je napsaný pomocí čistého PHP. Tento výsledek byl předvídatelný, protože tato aplikace neobsahuje žádné dodatečné funkce nebo složitější strukturu, které by omezovaly její výkon.

Nejlepších výsledků dosahuje framework DooPHP. Jeho velmi rychlé zpracování požadavků je způsobeno úsporným navržením základního jádra, které je možné rozšiřovat o další funkcionalitu. Při zasílání jednoduchých příkazů pro databázi využívá pouze základní funkce a jednoduchý model, což má pozitivní vliv na celkovou rychlost. Podobnou strukturu používají frameworky Jelix nebo CodeIgniter.

Framework Kohana dosahuje také velmi dobrých výsledků, ale je stále pomalejší než CodeIgniter, od kterého je odvozena. To může být způsobeno přidáním většího množství kontrolních funkcí k základní struktuře nebo využíváním rozdílných databázových knihoven.

Nejdelší odezvy byly naměřeny u projektů vytvořených frameworky Prado a Symfony. Oba tyto projekty jsou zaměřeny spíše na rychlý vývoj robustní aplikace s velice rozsáhlou funkcionalitou. Z tohoto důvodu musejí být i pro jednoduchou webovou stránku importovány rozsáhlé knihovny funkcí, které jsou sice užitečné, ale snižují celkovou rychlost aplikace.

7 Závěr

7.1 Hlavní body jednotlivých kapitol

V kapitole zabývající se úvodem do problematiky byl jednoduše popsán princip a tvorba vlastního frameworku využitelného pro tvorbu webových aplikací. Byla zde také provedena rešerše zkoumající dostupné metodiky používané na testování takovýchto projektů. Na základě nalezených zdrojů byl zjištěn nedostatek srovnání zaměřených na zátěžové testování aplikací, které využívají funkcionalitu odpovídající reálnému provozu webových stránek. Také zde byly lehce porovnány dva oficiální nástroje využívané k těmto testům.

V návrhu řešení byly shrnuty požadované výsledky a byla vybrána metodika, kterou se řídily všechny následující testy a postupy. V těchto předpisech byly definovány struktury jednotlivých příkazů, které byly následně rozděleny do několika částí obsažených ve výsledné aplikaci. Veškerá tato funkcionalita byla navrhována dle běžně užívaných standardů na reálných webových stránkách. Byla zde také shrnuta hlavní problematika při tvoření více projektů na jednom zařízení, které pracovaly se stejnými daty. Zdůrazněn zde byl i vliv okolního prostředí a použitých technologií na celkové výsledky testů.

V hlavní části byla uvedena podrobnější realizace celého procesu testování a konfigurace potřebných nástrojů. Byly zde zjednodušeně popsány veškeré testované frameworky, u kterých bylo poukázáno na jejich základní vlastnosti, rozšíření a dostupnou podporu. V této části byl zahrnut i podrobný popis jednotlivých kroků každého prováděného testu a postup při vytváření obslužných funkcí pomocí určitého frameworku.

Při vyhodnocování výsledků byly zobrazeny jen klíčové údaje, které byly znázorněny v souhrnných grafech obsahujících průměrné výsledky naměřené u všech testovaných projektů. Tyto naměřené hodnoty byly rozděleny do jednotlivých částí, kde každá byla zaměřena na jeden test. Ve výsledcích bylo zahrnuto zhodnocení znázorněných údajů.

7.2 Zhodnocení testů

Testy vytvořené v rámci této práce obsahují spíše orientační výsledky Na základě těchto měření nelze říci, který framework je nejlepší, protože záleží hlavně na názoru a vkusu programátora. Také zde byly použity aplikace, které byly vytvořeny bez předchozích znalostí kteréhokoliv frameworku. Z tohoto důvodu nebylo možno pochopit veškeré funkce každého projektu, což může mít negativní dopad na jeho výsledky v porovnávání.

7.3 Splnění cílů

Hlavním cílem této práce bylo vytvořit test většího množství různých frameworků na základě méně používané metodiky. S tím souvisí pochopení jednotlivých principů testování, správy vlastního webového serveru a tvorba aplikací využívajících různé struktury a programovací jazyky. I přes to, že nebylo možné prostudovat každý framework do detailů, byl tento cíl splněn.

Druhým cílem bylo převést tuto práci do podoby článku a pokusit se o jejich publikaci. Přestože v současnosti je o toto téma zájem, reakční doba kontaktovaného portálu byla velice dlouhá a článek prozatím nestihl být vydán. V současné době již byly obdrženy přihlašovací údaje s autorskými právy a probíhá přepisování celého článku do požadovaného formátu.

7.4 Možnost budoucího rozšíření

Veškeré zdrojové kódy použité v této práci byly uloženy a lze je opětovně použít pro případné nové testy. Jako pokračování tohoto porovnávání by mohlo být například zahrnutí příkazů využívajících složitější strukturu, spojování tabulek a podobně.

Seznam použité literatury

[1] VRÁNA, Jakub. *1001 tipů a triků pro PHP*. Vyd. 1. Brno: Computer Press, 2010, 456 s. ISBN 978-80-251-2940-1.

[2] 32 web frameworks to choose from for your next project. *Memeburn* [online]. 29. 6. 2012 [cit. 2013-05-10]. Dostupné z: http://memeburn.com/2011/06/32-web-frameworks-to-choose-from-for-your-next-project/

[3] TOP 5 most popular PHP frameworks of 2012. *WebCoderPro* [online]. 6. 12. 2012 [cit. 2013-05-10]. Dostupné z: http://webcoderpro.com/blog/top-5-most-popular-php-frameworks-of-2012/

[4] PHP Framework benchmark: Zend, CodeIgniter & CakePHP. *Left Blank* [online]. 21. 3. 2009 [cit. 2013-05-10]. Dostupné z: http://leftblank.nl/php-framework-benchmark-zend-codeigniter-cakephp-481.html

[5] PHP Framework MVC Benchmark. *Ruilog* [online]. 1. 12. 2011 [cit. 2013-05-10]. Dostupné z: http://www.ruilog.com/blog/view/b6f0e42cf705.html

[6] PHP Framework Benchmarks. *Elefant CMS* [online]. ©2013 [cit. 2013-05-10]. Dostupné z: http://www.elefantcms.com/wiki/PHP-Framework-Benchmarks

[7] 7 PHP Frameworks Tested For Speed. Dev Shed [online]. © 2003-2013 [cit. 2013-05-10]. Dostupné z: http://www.devshed.com/c/a/PHP/7-PHP-Frameworks-Tested-For-Speed/

[8] PHP Framework Benchmark. *PHPixie* [online]. 3. 1. 2013 [cit. 2013-05-10]. Dostupné z: http://phpixie.com/blog/php-framework-benchmark/

[9] Velký test PHP frameworků. *Root.cz* [online]. 21. 8. 2008 [cit. 2013-05-10]. Dostupné z: http://www.root.cz/clanky/velky-test-php-frameworku-2008/

[10] Using Apache Bench for Simple Load Testing. *Pete Freitag's Blog* [online]. 5. 2. 2009 [cit. 2013-05-10]. Dostupné z: http://www.petefreitag.com/item/689.cfm

[11] Most Used PHP Framework-The Popular Top 7 List in year 2011. *PHP DEVELOPER* [online]. © 2009 – 2013 [cit. 2013-05-10]. Dostupné z: http://www.php-developer.org/most-used-php-framework-the-popular-top-7-list-in-year-2011/

[12] 20 Great PHP frameworks for developers. *Woork* [online]. 19. 11. 2008 [cit. 2013-05-10]. Dostupné z: http://woork.blogspot.cz/2008/11/20-great-php-framework-for-developers.html

[13] Jaké PHP frameworky zahrnout do testu?. *Devel.cz* [online]. 21.10.2012 [cit. 2013-05-10]. Dostupné z: http://devel.cz/otazka/jake-php-frameworky-zahrnou-do-testu

[14] *CakePHP: the rapid development php framework* [online]. © 2005-2013 [cit. 2013-05-10]. Dostupné z: http://cakephp.org/

[15] ELLISLAB. *CodeIgniter / EllisLab* [online]. © 2002-2013 [cit. 2013-05-10]. Dostupné z: http://ellislab.com/codeigniter

[16] *DooPHP - The fastest MVC based php framework* [online]. © 2009-2011 [cit. 2013-05-10]. Dostupné z: http://doophp.com/

[17] *Jelix, PHP framework* [online]. © 2006-2013 [cit. 2013-05-10]. Dostupné z: http://jelix.org/

[18] *Kohana: The Swift PHP Framework* [online]. © 2007-2013 [cit. 2013-05-10]. Dostupné z: http://kohanaframework.org/

[19] *Laravel - A Clean & Classy PHP Framework* [online]. © 2013 [cit. 2013-05-10]. Dostupné z: http://laravel.com/

[20] *Rychlý a pohodlný vývoj webových aplikací v PHP | Nette Framework* [online]. © 2008 - 2013 [cit. 2013-05-10]. Dostupné z: http://nette.org/cs/

[21] *Planette | Nette Framework* [online]. © 2008 - 2013 [cit. 2013-05-10]. Dostupné z: http://pla.nette.org/cs/

[22] *PRADO PHP Framework* [online]. © 2004 - 2013 [cit. 2013-05-10]. Dostupné z: http://www.pradoframework.com/

[23] *Qcodo Development Framework - Home* [online]. © 2005 - 2013 [cit. 2013-05-10]. Dostupné z: http://www.qcodo.com/

[24] *Recess PHP Framework* [online]. © 2008 - 2009 [cit. 2013-05-10]. Dostupné z: http://www.recessframework.org/

[25] *Seagull PHP Framework* [online]. © 2013 [cit. 2013-05-10]. Dostupné z: http://seagullproject.org/

[26] *High Performance PHP Framework for Web Development - Symfony* [online]. © 2004-2013 [cit. 2013-05-10]. Dostupné z: http://symfony.com/

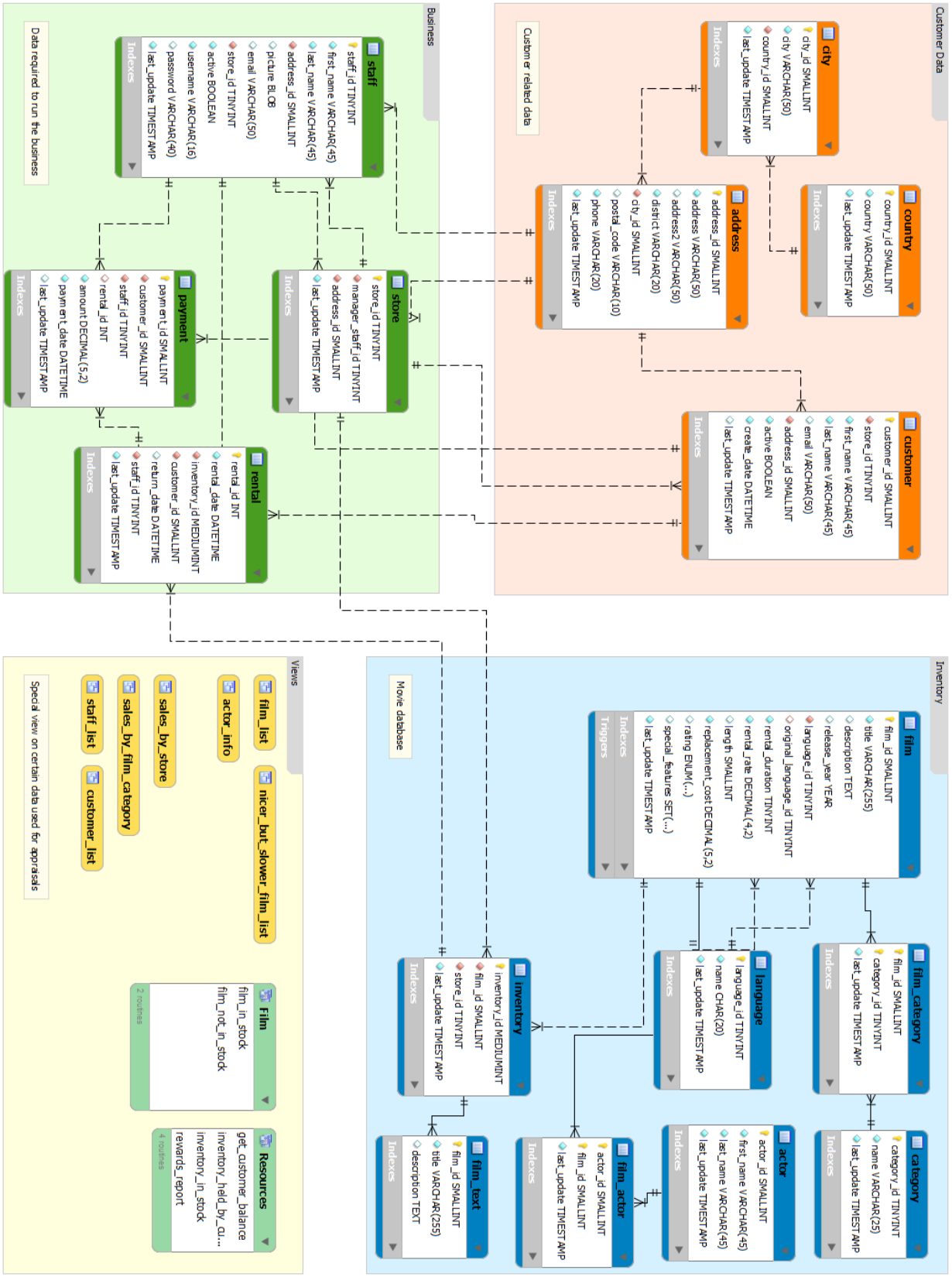
[27] *Yii Framework: Best for Web 2.0 Development* [online]. © 2013 [cit. 2013-05-10]. Dostupné z: http://www.yiiframework.com/

[28] *Zend Framework* [online]. © 2006 - 2013 [cit. 2013-05-10]. Dostupné z: http://framework.zend.com/

[29] Database schema - dbQuanti. Quanti [online]. © 2011 - 2013 [cit. 2013-05-10]. Dostupné z: http://www.db.quanti.cz/schema/

[30] PHP Manual - Manual [online]. © 1997-2013 [cit. 2013-05-10]. Dostupné z: http://www.php.net/manual/en/

Příloha B



Příloha B: Kompletní schéma struktury databáze

Zdroj:Quanti.cz [29]