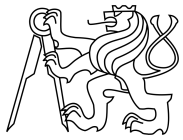
ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ



ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Název:	BigCloud - uživatelské rozhraní pro služby SaaS a PaaS
Student:	Jindřich Máca
Vedoucí:	Ing. Jiří Chludil
Studijní program:	Informatika
Studijní obor:	Web a multimédia
Katedra:	Katedra softwarového inženýrství
Platnost zadání:	Do konce letního semestru 2016/17

Pokyny pro vypracování

Na základě požadavků zadavatele (SIC, s.r.o.) navrhněte a implementujte minimálně 5 modulů k poskytování vybraných SaaS a PaaS služeb pro veřejný cloudový systém.

1. Analyzujte požadované SaaS a PaaS služby (OwnCloud, RemoteDesktop, Databázové úložiště, atd.) a jejich způsob zavedení do stávajícího systému.
2. Analyzujte stávající řešení (u IaaS služeb) komunikace s jádrem systému (správa virtuálních počítačů) a zvažte způsob jeho využití pro podporu SaaS a PaaS služeb.
3. Analyzujte stávající implementaci uživatelského rozhraní (MVP) pod prostředím NETTE a navrhněte jeho optimalizaci.
4. Na základě analýz navrhněte nové moduly (uživatelské rozhraní a jeho komunikaci s jádrem) k poskytování požadovaných SaaS a PaaS služeb.
5. Implementujte moduly (včetně navrhovaných optimalizací).
6. Podrobte implementované moduly vhodným testům (jednotkové, integrační a usability).

Seznam odborné literatury

Dodá vedoucí práce.

L.S.

Ing. Michal Valenta, Ph.D.
vedoucí katedry

prof. Ing. Pavel Tvrdík, CSc.
děkan

V Praze dne 24. prosince 2015

České vysoké učení technické v Praze
Fakulta informačních technologií
Katedra softwarového inženýrství

Bakalářská práce

BigCloud - uživatelské rozhraní pro služby SaaS a PaaS

Jindřich Máca

Vedoucí práce: Ing. Jiří Chludil

30. června 2016

Poděkování

Především bych chtěl poděkovat vedoucímu práce, Jiřímu Chludilovi, za jeho podporu a cenné rady při psaní této práce. Velké díky patří také panu Petru Marešovi za četné konzultace, vstřícnost a celkovou spolupráci. Také velmi děkuji Haně Kozákové za pomoc, rady a podporu. Poděkovat bych chtěl rovněž své rodině za podporu po celou dobu studia.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval(a) samostatně a že jsem uvedl(a) veškeré použité informační zdroje v souladu s Metodickým pokynem o etické přípravě vysokoškolských závěrečných prací.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů. Dále prohlašuji, že jsem s Českým vysokým učením technickým v Praze uzavřel dohodu, na základě níž se ČVUT vzdalo práva na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 autorského zákona. Tato skutečnost nemá vliv na ust. § 47b zákona č. 111/1998 Sb., o vysokých školách, ve znění pozdějších předpisů.

V Praze dne 30. června 2016 .....

Odkaz na tuto práci

Máca, Jindřich. *BigCloud - uživatelské rozhraní pro služby SaaS a PaaS*. Bakalářská práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2016.

Abstrakt

Cílem práce je analyzovat stávající řešení webového uživatelského rozhraní IaaS služeb projektu Big Cloud i nově požadované SaaS a PaaS služby. Následně na základě těchto analýz navrhnout, implementovat a otestovat moduly těchto nových služeb. Tato písemná část se zabývá především rozborem konkrétních použitých řešení a vylepšení pro jednotlivé služby, která staví na základech předešlé analýzy celého stávajícího projektu, jako taktéž prezentováním výsledků jejich realizace.

Klíčová slova Projekt Big Cloud, PaaS, SaaS, Webové uživatelské rozhraní, Nette, ownCloud, Vzdálená plocha Linux, Databázové úložiště, Redmine

Abstract

Goal of this thesis is to analyze current solution of web-based user interface for IaaS services of Big Cloud project and also newly required SaaS and PaaS services. Then based on these analysis to design, implement and test modules of these new services. This written part is mainly engaged in analysing concrete used solutions and improvements for individual services, that are based on foundations of previous analysis of whole current project, as also presenting the results of their realization.

Keywords Project Big Cloud, PaaS, SaaS, Web-based user interface, Nette, ownCloud, Remote desktop Linux, Database storage, Redmine

Obsah

Seznam obrázků

Seznam tabulek

Úvod

V dnešní době, kdy se většina služeb přesouvá nebo orientuje na internet, je důležitou otázkou infrastruktura, na které budou tyto služby postaveny. Projekt Big Cloud[5] cílí právě na tuto problematiku a snaží se ji obsáhnout ve všech jejích kategoriích jako jsou IaaS, SaaS a PaaS. Proto firma S.I.C. spol. s.r.o.[6], která tento cloudový systém vyvíjí, chce svým zákazníkům přinést jednotné uživatelské rozhraní pro všechny tyto kategorie.

Ve své práci se zabývám pouze částí z rozsáhlého zaměření tohoto projektu a to webovým uživatelským rozhraním pro jednotlivé služby z kategorií SaaS a PaaS. Konkrétně se pak jedná o analýzu, návrh, implementaci a otestování řešení pro vytváření úložišť ownCloud[7], zakládání uživatelů pro správu vzdálené Linuxové plochy, tvorby databázových úložišť a zakládání systémů Redmine[8]. Pro tvorbu uživatelského rozhraní všech těchto služeb je použit webový PHP[9] framework Nette[10] a jako databáze je zde použita PostgreSQL[11]. Dále se v práci věnuji analýze a řešení problémů souvisejících přímo s vývojem, jako jsou např. nastavení serveru pro průběžnou integraci, převod podpůrných procesů na systémové služby, vytváření instalačních balíčků nebo řešení modelu pro verzování projektu pomocí nástroje Git[12].

Struktura práce pak kopíruje posloupnost řešení jednotlivých částí a to podle standardních postupů softwarového inženýrství. Nejdříve se tedy věnuji analýze nejen daných služeb a jejich jednotlivých požadavků, ale i předchozího stavu projektu Big Cloud. Následuje návrh řešení opět pro jednotlivé služby tak, aby splnilo ony požadavky a zároveň bylo v souladu se stávajícím fungováním systému. Posléze toto řešení implementuji a v návaznosti na to ho testuji, zda funguje podle očekávání i zda odpovídá plánovanému návrhu. Tato práce vzdáleně navazuje na diplomovou práci Big Cloud - backend pro veřejný cloudový systém[13] od Ing. Petra Gregora, která se také zabývá projektem Big Cloud, ovšem řeší kategorii IaaS a to z hlediska backendu.

Rozbor zadání

Ve své práci se budu zabývat analýzou stávajícího řešení projektu Big Cloud, ve které se zaměřím na zhodnocení jeho stávajícího řešení a nalezení případných nedostatků.

Dále se také budu zabývat analýzou i návrhem SaaS a PaaS služeb, kde se budu věnovat jejich požadavkům, atributům, způsobu zavedení do stávajícího systému a uživatelskému rozhraní.

Výstupem práce pak bude, kromě analýz a návrhů, také implementace nejen těchto služeb, ale i jejich testů, stejně tak jako instalační, programátorská a uživatelská dokumentace. V následujících bodech bych pak rád popsal a přesněji specifikoval jednotlivé body zadání této práce.

Analyzujte požadované SaaS a PaaS služby (OwnCloud, RemoteDesktop, Databázové úložiště, atd.) a jejich způsob zavedení do stávajícího systému

Analyzuji funkční požadavky a případné atributy u všech požadovaných SaaS a PaaS služeb.

Analyzujte stávající řešení (u IaaS služeb) komunikace s jádrem systému (správa virtuálních počítačů) a zvažte způsob jeho využití pro podporu SaaS a PaaS služeb

Analyzuji stávající řešení projektu Big Cloud z hlediska použitých technologií, vývojářských nástrojů a především technické struktury.

Analyzujte stávající implementaci uživatelského rozhraní (MVP) pod prostředím NETTE a navrhněte jeho optimalizaci

V tomto bodě zhodnotím stávající programový kód uživatelského rozhraní z pohledu Nette vývojáře a vyhodnotím jeho celkovou dokumentaci. Také zde navrhnu implementační vylepšení pro nalezené nedostatky.

Na základě analýz navrhněte nové moduly (uživatelské rozhraní a jeho komunikaci s jádrem) k poskytování požadovaných SaaS a PaaS služeb

Navrhnu nové moduly a způsob jejich nasazení do stávajícího systému a zkontroluji také, že splňují zanalyzované požadavky. Dále také navrhnu jejich uživatelské rozhraní.

Implementujte moduly (včetně navrhovaných optimalizací)

Implementuji moduly SaaS a PaaS služeb v rámci stávajícího řešení projektu Big Cloud. Také vytvořím jejich instalační, programátorskou i uživatelskou dokumentaci.

Podrobte implementované moduly vhodným testům (jednotkové, integrační a usability)

Vytvořím jednotkové, integrační a akceptační testy na implementované SaaS a PaaS služby. Dále také provedu testování použitelnosti jejich uživatelského rozhraní.

Kapitola 1

Analýza

Analýza současného stavu projektu Big Cloud i jednotlivých požadovaných SaaS a PaaS služeb. Její část o stavu projektu jsem rozdělil na analýzy použitých technologií, vývojářský nástrojů, technické struktury projektu a analýzu kódu. U analýzy služeb se pak zabývám především požadavky zadavatele.

1.1 Projekt Big Cloud

Projekt Big Cloud je nový český výkonný hybridní cloudový systém. Zadavatel práce, firma S.I.C. spol. s.r.o., začala jeho vývoj v druhé polovině roku 2014, přičemž první část projektu se zaměřovala hlavně na IaaS služby. Backend této části vyvinul i v současnosti stále spravuje tehdejší student FIT ČVUT a nyní již Ing. Petr Gregor. Více o této části se můžete dočíst v jeho diplomové práci[13]. Nad tímto backendem, obsahujícím příslušné API, poté vývojový tým, podrobněji popsaný v následující sekci 1.1.1, vytvořil webové uživatelské rozhraní.

1.1.1 Vývojový tým

V této části bych ještě rád představil vývojový tým z hlediska rolí jednotlivých jeho členů, se kterými jsem na projektu v rámci zadavatelské firmy spolupracoval.

- Majitel firmy S.I.C. spol. s.r.o. a zadavatel projektu. Také osoba, která se stará o finanční stránku celého projektu.
- Technický ředitel firmy S.I.C. spol. s.r.o. a CEO projektu Big Cloud. Specifikuje požadavky a zodpovídá za jeho vývoj.
- Autor backendu IaaS a správce serverů firmy S.I.C. spol. s.r.o. využívaných pro projekt Big Cloud i další služby.

- Vedoucí vývojového týmu. Rozděluje práci v týmu, dohlíží na vývoj projektu a plnění požadavků. Autor webového frontendu pro IaaS i podpůrné služby pro napojení na IaaS API.

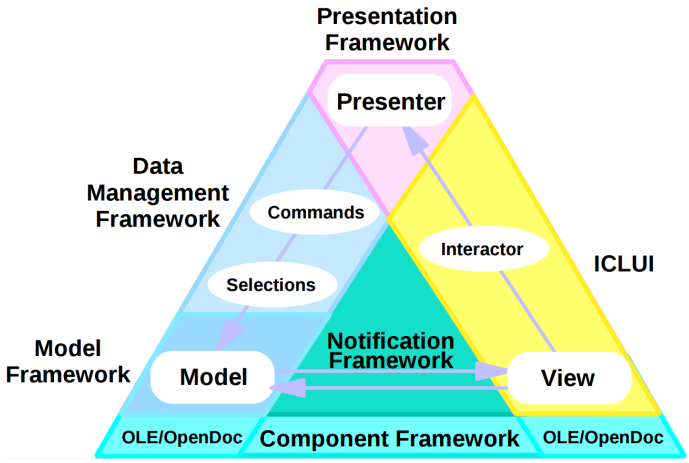
- Autor backendu webového uživatelského rozhraní. Je zodpovědný za celý model webové části projektu, většinu podpůrných služeb a jejich vzájemného napojení. Také to je osoba, se kterou jsem nejvíce spolupracoval při vývoji SaaS i PaaS služeb právě z hlediska jejich backendového zázemí.

1.2 Použité technologie

Mým prvotním úkolem bylo, v rámci práce ve výše uvedené firmě, seznámení se současným stavem projektu. Začal jsem tedy analýzou použitých technologií.

1.2.1 Webový framework

Implementace webového uživatelského rozhraní byla realizována, v úvodu již zmiňovaném, Nette frameworku pro jazyk PHP. Tento framework je specifický tím, že je postaven na třívrstvé MVP architektuře[1] viz. diagram 1.1 a skládá se z řady i samostatně použitelných knihoven.

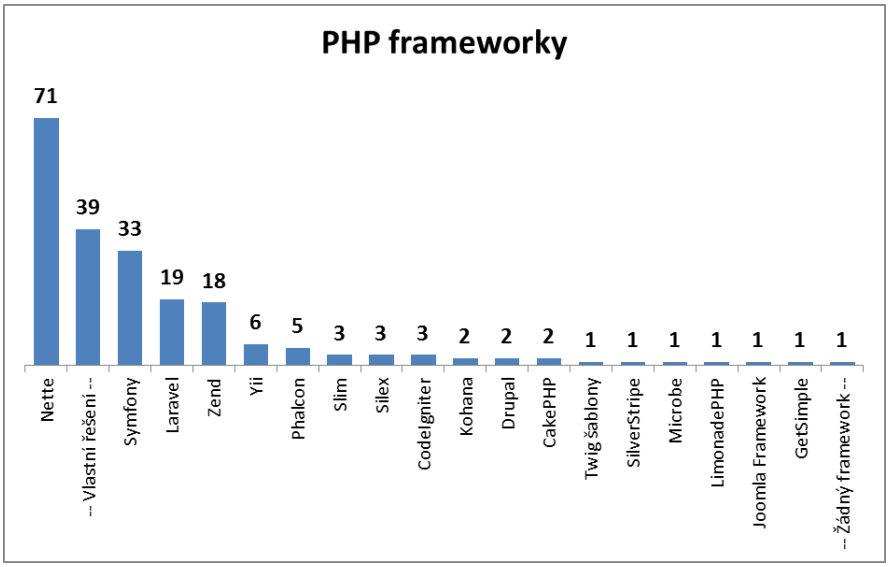


Obrázek 1.1: Diagram MVP architektury[1]

Změna vybraného programovacího jazyka v této fázi již nepřicházela v úvahu, protože by se musela zahodit v podstatě celá dosavadní práce týkající se webového rozhraní. Na druhou stranu při výběru frameworku pro PHP se nabízí hned několik různých alternativ, přičemž tento přechod by nemusel být tak náročný. Vyplatí se jimi tedy zaobírat.

1.2.1.1 Alternativní frameworky

Nejdříve bylo potřeba vybrat perspektivní alternativy. To jsem učinil na základě průzkumu používaných PHP frameworků na českém trhu ze začátku roku 2016[2]. Tyto výsledky můžete vidět na grafu 1.2.



Obrázek 1.2: Nejpoužívanější frameworky v ČR (2016)[2]

Pominu-li možnost vlastního řešení, jelikož takových může být velká spousta a nedají se dost dobře dohledat a zhodnotit, jsou vhodnými kandidáty pro porovnání s vedoucím **Nette**[10], hlavně frameworky **Symfony**[14], **Laravel**[15] a **Zend Framework 2**[16].

1.2.1.2 Výběr hodnotících kritérií

Než jsem se pustil do samotného srovnání, stanovil jsem si důležitá hodnotící kritéria, v jejichž kontextu jsem frameworky porovnával. Hodnocení každého z následujících bodů byla poté přiřazena i číselná hodnota na škále od 1 do 5, kde nejlepší možné hodnocení bylo 5. To mi i v závěru pomohlo srovnat výsledky do přehledné tabulky 1.1.

Jednoduchost použití

Framework používám především proto, aby mi usnadnil práci. Je tedy velice důležité, aby se dal používat snadno a pokud možno co nejvíce intuitivně. Pokud narazím na problém a musím velice dlouho hledat řešení ve svém zvoleném frameworku, bylo by možná lepší vyřešit problém po svém. I proto je tohle pro spoustu lidí jeden z nejdůležitějších faktorů při výběru nejen frameworku.

Kvalita dokumentace

To, že by měl mít framework k dispozici poctivě zdokumentované API, asi nemusím zdůrazňovat. Pokud už si nějaký framework vyberu a chci, aby se mi snadno používal, musí obvykle mít rozsáhlou, ale přesto přehlednou dokumentaci, ve které se dá relativně snadno vyhledávat. Pokud pak např. narazím na nějaký problém, měl bych po chvilce hledání být schopen najít řešení, ať už v dokumentaci nebo někde na fóru. To ovšem také vyžaduje početnější a hlavně ochotnou komunitu okolo.

Flexibilita

Každý projekt je jiný. I když mnoho prvků, jako např. přihlašování, práce s databází apod. je společných, klíčová funkcionalita se projekt od projektu většinou zásadně liší. Od frameworku je tak vyžadováno, aby se byl schopen všem těmto nárokům co nejlépe přizpůsobit, a poskytoval tak prostor pro jejich naplnění.

Rozšiřitelnost

Jak už bylo popsáno v předchozích požadavcích, projekty mají obvykle mnoho prvků společných i rozdílných. Ovšem i ty více rozdílné mohou být společné pro menší skupinu projektů. V tu chvíli do hry vstupuje možnost rozšíření daných frameworků. Jde především o to, aby bylo možné základní implementaci frameworku snadno doplnit o nějakou další rozšířenou funkcionalitu. Také je velice důležité, aby tuto rozšířenou funkcionalitu bylo možné snadno získat a nainstalovat třeba i do již rozpracované aplikace.

Výkon

Framework svojí robustností a množstvím připravené funkcionality nesmí zpomalovat chod aplikace jako takové. Ba naopak, měl by se snažit optimalizovat její výkon, např. pomocí cache tak, aby odpovídala na požadavky co nejrychleji. To opět není jednoduchý úkol, ale tento požadavek může hrát svojí roli při výběru.

1.2.1.3 Hodnocení Nette

Jednoduchost použití

Nette je na použití velice jednoduchý framework a je to jedna z jeho hlavních předností i důvod, proč je u nás tak populární. Ne vše zde sice funguje zcela intuitivně, ale jakmile si člověk např. projde základní tutoriál, je schopen snadno poskládat jednoduchou webovou aplikaci. Ovšem tato jednoduchost a přímočarost může být překážkou, pokud potřebuje použít framework trochu

jinak, než jak bylo zamýšleno. Každopádně v této oblasti hodnotím Nette velice kladně a dávám mu **4**.

Kvalita dokumentace

Zde leží veliký kámen úrazu celého frameworku. I když obsahuje API i základní tutoriál a dokumentace se nedá označit za vyloženě strohou, popisuje pouze základní užití jednotlivých částí. Pokud se například vyžaduje nějaké složitější řešení, musí se informace pracně dohledávat všude možně a občas něco i trochu domýšlet. Větší problém pak budou mít uživatelé z jinak než česky hovořících zemí, jelikož i přesto, že existuje anglická verze dokumentace, je na tom možná ještě hůře, než ta česká a hlavně zde chybí podpora zahraniční komunity. To je jeden z hlavních důvodů, proč si framework drží velkou míru popularity pouze u nás. Moje hodnocení je zde slabá **2**.

Flexibilita

Jak už bylo zmíněno u hodnocení jednoduchosti použití, Nette není ve všem úplně flexibilní. Na druhou stranu komplikace nastávají většinou až při řešení nestandardních situací a i zde se dá z pravidla najít nějaké východisko. Hodnotil bych tedy spíše kladně za **4**.

Rozšiřitelnost

Nette poměrně kvalitně podporuje rozšíření pomocí tzv. add-ons, jejichž přehled je k vidění na `https://componette.com/`, kde se dá najít množství různých rozšíření. Ty pak při použití často vyvažují nízkou flexibilitu frameworku. Jelikož je framework postaven na Composeru[17], tak i převážná část rozšíření se získává a instaluje tímto způsobem. Abych jen nechválil, je potřeba srovnat stav i s konkurencí, kde je rozšíření daleko více. Sečteno podtrženo, dávám v této kategorii frameworku **4**.

Výkon

Framework se dá rozhodně označit za velice výkonný. Díky jeho jednoduchosti podporuje cachování na úrovni zdrojů, vygenerovaných HTML stránek i databázových dotazů. Navíc díky jeho pokročilé implementaci DI šetří čas i při inicializaci aplikace u každého HTTP dotazu. Zde si dovolím udělit **5**, jelikož z hlediska optimalizace není Nette v podstatě co vytknout. Podkladem pro mé tvrzení také tvoří mimo jiné i nezávislé výkonnostní testy[18], ve kterých se řadí mezi jedny z nejrychlejších.

1.2.1.4 Hodnocení Symfony

Jednoduchost použití

Symfony je poměrně robustní framework a je potřeba se v něm nejdříve trochu zorientovat, než se dá začít používat naplno. Mohou zaskočit např. i poměrně základní věci jako DI, autorizace i autentizace nebo webové formuláře. Když si však člověk dá tu práci a do Symfony pronikne, dostane na oplátku velice komplexní nástroj. Celkově ale budu hodnotit **2**, jelikož v tomto kritériu jde hlavně o jednoduchost.

Kvalita dokumentace
Symfony má v anglickém jazyce velice kvalitní a rozsáhlou dokumentaci s velkým množstvím praktických příkladů. Řekl bych, že se tím snaží kompenzovat právě svoji komplexnost. Každopádně vzhledem k rozsáhlosti občas přeci jenom chvíli trvá něco konkrétního v dokumentaci najít. Z toho důvodu hodnotím za **4**. Samozřejmě v souvislosti s tímto hodnocením by bylo dobré ještě dodat, že API je plně k dispozici a zahraniční komunita uživatelů je velice vstřícná i aktivní, přičemž ta česká se hlavně v posledních letech také poměrně rozrůstá.

Flexibilita
Vzhledem k jeho komplexnosti je Symfony i velice flexibilní, ovšem občas s pocitem mírné těžkopádnosti, jelikož robustnost řešení poté často vyžaduje i náročnější konfiguraci apod. Celkově ale hodnotím Symfony v rámci kritéria kladně a to za **4**.

Rozšiřitelnost
Framework je velice dobře rozšiřitelný a to pomocí systému tzv. bundlů, jejichž seznam je k nalezení např. na stránce `http://knpbundles.com/`. Ty se dají dokonce v některých případech použít i samostatně bez Symfony. Jejich získávání a instalace probíhá většinou pomocí Composeru, jak tomu bylo např. i u Nette. Dobré je také zmínit, že takových bundlů existuje již opravdu velké množství. Celkově je rozšiřitelnost silnou stránkou Symfony a proto hodnotím **5**.

Výkon
Bohužel již několikrát zmiňovaná robustnost se často projevuje nejen na velikosti instalace Symfony, ale i na jeho rychlosti zpracování HTTP požadavku. Dají se sice provádět optimalizace zdrojů a pokud je dobře nastavené cachování, vše funguje relativně dobře, ale při běžném nastavení je stejně pomalejší odezva u hodně vytěžovaných aplikací patrná. Celkově hodnotím tento aspekt za **3**.

1.2.1.5 Hodnocení Laravel

Jednoduchost použití

Laravel je pro používání poměrně jednoduchý a pro projití základní dokumentace v něm jde relativně snadno vytvořit jednodušší webovou aplikaci. Také nabízí mnoho již vestavěných zajímavých funkcionalit jako např. účetnictví, fronty nebo SSH úkoly. Tady právě ale trochu tkví kámen úrazu, jelikož pro možnost správného používání jeho plného potenciálu, je potřeba zkrátka proniknout hlouběji. Každopádně za celkové hodnocení kritéria jednoduchosti použití si odnáší **4**.

Kvalita dokumentace

Dokumentace je veskrze standardní a přehledná, ovšem kvalit např. dokumentace Symfony ani zdaleka nedosahuje. API je samozřejmě plně k dispozici a Laravel se může pyšnit poměrně velkou a hlavně rychle se rozrůstající komunitou, především tedy v zahraničí. Celkově si ale odnáší průměrné hodnocení za **3**.

Flexibilita

Framework, jak už bylo řečeno, v sobě obsahuje již mnoho předpřipravené funkcionality a díky jeho vyvážení jednoduchosti s robustností je i poměrně flexibilní. Ovšem najdou se zde opět nestandardní situace, kdy vyřešit některé problémy je poměrně náročné. Ovšem celkově hodnotím za **4**.

Rozšiřitelnost

Laravel je opět postaven na Composeru a aktuálně disponuje bezmála 10 000 tzv. balíčků (package), jejichž přehled je k nalezení na stránce `http://packalyst.com/`. Avšak ne všechny balíčky se dají instalovat pomocí Composeru a hodně z nich je "šito horkou jehlou", proto si člověk musí dát chvíli práci, než najde ten vyhovující a technicky dostačující. I proto získává **4**.

Výkon

Z výkonnostního hlediska na tom framework není úplně dobře. I když se navenek tváří relativně jednoduše, už jsem naznačoval u hodnocení jeho flexibility, že uvnitř je poměrně robustní a také obsahuje v základu mnoho specializovaných služeb, které často nejsou ani využity. K tomu se nabalí ještě nějaké to ne úplně kvalitní rozšíření a výkonnostní problém je na světě. Samozřejmě to zase není tak zlé a při použití správných nastavení, případně balíčků pro cachování, se rychlost zpracování HTTP požadavků dostává přibližně na úroveň Symfony.

Nepomáhá zde ale fakt, že autoři vydali ještě ořezanou minimalizovanou verzi frameworku s názvem Lumen[19] zaměřenou především na výkon. Ta je tady určena hlavně pro malé služby a rychlé API, ale vyvstává logická otázka, proč v těchto případech nelze použít originální Laravel? A odpověď je asi jasná... Ve výsledku tedy z tohoto kritéria ale dostává průměrné hodnocení a to **3**.

1.2.1.6 Hodnocení Zend Framework 2

Jednoduchost použití

Druhá verze Zend Frameworku je opět velice robustní i komplexní záležitost, bohužel v tomto případě se to nesnaží ani nijak schovat. Z toho plyne, že jako jednoduchý bych ho rozhodně neoznačil, právě naopak bych řekl, že se do něho proniká nejhůře ze všech uvedených frameworků. Důkazem toho může být již úvodní tutoriál[20], jehož uvedené zdrojové kódy se jeví již na první pohled jako komplikované, i když jsou ve svém smyslu v podstatě jednoduché. Tímto si ode mě vysloužil pouze **2**.

Kvalita dokumentace

Dokumentace frameworku je rozhodně velice obsáhlá (cca. 1 000 stran) a plná příkladů (cca. 500 ukázek kódu), ale ve výsledku je velice nepřehledná a špatně se v ní vyhledává. Takže shrnuto podtrženo obsahuje nejspíše vše potřebné, ale je náročně to zde najít a tím je dané i hodnocení tohoto kritéria za **3**. Samozřejmě API je k dispozici a komunita se určitě nedá označit za malou.

Flexibilita

Framework je určitě velice flexibilní, což je ale vykoupeno jeho velikou robustností a "ukecaností". Za kritérium flexibility si ale zaslouží **4**.

Rozšiřitelnost

Zend Framework opět využívá Composer a má systém tzv. modulů, jejichž relativně velké množství je k vidění na stránce `https://zfmodules.com/`. Většinu modulů pak lze instalovat také pomocí Composeru. Obecně zde moc problém s rozšiřitelností není, ale v porovnání např. se systémem Symfony zde přeci jen něco chybí, takže výsledné hodnocení bude nakonec **4**.

Výkon

Z hlediska výkonu znovu musím podtrhnout již několikrát zmiňovanou robustnost. Framework je pak opět na podobné výkonnostní úrovni jako třeba Symfony a samozřejmě se dá pomocí dalších modulů a nastavení jeho výkon optimalizovat. Proto jako hodnocení obdrží neutrální **3**.

1.2.1.7 Shrnutí hodnocení

Celkové shrnutí hodnocení je přehledně vidět v následující tabulce 1.1.

Kritérium	Nette	Symfony	Laravel	Zend Framework 2
Jednoduchost použití	**4**	2	**4**	2
Kvalita dokumentace	2	**4**	3	3
Flexibilita	**4**	**4**	**4**	**4**
Rozšiřitelnost	4	**5**	4	4
Výkon	**5**	3	3	3
Celkové hodnocení	**19**	18	18	16

Tabulka 1.1: Shrnutí hodnocení frameworků

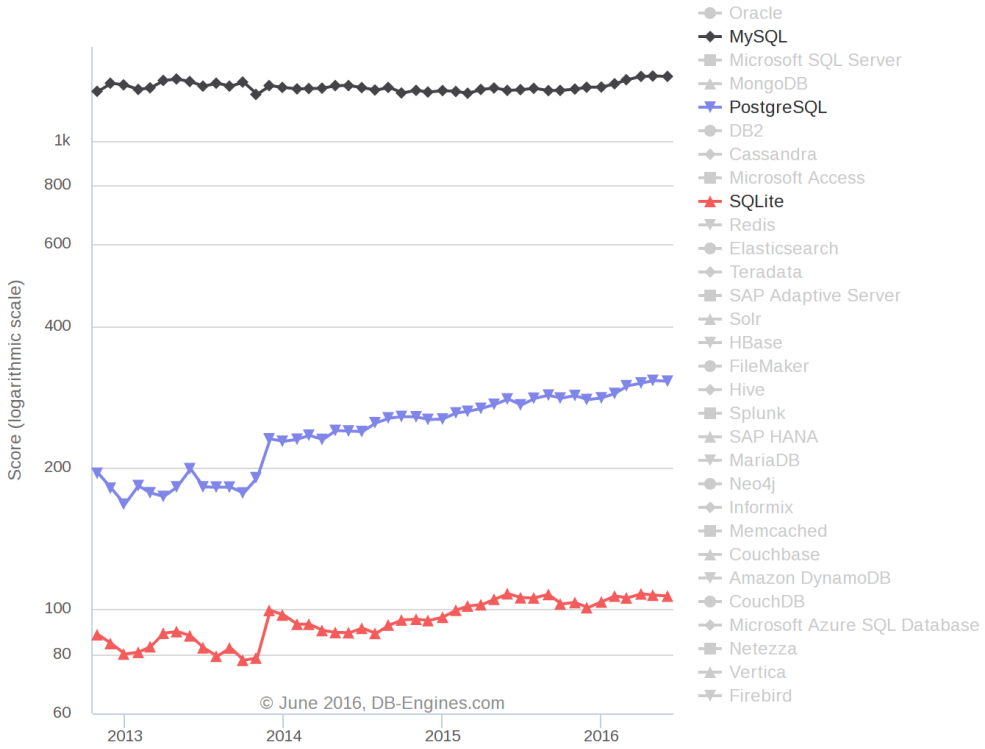
Z výsledků hodnocení je poměrně jasně vidět, proč je Nette u nás tak populární. Vděčí za to hlavně své jednoduchosti použití, relativně vysoké flexibilitě i výkonu. Z tohoto důvodu se také nevyplatí přecházet v rámci projektu Big Cloud na nějaký jiný PHP framework a dalo by se tak říct, že Nette byla správná volba.

1.2.2 Databáze

Jako databáze byla na projektu Big Cloud použita PostgreSQL. Jelikož i zde by nebyl případný přechod mezi řešeními tak náročný, opět se vyplatí zabývat jejími alternativami.

1.2.2.1 Alternativní databáze

Nejdříve bylo znovu potřeba vybrat perspektivní alternativy. Vycházel jsem z předpokladu, že databáze má být relační, protože např. na čistě objektovou by se už nedalo přejít tak snadno, a požadavku firmy, že použitá databáze má být nekomerční, jelikož firma nebude platit její licenci. Za těchto podmínek jsem vybral ze žebříčku nepopulárnějších databází, který můžete vidět na grafu 1.3, tři vhodně kandidáty a to kromě **PostgreSQL**[11] i **SQLite**[21] a samozřejmě velice populární **MySQL**[22].



Obrázek 1.3: Graf popularity databází (červen 2016)[3]

Podrobné informace o tom, jaké byly použity metody pro výpočet popularity databáze v grafu 1.3, naleznete na `http://db-engines.com/en/ranking_definition`.

1.2.2.2 Výběr hodnotících kritérií

Opět jsem si stanovil důležitá hodnotící kritéria, v jejichž kontextu jsem jednotlivé databáze srovnával. Zase zde platilo hodnocení od 1 do 5, kde nejlepší možné bylo 5 a výsledkem byla přehledná srovnávací tabulka 1.2.

Rychlost

Asi nejdůležitější rozhodovací faktor pro výběr databáze a na druhou stranu kritérium, které se objektivně poměrně špatně hodnotí. O většině relačních databází se dá prohlásit, že jsou relativně rychlé, protože to je jeden z jejich hlavních cílů, ale důležité je říci, že to platí při jejich správném použití a např. porovnávání výkonnostních testů bývá často zkreslené právě díky různému způsobu použití a dále např. i kvůli běhu na rozdílném hardwaru. Proto se budu v tomto kritériu zabývat spíše tím, při jakých podmínkách databáze běží nejrychleji a následně hodnotit, jestli je to vhodné pro projekt Big Cloud. To znamená, že pokud databáze dostane nižší hodnocení, nemusí být nutně

pomalá, ale při použití na tomto konkrétním projektu by pravděpodobně nedosahovala svého plného potenciálu.

Robustnost

Tento pojem v daném kontextu znamená, že databáze zachová a ochrání data i při nečekaných událostech, jako je např. výpadek proudu uprostřed nějaké její transakce. Tento faktor je opět velice důležitý, protože u projektu probíhají platební operace, asynchronní synchronizace mezi IaaS API a webovým rozhraním apod. Zde všude je potřeba, aby se databáze postarala o správné a úplné uložení dat.

Funkcionalita

Toto kritérium se zabývá množstvím poskytovaných funkcí, které konkrétní databáze nabízí. Ne všechny totiž umožňují např. pokročilé způsoby skládání dotazů nebo další specifické funkce. Také je zde potřeba nezapomenout zohlednit míru dodržení obecného standardu SQL. Celkově se v hodnocení chci opět zaměřit na to, aby měla databáze co nejvíce dostupné funkcionality, vzhledem ke složitosti projektu Big Cloud a jeho aktuálně používaným PostgreSQL.

Datové typy

Dalším velice důležitým faktorem je množství různých poskytovaných databázových datových typů. V rámci projektu se jich už teď používá velké i poměrně různorodé množství a tyto možnosti by bylo dobré co nejvíce zachovat.

Při následujícím hodnocení těchto kritérií byl, kromě samotné dokumentace, velice dobrým zdrojem informací článek „SQLite vs MySQL vs PostgreSQL: A Comparison Of Relational Database Management Systems“[23], který dle mého názoru velice objektivně srovnává právě tři vybrané databáze.

1.2.2.3 Hodnocení PostgreSQL

Rychlost

Tato kategorie není zrovna nejsilnější stránkou PostgreSQL. To má ale jeden pádný důvod a tím je velká robustnost i poskytovaná funkcionalita tohoto databázového systému. Důsledek toho je, že např. není úplně vhodný na vysoký počet jednoduchých čtecích operací, ale na druhou stranu vyniká ve vyváženém počtu složitých čtecích i zapisovacích operacích. Tím pádem dostává hodnocení z hlediska použití v projektu a to **4**, protože projekt spíše používá méně složitějších operací.

Robustnost

Toto je naopak poměrně silná stránka tohoto databázového systému. Díky jeho transakčnímu navržení je velice spolehlivý a odolný při práci s daty. Tato režie navíc se ale projevuje, jak už bylo řečeno, v jeho rychlosti. Ovšem z této kategorie si odnáší **5**.

Funkcionalita

PostgreSQL poskytuje velice širokou a rozvinutou škálu funkcionalit jako např. vlastní procedury nebo podporu objektových datových typů. Celkově tak umožňuje vytvářet velice složité databázové systémy. Rozhodně si v této kategorii zaslouží také **5**.

Datové typy

S velkou škálou funkcionality se pojí i velká paleta dostupných datových typů a samozřejmě možnost definovat svoje vlastní. Z těch nejdůležitějších bych zmínil například *numeric* a *text*, které se ve velké míře používají pro ukládání čísel a textu. Je zde rozdíl např. od MySQL, která preferuje pro tyto účely spíše typy *varchar* a *integer*, kvůli indexaci a tím rychlejšímu přístupu. PostgreSQL samozřejmě tyto typy také obsahuje, ale jelikož podporuje dobrou indexaci i pro výše zmíněné typy a ty jsou zároveň více dynamické, preferuje se jejich použití. Ze zajímavých typů bych ještě uvedl např. *money* pro ukládání peněžních hodnot nebo *xml*, který umožňuje ukládat XML dokumenty. Celkově si opět odnáší **5**.

1.2.2.4 Hodnocení SQLite

Rychlost

Asi nejsilnější stránka této databáze, pro kterou byla i navržena. Dala by se jedním slovem popsat jako "odlehčená", což je důvodem její velké rychlosti. Tím pádem je velice vhodná například pro velké množství jednoduchých čtecích operací. Také se skládá z jediného souboru na disku, což ji dělá i lehce přenosnou. Každopádně jako hodnocení v této kategorii si zkrátka odnáší **5**.

Robustnost

Co se týče robustnosti, tak i když samozřejmě obsahuje transakční zpracování, nedá se úplně srovnávat např. s PostgreSQL. Důvodem můžete být např. již zmiňovaná vlastnost, že celá databáze se ve své podstatě nachází v jednom souboru. Proto i s přihlédnutím na potřebu této vlastnosti pro projekt Big Cloud hodnotím pouze za **3**.

Funkcionalita

Díky její jednoduchosti neobsahuje SQLite žádné pokročilé funkcionality a např. zde chybí i jakákoliv správa uživatelů. Z tohoto důvodu rozhodně není vhodným kandidátem pro projekt Big Cloud a odnáší si hodnocení **2**.

Datové typy

SQLite obsahuje pouze 4 základní datové typy *INTEGER*, *REAL*, *TEXT*, *BLOB* a k tomu hodnotu *NULL*. I když by si projekt nejspíše s těmito typy vystačil, je to příliš svazující, hlavně opět z hlediska přechodu z aktuální PostgreSQL, takže toto kritérium hodnotím za **3**.

1.2.2.5 Hodnocení MySQL

MySQL by se dala v mnoha ohledech označit za zlatou střední cestu a vhodného kandidáta na přechod z PostgreSQL. Z toho se bude odvíjet její hodnocení. Jedná se také o velice populární a tím pádem i podporovanou databázi.

Rychlost

I když databáze asi nedosahuje pro čtecí operace takové rychlosti, jako třeba SQLite, ve vyváženém počtu složitějších čtecích i zapisovacích operací si povede minimálně stejně jako PostgreSQL, možná i lépe. To je dané částečně i tím, že v některých oblastech databáze mírně odbíhá od SQL standardu, což je tolerováno díky její popularitě a o něco vyšší rychlosti, kterou tím získá. Ovšem z hlediska hodnocení to nečiní žádný velký rozdíl a proto si odnáší **5**, o stupínek lepší než PostgreSQL.

Robustnost

Obecně je databáze v porovnání s ostatními méně spolehlivá i co se týče transakcí. Je to převážně kvůli zmiňovanému obcházení některých standardů. Celkově to není nikterak závažné, ovšem opět v porovnání s PostgreSQL, uděluji hodnocení kritéria za **4**.

Funkcionalita

Funkcionalitu nabízí MySQL poměrně širokou, ale jsou známy některé její limitace, opět způsobeny hlavně nedodržením standardu. Z toho důvodu opět trošičku ztrácí na PostgreSQL a dostává hodnocení **4**.

Datové typy

Datových typů zde existuje velké množství, stejně tak jako možnost vytvářet vlastní, ale vzhledem k již uvedenému porovnání stejného kritéria u PostgreSQL, nejsou tak pružné. Takže hodnocení je opět pouze **4**.

1.2.2.6 Shrnutí hodnocení

Celkové shrnutí hodnocení je přehledně vidět v následující tabulce 1.2.

Kritérium	PostgreSQL	SQLite	MySQL
Rychlost	4	**5**	**5**
Robustnost	**5**	3	4
Funkcionalita	**5**	2	4
Rychlost	**5**	3	4
Celkové hodnocení	**19**	13	17

Tabulka 1.2: Shrnutí hodnocení databází

Z výsledků hodnocení je poměrně dobře vidět, že tým předpokládal velkou komplexnost projektu a z toho důvodu raději volil robustnější řešení. Jelikož ostatní aspekty hodnocení jsou velice podobné, nevyplatí se přecházet na jinou databázi.

1.3 Vývojářské nástroje

V další části mé analýzy jsem se zaměřil na vývojářské nástroje použité na projektu.

1.3.1 Verzování

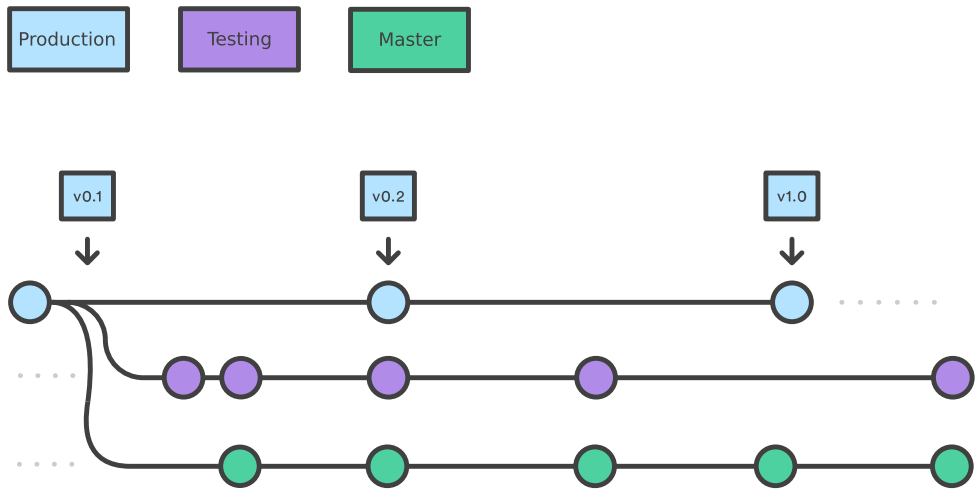
Na projektu je použit verzovací nástroj Git[12]. Nechci se zde zabývat jeho alternativami, jelikož změna verzování uprostřed vývoje projektu není přípustná a navíc Git nabízí vše, co je pro verzování projektu potřeba. Ovšem rád bych zde rozebral způsob jeho použití tzv. workflow, které je volitelné a tím pádem bychom ho eventuálně mohli přizpůsobit.

1.3.1.1 Současné použití

Jak je vidět na obrázku 1.4, využívá se tří hlavních větví a to *master*, *testing* a *production*. Pro další větve, ať už lokální nebo vzdálené, nebyla stanovena žádná další pravidla. Existuje jeden server s repozitářem jako centrální bod, přes který se synchronizují všechny změny. Na tomto serveru navíc běží GitLab[24],

který poskytuje k repozitáři i webové rozhraní. Každá z hlavních větví je pak navázána na vlastní aplikační server, přičemž její aktuální stav je na něj, buď automaticky nebo manuálně, průběžně nasazován z centrálního repozitáře pomocí CD nástroje Strider[25], který je blíže popsán v sekci 1.3.3. O skladbě a účelu serverů se pak blíže zmiňuji v kapitole 1.4.

Dále se pro zprávy commitů používají čísla úkolů ze systému pro správu projektu, popsaném v 1.3.2. Ovšem samotné commity byly vedeny poměrně špatným způsobem, tzn. velké soubory změn naráz se strohými komentáři nevypovídajícími o podstatě změny v commitu.



Obrázek 1.4: Hlavní větve projektu Big Cloud[4]

Celkově použití těchto pravidel není špatně, ba naopak, je to poměrně standardní postup. Ovšem má jednu zvláštnost a to, že většinou se pro produkční větev používá název *master* a pro vývojovou *delevop*. Celkově ale jde jen o pojmenování, které na funkčnost nemá žádný vliv. Dále by se rozhodně dal postup vylepšit při použití jednoho ze standardních workflow, popsaných v následující sekci 1.3.1.2.

1.3.1.2 Git Worflows

Přehled nejpoužívanějších workflow krásně porovnává manuál od firmy Atlassian[4]. Já bych ke každému jen krátce doplnil popis a zhodnocení z hlediska použití v projektu Big Cloud.

Centralizované Workflow

Jelikož je Git distribuovaný systém, to znamená, že každý má svojí vlastní kopii repozitáře, není vlastně potřeba zavádět žádný centrální bod, ale např. sdílet kód mezi jednotlivými kopiemi repozitářů podle potřeby. Ovšem pro

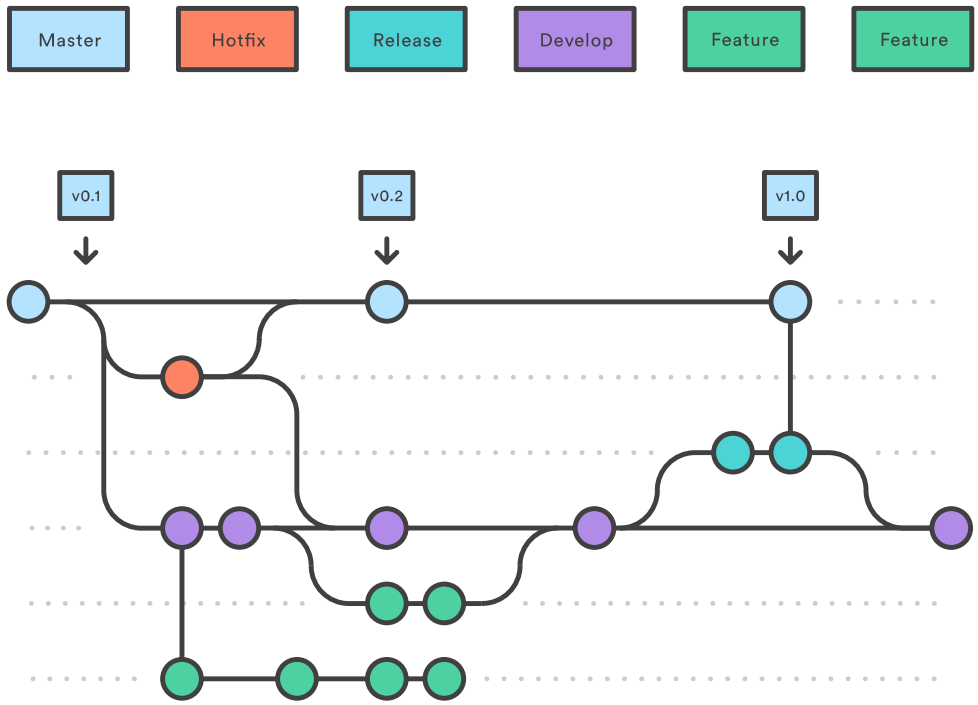
pravidelné synchronizace změn i sdílení kódu je dobré takový bod přesto zavést a právě to je hlavní účel tohoto workflow. Avšak případné konflikty změn se stále řeší na úrovni lokální kopie repozitáře a do centrálního bodu se posílají až vyřešené. V projektu Big Cloud takovýto bod samozřejmě existuje, takže toto workflow je zde zahrnuto.

Workflow větví s funkcionalitami

Toto workflow stále používá princip centrálního repozitáře a navíc nám v jednoduchosti říká, že na každou funkcionalitu softwaru bychom měli vytvořit samostatnou větev, ve které ji budeme vyvíjet, a po jejím dokončení změny aplikovat zpět do hlavní větve. Toto je velice zajímavá metodika, která by se mohla začít aplikovat i na projektu Big Cloud.

Gitflow Workflow

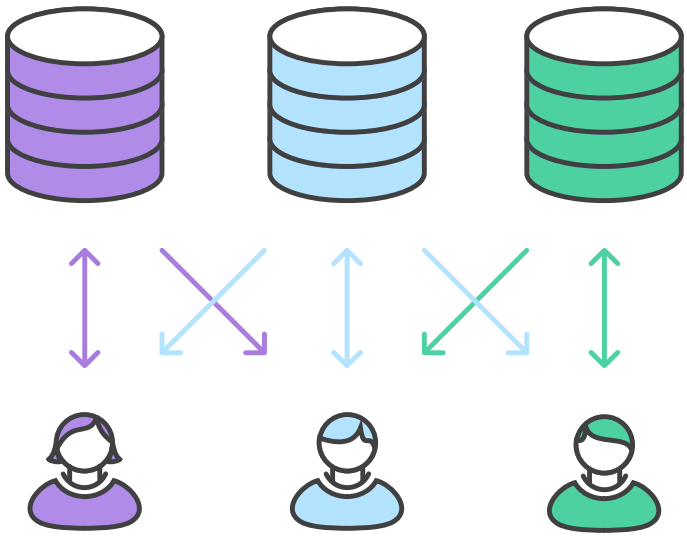
Opět rozšiřuje předchozí workflow a kromě větví pro samostatné funkcionality přidává i větve pro vydávání verzí softwaru a rychlé řešení chyb. Vzniká tak podobné větvení jako je vidět na obrázku 1.5. Podrobně je tento model popsán také v článku „A successful Git branching model"[26]. Pro projekt Big Cloud by toto byl, dle mého názoru, ideální stav.



Obrázek 1.5: Gitflow Workflow[4]

"Větvící" Workflow

Toto je poměrně odlišný přístup od všech předchozích. Každý má totiž nejen svojí lokální, ale i serverovou kopii repozitáře. Musí pak probíhat synchronizace mezi všemi navzájem, jak je naznačeno na obrázku 1.6. Přecházet na tento způsob workflow by bylo pro projekt Big Cloud velice pracné a nejspíš i zbytečné.



Obrázek 1.6: Rozštěpené Workflow[4]

1.3.2 Správa projektu

Pro management projektu Big Cloud je nasazen na samostatném serveru systém **Redmine**[8]. Jelikož projekt je již delší dobu vyvíjen a v systému jsou vedeny aktuální úkoly a další důležité informace i statistiky, byl by přechod velice obtížný a drahý. Proto se ve své práci ani nebudu zabývat jeho alternativami. Jen dodám, že systém slouží jak pro zadávání úkolů, či reportování nalezených chyb, tak i pro vedení a monitorování postupu verzí Big Cloudu nebo také pro vykazování odpracovaného času a tím i pro podklad měsíčních výplat. To je mimochodem další důvod, proč změna systému v podstatě nepřichází v úvahu.

1.3.3 Nasazování

Pro CD na projektu slouží, jak už bylo řečeno, nástroj Strider[25]. Jeho úkolem je automaticky nasadit novou verzi softwaru na příslušný ze serverů, popsaných v sekci 1.4.1, po každém commitu do větví *master*, *testing* a manuálně, po stisknutí tlačítka, lze nasadit novou verzi i z větve *production*. Propojení Strideru je realizované pomocí tzv. "Git Hooks"[27], díky kterým se Strider dozvídá o změnách v repozitáři a může je následně stáhnout i aplikovat.

Aplikace probíhá jednoduše vysvětleno tak, že se v kontextu ručně napsaného skriptu nejdříve za pomoci Composeru provede instalační proces projektu Big Cloud nataženého z repozitáře přímo do přidělené složky na serveru, kde běží Strider. Následně se pomocí SSH nahraje celá tato složka na příslušný server.

Celkově tento proces neshledávám úplně vhodným a i když by šel určitě vylepšit jako takový např. o fázi testování, přikláněl bych se spíše k variantě použití CI místo CD. Jelikož vše souvisí s Gitem a nezávislými servery, přechod na jiný systém nasazování je určitě možný, takže se jeho analýzou vyplatí zabývat.

Jenkins

Když se zmíní termín průběžná integrace, většině lidí se vybaví Jenkins[28]. Tento komplexní systém určený právě k tomuto účelu nabízí nespočet možností a mnoho dalších pluginů. Ovšem jeho případná instalace a nastavování by pro projekt Big Cloud nejspíše bylo dosti složité a krkolomné.

Gitlab-CI

Druhá alternativa, která se příhodně nabízí, je podpora CI přímo v GitLab[24], pomocí tzv. Gitlab-CI[29], které je jeho součástí. To je zároveň jeho první výhoda, protože není potřeba již nikde nic instalovat. Druhou velkou výhodou je, že celá jeho konfigurace se provádí jednoduše v rámci jediného konfiguračního souboru, který je součástí repozitáře samotného. A třetí výhodou je, že tento systém využívá na pozadí Docker[30], díky čemuž se dá např. pro testování pokaždé snadno sestavit nové a čisté běhově prostředí.

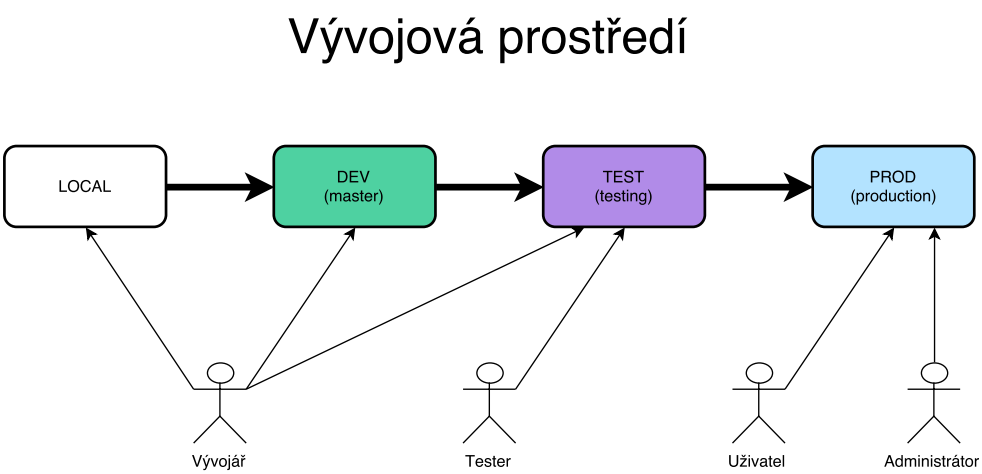
Z výše uvedeného plyne, že i když je Jenkins velice mocný nástroj, pro projekt Big Cloud by možná spíše stačilo správně nastavit GitLab-CI.

1.4 Technická struktura

V této části bych rád podrobněji popsal některé další aspekty projektu. Jedná se hlavně o podpůrné služby běžící samostatně na pozadí a způsob nasazování.

1.4.1 Vývojová prostředí

Jak už bylo několikrát naznačováno, projekt Big Cloud používá pro vývoj hned několik serverů, jejichž přehled můžete vidět ná obrázku 1.7. Je zde také vidět, že na každé prostředí mají přístup uživatelé s konkrétními rolemi a kompetencemi.



Obrázek 1.7: Schéma vývojových prostředí projektu Big Cloud

Server pro vývoj (DEV) navázaný na Git větev *master*, testování (TEST) spojený s větví *testing* a server s produkčním prostředím (PROD) využívající větev **production**. V rámci vývoje pak programátoři společnou práci po nasazení hned vidí na serveru DEV. Jakmile je připravená příslušná funkcionalita a verze je tak stabilní, udělá se Git merge větve *master* do *testing* a tím se automaticky pošle funkcionalita na server TEST pro další testování. Když je dostatečně otestována, pošle na produkční server PROD, opět pomocí Git merge odpovídajících větví a navíc zmáčknutím tlačítka ve Strider, kvůli potvrzení.

1.4.2 Podpůrné služby

Dalším již zmíněným konceptem jsou podpůrné služby projektu Big Cloud běžící nezávisle na pozadí. Jsou plně integrovány s projektem tj. frameworkem Nette, ale spouštějí konzolový skript jako samostatné procesy systému. To samo o sobě tolik nevadí, ovšem čas od času se stane, že některé tyto procesy se na systému např. špatně ukončují a pak tam zůstávají spuštěné i několikrát naráz, což už samozřejmě vadí, protože se nejspíše pak navzájem blokují. Každopádně všechny tyto služby fungují paralelně ke zbytku webové aplikace a tím pádem jsou kompletně asynchronní.

Jako alternativu k řešení služeb jako systémových procesů, která by pravděpodobně eliminovala většinu jejich problémů, vidím spouštění těchto služeb jako systémových démonů[31], jelikož aktuální spouštěcí skripty jsou stejně cíleny výhradně na systém Linux.

Nyní bych ještě pro úplnost vyjmenoval a stručně popsal jednotlivé služby:

Exchanger

Tato služba poskytuje především obsluhu pro API backendu IaaS služeb, kde synchronizuje stav serverů do lokální PostgreSQL databáze, odkud si data pak přebírá také samotná webová aplikace.

Billing

Stará se především o správu účtování poskytovaných služeb i produktů uživatelům systému. Např. denně účtuje taxu za vytvořené a běžící virtuální stroje apod. Opět pracuje s daty ve společné PostgreSQL databázi.

Mailer

Vykonává odesílání všemožných emailů v rámci projektu Big Cloud. Např. potvrzení registrace, ale i mnoho dalších...

Pohoda

Služba, která se stará o propojení s účetním systémem POHODA[32]. Tzn., že např. všechna vyúčtování provedené službou Billing je zároveň potřeba zahrnout do celkového účetnictví, kde se používá program POHODA. A právě pro předání takovýchto dat z databáze slouží tato služba.

1.5 Analýza kódu

V této části bych se rád zabýval analýzou kódu v projektu a především pak principům Nette. Co se týče implementace samotné a využití zvolených technologií, jsem poměrně kritický. Na úvod bych v rámci obecného popisu shrnul, že kód celkově obsahuje hodně nežádoucích duplicit a celková struktura projektu není příliš složitá, což ve zkratce znamená, že zde není výjimkou i přes dvacet souborů uvnitř jedné složky. Důsledkem toho se v této hierarchii pak obtížně něco hledá a díky duplicitám je potřeba z pravidla opravovat chyby na více místech. Často zde také dochází k porušování i navazujícímu nucenému obcházení principů objektového programování. Konkrétní možná řešení těchto problémů pak popisuji dále v 1.5.2.

1.5.1 Dokumentace

Moje první výtka k implementaci se týká celkové dokumentace kódu, protože pokud byl vůbec komentován, tak špatně, čímž je myšleno v rozporu s jakoukoliv normou pro použitý programovací jazyk, např. PHPDoc[33]. Použití tohoto způsobu komentování nám umožňuje u metod uvést jejich popis, definice parametrů i návratových hodnot a samozřejmě spoustu dalších atributů. Základní dokumentace, která by měla být uvedena alespoň u každé veřejné nebo chráněné metody, je vidět na ukázce 1. Pak lze také zdokumentovat samotné třídy, opět minimálně jejich popisem.

```php
<?php

/**
 * A summary informing the user what the associated element does.
 *
 * A *description*, that can span multiple lines, to go _in-depth_
 * into the details of this element and to provide some background
 * information or textual references.
 *
 * @param string $myArgument With a *description* of this argument,
 *     these may also span multiple lines.
 *
 * @return void
 */
function myFunction($myArgument)
{
}
```

Listing 1: Ukázka dokumentace kódu v PHPDoc[33]

1.5.2 Principy Nette

Za použití svých dosavadních zkušeností z jiných projektů s touto technologií jsem zanalyzoval stávající řešení a následně zde popíši principy, jejichž aplikace do projektu by mohla být velice přínosná. Celkově je tady velký prostor pro budoucí zdokonalování a já se samozřejmě pokusím všechny následující principy již aplikovat v SaaS i PaaS službách.

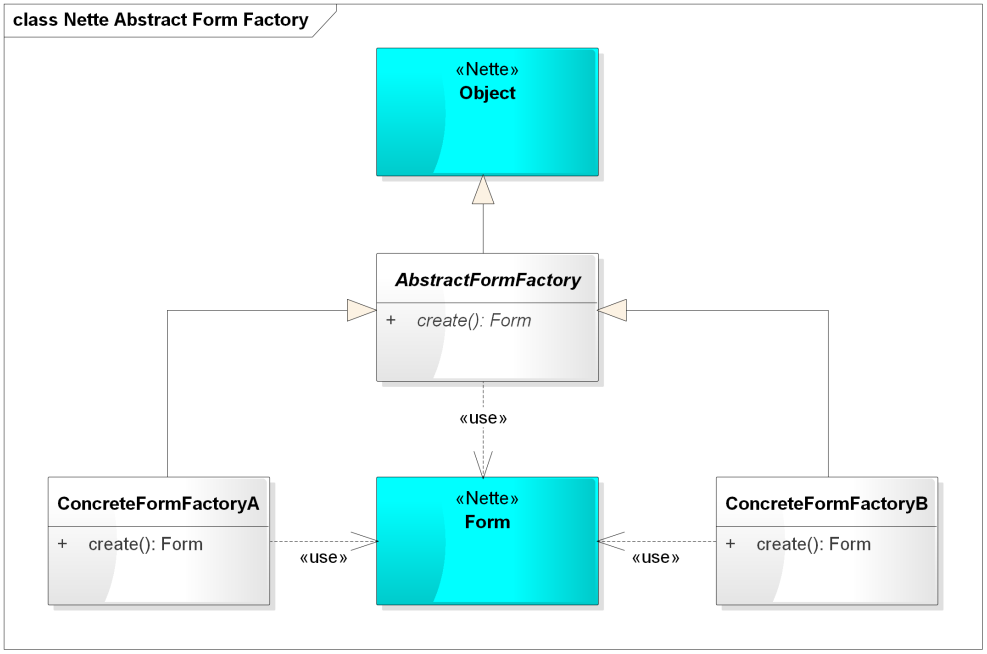
1.5.2.1 "Továrničky"

Jedná se o mírně upravenou aplikaci návrhového vzoru abstraktní továrny[34], v Nette slangově označovaném jako "továrničky", jež se standardně používají pro vytváření Nette komponent a podporují tzv. autowiring[35]. Tento vzor se

pak dá použít také pro vytváření instancí se stejnou konfigurací a jeho aplikace v projektu Big Cloud je velice žádaná, protože pomůže výrazně redukovat duplicity v kódu.

Konkrétně se dá použít např. při vytváření webových formulářů, které se pomocí Nette vytvářejí na úrovni presenterů. Aktuálně se v projektu často ten samý formulář vytváří dvakrát, jednou pro vytváření nové položky a po druhé pro její editaci, i když jsou tyto formuláře v podstatě identické. Nemluvě pak o vytváření takových stejných formulářů ještě na více místech aplikace. Duplicita kódu tak rychle stoupá, přičemž přehlednost úměrně klesá.

Při použití továrniček se pak všechen tento kód může univerzálně přesunout do samostatné třídy, na jedno místo, díky čemuž se formuláře stanou znovupoužitelnými. Dále, za pomocí dědičnosti, se společný kód dá snadno přesunout do rodičovské abstraktní třídy. Tento postup pak lze aplikovat na všechny formuláře v projektu a navíc bude toto nové vytváření formulářů plně kompatibilní s tím původním. Jednoduché vyjádření obecné továrničky na formuláře pomocí UML diagramu tříd můžete vidět níže na obrázku 1.8.



Obrázek 1.8: Diagram tříd pro obecné továrničky na formuláře v Nette

Existuje zde pak ještě vylepšená varianta při použití tzv. “generických továrniček”, kdy formulář obalíme do Nette komponenty, což umožní vytvářet tyto továrničky téměř automaticky[36].

1.5.2.2 Modální formuláře

Když už zmiňuji formuláře, tak jejich zobrazování je také realizováno velice zvláštně. Bylo zamýšleno, aby formuláře fungovaly na bázi AJAXu v rámci modálních oken na webové stránce. Ovšem výsledek je takový hybrid, který si bohužel vzal z obou přístupů to horší. Před otevřením modálního okna s formulářem se stránka celá obnoví. Stejně tak po jeho odeslání, či zavření, se modální okno zavře a pak se stejně provede obnova celé stránky. Takže tyto formuláře rozhodně nejde považovat za AJAXové, ale co je horší, bez JS vůbec nefungují, takže nejsou použitelné ani v bez AJAXové variantě.

Pro řešení SaaS i PaaS bych tedy chtěl formuláře vytvořit tak, aby jejich modální fungování probíhalo čistě přes AJAX tzn., aby se při jejich otevírání i zavírání stránka zbytečně neproblikávala kvůli bezdůvodnému přesměrování. Dále bych je chtěl inovovat ještě tak, aby fungovaly např. i pokud je na stránce JavaScript úplně vypnutý.

1.5.2.3 Dependency Injection (DI)

Dalším problémem bylo špatné nebo spíše žádné použití DI, která je ale jedním ze stavebních kamenů Nette. To vedlo například k ruční inicializaci všech tříd modelů a jejich vzájemnému předání úplně všude nebo ke zbytečnému několikanásobnému vytváření instancí tříd napříč celým programem. Všechny tyto třídy by však mohly mít díky Nette DI[37] jednu instanci v rámci celé aplikace.

V SaaS i PaaS službách se chci samozřejmě špatného zacházení s DI vyvarovat a dodržovat správné postupy uvedené v dokumentaci, jako jsou např. registrace služeb v rámci konfigurace a následné předávání závislostí konstruktory, kde Nette pomocí DI automaticky dosadí již inicializované instance.

1.5.2.4 Persistentní parametry

Posledním větším nedostatkem, lépe řečeno přebytkem, který jsem při analýze kódu původního řešení nalezl, bylo zbytečné nadužívání tzv. persisteních parametrů. To je funkcionalita Nette, která automaticky převádí parametry URL mezi jednotlivými dotazy v rámci presenteru i jeho potomků.

Její použití rozhodně není špatně, ale má své místo, např. pokud chceme v rámci URL parametru všude přenášet informaci o jazykové mutaci. Rozhodně není dobré definovat různé persistentní parametry všude, kde chceme přenést jakoukoliv informaci v přechodu mezi dvěma stránkami. To může v důsledku být nejen nepřehledné, ale i zpomalovat aplikaci jako takovou.

V rámci SaaS i PaaS služeb se takovému použití opět chci vyhnout a zavádět persistetní proměnné pouze tam, kde dávají smysl. Všude jinde můžu takovou informaci v URL předat ručně.

1.6 PaaS

Když už jsem byl dostatečně obeznámen se současným stavem projektu Big Cloud, mým úkolem se stala analýza požadavků jednotlivých služeb a začal jsem s kategorií PaaS.

1.6.1 Databázové úložiště

Bude umožňovat vytvářet, editovat a mazat databáze přes webové rozhraní projektu Big Cloud. K tomu je hlavně potřeba mít někde požadovaný databázový software nainstalovaný. Pro vytvoření jedné databáze je pak potřeba znát především její název, jméno a heslo administrátora. K ovládání celého databázového úložiště pak stačí být jeho administrátor a po připojení použít jazyk SQL. Další detaily už jsou specifikovány funkčními požadavky v následující sekci 1.6.1.1.

1.6.1.1 Funkční požadavky

F1.1 Vytváření databází
Bude možné vytvořit databáze s daným názvem, typem, přičemž zatím budou podporované MySQL[22] a PostgreSQL[11], heslem administrátora, velikostí alokovaného úložiště pro databázi, která ale nesmí přesáhnout maximální povolenou kvótu, a případným popisem. Po vytvoření uživatel najde vytvořenou databázi na příslušném přiděleném serveru a může se do ní přihlásit pod uživatelským účtem, který má stejné jméno jako je název databáze, a zvoleným heslem. Platí, že tento uživatel má plná administrátorská práva, ale pouze nad danou databází.

F1.2 Editace databází
Bude možné editovat heslo administrátora, velikost alokovaného úložiště, které logicky nesmí být menší, než již využitá velikost a větší, než maximální povolená kvóta, a popis. Název databáze, a s ním i jméno uživatele společné s typem databáze, jsou po jejím vytvoření neměnné.

F1.3 Mazání databází
Bude možné databáze kompletně smazat.

F1.4 Zobrazení přehledu databází
Uživatel bude mít k dispozici přehled všech svých vytvořených databází a uvidí u nich i informace o jejich aktuálně zabrané velikosti. Také zde bude jasná a přehledná informace o tom, jak i kde se může k dané databázi připojit a přihlásit.

F1.5 Filtrování a řazení přehledu databází
V rámci přehledu databází je bude možné filtrovat i řadit podle názvu, typu a využité i alokované velikosti.

F1.6 Zobrazení přehledu pro administrátora
V sekci systému pro administrátora by měl být přehled všech databází a jejich uživatelů, stejně tak jejich alokovaných velikostí.

F1.7 Filtrování a řazení přehledu pro administrátora
V administrátorském přehledu databází bude možná filtrace i řazení podle názvu, typu, uživatele, kterému patří, a použité i alokované velikosti.

F1.8 Zablokovaní databáze
Pro administrátora by zde měla být možnost konkrétní databázi dočasně zablokovat.

1.7 SaaS

Následovaly analýzy požadavků dalších služeb z kategorie SaaS, jež probíhaly velice obdobným způsobem jako u PaaS1.6.

1.7.1 ownCloud úložiště

Služba bude umožňovat vytvářet, editovat a mazat datová úložiště ownCloud[7]. Pro tyto účely je důležité znát především parametry jako webová adresa, pod kterou se úložiště bude nacházet, přihlašovací údaje jeho administrátora a samozřejmě alokovaná velikost, kterou bude mít. Instalace takového úložiště je pak možná pouhým kopírováním jeho zdrojových kódů, jelikož je psané jako webová aplikace v jazyce PHP. Pro konfiguraci jeho konkrétní instance se pak dá použít RESTful API[38], které nabízí. Další požadované vlastnosti služby už jsou pak opět specifikovány funkčními požadavky v následující sekci 1.7.1.1.

1.7.1.1 Funkční požadavky

F2.1 Vytváření ownCloud úložiště
Bude možné vytvořit jej s daným názvem, doménou, na které poběží, alokovanou velikostí, která nesmí přesáhnout maximální kvótu, heslem administrátora a popisem. Název bude sloužit pouze pro uživatele a dá se eventuálně měnit, přičemž by se z něj teoreticky dal při vytváření odvodit název domény. Po vytvoření úložiště ho pak uživatel najde pod zvolenou doménou, přičemž zatím budou povoleny pouze subdomény na firemních serverech, ale do budoucna se to může změnit a mělo by se s tím počítat. Z toho plyne, že adresa úložiště bude mít tvar *<doména>.<server>.bigcloud.cz* a přihlásit se do něj může jako uživatel “admin” s heslem, které si zvolil.

F2.2 Editace ownCloud úložiště
U úložiště bude možné měnit název, velikost, heslo administrátora a popis, přičemž doména bude logicky neměnná a změna velikosti úložiště bude podléhat

kontrolám kvót. Nesmí být menší, než již využitá velikost a větší, než maximální povolená kvóta.

F2.3 Mazání ownCloud úložiště
Bude možné úložiště kompletně smazat.

F2.4 Zobrazení přehledu databází
Uživatel bude mít k dispozici přehled všech svých vytvořených ownCloud úložišť a uvidí u nich i informace o jejich aktuálně zabrané velikosti. Také zde musí najít jasnou a přehlednou informaci, jak se do daného ownCloud úložiště přihlásí, případně odkaz na webovou adresu, pod kterou ho najde.

F2.5 Filtrování a řazení přehledu ownCloud úložišť
V rámci přehledu svých datových úložišť je bude uživatel moci filtrovat i řadit podle názvu, domény, použitého i celkového alokovaného prostoru.

F2.6 Zobrazení přehledu pro administrátora
V sekci systému pro administrátora by pak měl být přehled všech ownCloud úložišť a jejich uživatelů, stejně tak velikostí, které mají alokované, i které aktuálně zabírají.

F2.7 Filtrování a řazení přehledu pro administrátora
V administrátorském přehledu úložišť bude možná filtrace i řazení podle názvu, domény, uživatele, kterému patří a použité i alokované velikosti úložiště.

F2.8 Zablokovaní ownCloud úložiště
Pro administrátora by zde měla být možnost konkrétní úložiště dočasně zablokovat.

F2.9 Aktualizace zabrané velikosti ownCloud úložišť
V rámci podpory kvalitního přehledu uživatelů i administrátora je potřeba pravidelně a často aktualizovat informaci o momentálně zabraných velikostech úložišť.

1.7.2 Vzdálená plocha Linux

Tato služba bude sloužit pro správu vzdálené plochy Linuxu, který bude umožňovat vytvářet, editovat a mazat uživatele linuxového operačního systému. K tomu stačí mít tento linuxový systém nainstalovaný. Pro vytvoření takového uživatele je pak potřeba znát v podstatě pouze jeho jméno a heslo. K ovládání tohoto systému pak stačí pouze jeho systémové příkazy pro vytváření uživatelů, změnu jejich hesel apod. Další detaily už jsou specifikovány funkčními požadavky v následující sekci 1.7.2.1.

1.7.2.1 Funkční požadavky

F3.1 Vytváření uživatele na vzdálené ploše Linux
Uživatele na vzdálené ploše Linux bude možné vytvořit s daným uživatelským jménem a heslem, což reprezentuje jeho přihlašovací údaje do systému, a popisem. Po vytvoření se za něj následně bude možné přihlásit do Linuxového operačního systému běžícím na jednom z firemních serverů. Vytváření uživatelů by tak mělo být rozloženo na několik takových serverů, kvůli vyvažování výkonu, přičemž je ale potřeba u každého uživatele interně vědět, na kterém systému byl vytvořen, aby bylo jasné, kde se může přihlásit.

F3.2 Editace uživatele na vzdálené ploše Linux
U uživatele na vzdálené ploše Linux bude možné měnit jeho uživatelské jméno i heslo a popis.

F3.3 Mazání uživatele na vzdálené ploše Linux
Uživatele na vzdálené ploše Linux bude možné smazat.

F3.4 Nastavování stavu uživatele na vzdálené ploše Linux
Uživatel bude moci nastavit stav uživatelského účtu vzdálené plochy na aktivního či neaktivního podle potřeby. Pokud je uživatel neaktivní, nemělo by jít se za něj přihlásit.

F3.5 Zobrazení přehledu uživatelů na vzdálené ploše Linux
Uživatel webového rozhraní Big Cloud bude mít k dispozici přehled svých uživatelů vzdálené plochy Linux. Také zde musí najít, již zmíněnou, informaci, na jakém serveru se může za uživatele vzdáleně přihlásit.

F3.6 Filtrování uživatelů na vzdálené ploše Linux
V rámci přehledu uživatelů na vzdálené ploše Linux bude možné filtrovat i řadit podle jména a stavu.

F3.7 Zobrazení přehledu pro administrátora
V sekci systému pro administrátora by pak měl být přehled všech uživatelů Linuxové vzdálené plochy a jejich vlastníků.

F3.8 Filtrování a řazení přehledu pro administrátora
V administrátorském přehledu uživatelů vzdálené plochy Linux bude možná filtrace i řazení tohoto výpisu podle jmen uživatelů i vlastníků a samozřejmě serverů, kde se nacházejí.

F3.9 Zablokovaní uživatele na vzdálené ploše Linux
Pro administrátora by zde měla být možnost konkrétního Linuxového uživatele deaktivovat z pozice administrátora, tzn. že znovu aktivovat ho může zase jen administrátor a ne jeho vlastník. Je tedy potřeba rozlišit, kdo deaktivaci provedl.

1.7.3 Redmine

Poslední služba pak bude umožňovat vytvářet, editovat a mazat vlastní Redmine[8], což je nástroj sloužící pro projektový management. Tyto účely vyžadují znát především parametry jako webová adresa, pod kterou se Redmine bude nacházet, přihlašovací údaje jeho administrátora a samozřejmě velikost, kterou bude mít. Instalace takového úložiště je pak možná zase pouhým kopírováním jeho zdrojových kódů, přičemž pro konfiguraci jeho konkrétní instance se nabízí opět REST API[39]. Obecně požadavky této služby jsou velice podobné těm, které má služba ownCloud 1.7.1. Další požadované vlastnosti služby už jsou pak opět specifikovány funkčními požadavky v následující sekci 1.7.3.1.

1.7.3.1 Funkční požadavky

F4.1 Vytváření Redmine
Bude možné vytvořit jej s danou doménou, na které poběží, pod daným uživatelským názvem, s alokovanou velikostí, která nesmí přesáhnout maximální kvótu a libovolným uživatelským jménem a heslem, které bude sloužit pro výchozího administrátora daného systému, a popisem. Po vytvoření pak uživatel Redmine najde pod zvolenou doménou, přičemž zatím budou opět povoleny pouze subdomény na firemních serverech, ale do budoucna se to může zase změnit a mělo by se s tím počítat. Tzn. že adresa úložiště bude mít tvar *<doména>.redmine.bigcloud.cz* a přihlásit se do něj může jako uživatel se jménem a heslem, které si zvolil.

F4.2 Editace Redmine
U Redmine bude možné editovat uživatelský název, velikost, jméno i heslo administrátora a popis, přičemž doména bude logicky neměnná a změna velikosti bude podléhat kontrolám kvót a nemůže být menší, než již zabraná velikost.

F4.3 Mazání Redmine
Bude možné Redmine kompletně smazat.

F4.4 Zobrazení přehledu instancí Redmine
Uživatel bude mít k dispozici přehled všech svých vytvořených instancí Red-

mine a uvidí u nich i informace o jejich aktuálně zabrané velikosti. Také zde musí najít jasnou a přehlednou informaci, jak se do daného Redmine přihlásí, případně odkaz na webovou adresu, pod kterou ho najde.

F4.5 Filtrování a řazení přehledu instancí Redmine
V rámci přehledu svých instancí Redmine si je bude moci uživatel filtrovat i řadit podle názvu, domény, použitého i celkového prostoru a jména jejich administrátora.

F4.6 Zobrazení přehledu pro administrátora
V sekci modulu pro administrátora by pak měl být přehled všech instancí Redmine a jejich uživatelů, stejně tak velikostí, které mají alokované, i které aktuálně zabírají.

F4.7 Filtrování a řazení přehledu pro administrátora
V administrátorském přehledu úložišť bude možná filtrace i řazení podle názvu, domény, uživatele, kterému patří i jeho administrátorského uživatele a alokované i využité velikosti úložiště.

F4.8 Zablokovaní Redmine
Pro administrátora by zde měla být možnost konkrétní Redmine dočasně zablokovat.

Kapitola 2

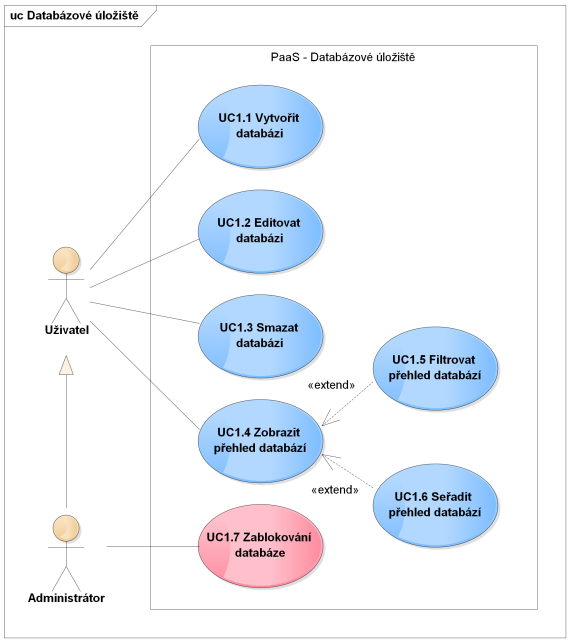
Návrh

Tato část mé práce už se věnuje návrhu výhradně nových SaaS a PaaS služeb. Nejdříve jsem sestavil jejich případy užití, na nichž jsem následně provedl kontroly splnění požadavků. Posléze jsem také pro jednotlivé služby vytvořil jednoduché doménové modely, hlavně pro upřesnění jejich dílčích atributů. V další části jsem pak vyhotovil ještě diagramy nasazení, opět pro každou službu zvlášť, a na úplný závěr vizuální návrh jejich uživatelského rozhraní v podobě wireframů.

2.1 Případy užití

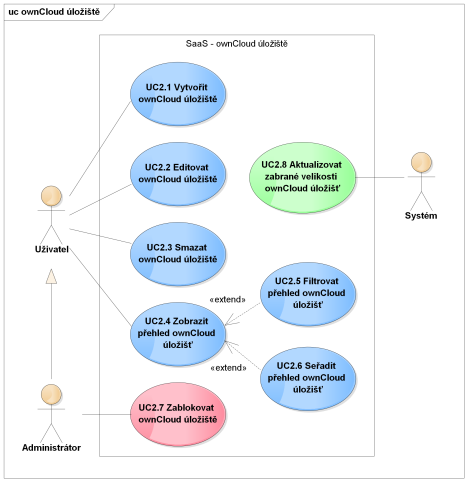
Začal jsem s tvorbou diagramů případů užití, které vznikaly na základě požadavků zadavatele. Diagramy jsem podle zvyklosti opět rozdělil dle jednotlivých SaaS i PaaS služeb. Dále jsem k nim dodělal příslušné aktéry, kteří služby mohou v systému vykonávat. Pro přehlednost diagramů jsem u těchto aktérů použil dědičnost.

Databázové úložiště



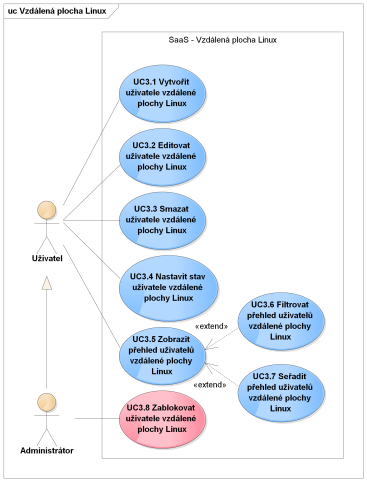
Obrázek 2.1: Diagram případů užití pro Databázové úložiště (PaaS)

ownCloud úložiště



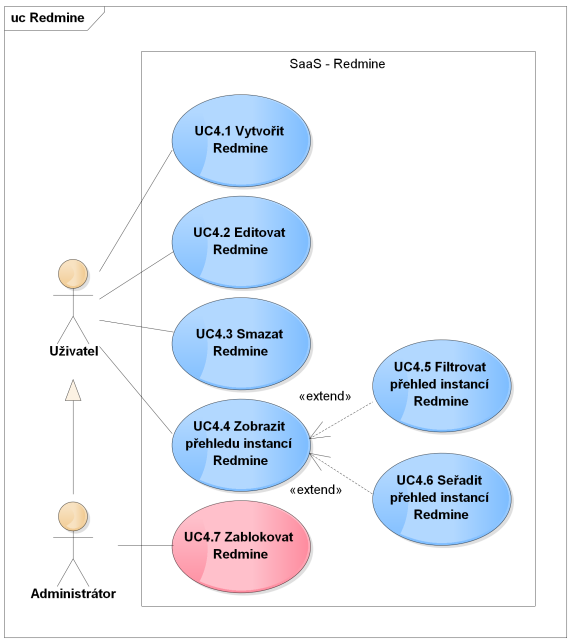
Obrázek 2.2: Diagram případů užití pro ownCloud úložiště (SaaS)

Vzdálená plocha Linux



Obrázek 2.3: Diagram případů užití pro Vzdálenou plochu Linux (SaaS)

Redmine



Obrázek 2.4: Diagram případů užití pro Redmine (SaaS)

2.2 Kontrola splnění požadavků

Pro kontrolu splnění požadavků jednotlivých služeb jsem použil následující přehledné tabulky, kde je jasně vidět jejich vztah s případy použití. Je tak možné zkontrolovat, že se na nic nezapomnělo.

Databázové úložiště

	Případy užití						
Požadavky	UC1.1	UC1.2	UC1.3	UC1.4	UC1.5	UC1.6	UC1.7
F1.1	+						
F1.2		+					
F1.3			+				
F1.4				+			
F1.5					+	+	
F1.6				+			
F1.7					+	+	
F1.8							+

Tabulka 2.1: Kontrola splnění požadavků pro Databázové úložiště (PaaS)

ownCloud úložiště

	Případy užití							
Požadavky	UC1.1	UC1.2	UC1.3	UC1.4	UC1.5	UC1.6	UC1.7	UC1.8
F1.1	+							
F1.2		+						
F1.3			+					
F1.4				+				
F1.5					+	+		
F1.6				+				
F1.7					+	+		
F1.8							+	
F1.9								+

Tabulka 2.2: Kontrola splnění požadavků pro ownCloud úložiště (SaaS)

Vzdálená plocha Linux

	Případy užití							
Požadavky	UC1.1	UC1.2	UC1.3	UC1.4	UC1.5	UC1.6	UC1.7	UC1.8
F1.1	+							
F1.2		+						
F1.3			+					
F1.4				+				
F1.5					+			
F1.6						+	+	
F1.7					+			
F1.8						+	+	
F1.9								+

Tabulka 2.3: Kontrola splnění požadavků pro Vzdálenou plochu Linux (SaaS)

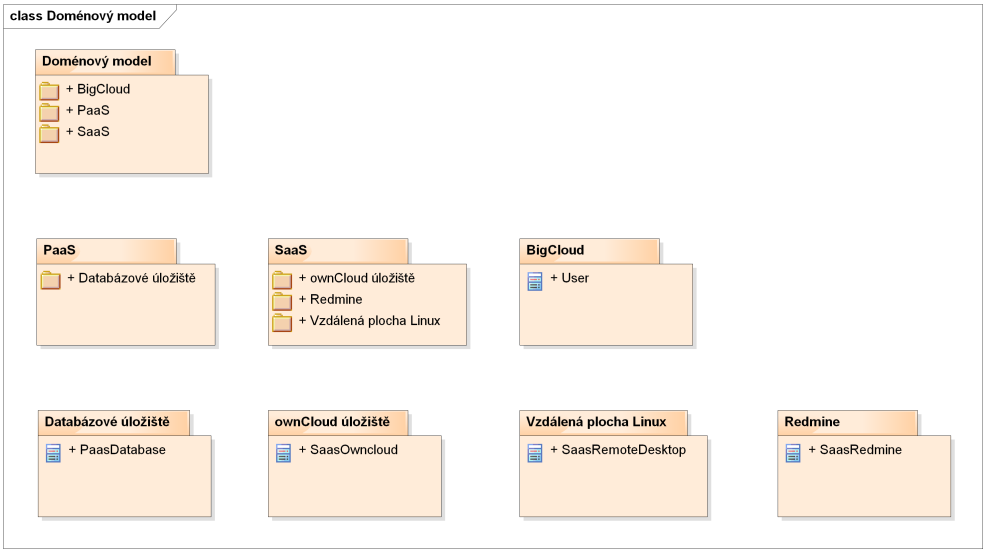
Redmine

	Případy užití						
Požadavky	UC1.1	UC1.2	UC1.3	UC1.4	UC1.5	UC1.6	UC1.7
F1.1	+						
F1.2		+					
F1.3			+				
F1.4				+			
F1.5					+	+	
F1.6				+			
F1.7					+	+	
F1.8							+

Tabulka 2.4: Kontrola splnění požadavků pro Redmine (SaaS)

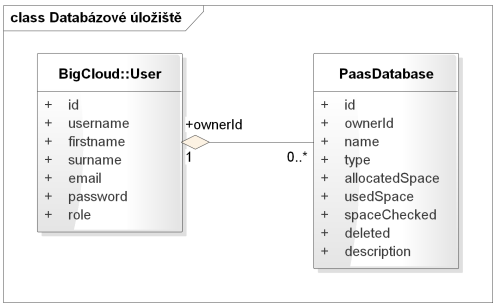
2.3 Doménový model

Jako další diagram, který jsem zvolil, je doménový model. Rozdělil jsem ho zase podle jednotlivých služeb, ale celkový přehled můžete vidět na diagramu níže 2.5. Celkově jsou tyto diagramy velice jednoduché a slouží především pro ujasnění atributů jednotlivých entit.



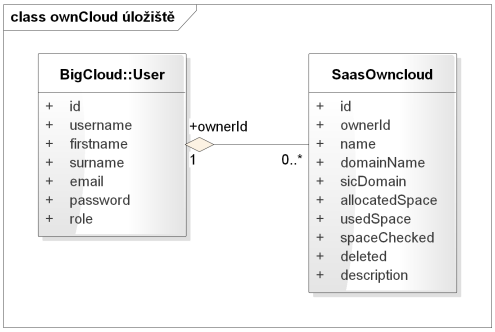
Obrázek 2.5: Přehled hierarchie doménových modelů

Databázové úložiště



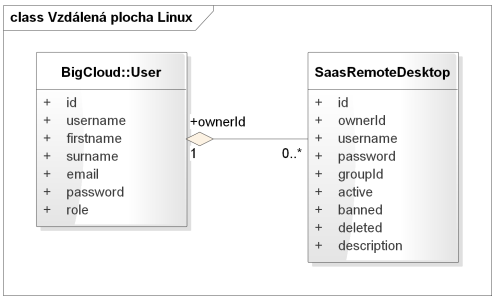
Obrázek 2.6: Doménový model pro Databázové úložiště (PaaS)

ownCloud úložiště



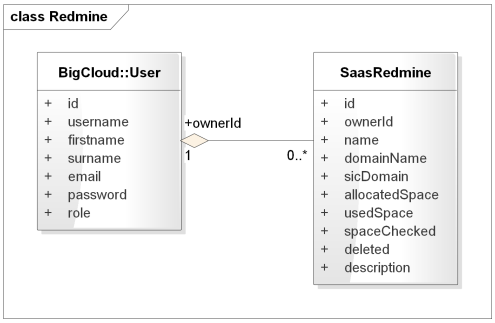
Obrázek 2.7: Doménový model pro ownCloud úložiště (SaaS)

Vzdálená plocha Linux



Obrázek 2.8: Doménový model pro Vzdálenou plochu Linux (SaaS)

Redmine

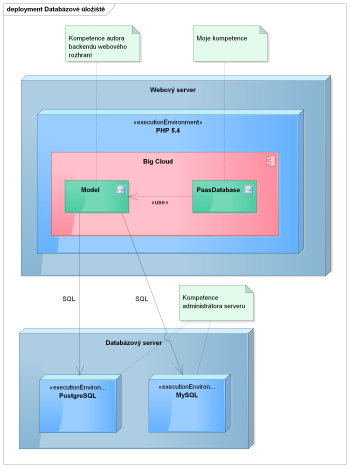


Obrázek 2.9: Doménový model pro Redmine (SaaS)

2.4 Nasazení

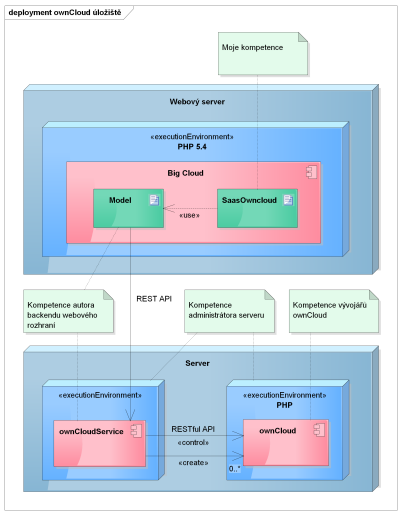
Účel následujících diagramů je zhruba popsat finální nasazení a s tím hlavně komunikaci jednotlivých služeb s okolím pro lepší představu fungování celého systému.

Databázové úložiště



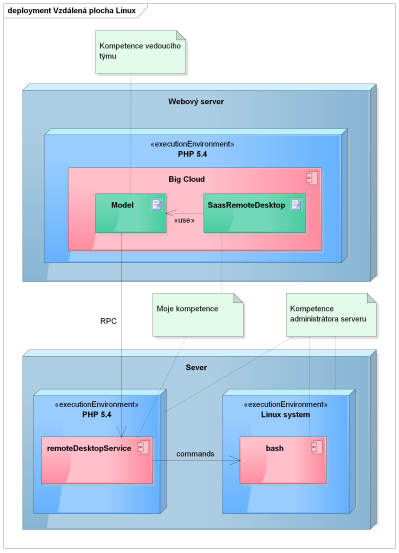
Obrázek 2.10: Diagram nasazení pro Databázové úložiště (PaaS)

ownCloud úložiště



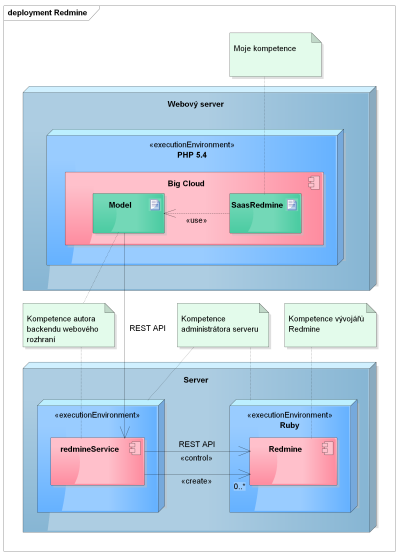
Obrázek 2.11: Diagram nasazení pro ownCloud úložiště (SaaS)

Vzdálená plocha Linux



Obrázek 2.12: Diagram nasazení pro Vzdálenou plochu Linux (SaaS)

Redmine



Obrázek 2.13: Diagram nasazení pro Redmine (SaaS)

2.4.1 Shrnutí

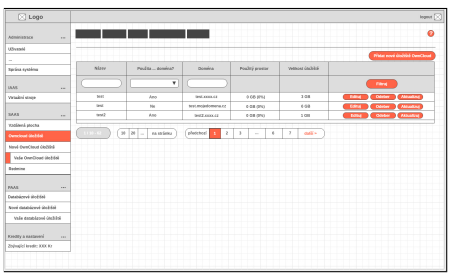
Všechny služby momentálně komunikují synchronně, což znamená, že se vždy čeká na odpověď a v systému se během tohoto čekání nedá nic dělat. To by obecně mohl být problém, např. pokud bude některá operace trvat příliš dlouho, budu muset uživatel i dlouho čekat.

Rozdílný asynchronní přístup se používá u služeb IaaS, kde se pošle požadavek a až je splněn, systém pouze upozorní uživatele. Během toho s ním může nadále normálně pracovat. Do budoucna by tak stálo za úvahu, pokud se některá ze služeb SaaS nebo PaaS ukáže jako příliš pomalá, zavést u ní stejný asynchronní přístup.

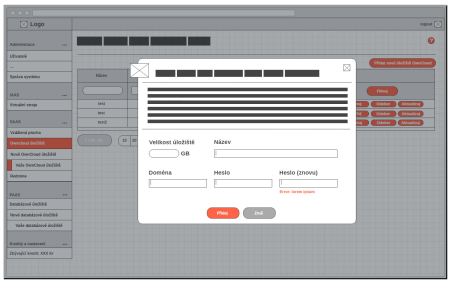
2.5 Wireframy

Jako poslední jsem vyhotovil wireframy pro návrh samotného uživatelského rozhraní SaaS i PaaS služeb a to tak, aby co nejvíce zapadaly do původního designu celého projektu Big Cloud.

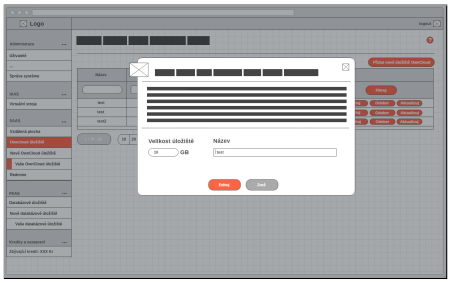
Jako ukázku zde uvedu wireframy pouze pro jednu službu a to ownCloud z kategorie SaaS. Je to proto, abych zbytečně neplýtval místem na velice podobné ukázky. Zbytek wireframů i pro ostatní služby naleznete samozřejmě na přiloženém CD ve složce `src/wireframes/`.



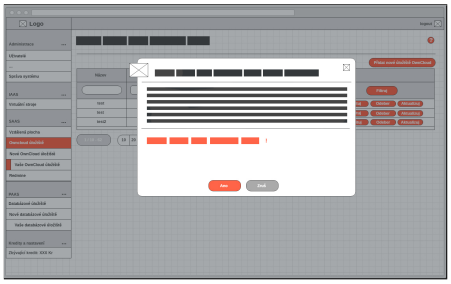
Obrázek 2.14: Přehled ownCloudů



Obrázek 2.15: Přidání ownCloudu



Obrázek 2.16: Editace ownCloudu



Obrázek 2.17: Smazání ownCloudu

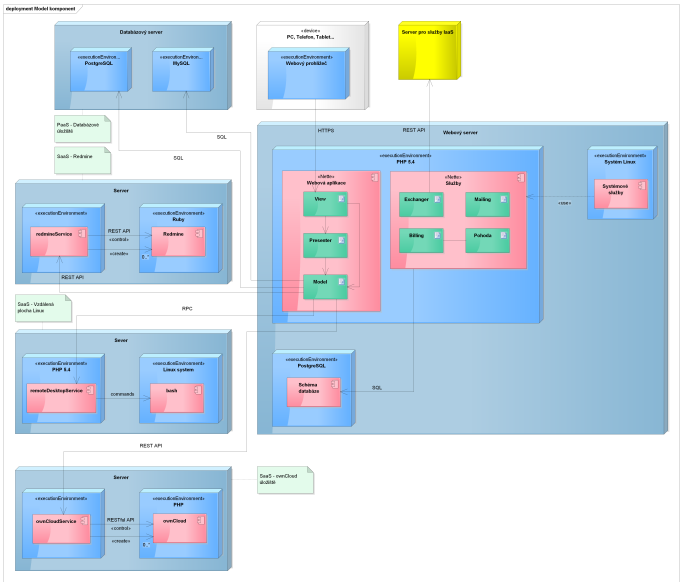
Kapitola 3

Realizace

V této části bych se chtěl věnovat především výstupům z realizační fáze. Nejdříve ukáži kompletní blokové schéma projektu Big Cloud po implementaci všech požadovaných SaaS i PaaS služeb. Dále obrázky GUI jako výstup z realizace wireframů a nakonec uvedu informace o výstupních příručkách jako je instalační, programátorská i uživatelská. Samotné zdrojové kódy implementace potom najdete na přiloženém CD ve složce `src/impl/`.

3.1 Blokové schéma celého projektu Big Cloudu

Následující blokové schéma na obrázku 3.1 popisuje celkovou strukturu projektu Big Cloud, včetně nově realizovaných služeb.

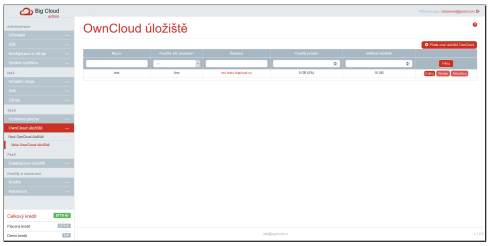


Obrázek 3.1: Model komponent a jejich komunikace pro celý projekt Big Cloud

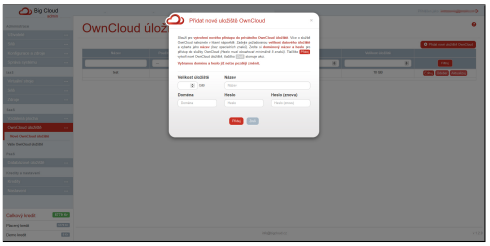
Je vidět, že projekt se stal díky novým SaaS i PaaS službám mnohem robustnější a komplexnější, než před tím. Také je zde vyznačené napojení na IaaS služby, jejichž rozbor ale nebyl součástí mé práce.

3.2 GUI

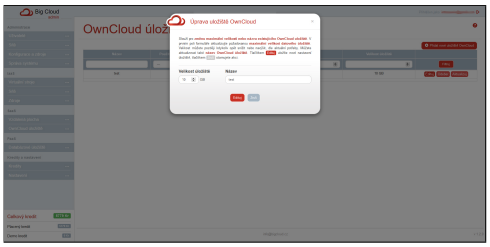
Dále přidávám několik obrázků již implementovaného uživatelského rozhraní. Opět pouze pro službu ownCloud úložiště z kategorie SaaS a to proto, aby je bylo možné porovnat s wireframy v sekci 2.5. Zbytek obrázků GUI dalších služeb naleznete samozřejmě na přiloženém CD v uživatelské příručce (`text/attach/user-manual.pdf`).



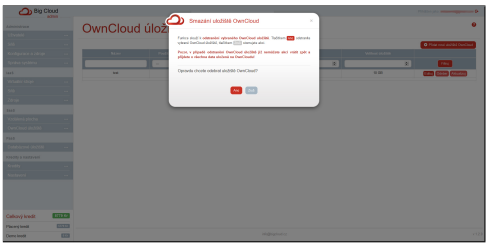
Obrázek 3.2: Přehled ownCloudů



Obrázek 3.3: Přidání ownCloudu



Obrázek 3.4: Editace ownCloudu



Obrázek 3.5: Smazání ownCloudu

3.3 Instalační příručka

V této části bych chtěl stručně něco říci o instalaci a konfiguraci jednotlivých služeb v rámci projektu Big Cloud. Jak jste mohli vidět v úvodu na obrázku 3.1, který shrnuje celkové rozložení projektu, má v podstatě každá SaaS, PaaS i IaaS vlastní server, na který se instaluje a který se musí správně nakonfigurovat. Z toho plyne, že celková instalace je poměrně náročná záležitost.

Dále bych tedy popsal pouze obecné hlavní kroky instalace pro jednotlivé SaaS a PaaS služby, přičemž detailnější popis instalace je popsán v instalační příručce (`text/attach/instal-manual.pdf`), která se nachází na CD.

Databázové úložiště

Na příslušný server je potřeba nainstalovat všechny podporované databáze, v tomto konkrétním případě PostgreSQL a MySQL. Dále v nich vytvořit administrátorského uživatele a povolit vzdálený přístup.

V nainstalovaném projektu Big Cloud je posléze potřeba nastavit tyto přístupy v konfiguračním souboru pod sekcí pro tuto službu.

ownCloud úložiště

Opět na serveru, kde má běžet tato služba, je nejprve nutné zavést skripty pro kopírování i nastavování instancí ownCloud, poté nainstalovat jeho běhové prostředí a posléze je třeba zkopírovat vzorovou instanci, tím jsou myšleny stažené zdrojové kódy, na příslušné místo. Zavedené skripty pak vystaví REST API rozhraní, pro které je třeba umožnit vnější přístup.

Na projektu Big Cloud se pak do konfiguračního souboru v příslušné sekci pro tuto službu nastaví informace o tomto API, což umožní ovládání vzdálených skriptů na serveru a tím vytváření nakonfigurovaných kopií úložišť z originální instance.

Vzdálená plocha Linux

Na vzdáleném serveru s operačním systémem Linux, je nejdříve potřeba nastavit konfiguraci pro nově vytvářené uživatele tzn. např. kde se bude zakládat jejich domovský adresář i jakou bude mít maximální velikost nebo jaká budou mít přístupová práva a počáteční nastavení. Následně se na serveru zavedou skripty, které budou komunikovat s aplikací Big Cloud přes RPC rozhraní a systém Linux ovládat pomocí spouštění konzolových příkazů, k čemuž musí mít nastavena příslušná oprávnění.

Na straně Big Cloudu je pak pro tuto službu také podporováno RPC rozhraní pro komunikaci s uvedeným vzdáleným serverem, které tak na něm umožní vytvářet i spravovat Linuxové uživatele.

Redmine

Tato služba se instaluje velice podobně jako ownCloud. Nejdříve se zavedou skripty pro kopírování i nastavování instancí Redmine, poté je potřeba nainstalovat jeho běhové prostředí a posléze zkopírovat jeho vzorovou instanci, tj. zdrojové kódy, na příslušné místo. Zavedené skripty pak opět vystaví REST API rozhraní pro ovládání této služby, přičemž pro toto rozhraní je potřeba povolit vnější přístup.

V rámci projektu Big Cloud se poté zase do konfiguračního souboru v příslušné sekci pro službu Redmine nastaví informace o vystaveném API. To umožní ovládání vzdálených skriptů a tím vytváření i následnou konfiguraci kopií Redmine z originální instance.

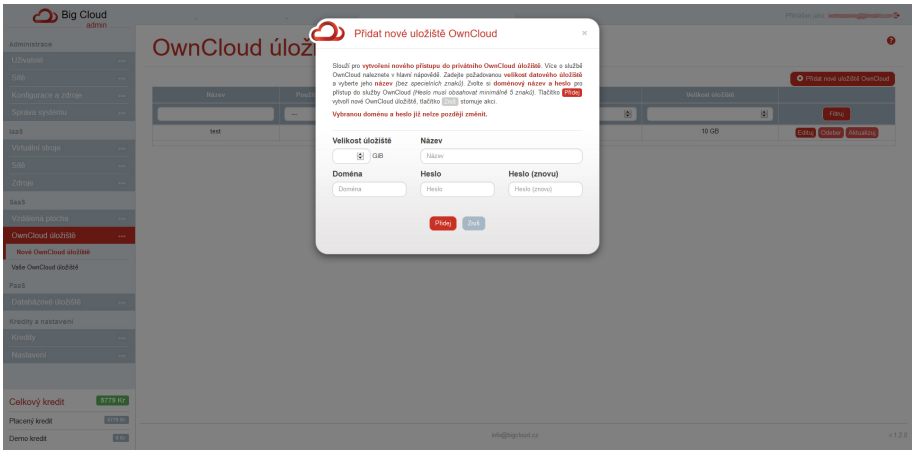
3.4 Programátorská dokumentace

Další konkrétní informace k samotné implementaci jsou k nalezení v programátorské dokumentaci, která byla vygenerována za pomocí PHPDoc[33] přímo ze zdrojových kódů, a nachází se na CD ve složce `src/prog-manual/`.

3.5 Uživatelská příručka

Na přiloženém CD je také zhotovena kompletní uživatelská příručka (`text/attach/user-manual.pdf`) pro všechny SaaS i PaaS služby. Pro ukázku zde na další straně uvádím jednu stránku přímo z příručky, která popisuje vytvoření nového ownCloud úložiště.

1. Vytvoření nového ownCloud úložiště



V administračním prostředí Big Cloudu, v sekci ,,ownCloud úložiště →Vaše ownCloud úložiště,, zvolíte položku ,,Vytvořit nové ownCloud úložiště“. Ve formuláři je třeba definovat požadovanou maximální velikost cloudového úložiště. Později ji můžete kdykoliv znovu změnit. Dále je třeba vybrat si název pro nové ownCloud úložiště a specifikovat doménu, pod kterou budete k vytvořenému cloud účtu přistupovat (jednoduchý název bez speciálních znaků, minimálně však musí obsahovat 3 znaky). Pokud nespecifikujete vlastní doménový název, systém Vám automaticky vygeneruje přístupovou doménu s koncovkou BigCloud.cz. V neposlední řadě musí uživatel zvolit heslo pro přístup do služby ownCloud (minimální požadavek pro délku hesla je 5 znaků). Nastavená přístupová doména a heslo nového ownCloud účtu nejde později změnit. Červeným tlačítkem ,,Přidej“ v dolní části formuláře vytvoříte nový ownCloud účet a po automatickém uzavření modal okna se Vám v hlavní administrační tabulce služby objeví nový záznam ownCloud účtu.

Kapitola 4

Testování

Pro automatizované testování na projektu Big Cloud byl použit poměrně populární framework Codeception[40] pro PHP. Ten umožňuje realizovat všechny kategorie požadovaných testů, kterými jsou jednotkové, integrační i akceptační.

Jediné, co v sobě framework neobsahuje, je širší podpora mockování, které se používá hlavně v oblasti jednotkového testování pro izolaci jednotlivých testů, takže pro tento účel byl použit doplňkový mockovací framework Mockery[41].

Poslední, ještě nezmíněnou skupinou vyžadovaných testů, jsou testy použitelnosti, které se ovšem z logických důvodů provádějí manuálně mimo implementaci.

Následně se tedy v této kapitole budu zabývat jednotlivými skupinami testů a jejich realizací na službách, tentokrát však v kontextu všech služeb dohromady. Implementaci samotných testů poté můžete opět nalézt na přiloženém CD ve složce `src/tests/`.

4.1 Jednotkové (Unit) testy

Základní kategorie testů, která se používá pro ověření funkcionality jednotlivých tříd a jejich metod. Testy lze pouštět např. i pokud aplikace jako taková není ještě hotová, ba ani spustitelná. To ale znamená, že je potřeba tyto testy poctivě tzv. izolovat od závislostí právě testované třídy na ostatních. K tomu se často využívá technika mockování popsaná v následující kapitole 4.1.1.

Pro jednotkové testy se na projektu používá, jak již bylo řečeno v úvodu, framework Codeception, který z velké části rozšiřuje základní PHP framework pro jednotkové testy PHPUnit[42].

Realizace na službách

Těmito testy byly pokryty především presentery jednotlivých služeb a dále pak komponenty i jejich "továrničky", kde se testovalo, zda metody daných tříd vracejí očekávané výstupy. Celkově se však těchto testů v porovnání s ostatními skupinami používá poskrovnu.

4.1.1 Mockování

Technika sloužící především k izolaci testů daných tříd od jejich závislostí na zbytku aplikace. Používá se především u jednotkových testů, protože u ostatních kategorií se již převážně testuje i správné fungování těchto závislostí.

Realizace na službách

V rámci jednotkových testů SaaS a PaaS služeb je tato technika implementována pomocí frameworku Mockery na příhodných místech, např. na izolaci závislostí v konstruktoru, standardně předávaných pomocí DI.

4.2 Integrační testy

Další kategorie testů, která se využívá především pro testování komunikace jednotlivých částí systému. Pro tyto testy nemá framework Codeception žádnou speciální kategorii, ale dají se zde využít klasické jednotkové testy s mnohem menším poměrem izolace.

Realizace na službách

U služeb se tyto testy používají např. na testování komunikace modelu a databáze nebo modelu a API rozhraních dané služby. Celkově je tato kategorie testů na projektu Big Cloud i na jeho službách hojně zastoupena.

4.3 Akceptační testy

Tyto testy se poměrně liší od těch předchozích a testují aplikaci jako celek na té nejvyšší úrovni a používají se především na testování webového uživatelského rozhraní. Testy fungují jednoduše nastíněno tak, že umí přistupovat k webové aplikaci přes internetový prohlížeč, čímž se tváří jako běžný uživatel, ale na rozdíl od něj provádějí všechny úkony v prohlížeči automatizovaně.

Selenium

Codeception pro tyto testy využívá framework Selenium[43] a deleguje na něm všechny úkony. Selenium je pak jeden z nejrozšířenějších nástrojů pro tento druh automatizovaných testů a umí pracovat na různé způsoby, s různými prohlížeči i jejich verzemi a dokonce i bez prohlížeče, tzv. "headless", pouze na nějakém prohlížečovém jádru.

Všech těchto vlastností se dá v Codeception využít a navíc k nim tento framework doplňuje ještě jednoduché programátorské rozhraní a při jeho použití vypisuje i textový proces testování v uživatelský srozumitelné podobě. Z něj je potom jasně vidět, např. pokud nastane chyba, kde konkrétní test selhal a co se pokazilo.

Realizace na službách

Na SaaS a PaaS službách, jak již bylo zmiňováno, se tyto testy využívají především k ověření funkčnosti jejich webového rozhraní. Obecně pak mají tyto testy na projektu Big Cloud i na jeho službách velké zastoupení.

4.4 Testy použitelnosti

Velice používaná technika testování, která ověřuje nejen samotnou funkčnost systému, ale hlavně zkoumá reakce uživatele při samotném testování. Tyto testy se na projektu Big Cloud jako takovém nepoužívají, ale v rámci své práce jsem je samozřejmě chtěl realizovat pro SaaS i PaaS služby.

Příprava testovacích scénářů

Prvním krokem je vytvoření vstupního i výstupního dotazníku a především testovacích scénářů. Jednostrannou ukázku pro SaaS službu ownCloud úložiště můžete vidět na následující straně. Ostatní scénáře i dotazníky pro všechny další služby pak opět naleznete na přiloženém CD ve složce `test-scenarios/`.

Průběh testování

Samotné testování se pak bohužel nestihlo, ale je plánované na léto tohoto roku (2016) a bude pravděpodobně probíhat v laboratoři pro testování použitelnosti na FIT ČVUT.

Testování systému Big Cloud

Projekt Big Cloud je nový český výkonný hybridní cloudový systém. Mezi jednu z jeho funkcionalit patří tvorba a správa cloudového úložiště ownCloud, která bude předmětem tohoto testování.
Moderátor testů bude číst instrukce nahlas, Vaším úkolem je pokusit se instrukce splnit a pokud je to možné, říkejte nahlas, co si myslíte. Tyto testy netestují Vás a Vaší zdatnost, ale správnou implementaci a použitelnost testovaných funkcionalit.

Scénář 1 - Registrace a přihlášení

1. Na obrazovce před sebou vidíte hlavní stranu systému Big Cloud. Na obrazovce se nachází formulář pro registraci. Vyplňte všechna povinná pole validními daty a odešlete registraci.

2. Přihlaste se do systému Big Cloud pomocí přihlašovacích údajů zadaných při registraci.

3. Nyní byste se měl nacházet na hlavní stránce klientské sekce. Ověřte, že na stránce vidíte na levé straně v menu ,,OwnCloud úložiště“ odkaz na ,,Vaše OwnCloud úložiště“.

Scénář 2 - Tvorba ownCloud úložiště

1. Jste přihlášený a nacházíte se v klientské sekci. V menu na levé straně obrazovky klikněte na ,,Vaše OwnCloud úložiště“.

2. Na obrazovce by se měla zobrazit stránka pro správu ownCloud úložišť. Ověřte, že se v seznamu Vašich úložišť zatím žádné nenachází.

3. V menu na levé straně obrazovky klikněte na ,,Nové OwnCloud úložiště“. Zobrazí se modálové okno s formulářem pro tvorbu nového ownCloud úložiště.

4. Vyplňte ve formuláři položky ,,Název“, ,,Doména“, ,,Velikost úložiště“, ,,Heslo“ a ,,Heslo (znovu)“. Při vyplňování pole ,,Velikost úložiště“ dejte pozor, abyste zadal hodnotu menší, než je povolená maximální kvóta. Doménu napište tak, aby měla alespoň 3 znaky a neobsahovala speciální znaky. Heslo zvolte alespoň 5 znaků dlouhé.

Závěr

Největší přínos této práce podle mého názoru spočívá v návrzích implementačních i technických vylepšení pro celý projekt Big Cloud a samozřejmě v naplnění požadavků zadavatele na implementaci nových SaaS a PaaS služeb.

Mně osobně práce přinesla další velkou zkušenost na poli praxe, hlavně z hlediska možnosti být součástí týmu, který se podílí na vývoji relativně velkého projektu.

Co se týče samotného projektu Big Cloud, tak jeho vývoj tímto zdaleka nekončí. Do budoucna by se ve stávajícím návrhu i v kódu daly provádět určitě ještě další optimalizace. Např. pro SaaS službu ownCloud úložiště by se mohl realizovat, v textu zmíněný, přechod ze synchronní na asynchronní komunikaci, protože synchronizované čekání při vytváření úložiště trvá poměrně dlouho. Nepochybně by se také dalo, a nejspíše se i bude, rozšiřovat portfolio poskytovaných služeb pomocí dalších odpovídajících modulů.

Zhodnocení výsledku

Dle mého názoru byly všechny body zadání splněny, snad s výjimkou testování použitelnosti, které má teprve proběhnout, ale na základě již vypracovaných scénářů.

Literatura

[1] Potel, M.: MVP: Model-View-Presenter. Dostupné z: http://www.wildcrest.com/Potel/Portfolio/mvp.pdf

[2] Zeman, M.: Výsledky: Technologie na českém webu. Dostupné z: https://www.zdrojak.cz/clanky/vysledky-technologie-na-ceskem-webu/

[3] DB-Engines Ranking - Trend Popularity. Dostupné z: http://db-engines.com/en/ranking_trend

[4] Alassian: *Comparing Workflows*. [cit. 2016-06-25]. Dostupné z: https://www.atlassian.com/git/tutorials/comparing-workflows

[5] Big Cloud. [cit. 2016-05-05]. Dostupné z: http://www.bigcloud.cz/

[6] S.I.C. Vas IT partner. 2007, [cit. 2016-05-05]. Dostupné z: http://cz.sic.cz/

[7] A safe home for all your data. 2016, [cit. 2016-05-05]. Dostupné z: https://owncloud.org/

[8] Redmine. 2014, [cit. 2016-05-05]. Dostupné z: http://www.redmine.org/

[9] PHP Documentation Group: *PHP Manual*. [cit. 2016-05-05]. Dostupné z: http://php.net/manual/en/

[10] Nette Foundation: *Nette Dokumentace*. 2016, [cit. 2016-05-05]. Dostupné z: https://doc.nette.org/

[11] The PostgreSQL Global Development Group: *PostgreSQL: Documentation*. 2016, [cit. 2016-05-05]. Dostupné z: https://www.postgresql.org/docs/

[12] Git. [cit. 2016-05-05]. Dostupné z: https://git-scm.com/

[13] Gregor, P.: *BigCloud - backend pro veřejný cloudový systém*. Diplomová práce, Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2015.

[14] Potencier, F.: *Learn Symfony*. [cit. 2016-06-23]. Dostupné z: `https://symfony.com/doc/current/index.html`

[15] Otwell, T.: *Installation*. [cit. 2016-06-23]. Dostupné z: `https://laravel.com/docs/`

[16] Zend Technologies Ltd.: *Programmer's Reference Guide of Zend Framework 2*. 2016, [cit. 2016-06-23]. Dostupné z: `http://framework.zend.com/manual/current/en/index.html`

[17] Nils Adermann, J. B.: *Composer*. 2016, [cit. 2016-06-21]. Dostupné z: `https://getcomposer.org/doc/`

[18] Daněk, P.: Velký test PHP frameworků: Zend, Nette, PHP a RoR. Dostupné z: `http://www.root.cz/clanky/velky-test-php-frameworku-zend-nette-php-a-ror/`

[19] Otwell, T.: Lumen. [cit. 2016-06-23]. Dostupné z: `https://lumen.laravel.com/`

[20] Zend Technologies Ltd.: *Getting Started with Zend Framework 2*. 2016, [cit. 2016-06-23]. Dostupné z: `http://framework.zend.com/manual/current/en/user-guide/overview.html`

[21] SQLite Consortium: *SQLite Documentation*. [cit. 2016-06-23]. Dostupné z: `https://www.sqlite.org/docs.html`

[22] Oracle Corporation: *MySQL*. Červen. Dostupné z: `http://dev.mysql.com/doc/refman/en/`

[23] Tezer, O.: SQLite vs MySQL vs PostgreSQL: A Comparison Of Relational Database Management Systems.

[24] Code, test, and deploy together. [cit. 2016-06-26]. Dostupné z: `https://about.gitlab.com/`

[25] *Strider CD*. [cit. 2016-05-05]. Dostupné z: `https://github.com/Strider-CD/strider`

[26] Driessen, V.: *A successful Git branching model*. 1 2010. Dostupné z: `http://nvie.com/posts/a-successful-git-branching-model/`

[27] Software Freedom Conservancy: *Git - githooks Documentation*. [cit. 2016-06-26]. Dostupné z: `https://git-scm.com/docs/githooks`

[28] *Jenkins Documentation*. [cit. 2016-06-26]. Dostupné z: https://jenkins.io/doc/

[29] GitLab: *Configuration of your builds with .gitlab-ci.yml*. [cit. 2016-05-05]. Dostupné z: http://doc.gitlab.com/ce/ci/yaml/README.html

[30] Docker Inc.: *Welcome to the Docker Documentation*. 2016, [cit. 2016-06-26]. Dostupné z: https://docs.docker.com/

[31] redhat: *8.6. CREATING AND MODIFYING SYSTEMD UNIT FILES*. [cit. 2016-05-05]. Dostupné z: https://access.redhat.com/documentation/en-US/Red_Hat_Enterprise_Linux/7/html/System_Administrators_Guide/sect-Managing_Services_with_systemd-Unit_Files.html

[32] Účetní program POHODA. Dostupné z: http://www.pohoda.cz/

[33] *phpDocumentor*. [cit. 2016-05-05]. Dostupné z: https://phpdoc.org/docs/latest/index.html

[34] Mlejnek, I. J.: Návrh – rozhraní a komponenty. Technická zpráva, Katedra softwarového inženýrství, Fakulta informačních technologií, České vysoké učení technické v Praze, 2011, [cit. 2016-05-05]. Dostupné z: https://edux.fit.cvut.cz/courses/BI-SI1/_media/lectures/07/07.prednaska.pdf

[35] Nette Foundation: *Best practise: formuláře jako komponenty*. 2016, [cit. 2016-05-05]. Dostupné z: https://pla.nette.org/cs/best-practise-formulare-jako-komponenty

[36] Dobeš, V.: *Tvorba komponent s využitím autowiringu*. 2014, [cit. 2016-06-27]. Dostupné z: https://pla.nette.org/cs/create-components-with-autowiring

[37] Nette Foundation: *Dependency Injection*. 2016, [cit. 2016-05-05]. Dostupné z: https://doc.nette.org/dependency-injection

[38] ownCloud: *RESTful API - ownCloud Developer Manual*. 2016, [cit. 2016-06-28]. Dostupné z: https://doc.owncloud.org/server/9.0/developer_manual/app/api.html

[39] Redmine: *Rest api - Redmine*. 2014, [cit. 2016-06-28]. Dostupné z: http://www.redmine.org/projects/redmine/wiki/Rest_api

[40] Bodnarchuk, M.; aj.: Codeception. 2016, [cit. 2016-05-05]. Dostupné z: http://codeception.com/

[41] Pádraic Brady, W. G. C., Dave Marshall: *Mockery*. 2014, [cit. 2016-06-29]. Dostupné z: http://docs.mockery.io/en/latest/

[42] Bergmann, S.: *PHPUnit Manual*. 2016, [cit. 2016-06-29]. Dostupné z: https://phpunit.de/manual/current/en/index.html

[43] Selenium Project: *Selenium Documentation*. 2016, [cit. 2016-06-29]. Dostupné z: http://www.seleniumhq.org/docs/

Příloha A

Seznam použitých zkratek

AJAX Asynchronous JavaScript and XML - Asynchronní JavaScript a XML

API Application Programming Interface - Aplikační programovací rozhraní

CD Continuous Deployment - Průběžné nasazování projektu

CI Continuous Integration - Průběžná integrace projektu

DI Dependency Injection - Vkládání závislostí

HTML HyperText Markup Language - Hypertextový značkovací jazyk

HTTP Hypertext Transfer Protocol - Hypertextový přenosový protokol

IaaS Infrastructure as a service - Infrastruktura jako služba

JS JavaScript - Interpretovaný programovací jazyk

PaaS Platform as a service - Platforma jako služba

REST Representational State Transfer - Reprezentativní stavový přenos

RPC Remote Procedure Call - Vzdálené volání procedur

SaaS Software as a service - Software jako služba

SQL Structured Query Language - Strukturovaný dotazovací jazyk

SSH Secure Shell - Zabezpečený komunikační protokol

UML Unified Modeling Language - Sjednocený modelovací jazyk

URL Uniform Resource Locator - Jednotná adresa zdroje

XML Extensible Markup Language - Rozšiřitelný značkovací jazyk

Příloha **B**

Obsah přiloženého CD

- `readme.txt` ..... stručný popis obsahu CD
- `src`
  - `impl` ..... zdrojové kódy implementace
  - `tests` ..... zdrojové kódy testů
  - `thesis` ..... zdrojová forma práce ve formátu LaTeX
    - `diagrams` ..... diagramy použité v textu práce
    - `examples` ..... příklady použité v textu práce
    - `pictures` ..... obrázky použité v textu práce
    - `wireframes` ..... wireframy použité v textu práce
  - `user-manual` ... zdrojová forma uživatelské příručky ve formátu LaTeX
  - `instal-manual` .. zdrojová forma instalační příručky ve formátu LaTeX
  - `prog-manual` .... programátorská příručka ve formátu webové stránky
- `text`
  - `thesis.pdf` ..... text práce ve formátu PDF
  - `attach` ..... přílohy
    - `user-manual.pdf` ..... uživatelská příručka ve formátu PDF
    - `instal-manual.pdf` ..... instalační příručka ve formátu PDF
- `test-scenarios` ..... testovací scénáře a dotazníky