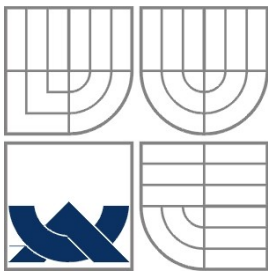


VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA STROJNÍHO INŽENÝRSTVÍ
ÚSTAV AUTOMATIZACE A INFORMATIKY

FACULTY OF MECHANICAL ENGINEERING
INSTITUTE OF AUTOMATION AND COMPUTER SCIENCE

VERZOVANÍ SOUBORŮ A PROJEKTŮ
VERSIONING OF FILES AND PROJECTS

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE **PETR JINDRA**
AUTHOR

VEDOUCÍ PRÁCE **ING. JAN ROUPEC, PH.D.**
SUPERVISOR

BRNO 2011

ZADÁNÍ ZÁVĚREČNÉ PRÁCE

(na místo tohoto listu vložte originál a nebo kopii zadání Vaš práce)

ABSTRAKT

Tato bakalářská práce se zabývá problematikou verzování souborů a projektů. V první části popisuje historické i současné systémy sloužící k tomuto účelu a porovnává je mezi sebou. Ve druhé je navržen nový systém, který by měl být snadno instalovatelný na kterýkoliv standardní webový hosting.

ABSTRACT

This bacheleor thesis is about a revision control of files and projects. First part describes historical and current systems designed for the above mentioned purpose and compares them. In the second one a new system is developed, which should be easy to install to whichever web hosting.

KLÍČOVÁ SLOVA

Verzování, webová aplikace, PHP

KEYWORDS

Revision control, web application, PHP

PODĚKOVÁNÍ

Rád bych poděkoval Ing. Janu Roupcovi Ph. D., který mi poskytl cenné rady během tvorby této práce a také své přítelkyni Hance, která provedla korekturu.

OBSAH:

1 ÚVOD

Vzhledem k aktuálnímu směru vývoje aplikací schopných pracovat na více zařízeních (počítače, mobilní telefony, tablety) jsou stále populárnější webové aplikace umožňující běžnou práci (psaní dokumentů, organizace času pomocí kalendáře, ale i úprava obrázků nebo stříhání videa).

Pro menší vývojové týmy zde ovšem existuje problém spolupráce a předávání aktuálních verzí jejich projektu mezi sebou. Pro tyto účely sice existují volně dostupné služby s již nainstalovaným verzovacím systémem, kde po registraci získá uživatel vyhrazené místo na serveru, které může využít pro správu svého projektu. Tyto služby již ale zpravidla neřeší aktualizaci cílového serveru, na kterém má výsledná aplikace běžet.

Aktualizace serveru se tedy často provádí tak, že si jeden vývojář stáhne verzi projektu, která byla označena jako funkční, a cílový server musí ručně zaktualizovat (zpravidla přes FTP). Tato metoda je zdlouhavá a může vést k chybám. Člověk například zapomene přenést na server některé soubory nebo naopak přenese soubory, které měly sloužit jako testovací data či jako konfigurační soubory pro připojení k testovacímu serveru. Alternativou k této metodě jsou jednoduché skripty, které umožňují zaktualizovat server automaticky z lokálního adresáře. Ani při této metodě ale neodpadá nutnost stáhnout aktuální verzi celého projektu na lokální počítač uživatele.

Tento problém by měla vyřešit tato práce, která má za cíl vytvořit verzovací systém, schopný provozu na běžném webovém hostingu bez nutnosti instalace specializovaného softwaru nebo nástrojů na server.

V dalším textu jsou používány odborné výrazy a pojmy, které už přešly do terminologie problematiky verzování jako všeobecně zavedené nebo výrazy a pojmy, které není možné vhodně nebo vůbec přeložit do češtiny. V příloze je uveden seznam pojmů se stručným vysvětlením, z nichž některé vybrané pojmy jsou vysvětleny podrobněji níže.

1.1 Důvody verzování souborů

Při vývoji větších projektů (stovky a více souborů) skupinou vývojářů nebo jednotlivcem dochází v případě uložení souborů projektu v klasickém adresáři na serveru k riziku nechtěného přepisu nových souborů staršími, vzájemnému přepisování dat jednotlivými vývojáři, i k neopatrnému smazání souboru. Hledat takovéto chyby je časově velmi náročné a hlavně značně snižuje efektivitu práce.

Řešením tohoto problému je použití některého verzovacího systému, který kromě kontroly „přepisu nových dat“ většinou zajišťuje automatické zálohování a řízení přístupu k položkám repozitáře (projektu). Veškeré operace jsou většinou logovány, tudíž zaměstnavatel může i sledovat činnost jednotlivých členů týmu.

Tyto systémy nemusí být použity pouze pro správu zdrojového kódu. Najdou použití i v širokém spektru dalších oborů jako je správa výkresů (z praxe například propojení CADu Unigraphic a SAPu), v elektrotechnice při návrhu nových zařízení, ale i mimo technické obory jako např. ve školství a podobně.

2 POJMY

Pro čtenáře neznajícího problematiku verzování jsou na následujících řádcích rozebrány a vysvětleny některé podstatné pojmy, které jsou důležité pro pochopení jak problematiky verzování, tak i principu fungování programů, které k tomuto účelů slouží.

2.1 Branch, fork

Branch (fork) je „rozštěpení“ vývojové cesty souboru na dvě, nebo více větví, přičemž u nové větve stále existuje vazba na původní vývojovou linii.

Větvení vývoje programu může mít několik příčin. Zpravidla to bývají specifické požadavky klienta na konkrétní úpravy vyvíjeného projektu, vývoj více edic aplikace lišící se omezením funkcí, a nezřídka kdy (zejména ve světě open source) i příznak odštěpení části komunity vývojářů jako v roce 2010, kdy se od projektu OpenOffice.org oddělil fork LibreOffice.

2.2 Merge

Operací Merge se chápe sloučení větví, které probíhá například při současné práci více lidí na jednom souboru. Je provedeno tak, že se vyhodnotí rozdíly ve slučovaných souborech a poté se vygeneruje výsledný soubor, který je sestaven ze dvou, nebo více vstupních.

```
petr@linux-3oy1:~/nuke/temp$ diff prvni.php druhy.php
2,3d1
< $foo = "bar";
<
9c7,10
< $result = base64_encode($argv[1]);
---
> if (!empty($argv[1]))
>       $result = base64_encode($argv[1]);
> else
>       $result = "Zadal jste prázdný text"
petr@linux-3oy1:~/nuke/temp$
```

Obrázek 1: Ukázka porovnání dvou souborů (použit program Diff)

Jako změna se bere například odebrání nebo přidání řádku textu a modifikace řádku textu. Zpravidla je ale ignorována změna bílých znaků (logické odřádkování, odsazení od začátku řádku), neboť tím se smysl textu (kódu) nemění (výjimkou jsou ovšem programovací jazyky jako je Python, kde se odsazením uzavírají bloky kódu). Pokud je změn více u sebe, je značen jako změněný celý blok kódu (Obrázek 1).

Někdy nelze strojově rozdíly vyřešit (změny byly provedeny v obou souborech ve stejném bloku kódu) a je nutný zásah uživatele. Tento scénář probíhá zejména při slučování dvou větví, které byly dlouho odloučené.

2.3 Repozitář

Repozitář je logické uskupení dokumentů (souborů), případně i adresářů, které jsou součástí vyššího celku (například zdrojové kódy programu, výkresy a modely stroje...).

Repozitáře můžeme rozdělit na lokální (přístup má pouze jeden uživatel) a vzdálené (přístup mají všichni členové týmu). Podrobnější popis rozdílů mezi lokálním a vzdáleným repozitářem je uveden níže, v kapitole „Rozdělení systému podle modelu ukládání dat“.

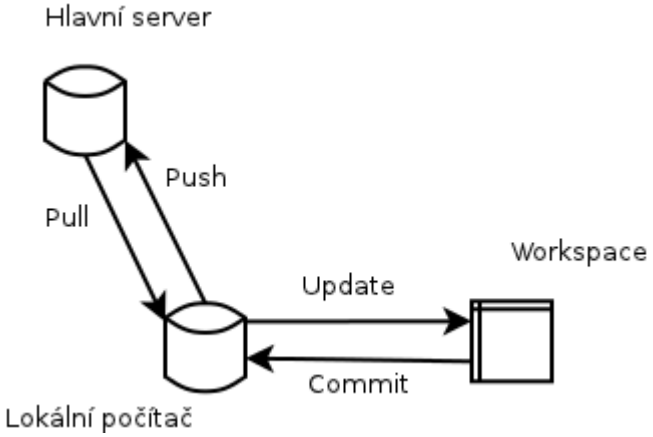
3 ROZDĚLENÍ SYSTÉMŮ PODLE MODELU UKLÁDÁNÍ DAT

3.1 Lokální

Nejjednodušší model je lokální, kdy repozitář existuje pouze na jednom stroji, kde je vytvořen i workplace. Používá se pouze okrajově, zpravidla na „one man“ projekty, kdy není potřeba spolupráce s jinými vývojáři.

3.2 Model Klient-Server

V případě modelu Klient-Server existuje jeden centrální server, na kterém jsou všechny soubory a historie jejich změn, a vývojáři mají zpravidla staženou pouze aktuální verzi souboru (projektu).



Obrázek 2: Model Klient-Server

Značnou nevýhodou tohoto přístupu je ztráta všech dat v případě havárie, pokud se server pravidelně nezálohuje.

3.3 Distribuované

Distribuovaný model je v nynější době je nejběžnější model systému. V těchto systémech nemusí existovat centrální server a každý repozitář každého uživatele působí zároveň jako záloha všech dat. Z toho důvodu nelze použít model „Lock – Edit – Unlock“ (nevíme, kdo dělá na kterém souboru), ale je nutné použít model, kdy je nad editovanými soubory provedeno sjednocení (merge).

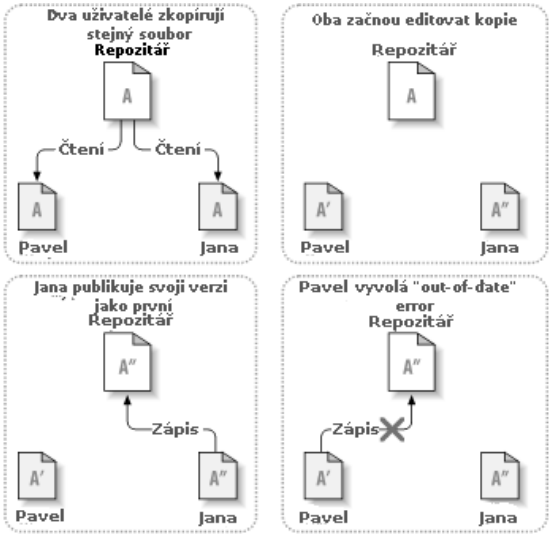
Výhodou tohoto modelu je možnost práce více vývojářů na jednom souboru zároveň. Mohou pracovat nezávisle na připojení k internetu, kdy je zálohování provedeno s každou synchronizací s uživatelem, který momentálně působí jako server.

Nevýhodou tohoto přístupu je posloupnost verzí, kde stejné číslo verze nemusí ještě znamenat stejný obsah souboru.

4 ROZDĚLENÍ SYSTÉMU PODLE MODELU PŘÍSTUPU K DATŮM

4.1 Model Zkopíruj-Modifikuj-Sluč

Nejčastěji používaný model pro verzování textově orientovaných dokumentů spočívá na filozofii slučování změn provedených ve stejnou dobu na stejném souboru se nazývá Zkopíruj-Modifikuj-Sluč.

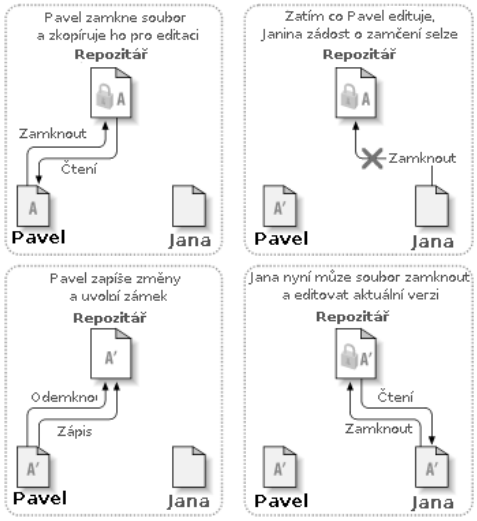


Obrázek 3: Práce v modelu Zkopíruj-Modifikuj-Sluč [1]

Obrázek odkazuje scénář, kdy si dva vývojáři (Pavel a Jana) stáhnou jeden soubor a začnou v něm provádět změny. První z nich bez problémů uloží změny, ale při ukládání kopie od Pavla repozitář ohlásí chybu, že soubor byl od svého stažení změněn. V tomto případě je nutné sáhnout soubor se změnami od Jany a provést merge (ne vždy se tento proces obejde bez zásahu uživatele), poté je možné soubor konečně odeslat na server.

4.2 Model Zamkni-Uprav-Odemkni

Model Zamkni-Uprav-Odemkni se zpravidla používá pro správu projektů s binárními soubory jako např. projekt z CAD systému apod. (příklad z praxe je systém SAP + CAD UniGraphic).



Obrázek 4: Práce v module Zamkni-Uprav-Odemkni [1]

Uživatel, který chce soubor editovat, pošle na server požadavek na zamčení souboru. V případě, že mu systém vyhoví, mu je dovoleno stáhnout požadovaný dokument pro editaci. Po uložení změn a odeslání souboru zpět na server dojde k jeho opětovnému odemčení. Po celou dobu, kdy je soubor zamčen, nesmí nikdo soubor editovat (systém to nepovolí).

Z toho vyplývají rizika, se kterými je tento model spojen. Nejčastější případ chyby je pravděpodobně neodemknutí souboru po editaci a zapomenutí odeslat zeditovaný soubor zpět na server. Tento problém se dá řešit buď časovými zámky (soubor lze zamknout pouze na omezenou dobu), nebo zásahem administrátora, který soubor opět odemkne.

5 PRVNÍ VERZOVACÍ SYSTÉMY

5.1 SCCS – Source Code Controll systém

SCCS Tento systém byl vyvinut v roce 1972 v Bellových laboratořích pro korporaci IBM. V té době používal velmi zvláštní architekturu pro ukládání programových jednotek (souborů) – veškeré verze a revize byly uloženy v jednom souboru pomocí tzv. delta balíčků, které byly řetězeny tak, jak byly postupně ukládány. Ukládaly se pouze změny oproti předchozí verzi. Díky této architektuře není prakticky možné provést neautorizovanou změnu uloženého kódu [2].

Zajímavostí je, že původní soubor je uložen jako delta balíček vzhledem k prázdnému souboru. Po každém uložení na server (commit) je zároveň zdokumentováno, kdo a kdy změnu provedl, a zároveň je systém schopen provést porovnání jednotlivých revizí a zjistit, ve kterých místech ke změně došlo.

Při požadavku ke stažení aktuálních dat (Pull) je proveden proces poskládání jednotlivých delt a výsledné soubory jsou odeslány uživateli.

V současné době je tento software ve verzi 1.0, která je dostupná pro Windows a operační systémy odvozené od Unixu. V době psaní této práce byla poslední změna stabilní verze zveřejněna na začátku roku 2007.

5.2 RCS – Revision Controll System

RCS je dnes již značně zastaralý software, který umí pracovat s textovými dokumenty a omezeně i s binárními soubory [3].

Byl vyvinut vyvinut Walterem F. Tichým na Univerzitě Purdue v USA na počátku osmdesátých let minulého století [4].

Je to poměrně jednoduchý systém, který dokáže pracovat pouze s jednotlivými soubory (s projekty pracovat neumí), které ukládá podobně jako SCCS pomocí delt. Na rozdíl od něj je ale ukládá do samostatných souborů.

Umí vytvářet a slučovat vývojové větve souboru, což je velice obtížná práce, a proto se většinou pracuje pouze s hlavní vývojovou linií souboru (takzvaná mainline). Tyto problémy později vyřešil systém CVS, který je popsán níže.

Poslední stabilní verze je z roku 1995 a od té doby je projekt z hlediska vývoje prakticky mrtvý. Jediná změna byla provedena na konci roku 2010, kdy byl projekt převeden na licenci GPL v3.

5.3 PVCS – Polytron Version Control system

Polytron Version Control System byl vyvinut firmou Polytron v roce 1985. Polytron poté prošel sérií různých vlastníků a dnes je PVCS vyvíjeno firmou Serena Software Inc.

Je to komerční software, který je dnes patrně jedním z nejkomplexnějších řešení verzování projektů. Balík obsahuje sadu několika mocných nástrojů, které usnadňují práci v systému a kontrolu nad soubory a projekty.

Systém je možné ovládat jak pomocí příkazové řádky, tak i pomocí grafického rozhraní, které je dodáváno [5].

V momentální době je PVCS ve verzi 8.4.1, která byla vydána 29. listopadu 2010.

6 NEJBĚŽNĚJŠÍ VERZOVACÍ SYSTÉMY

6.1 Git

Systém Git začal být vyvíjen Linusem Torvaldsem v roce 2005 (projekt začal ve stejné době jako Mercurical [6]) pro řízení vývoje jádra operačního systému Linux. Původně byl zamýšlen pouze jako platforma, na které se měly stavět vlastní verzovací systémy. V průběhu používání se ale Git vyvinul
v plnohodnotný RCS systém, používaný s oblibou mnoha vývojáři.

Díky tomu, že se jedná o open source program, ho je možné volně využívat jak soukromě pro osobní potřebu, tak i komerčně ve velkých firmách, a modifikovat ho podle svých potřeb (samozřejmě v souladu s licencí GPL).

6.1.1 Princip ukládání dat

Na rozdíl od většiny ostatních systémů ukládá Git při každém commitu všechna data souboru znovu. To má za následek větší odolnost proti poškození repozitáře, např. neopatrným smazáním starých dat uživatelem nebo chybou hardwaru. Značnou nevýhodou tohoto přístupu je ovšem větší diskový prostor, který repozitář zabere. Z toho důvodu jsou staré commity komprimovány a archivovány zvlášť.

Další vlastností při ukládání dat je distribuovanost systému. To znamená, že každý vývojář má na svém stroji vlastní repozitář (kopii aktuálního repozitáře na hlavním serveru), se kterým pracuje, a po ukončení práce všechny změněné soubory odešle na hlavní server. Při přenosu jsou soubory zkomprimovány a odeslány atomicky najednou, aby nedocházelo ke zbytečnému zatěžování sítě.

6.1.2 Běžné používání RCS Git

Pro založení repozitáře jsou k dispozici dva postupy - založení prázdného repozitáře nebo stažení už existujícího repozitáře ze serveru. Pro ukázky je v následujícím textu použit klientský program git, dostupný zpravidla ve všech linuxových distribucích. Platformou Windows se zabývat nebudu.

Založení prázdného repozitáře se provede příkazem

```
git init
```

spuštěným v adresáři s projektem. Příkaz vytvoří skrytý adresář .git, který obsahuje vlastní data repozitáře.

Pokud je potřeba provázat vzniklý repozitář s již existujícím na serveru, provede se to příkazem

```
git remote add origin <adresa repozitá e>
```

Pro tento krok je nutné mít nainstalovaný SSH klíč, neboť většina serverů podporuje pro zápis pouze šifrované spojení, aby byla jasná identifikace uživatele (číst lze z repozitáře bez klíče).

Instalaci repozitáře existujícího na vzdáleném serveru je možné provést příkazem

```
git clone <adresa repozotá e>
```

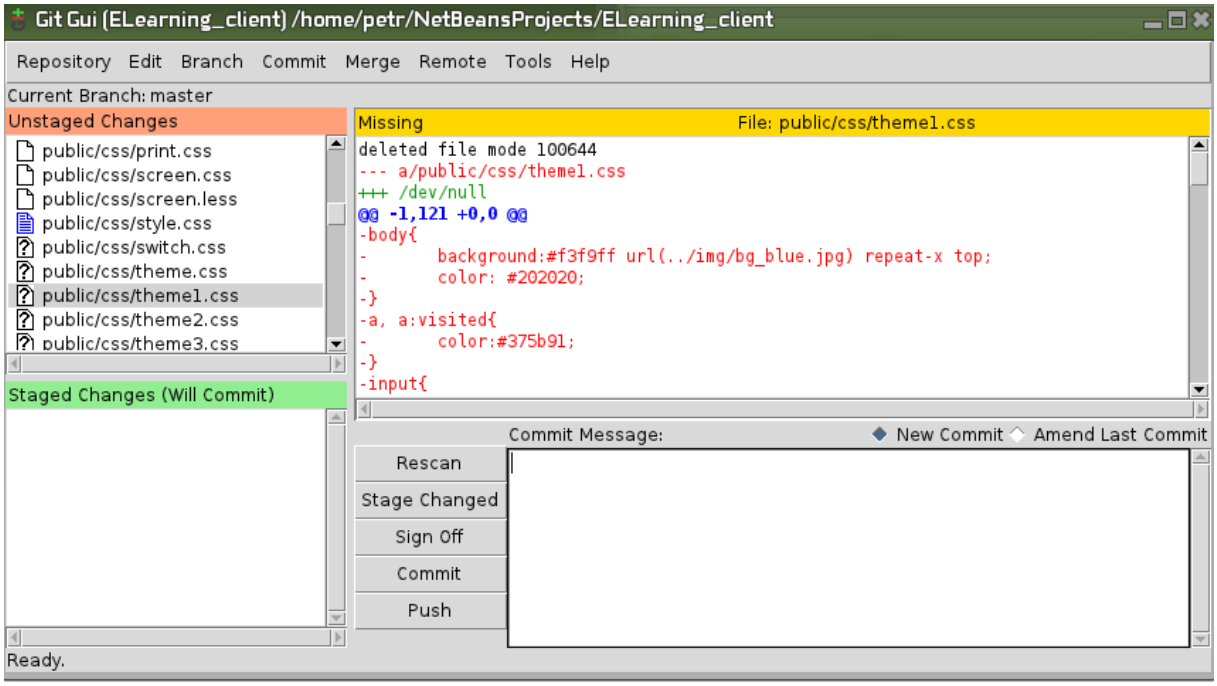
spuštěném v cílovém adresáři. Tento příkaz zinicializuje repozitář, stáhne aktuální data, nakopíruje je do pracovního adresáře a nastaví informace potřebné pro další spojení serverem.

Po provedení změn v adresáři je nutné nové soubory přidat do repozitáře a staré odstranit. Slouží k tomu příkazy

```
git add <název souboru>
git remove <název souboru>
```

Je možné použít také integrované grafické rozhraní (obrázek 5) spustitelné příkazem

```
git gui
```



Obrázek 5: Integrované grafické rozhraní Git

Tady je možné provádět všechny úkony v poměrně přehledné aplikaci bez nutnosti znát konzolové příkazy. Okno aplikace je rozděleno na čtyři části. V levé horní jsou změny, které ještě nebyly odeslány a nejsou označeny ke commitu. Pod ní je seznam změněných souborů označených k následujícímu commitu. V pravé horní části je náhled změn, ke kterým došlo od posledního commitu (v případě binárních souborů je zobrazena pouze informace že změny nebylo možné zjistit). A ve spodní pravé části je okno pro ovládání vlastního repozitáře (vstupní pole pro komentář commitu, tlačítka pro commit, push, atd.). V dalším textu bude věnována pozornost zpravidla pouze konzolovým příkazům.

Po provedení změn v pracovním adresáři se pomocí příkazu

```
git commit -m '<komentá   zm n>'
```

tyto změny uloží do lokálního repozitáře a následným příkazem

```
git push origin master
```

se změny odešlou na server.

Předpokládejme, že v repozitáři došlo od provedení našeho posledního odeslání dat na hlavní server (push) ke změnám, proto je nutné provést synchronizaci lokální kopie repozitáře. Toho je možné docílit příkazem

```
git pull origin master
```

Ten provede synchronizaci lokální kopie s hlavním repozitářem a zároveň zaktualizuje pracovní adresář. Pro synchronizaci pracovního adresáře pouze s lokálním repozitářem se spustí příkaz

```
git update
```

V případě, že jsme od posledního odeslání na hlavní server změnili některý soubor, vyvolá žádost o synchronizaci s hlavním repozitářem konflikt. Poté je potřeba pro daný konflikt provést operaci merge (ručně nebo použít nástroj k tomu určený).

6.2 CVS

CVS původně vzniklo jako nadstavbová kolekce skriptů pro RCS. Záměrem bylo zjednodušit práci s tímto systémem. V průběhu let se ale projekt vyvinul v plnohodnotný systém, který je dnes ale už relativně zastaralý a byl nahrazen níže zmíněným SubVersion.

Pracuje nad modelem Client-Server s možností tzv. „unreserved checkout“ [7], tedy že více vývojářů může pracovat na jedné entitě zároveň (soubor se nezamyká a vytvoří se samostatná vývojová větev pro každého vývojáře) a po odeslání na server se provede operace merge.

6.2.1 Princip ukládání dat

Metoda ukládání dat je z historických důvodů převzata z RCS a rozšířena o modulárnost datového modelu, aby u velkých projektů byla zlepšena přehlednost.

Dále systém podporuje import vývojových větví z jiných repozitářů (více týmů pracuje na stejném projektu v různých repozitářích), což je realizováno pomocí operace merge nad původní a importovanou vývojovou větví (branche).

6.3 SubVersion

První verze SubVersion byla uvolněna v roce 2000. Záměrem bylo nahradit v tu dobu nejpoužívanější,
ale už značně zastaralé CVS.

Subversion používá architekturu Klient-Server a model Zkopíruj-Modifikuj-Sluč s možností vynucení zamčení souboru, pokud je to nutné [1].

6.3.1 Princip ukládání dat

Data jsou ukládána pomocí delt. V případě potřeby, aby byl soubor na dvou místech v repozitáři, je na druhém místě vytvořen pouze odkaz na původní entitu. Potom nedojde ke zbytečné duplikaci dat.

7 VOLBA PLATFORMY

Protože systém je webová aplikace, byla jako její základ zvolena LAMP platforma, která je instalovaná na většině českých hostingových služeb. Zkratka LAMP znamená operační systém Linux, webový server Apache, databázový stroj MySQL a skriptovací jazyk PHP.

7.1 Operační systém Linux

V prostředí stolních počítačů je Linux málo rozšířený operační systém [8], ale na serverových strojích má převahu [9]. Je k dispozici v mnoha distribucích, které jsou zaměřeny na specifické účely (stolní počítače, webové servery, pracovní stanice,...). Tyto distribuce mohou být rozděleny ještě na specializované edice.

7.2 Webový server Apache

Serverový program Apache je jeden z nejrozšířenějších webových serverů, který je standardně dodáván s většinou linuxových distribucí. Je snadno rozšiřitelný pomocí zásuvných modulů, podporuje virtuální servery a je snadno konfigurovatelný pomocí konfiguračních souborů.

7.3 Databázový stroj MySQL

MySQL je databázový stroj, původně vyvíjen Sun Microsystems a nyní Oracle Corporation. MySQL umožňuje vytvářet klasické relační tabulky (formát MyISAM) i tabulky s cizími klíči (formát InnoDB), které samy dokáží kontrolovat integritu zapsaných dat.

7.4 Skriptovací jazyk PHP

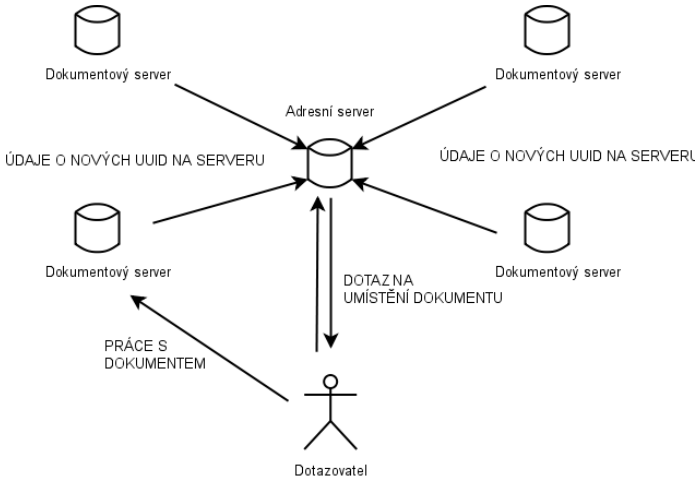
Jazyk PHP byl původně vyvinut pro jednoduchou tvorbu dynamických webů a postupně se vypracoval na plnohodnotný programovací jazyk. Kromě webových aplikací ho je možné rozšířit o modul GTK, díky kterému je možné vytvářet plnohodnotné desktopové aplikace s grafickým prostředím.

Jazyk se nyní nachází ve verzi 5.3, která podporuje anonymní funkce, jmenné prostory, viditelnost vlastností uvnitř objektů apod.

8 NÁVRH SYSTÉMU

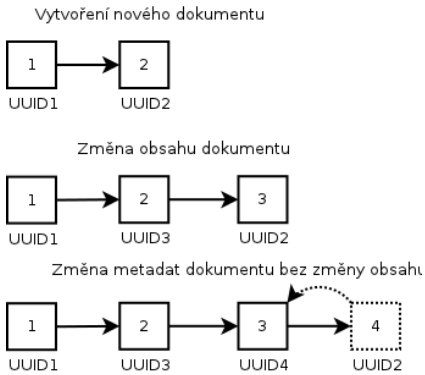
8.1 Ukládání dat

Vzhledem k pravděpodobné potřebě rozložení úložiště na více serverů (škálovatelnost systému) nepoužívá program standardní celočíselný inkrementovaný klíč, ale jednoznačné identifikátory dokumentu označované jako UUID (univerzální unikátní identifikátor). Ty jsou založené na kombinaci UUID v.4 a UUID v.5 dle RFC 4122. Tato strategie umožňuje v případě škálování vytvořit adresní server obsahující mapu rozložení dokumentů na serverech, viz obrázek 6.



Obrázek 6: Možné škálování dokumentového serveru

Do databáze jsou ukládány pouze metadata o dokumentu, tedy jméno, přístupová práva a vlastník. Vlastní obsah souboru je ukládán do adresáře na disku. Při vytvoření nového dokumentu jsou uloženy do databáze dva záznamy. První reprezentuje původní stav dokumentu a druhý reprezentuje nejauktuálnější stav. Při dalších commitech jsou staré verze vkládány mezi tyto dva dokumenty. Díky tomu má původní a aktuální verze dokumentu stále stejné UUID a lze na něj bez problémů odkazovat.



Obrázek 7: Schéma generování UUID dokumentu

V případě, že je commitován dokument, který se obsahově nezměnil (bylo mu změněno například jméno, práva), vytvoří se místo nového obsahu pouze symbolický odkaz na původní verzi dokumentu (viz obrázek 7). Toto řešení má smysl zejména pro velké soubory, např. pokud spravujeme projekt z CAD systému.

Přístupová práva k dokumentu se berou vždy pouze z aktuální verze. Informace o přístupových právech ve starých verzích dokumentu jsou uchovávány pouze z důvodů uchování změn.

Adresářová struktura systému je ryze virtuální a je uložena v databázové tabulce pomocí stromové reprezentace, kdy u každého adresáře je informace o jeho rodiči. Informace o potomcích se u adresáře neukládá, neboť je možné ji jednoduše vyfiltrovat pomocí SQL dotazu

```
select * from documents_directories where parent_id = [parentId]
```

Podrobný popis databázové struktury je uveden níže.

8.2 Přístup k datům

K datům uloženým v systému je možné přistupovat buď pomocí integrovaného uživatelského rozhraní, nebo pomocí implementace REST metod.

Uživatelské rozhraní v systému je určené zpravidla pouze pro administrátory. Je navrženo pro obsažení všech možností systému na úkor přívětivosti k uživateli, protože je implementováno pouze z důvodů havarijních stavů nebo pro účely údržby.

Vzdálený přístup pomocí REST metod je řešen pomocí URL dotazů, protože většina dnešních serverů implementuje z HTTP standardu pouze přístupové metody POST a GET (metody PUT a DELETE zpravidla implementovány nejsou). Informaci o úspěchu, nebo neúspěchu požadované operace je vrácena stavovým kódem v HTTP hlavičce a případným popisem chyby v těle odpovědi.

Struktura dotazu je následující:

```
/<typ objektu>/<metoda>/[<identifikátor objektu>][_jméno objektu][<formát>]
```

Významy jednotlivých částí adresy jsou následující:

1. TYP OBJEKTU – sděluje informaci o typu objektu, se kterým chceme pracovat, např. jestli chceme pracovat s dokumentem, adresářem apod.
2. METODA – může obsahovat hodnoty PUT, GET, POST nebo DELETE a případně další speciální metody definované typem objektu. Významy jednotlivých standardních metod jsou vysvětleny níže.
3. IDENTIFIKÁTOR OBJEKTU – obsahuje identifikační číslo nebo UUID nebo jiný identifikátor, podle kterého lze jednoznačně vybrat objekt, se nímž chce uživatel pracovat.
4. JMÉNO OBJEKTU – jedná se o dobrovolnou část, která nemá na chod programu, ani na vyhodnocení požadavku žádný vliv, a slouží pouze k informování uživatele o jménu objektu.
5. FORMÁT – blíže specifikuje požadovaný formát dat. Standardně systém podporuje HTML a JSON. V případě, že formát není uveden, vrací systém zpravidla informace kódované právě v JSON. Pouze u požadavku GET na dokument bez specifikace formátu vrací systém obsah dokumentu.

Standardizované metody REST jsou následující:

- DELETE – požadavek na smazání objektu (vrací se indikace úspěchu).
- GET – požadavek na informace o objektu (vrací se požadovaná data).
- POST – žádost o vytvoření nového objektu (vrací se indikace úspěchu a identifikátor nového objektu).
- PUT – žádost o změnu existujícího objektu (vrací se indikace úspěchu, popř. aktualizovaná data).

Pokud metoda v nějakém modulu vrací nějaká nestandardní data nebo je ovlivněna parametry, bude u daného modulu popsána podrobněji.

9 IMPLEMENTACE PROGRAMU

Implementace systému je možná několika způsoby. Jako nejpoužitelnější se jeví dva, kdy obě možnosti mají část adresářové struktury skrytou pomocí zanořeného kořenového adresáře dokumentů (DocumentRoot) v nastavení (virtuálního) serveru:

1. Každá akce je v samostatném souboru a při požadavku je vybrán vhodný soubor, který se zpracuje.
2. Je použit model MVC (Model-View-Controller) společně s vhodným aplikačním jádrem (Zend, Nette), kdy jednotlivé objekty jsou rozděleny do modulů, jejichž kontrolery jsou přetěžovány.

Program byl implementován nejprve první metodou. Ta má značnou výhodu v tom, že není závislá na povolení RewriteEngine (modul serveru, který umožňuje rozložit URL adresu a přesměrovat ji jinam) na serveru, protože soubory s kódem vykonávajícím práci s objekty jsou přímo ve veřejném adresáři. Se znalostí struktury dotazů a jmen souborů, které je zpracovávají, je tedy možné přistupovat k datům přímo přes tyto soubory.

Po dokončení základní části programu ale bylo zjištěno, že tato architektura je nevyhovující z dlouhodobého hlediska. Protože budou tvořeny další části programu (moduly), které by v tomto modelu byly uloženy ve veřejném adresáři, vznikal v tomto adresáři chaos v souborech.

Proto bylo jádro převedeno na architekturu Zend Aplikace. Aby bylo možné přistupovat k datům tříúrovňově (typ objektu, metoda, identifikátor), byly pro jednotlivé objekty vytvořeny samostatné moduly a pro každou metodu přístupu samostatný kontroler. Třetí úroveň navigace je realizována pomocí „magické“ metody __call, která je interpretem PHP provedena vždy, pokud je volána neexistující metoda objektu.

9.1 Vzdálený přístup k datům

Ke vzdálenému přístupu (např. pomocí klientského programu nebo z jiného serveru) slouží návratové hodnoty ve formátu JSON. Na rozdíl od formátu HTML (integrované uživatelské rozhraní) obsahuje formát JSON pouze data, která jsou bez vhodného klientského programu prakticky nečitelná.

Vrácená datová struktura vždy obsahuje standardizovanou část s obecnými informacemi o zpracování a poté datovou část, obsahující data požadovaného objektu. Obsah standardizované části je:

Hodnota	***Typ***	***Popis***
success	logická hodnota	Indikace úspěchu (TRUE) nebo neúspěchu (FALSE) operace
status	řetězec	Vyjádření konečného stavu operace (například chybové hlášení)
statusCode	celé číslo	Kód konečného stavu operace dle HTTP
data	pole	Asociatvní pole obsahující data vráceného objektu

Tabulka 1: Formát standardní části odpovědi

9.2 Implementace modulů

Základem programu jsou celkem čtyři moduly, které se starají o jednotlivé datové objekty. Přístup k datům skrz tyto moduly je čistě atomický, výjimečně jsou vráceny některé podružné objekty (výpis adresáře nebo dokumentu), a obsluhuje skutečně nejnižší úroveň systému.

Zbytek funkcí je realizován pomocí dalších modulů, které umožňují tvorbu projektů, komunikaci mezi uživateli nebo psaní poznámek k dokumentům. Tyto moduly nejsou nijak potřeba pro chod základního systému a jejich užití není nutné.

Moduly, které jsou vytvořeny v rámci této práce jsou tyto:

- Documentation – správa dokumentace.
- FTP – umožňuje celý projekt nahrát na cílový server.
- Message – komunikace mezi uživateli.
- Note – tvorba poznámek k dokumentům.

- Plan – jednoduché plánování postupu na projektu.
- Project – správa projektů a projektové dokumentace.

9.2.1 Modul Document

Modul Document zajišťuje manipulace s obsahy dokumentů v systému. Má na starosti generování unikátních UUID, správu řetězení fyzických dokumentů (obsahů) a vytváření a slučování vývojových větví. Jako identifikátor objektu je mu předáváno UUID dokumentu, se kterým chce uživatel pracovat. Při metodě POST navíc umožňuje přiřadit nový dokument do vybraného adresáře. Obsahuje zvláštní metodu TRASH, která vrací výpis dokumentů právě přihlášeného uživatele, které nejsou přiřazeny v žádném z adresářů.

Metoda GET

Metoda GET vrací dokument, který je specifikovaný identifikátorem ve třetí části URL dotazu. Pokud není specifikován formát, vrací obsah dokumentu, v opačném případě vrací pouze informace o dokumentu.

Datová struktura odpovědi na dotaz se specifikovaným formátem je následující

Hodnota	***Typ***	***Popis***
uuid	řetězec	UUID dokumentu
document_name	řetězec	jméno dokumentu
mime_type	řetězec	MIME typ
user_id	celé číslo	identifikační číslo majitele dokumentu
group_id	celé číslo	identifikační číslo skupiny, které dokument patří
mask	řetězec	maska oprávnění jako devíti znakový řetězec
created_at	časová značka	časová značka vytvoření dokumentu
is_latest	logická hodnota	indikace, jestli se jedná o aktuální obsah
size	celé číslo	velikost obsahu dokumentu v bytech

Tabulka 2: Datová struktura objektu informací o dokumentu

Metoda POST a PUT

Metody pro vytvoření (POST) a aktualizaci (PUT) vrací pouze standardní odpověď, ale pro jejich správný chod je potřeba vstupní objekt, který je předáván formou asociativního pole, zpravidla uvnitř HTTP požadavku nebo případně adrese požadavku.

Jméno objektu, které pole vytváří je `document` a obsahuje následující vlastnosti:

Hodnota	***Typ***	***Popis***
document_name	řetězec	jméno dokumentu
mime_type	řetězec	MIME typ
user_id*	celé číslo	identifikační číslo majitele dokumentu
group_id	celé číslo	identifikační číslo skupiny, které dokument patří
mask	řetězec	maska oprávnění jako devíti znakový řetězec
content**	binární data	nový obsah dokumentu
document_directory_id***	celé číslo	identifikační číslo adresáře, kam má být nový dokument přiřazen

* pouze uživatel root nebo DOCUMENT_MANAGER

** může být odeslán jako přiložený soubor

*** pouze pro metodu POST

Tabulka 3: Vlastnosti objektu pro metody POST a PUT

Pro metodu POST jsou povinné parametry `document_name` a `content`. Ostatní, pokud nejsou odeslány, jsou nastaveny automaticky.

Pro metodu PUT jsou všechny hodnoty dobrovolné a jsou zaktualizovány pouze ty hodnoty, které byly odeslány. Pokud během aktualizace dojde k chybě (např. nedostatečné oprávnění), celá aktualizace skončí s chybovým stavem a data nejsou uložena.

Metoda TRASH

Nestandardní metoda TRASH (vzhledem k REST) slouží ke správě dokumentů, které nejsou asociovány do žádného adresáře. Jako třetí část dotazu v URL není uveden identifikátor objektu, ale akce, kterou chce uživatel provést. Může to být:

- GET – vrací seznam dokumentů, které nejsou v žádném adresáři.
- REFRESH – obnoví seznam dokumentů, které nejsou asociované do adresáře.

Vzhledem k náročnosti této metody jsou zkoumány pouze poslední verze dokumentů (obrázek 8).

Obrázek 8: Postup hledání neasociovaných dokumentů

Akce REFRESH vrací pouze standardní odpověď bez žádných přidaných dat, na rozdíl od akce GET, která vrací sekvenční pole indexované od nuly, kde každý prvek obsahuje následující strukturu:

Hodnota	***Typ***	***Popis***
uuid	řetězec	UUID dokumentu v koši
user_id	celé číslo	identifikační číslo majitele dokumentu
document_name	řetězec	jméno dokumentu

Tabulka 4: Datová struktura jednoho záznamu o dokumentu v "koši"

9.2.2 Modul Directory

Adresářový modul Directory zabezpečuje funkčnost a integritu virtuální adresářové struktury, která je uložena v databázi. Pomocí tohoto modulu lze také asociovat soubory do adresářů nebo je z nich mazat. Navíc je možné přiřadit dokumentu v adresáři lokální jméno, které může být použito pro přepsání vlastního jména dokumentu.

Každý adresář je označen jednoznačným celočíselným inkrementovaným identifikátorem, neboť na rozdíl od dokumentů není pravděpodobné, že by bylo potřeba tuto strukturu v budoucnu škálovat napříč několika servery. Pokud by tato potřeba vznikla, je možné strukturu přesouvat pomocí kopírování a překladu identifikačního čísla adresáře nebo vytvořit zvláštní server (virtuální nebo fyzický), kde budou adresářové struktury uloženy.

První metoda má nevýhodu, že v případě vytvoření této kopie jsou ztracena data o asociaci k původní verzi adresářové struktury (jsou vygenerována nová identifikační čísla adresářů), ale na druhou stranu je zátěž přístupů rozložena na více serverů.

Druhá metoda, která vyhrazuje jeden server výhradně pro uchování virtuální adresářové struktury, problém ztráty asociativity nemá (za předpokladu, že při existenci jediného adresářového serveru nebudou další virtuální adresářové struktury vytvářeny na dokumentových serverech). Všechny dotazy na informace o adresářích jsou ale zpracovány jediným serverem, což může vést k jeho zvětšené zátěži.

Metoda GET

Při dotazu na vrácení objektu adresáře (metoda GET) je vrácena rozsáhlá datová struktura, která se skládá z několika podstruktur. Tyto podstruktury obsahují:

- Informace o daném adresáři.
- Informace o dokumentech v adresáři (datová struktura informací o dokumentu je uvedena výše v tabulce 2).
- Informace o podadresářích.

- Informace přímém rodiči.
- Informace o cestě k dokumentu v adresářové struktuře.
- Oprávnění daného uživatele přistupovat k danému adresáři.

U informací o adresářích a dokumentech jsou uloženy veškeré informace, aby se zamezilo velkému množství dotazů na zdrojový server, i za cenu porušení atomicity systému (systém vrací více než jeden objekt).

Hodnota	***Typ***	***Popis***
directory	struktura	obecné informace o adresáři
subdirs	pole	pole, kde každý prvek obsahuje informace o jednom adresáři
documents	pole	pole, kde každý prvek obsahuje informace o jednom dokumentu
documentsNames	asociativní pole	asociativní pole, kde klíčem je UUID dokumentu a hodnotou jméno dokumentu v adresáři
permisions	pole	pole logických hodnot indikující oprávnění k aktuálním adresáři
directParent	struktura	informace o přímém rodiči adresáře
path	pole	pole obsahující cestu k adresáři v adresářovém stromu

Tabulka 5: Kořenová struktura informací o adresáři

Pokud je vybraný adresář kořenový, obsahuje položka `directParent` stejné informace jako položka `directory`, která obsahuje obecné informace o aktuálním adresáři.

Hodnota	***Typ***	***Popis***
id	celé číslo	identifikační číslo adresáře
directory_name	řetězec	jméno adresáře
user_id	celé číslo	identifikační číslo majitele dokumentu
group_id	celé číslo	identifikační číslo skupiny, které dokument patří
mask	řetězec	maska oprávnění jako devíti znakový řetězec
parent_id	celé číslo	identifikační číslo rodičovského adresáře
depth	celé číslo	hloubka zanoření v adresářové hiearchii
created_at	časová značka	časová značka vytvoření dokumentu

Tabulka 6: Datová struktura uchovávající obecné informace o adresáři

Metoda POST

Pro vytvoření adresáře se metodě POST odesílá objekt jménem `directory` obsahující základní informace o adresáři, které jsou uloženy jako vlastnosti

Hodnota	***Typ***	***Popis***
directory_name	řetězec	jméno adresáře
group_id	celé číslo	identifikační číslo skupiny, které dokument patří
mask	řetězec	maska oprávnění jako devíti znakový řetězec
parent_id	celé číslo	identifikační číslo rodičovského adresáře

Tabulka 7: Vlastnosti objektu pro vytvoření adresáře

Metoda PUT

Metoda PUT slouží ke změně obecných informací adresáře a ke správě asociací dokumentů k adresáři. Hodnoty určené ke změnám jsou uloženy v objektu jménem `directory`.

Hodnota	***Typ***	***Popis***
directory_name	řetězec	jméno adresáře
group_id	celé číslo	identifikační číslo skupiny, které dokument patří
mask	řetězec	maska oprávnění jako devíti znakový řetězec
document	asociativní pole	asociativní pole, kde klíčem je UUID dokumentu a hodnotou je asociativní pole s pvkem „write“ nesoucím logickou hodnotu, jestli se doment nachází v adresáři (TRUE) nebo nenachází (FALSE)

Tabulka 8: Vlastnosti objektu pro aktualizaci adresáře

9.2.3 Modul Group

Modul správy skupin Group má na starosti uživatelské skupiny a přiřazování uživatelů. Pokud je skupina smazána, jsou asociované dokumenty a adresáře převedeny na skupinu nula nebo na uživatelem specifikovanou skupinu pomocí parametru.

Metoda GET

V případě, že je metoda GET volána s identifikátorem LIST, vrací metoda sekvenční pole obsahující struktury s obecnými informacemi o skupině:

Hodnota	***Typ***	***Popis***
id	celé číslo	identifikační číslo skupiny
group_name	řetězec	jméno skupiny

Tabulka 9: Obecné informace o skupině

Pokud je jako třetí parametr použito identifikační číslo, vrací metoda objekt obsahující dvě vlastnosti:

- Group – obsahuje obecné informace o skupině (Tabulka 8).
- Users – sekvenční pole obsahující informace o uživatelích, kteří jsou ve zpracovávané skupině.

Metoda POST

Při vytváření skupiny metodou POST je odeslán na server objekt `group`, s jedinou vlastností – `group_name`, která obsahuje jméno skupiny.

Metoda PUT

Metodě PUT je odeslán objekt `group`, s informacemi o případné změně jména a asociovaných uživatelů.

Hodnota	***Typ***	***Popis***
group_name	řetězec	jméno skupiny
user	pole	indexem je identifikační číslo uživatele a hodntou je logická hodnota, jestli má být uživatel do skupiny zapsán (TRUE) nebo ne (FALSE)

Tabulka 10: Struktura k aktualizaci informací o skupině

9.2.4 Modul User

Obsluhuje správu uživatelů a kromě základních kontrolerů obsahuje navíc jednoúčelový kontorler login, který obsahuje dvě akce viz tabulka 11.

signin	Přihlásí uživatele
signout	Odhlásí uživatele

Tabulka 11: Dodatečné kontrolery modulu User

Při zavedení programu existují vždy dva předem vytvoření uživatelé, kteří jsou potřeba pro správný chod systému. Jsou to root a guest.

root	Systémový administrátor
guest	Neregistrovaný uživatel

Tabulka 12: Výchozí uživatelé systému

Uživatel root je nejvyšší správce systému a má oprávnění přistupovat, měnit, vytvářet a mazat jakýkoliv obsah, včetně změn vlastníků a přístupových práv.

Při požadavku na objekt je jako identifikátor dosazeno buď identifikační číslo uživatele, nebo jeho login (login proto nesmí začínat číslicí).

Guest je uživatel pro nepřihlášené a neregistrované uživatele, není v žádné skupině a nesmí nijak manipulovat s obsahem (ani do veřejně zapisovatelného obsahu).

Každému uživateli je možné přidělit několik rolí vyjadřujících rozsah oprávnění, které uživatel v systému má. Při vytvoření uživatele není standardně přidělena žádná role, může tedy pouze používat vlastní adresář a adresáře, kde má z hlediska oprávnění možnost zapisovat. Tato oprávnění jsou daná systémem a není možné je nijak měnit. V základní verzi jsou tato oprávnění tři:

- USER_MANAGER – smí spravovat účty cizích uživatelů, vytvářet nové uživatele a přidělovat role.
- GROUP_MANAGER – smí vytvářet skupiny, spravovat uživatele ve skupinách a upravovat a mazat skupiny.
- DOCUMENT_MANAGER – smí spravovat dokumenty a adresáře, ke kterým by jinak neměl přístup.

Super uživatel root ke správě informací tyto role nepotřebuje, protože je má zabudované přímo v programu systému.

Metoda GET

Při volání metody GET s parametrem LIST místo identifikačního čísla uživatele vrací pole obsahující seznam všech uživatelů v systému. To je obsaženo v objektu jménem users. Informace, které jsou v každém prvku pole jsou uvedeny v tabulce 13.

Při dotazu s identifikačním číslem jsou vráceny objekty:

- User – obsahuje obecné informace o uživateli (tabulka 13).
- Groups – pole obsahující informace o skupinách, ve kterých je uživatel členem (tabulka 9).
- Roles – pole obsahující informace o rolích, které má uživatel přidělen (tabulka 14).

Hodnota	***Typ***	***Popis***
id	celé číslo	identifikační číslo uživatele
username	řetězec	uživatelské jméno
email	řetězec	emailová adresa
root_directory_id	celé číslo	identifikační číslo kořenového adresáře

Tabulka 13: Obecné informace o uživateli

Hodnota	***Typ***	***Popis***
id	celé číslo	identifikační číslo role
role_name	řetězec	jméno role

Tabulka 14: Informace o jedné uživatelské roli

Metoda POST

Při požadavku na vytvoření uživatele je odeslán objekt user, který obsahuje uživatelské jméno, emailovou adresu a přístupové heslo.

Hodnota	***Typ***	***Popis***
password	řetězec	heslo k účtu
username	řetězec	uživatelské jméno
email	řetězec	emailová adresa

Tabulka 15: Objekt user odesílaný při požadavku na vytvoření nového uživatele

Metoda PUT

Při změně informací o uživateli je možné změnit jak obecné informace, včetně hesla, tak i asociace k rolím a skupinám. Pro změnu hesla musí být odeslána kontrola hesla a staré heslo k účtu pro ověření totožnosti.

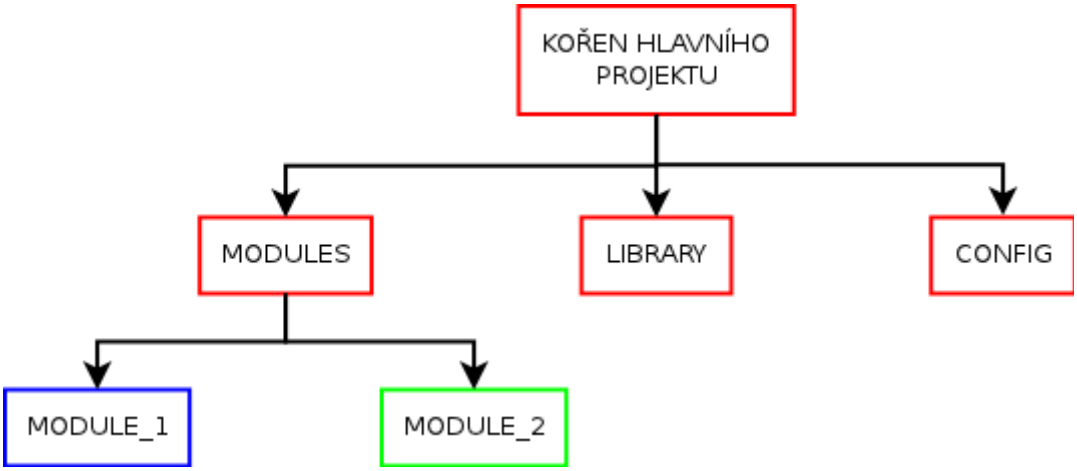
Hodnota	***Typ***	***Popis***
password	řetězec	nové heslo k účtu
pasword_check	řetězec	kontrola nového hesla
password_old	řetězec	staré heslo k účtu
username	řetězec	uživatelské jméno
email	řetězec	emailová adresa
group	pole	obsahuje informace pro update skupin
role	pole	obsahuje informace pro update rolí

Tabulka 16: Objekt user pro aktualizaci informací o uživateli

Hodnoty `group` a `role` jsou pole, kde indexem je identifikační číslo skupiny, resp. role, a hodnotou je logická hodnota vyjadřující, jestli má být uživateli přidělena skupina, resp. role (hodnota TRUE), nebo ne (hodnota FALSE).

9.2.5 Modul Project

Modul Project má účel zejména pro odstranění nedostatku systému při práci s větším množstvím souborů, a sice jeho atomicitu, kdy najednou můžeme pracovat vždy pouze s jedním objektem.



Obrázek 9: Projekt se založenými podprojekty

Základní jednotkou moduluje objekt `Project`. Jedná se o adresář, který vytváří virtuální kořen pro projektovou strukturu. Jeden adresář nemůže být kořenovým pro více projektů, ale je možné založit podprojekt, jehož kořenový adresář se nachází uvnitř jiného projektu. Toho lze využít například při spolupráci více vývojových týmů na jednom projektu, kdy je nežádoucí, aby jeden tým měnil práci jiného týmu.

Na obrázku 9 je znázorněna možná adresářová struktura projektu, na kterém spolupracuje více vývojových týmů. Červeně jsou vyznačeny adresáře patřící do hlavního projektu, který spravuje například vedoucí vývoje. Ten má přístup i do adresářů obou týmů. Červeně a zeleně jsou vyznačeny kořenové adresáře podprojektů, na kterých pracují dva rozdílné týmy a které nemají možnost si navzájem měnit data, pokud to není povoleno právy adresáře.

Jako identifikátor projektu pro dotazy je bráno identifikační číslo adresáře, který slouží jako kořen. V případě, že adresář byl ze serveru smazán skrz modul `Directory` a záznam o projektu stále existuje, je tento projekt smazán při prvním dotazu na něj, kdy systém zjistí, že kořenový adresář neexistuje.

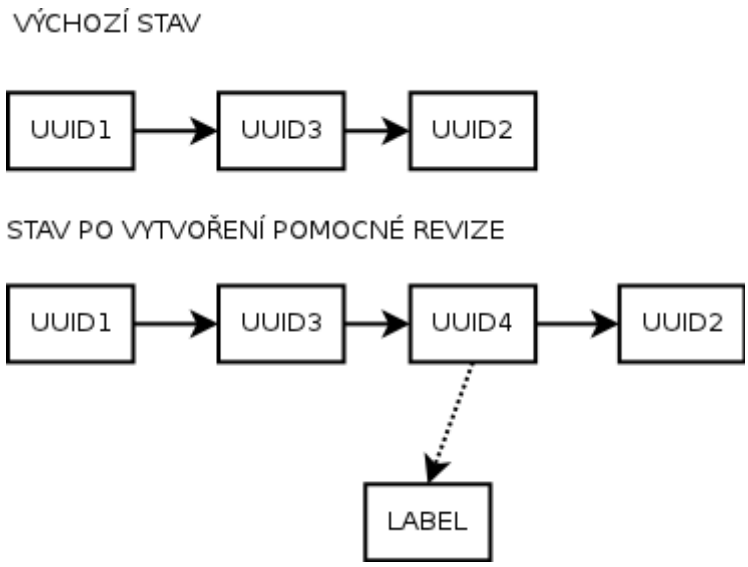
Významy standardních REST metod jsou následující:

- DELETE – smaže projekt.
- GET – vrací projekt ve formě zabaleného ZIP souboru.
- POST – vytvoří nový projekt.
- PUT – upraví data o projektu.

Vzhledem k poměrně velkému množství informací týkajících se projektu jsou kromě těchto obecných dat zavedeny ještě metody:

- COLABOURS – spravuje spolupracovníky na projektu.
- LABEL – Vytvoří „obraz“ projektu ve stavu, ve kterém se projekt v danou chvíli nachází.

Při vytváření obrazu projektu jsou vytvořeny u všech aktuálních verzí dokumentů nové revize bez změny jakýchkoliv dat, aby bylo možné odkazovat na momentální stav dokumentů. Nové UUID jsou uloženy do databáze (obrázek 10). V případě, že revize dokumentu v adresáři projektu není aktuální, je zapsáno přímo její UUID.



Obrázek 10: Vytvoření pomocné revize při vkládání aktuální verze do obrazu projektu

Společně s UUID dokumentu jsou navíc uloženy cesty ke všem instancím této revize dokumentu v projektu, aby bylo zabráněno vzniku porušení integrity, pokud by se adresářová struktura projektu změnila. Tyto cesty jsou uloženy jako sekvenční pole kódované ve formátu JSON.

Modul navíc přidává novou roli PROJECT_MANAGER, která umožňuje neomezeně přistupovat ke kterémukoliv projektu.

Atributy metod

Metoda `GET` může být volána s atributem `LABEL` obsahujícím identifikační číslo obrazu a specifikujícím, který obraz projektu je požadován. Pokud tento atribut uveden není, je vrácen projekt ve stavu, ve kterém se nachází v době požadavku.

Metoda GET

Pokud je metoda GET volána bez specifikace formátu, vrací ZIP soubor obsahující projekt. V případě, že je specifikován formát JSON, vrací obecné informace o daném projektu uložené v objektu `project`.

Hodnota	***Typ***	***Popis***
directory_id	celé číslo	identifikační číslo kořenového adresáře
project_name	řetězec	jméno projektu
description	řetězec	popis projektu
user_id	celé číslo	identifikační číslo majitele projektu
created_at	řetězec	časová značka vytvoření projektu

Tabulka 17: Obecné informace o projektu

Pokud je odeslán místo identifikačního čísla projektu řetězec `LIST`, vrací systém seznam všech projektů, které jsou zavedeny v systému jako sekvenční pole obsahující datové struktury popsané v tabulce 17.

Metoda POST

Pro vytvoření projektu je odeslán metodě POST objekt jménem `project`, kde je jako identifikátor použito identifikační číslo adresáře, který bude sloužit jako kořen celého projektu.

Objekt `project` obsahuje hodnoty uvedené v tabulce 17, vyjma hodnoty `directory_id`, která je uvedena v identifikátoru (v případě, že dotaz tuto hodnotu obsahuje, je ignorována).

Metoda PUT

Při aktualizaci dat metodou PUT je odeslán objekt `project` obsahující pouze dvě hodnoty, které je možné aktualizovat – viz tabulka 18.

Hodnota	***Typ***	***Popis***
project_name	řetězec	jméno projektu
description	řetězec	popis projektu

Tabulka 18: Data objektu pro aktualizaci informací o projektu

Metoda COLABOURS

Pokud je metoda COLABOURS volána bez vstupního objektu, je vrácen seznam lidí pracujících na projektu. Seznam je uložen v sekvenčním poli `user`, kde každý prvek obsahuje strukturu podle tabulky 13 na straně 34.

Pro změnu spolupracovníků projektu je nutné poslat na server objekt `user`. Je to sekvenční pole, kde každý prvek obsahuje identifikační číslo uživatele, který se má na projektu podílet.

Metoda LABEL

Obdobně jako metoda `COLABOURS`, má LABEL dva režimy. První je bez vstupního objektu, kdy vrací objekt `label`, který je sekvenční pole. Každý prvek pole obsahuje strukturu popsanou v tabulce 19.

Hodnota	***Typ***	***Popis***
id	celé číslo	identifikační číslo obrazu
directory_id	celé číslo	identifikační číslo projektu
user_id	celé číslo	identifikační číslo uživatele
label_name	řetězec	jméno obrazu
comment	řetězec	komentář
created_at	řetězec	časová značka vytvoření obrazu

Tabulka 19: Informace o obrazu projektu

Pokud je metoda volána se vstupním objektem label, který obsahuje prvky popsané v tabulce 19 kromě hodnot `id`, které jsou přiděleny systémem, a `directory_id`, které jsou definovány v těle URL.

9.2.6 Modul Note

Poznámovací modul Note umožňuje ukládat do systému poznámky k jednotlivým dokumentům. Poznámky je možné vztahovat buď k celému dokumentu, nebo k určité jeho revizi.

U poznámek s platností nad aktuální revizí je ovšem problém v tom, že modul nemůže poznat, kdy došlo ke změně revize, a tím pádem změnit asociaci poznámek na nové UUID neaktuálního obsahu. Proto jsou vždy při dotazu na poznámky u aktuálního dokumentu zkontrolovány časové značky (timestampy) poznámek a pomocí algoritmu jsou tyto poznámky převedeny na správný dokument.

Tento modul je zvláštní tím, že pro přístup k datům používá dva různé identifikátory objektu: identifikační číslo poznámky, identifikační číslo adresáře nebo UUID dokumentu s poznámkami (pro dotazy vedoucí k vrácení všech poznámek apod.).

Chování REST metod je následující:

- DELETE – smaže poznámku definovanou pomocí jejího identifikačního čísla.
- GET – získá informace v závislosti na typu použitého identifikátoru.
  - Identifikační číslo poznámky – získá informace o poznámce.
  - UUID dokumentu – získá výpis všech poznámek týkajících se dokumentu řazené o nejnovějšího po nejstarší.
  - Identifikační číslo adresáře – před číslem musí být písmeno „d“. To systém informuje,

že není požadována konkrétní poznámka, ale seznam poznámek k adresáři.

- POST – vytvoří novou poznámku, jako identifikátor je použito UUID cílového dokumentu nebo identifikační číslo adresáře.
- PUT – změní poznámku. Identifikátorem je identifikační číslo poznámky.

Uživatel smí mazat a upravovat pouze své poznámky. Prohlížet si však může kterékoliv (za předpokladu, že u dokumentu má právo pro čtení). Přidávat poznámky může ke kterémukoliv dokumentu, do kterého má oprávnění zapisovat.

Modul navíc přidává roli NOTE_MANAGER umožňující spravovat všechny poznámky, které jsou zavedeny v systému (role je nadřazena i přístupovým právům dokumentu).

Metoda GET

Metoda GET vrací seznam poznámek nebo jednu konkrétní poznámku (v závislosti na typu použitého identifikátoru).

Při dotazu na konkrétní poznámku vrací metoda objekt `note` obsahující datovou strukturu dle tabulky 20.

Hodnota	***Typ***	***Popis***
id	celé číslo	identifikační číslo poznámky
directory_id	celé číslo	identifikační číslo adresáře
document_uuid	řetězec	UUID dokumentu
user_id	celé číslo	identifikační číslo uživatele, který poznámku vytvořil
note	řetězec	poznámka
created_at	řetězec	časová značka vytvoření poznámky

Tabulka 20: Informace o poznámce

Z položek `directory_id` a `document_uuid` je použita vždy jen jedna podle typu objektu, ke kterému poznámka patří. Pokud poznámka patří k dokumentu, je vyplněna hodnota `document_uuid` a `document_id` je `NULL`. Pokud je vyplněno `directory_id`, tak je `document_uuid` `NULL`.

Pokud je odeslán identifikátor adresáře nebo dokumentu, server vrací objekt `note`, který obsahuje sekvenční pole. Každý prvek pole potom obsahuje strukturu popsanou v tabulce 20. Data jsou řazena od nejnovějšího po nejstarší.

Metoda POST a PUT

Pro vytvoření nové poznámky pomocí metody POST je jako identifikátor v URL použito identifikační číslo adresáře nebo UUID dokumentu, ke kterému bude poznámka přiřazena. Při změně existující poznámky (metoda PUT) je jako identifikátor použito identifikační číslo poznámky.

Data poznámky jsou odeslána v objektu `note`, který obsahuje hodnotu `note` s obsahem poznámky.

9.2.7 Modul Documentation

Dokumentační modul documentation poskytuje vývojářům svázat dokument nebo adresář s jeho dokumentací, která se nachází v jiném dokumentu. Při vytváření vazby dokumentace je nutné, aby UUID dokumentu s dokumentací odkazovalo na aktuální verzi tohoto dokumentu, jinak volání metody skončí chybou. Stejně tak musí být při vazbě typu dokument-dokument dokumentovaný objekt aktuální verzí.

Jeden objekt může být navázán na více objektů dokumentace a naopak jeden objekt dokumentace může popisovat více objektů (relační vztah „Many-To-Many“).

Při volání metody GET existují tři typy identifikátorů:

- Identifikační číslo asociace – vrací informace o jedné asociaci.
- UUID dokumentu – vrací seznam asociací k dokumentu.
- Identifikační číslo adresáře – identifikačnímu číslu musí předcházet znak „d“, aby se zabránilo záměně s identifikačním číslem asociace.

Metoda GET

Pokud je metoda GET volána s identifikátorem asociace, je vrácena datová struktura popsaná v tabulce 21. Tato struktura je uložena v objektu `documentation`.

Hodnota	***Typ***	***Popis***
id	celé číslo	identifikační číslo asociace
documented_document_uuid	řetězec	UUID dokumentovaného dokumentu
documented_directory_id	celé číslo	identifikační číslo dokumentovaného adresáře
documentation_document_uuid	řetězec	UUID dokumentu sloužícího jako dokumentace
user_id	celé číslo	identifikační číslo uživatele, který asociaci vytvořil
comment	řetězec	komentář k asociaci
created_at	řetězec	časová značka vytvoření asociace

Tabulka 21: Struktura informací o asociaci dokumentace

Pokud je použit jiný typ identifikátoru, vrací metoda dva objekty – `documented_by`, který obsahuje seznam asociací dokumentace vázané ke zkoumanému objektu, a `documentation_for` obsahující seznam asociací k objektům, kterým zkoumaný objekt slouží jako dokumentace.

Metoda GET a POST

Při metodě POST je jako identifikátor v těle URL použito identifikační číslo adresáře nebo UUID dokumentu, který bude dokumentován. Tělo požadavku potom obsahuje objekt `documentation` s hodnotami uvedenými v tabulce 22.

Hodnota	***Typ***	***Popis***
documentation_document_uuid	řetězec	UUID dokumentu sloužícího jako dokumentace
comment	řetězec	komentář k asociaci

Tabulka 22: Struktura pro vytvoření nové asociace

Pro aktualizace informací se jako identifikátor v URL použije identifikační číslo asociace a jako objekt nesoucí informace se použije stejný objekt jako u metody POST, ale bez hodnoty `documentation_document_uuid`, která je v případě použití ignorována.

9.2.8 Modul Message

Modul Message umožňuje posílání zpráv mezi uživateli uvnitř systému. Jedná se o modul, který je naprosto nezávislý na ostatních modulech a slouží pouze ke zjednodušení práce na projektech.

Definuje tři nové objekty:

- ADRESARY – adresář kontaktů (neimplemetuje metodu PUT).
- RECIEVED – přijaté zprávy (implementuje pouze GET a DELETE).
- SENT – odeslané zprávy.

Zajímavostí je objekt ADRESARY, ke kterému se přistupuje jinak než ke standardním objektům. Adresář má uživatel vždy právě jeden, a proto odpadá poslední část dotazu (identifikátor), který je v tomto případě zbytečný (v rámci jednoho uživatele můžeme přeneseně považovat adresář kontaktů za objektový návrhový vzor singleton).

Singleton specifikuje, jak vytvořit třídu, která bude mít nejvýše jednu instanci. Tato instance přitom nemusí být instancí vlastní třídy [10].

Volání objektů RECIEVED a SENT bez identifikátoru vrací výpis pošty, který je možné upravit podle níže uvedených parametrů, a který obsahuje všechny informace kromě vlastního obsahu sdělení.

Vzhledem k udržení soukromí komunikace, tento modul nevytváří žádné role umožňující přístup ke komunikaci třetích osob.

Parametry metod

U metody GET volané s identifikátorem LIST je navíc možné použít filtrovací parametry pro vyhledávání v poště:

- Without – pole nebo řetězec, umožňuje vypustit některé složky pošty.

    - Inbox – vypustí příchozí zprávy.
    - Outbox – vypustí odchozí zprávy.
    - Concepts – vypustí uložené zprávy.
- ItemsPerPage – celé číslo, počet záznamů na stránku jednoho výpisu (standardně 100).
- Page – celé číslo větší než jedna, stránka výpisů (standardně 1).

Metoda GET objektu ADDRESSARY

Metoda GET provedená u objektu ADDRESSARYvrací objekt `adresary`, který je sekvenčním polem. Každý prvek tohoto pole obsahuje strukturu dle tabulky 23.

Hodnota	***Typ***	***Popis***
id	celé číslo	identifikační číslo záznamu v adresáři
user_id	celé číslo	identifikační číslo majitele adresáře
target_user_login	řetězec	login cílového uživatele

Tabulka 23: Struktura záznamu v adresáři

Metoda POST objektu ADRESARY

Jako identifikátor je uvedeno identifikační číslo uživatele, který má být do adresáře přidán, žádný objekt není posílán na server.

Metoda GET objektu RECIEVED

Identifikátorem pro metodu GET objektu RECIEVED je použito identifikační číslo zprávy. Vrácen je objekt `recieved`, který má strukturu popsanou v tabulce 24.

Hodnota	***Typ***	***Popis***
id	celé číslo	identifikační číslo zprávy
user_id	celé číslo	identifikační číslo příjemce
from_user_id	celé číslo	identifikační číslo odesilatele
subject	řetězec	předmět zprávy
content	řetězec	obsah zprávy
recieved_at	řetězec	časová značka přijetí zprávy
is_readed	logická hodnota	indikátor přečtení zprávy

Tabulka 24: Struktura záznamu o doručené poště

Pokud identifikátor není uveden, obsahuje objekt `recieved` sekvenční pole. Prvek pole je struktura z tabulky, ovšem bez hodnoty `content`. Tato hodnota je vynechána z důvodu úspory přenesených dat.

Metoda GET objektu SENT

Jako identifikátor pro metodu GET u objektu SENT je uvedeno identifikační číslo odeslané zprávy. Metoda vrací objekt `sent`, který obsahuje datovou strukturu popsanou v tabulce 25.

Pokud není identifikátor uveden, vrací metoda v objektu `sent` sekvenční pole. Prvkem toho pole je opět struktura z tabulky 25, ale bez hodnoty `content`. Tato absence je stejně jako u metody RECIEVED z důvodu úspory přenesených dat.

Hodnota	***Typ***	***Popis***
id	celé číslo	identifikační číslo zprávy
user_id	celé číslo	identifikační číslo příjemce
targets	řetězec	seznam loginů cílových uživatelů
subject	řetězec	předmět zprávy
content	řetězec	obsah zprávy
created_at	řetězec	časová značka vytvoření zprávy
sendend_at	řetězec	časová značka odeslání zprávy

Tabulka 25: Datová struktura informací o odeslané zprávě

Metody POST a PUT objektu SENT

Pro aktualizaci dat pomocí metody PUT je uveden jako identifikátor identifikační číslo zprávy, zatímco metoda POST je volána bez identifikátoru. Aby bylo možné metodu PUT volat, nesmí být zpráva odeslána.

Data potřebná pro vytvoření nebo aktualizaci jsou uložena v objektu `sent`, který obsahuje hodnoty uvedené v tabulce 26.

Hodnota	***Typ***	***Popis***
targets	řetězec	seznam loginů cílových uživatelů
subject	řetězec	předmět zprávy
content	řetězec	obsah zprávy

Tabulka 26: Struktura ke změně odesílané zprávy

9.2.9 Modul FTP

Modul FTP slouží k automatické aktualizaci cílového serveru pomocí FTP protokolu a funguje pouze, pokud je nainstalován modul Project, se kterým úzce spolupracuje. V případě velkého projektu je možné využít funkce, která na cílový server odešle projekt zabalený v ZIP souboru a nahraje skript, který projekt rozbalí přímo na serveru. Při použití této funkce ale může dojít ke konfliktu, pokud je na cílovém serveru povoleno přepisování adres pomocí RewriteEngine. Doporučuje se tedy RewriteEngine před aktualiuzací vypnout nebo potlačit jeho funkci.

Implementuje pouze metodu POST, který jako identifikátor používá identifikační číslo projektu (identifikační číslo kořenového adresáře projektu).

Parametry metod

Metoda POST může být rozšířena o parametr LABEL, který specifikuje obraz projektu aktualizovaný na server. Pokud tento parametr uveden není, je aktualizován stav projektu v okamžiku požadavku.

9.2.10 Modul Plan

Organizační modul Plan slouží k jednoduché organizaci práce více lidí na jednom projektu. Stejně jako pro modul FTP je potřeba k chodu modulu Plan mít nainstalovaný modul Project, ze kterého získává informace o dokumentech.

Plánování se může týkat celého projektu, adresáře nebo dokumentu. Proto je zde využit systém pro více typů identifikátorů, který vychází z předpokladu, že počet cifer identifikační ho čísla adresáře nikdy nedosáhne čtyřiceti (délka řetězce UUID).

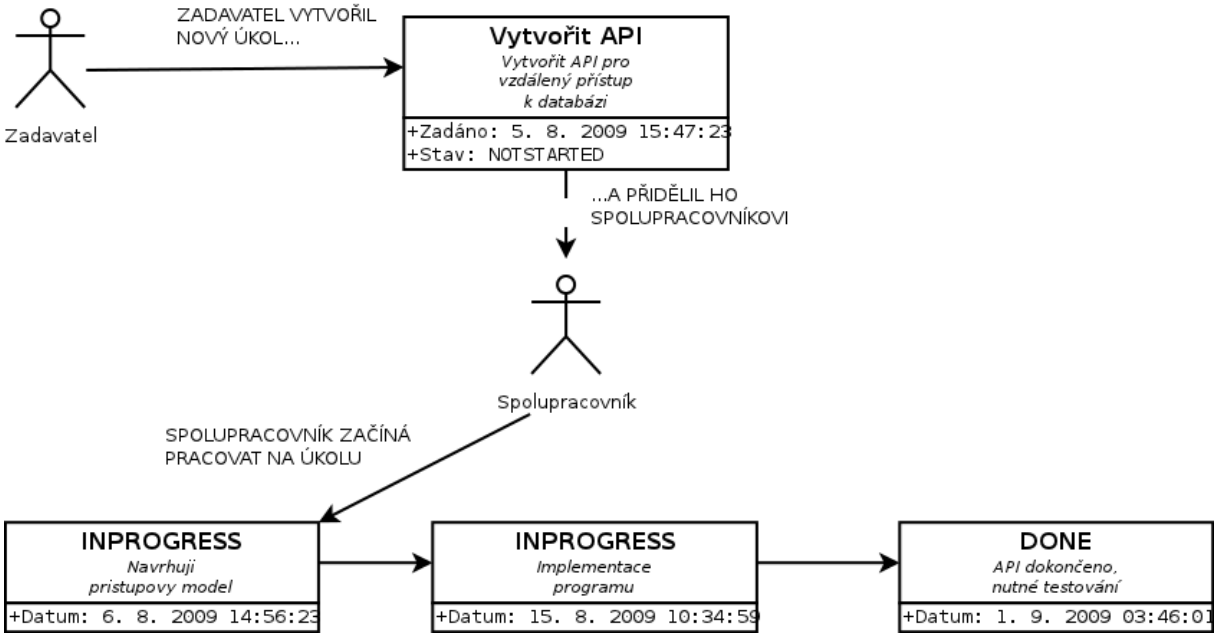
Pro správu plánování je možné zvolit jednoho člověka nebo více lidí (určuje správce projektu), takzvané koordinátory, kteří mohou pro ostatní vývojáře plánovat postup práce.

Cílový uživatel, který má úkon provést, mu může přidělit jeden z následujících stavů

- DONE – zadaný úkol byl splněn.
- DELAY – úkol se opozdí oproti termínu.
- INPROGRESS – úkol právě probíhá.

- NOTSTARTED – úkol nebyl započat.
- PUTOFF – úkol byl odložen na později.
- NEVER – úkol nebyl splněn a uživatel nepočítá s tím, že by ho kdy splnil.

Výchozím stavem při zadání úkolu je stav NOTSTARTED (pokud zadavatel neuvede jinak). Při každé změně stavu je možné uvést komentář, který vysvětluje, proč k té dané změně došlo. Veškeré tyto změny jsou zaznamenávány, a je tedy možné si kdykoliv projít jejich historii. V případě potřeby je možné přidat nový komentář bez změny stavu (jako nový stav se zadá stav, ve kterém se úkol právě nachází). Příklad změn stavů je na obrázku 11



Obrázek 11: Ukázka možného životního cyklu úkolu

Modul zavádí navíc jednu novou roli PLAN_MANAGER, která umožňuje manipulovat s plány všech projektů.

Základní objekt modulu má navíc metodu MANAGER, která používá jako identifikátor identifikační číslo projektu a umožňuje definovat projektové manažery plánování.

Modul definuje navíc objekt HISTORY, který definuje pouze metodu GET.

Metoda GET základního objektu

Metoda GET (základního objektu) volaná bez identifikátoru vrací dva objekty – `recieved` a `sent`, které obsahují sekvenční pole. Každý prvek pole je struktura definována v tabulce 27.

Objekt `recieved` obsahuje seznam úkolů, které byly zadány uživateli a naopak objekt `sent` obsahuje úkoly, které uživatel zadal.

Hodnota	***Typ***	***Popis***
id	celé číslo	identifikační číslo úkolu
user_id	celé číslo	identifikační číslo zadavatele úkolu
for_user_id	celé číslo	identifikační číslo příjemce úkolu
task_content	řetězec	zadání úkolu
current_status	řetězec	momentální stav kolu
current_comment	řetězec	komentář k momentálnímu stavu
created_at	řetězec	časová značka vytvoření úkolu
modified_at	řetězec	časová značka poslední změny úkolu
deadline_at	řetězec	časová značka termínu splnění úkolu

Tabulka 27: Struktura dat jednoho úkolu

Pokud je metoda volána s identifikátorem, vrací objekt `task` obsahující stejnou strukturu jako prvky polí `recieved` a `sent`.

Metoda POST základního objektu

Jako identifikátor u metody POST je použito identifikační číslo uživatele, kterému je úkol přidělován. Data jsou uložena v objektu `task` se strukturou popsanou v tabulce 28.

Hodnota	***Typ***	***Popis***
task_content	řetězec	zadání úkolu
current_status	řetězec	momentální stav kolu
current_comment	řetězec	komentář k momentálnímu stavu
deadline_at	řetězec	časová značka termínu splnění úkolu

Tabulka 28: Struktura odesílaná pro vytvoření úkolu

Pokud jsou hodnoty `current_status` a `deadline_at` nepovinné a nejsou uvedeny, vloží se výchozí hodnoty (NOTSTARTED pro `current_status` a 0 pro `deadline_at`).

Metoda PUT základního objektu

Při aktualizaci úkolu (metoda PUT) je vytvořen nový záznam v historii jeho změn a jsou uložena nová data o aktuálním stavu.

Aktualizační objekt je uložen v parametru `task`, který obsahuje pouze dvě hodnoty. Tyto hodnoty jsou uvedeny v tabulce 29.

Hodnota	***Typ***	***Popis***
current_status	řetězec	momentální stav kolu
current_comment	řetězec	komentář k momentálnímu stavu

Tabulka 29: Aktualizační objekt úkolu

Metoda MANAGER základního objektu

Jako identifikátor pro metodu MANAGER základního objektu je použito identifikační číslo kořenového adresáře projektu a obsahuje objekt `user`, který je sekvenčním polem, kde prvky jsou identifikační čísla uživatelů, které v projektu smějí přidělovat úkoly.

Metoda GET objektu HISTORY

GET metoda objektu HISTORY Vrací objekt `history`, který obsahuje sekvenční pole. Každý prvek pole je struktura dle tabulky. Tyto struktury jsou řazeny od nejnovějšího po nejstarší.

Hodnota	***Typ***	***Popis***
id	celé číslo	identifikační číslo záznamu
task_id	celé číslo	identifikační číslo rodičovského úkolu
status	řetězec	stav
comment	řetězec	komentář ke stavu
created_at	řetězec	časová značka vytvoření změny

Tabulka 30: Struktura záznamu historie úkolu

9.3 Implementace datového modelu

Datový model je rozdělen na dvě části, souborovou a databázovou.

V souborové části jsou uchovávány obsahy jednotlivých dokumentů, které jsou uloženy pod svým UUID bez informace o typu souboru. V případě, že v linii vývoje dokumentu jsou, se při změně informací o dokumentu nezmění jeho obsah (dojde například pouze k přejmenování). Program starý

obsah nenakopíruje, ale z důvodu úspory diskového prostoru na něj vytvoří pouze symbolický odkaz (obdoba zástupce z MS Windows). V případě, že předchozí soubor je také pouze odkazem, najde program původní skutečný soubor a odkáže na něj. Předejde tak havarijnímu stavu, kdy by se při smazání jednoho odkazu následující řetězené odkazy staly neplatnými.

Databázová část modelu obstarává zbytek datové infrastruktury a je rozdělena v základu do dvou sekcí podle účelu, kterému slouží. Sekce je možné poznat podle prefixu jména tabulky (další programové moduly přidávají své sekce, které jsou popsány níže).

documents	Správa úložiště dat
users	Správa informací o uživatelích

Tabulka 31: Sekce databázových tabulek

Tabulky jsou mezi sebou provázány simulací cizích klíčů, která se nachází v databázové vrstvě Zend frameworku (knihovna Zend_Db).

9.3.1 Tabulky uživatelů

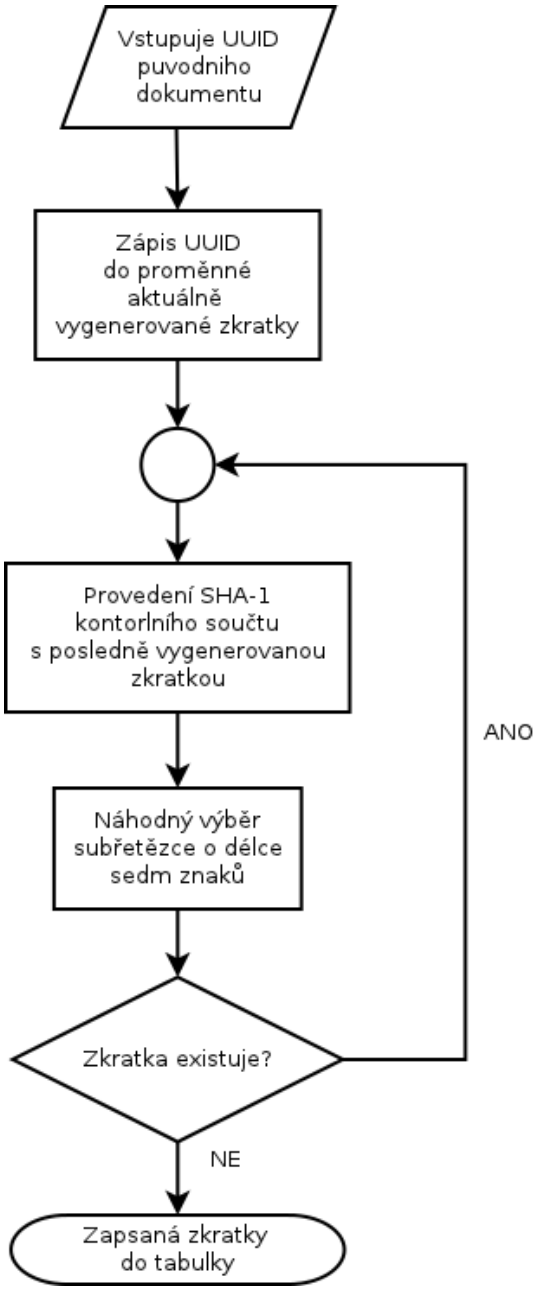
V tabulkách uživatelů jsou uloženy informace o uživatelích, jejich přiřazení k rolím a přiřazení ke skupinám.

Základem je tabulka `users`, která obsahuje vlastní informace o uživateli a na kterou se většina tabulek modelu odkazuje. Je v ní také informace o kořenovém adresáři uživatele. Heslo uživatele je uloženo jako 40 bytový SHA-1 otisk skutečného hesla a 40 bytové „soli“, která je náhodně generována zároveň s vytvořením účtu a slouží ke ztížení prolomení hesla v případě, že potenciální útočník zcizí SHA-1 otisky hesel.

Role uživatelů jsou uloženy v tabulce `users_roles`, která je asociována s tabulkou `users` pomocí „many-to-many“ vztahu skrz tabulku `users_has_many_roles`.

Skupiny uživatelů jsou uloženy v tabulce `users_groups`, která je provázána s tabulkou `users` pomocí „many-to-many“ vztahu skrz tabulku `users_has_many_groups`.

9.3.2 Tabulky dokumentů



Obrázek 12: Algoritmus generování zkratek

Základem databázové struktury dokumentů je tabulka `documents`, která obsahuje obecná data o měnících se dokumentech. To znamená:

- UUID.
- Jméno dokumentu.
- MIME typ dokumentu.
- Identifikační číslo uživatele, ke kterému dokument náleží.
- Identifikační číslo skupiny uživatelů dokumentu.
- Informace o přístupových právech, která jsou uložena v ASCII formátu.
- Časová značka (timestamp) vytvoření záznamu.
- Informaci, jestli se jedná o aktuální revizi dokumentu.
- Velikost dokumentu v bytech.

S touto tabulkou úzce souvisí tabulky `documents_history`, uchovávající informace

o postupných změnách dokumentu. Každý řádek této tabulky obsahuje inkrementované identifikační číslo položky v historii, UUID o jednu revizi starší formy dokumentu, UUID dokumentu, ke kterému se záznam vztahuje, a poté UUID původního obsahu dokumentu a UUID nejaktuálnějšího obsahu dokumentu. Jako poslední je sloupec s časovou značkou vytvoření záznamu, který sice obsahuje i vlastní dokument, ale při procházení historie změn by musel systém získávat i informace z jiné tabulky. To by zpomalovalo chod programu, proto je zde tento údaj uložen duplicitně.

Pomocnou tabulkou k dokumentům je `documents_shortcuts`, která obsahuje zkrácené UUID aktuálních dokumentů. Jedná se o 7 bytový, unikátní řetězec, pomocí kterého lze k nejaktuálnějšímu obsahu dokumentu přistupovat bez nutnosti použití čtyřiceti znakového plného UUID. Teoreticky lze vytvořit takovouto zkratku pro:

$$x^n = 16^7 = 268435456\ dokumentů$$

kde x vyjadřuje počet použitelných znaků na jedné pozici (SHA-1 využívá šestnáctkovou soustavu) a n je počet míst pro znaky. Přímo v definici databázové tabulky je uvedeno, že sloupec uchovávající zkratku je unikátní. To zabezpečuje jedinečnost zkratky přímo na databázové úrovni. Zkratka je generována reprezentací tabulky v programu vždy při vložení nového záznamu pomocí algoritmu na obrázku 12.

Implementace výše zmíněného algoritmu generování zkratek je (výtah ze souboru /Application/Model/DocumentsShortcuts.php, metoda generateShortcut):

```
...
//inicializace prazdne zkratky
$shortCut = "";

//inicializace generatoru nahodnych cisel
srand(time() + microtime(false));

//generovani zkratky, dokud zkoumana zkratka existuje
do {
	//volna zkratka zatim nebyla nalezena

	//hashovani stare zkratky spolu s UUID dokumentu
	$shortCut = sha1($shortCut.$uuid);

	//nahodny vyber sedmi znaku, ktere se stanou zkratkou
	$startChar = rand(0, 33);
	$shortCut = substr($shortCut, $startChar, 7);

	//kontrola, jeslti je zkratka vyuzivana
	$row = $this->fetchRow($this->select(false)
		->where("shortcut like ?", $shortCut));
} while ($row);

//vytvoreni radku nove zkratky
$row = $this->createRow(array(
	"uuid" => $uuid,
	"shortcut" => $shortCut
));

//uozeni radku zkratky
$row->save();

//vraceni objektu se radkem zkratky
return $row;
...
```

Informace o virtuální adresářové struktuře jsou uloženy v tabulce `documents_directories`. Tato data nejsou verzovaná, a proto není možné zpětně zjistit změny v obsahu adresáře v průběhu jeho existence. Tabulka uchovává obecné informace o adresáři (jméno, přístupová práva, vlastník apod.) i informace o jeho pozici ve stromové struktuře, což je hloubka zanoření ve struktuře a identifikační číslo jeho rodiče. Dokud adresář není prázdný (obsahuje vnořené adresáře nebo dokumenty), není možné ho smazat.

Asociace dokumentů a adresářů jsou realizovány tabulkou `documents_has_many_directories`, která obsahuje vždy UUID dokumentu a identifikační číslo adresáře. Tyto dvě hodnoty slouží zároveň jako primární klíč této tabulky.

Pokud není dokument asociován v žádném adresáři, je UUID tohoto dokumentu zaneseno do tabulky `documents_unassigned`, která slouží účelu koše a zajišťuje dohledatelnost těchto dokumentů mimo adresářovou strukturu. Tato tabulka obsahuje kromě UUID dokumentu ještě identifikační číslo majitele a jméno dokumentu, aby nebylo nutné při výpisu „koše“ vést dotaz přes více než jednu tabulku.

9.4 Datový model modulů

Všechny moduly, popsané v části o implementaci programu, využívají své databázové tabulky, které jsou zpravidla provázány s tabulkami hlavního systému. Některé se váží i na jiné moduly, na kterých jsou závislé.

9.4.1 Modul Project

Základem databázové struktury modulu Project je tabulka `projects`, která obsahuje informace o kořenovém adresáři projektu, jeho jméně, uživateli, který projekt vytvořil, a datu a času vytvoření projektu. Na tuto tabulku jsou dále přímo navázány tabulky `projects_colabours` a `projects_labels`.

Tabulka `projects_colabours` uchovává N:M asociace mezi projekty a uživateli. Tím přiděluje danému projektu spolupracovníky, kteří mohou manipulovat s projektem, aniž by k tomu měli oprávnění vyplývající z masky.

Data tabulky `projects_labels` udržují obecné informace o obrazu projektu. Je to jeho identifikační číslo, jméno, informace o uživateli, který obraz vytvořil, a informace o datu a času jeho vytvoření.

K tabulce `projects_labels` je navázána N:M relační tabulka `projects_labels_documents`. Ta kromě identifikačního čísla obrazu a UUID dokumentu navíc obsahuje vlastní identifikační číslo a binární sloupec se seznamem cest k instancím dokumentu v projektu. Dále k tabulce `projects_labels` existuje relace 1:1 na tabulku `projects_labels_contents`, která obsahuje data ZIP archivů s obrazy projektů.

9.4.2 Modul Notes

Modul Notes ukládá informace do jediné tabulky jménem notes, která obsahuje informace o asociacích poznámek k dokumentům a adresářům. Vazby jsou řešeny pomocí UUID dokumentu, ke krému je poznámka určena nebo identifikačního čísla adresáře, kterému je poznámka určena. Pokud se poznámka vztahuje k adresáři, obsahuje sloupec s UUID dokumentu hodnotu `NULL` a pokud se poznámka vztahuje k dokumentu, má hodnotu `NULL` sloupec s UUID dokumentu.

9.4.3 Modul Documentation

Modul Documentation obsahuje jednu tabulku jménem `documentations`, která obsahuje informace o asociacích dokumentace (objekty `document`) k adresářům a jiným dokumentům. Obsahuje unikátní klíč skládající se z UUID dokumentace, cílového dokumentu a cílového adresáře. Pokud se dokumentace vztahuje k adresáři, má informace o cílovém dokumentu hodnotu `NULL` a pokud se dokumentace vztahuje k dokumentu, má informace o cílovém adresáři hodnotu `NULL`.

9.4.4 Modul Message

Datový model zpráv je rozdělen do tří nezávislých tabulek, které uchovávají data o:

- Odchozích zprávách.
- Příchozích zprávách.
- Adresáři častých kontaktů.

V tabulce odchozích zpráv (`messages_sent`) jsou uloženy informace o obsahu zprávy, uživateli, který zprávu odeslal, značka času vytvoření, značka času odeslání a informace o příjemcích. Pokud je značka odeslání rovna nule, je zpráva pouze rozepsaným konceptem.

Příjemci jsou uloženi jako řetězec obsahující seznam uživatelských jmen oddělených mezerou. K rozdělení a zjištění identifikačních čísel příjemců dochází až v okamžiku odeslání zprávy.

Tabulka `messages_recieved` obsahuje informace o příchozích zprávách a je do ní zapisováno v okamžiku odeslání zprávy. Obsahuje data o příjemci, odesílateli, obsahu zprávy, času přijetí a stavovou informaci o přečtení.

Adresář obsahuje pouze identifikační číslo záznamu, informaci o uživateli, kterému kontakt patří a uživatelské jméno kontaktu. Kombinace uživatelského jména a cílového loginu musí být vždy unikátní.

9.4.5 Modul FTP

FTP modul ukládá pouze logovací informace o proběhnutých aktualizacích na vzdálený server.

Každý záznam obsahuje informaci, kdo aktualizaci provedl, který projekt byl aktualizován (případně informaci o obrazu projektu), čas aktualizace a parametry, se kterými byla aktualizace spuštěna.

9.4.6 Modul Plan

Datový model modulu Plan se stává ze tří tabulek, které uchovávají data o:

- Správcích úkolů k projektům.
- Přidělených úkolech.
- Historii změn v úkolech.

Tabulka správců projektu (`plans_managers`) je tabulka vztahu N:M mezi tabulkou projektů a tabulkou uživatelů.

Tabulka `plans_tasks` obsahuje informace o vytvořených úkolech, jejich přiděleních k uživatelům a informace o vytvoření, poslední změně a aktuálním stavu úkolu.

V tabulce historie změn (`plans_task_history`) jsou uloženy změny ve statusech (ke každému úkolu musí být vždy uložena alespoň jedna změna – založení úkolu). Ukládá se čas změny, nový stav a komentář ke změně.

10 KLIENTSKÁ APLIKACE

Pro přístup k datům je možné použít aplikaci psanou v jazyce C++ pro operační systém Linux. Tato aplikace umožňuje provádět operace se základními objekty bez nutnosti použití webového rozhraní.

Jako základní objekt se rozumí:

- Adresář.
- Dokument.
- Uživatel.
- Skupina.

Rozšiřující objekty (projekt, poznámky,...) program nepodporuje, protože se jedná o datové struktury pocházející nikoliv z jádra aplikace, ale ze zásuvných modulů.

Pro komunikaci se serverem používá program knihovnu CURL, která podporuje nejběžnější komunikační protokoly (HTTP, FTP, POP3,...) a knihovnu JSONCPP pro parsování odpovědí zaslaných serverem.

Pro zpracování informací by bylo možné využít knihovnu JSON-GLIB, která se nachází přímo v Linuxovém prostředí Gnome, ale potom by byla aplikace svázána přímo s tímto prostředím.

Uživatelské rozhraní je řešeno formou provázaných textových nabídek, kdy uživatel píše příkazy. Přístup k jednotlivým objektům je vyřešen lokálním identifikačním číslem, které je každému objektu přiděleno přímo programem. Toto identifikační číslo není nijak svázané s identifikátorem objektu na serveru.

Nabídka menu je řešena jako třída, která dědí ze základní třídy definující společné metody všech nabídek. Společné metody jsou:

- Help – vypíše krátkou nápovědu.
- Menu – vypíše možné příkazy.
- Refresh – znovu načte a zobrazí data.
- Toroot – přesune se do základní nabídky.
- Path – vypíše cestu k aktuální nabídce.

Mapa nabídek a operací je na obrázku 13.

10.1 Základní třída nabídky

Třída `Menu` obsahuje několik asociativních a sekvenčních polí (typ `Map` a `Vector` ze standardní knihovny C++ - STL).

První pole s názvem `commands` obsahuje ukazatele na funkce příkazů, které je možné v daném menu použít. Je to asociativní pole `Map`, které jako klíč používá řetězce typu `string` (STL).

Další je pole je sekvenční `items`, které ke klíčování používá interní identifikátory (pořadí prvku v poli) a hodnotou je potomek třídy `Item`.

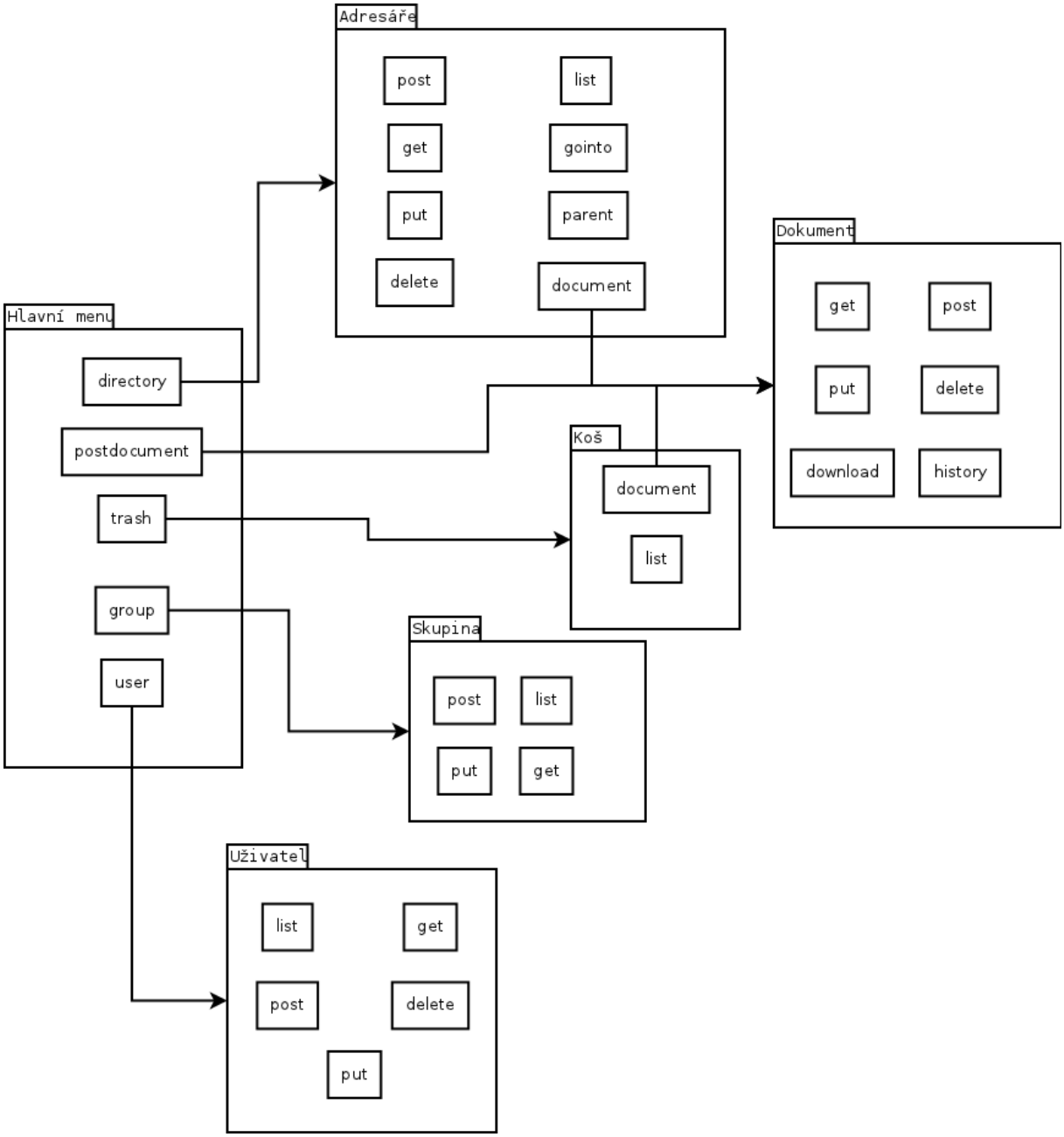
10.2 Základní třída položky

Třída s názvem `Item` slouží pouze ke sjednocení základních metod u různých typů objektů a aby je bylo možné uchovávat v jednotném poli v základním objektu nabídky. Metody deklarované touto třídou jsou zpravidla abstraktní a definice se nachází až v potomku této třídy.

Tyto metody jsou:

- Info – vypíše informace o objektu.
- ShortInfo – vypíše zkrácené informace o objektu.
- Put – upraví informace o objektu.
- Delete – smaže objekt.

Některé objekty definují i další metody, které jsou naznačeny na obrázku 13. Podrobnější popis těchto metod je přímo v aplikaci a lze vyvolat příkazem `help`.



Obrázek 13: Schéma nabídek uživatelského rozhraní

11 ZÁVĚR

Cílem bakalářské práce bylo vytvořit verzovací systém, který by byl schopen vývojář zprovoznit na jakémkoliv českém WWW hostingu, který používá platformu LAMP.

Práce pojednává o základech problematiky verzování souborů, o pojmech s tím spojených a modelech, které jsou používány pro přístup a manipulaci s daty, uloženými v systému.

Popisuje počátky verzovacích systémů a jejich rozdílné přístupy k datům. Informuje o momentálně nejpoužívanějších systémech, kde je u systému Git uveden příklad použití. Tyto systémy sloužily částečně i jako inspirace pro tvorbu nového systému popsaného ve zbytku práce.

Vzhledem k horší možnosti optimalizace u interpretovaných jazyků nebylo možné vytvořit plnohodnotného zástupce za moderní systémy, jako SubVerion nebo Git. Pro jednoduchou správu menších projektů lze software bez problémů využít. Další možností, jak využít systém je internetové úložiště dat, které může sloužit jako multimediální knihovna, kam přistupují vzdálené aplikace, například pomocí technologie AJAX, sloužící k asynchronní komunikaci se serverem.

Instalace programu na veřejném hostingu ale zároveň přináší komplikace v rozdílech u nastavení serverů u různých poskytovatelů. Hlavním kamenem úrazu jsou značná omezení pro přidělování paměti, maximální velikosti souboru, odesílaného na server pomocí HTTP a maximální doba otevřeného spojení – pokud je velký soubor odesílán přes pomalé připojení, může se stát, že server uzavře spojení dřív než je celý přenos dokončen.

Dalším možným problémem tohoto systému je absence klientských programů, které by dokázaly využít všech možností, které systém nabízí, aniž by k nim musel uživatel přistupovat skrz integrované uživatelské rozhraní. Do budoucna by bylo vhodné vytvořit například JavaScriptovou knihovnu obsahující funkce pro snadnou tvorbu uživatelský příjemného webového rozhraní nebo pro integraci dat do webových aplikací třetích stran.

Díky této práci byla vytvořena aplikace pro ukládání a verzování dat na vzdáleném serveru. V současné době je tato aplikace nasazena jako multimediální knihovna pro e-learningový systém, pro který ukládá obrázky, videa a zvukové soubory. V tomto nasazení pracuje bezchybně a spolehlivě.

Jestli se systém prosadí na poli verzovacích systémů bude záležet zejména na programových knihovnách, které by měly pro tento účel vzniknout, a na tom, jak bude systém vyhovovat vývojářům, kteří by ho případně chtěli používat.

SEZNAM POUŽITÉ LITERATURY

[1] COLLINS-SUSSMAN, Ben; Fitzpatrick, Brian W.; Pilato, C. Michael Version Control with Subversion. 1.5. vyd Stanford : Creative Commons, 2008. 26 s. 978-1-59682-169-9.
[2] ROCKING, Marc J.: Transactions on Software Engineering[online] 4. 12. 1975, , [cit. 20. 4. 2011], Dostupné z: http://basepath.com/aup/talks/SCCS-Slideshow.pdf.
[3] : Revision Control System[online] 20. 3. 2011, , [cit. 6. 5. 2011], Dostupné z: http://cs.wikipedia.org/wiki/Revision_Control_System.
[4] : GNU RCS[online] 13. 3. 2010, , [cit. 6. 5. 2011], Dostupné z: http://www.gnu.org/software/rcs/
[5] DELROSSI, Robert A.: Version control meets an expanding need[online] 20. 6. 1994, , [cit. 15. 3. 2011], Dostupné z: http://books.google.cz.
[6] KUČERA, František Distribuované verzovací systémy – úvod (2). 1.5. vyd : abclinuxu.cz, 2011. 26 s.
[7] PRICE, Derek Robert: CVS - Concurrent Versions System[online] 3. 12. 2006, 3. 12. 2006 , [cit. 2. 4. 2011], Dostupné z: http://www.nongnu.org/cvs/.
[8] KRČMÁŘ, Petr: Linux na desktopu roste, ale už pomaleji[online] 21. 9. 2009, , [cit. 11. 5. 2011], Dostupné z: http://www.root.cz/clanky/linux-na-desktopu-roste-ale-uz-pomaleji/.
[9] KRČMÁŘ, Petr: Exkluzivně: Linux je na 70 % českých serverů, Apache na 88 %[online] 21. 7. 2008, , [cit. 12. 5. 2011], Dostupné z: http://www.root.cz/clanky/exkluzivne-linux-je-na-70-serveru-apache-na-88/.
[10] PECINOVSKÝ, Rudolf Návrhové vzory. 1. vyd Praha : Computer Press, 2007. 107 s. 978-80-251-1582-4.

PŘÍLOHY

- Příloha 1 – slovníček pojmů
- CD obsahující:
  - elektronickou verzi této práce ve formátu PDF
  - zdrojové kódu verzovacího programu
  - zdrojové kódy klientské aplikace