Verzování souborů

rešerše bakalářské práce

Obsah

Obsah

Úvod

Důvody nového systému

Vzhledem k aktuálního 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 střihání 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, ale tyto služby již 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 ručně zaktualizovat (zpravidla přes FTP). Tato metoda je zdlouhavá a může vést k chybám, kdy daný člověk zapomene přenést na server některé soubory nebo naopak přenese soubory, které měly sloužit jako testovací data, nebo jako konfigurační soubory pro připojení k testovacímu serveru. Alternativou k této metodě jsou jednoduché skripty, které umožňují zaktualizovat server automaticky z lokálního adresáře, ale ani při této metodě 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 software 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. Pokud by čtenář některému pojmu užitému v textu práce nerozuměl, je v příloze uveden seznam pojmů se stručným vysvětlením, z nichž některé vybrané pojmy jsou vysvětleny podrobněji níže.

Důvody verzování souborů

Při vývoji větších projektů (stovky a více souborů) skupinou vývojářů, ale i 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 je školství a podobně.

Pojmy

Pro čtenáře neznajícího problematiku verzování jsou v 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.

Branch, fork

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 vývojářům jako v roce 2010, kdy se od projektu OpenOffice.org oddělil fork LibreOffice.

Merge

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
>       $result = "Zadal jste prázdný text"
petr@linux-3oy1:~/nuke/temp$
```

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

Jako změna se bere například odebrání nebo přidání řádku textu, modifikace řádku textu, ale zpravidla je 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).

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

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“.

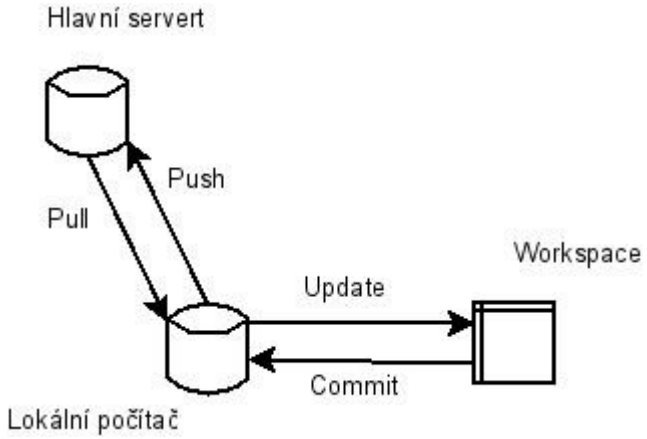
Rozdělení systémů podle modelu ukládání dat

Lokální

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

Model Klient-Server

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



Ilustrace 2: Model Klient-Server

Značnou nevýhodou tohoto přístupu je, pokud se server pravidelně nezálohuje, jsou v případě havárie ztraceny všechna data.

Oproti tomu ale v těchto systémech můžeme soubor „zamknout“ (model Lock – Edit – Unlock) a tím zabezpečíme že v průběhu našich editací nebude moci nikdo daný soubor změnit.

Distribuované

V nynější době je to nejběžnější model. 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 výce vývojářů na jednom souboru zároveň, mohou pracovat nezávisle na připojení k internetu a zálohování je provedeno s každou synchronizací

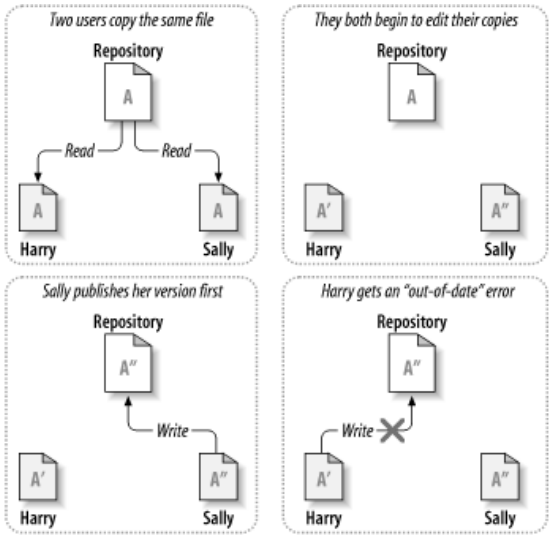
s uživatelem, který momentálně působí jako server.

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

Rozdělení systému podle modelu přístupu k datům

Model Zkopíruj-Modifikuj-Sluč

Nejčastěji používaný model pro verzování textově orientovaných dokumentů spočívající na filosofii slučování změn provedených ve stejnou dobu na stejném souboru.

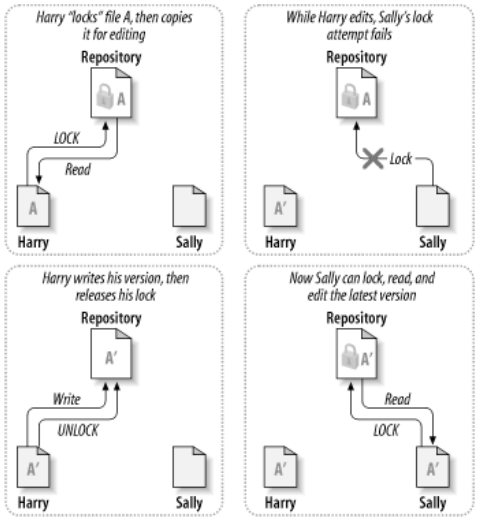


Obrázek 1: Práce v modelu Zkopíruj-Modifikuj-Sluč

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

Model Zamkni-Uprav-Odemkni

Tento model se zpravidla používá pro správu projektů s binárními soubory, jako je projekt z CAD systému a podobně (příklad z praxe je systém SAP + CAD UniGraphic).



Obrázek 2: Práce v module Zapkni-Uprav-Odemkni

Uživatel, který chce soubor editovat pošle na server požadavek na zamčení souboru. V případě, že mu systém vyhoví, je mu 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.

První verzovací systémy

SCCS – Source Code Controll systém

Tento systém byl vyvinut v roce 1972 v Bellových laboratořích pro korporaci IBM.

Používá velmi zvláštní architekturu pro ukládání programových jednotek (souborů), asice že veškeré verze a revize jsou uloženy v jednom souboru pomocí takzvaných delta balíčků (ukládají se pouze změny oproti předchozí verzi), které jsou řetězeny tak, jak byly postupně ukládány. Díky této architektuře není prakticky možné provést neautorizovanou změnu uloženého kódu.

Zajímavostí je, že původní soubor je reprezentován také jeho delta balíček, který je brán vzhledem k prázdnému souboru.

Po provedení každého commitu je zároveň zdokumentováno, kdo a kdy změnu provedl a zároveň je schopen provést diferenci 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 dostupný ve verzi 1.0, která je dostupná pro Windows a Unix like platformy. V době psaní této práce byla poslední změna stabilní verze zveřejněna na

začátku roku 2007.

RCS – Revision Controll System

Jedná se dnes již značně zastaralý software, který umí pracovat s textovými dokumenty a omezeně i s binárními soubory.

Ilustrace 3: Walter F. Tichy - tvůrce RCS

Byl vyvinut vyvinut Walterem F. Tichým na Univerzitě Purdue na počátku osmdesátých let minulého století.

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

Přestože umí vytvářet a slučovat vývojové větve souboru, práce s těmito funkcionalitami je velmi obtížná, a proto se většinou pracuje pouze s mainline souboru. Tyto problémy později vyřešil systém CVS, kterým se budu zabývat později.

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.

PVCS – Polytron Version Control system

Program 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.

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

Nejběžnější verzovací systémy

Git

Systém začal být vyvíjen Linusem Torvaldsem v roce 2005 (projekt byl započat ve stejné době jako Mercurical) 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, je možné ho 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).

Na serveru http://www.github.com se může kdokoliv registrovat a vytvořit se neomezený počet repozitářů.

Princip ukládání dat

Na rozdíl od většiny ostatních systémů ukládá Git vždy při každém commitu všechna data souboru znovu. To má za následek větší odolnost proti poškození repozitáře například 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 zakomprimovány a odeslány atomicky najednou, aby nedocházelo ke zbytečnému zatěžování sítě.

Běžné používání RCS Git

Pro založení repozitáře jsou k dispozici dva postupy - založení prázdného repozitáře a nebo stažení už existujícího repozitáře ze serveru. Pro ukázky v následujícím textu je 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 nainstalovaná SSH klíč, neboť většina serverů podporuje pro zápis pouze šifrované spojení, aby byla jasná identifikace uživatele, který zápis provedl (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 se serverem.

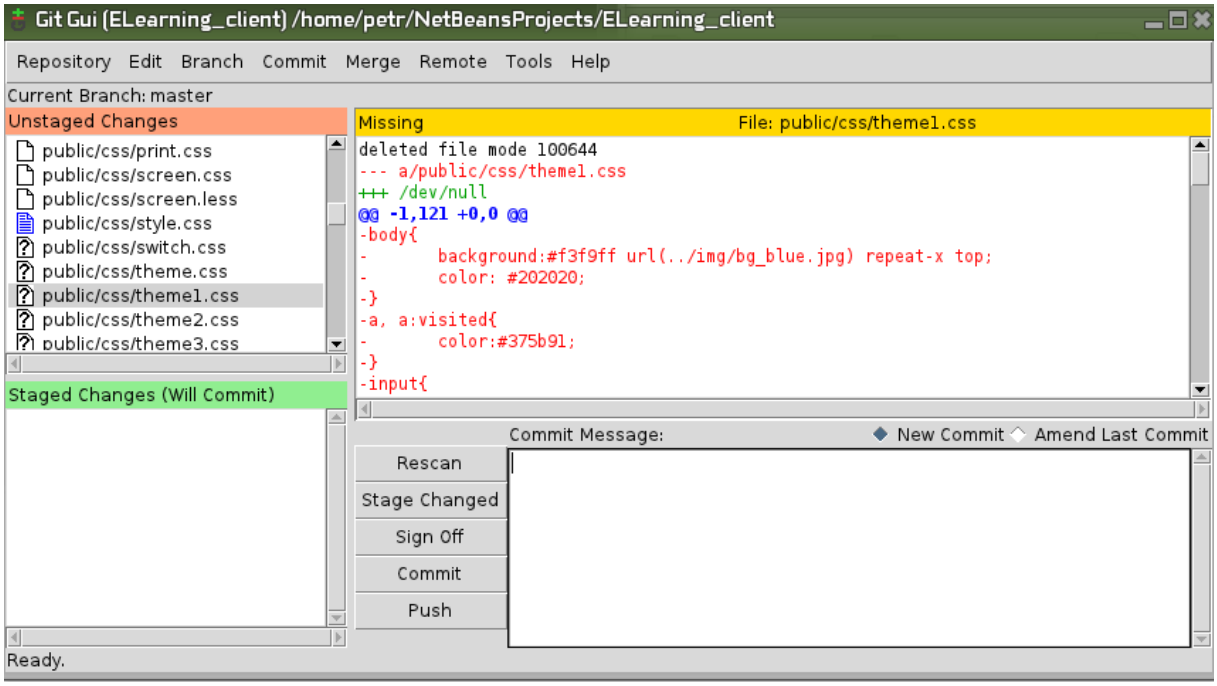
Po provedení změn v adresáři je nutné nové soubory přidat do repozotáře a staré odstranit. Slouží k tomu příkazy

```
git add <název souboru>
```

git remove <název souboru>

Nebo je možné použít integrované grafické rozhraní spustitelné příkazem

```
git gui
```



Obrázek 3: Integrované grafické rozhraní Git

kde je možné provádět všechny úkony v poměrně přehledné aplikaci bez nutnosti znát konzolové příkazy. Okno aplikace je rozděleno na čtyři části, kde v levé horní jsou změny, které ještě nebyly commitovány a nejsou označeny ke commitu, pod ní je seznam změněných souborů, které jsou označeny 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, tláčitka pro commit, push, atd.). V dalším textu bude věnována pozornost zpravidla pouze konzolovým příkazům.

Po provedení změn v pracovním adresáři se pomocí

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

uloží do lokálního repozitáře a následným

```
git push origin master
```

se změny odešlou na server. V případě potřeby, je možné origin nahradit názvem jiné větve, kam chceme změny uložit.

Předpokládejme, že v repozitáři došlo od provedení našeho posledního push ke změnám, proto je nutné provést synchronizaci lokální kopie repozotáře. Toho je možné docílit příkazem

```
git pull origin master
```

který 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 od posledního push jsme změnili některý soubor, vyvolá žádost o synchronizaci s hlavním repozitářem konflikt a je potřeba pro daný konflikt provést merge (ručně nebo použít nástroj k tomu určený).

CVS

Původně vzniklo jako nadstavbová kolekce skriptů pro RCS a 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 takzvaného „unreserved checkout“, 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 merge.

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 merge nad původní a importovanou branche.

SubVersion

První release byl uvolněn v roce 2000 a 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.

Princip ukládání dat

Data jsou ukládány v pomocí delt a v případě, že je potřeba, aby byl soubor na dvou místech v repozitáři, je na druhém místě vytvořen pouze odkaz na původní entitu, aby nedošlo ke zbytečné duplikaci dat.

Volba platformy

Vzhledem k mým několikaletým zkušenostem s vývojem webových aplikací jsem se rozhodl pro LAMP, tedy operační systém Linux, webový server Apache, databázový systém MySQL a

programovací jazyk PHP.

Operační systém Linux jsem vybral protože ho používám už několik let, jsem s ním poměrně dobře sžitý a pro vývoj webových aplikací je vhodný zejména svojí bezpečností, konfigurovatelností a vnitřní architekturou (hiearchie adresářů je odvozena od kořene, všechny entity jsou reprezentovány jako soubor).

Apache jsem zvolil pouze proto, že na dnešních webových serverech se jedná o standard a je tedy velice pravděpodobné, že aplikace bude fungovat na většině hostingů.

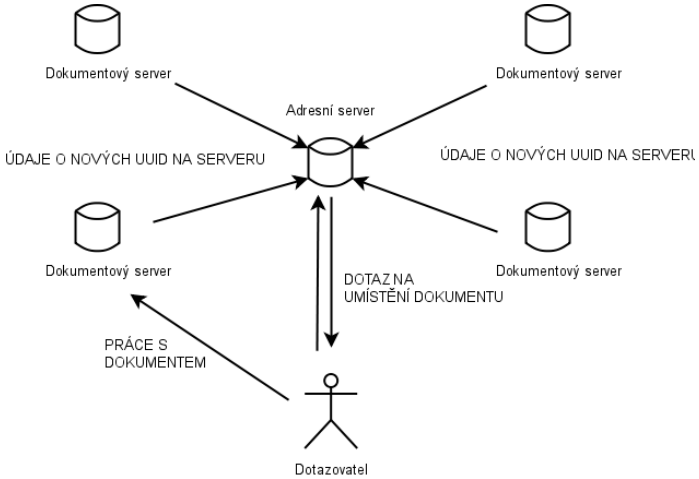
Jako databázový software jsem vybral MySQL, který je opět standardem a mám s ním bohaté zkušenosti.

S jazykem PHP pracuji již od verze 4, ve které byly poprvé implementovány objekty (velmi omezeně, použitelnou implementaci přinesla až verze 5) a vzhledem k tomu, že verze 5.3 podporuje anonymní funkce, jmenné prostory a další možnosti, není důvod proč se použití tohoto jazyka bránit. Jako framework použiji Zend, který vyvíjí přímo vývojáři jazyka a tím pádem je to jeden z nejschopnějších frameworků (poslední dobou mu ovšem začíná konkurovat české Nette).

Návrh systému

Ukládání dat

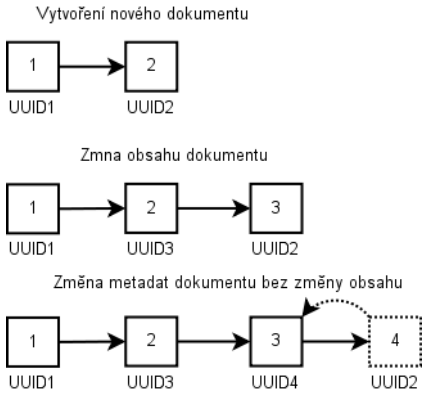
Vzhledem k pravděpodobné potřebě škálovatelnosti systému (rozložení úložiště na více serverů), nepoužívá program standardně používaný celočíselný inkrementovaný klíč, ale jednoznačné identifikátory dokumentu, označované jako UUID (univerzální unikátní identifikátor), založený na kombinaci UUID v.4 a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



Obrázek 4: Možné škálování dokumentového serveru

Do databáze jsou ukládány pouze metadata o dokumentu, jako je jméno, přístupová práva, vlastník a 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 se na něj bezproblémů odkazovat.



Obrázek 5: Schéma generování UUID dokumentu

V případě že je commitován dokument, který se obsahově nezměnil (ale bylo mu změněno například jméno, práva), vytvoří se místo nového obsahu pouze symbolický odkaz na původní verzi dokumentu (toto řešení má smysl zejména pro velké soubory, například 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 logování změn.

Adresářová struktura systému je ryze virtuální a uložená 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 je jednoduše vyfiltrovat pomocí SQL dotazu

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

Podrobný popis databázové struktury je uveden níže.

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 pouze pro administrátory a je navrženo pro obsažená všech možností systému na úkor přívětivosti uživatele, neboť není předpoklad častého přímého přístupu (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 ze 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>]
```

Kde významy jednotlivých částí adresy jsou následující

1. TYP OBJEKTU – sděluje informaci o typu objektu, se kterým chceme pracovat, například jestli chceme pracovat s dokumentem, adresářem, apod.
2. METODA – může obsahovat hodnoty PUT, GET, POST nebo DELETE a navíc případně další speciální metody definované typem objketu. 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 kterým che uživatel pracovat. V případě že identifikátor není uveden, systém vrací seznam možných objektů
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 pouze stavový kód v HTTP hlavičce. 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 idnetifikátor nového objektu)
- PUT – žádost o změnu existujícího objektu (vrací se indikace úspěchu, popřípadě aktualizovaná data)

Implementace programu

Implementace je možná několika způsoby. Jako nejpoužitelnější se jeví dva, kdy obě možnosti mají část adresářové struktury skrytou pomocí zanořené DocumentRoot v nastavení (virtuálního) serveru, a sice

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) za použití vhodného aplikačního jádra (Zend, Nette), kdy jednotlivé objekty jsou rozděleny do modulů, jejichž kontrolery

jsou přetěžovány.

Program byl implementován nejprve první metodou, která má značnou výhodu v tom, že není závislá na povolení RewriteEngine na serveru, protože soubory s kódem vykonávajícím práci s objekty jsou přímo ve veřejném adresáři. Tudíž, se znalostí struktury routovaných parametrů, je možné přistupovat k datům přímo přes tyto soubory.

Po dokončení základní části programu ale bylo zjištěno, že tato architektura je nevyhovující z dlouhodobého hlediska, z důvodu tvorby dalších částí programu, kdy by vznikal v hlavním adresáři chaos v souborech.

Z toho důvodu bylo jádro převedeno na architekturu Zend Aplikace. Aby bylo možné přistupovat k datům tří úrovňově (typ objektu, metoda, identifikátor), byly pro jednotlivé objekty vytvořeny samostatné moduly a pro každou metodu přístupu byl vytvořen samostatný kontroler.

Třetí úroveň navigace je realizována pomocí „magické“ metody __call, která je provedena vždy, pokud je volána neexistující metoda objektu.

Vzdálený přístup k datům

Implementace programu

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:

- Project – správa projektů a projektové dokumentace
- Message – komunikace mezi uživateli
- Note – tvorba poznámek k dokumentům

Modul Document

Tento modul zajišťuje manipulace s obsahy dokumentů v systému. Má na starosti generování unikátních UUID, správu řetězení dokumentů 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 zválštní metodu TRASH, která vrací výpis dokumentů právě přihlášeného uživatele, které nejsou přiřazeny v žádném z adresářů.

Metoda Trash

Je to nestadardní metoda (vzhledem k REST) a 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 6: Postup hledání neasociovaných dokumentů

Modul Directory

Jedná se o modul zabezpečující 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 vytvoří 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, sice problém ztráty asociativity nemá (za předpokladu, že při existenci jediného adresářového serveru nebudou další virtuální adresářové struktury vytvářeny na dokumentových serverech), ale všechny dotazy na informace o adresářích jsou zpracovány jediným serverem, a to může vést k jeho zvětšené zátěži.

Modul Group

Tento modul má nastarosti 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.

Modul User

Obsluhuje správu uživatelů a kromě základních kontrolerů obsahuje navíc jednoúčelové

kontrolery signin a signout, které slouží k autentizaci uživatele.

signin	Přihlásí uživatele
signout	Odhlásí uživatele

Tabulka 1: Dodatečné kontrolery modulu User

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

root	Systémový administrátor
guest	Neregistrovaný uživatel

Tabulka 2: 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, které vyjadřují rozsah oprávnění, které uživatel v systému má. Při vytvoření uživatele není standardně přidělena žádná role a může tedy pouze používat vlastní adresář a adresáře, kde má z hlediska oprávnění možnost zapisovat. Tyto oprávnění jsou daná systémem a není možné je nijak měnit. V základní verzi jsou tyto oprávnění dvě

- 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

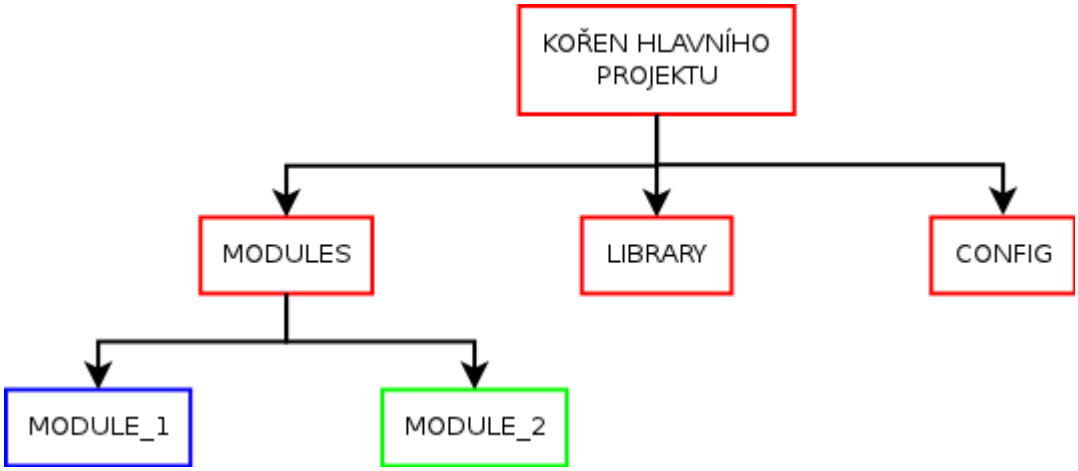
Super uživatel root ke správě informací tyto role nepotřebuje, protože je má zabudované přímo v programu systému.

Modul Project

Tento modul 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.

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, tedy projekt, 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.



Obrázek 7: Projekt se založenými podprojekty

Na obrázku 7 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 celý projekt nebo jeho část 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 konrolery

- COLABOURS – spravuje spolupracovníky na projektu
- STATUS – umožňuje měnit informace o aktuálním stavu projektu (funkčně podobný modulu Note, ale vztahuje se k celému projektu)

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

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 vždy při dotazu na poznámky u master dokumentu respektive dokument, u kterého žádné poznámky nejsou nalezeny, jsou zkontrolovány timestampy poznámek respektive poznámek o revizi novějšího dokumentu 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: může to být identifikační číslo poznámky, nebo UUID dokumentu s poznámkami (pro dotazy vedoucí k vrácení všech poznámek, a podobně).

Chování REST metod je následující:

- DELETE – smaže poznámku definovanou pomocí jejího identifikačního čísla
- GET – získá informace v závislosti na typu použitého identifikátoru
  - identifikační číslo poznímky – získá informace o poznámce
  - UUID dokumentu – získá výpis všech poznámek týkajících se dokumentu řazené o nejnovějšího po nejstarší
- POST – vytvoří novou poznámku, jako identifikátor je použito UUID cílového dokumentu
- 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) a přidávat poznámky může ke kterémukoliv dokumentu, do kterého má oprávnění zapisovat.

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

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.

Jako identifikátor objektu je použito identifikační číslo zprávy.

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)

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

Implementace datového modelu

Datový model je rozdělen na dvě části, a sice souborová a databázová.

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 nenaduplikuje, 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, aby se předešlo havarijnímu stavu, kdy při smazání jednoho odkazu by se následující řetězené odkazy staly neplatnými.

Databázová část modelu obstarává zbytek datové infrastruktury a je rozdělena do dvou sekcí podle účelu, ke kterému slouží. Sekce je možné poznat podle prefixu jména tabulky.

documents	Správa úložiště dat
users	Správa informací o uživatelích

Tabulka 3: Sekce databázových tabulek

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

Tabulky uživatelů

V těchto tabulkách 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 a informaci 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 užívtelů jsou uloženy v tabulce users_groups, která je provázána s tabulkou users pomocí many-to-many vztahu skrt tabulku users_has_many_groups.

Tabulky dokumentů

Základem je tabulka documents, která obsahuje obecná data o měnících se dokumentech, to

znamená

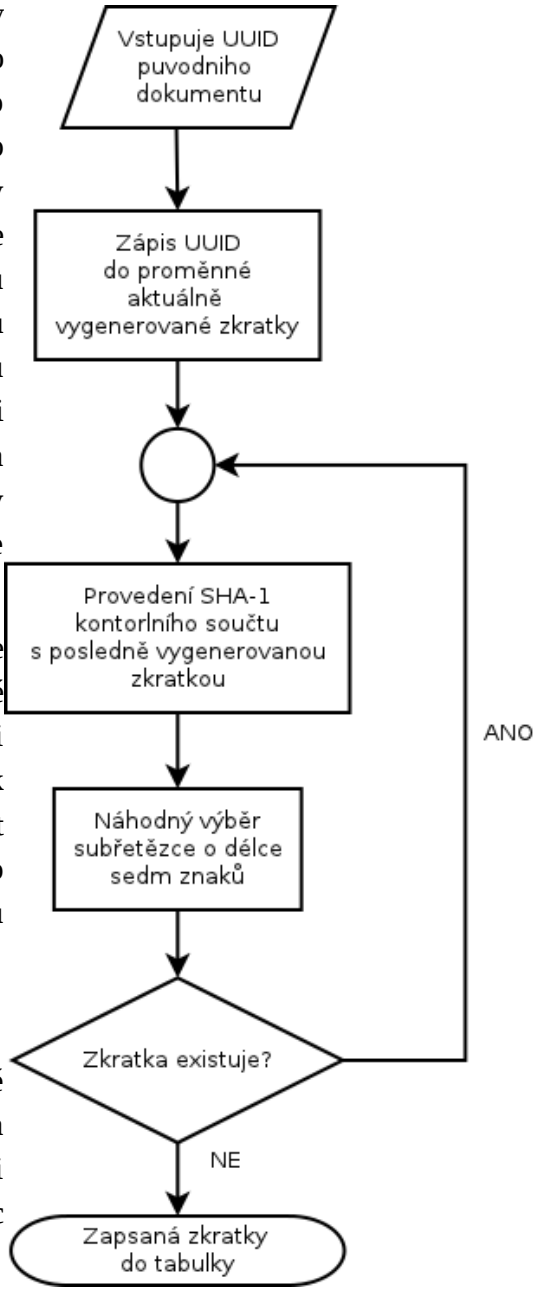
- UUID
- jméno dokumentu
- MIME typ dokumentu
- identifikační číslo uživatele, ke kterému dokument náleží
- identifikační číslo skupiny uživatelů dokumentu
- infomrace o přístupových právech, která jsou uložena v ASCII formátu
- časová značka (timestamp) vytvoření záznamu
- informaci, jestli se jedná o aktuální revizi dokumentu
- velikost dokumentu v bytech

S touto tabulkou úzce souvisí tabulky documents_history, uchovávající informace o postupných změnách dokuemntu. Každý řádek této tabulky obsahuje inkrementované identifikační číslo položky v historii, UUID o jednu revizi starší formy dokuemntu, 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 a 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 sedmi 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



Obrázek 8: Algoritmus generování zkratek

uchovávající zkratku je unikátní, a to zabezpečuje jedinečnost zkratky přímo na databázové úrovni. Zkratka je generována reprezentací tabulky v programu vždy při vložení nového záznamu pomocí algoritmu na obrázku 8.

Informace o virtuální adresářové struktuře jsou uloženy v tabulce documents_directories. Tato data nejsou revizovaná a proto není možné zpětně zjistit změny v obsahu adresáře v průběhu jeho existence. Tabulka uchovává obecné informace o adresáři (jméno, přístupová práva, vlastník a podobně), 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.

Příloha 1 – pojmy

- Branch, Fork – vývojová větev souboru, projektu, atd. (někdy se pojem Fork používá pro dočasný Branch, například v případě testovací verze)
- File lock – zamknutí souboru proti přepisu při checkout a odemknutí při checkin
- Checkout – vytvoření pracovní kopie souboru
- Checkin – odeslání pracovní kopie zpět do repozitáře a vytvoření revize
- Checksum, kontrolní součet – otisk dat sloužící zejména pro kontrolu neautorizovaných změn a poškození obsahu
- Mainline, Trunct – hlavní větev souboru
- Merge – sloučení více větví zpět do jedné
- Push – odeslání dat na hlavní server
- Pull – stažení aktuálních dat ze serveru
- Commit – zapsání změn do lokálního repozitáře
- Update – aktualizace pracovního adresáře z lokálního repozitáře
- RCS – anglická zkratka pro verzovací systémy (Revision Controll System) a také název jednoho z těchto programů
- Repository, Repozitář – skupina souborů a adresářů (projekt) zavedený v systému
- Revize, verze – označuje změnu v souboru nebo v jiné entitě
- Tag, label – označuje důležitou revizi ve vývoji, například stabilní verzi programu a podobně
- Workplace – pracovní adresář
- Diff – rozdíl mezi jednotlivými soubory a nebo také program, který rozdíly hledá