TECHNICKÁ UNIVERZITA V LIBERCI
Fakulta mechatroniky, informatiky
a mezioborových studií

Framework pro dynamická rozhraní webových aplikací, emulující desktopové programování

Diplomová práce

Studijní program: N2612 – Elektrotechnika a informatika
Studijní obor: 1802T007 – Informační technologie

Autor práce: **Bc. Jakub Petržílka**
Vedoucí práce: prof. Ing. Zdeněk Plíva, Ph.D.

Liberec 2014

TECHNICAL UNIVERSITY OF LIBEREC
Faculty of Mechatronics, Informatics
and Interdisciplinary Studies

Framework for dynamical web interfaces enabling traditional approach for developement of desktop applications

Diploma thesis

Study programme: N2612 – Electrical Engineering and Informatics
Study branch: 1802T007 – Information Technology

Author: **Bc. Jakub Petržílka**
Supervisor: prof. Ing. Zdeněk Plíva, Ph.D.

Liberec 2014

Tento list nahraďte
originálem zadání.

Prohlášení

Byl jsem seznámen s tím, že na mou diplomovou 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 diplomové práce pro vnitřní potřebu TUL.

Užiji-li diplomovou 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.

Diplomovou práci jsem vypracoval samostatně s použitím uvedené literatury a na základě konzultací s vedoucím mé diplomové práce a konzultantem.

Současně čestně prohlašuji, že tištěná verze práce se shoduje s elektronickou verzí, vloženou do IS STAG.

Datum:

Podpis:

Poděkování

Děkuji ing. Jiřímu Jeníčkovi, Ph.D., Bc. Jiřímu Veselkovi a Bc. Martinu Pešovi za poskytování konzultací a sdílení zkušeností. Dále bych chtěl poděkovat Bc. Petru Vejvodovi za nemalý podíl na vývoji frameworku Phem a Bc. Janu Petřvalskému za pomoc při úpravě formální stránky této práce. V neposlední řadě patří poděkování také komunitě kolem serveru Stackoverflow.com, kde jsem vždy obdržel cenné rady a urgentní pomoc.

Abstrakt

Práce se zabývá realizací PHP frameworku pro tvorbu dynamických webových aplikací, navrženým s důrazem na vysoký výkon, přenositelnost a snadné nasazení i ve složitějších případech užití. Tento framework disponuje možnostmi a funkcemi, vhodnými zejména pro tvorbu administračních rozhraní, vyžadujících paralelní či kooperativní účast uživatelů. Umožňuje též využití tradičního deklarativního přístupu, známého například z Windows Forms či jiných desktopových vývojových platforem. První část práce věnuje pozornost zejména výběru technologií (HHMV, WebSocket, React, ZeroMQ, Ajax Comet a další) a doporučené konfiguraci běhové platformy. Dále je popisována struktura frameworku, jeho komponenty a začlenění využitých knihoven a nástrojů třetích stran (například Twig, Doctrine 2 ORM, Composer, Ratchet, JQuery, Twitter Bootstrap a další). Poslední část práce pak zahrnuje podrobný popis klíčových komponent a implementačních detailů. Několik kapitol je zde též věnováno výkonnostnímu porovnání různých konfigurací.

Klíčová slova: PHP framework, HHVM, WebSocket, realtime, push technologie, Ajax, dynamický web, PHP Forms, PHP MVC

Abstract

This thesis is dedicated to realization of PHP framework for developement of dynamical web applications focused to high throughput performance, portability and easy use even in more sophisticated use cases. It features a collection of tools and functions suitable mainly for developement of administration interfaces which requests either concurent or cooperative participation of many users at the same time. It also enables the traditional declarative approach known for example from Windows Forms and many other platforms for desktop developement. The first part of thesis is focused to selection of technologies (such as HHVM, WebSocket, React, ZeroMQ, Ajax Comet, etc.) and to recommanded configuration of the runtime platform. In the further section, the structure of the Framework is described, together with it's components and inclusion of third party tools and libraries such as for example Twig, Doctrine 2 ORM, Composer, Ratchet etc. The final part of the thesis is focused on detailed description of essential components and implementation details. Several chapters are also dedicated to a performance comparation of various configurations.

Keywords: PHP framework, HHVM, WebSocket, realtime, push technology, Ajax, dynamic web, PHP Forms, PHP MVC

Obsah

Seznam použitých zkratek

TCP	**T**ransmission **C**ontrol **P**rotocol, protokol pro přenos dat v internetu, se zaručeným doručením datagramů
HTTP	**H**yperText **T**rasnport **P**rotokol, protokol aplikační vrstvy používaný pro přenos obsahu webových stránek v internetu
PHP	Rekurzivní zkratka, **P**HP **H**ypertext **P**reprocesor, skriptovací programovací jazyk
HHVM	**H**ip**H**op **V**irtual **M**achine, běhová platforma transformující PHP skripty na předkompilovaný kód
CGI	**C**ommon **G**ateway **I**nterface, rozhraní pro propojení externích aplikací s webovým serverem
FPM	**FastCGI P**rocess **M**anager
ZMQ	ZeroMQ, **Z**ero **M**essage **Q**ueue, knihovna pro výměnu zpráv prostřednictvím fronty
JSON	**J**ava**S**cript **O**bject **N**otation, začkovací jazyk pro reprezentaci (nejen) javascriptových objektů
XML	e**X**tensible **M**arkup **L**anguage, obecný rozšířitelný značkovací jazyk
W3C	**W**orld **W**ide **W**eb **C**onsortium, konsorcium spravující webové standarty
MVC	**M**odel **W**iew **C**ontroller, návrhový vzor softwarové architektury
WS	**W**eb **S**ocket, protokol pro přenos dat na pozadí stránek
CSS	**C**ascading **S**tyle **S**heets, kaskádové styly pro úpravu vizuální podoby webových stránek
JS	**J**ava**S**cript, skriptovací jazyk používaný ve webových prohlížečích
DOM	**D**ocument **O**bject **M**odel, objektová reprezentace HTML dokumentu
OOP	**O**bject **O**riented **P**programming, objektově orientované programování
UI	**U**ser **I**nterface, uživatelské rozhraní
ORM	**O**bject **R**elational **M**apping, technika pro automatickou konverzi mezi relační databází a aplikačními objekty

Seznam ilustrací a grafů

Seznam tabulek

Seznam zdrojových kódů

Konvence v sazbě

- `Barevně`, neproporcionálním písmem jsou psány názvy tříd a jmenných prostor
- *kurzívou* jsou psány názvy funkcí a atributů
- `neproporcionálním` jsou zvýrazněny cesty a URL adresy a také ukázky kódu (v ohraničených segmentech)
- KAPITÁLKAMI_S_PODTRŽÍTKY místo mezer a bez diakritiky jsou psány konfigurační konstanty
- *"kurzívou v anglických uvozovkách"* jsou psány řetězcové hodnoty

Úvod

Od okamžiku, kdy Tim Berners-Lee spustil historicky první webový server s prvními webovými stránkami, napsanými ve značkovacím jazyce HTML, uplyne letos čtyřiadvacet let. Během této doby učinili technologie pro vývoj a provoz webových stránek nesmírný pokrok. Zatímco v počátcích se jednalo pouze o prostý text propojený hypertextovými odkazy, dnes většina webů nabízí pestrý multimediálně interaktivní obsah, generovaný a obsluhovaný robustními aplikacemi. Tuto proměnu doprovázela celá řada významných milníků. V následujících čtyřech odstavcích přiblížím čtyři nejvýznamnější.

Jedním z prvních zásadních nedostatků webových stránek, který začali jejich tvůrci řešit, byla nutnost stahovat celý obsah při průchodu stránkami znovu i přes to, že změna obsahu byla nepatrná. Tato skutečnost byla nepříjemná zejména v době kdy se rychlost běžného uživatelského připojení k internetu pohybovala pouze v řádech jednotek kilobytů za sekundu. S řešením přišla společnost Netscape, která v roce 1996 zavedla a prosadila podporu rámců (frames), pomocí kterých bylo možné stránky rozdělovat na segmenty a v každém z nich zobrazovat jiný dokument. Při navigaci stránkami pak stačilo například aktualizovat obsah pouze vybraného segmentu.

Později se ukázalo, že rámce nejsou ideálním řešením z hlediska indexování vyhledávacími roboty, bezpečnostních rizik, či skriptování na straně prohlížeče. Asynchronní aktualizace stránek se začala řešit pomocí různých zásuvných modulů třetích stran, jako například Java applety poskytující rozhraní pro javascriptové funkce nebo Flash embed objekty. Převratný zlom v tomto ohledu nastal až začátkem roku 2006, kdy většina majoritně používaných prohlížečů již implementovala XMLHttpRequest rozhraní a W3C konsorcium je přijalo jako pracovní koncept pro novou specifikaci. Toto rozhraní začalo být využíváno v rámci technologie nazývané Ajax,

která umožňuje jednoduchým způsobem dynamicky načítat součásti stránky ze serveru pomocí HTTP požadavků prováděných na pozadí.

Další výzvou se stala otázka, jak efektivně aktualizovat obsah stránky v okně prohlížeče, na základě událostí na serveru. Nedlouho poté, co se ujala technologie Ajax, koncem roku 2006 se objevila řada webů, včetně Meebo, Gmailu či Facebooku, která začala využívat Ajax požadavky vyčkávající na serveru na relevantní událost. Server odpovídá vždy až v okamžik, kdy má potřebu informovat klienta o nové události nebo změně. Tento princip se začal označovat jako Comet, Ajax push či reverse Ajax.

Ačkoliv se Comet princip využívá dodnes, s narůstajícím objemem takto přenášených informací se vývojáři začali zabývat otázkou režie HTTP protokolu. Postupně byla do prohlížečů implementována podpora pro technologii WebSocket, která umožňuje vytvořit permanentní spojení mezi klientem a serverem a prostřednictvím tohoto spojení předávat zprávy o veškerých změnách na obou stranách. WebSocket byl standardizován konsorciem W3C v roce 2011, jeho podpora prohlížeči však dodnes není stoprocentní.

Předmětem této práce je tvorba frameworku pro dynamická rozhraní webových aplikací, emulující desktopové programování. Pojmem dynamické rozhraní webové aplikace se rozumí takové aplikační rozhraní, které klade vysoké požadavky na souběžný přístup více uživatelů ve stejný okamžik a na asynchronní aktualizaci zobrazovaných dat v reálném čase. Tyto aktualizace jsou zpravidla vyvolávány jinými uživateli, či událostmi na serveru, proto je zde nutné použít některou z technologií uvedených v předchozích dvou odstavcích. Desktopové programování je pak použito jako souhrnné označení pro vývojové techniky používané při programování desktopových aplikací, kde je možné jednoduchým způsobem definovat ovládací prvky a přiřadit jim obslužné metody pro různé typy událostí.

V desktopovém programování narozdíl od webového není nutné používat kromě primárního vývojového jazyka žádný další značkovací jazyk, skripty pro stranu prohlížeče, kaskádové styly ani například soubory s šablonami. Vše je soustředěno na jedno místo. Příkladem platformy pro desktopové programování může být například Windows Forms nebo QT API.

Jazyků a platforem pro tvorbu webových aplikací je dnes nepřeberné množství. Jazyk PHP byl pro realizaci této práce zadán především pro jeho rozšířenost a oblíbenost mezi vývojáři. Například server W3Techs.com uvádí, že k počátku roku 2014 téměř 82% webových stránek používalo právě jazyk PHP.

1 Analýza a způsob realizace

Zadání práce zahrnuje poměrně obecné zásady pro vypracování. Jak již bylo zmíněno, jejím hlavním cílem bylo realizovat komponenty specializovaného frameworku pro jazyk PHP, umožňující vyvýjet webové aplikace s komfortem desktopového programování. Vzhledem k návaznosti na komerční projekt a předešlým zkušenostem však bylo úvodní snahou vytyčit podrobnější a přísnější požadavky na realizaci, aby se stal výsledek práce skutečně přínosným a mohl být dále v praxi využit. V této kapitole budou, vyjma představení stanovených požadavků, dále popsány veškeré zvažované koncepty a myšlenky týkající se realizace. Podstatná část je zde také věnována výsledkům úvodní rešerše a popisu dostupných technologií, které pro realizaci přicházeli v úvahu.

Při sestavování podrobnějších požadavků pro realizaci vyplynulo, že framework, použitý v realizovaném řešení, by měl mít zejména tyto vlastnoti:

- Modularita a snadná rozšířitelnost
- Přizpůsobitelnost a předefinovatelnost funkcionality jednotlivých součástí
- Přehledná struktura a jednotné konvence
- Vysoký výkon a propustnost
- Podpora různých běhových platforem a kompatibilita s HHVM

V rámci své podnikatelské činnosti jsem koncem roku 2012 inicioval vývoj nového frameworku pro makléřskou společnost Credit & Hypo Consult, která provozuje internetové srovnávače produktů z oblasti pojišťovnictví. Framework vznikal v rámci rozsáhlého projektu a postupně se pro něj vžil název Phem. Hlavní motivací pro jeho vznik byla snaha vybrat to nejlepší z již existujících frameworků a nástrojů, dostupných pod licencí permisivního typu a dále doplnit některé součásti, pro které v PHP neexistovala uspokojivá implementace. Navíc bylo při jeho vývoji dbáno na

dodržení výše zmíněných vlastností. Framework Phem se tedy jevil jako vhodný základ pro realizaci komponent, jež jsou předmětem této práce, nakonec byl pro realizaci i vybrán. Nicméně, jak bylo nastíněno v úvodu, vývoj technologií pro tvorbu webových stránek postupuje velmi rychle, a proto byla zvažována i možnost použít jako základ jiný moderní framework, nebo začít úplně od základu. Podrobnějšímu průzkumu a srovnání frameworků je věnována sekce 1.1. Představení frameworku Phem je věnována samostatná kapitola 2.

Bezprostředně po výběru základového frameworku bylo nutné vytvořit koncept celé architektury nových frameworkových součástí, které umožní vývoj obdobným způsobem jako v případě desktopových aplikací a které umožní sdílení definovaných vizuálních komponent mezi více uživateli. Vytvořený koncept je popsán v sekci 1.2. Z tohoto konceptu ihned vyvstalo několik dalších otázek, které bylo nutné před samotnou realizací vyřešit. První z nich se týkala persistence vytvořených instancí vizuálních komponent, které je nutné ukládat do úložiště společného pro všechny uživatele. Další otázka byla spjata s možnými způsoby, jak efektivně vyměňovat zprávy a upozorňovat uživatele o změnách vyvolaných jinými uživateli. Jinými slovy, která z push technologií by měla být použita. Poslední zásadní otázkou pak byla rychlost a propustnost a s tím spjatá možnost využití technologie HHVM. Všechny tyto zmíněné otázky jsou podrobněji řešeny ve třech odpovídajících podsekcích na konci této kapitoly.

1.1 Dostupné frameworky a nástroje

Z nepřeberného množství dostupných PHP frameworků bylo v rámci rešerše vybráno a podrobněji prozkoumáno pouze několik takových, které jsou v poslední době mezi vývojáři velmi oblíbené a svými vlastnostmi se přibližují požadavkům, stanoveným v této práci. Každému z nich byly za každou požadovanou vlastnost uděleny body v rozsahu 1-5 za to, do jaké míry byly shledány v daném směru vyhovujícími.

1.1.1 Symfony

Symfony je framework vyvíjený částečně komerční firmou SensioLabs a částečně komunitou, distribuovaný pod MIT licencí. Symfony (dnes již ve verzi 2.4) je jedním z nejoblíbenějších PHP frameworků a díky tomu je dobře dokumentován a široce podporován komunitou.

Modularita a snadná rozšiřitelnost: Jelikož Symfony samotné má architekturu jako stavebnice, lze použít jen vybrané moduly v libovolné kombinaci, tak i moduly Symfony frameworku jsou navrženy velmi obecně. V podstatě se jedná o samostatné aplikace, protože z principu Symfony nemají žádnou jistotu s čím budou spolupracovat. Proto je možné je snadno využít i v jiných php frameworcích. Zisk bodů: 4/5.

Prizpůsobitelnost a předefinovatelnost funkcionality jednotlivých součástí: Symfony k tomuto problému přistupuje svým způsobem a to tak že pokud nějaký modul nevyhovuje potřebám, prostě se vymění za jiný. Případně je možné stávající moduly přepsat nebo doplnit, jelikož licence Symfony to umožňuje. Zisk bodů: 3/5.

Přehledná struktura a jednotné konvence: Symfony jako tradiční framework má už ustálené konvence a kvůli samostatným a dobře dokumentovaným komponentám jsou jasné jejich zodpovědnosti. Na druhou stranu stavebnicová struktura Symfony znamená, že vývojáři nemají k dispozici vždy jednoznačný základ, protože se mezi projekty struktura frameworku liší. Zisk bodů: 4/5.

Vysoký výkon a propustnost: Symfony je poměrně mohutný framework ale ve své dlouhé historii prošel několika optimalizačními vlnami, naposledy mezi verzemi 1.4 a 2, takže dnes patří k těm rychlejším PHP frameworkům. Ovšem kvůli způsobu, jakým toho dociluje – bytecode cache – bude tento náskok z větší části negován při použití HHVM. Zisk bodů: 2/5.

Podpora různých běhových platforem a kompatibilita s HHVM: Podpora HHVM v Symfony kupodivu ještě není perfektní, podle posledních dat 98.9%. 3/5

1.1.2 Larvel

Pravděpodobně nejmladší a v poslední době téměř nejrozšířenější PHP framwork je Larvel, který byl vyvinut Taylorem Otwellem a je vydávaný pod MIT licencí. Je založený na Symfony a jeho aktuální kód je dostupný na serveru GitHub.

Modularita a snadná rozšiřitelnost: Larvel obsahuje balíčkovací systém Bundles, který umožňuje přidat do projektu nespočet již hotových komponent. Jeho použití je velmi jednoduché a intuitivní. Zisk bodů: 5/5.

Přizpůsobitelnost a předefinovatelnost funkcionality jednotlivých součástí: Přizpůsobitelnost součástí je v Larvelu velmi obtížná, avšak za normálních okolností se s ní zde nepočítá, jelikož 99% celého frameworku je umístěno mezi dalšími závislostmi ve složce `vendor`. Pokud je u některé funkce či komponenty požadována jiná funkcionalita, než výchozí, je pravděpodobně lepší využít jiný dostupný bundle. Zisk bodů: 4/5.

Přehledná struktura a jednotné konvence: Struktura adresářů nově vytvořeného projektu v Larvelu je poměrně chaotická a zabere hodně času se v ní zorientovat. Nepřehledné je i uspořádání konfiguračních souborů v novém projektu. Části aplikace se v Larvelu píší mimo jakýkoliv jmenný prostor Zisk bodů: 3/5.

Vysoký výkon a propustnost: Ve srovnání s ostatními frameworky dosahuje ve většině testů při provozu na běžných PHP interpretrech průměrných výsledků. Toto však nepředstavuje velký problém. vzhledem k plně kompatibilitě ze strany HHVM. Zisk bodů: 4/5.

Podpora různých běhových platforem a kompatibilita s HHVM: Dle posledních výsledků unit testů toho frameworku pro HHVM bylo dosaženo 100% kompatibility. Zisk bodů: 5/5.

1.1.3 Kohana

Kohana je komunitou vyvíjený framework vydávaný pod BSD Licencí, vytvořený především pro již zkušené PHP vývojáře, kterým nabízí vyšší flexibilitu než jiné frameworky zejména kvůli atypickému strukturálnímu návrhu, kterému říká „cascading filesystem"(úzce souvisí s modularitou a předefinovatelností, bude vysvětleno dále), ale i například kvůli pokročilému routingu.

Modularita a snadná rozšiřitelnost: Modularita je klíčovou vlastností Kohany. Vzhledem k tomu že moduly mohou pracovat s čímkoliv jako vlastní aplikace a dokonce i na úrovni jádra je obvyklý model alespoň 90% kódu i specifických aplikací přesouvat do modulů. Pro některé typické případy se ovšem hodí pevněji definovaná, událostmi řízená modularizace, kterou Kohana nepodporuje. Zisk bodů: 4/5.

Přizpůsobitelnost a předefinovatelnost funkcionality jednotlivých součástí: Kohana přistupuje ke vztahu jádro-moduly-aplikace nekonvenčně. Na rozdíl od tradičních frameworků které pro každý z těchto prvků definují specifickou strukturu a pevná pravidla pro interakci, Kohana ve všech kopíruje stejnou strukturu a pro výsledné chování se jejich obsah zkombinuje. V případě konfliktů mají soubory z aplikace nejvyšší prioritu, dále pak moduly a nakonec systém. Tento koncept se v Kohaně nazývá „cascading filesystem"a umožňuje z modulů nebo aplikace libovolně změnit fungování jádra bez jeho přímého přepsání. Zisk bodů: 5/5.

Přehledná struktura a jednotné konvence: Kohana kvůli cílení na pokročilé vývojáře nevynucuje příliš konvencí. Přestože je architektura průzračná, snadno se stane nepřehlednou. Především kvůli tomu, že moduly nejsou explicitně rozdělené podle zodpovědností – jakýkoliv modul může obsahovat cokoliv. Zisk bodů: 1/5.

Vysoký výkon a propustnost: Kohana sama o sobě je extrémně odlehčená a proto i rychlá, ale závisí na správně použité modularizaci. Zisk bodů: 4/5.

Podpora různých běhových platforem a kompatibilita s HHVM: Podpora HHVM v Kohaně se postupně zlepšuje. Dříve bylo nutné použít HHVM modul, nyní je

kompatibilita Kohany s HHVM 99.19% kde poslední problémy jsou s cachingem, takže Kohana je pod HHVM použitelná. Zisk bodů: 4/5.

1.1.4 Phem

Framework Phem, rozšiřovaný v rámci této práce, je poměrně podrobně popsán v kapitole 2, proto není třeba jej zde popisovat příliš detailně. Jeho předností je jednoduchost, přehlednost a snaha řešit věci jiným způsobem, podobajícím se spíše řešením z platforem Java a Microsoft .NET. Zároveň je z velké části inspirovaný frameworky Symfony2 a Nette.

Modularita a snadná rozšířitelnost: K dispozici jsou dvě možnosti rozšiřování frameworku - pomocí vlastní implementace BootStrapperu či prostřednictvím Composeru. Moduly třetích stran, zahrnuté ve výchozí instalaci, lze navíc v případě potřeby snadno odebrat. Zisk bodů: 5/5.

Přizpůsobitelnost a předefinovatelnost funkcionality jednotlivých součástí: Podstatné součásti lze předefinovat vlastní implementací abstraktních tříd nebo rozhraní. Zároveň jsou do konfiguračních souborů aplikace a frameworku zaneseny veškeré relativní cesty, sufixy, názvy výchozích implementací apod. Součásti však nelze přepisovat maskováním tak snadno jako například v Kohaně. Zisk bodů: 3/5.

Přehledná struktura a jednotné konvence: Struktura frameworku je přehledná, nepříliš hluboká a logicky uspořádaná. Jmenné konvence pro názvy tříd a jmenných prostor odpovídají adresářové struktuře. Zisk bodů: 5/5.

Vysoký výkon a propustnost: Základní kostra frameworku je velmi výkonná. Problematické mohou být některé komponenty využívající reflexi. Vzhledem k možnosti cachování a podpory HHVM to však není tak zásadní nedostatek. Zisk bodů: 4/5.

Podpora různých běhových platforem a kompatibilita s HHVM: V Březnu 2014 vývojáři HHVM oznámili, že na této platformě již procházejí všechny unit testy Doctrine2 a Twig. Tím pádem by měl být s HHVM z kompatibilní i framework Phem, jelikož toto byly jediné dvě součásti, které kompatibilitu omezovaly. Zisk bodů: 5/5.

1.1.5 Vyhodnocení

Všechny testované frameworky nabízejí v základní funkcionalitě velmi podobné možnosti. Frameworku Phem byl nakonec upřednostněn zejména kvůli dobrým zkušenostem a zahrnutí některých komponent, které by mohly být užitečné pro realizaci této práce. Vyjma něj se jako nejperspektivnější jevil Larvel. Tabulka 1 zobrazuje podrobné výsledky srovnání vybraných frameworků, včetně celkového získaného počtu bodů.

Tabulka 1: Srovnání vybraných frameworků

Název	Modul.	Přizpůs.	Strukt.	Výkon	Kompat.	Celkem
Symfony	4	3	4	2	3	16
Larvel	5	4	3	4	5	21
Kohana	4	5	1	4	4	18
Phem	5	3	5	4	5	22

1.2 Koncept uživatelského rozhraní

Navrženo bylo rozhraní, které sestává ze tří základních typů vizuálních komponent. Prvním z nich je pracovní plocha (workspace), která smí obsahovat libovolný počet statických či pohyblivých oken (window). Tato okna tvoří druhý typ vizuálních komponent. Posledním typem jsou pak ovládací prvky (control), které mohou být v libovolném počtu obsaženy v oknech. K dispozici by měla být základní sada ovládacích

prvků a další by měly být snadno doplnitelné pomocí rozšíření již existujících či jejich abstraktního předka.

Veškeré vytvořené instance pracovních ploch či samostatných oken by měly být uloženy v úložišti aplikačních proměnných a měla by existovat možnost sdílení na třech různých úrovních (mezi různými aktivními spojeními, mezi sezeními a mezi uživateli). Veškeré události v okně prohlížeče by měly být okamžitě propagovány na server, kde se zapracují do modelových dat konkrétní instance okna či pracovní plochy. Pokud je dané okno či pracovní plocha ve stejnou chvíli v rámci sdílení zobrazená i z jiného spojení, toto by mělo být neprodleně notifikováno o nastalé události a změnách.

1.3 Ukládání aplikačních proměnných

Ukládání proměnných, platných v rozsahu celé aplikace lze v PHP realizovat mnoha způsoby. Mezi nejběžněji používané implementace úložiště aplikačních proměnných patří:

- APC
- XCache
- Wincache
- Sdílená paměť
- Relační databáze
- Souborový systém

Většina moderních frameworků podporuje alespoň některou z výše uvedených implementací. První tři z nich slouží primárně jako cache pro výstupy skriptů, které jsou volány opakovaně a nabízejí řadu funkcí pro snadnou manipulaci s ukládanými a načítanými daty. Poslední tři implementace jsou spíše méně vhodné. Ukládání přímo do sdílené paměti vyžaduje obtížnou údržbu, databáze je příliš robustní a neuniverzální a ukládání do souborů příliš pomalé (pokud tedy nejsou umístěny například v ramdisku). Podle většiny rychlostních testů je nejlepší volbou XCache,

součástí které je i jednoduché administrační rozhraní. Framework Phem aktuálně podporuje APC a XCache, což se jeví jako dostatečná možnost volby.

1.4 Výměna zpráv a notifikace

V úvodu byly nastíněny dva způsoby pro notifikaci okna prohlížeče ze strany serveru. Tyto dva způsoby byly jako nejčastěji používané nakonec také zapracovány do komponenty MessageBus. Pro úplnost jsou v této sekci popsány hlavní dostupné technologie sloužící pro tento účel. Jejich detailnější popis lze nalézt například v kapitole 2.3 kvalifikační práce Push-teknik på webben (Bruksås et al., 2010).

1.4.1 Flash RTMP

Platforma Flash podporuje push metodu prostřednictvím protokolu Real Time Messaging (RTMP), který je používán pro přenos dat mezi serverem a Flash klientem. RTMP komunikuje přes TCP/IP, ale může být nahrazen duálním připojením přes HTTP. Jednou z nevýhod RTMP je to, že vyžaduje instalaci Flash pluginu do prohlížeče.

1.4.2 Comet a dlouhé dotazování

V roce 1995 představila společnost Netscape techniku pro odesílání webových stránek po více částech využitím MIME typu multipart/x-mixed-replace. Později na toto navázali modernější webové prohlížeče, které pomocí XmlHttpRequest objektu spolu s dalším obslužným javascriptem využívat možnosti asynchronně aktualizovat části stránek. Tato technologie začala být později nazývána jako AJAX (Asynchronous Javascript And XML) či také Ajax. Ajax Comet či Ajax Push je pak označována ještě modernější technika založená na požadavcích tohoto typu. V tomto případě je však požadavek uměle zdržován na serveru a vyčkává do doby kdy jsou dostupná relevantní data pro odpověď. Jakmile je odpověď doručena spojení se ukončí, ale okamžitě se naváže nové.

1.4.3 XHRStreaming

Technologie XmlHttpRequestStreaming (XHRStreaming), občas též nazývaná jako HTTP streaming, funguje velice podobně jako dlouhé dotazování, s tím rozdílem, že spojení se udržuje aktivní po celou dobu zobrazení stránky. Toho je dosaženo tak, že server odpovídá po částech bloky javascriptového kódu. Výhodou tohoto řešení oproti předchozímu je menší režie přenesených dat a vyšší výkon. Nevýhodou je šak podpora pouze prohlížeči založenými na jádrech Gecko a Webkit.

1.4.4 WebSocket

WebSocket je nový standard zahrnutý v HTML5. Umožňuje otevřít obousměrné spojení mezi prohlížečem a serverem, založené na TCP socketu. Často je označován za „skutečný" push, pro odlišení od simulace pomocí dlouhého dotazování a jiných technik. Po standardizaci konsorciem W3C je pravděpodobné, že se WebSocket brzy stane primárním způsobem pro implementaci push techniky. V současné době však zůstává problémem kompatibilita se staršími prohlížeči, proto bývá WebSocket často doplněn o nějakou rezervní alternativu v případě, že není daným prohlížečem podporován.

1.5 Technologie HHVM a její podpora

Jak je psáno například v jednom z nedávno vydaných článků o HHVM (Berchet, 2014), historie této technologie, jejíž plný název zní HipHop Virtual Machine (též HipHop for PHP), se začala psát až v roce 2008. V té době na jejím vývoji začala pracovat společnost Facebook. HHVM by mělo řešit jeden z nejzásadnějších problému PHP, kterým je rychlost. PHP je skriptovací jazyk, který je nutné interpretovat při každém požadavku. V některých případech lze sice využitím cache výsledky některých frekventovaných dotazů ukládat a používat opakovaně, nicméně PHP svou rychlostí nikdy nebylo schopné překonat konkurenční platformy jako ASP.NET či JSP.

Princip činnosti HHVM spočívá v lexikální analýze a transformaci PHP skriptů do vysokoúrovňového byte kódu. Tento kód je následně za běhu pomocí just-in-time (JIT) kompilátoru překládán do nativního kódu dané počítačové architektury. HHVM se tedy v tomto ohledu chová podobně jako .NET nebo JRE. Pro zvýšení efektivity a komfortu vývoje byl navíc pro HHVM vyvinut jazyk Hack, který má téměř stejnou syntaxi jako PHP a zavádí do jazyka silnou typovost, generické třídy, lambda výrazy a další prvky moderních vysokoúrovňových jazyků. Výhodou je, že Hack a PHP lze v rámci jednoho projektu kombinovat. Při použití Hacku je však třeba počítat, že danou aplikaci nebude možné provozovat např. na jiném serveru s běžným PHP interpretrem.

Jelikož jde o technologickou novinku, která se většího rozšíření a použitelnosti dočkala až počátkem roku 2014, existuje stále řada knihoven a nástrojů, které nejsou s HHVM stoprocentně kompatibilní. Podrobnosti o kompatibilitě jsou zmíněny v závěru této práce, v kapitole 6.

2 Představení frameworku Phem

Jak již bylo naznačeno v předchozí kapitole, framework Phem vznikl jako součást rozsáhlého komerčního projektu a zatím nebyl zveřejněn široké veřejnosti. Je však velice pravděpodobné, že tak bude v dohledné době učiněno. Klasifikovat by se dal jako framework postavený na klasické MVC architektuře typu push. Zpracovávání požadavku tedy začíná v některé z akcí definovaných dostupnými controllery. Během zpracování se pak vytvoří datový model, který se vloží do výsledného pohledu a ten je následně vykreslen a odeslán jako tělo odpovědi. Podrobně se principům MVC věnuje například publikace Pro PHP MVC (Chris, 2012).

2.1 Základní struktura

V adresáři s webem, založeným na tomto frameworku, se vždy nacházejí adresáře `Application` a `Framework` a soubory `application.conf.php`, `framework.conf.php`, `index.php` a `init.php`. Tato struktura zajišťuje striktní oddělení frameworku od konkrétní aplikace. Struktura adresáře `Framework` bude přiblížena v následujícím odstavci. Struktura adresáře s aplikací je zcela volitelná, vhodné je však dodržovat obdobnou strukturu jako v případě frameworku. Soubory `application.conf.php` a `framework.conf.php` obsahují veškeré konfigurační konstanty pro danou aplikaci i framework, kterými lze ovlivnit výchozí chování a nastavení cest, přístupů do databáze apod. Ve většině případů však stačí upravovat soubor `application.conf.php`, do kterého je vhodné přidávat i veškeré uživatelem definované konstanty. Soubor `framwork.conf.php` je nutné upravovat pouze ve výjimečných případech, například pokud je framework zahrnut do jiné aplikace jako podadresář. Soubor `index.php` je vstupním bodem a `init.php` slouží k inicializaci frameworku, zejména načtení minimální sady souborů, nezbytných pro zavedení automatického načítání tříd.

Struktura adresáře `Framework` sestává z následujících položek:

- `3rdParty` – Knihovny a závislosti třetích stran
- `ClassLoading` – Automatické načítání tříd
- `Controllers` – Základní univerzální controllery
- `Core` – Jádro, obsahující prapředka – typ `Object`, kolekce, výjimky či zámky
- `Environment` – Prostředí, obsahující statické factory třídy, zejména `EnvironmentManager` a `Application`
- `Libraries` – Vlastní knihovny frameworku
- `Resources` – Šablony a zdroje pro stranu prohlížeče – kaskádové styly, javascripty, obrázky a další typy zdrojů
- `Views` – Univerzální pohledy – například `SimpleView` pracující s šablonami

Srdce frameworku tvorí automatické zavaděče tříd (ClassLoadery), které zajišťují automatické načítání tříd pomocí PHP autoload funkcí. Vyjma nejsvrchnějších adresářů `Framework` resp. `Application`, kterým je přiřazen jmenný prostor (namespace) `Phem` resp. název aplikace, je napříč celým frameworkem dodržena konvence, že každá třída je umístěna v samostatném souboru, v takovém relativním umístění, v jakém se nachází jmenném prostoru. Navíc soubor se třídou je vždy, až na příponu „.php", pojmenován stejně jako daná třída. K dispozici jsou tři základní implementace ClassLoaderů, jedna pro načítání tříd frameworku, další pro načítání tříd aplikace a poslední pro načítání závislostí třetích stran. Ta je o něco komplikovanější, proto je popsána v samostatné sekci 2.2. Zavaděč tříd pro aplikaci počítá s dodržením stejné konvence, jaká je dodržena ve frameworku. V případě že vývojář aplikace hodlá použít konvence jiné, je potřeba aby si vytvořil vlastní implementaci ClassLoaderu, odvozením abstraktní třídy `Phem\ClassLoading\Loader` a zajistil její aktivaci v inicializačním souboru aplikace, který je třeba vždy umístit do složky s aplikací. Tento soubor musí být pojmenován stejně jako je pojmenována daná aplikace v konfiguračním souboru, s extenzí „*Boot.php".

2.2 Rozšířitelnost

Framework Phem lze snadno rozšířit, vytvořením komponenty třetí strany. Stačí vytvořit podsložku v umístění `Framework/3rdParty` a do ní kromě vlastního kódu umístit takzvaný bootstrapper, který zajistí její správné načítání. Bootstrapper musí dědit od abstraktní třídy `Phem\ClassLoading\Bootstrapping\Bootstrapper` a musí být pojmenován stejně jako vytvořená složka s extenzí „*Boot.php". V rámci tohoto bootstraperu je nutné implementovat abstraktní metodu *load*, ve které lze například zaregistrovat vlastní autoloader nebo čistě staticky načíst potřebné soubory pomocí funkcí *include* či *require*.

Jinou možností pro připojení knihoven třetích stran je použití Composeru, který je již výše uvedeným způsobem zahrnut. Pokud vývojář aplikace umí nástroj Composer používat, stačí pouze upravit soubor `composer.json` v jeho adresáři a nechat aktualizovat strom závislostí.

2.3 Zahrnuté knihovny třetích stran

Prostřednictvím dvou principů, popsaných v předchozí sekci (komponent třetích stran či Composeru) jsou do frameworku již v základu připojeny některé elementární knihovny. Mezi tyto knihovny patří zejména Doctrine 2 ORM – objektově relační mapovací nástroj pro jednoduchou práci s relační databází, šablonovací systém Twig či knihovny pro import a export a konverzi souborů (TFPDF, h2t, a další).

2.4 Další funkcionalita a klíčové komponenty

V této sekci jsou představeny ty nejpodstatnější funkce a vlastní knihovny a komponenty frameworku Phem. Účelem zde není podrobná dokumentace všech dostupných tříd a metod, ale pouze ilustrativní představení.

2.4.1 Základní controllery

Controllery a jednotlivé úlohy (akce) lze definovat velmi snadným způsobem. Každý controller by měl rozšiřovat třídu `Phem\Controllers\Controller`, tím získá základní schopnosti jako například automatické dosazování argumentů akcí, vyextrahovaných z argumentů HTTP požadavku (GET a POST) dle jmenné konvence. V průběhu zpracovávání kterékoliv akce lze kdykoliv požadavek přesměrovat do jiné akce a to buď okamžitě a nebo ho naplánovat, aby byl vykonán po dokončení současné akce. Během přesměrování lze předávat argumenty vnějškem (přes klientskou stranu) i vnitřkem (přes dočasné úložiště).

V případě, že je pro některý controller použita jako rodičovská třída `Phem\Controllers\SimpleViewController`, je vývojáři nabídnuta další implicitní konvence, která zajišťuje jednoduché párování akcí s výstupní pohledy, realizovanými pomocí Twig šablon. Pokud je například nějaký controller pojmenován DummyController a v něm je nadefinována akce *makelove*, stačí ve složce se šablonami (cesta lze nastavit v konfiguraci) vytvořit podsložku `Dummy` a v ní šablonový soubor `makelove.twig`. `SimpleViewController` po dokončení dané akce na základě této konvence zajistí automatické předání modelu, připraveného v rámci akce, správné šabloně pohledu. Tuto šablonu lze navíc definovat jako rozšíření některého z předdefinovaných rozvržení.

2.4.2 Generování odkazů

K vytváření odkazů na jiné stránky (akce) téhož webu lze použít `LinkBuilder`. Obsahuje několik metod, kterým se jako argument předává asociativní pole, kolekce nebo řetězec ve formátu JSON či CSV. Tímto argumentem je možné nadefinovat veškeré argumenty odkazu, který se sestaví automaticky a zahrne případně relativní cestu. Není tedy nutné zakotvit odkazy přímo do kódu, čímž se předejde komplikacím v případě restrukturalizace aplikace. Dále je v rámci `LinkBuilderu` nabídnuta metoda *navigate*, sloužící též pro vygenerování odkazu. Té stačí předat pouze dva jednoslovné argumenty – název controlleru a název akce. Pro vytváření

odkazů na zdroje, jakými jsou např. obrázky, je možné použít metodu *linkToResource*. V případě SimpleViewControlleru se `LinkBuilder` vkládá automaticky do modelu pod klíčem *lb*, tím se stává dostupný i v rámci šablonovacího jazyka Twig.

2.4.3 Instantní formuláře

Komponenta `AnnotationDriverFormBuilder` představuje možnost generování formulářů pouhým doplněním anotací nad třídy datového modelu. Pomocí jednoduchých klíčových slov lze nastavit, který atribut dané třídy bude reprezentován jako textové pole, rolovací nabídka či zatržítko apod. Podporovány jsou i podskupiny (fieldsety), do kterých lze například generovat formuláře referencovaných tříd a rozšíření pomocí dědičnosti. Lze tedy generovat formuláře pro rozsáhlé struktury a vyhnout se ručnímu mapování hodnot. Tímto způsobem je zajištěn pokročilý databinding, známý například z platformy ASP.NET.

2.4.4 XML import a export

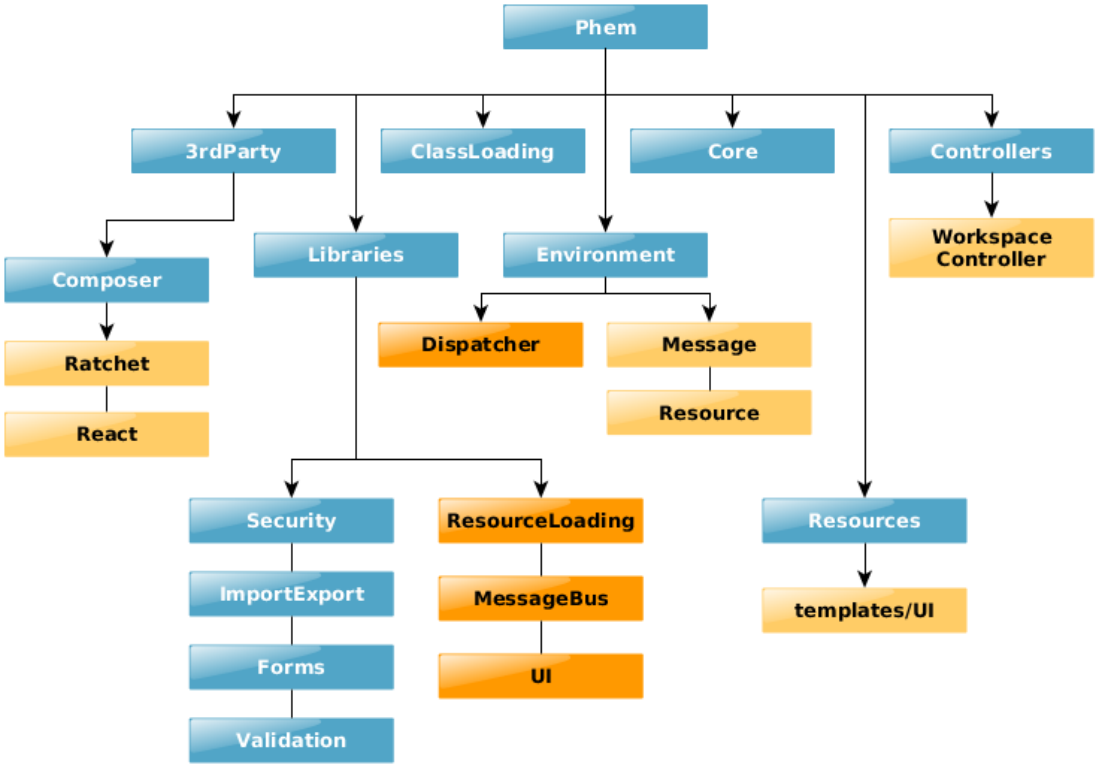
Pro snadnou manipulaci s XML soubory byla v rámci frameworku vytvořena vlastní knihovna `XmlSerializer`, která stejně jako `AnnotationDriverFormBuilder`, zmíněný v předchozí sekci, využívá anotace k zajištění mapování objektu na XML řetězec a zpět. Předlohou XmlSerializeru se stala knihovna LexaXmlSerializer, jejíž vývoj byl však ukončen.

2.4.5 Správa uživatelů a oprávnění

Pro správu uživatelů a definici přístupových práv k jednotlivým controllerům a akcím lze použít komponentu Security. V rámci této komponenty lze volit různé poskytovatele pro autentizaci a autorizaci, pro zdroj uživatelů a skupin a zdroj přiřazených oprávnění.

3 Realizované komponenty

Tato kapitola je věnována popisu všech komponent realizovaných a začleněných do frameworku Phem v rámci této diplomové práce. Pozornost je věnována jejich obecnému účelu, součinnosti s ostatními komponentami a také implementačním detailům. Obrázek 1 znázorňuje základní strukturu frameworku a jeho hlavních součástí. Tmavě oranžovou barvou jsou zvýrazněny komponenty, které byly doplněny. Světle oranžovou jsou pak znázorněny další závislé součásti doplněné současně s novými komponentami.



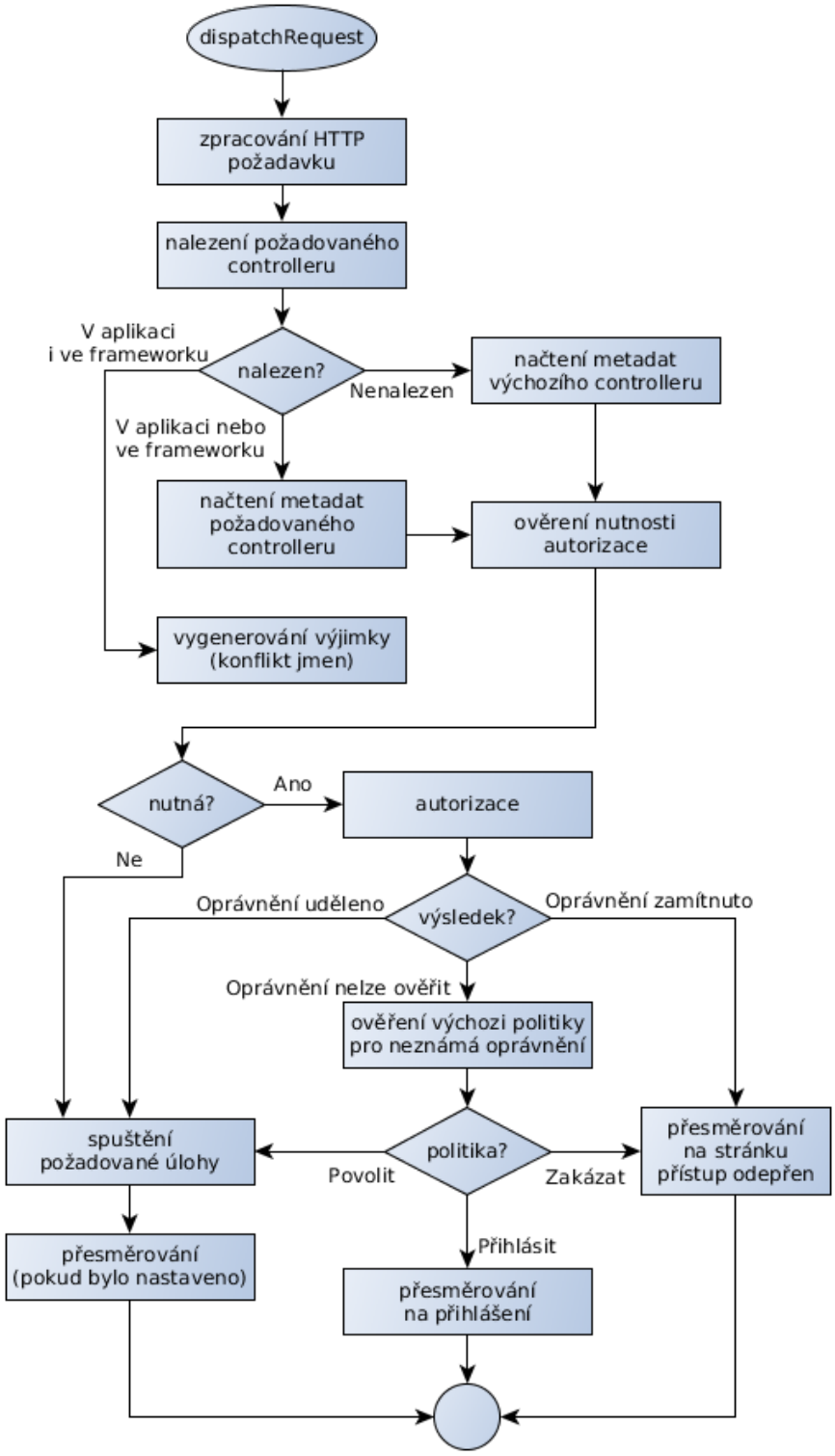
Obrázek 1: Diagram způsobu začlenění nových komponent do frameworku Phem

3.1 Komponenta Dispatcher

Prvním nutným krokem před realizací komponent, klíčových v rámci této práce, bylo sjednocení inicializace frameworku, přihlašování, ověřování totožnosti uživatele, oprávnění a směrování požadavků do metod příslušných controllerů. Toto vše bylo doposud řešeno poměrně chaotickým způsobem, přímo v indexovém souboru aplikace, navíc se kód s postupným vývojem v různých případech nasazení diverzifikoval a stal se vzájemně nekompatibilním. Tato komponenta by měla zahrnout podporu pro všechna doposud známá řešení a navrátit kompatibilitu a možnost snadných aktualizací. Dále přidává několik nových funkcí (například univerzální přihlašovací dialog) a také několik konfiguračních konstant, pomocí kterých lze snadno definovat chování aplikace v případě selhání autentizace či autorizace daného požadavku.

3.1.1 Princip činnosti

Na obrázku 2 je vývojový diagram, který znázorňuje základní funkcionalitu komponenty `Dispatcher`. Pokaždé, při načtení kterékoliv stránky aplikace se ihned po inicializaci frameworku aktivuje metoda *dispatchRequest* třídy `Dispatcher`. V této metodě se nejprve zpracují parametry HTTP požadavku a extrahuje se argument, který určí, do jakého controlleru má být požadavek dále směrován. Na základě tohoto argumentu je controller vyhledán ve složkách s controllery v aplikaci i ve frameworku. Pokud existuje alespoň v jednom z těchto umístění, jsou načtena další metadata a pokračuje se ověřením autorizace. Pokud controller nebyl nalezen v žádném z těchto umístění, na základě použije se výchozí, který lze nastavit pomocí konfigurační konstanty DEFAULT_CONTROLLER. V opačném případě, pokud controller existuje současně i ve frameworku i v aplikaci, znamená to, že vývojář aplikace pravděpodobně použil pro název svého controlleru některý z rezervovaných názvů a je tudíž vygenerována výjimka.



Obrázek 2: Vývojový diagram funkce *dispatchRequest*

V další fázi se nejprve ověří, zda je nutné daný HTTP požadavek autorizovat. Možnosti, kdy lze autorizaci pominout jsou dvě. První nastává v případě, kdy je prostřednictvím konfigurační konstanty LOGIN_REQUIRED zcela deaktivován přihlašovací a ověřovací mechanizmus. Další možností je výjimka pro `DispatcherController`, který je přístupný vždy. V těchto případech je požadovaná úloha automaticky vykonána, bez nutnosti dalšího ověřování. Samotná autorizace je přenechána komponentě Security, která prostřednictvím metody *canTask* nad některou z implementací rozhraní `IAuthorizationProvider` určí, zda je daný uživatel oprávněn provádět požadovanou úlohu z požadovaného controlleru. Pokud je oprávnění uděleno, úloha je vykonána. Pokud je odepřeno, dojde k přesměrování na stránku informující o odepřeném přístupu. V případě, že oprávnění nelze ověřit, je rozhodnuto na základě výchozí přihlašovací politiky, jak bude s požadavkem naloženo dále. Tuto politiku lze nastavit konfigurační konstantou LOGIN_POLICY, přičemž jsou k dispozici tři různé volby: povolení přístupu, odepření přístupu a přesměrování na přihlašovací stránku.

3.1.2 Přihlašování uživatelů

Přihlašování je doporučeno řešit prostřednictvím `DispatcherControlleru`, který nabízí veškeré potřebné metody, pro přihlášení, odhlášení, přesměrování a zachycení případných chyb. Využívá jednoduchý přihlašovací dialog, vykreslovaný prostřednictvím Twig šablony, který je možné přizpůsobit pomocí kaskádových stylů, definovaných v rámci aplikace. Možné je však použít i vlastní řešení, výběrem jiného controlleru pro správu přihlašování, pomocí konfiguračních konstant DISPATCHER_CONTROLLER_NAME a DISPATCHER_LOGIN_TASK_NAME.

3.2 Komponenta ResourceLoading

Jednou ze slabších stránek frameworku doposud zůstávala správa zdrojů pro klientskou stranu (stranu prohlížeče). Mezi tyto zdroje patří javascriptové knihovny, kaskádové styly, fonty, ikony, obrázky, zvuky apod. Přestože ve frameworku byla

snaha tyto soubory třídit podle typu do jednotlivých složek, existuje celá řada knihoven třetích stran, které obsahují různé typy zdrojů. Jejich začlenění a roztřídění do správných složek (a s tím leckdy související nutná modifikace) je vysoce kontraproduktivní. Často tedy bylo výhodnější řešit začlenění těchto knihoven na straně aplikace, nikoliv v rámci frameworku. Nyní lze však snadno tyto závislosti spravovat pomocí nového mechanizmu v rámci komponenty ResourceLoading.

3.2.1 Prohledávaná umístění a typy zdrojů

Soubory zdrojů se mohou nacházet ve třech různých umístěních, podle jejich původu, rozsahu použitelnosti a účelu. Tabulka 2 přibližuje jejich význam.

Tabulka 2: Přehled umístění soubrů se zdroji pro klientskou stranu

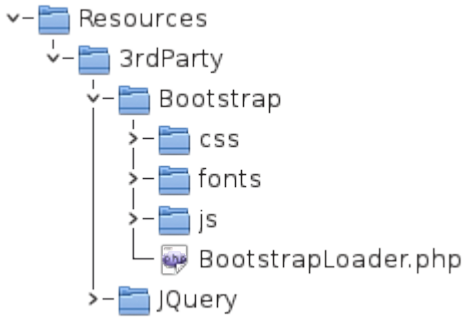
Umístění	Popis obsahu
Application/Resources/	zprávy jsou přenášeny prostřednictvím permanentního websocketového spojení
Framework/Resources/	pro přenos zpráv je použita metoda comet (vyčkávající Ajaxový dotaz)
Framework/Resources/3rdParty	kombinace obou výše uvedených metod

V prvních dvou případech existuje vždy pro každý typ zdroje samostatná složka, která je při načítání procházena rekurzivně a tedy pro odpovídající typ zdroje jsou soubory načteny z dané složky i ze všech podsložek. Do složky `Resources` pod složkou aplikace jsou umísťovány zdroje vytvořené v rámci dané aplikace. Ve složce `Resources` pod složkou frameworku jsou pak univerzální zdroje, kterými disponuje framework sám o sobě. Zdroje javascriptů a kaskádových stylů jsou spravovány ResourceManagerem, o kterém pojednává samostatná sekce 3.2.3. Obrázky, zvuky a ostatní zdroje z těchto dvou umístění lze snadno vkládat do kódu stránek prostřednictvím třídy `LinkBuilder`, konkrétně jeho metody *linkToResource*. Referenci na `LinkBuilder` lze získat z metody *getLinkBuilder* ve třídě `EnvironmentManager`.

Zatímco načítání zdrojů z prvních dvou lokací je poměrně přímočaré, v případě posledního zmíněného umístění je situace značně komplikovanější. Začlenění a načítání knihoven a zdrojů třetích stran je věnována samostatná sekce 3.2.2.

3.2.2 Knihovny a zdroje třetích stran

Ve složce se zdroji přibyla podsložka pro knihovny třetích stran, kam je nyní možné knihovny vkládat tak jak jsou, bez nutnosti jakékoliv úpravy. Jediným nutným krokem při začleňování je vytvoření nové implementace abstraktní třídy `ThirdPartyResourceLoader`, ve které se v rámci metody `load` definuje, které zdroje se mají jakým způsobem zavádět. Tato třída musí být pojmenována stejně, jako nadřazená složka (obvykle název knihovny) a za názvem musí být ještě extenze `„Loader"`. Obrázek 3 ukazuje příklad struktury začleněné knihovny Twitter Bootstrap.



Obrázek 3: Struktura začleněné knihovny Twitter Bootstrap

Zdrojový kód 1 znázorňuje, jakým způsobem lze definovat zaváděné zdroje z této knihovny uvnitř zavaděče `BoostrapLoader`. Klíčovou úlohu zde sehrává metoda *addResource*, zděděná od rodičovské třídy, která registruje zdroje do kolekcí, které jsou pak kdykoliv dostupné přes `ResourceManager`, jenž je podrobněji popsán v sekci 3.2.3. Metodě *addResource* stačí předat pouze zvolený název pro daný zdroj, cestu k souboru či složce, typ (kategorii) zdroje a volitelně lze také uvést příznak, jestli má být složka prohledávána rekurzivně, a navíc také prioritu.

```
1  class BootstrapLoader extends ThirdPartyResourceLoader
2  {
3      public function load()
4      {
5          $this->addResource("Bootstrap minimal JS",
6              "js/bootstrap.min.js",ResourceType::JS,
7              false, 8);
8          $this->addResource("Bootstrap minimal CSS",
9              "css/bootstrap.min.css",ResourceType::CSS,
10             false, 8);
11     }
12 }
```

Zdrojový kód 1: Příklad implementace zavaděče knihovny zdrojů

3.2.3 ResourceManager

V běžných případech užití není nutné s ResourceManagerem nijak manipulovat. Pokud však aplikace vyžaduje implementaci vlastního controlleru, který není založen na žádné ze tříd `SimpleViewController`, ani na `WorkspaceViewController` (viz kapitola 3.4), lze jeho prostřednictvím získat kolekce všech relativních cest k javascriptovým zdrojům nebo zdrojům s kaskádovými styly. Pro tyto účely slouží funkce *getResourcePathCollection*, která přebírá jako argument požadovaný typ zdroje. Pokud je potřeba zdroje nějakým způsobem dále filtrovat nebo třídit, lze získat i kolekci všech zdrojů se všemi metadaty, využitím metody *getResources*. Zdroje jsou v tomto případě reprezentovány objektem třídy `Resource`. Instanci ResourceManageru lze získat z tovární metody *getResourceManager* ve třídě `EnvironmentManager`.

3.2.4 Automatická minifikace

V produkční režimu každé aplikace je vhodné využít automatickou minifikaci zdrojů, která zajistí, že každý javascriptový soubor, nebo soubor s kaskádovými styly je zredukován na co možná nejmenší velikost (odstraněním mezer, zalomení řádků, a zkrácením názvů proměnných) a zároveň jsou všechny zdroje daného typu zkombi-

novány do jediného souboru. Tento mechanizmus tak redukuje množství jednotlivých požadavků a objem přenášených dat mezi webovým serverem a prohlížečem.

Využita je zde knihovna Minify, které jsou veškeré soubory zdrojů předány ke zpracování. Poskytování minifikovaných zdrojů lze nastavit konfigurační konstantou MINIFY_RESOURCES. Ručně je možné minifikované zdroje daného typu získat z `ResourceManageru` metodou *getMinifiedResourcesPath*.

3.3 Komponenta MessageBus

Komponenta `MessageBus` zajišťuje výměnu systémových (aktualizačních) zpráv mezi aplikací a připojenými uživateli i mezi uživateli samotnými. Lze jí použít i například pro jednoduchou realizaci rozhraní pro instantní zprávy (chat). Přenášené zprávy jsou kódovány do formátu JSON, který je velmi jednoduchý a například oproti XML vyniká nízkou režií. Do frameworku byla začleněna v rámci jmenného prostoru `Phem\Libraries\MessageBuss`. Zároveň byly rozšířeny části jmenného prostoru `Phem\Environment`, zejména statická třída `Application`, která nyní disponuje funkcemi pro zasílání zpráv. Páteř této komponenty tvoří kromě třídy `MessageBus`, reprezentující samotnou sběrnici zpráv, také dvě abstraktní třídy `Pusher` a `Puller`. Zatímco `Pusher` zajišťuje vkládání zpráv na sběrnici, `Puller` slouží pro jejich vyzvedávání a předávání adresátům. K dispozici je několik základních implementací, které vyplývají z podporovaných běhových režimů (viz. sekce 3.3.1). Je však možné v případě potřeby přidat vlastní implementaci.

3.3.1 Běhové režimy

K dispozici je několik běhových režimů, které se odvíjejí od použité metody pro notifikaci okna prohlížeče ze strany serveru. Běhový režim lze vybrat pomocí konfigurační konstanty MESSAGEBUS_MODE, která byla přidána do konfiguračního souboru aplikace. Přehled režimů je zobrazen v tabulce 3.

Tabulka 3: Přípustné hodnoty pro MESSAGEBUS_MODE

Hodnota	Popis
WebSocket	zprávy jsou přenášeny prostřednictvím permanentního websocketového spojení
Comet	pro přenos zpráv je použita metoda comet (vyčkávající Ajaxový dotaz)
Both	kombinace obou výše uvedených metod
Disabled	přenos notifikačních zpráv je zakázán

Režim Websocket

Režim WebSocket je nejefektivnější. Disponuje nízkou latenci při doručování zpráv a klade nejnižší nároky na objem přenesených dat. V tomto režimu je nutné spustit samostatný skript `socket.run.php`, který websocketová spojení obsluhuje. Tento skript je též napsán v PHP a využívá součásti frameworku. Umístěn byl přímo do kořenového adresáře. V unixovém prostředí je vhodné, aby byl spuštěn pomocí obalového shellového skriptu `socket.run.sh`, který zajistí předání PHP skriptu k interpretaci PHP procesovému manažeru nebo HHVM interpreteru, čímž se zajistí přístup ke stejné sdílené paměti a aplikačním proměnným, jako přímo z webového serveru (podrobněji o této problematice viz. kapitola 6). Také je vhodné využít Supervisor, což je démon, který dohlídne na to, aby skript zůstal běžet, případně se při selhání opětovně spustil.

Základ skriptu tvoří smyčka událostí z knihovny React, se kterou je svázán naslouchající websocketový server Ratchet a obvykle také puller naslouchající pro pro příjem zpráv ze strany webového serveru. Na základě událostí jsou volány příslušné metody nad instancí třídy `MessageBus`, která sběrnici zpráv reprezentuje. Prostřednictvím konfigurační konstanty WEBSOCKET_PULLER_MODE lze vybrat konkrétní implementaci způsobu předávání zpráv socketovému serveru ze serveru webového. Přehled dostupných implementací je zobrazen v tabulce 4.

Tabulka 4: Přípustné hodnoty pro WEBSOCKET_PULLER_MODE

Hodnota	Popis
ReactSocket	jednoduchá implementace fronty zpráv pomocí socketového serveru z knihovny React
ZMQ	využití knihovny ZeroMQ
Disabled	předávání zpráv websocketovému serveru je zakázáno

Využití ZeroMQ je univerzálnější řešení, umožňující snadné začlenění do komplexnějšího distribuovaného systému. Nevýhoda této volby spočívá v nutnosti instalovat na běhový server knihovnu libzmq a podporu této knihovny pro jazyk PHP (php_zmq modul). Podpora ZMQ pro HHVM navíc stále chybí, přestože se v průběhu dubna objevila na serveru github portovaná verze php_zmq pro HHVM, její použitelnost je stále spíše v mezích experimentálních účelů. Předávání zpráv pomocí ReactSocket implementace funguje spolehlivě i s HHVM.

Pro konfiguraci rozhraní a portů, na kterých má websocketový server i puller naslouchat, lze použít tyto konfigurační konstanty:

- WEBSOCKET_BIND_ADDR
- WEBSOCKET_PORT
- WEBSOCKET_PULER_BIND_ADDR
- WEBSOCKET_PULER_PORT

Podrobnějším informacím a doporučením ohledně konfigurace běhové platformy se věnuje kapitola 6.

Režim Comet

Pokud běhová platforma neumožňuje nasazení technologie WebSocket, nebo je vyžadována stoprocentní podpora napříč dostupnými prohlížeči, lze použít režim Comet. Je však nutné počítat s vysokými nároky na propustnost webového

serveru, obzvláště v případě použití komponenty UI, nebo jiných aplikačních součástí, náročných na objem asynchronně přenášených dat. Webový server by měl v tomto případě nabízet dostatek volných vláken či procesů pro obsluhu příchozích požadavků. Nevýhodou toho řešení je latence při detekci ukončení spojení a především závislost na modulu php_pcntl, který je dostupný pouze na unixových systémech. Na rozdíl od režimu WebSocket, kde je využita fronta zpráv a naslouchající puller, zde jsou zprávy při odeslání (v rámci CometPusheru) ukládány mezi aplikační proměnné, kde vyčkávají na vyzvednutí. Jednotlivá spojení jsou realizována pomocí ajaxového požadavku, jehož obsluhující PHP skript vloží mezi aplikační proměnné identifikátor daného spojení spárovaný s identifikátorem procesu či vlákna, pod kterým běží a dále na serveru vyčkává (v rámci CometPulleru) na signál SIGUSR1. Tento signál je procesu zaslán bezprostředně po vložení zprávy mezi aplikační proměnné, prostřednictvím funkce pcntl_sigtimedwait. Po vyzvednutí všemi adresáty je zpráva z aplikačních proměnných odstraněna. Pokud k vyzvednutí všemi adresáty nedojde, je zpráva odstraněna po uplynutí lhůty, stanovená konfigurační konstantou COMET_MESSAGE_TIMEOUT. Toto řešení je tedy zároveň náročnější na alokovanou paměť a není příliš vhodné pro současný přístup velkého množství uživatelů.

Režim Both

V některých případech může být výhodné použití běhového režimu Both, který lze označit za režim kompatibility. Nabízí Webosocketové spojení pouze prohlížečům, které je podporují. Ostatní prohlížeče mohou získávat notifikace prostřednictvím Comet principu. MessageBus sám zajistí interoperabilitu mezi těmito dvěma přístupy. Režim Both je dostupný pouze na unixových platformách, stejně jako režim Comet.

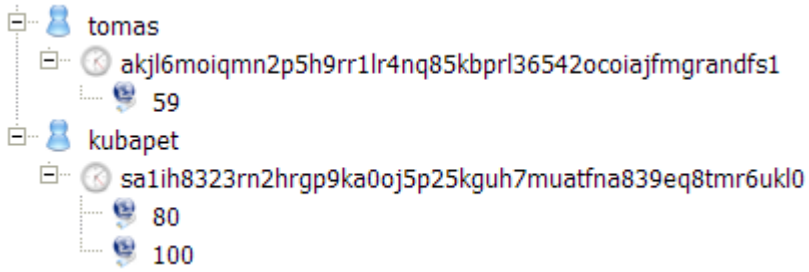
3.3.2 Adresace

MessageBus nabízí sedm metod pro různé typy adresace příjemců zprávy. Tyto metody byly doplněny do statické třídy Application. V tabulce 5 je uveden seznam metod se stručným popisem.

Tabulka 5: Varianty metody notify pro různé typy adresace

Název metody	Cíl zprávy
notifyAll	všechna aktivní spojení
notifyUsers	všechna aktivní spojení všech uvedených uživatelů
notifyUser	všechna aktivní spojení konkrétního uživatele
notifySessions	všechna aktivní spojení všech uvedených sezení
notifySession	všechna aktivní spojení konkrétního sezení
notifyConnections	všechna vybraná aktivní spojení
notifyConnection	konkrétní aktivní spojení

Pro použití metod *notifyUser*, *notifyUsers*, *notifySession* a *notifySessions* je nutné mít zapnutu podporu pro ukládání aplikačních proměnných. V opačném případě tyto metody generují výjimku. Pokud jsou aplikační proměnné povoleny, pod klíčem ve tvaru Phem.[jmenný prostor aplikace].Users se vytváří strom aktivních uživatelů, sezení a spojení, na základě kterého je možné takto specifickou adresaci realizovat. Obrázek 4 je příkladem grafické reprezentace takto vytvořeného stromu.



Obrázek 4: Přílad výpisu aktivních uživatelů, sezení a spojení

3.3.3 Vytvoření a odeslání zprávy

Zprávu je možné vytvořit ručním sestavením JSON řetězce (např. pomocí Twig šablony), vhodnějším způsobem je však použití univerzálního objektu zprávy. Tento objekt stačí inicializovat, naplnit daty a voláním příslušné statické metody *notify** třídy `Application` odeslat požadovaným adresátům. Metodám *notify** lze

předat přímo výsledný řetězec nebo instanci univerzálního objektu zprávy (třídy `Message`). V případě, že předávaná zpráva není validní JSON řetězec, ani instance třídy `Message`, je vygenerována výjimka. Odesílání zpráv je možné v kterékoliv fázi zpracování požadavku webovým serverem. V režimu Websocket je možné zprávy přeposílat ostatním uživatelům přímo (bez průchodu přes webový server) pokud je hodnota konfigurační konstanty WEBSOCKET_FORWARD_MESSAGES nastavena na Enabled. Příklad vytvoření zprávy a rozeslání na všechna aktivní spojení zachycuje zdrojový kód 2.

```
1 $msg = new Message();
2 $msg->setFromUsr($user->getUsername());
3 $msg->setSubject('User is online');
4 $msg->setText($user->getUsername() . ' is now online');
5 try { Application::notifyAll($msg); }
6         catch (ApplicationException $ex){ ... }
```

Zdrojový kód 2: Příklad vytvoření a odeslání zprávy

3.3.4 Argumenty zpráv

Ke každé zprávě lze přiložit kolekci rozšiřujících argumentů. Každý atribut pak pomocí atributů actionUrl, actionTarget a actionType definuje určitou akci. Atribut actionUrl definuje zdrojovou adresu, actionTarget cílový prvek a actionType určuje typ akce, který může nabývat jedné ze čtyř hodnot: append, prepend, replace nebo remove. Pomocí toho jednoduchého mechanizmu je možné snadno aktualizovat obsah běžné stránky, která není řízena WorkspaceViewControllerem (viz. kapitola 3.4, na základě událostí na serveru).

3.4 Komponenta UI

Pravděpodobně nejrozsáhlejší a nejzásadnější realizovanou komponentou je komponenta UI. Jejím cílem je poskytnout veškeré zázemí pro již několikrát zmiňovaný desktopově orientovaný přístup k vývoji uživatelských rozhraní. Umožňuje snadným způsobem definovat komplexní rozhraní, založená na formulářových oknech, obsahující nejrůznější interaktivní prvky. Přitom není nutné psát prezentační vrstvu samostatně, ve značkovacím jazyce HTML a doplňovat ji o kaskádové styly a podpůrné javascripty pro skriptování na straně prohlížeče, jak je zvykem při tvorbě běžných webových stránek. Tyto možnosti však nejsou odepřeny a pokud je to nutné, lze tyto techniky kombinovat. Dále je touto komponentou nabídnuta funkcionalita pro zajištění interakce a kooperace připojených uživatelů. Celé rozhraní nebo jednotlivá okna lze na různých úrovních sdílet a sledovat tak například činnost jiného uživatele v přímém přenosu, respektive do jeho činnosti lze i zasahovat. Možnosti sdílení jsou podrobněji popsány v sekci 3.4.3.

Komponentu tvoří `WorkspaceViewController`, kolekce tříd pro jednotlivé vizuální prvky, společně s několika doplňkovými třídami, dále pak kolekce Twig šablon, mapovací anotace a podpůrné javascripty. Tyto součásti jsou podrobněji popsány v následujících několika podsekcích.

Základním principem použití je vytváření potomků standardních součástí prostřednictvím dědičnosti. V první fázi je nutné vytvořit vlastní třídu vycházející z WorkspaceViewControlleru, která bude na základě požadovaných podmínek poskytovat danou pracovní plochu (workspace). Tento workspace je nutné opět realizovat jako třídu dědící z univerzální třídy *Workspace* a rozšířit ji o okna a widgety, které bude ve výchozím stavu obsahovat. S okny a widgety je to stejné, definují se také pomocí rozšíření daných univerzálních tříd. Situace se mění až v případě interaktivních prvků, které tvoří obsah oken a widgetů. Tyto prvky lze použít přímo, bez nutnosti dědění. Dědičnost je zde však užitečná, pokud je potřeba předefinovat jejich funkcionalitu nebo je nějakým způsobem rozšířit.

3.4.1 WorkspaceViewController

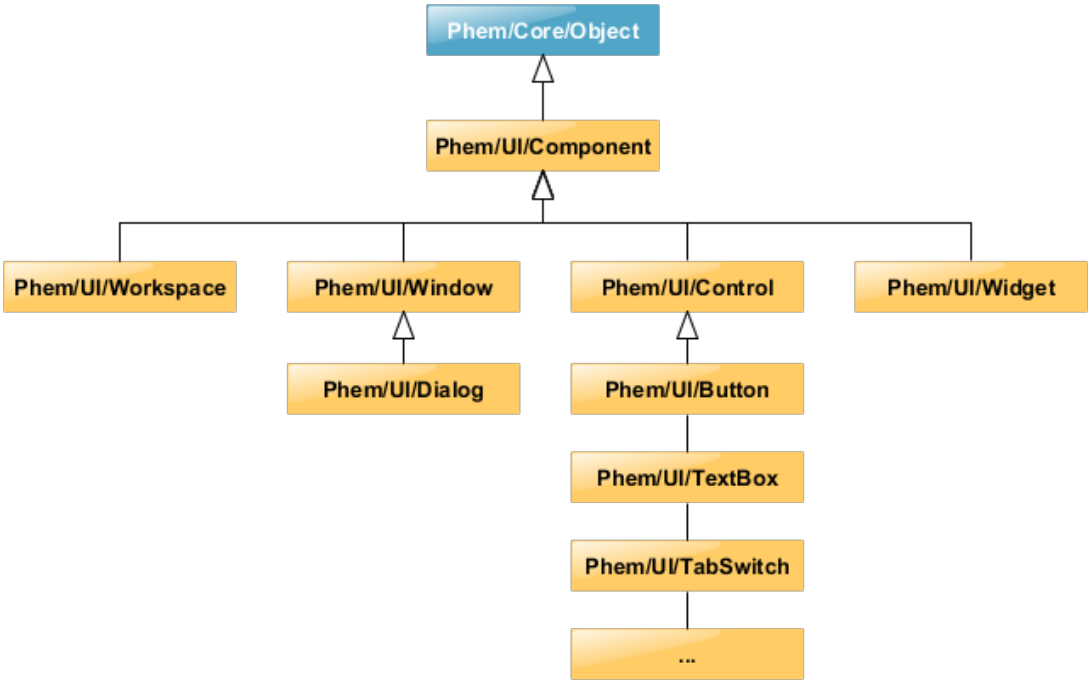
`WorkspaceViewController` do jisté míry vychází svým konceptem ze `SimpleViewControlleru`, který je učen pro zobrazování stránek pomocí Twig šablon. Není z něj však odvozen dědičností, jelikož některé kroky jsou zde záměrně prováděny v jiném pořadí, či zcela vynechány. Jeho primárním úkolem je připravit podklady pro vykreslení workpsacu na základě logiky definované vývojářem. Dále disponuje metodami *loadWorkspace* a *persistWorkspace*, které slouží k načtení resp. uložení workspacu z resp. do příslušného úložiště, daného nastaveným rozsahem platnosti. Podrobněji je tento proces popsán v sekci 3.4.3. Vyjma zmíněných funkcí je zde dále zajištěna obsluha událostí, vyvolaných v okně prohlížeče. K těmto účelům slouží metody *onEvent* a *onStrongEvent*, jejichž funkcionalita je přiblížena v sekci 3.4.4.

Úkolem vývojáře aplikace, při využití komponenty UI je pouze vytvoření potomka tohoto controlleru, ve kterém lze nadefinovat libovolné metody, implementující požadovanou aplikační logiku. V každé této metodě je nutné pomocí rodičovské metody *setWorkspace* nastavit instanci na workspace, který má být vykreslen. Pokud tato instance nastavena není, je při pokusu o načtení stránky vygenerována výjimka. Pro účely získání instance je vhodné použít výše zmíněnou metodu *loadWorkspace*.

Pro samotné vykreslení workspacu je použít `SimpleView`, který vykresluje stránky pomocí Twig šablon. Kořenovou šablonou je zde `Workspace.twig`, která v případě potřeby vkládá šablony pro obsažená okna a widgety a ty následně vkládají další šablony pro jednotlivé interaktivní prvky.

3.4.2 Vizuální komponenty

Vizuální komponentou se rozumí vše, co je možné vykreslit v rámci workspacu, i workspace samotný. Jsou to všechny objekty, reprezentované třídami odvozenými ze třídy `Component` ve jmenném prostoru `Phem\UI`. Obrázek 5 znázorňuje základní hierarchii vizuálních komponent.



Obrázek 5: Přehled zákldních tříd vizuálních komponent

Každá vizuální komponenta, která není zcela abstraktní, jako například `Component` nebo `Control`, je složena z páru třída – šabolna. Zatímco třída definuje vlastnosti, události a chování, šablona určuje jakým způsobem bude komponenta graficky reprezentována v okně prohlížeče.

Speciálním případem je vizuální komponenta `Workspace`, která obsahuje kontejner oken a dále komponenty `Widget`, `Window` a zněj dvozený `Dialog`, které mohou obsahovat další vnořené prvky odvozené z typu `Control` (interaktivní prvek).

3.4.3 Ukládání a sdílení instancí

Instance vizuálních komponent, kterým lze nastavovat rozsah platnosti (aktuálně pouze `Workspace` a `Window`), se uchovávají mezi aplikačními proměnnými, nebo mezi proměnnými sezení, a to včetně vizuálních komponent, které obsahují ve svém interním kontejneru.

Rozsahy platnosti existují čtyři a úzce souvisejí s možnostmi adresace v komponentě MessageBus:

- APPLICATION – instance viz. komp. je sdílena mezi všemi uživateli aplikace, kteří mají příslušné oprávnění
- USER – instance viz. komp. je společná pro všechna sezení daného uživatele
- SESSION – instance viz. komp. je společná pro všechna spojení daného sezení
- CONNECTION – instance viz. komp. je přístupná pouze pro dané spojení

K persistenci vizuální komponenty stačí pouze reference na její instanci, ze které si metoda *persistWorspace* či *persistWindow* získá veškeré potřebné informace. V případě načítání metodami *loadWorkspace* resp. *loadWindow* je potřeba předat název požadované instance, název třídy a identifikátor aktuálního spojení. Tyto metody pak sestupně od celé aplikace směrem k aktuálnímu spojení procházejí úložiště a hledají vhodnou instanci. Pokud není nalezena, vytvoří se nová, která je okamžitě uložena, dle výchozího nastavení dané třídy. Okna, která mají jiný rozsah než nadřazený workspace jsou ukládána do úložiště samostatně a v rámci workspacu jsou při uložení reprezentována pouze pseudo-objekty, které jsou metodou *loadWorkspace* detekovány a nahrazeny zpět referencemi na správnou instanci okna.

Pokud je pro danou vizuální komponentu nastaven rozsah platnosti Application nebo User, instance je uchovávána mezi aplikačními proměnnými, tak aby byla přístupná ostatním uživatelům či z jiných sezení téhož uživatele. Pro rozsahy platnosti Session a Connection je použito úložiště proměnných sezení. Klíčování instancí v úložišti je založeno na názvu třídy, zvoleném instančním názvu a eventuálně v případě rozsahu platnosti User na identifikátoru uživatele, či v případě rozsahu Connection na identifikátoru spojení. Nastavení rozsahu workspacu nebo okna lze provést snadno v inicializaci dané vizuální komponenty, jak je naznačeno ve zdrojovém kódu 3.

```
1 ...
2 $this->setBackgroundColor("rgb(246,240,218)");
3 $this->setScope(ScopeType::APPLICATION);
4 ...
```

Zdrojový kód 3: Inicializace workspacu s nastavením rozsahu platnosti

3.4.4 Ošetřování událostí

Každá neabstraktní vizuální komponenta nabízí minimálně jednu událost, kterou lze ošetřit (reagovat na ní vykonáním nějakého skriptu). Veškeré události jsou reprezentovány kolekcí metod zpětného volání (callback metod). Pro každou definovanou událost komponenty se při jejím vykreslování šablonou nastaví ošetření odpovídající javascriptové události v příslušném HTML elementu, avšak pouze v případě, že daná kolekce není prázdná, tj. obsahuje-li alespoň jednu referenci na přiřazenou metodu zpětného volání.

```
1 ...
2 $this->textbox1->OnChange()->add(function($sender){
3     $this->textbox3->setText($sender->getText());
4 });
5 ...
```

Zdrojový kód 4: Ošetření události OnChange komponenty TextBox

Na straně javascriptu každá ošetřená událost volá metodu *onStrongEvent*, respektive její odlehčenou verzi *onEvent*. Tyto metody předávají pomocí ajaxového volání na server informace o události která nastala, včetně jejího typu, unikátního identifikátoru komponenty v rámci zvoleného okna či workspacu, identifikátoru spojení a

dalších podrobností. Liší se v tom, že *onStrongEvent* přikládá kompletní aktuální stav dané komponenty, kdežto *onEvent* posílá pouze jednu vybranou (hlavní) hodnotu. Pro jednoduché události typu změna textu v textovém poli komponenty TextBox je použita metoda *onEvent*, naopak v případě složitějších událostí, například při tažení oknem je použita metoda *onStrongEvent*. Při implementaci vlastních komponent je vhodné využít metodu *onEvent* na úkor metody *onStrongEvent* vždy, kdy je to možné, pro úsporu objemu přenesených dat.

V případě metody *onStrongEvent* je aktuální stav komponenty zasílaný na server reprezentovaný datovým objektem, serializovaným do formátu JSON. Odesílán však není celý DOM uzel reprezentující v danou chvíli danou komponentu, jelikož obsahuje velké množství metadat, která je zbytečné přenášet. Vyselektován je proto pouze typ elementu, přímý textový obsah a hodnoty atributů. Pokud je vizuální komponenta složena z více do sebe vnořených elementů, jsou samozřejmě do stavového objektu zakomponovány rekurzivně i tyto. O sestavení odesílaného stavového objektu se stará javascriptová funkce elementToObject, nacházející se v souboru `phemUIStateObject.js`, který je umístěn ve složce s javascriptovými zdroji frameworku.

Voláním výše zmíněného ajaxového požadavku se na serveru zavolá příslušná stejnojmenná metoda – *onEvent* či *onStrongEvent*. Zde se opět načte příslušný workspace či okno z úložiště a na základě předaného identifikátoru se v něm vyhledá komponenta, která událost vyvolala. V případě metody *onEvent* se pouze dosadí nová hodnota pomocí metody setValue nad touto komponentou. V případě metody *onStrongEvent* se znovu z JSON řetězce sestaví ekvivalentní PHP objekt třídy *stdObject* a pomocí reflexe se načtou anotace nad atributy třídy této komponenty. Tyto anotace určují, jakým způsobem má být hodnota daného atributu aktualizována ze stavového objektu. Dostupné anotace pro komponentu UI jsou popsány v tabulce 6. Přepsány jsou pouze ty hodnoty, které se liší, což umožňuje sestavovat aktualizační list, jak je popsáno dále v sekci 3.4.5. Poté, co je stav komponenty aktualizován, zavolají se všechny metody zpětného volání, registrované k dané události.

Tabulka 6: Přehled anotací pro mapování vlastností vizuálních komponent

Název anotace	Význam
@DomAttr	Hodnota DOM atributu
@StyleAttr	Hodnota CSS atributu uvnitř DOM atributu style
@ContentAttr	Textový obsah elementu
@DomPath	Cesta k elementu, na kterém je vlastnost uplatněna
@NoUpdateAttr	Neaktualizovatelná vlastnost

3.4.5 Princip aktualizace uživatelských rozhraní

Každá vizuální komponenta má své atributy nastaveny jako privátní (nedostupné vně komponenty). Pro vnější přístup je použito tradiční zapouzdření pomocí metod "set"a "get", běžně používané v rámci objektově orientovaného programování (OOP). Zde je však navíc využito volné typovosti PHP, které nabízí možnost zachytit volání neexistující funkce pomocí volání univerzální magické metody __call. Všechny set metody nad atributy vizuálních komponent jsou nastaveny jako protected (dostupné pouze zevnitř a v rámci potomků), nicméně pokud jsou zavolány zvnějšku, volání je odchyceno právě metodou __call, která vytvoří poznámky o prováděné změně do aktualizačního listu a následně teprve zavolá skutečnou "set"metodu. Při implementaci vlastních vizuálních komponent je tedy nutné dbát na to, aby byly metody set nastaveny jako protected. V opačném případě nebude aktualizace odpovídajících atributů funkční.

V závěru zpracování každé události ve funkcích *onEvent* a *onStrongEvent* je z daného workspacu či okna převzat aktualizační list. Tento list je doplněn o identifikátor spojení, ze kterého byla tato událost vyvolána, a také o číslo aktuální verze okna či workspacu. Během manipulace s daným oknem nebo workspacem je jeho instance v úložišti aplikačních proměnných nebo proměnných sezení uzamčena, aby nedošlo k hazardu způsobeným konkurenčním přístupem. Po uvolnění zámku a zapsání dané instance zpět do úložiště je odeslána notifikace s přiloženým změnovým lis-

tem všem dotčeným účastníkům. U každého z nich je na straně javascriptu tato změna adekvátně zpracována a části rozhraní odpovídající jednotlivým položkám aktualizačního listu jsou překresleny. Položky aktualizačního listu (změny) mohou být sedmi různých typů, které jsou přiblíženy v tabulce 7.

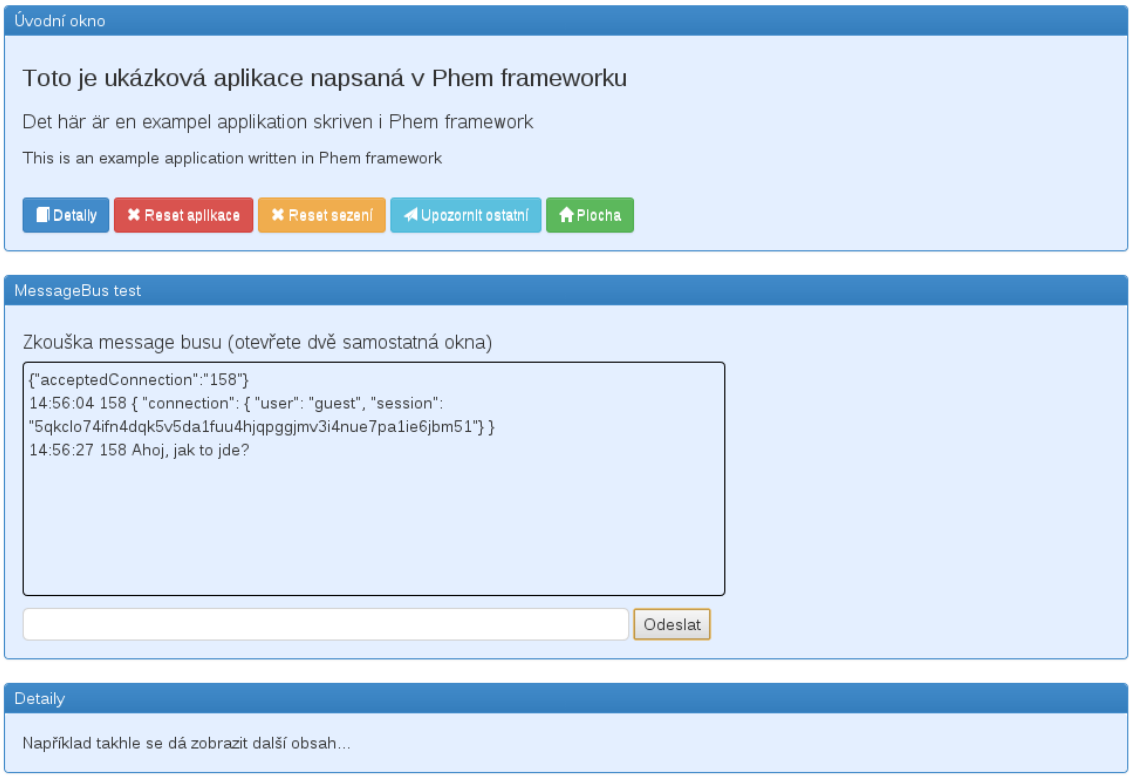
Tabulka 7: Typy aktualizací uživatelského rozhraní

Kód	Označení	Popis
1	APPEND	vložení do nadřazeného elementu (na konec)
2	PREPEND	vložení do nadřazeného elementu (na začátek)
3	REMOVE	odstranění
4	REPLACE	nahrazení
5	CHANGE_ATTR	změna atributu elementu
6	CHANGE_STYLE	změna stylového atributu
7	UPDATE_CONTENT	změna textového obsahu

První tři uvedené typy změn nastávají v případě dynamického přidání či odebrání vizuální komponenty do resp. z kontejneru nadřazené komponenty. Ostatní typy změn nastávají při aktualizaci stavu dané vizuální komponenty (změna hodnot atributů) a vyplývají z anotací uvedených nad atributy (viz tabulka 6).

4 Demonstrační aplikace

Pro demonstraci funkcionality doplněných komponent byla vytvořena demonstrační aplikace nazvaná `PhemDemo`. Tato aplikace je rozdělena na dvě části. První z nich je postavená na SimpleViwControlleru a ukazuje základní činnost MessageBusu pomocí chatovacího okna. Druhá z nich využívá `WorkspaceController` a ukazuje, jakým způsobem lze vytvořit a inicializovat několik formulářových oken s různými interaktivními prvky, jak lze zachycovat události a nastavit rozsah platnosti pracovní plochy na sezení či celou aplikaci. Na obrázku 6 je zachycena úvodní stránka.



Obrázek 6: Úvodní stránka demonstrační aplikace

Obrázek 7 zahycuje dva formuláře, ze druhé části demonstrační aplikace. Událost *OnChange* (tedy změna hodnoty) textového pole se jménem je ošetřena stejným

způsobem, jak bylo znázorněno v ukázkovém zdrojovém kódu 4. Pokud uživatel mění jméno, předvyplňuje se mu tak automaticky i přezdívka. Pokud má stejný workspace navíc otevřený v několika záložkách, oknech, či prohlížečích vidí prováděné změny v přímém přenosu i tam.

Obrázek 7: Formulářová okna z demonstrační aplikace

V demonstrační aplikaci je navíc nastaveno sdílení pozic oken. Při jejich přemísťování v rámci pracovní plochy se změna pozice projevuje také v přímém přenosu ve všech ostatních spojeních, kde je dané okno viditelné.

Ukázáno je dále ještě několik dalších možností jak dynamicky aktualizovat obsah na základě různých událostí, nicméně předmětem této práce není jejich detailní popis. Demonstrační aplikace je vhodným výchozím bodem pro zahájení nového projektu ve frameworku Phem. Pouhým prozkoumáváním funkcí a souběžným sledováním zdrojového kódu lze snadno pochopit základní principy vývoje v tomto frameworku.

5 Náměty pro další rozšíření

V následujících několika podsekcích jsou vypíchnuty nejpodstatnější náměty na rozšíření realizovaných komponent. Tato rozšíření budou nejspíše v dohledné době pro zefektivnění vývoje realizována i přesto, že jsou nad rámec této práce.

5.1 Značkovací jazyk pro UI

Definice uživatelského rozhraní prostřednictvím dědičnosti a rozšiřování tříd pro pracovní plochu, formulářová okna nebo interaktivní komponenty je sice velice univerzální a jednoduché, ale zbytečně výřečné a v případě komplexnějších rozhraní může ztrácet přehlednost. Dobrým řešením pro tento případ by bylo zavedení podpory nějakého značkovacího jazyka, založeného pravděpodobně na XML. V tomto jazyce by se dalo přehledným způsobem nadefinovat jaké vizuální komponenty budou v připravované aplikaci použity, co budou obsahovat a zároveň inicializací hodnot atributů určit jak budou ve výchozím stavu vypadat. Podobný značkovací jazyk je například XAML, který je již znám z Windows Presentation Foundation.

5.2 Zamykání vizuálních komponent

V aktuální verzi lze sice vizuální komponenty sdílet na všech možných úrovních, mezi libovolným počtem uživatelů, ale nelze zajistit, aby některou z nich směl v danou chvíli obsluhovat pouze jeden z nich a pro ostatní byla pouze ke čtení (například použitím atributu disabled nad prvky dané komponenty). V tomto případě by však bylo nutné vyřešit také uvolňování zámků v případě ztráty spojení navrhnout mechanizmus, jakým půjde o zámek požádat server nebo případně upozornit jiné spojení, které potřebný zámek v daný okamžik vlastní.

5.3 Propagace událostí přes socketové spojení

Socketový server nemá přístup k úložišti proměnných jednotlivých sezení, které je v PHP standartně realizováno pomocí souborů. Každému sezení odpovídá právě jeden soubor, jehož názvem je identifikátor sezení. Teoreticky by to možné bylo, ale znamenalo by to načítat soubory sezení ručně a navíc s výlučným zámkem pro čtení, což by mohlo zdržet obsluhu běžných požadavků na webový server. Jelikož jsou některé instance vizuálních komponent uchovány právě v úložišti proměnných sezení, není možné události propagovat přes persistentní komunikační kanál socketovému serveru, protože by je nedokázal uložit. Všechny události v rámci pracovní plochy založené na komponentě UI jsou proto zasílány přímo na webový server prostřednictvím ajaxového požadavku. Pokud by se podařilo využívat pro tyto notifikace persistentní kanál, efektivita by se ještě zvýšila. Možností by bylo například ukládat kmen instancí celý mezi proměnné aplikace. Tím by se však zase zvýšily nároky na propustnost tohoto úložiště. Výhodou je však skutečnost, že implementaci úložiště aplikačních proměnných lze snadno nahradit.

5.4 UI komponenta pro instantní formulář

Framework Phem disponuje nástrojem `AnnotationDrivenFromBuilder`, který slouží pro tvorbu formulářů přímo z datového modelu. Kteroukoliv třídu tak lze doplnit o anotace, které zajistí vygenerování formuláře pro editaci či tvorbu nových instancí dané třídy. Dokonce je možné takto „oanotovat"komplexní strukturu tříd obsahující reference a dědičnost a referované třídy nebo rozšíření nechat generovat jako samostatnou podskupinu (fieldset) požadovaného formuláře. V rámci této práce však nebyla vytvořená vizuální komponenta, kompatibilní s komponentu UI, která by toto generování formulářů zapouzdřovala a umožnila jeho snadné použití v novém prostředí emulující desktopové programování.

6 Běhová platforma

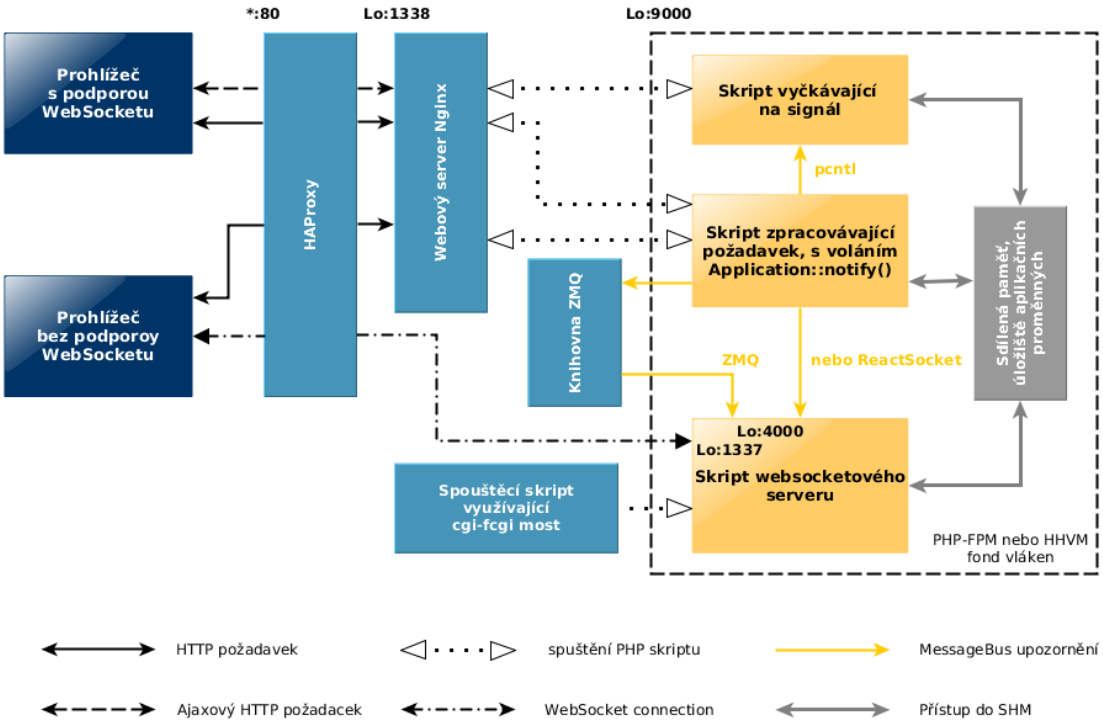
Přestože framework Phem je navržen tak, aby byl provozuschopný na jakékoliv PHP platformě, jeho výkonnost se v závislosti na volbě platformy značně liší. Dalším aspektem vedle výkonu je bezpečnost a podpora všech technologií, které framework nabízí. Zejména z těchto důvodů je zde popsána doporučená běhová platforma a na kompaktním disku jsou k ní přiloženy ukázkové konfigurační soubory pro její jednotlivé součástí.

6.1 Doporučená konfigurace serveru

Pro použití MessageBusu v režimu WebSocket nebo Both je vhodné předřadit před webový server nějaký proxy server, který umožní vytvořit persistentní spojení mezi klientem a socketovým serverem na stejném portu, na který chodí běžné HTTP požadavky. Vhodným kandidátem je například HAProxy. Toto nastavení umožní, že z vnějšku bude na serveru otevřený pouze jediný port (obvykle standartní HTTP port 80), což zároveň zvyšuje bezpečnost celého serveru a zároveň umožňuje bezproblémový průchod websocketového provozu přes firewally firemních sítí, které často blokují odchozí komunikaci na jiné porty.

Doporučit lze webový server Nginx, který pro dobrou správu systémových prostředků, přehlednou konfiguraci a vyšší propustnost v poslední době získává na popularitě na úkor velmi rozšířeného Apache. Webový server může v případě použití předřazeného proxy serveru naslouchat na libovolném portu a pouze na lokální smyčce. Proxy server mu bude požadavky takto lokálně předávat. Případně, pokud server běží na Linuxové či Unixové platformě, lze pro tento účel využít i Unixové sockety, pokud takovou možnost webový server i proxy server nabízejí. Konfiguraci Nginxu výborně popisuje kniha Mastering Nginx (Aivaliotis, 2013).

Interpretaci PHP skriprů je vhodné zajistit pomocí fondu vláken poskytovaného PHP FastCGI Managerem nebo platformou HHVM. V obou případech je potřeba opět nastavit naslouchání na lokální smyčce na zvoleném portu. Pokud je na serveru více webů, je dobré, z hlediska bezpečnosti, aby každý měl vlastní fond vláken, spuštěný pod různými systémovými uživateli. Použití php interpreteru ve formě modulu do webového serveru Apache nelze doporučit ani z hlediska rychlosti, ani z hlediska bezpečnosti. Při nastavování poskytovatele fondu vláken pro zpracovávání PHP skriptů jde třeba nastavit s rozvahou maximální počet paralelních vláken obsluhující požadavky. Při příliš nízkém počtu se může stát, že budou požadavky čekat na vyřízení frontu. Naopak při větším počtu může dojít k zahlcení serveru.



Obrázek 8: Přehledové schéma doporučené platformy

Další konfiguraci je třeba učinit již zcela na základě zvoleného režimu MessageBusu. Pokud je zvolen režim Websocket nebo režim Both, je nutné spouštět websocketový server samostatným skriptem.

tt socket.run.php. Tento skript běží permanentně a je tedy vhodné aby běžel na po-

zadí, jako démon či služba. Pro tento účel lze použít například nástroj Supervisor. Pro použití WebSocketu v kombinaci s komponentou UI je nutné, aby socketový server měl přístup k proměnným aplikace. To lze zajistit tak, že skript který tento server spouští je pomocí CGI-FastCGI mostu spuštěn nad stejným fondem PHP vláken, nad kterým běží primární webový server. Pro tyto účely slouží obalový shellový skript `socket.run.sh`.

6.2 Kompatibilita

Jak již bylo řečeno, framework je navržen tak, aby fungoval na všech PHP platformách. Použití některých částí však zahrnuje jistá omezení:

- V případě použití fronty zpráv implementované pomocí ZMQ je nutné mít v systému nainstalovánu knihovnu libzmq.
- Režimy Comet a Both pro MessageBus jsou dostupné pouze na Linuxových ču Unixový platformách, protože tyto režimy využívají funkcionalitu PHP modulu pcntl, který je platformě závislý.
- Použití ZMQ není možné společně s HHVM. Problematický článek zde tvoří knihovna React, která zatím není s platformou HHVM zcela kompatibilní. Na serveru GitHub se však již objevila nová verze knihovny, která by tuto kompatibilitu měla vyřešit

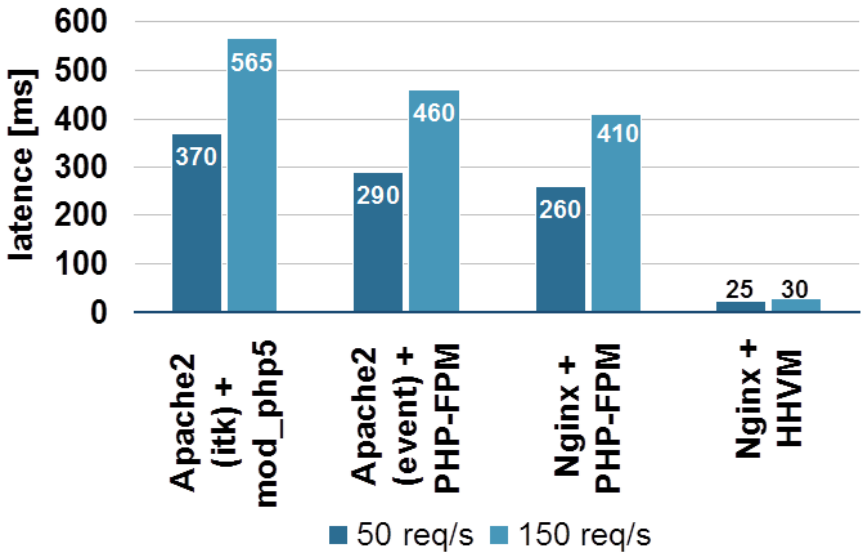
6.3 Výkon

Zejména komponenta UI využívá hojně reflexi a provádí časté přístupy do sdílené paměti, kde se nachází úložiště aplikačních proměnných. Tyto operace se navíc provádějí v četných aktualizačních požadavcích, například při tažení okna se jich v prohlížeči Google Chrome vygeneruje v průměru 50 za sekundu. V rámci této práce byl proveden výkonnostní test pro různé běhové platformy (resp. kombinace webového serveru a PHP interpretrů), založený právě na emulaci tažení okna.

Testovány byly tyto čtyři platformy:

- Webový server Apache2 s PHP5 interpretrem jako modulem
- Webový server Apache2 využívající PHP-FPM interpretr přes FastCGI rozhraní
- Webový server Nginx využívající PHP-FPM interpretr přes FastCGI rozhraní
- Webový server Nginx využívající HHVM interpretr přes FastCGI rozhraní

Pro každou platformu byly provedeny dva zátěžové testy. První z nich emuloval tažení okna jedním uživatelem a druhý emuloval tuto stejnou akci prováděnou třemi uživateli ve stejný okamžik. Na obrázeku 9 je vyobrazen graf s výsledky testu. Graf zobrazuje průměrnou dobu čekání (latenci) na vygenerování odpovědi při dané zátěži. Výkonnostní převaha platformy HHVM je evidentní.



Obrázek 9: Výkonnostní srovnání běhových platforem

Závěr

Výsledkem této práce je rozšíření frameworku Phem o čtyři hlavní komponenty a několik dalších součástí, podporujících jejich funkcionalitu. Pro vývojáře aplikací byla připravena především možnost snadného zasílání asynchronních zpráv napříč celou aplikací a také byl nabídnut nový způsob vývoje vizuální části, který vychází z principů desktopového programování. Rozšíření a provedené změny navíc nijak neovlivňují kompatibilitu se stávajícími aplikacemi, které tak zůstanou plně funkční i po aktualizaci na novou verzi frameworku.

Otestováno bylo několik běhových platforem, které byly srovnány zejména z výkonnostního hlediska. Velice nadějně vypadá nová technologie HHVM s jazykem Hack, proto byl kladen velký důraz na zavedení kompatibility a zprovoznění frameworku nad touto technologií. Problém bohužel zatím představuje špatná podpora ze strany serverových distribucí a absence některých PHP modulů, které bohužel zatím nebyly portovány. I toto sou důvody, které vedly k tomu, že v kapitole 6 jsou navrženy dvě varianty doporučené běhové platformy pro aplikace postavené nad frameworkem Phem. První z nich, založená na klasickém PHP interpretru využívajícím FastCGI rozhraní, se zaměřuje na širokou kompatibilitu a stabilitu. Druhá navržená varianta, založená na HHVM je zatím poněkud experimentálnějšího charakteru a je zaměřena na vysoký výkon. Řada expertů je však přesvědčena, že HHVM společně s jazykem Hack v dohledné době PHP zcela nahradí.

Vzhledem ke skutečnosti, že se podařilo komponentu MessageBus dopracovat do relativně stabilní podoby, která navíc v kombinaci s HHVM představuje oproti stávajícím řešením velikou úsporu systémových prostředků, je pravděpodobné, že po další sérii testů bude toto rozšíření frameworku připojeno zpět k jeho hlavní vývojové větvi. Tím vznikne prostor pro jeho nasazení v nově připravovaných verzích aplikací, které jsou na něm založeny. Například komponenta MessageBus však již byla

zpětně začleněna do aktuální verze frameworku v rámci administračního rozhraní internetového srovnávače společnosti Credit & Hypo Consult (Srovnavac24.cz), kde zajišťuje notifikaci operátorů o aktuálních událostech na webu a požadavcích klientů.

Na závěr bych podotkl, že bych rád do vývoje frameworku zapojil více vývojářů. Zvažuji zveřejnění kódu na serveru GitHub, nicméně tento krok nelze uspěchat. Je potřeba připravit informační stránky s prezentací projektu a vytvořit sadu unit testů, bez kterých by vývoj nebyl efektivní.

Použitá literatura

Aivaliotis, D. *Mastering Nginx*. Packt Publishing Ltd, 2013. ISBN 9781849517454.

Berchet, V. HHVM and Hack – Can We Expect Them to Replace PHP? 2014. Dostupné z: <http://www.sitepoint.com/hhvm-hack-part-1/>.

Boden, C. cboden/Ratchet. *GitHub*. 2013. Dostupné z: <https://github.com/cboden/Ratchet>.

Bruksås, J.-H. – Evertsson, F. – Gustavsson, N. *Push-teknik på webben*. 2010. Dostupné z: <http://www.diva-portal.org/smash/record.jsf?searchId=5&pid=diva2:329630>.

Chen, L. – Banflavi, G. *Methodologies and Architecturefor the Implementation of a Web Application*. 2012. Dostupné z: <http://www.diva-portal.org/smash/record.jsf?pid=diva2:618769>.

Chris, P. *Pro PHP MVC*. Apress, 2012. ISBN 978-1-4302-4164-5.

Leggetter, P. Building Realtime Web Apps with PHP. *Web & PHP Magazine*. 2014. Dostupné z: <http://webandphp.com/BuildingRealtimeWebAppswithPHP-166928>. In the first part of this two part series I discussed the history of realtime web technologies and how using them directly in PHP may.

Lengstorf, J. – Leggetter, P. *Realtime Web Apps: With HTML5 WebSocket, PHP, and jQuery*. Apress, April 2013. ISBN 9781430246213.

Nyklíček, J. Webové aplikace pracující v reálném čase, May 2013. Dostupné z: <http://is.muni.cz/th/256349/fi_m/dp.pdf?info=1>.

Pimentel, V. – Nickerson, B. Communicating and Displaying Real-Time Data with WebSocket. *IEEE Internet Computing*. July 2012, 16, 4, s. 45–53. ISSN 1089-7801. doi: 10.1109/MIC.2012.64.

Puranik, D. – Feiock, D. – Hill, J. Real-Time Monitoring using AJAX and WebSockets. In *Engineering of Computer Based Systems (ECBS), 2013 20th IEEE International Conference and Workshops on the*, s. 110–118, April 2013. doi: 10.1109/ECBS.2013.10.

Sareuon, e. a. What browsers support HTML5 WebSocket API? Dostupné z: <http://stackoverflow.com/questions/1253683/what-browsers-support-html5-websocket-api>.

Svensson, M. *WebSocket eller Ajax i Webbapplikationer : Är WebSockets prestanda tillräcklig för att ersätta Ajax?* 2012. Dostupné z: <http://www.diva-portal.org/smash/record.jsf?pid=diva2:538356>.

Příloha A - Obsah přiloženého CD

Data na přiloženém kompaktním disku obsahují kompletní zdrojové kódy frameworku Phem, včetně všech nových rozšíření. Dále jsou obsaženy zdrojové kódy demonstrační aplikace, ukázkové konfigurační soubory pro nastavení součástí doporučené běhové platformy, text této zprávy ve formátu PDF a zdrojový kód této zprávy pro LaTeX. Organizace dat na CD odpovídá adresářové struktuře znázorněné odrážkami.

- src

  Phem – kompletní zdrojové kódy frameworku, včetně nových rozšíření

  PhemDemo – zdrojové demonstrační aplikace

- doc

  zprava – vlastní text této diplomové práce (PDF a TEX)

  podklady – podkladový materiál k této práci, obrázky, diagramy a exporty

- conf

  nginx – ukázková konfigurace webového serveru

  haproxy – ukázková konfigurace proxy serveru

  php-fpm – ukázková konfigurace php FastCGI procesového manažeru

  hhvm – ukázková konfigurace platformy HHVM