Sem vložte zadání Vaší práce.

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

Bakalářská práce

Replikace dat z cloudového systému přes rozhraní webových služeb

David Passler

Vedoucí práce: Ing. Pavel Müller

7. května 2015

Poděkování

Chtěl bych poděkovat vedoucímu práce panu Ing. Pavlu Müllerovi a Ing. Aleši Fišerovi za odborný dohled a pomoc při realizaci této bakalářské práce, Mgr. Radkovi Vařbuchtovi za profesionální přístup a spolupráci. Dále bych chtěl poděkovat své rodině a přátelům za podporu, kterou mi projevovali nejen při studiu, ale i při psaní této práce.

Prohlášení

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

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů, zejména skutečnost, že České vysoké učení technické v Praze má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 autorského zákona.

V Táboře dne 7. května 2015 .....

Odkaz na tuto práci

Passler, David. *Replikace dat z cloudového systému přes rozhraní webových služeb*. Bakalářská práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2015.

Abstrakt

Tato bakalářská práce se zabývá problémem replikace dat pomocí webových služeb do vlastní databáze za účelem jejich zálohy a otevírá možnost spouštění rozsáhlých reportů nad těmito daty. Problém replikace dat byl vyřešen pomocí aplikace napsané v programovacím jazyce Java, umístěné na vrstvě mezi cloudovým systémem a klientskými aplikacemi, které se dosud dotazovaly přímo na cloudový systém. Navrhované řešení v pravidelných definovaných intervalech stahuje data z cloudového systému a ukládá je do vlastní databáze, na kterou se následně dotazují klientské aplikace, čímž dochází ke zrychlení veškeré komunikace. Správnost svého návrhu jsem poté konkrétně ověřil při implementaci replikace dat z cloudového systému Salesforce. Na CD přiloženém k této práci lze nalézt administrátorskou a uživatelskou příručku. Dále lze na tomto CD nalézt i popis RESTového rozhraní aplikace.

Klíčová slova Salesforce mirror, replikace dat, middleware, Java aplikace, cloudový systém, databáze

Abstract

This bachelor thesis is managing the data replication from Salesforce to own local database for the purpose of their backup and opens the opportunity to

perform very long reports over this data. Data replication problem was solved by application written in Java programming language, which is set between Salesforce and client applications, which were querying the Salesforce itself. The suggested solution downloads data from Salesforce in certain defined moments and saves them to local database, which is an endpoint for client applications. This approach is beside fastering all communications. The correctness of my approach was validated on implementing data replication solution from Salesforce. On CD enclosed of this paper you can find administrator and user guide. You can find REST definition of application interface on CD enclosed to this thesis aswell.

Keywords Salesforce mirror, data replication, middleware, Java application, cloud system, database

Obsah

Úvod **1**

1 Cíl práce **3**

2 Možné problémy při replikaci dat přes webové služby **5**
2.1 Nedostupnost služby 5
2.2 Synchronizace času 5
2.3 Zabezpečení komunikace 6
2.4 Nesprávný formát dat 7
2.5 Velké objemy dat 7

3 Analýza **9**
3.1 Salesforce 9
3.2 Funkční požadavky 10
3.3 Nefunkční požadavky 11
3.4 Analýza konkurence 11
3.5 Zhodnocení a výběr řešení 12

4 Návrh řešení **15**
4.1 Salesforce 15
4.2 Uživatelské role 16
4.3 Výběr vhodných technologií 16
4.4 Architektura 19
4.5 Databáze 20
4.6 Activity log 20
4.7 Strategie mazání záznamů 20
4.8 Proces replikace 21
4.9 Activity diagram vytváření synchronizace 24

5 Implementace zvoleného řešení **25**

5.1 Implementace replikačního algoritmu 25
5.2 Změna Spring properties za běhu aplikace 28
5.3 Šifrování citlivých informací 30
5.4 Opakování dotazů na Salesforce v případě problémů 31
5.5 REST API 33

6 Testování a dokumentace **37**
6.1 Testování 37
6.2 Dokumentace 38

Závěr **41**

Literatura **43**

A Seznam použitých zkratek **45**

B Obsah přiloženého CD **47**

C Ukázka aplikace **49**

Seznam obrázků

2.1 VPN Tunel [17] 6

4.1 UC diagram 17
4.2 UC - Přehled uživatelských rolí 18
4.3 Architektura aplikace 19
4.4 Proces replikace 22
4.5 Aktivity diagram vytvoření synchronizace 24

C.1 Počet záznamů dle synchronizace 49
C.2 Detail synchronizace 50
C.3 Nastavení přihlášení do Salesforce 51
C.4 Nastavení replikační databáze 51
C.5 Editace uživatele a přiřazování rolí 52
C.6 Volba barevného schématu aplikace 52

Úvod

V dnešní době mají firmy mnoho dat a je pro ně nevýhodné všechna data ukládat v lokálních databázích uvnitř firemní infrastruktury. Dat je mnoho a kladlo by to vysoké nároky na úložiště. Dalším, neméně důležitým, důvodem k přestěhování dat do cloudu je přenesení odpovědnosti za škodu na poskytovatele cloudové služby. Proto se firmy stále více a více uchylují k ukládání dat do cloudových systémů [3]. Na druhou stranu je ale požadavek, aby byla tato data dostupná velmi rychle a pokud možno 24 hodin denně, 7 dní v týdnu. Dále firmy musí být připraveny se vyrovnat se situací, kdy by například cloudová služba ze dne na den ukončila činnost bez možnosti získání dat v ní uložených. Z těchto důvodů je vhodné použití aplikace, která všechna (nebo jen část dat) dokáže zreplikovat do vlastních databází. Dále je také nutné zmínit, že některá cloudová řešení mohou mít omezení na objem přenesených dat za určitou časovou jednotku (například měsíčně, ...). Pokud by byl tento limit rychle vyčerpán, firmy se nedostanou ke svým datům a pokud si nechtějí doplatit využití cloudové služby nad limit svého aktuálního tarifu, tak mohou v důsledku dokonce přijít o zákazníky. Právě z těchto důvodů jsem se rozhodl takovouto aplikaci vyvinout.

Motivace

Toto téma se mi naskytlo jako možnost realizace nového firemního projektu, kde si vyzkouším pod vedením zkušeného vývojáře tvorbu rozsáhlejšího projektu s obrovským praktickým využitím.

Kapitola 1

Cíl práce

Moje bakalářská práci si klade za cíl popsat možné problémy při replikaci dat přes webové služby, navrhnout řešení těchto problémů a správnost těchto navržených řešení demonstrovat na aplikaci pro replikaci dat z cloudového systému Salesforce. Bude se jednat o aplikaci, která bude v pravidelných, uživatelem definovaných intervalech, stahovat data z cloudového systému a tato data ukládat do vlastní, předem vybrané, databáze. Výstupem teoretické části mé bakalářské práce bude přehled možných problémů při synchronizaci dat přes webové služby a návrh jejich řešení, výstupem praktické části pak bude funkční aplikace, dostupná jako webová aplikace přes webový prohlížeč, replikující data z cloudového systému Salesforce do vlastní databáze.

Kapitola 2

Možné problémy při replikaci dat přes webové služby

2.1 Nedostupnost služby

2.1.1 Popis

Čas od času se z důvodu nutné odstávky nebo útoku může stát, že služba není dostupná. I když mnoho cloudových systémů garantuje dostupnosti 99% a více ([12], [13], [14]) vždy se mohou přihodit nečekané události, které mohou způsobit výpadek samotné služby. Mezi tyto nečekané události patří například přírodní katastrofy, které mohou vyřadit servery (jako tomu bylo v případě Hurikánu Sandy, který postihl východní pobřeží USA [1]).
Další možnost, kdy se producent dat může tvářit jako nedostupný, je při přetížení. Jedná se o stav, kdy služba již nedokáže vyřizovat požadavky na ni mířící z důvodu zahlcení, ať už úmyslného či neúmyslného, jinými požadavky.

2.1.2 Řešení

Pokud služba neodpovídá, je možné požadavek několikrát zopakovat a v případě, že je pouze zahlcena velkým množstvím požadavků (pokud zrovna není například pod nějakým útokem jako v případě útoku na Amazon, Salesforce a Petco v roce 2009 [2]) služba odpoví.

2.2 Synchronizace času

2.2.1 Popis

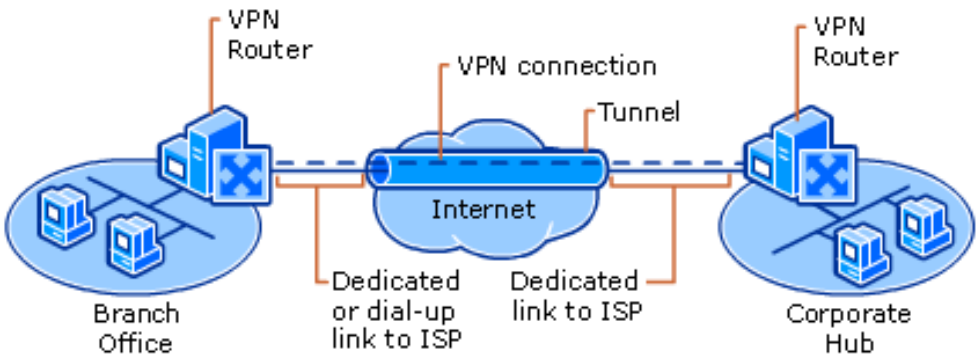
Problém nesouhlasícího času mezi serverem a lokálním počítačem může způsobit synchronizaci špatného počtu záznamů. Pokud si představíme situaci, kdy na lokálním počítači máme čas 17:00 a na serveru odkud data stahujeme

je teprve 16:55. Až replikace skončí, nastaví se datum a čas konce poslední replikace na 17:00 hodin. Mezi tím na serveru někdo vytvoří záznam s názvem *record1* (pro server v 16:58, což je 17:03 na lokálním počítači). Až se provede další replikace z lokálního počítače (například v 17:30), tak se vyžádají záznamy mezi 17:00 a 17:30, ovšem záznam *record1* mezi nimi nebude, tudíž nebude nikdy replikován.

2.2.2 Řešení

Řešení tohoto problému je nasnadě, jako referenční časy pro začátek a konec replikace je nutné ukládat si datum a čas vygenerovaný serverem. Opačný případ tohoto problému, kdy na lokálním počítači je čas urychlený oproti serveru nevadí, jelikož záznam je replikován při pozdější synchronizaci, takže se neztratí, pouze bude stažen o něco později.

2.3 Zabezpečení komunikace



Obrázek 2.1: VPN Tunel [17]

2.3.1 Popis

Při putování dat otevřenými kanály pomocí internetu jsou tato data předávána mezi mnoha servery a sítěmi. Pokud jsou data v otevřené formě, není tak těžké pro zkušenějšího uživatele se znalostmi počítačových sítí tato data odposlechnout a získat.

2.3.2 Řešení

Pro řešení tohoto problému je nutná součinnost více stran. Server a klient mezi sebou mohou vytvořit tzv. privátní tunel (například pomocí VPN, viz obrázek 2.1) a nebo komunikovat přes zabezpečené protokoly (HTTPS). Tyto možnosti

zprostředkovávají vytvoření šifrovaného kanálu, tudíž potenciální útočník musí nejen odposlechnout komunikaci, ale ještě přenášená data rozšifrovat, což je v drtivé většině případů nemožné (při použití dostatečně silné šifry).

2.4 Nesprávný formát dat

2.4.1 Popis

Tento problém nastává typicky chybou poskytovatele dat. Jedná se o stav, kdy poskytovatel dat má data uložena ve špatném formátu z důvodu nějaké chyby nebo vnějšího zásahu. Data mohou být poškozena nebo část dat chybět. Poté nám poskytovatel automaticky vygeneruje připravenou dávku dat, ovšem již se nedozví o tom, že v těchto datech je chyba. A v klientské aplikaci, která na tuto možnost není připravena, může tato chyba způsobit fatální selhání.

2.4.2 Řešení

Ověřovat data již při jejich zpracování. Konzument dat musí být připraven na možnost, že data nejsou validní a musí se umět s touto situací vyrovnat.

2.5 Velké objemy dat

2.5.1 Popis

Pro získávání dat přes webové služby existují v zásadě tři možné přístupy, *synchronní volání*, *asynchronní volání* a *notifikace od producenta o změně*. Synchronní volání, které je blokující svou činnost provádí tak, že cílového producenta dat požádá o určitá data a čeká, než je producent zpracuje a odešle konzumentovi. Tato žádost je blokující a dokud producent nevygeneruje data pro konzumenta, nemůže konzument požádat o jiná data.

2.5.2 Řešení

Řešení nabízí použití asynchronního volání nebo získáváním notifikací od producenta o změně. Asynchronní volání probíhá tak, že konzument požádá producenta o data a ukončí spojení. Poté se opakovaně ptá, jestli producent už data zpracoval a pokud producent odpoví, že ano, pak konzument začne data stahovat. Pokud je to možné, tak druhým řešením tohoto problému je zaregistrovat se u producenta dat tak, aby producent vždy při změně dat nějakým způsobem notifikoval konzumenta o změně.

Kapitola 3

Analýza

3.1 Salesforce

Salesforce je tzv. all-in-one CRM řešení pro firmy. Tento cloudový systém umožňuje řízení vztahu se zákazníky. Firmy si do tohoto systému ukládají informace o všech svých zákaznících, dodavatelích, dále lze vkládat obchodní příležitosti, vytvářet smlouvy, pořizovat objednávky a tak dále. Platforma Salesforce se skládá z komponent Sales, Service, Marketing, Community, Analytics a dalších platform a aplikací.

3.1.1 Sales

Komponenta *Sales* zabezpečuje základní (tzv. core) funkčnost celého portálu Salesforce. Provádí se zde management zákazníků, vytváření obchodních příležitostí, tvorba nových nabídek apod. Možností je též dokoupení předpřipravené celosvětová databáze subjektů, včetně hierarchie s názvem *Data.com*.

3.1.2 Service

Komponenta *Service* slouží pro správu a řízení podnikového IT. Zde mohou pracovníci podniku vytvářet nové incidenty (pokud jim přestane fungovat například tiskárna), dále mohou generovat požadavky na nákup vybavení, management má možnost tyto nabídky schvalovat, odmítat atd. Tato komponenta není určena pro menší podniky z důvodu její komplexnosti a ceny, proto Salesforce přichystal i komponentu s názvem *Desk.com*, která je určena právě menším podnikům s méně funkcemi, ale za nižší cenu.

3.1.3 Marketing

Komponenta *Marketing* slouží firmě pro správu kampaní. Je zde možné vytvářet mailové kampaně, SMS kampaně a dokonce je zde i možnost propojení s mobilní aplikací, kdy zákazník na určitém místě na světě otevře v mobilním

zařízení aplikaci a podle jeho polohy může dostávat odlišný typ kampaně. Jak je asi zřejmé z popisu, tato komponenta též není pro malé firmy, ale je zde připravena komponenta *Pardot* s méně funkcemi za přiznivější cenu.

3.1.4 Community

Komponenta sloužící ke tvorbě veřejných portálů pro zákazníky. Tyto portály mohou sloužit například jako zákaznická zpětná vazba [18]. Součástí této komponenty je i malá sociální síť uvnitř samotného Salesforce s názvem *Chatter*. Všichni uživatelé si na *Chatteru* vytvářejí své osobní profily, mohou spolu chatovat, sdílet si navzájem soubory apod.

3.1.5 Analytics

Tato komponenta je určena pro mobilní zařízení. Umožňuje zobrazení všech možných grafů pro vyhodnocování marketingových kampaní nebo prosperity firmy. Vzhledem k tomu, že je tato komponenta určena pro mobilní zařízení, majitel firmy si může kdekoli na svém chytrém telefonu zobrazit jakákoli data a vyhodnocovat nebo porovnávat bez nutnosti připojení do Salesforce a použití jiných BI nástrojů.

3.1.6 Platformy

Poslední komponenta nabízí vlastní vývojové platformy pro Salesforce. *Force.com* slouží pro vývoj aplikací přímo v cloudovém systému Salesforce s použitím vlastního programovacího jazyka zvaného *Apex*. *Salesforce1* je platforma pro vývoj mobilních aplikací, které je možné následně propojit se Salesforce a mít tak informace z cloudové instance k dispozici i na vlastním chytrém zařízení. *Heroku* je platforma, která nabízí možnost jakoukoli vlastní aplikaci napsanou v některém z programovacích jazyků (Ruby on Rails, Node.js, Python, Java aj.) spustit přímo v cloudovém systému tak, aby byla vždy dostupná.

3.2 Funkční požadavky

Aplikace bude podporovat následující funkcionality:

- Aplikace bude umět synchronizovat tabulky ze Salesforce do vlastní vybrané databáze (jak manuální spuštění, tak naplánovaný průběh)
- Aplikace bude fungovat na základě vložené licence, kterou lze změnit v nastavení aplikace
- Při prvním startu aplikace se zobrazí průvodce prvotním nastavením základních funkcí

- Aplikace bude obsahovat nastavení, kde bude možnost měnit přístupové údaje do Salesforce, replikační databáze a vytváření uživatelských účtů
- Řízení přístupu k určitým funkcím v aplikaci bude realizováno pomocí uživatelských rolí
- Každý uživatel bude moci mít více než jednu uživatelskou roli
- Nové uživatele vytvářet a jejich editaci smí provádět pouze administrátor
- Přidělování a odebírání rolí uživatelům je možné z administrátorského účtu
- V aplikaci si bude uživatel moci vytvořit notifikace emailem o stavu určitých synchronizací
- Aplikace bude umožňovat ovládání přes REST API

3.3 Nefunkční požadavky

- Aplikace bude dostupná přes webové rozhraní jako webová aplikace bez nutnosti instalace
- Aplikace poběží na operačních systémech Microsoft Windows XP a novějších, Linux
- K aplikaci bude dodána i administrátorská, uživatelská dokumentace a popis REST API
- Pokrytí zdrojových kódů testy bude minimálně 40%

3.4 Analýza konkurence

V současné době existuje jen pár řešení, která seriózně nabízí možnost efektivní replikace.

3.4.1 SnapLogic

Jedním z takových řešení je SnapLogic. Snaplogic je iPaaS. Tento software je určen pro manipulaci s daty z různých SaaS systémů (Salesforce, ServiceNow, ...) dále také umožňuje brát data například ze sociálních sítí (Twitter, FB, LinkedIN), dělat různé manipulace s takto získanými daty (sjednocení, seřazení, filtrování, ...) a dále je například uložit do nějaké lokální databáze. Toto workflow (v terminologii Snaplogic pipeline) se definuje pomocí jednoduchého HTML 5 Designeru, který je dostupný po přihlášení do aplikace přímo z webového prohlížeče. Dále si všechna tato workflow může uživatel zobrazit

na interaktivním dashboardu. Software umožňuje integrovat přes 160 systémů (Snaplogic je nazývá Snaps). SnapLogic též nabízí mobilního klienta, ovšem pouze pro iOS.[8]

3.4.2 Informatica Cloud

Dalším zajímavým řešením je Informatica Cloud. V podstatě to samé řešení jako je SnapLogic probraný výše. InformaticaCloud umožňuje stahování a integraci dat z různých systémů. V základní verzi (Professional) umožňuje pouze synchronizaci dat. Verze Basic k těmto možnostem přidává možnost sdílení znovupoužitelných integračních schémat. Za příplatek už lze ve verzi Basic využívat možnost real-time integrace. Ve verzi Advanced máme možnost vytváření REST a SOAP API, pokročilé administrace (delegovaná administrace, 2 stupňové ověřování uživatelů ...) a dále také tato verze nabízí Sandbox. Nejvyšší verze Premium přidává real-time synchronizaci už v ceně.[4]

3.4.3 MyDBSync

Posledním relevantním řešením je MyDBSync. Toto řešení umí synchronizovat více systémů, z nichž je pro účely této práce podstatný pouze Salesforce. Toto řešení je svou funkčností nejbližší mnou navrhované aplikaci. Umožňuje automatické vytvoření Salesforce Object schématu, replikace po dávkách nebo real-time a vytváření a upravování záznamů z databázových tabulek. Výběr jednotlivých funkčností následuje (úplný výčet dostupný na webových stránkách [6]):

- Obousměrná synchronizace mezi Salesforce instancí a lokální databází
- Automatická úprava databázového schématu bez nutnosti opětovného vytvoření celého schématu
- Windows, Linux, Solaris podpora
- Batch a shell aplikace pro integraci
- Intuitivní UI
- Úplné a inkrementální stahování Salesforce dat
- Podpora SQL Server, Oracle, DB2, MySQL, PostgreSQL, Apache Cassandra databází

3.5 Zhodnocení a výběr řešení

Základním požadavkem na aplikaci je možnost synchronizace dat ze Salesforce do lokální databáze a to jak úplná synchronizace všech dat, tak i synchronizace

pouze vybraných dat. Dále je důležité, aby aplikace umožňovala tzv. inkrementální obnovování dat, což znamená, že synchronizovat se budou jen ta data, která se od poslední synchronizace nějak změnila. Vzhledem k tomu, že všechna výše zmíněná řešení tyto požadavky splňují je nutné zvážit další okolnosti, které budou více probrány dále. Posledním řešením je vytvořit aplikaci podle návrhu v jednom z mnoha programovacích jazyků úplně od začátku.
První a druhé výše zmíněné řešení je pouze pro účely synchronizace dat do vlastní databáze z jednoho CRM systému (Salesforce) velmi komplexní a nebylo by za jejich cenu moc využito.
MyDBSync se jeví jako optimální pro toto použití a proto jsem se rozhodl ho prozkoumat podrobněji. Instalace zkušební verze proběhla jednoduše, následná konfigurace už ale tak jednoduchá není. Aplikace nebyla intuitivní na ovládání, měla složité nastavení, neobsahovala instalační příručku, ani návod pro používání.
Z těchto důvodů jsem se rozhodl aplikaci vytvořit od začátku podle požadavků specifikovaných zadavatelem.

Kapitola 4

Návrh řešení

4.1 Salesforce

Salesforce poskytuje pro přístup přes webové služby knihovnu WSC pomocí svého github účtu forcedotcom [9]. Tato knihovna je předgenerována z WSDL souboru, který je pro většinu Salesforce organizací stejný. V ostatních případech je možné pomocí návodu vygenerovat knihovnu přímo z WSDL. Tato knihovna umožňuje provádět SOAP požadavky a požadavky založené na REST velmi jednoduše pomocí volání metod přímo v programovacím jazyce Java. K volání lze využít SOAP API a Bulk API.

4.1.1 SOAP vs Bulk API

Knihovna WSC poskytuje dvě možnosti, jakým způsobem provádět požadavky na Salesforce. Hlavní rozdíl mezi těmito způsoby je v jejich limitech a ve způsobu dotazování, což je potřebněji popsáno dále.

4.1.1.1 SOAP API

Jedná se o API využívající synchronní volání. Vhodné použití pro SOAP API je pro volání tzv. CRUD operací nad objekty (vytváření nových Účtů, editace stávajících Objednávek aj.) [10].

4.1.1.2 Bulk API

Toto API používá asynchronní volání, to znamená, že se vytvoří požadavek na zpracování a poté je nutné se Salesforce dotazovat, zda už má požadavek vyřízen. Pokud je požadavek vyřízen, stáhnu výsledek a zpracuji. Výsledek je poskytnut ze Salesforce ve formátu CSV souborů. Popsaný způsob získávání záznamů je složitější než v případě *SOAP API*, ale dokáže stáhnout mnohem více dat. Bulk API je založeno na podobných typech požadavků jako REST a je optimalizované pro stahování velkého množství dat.[11]

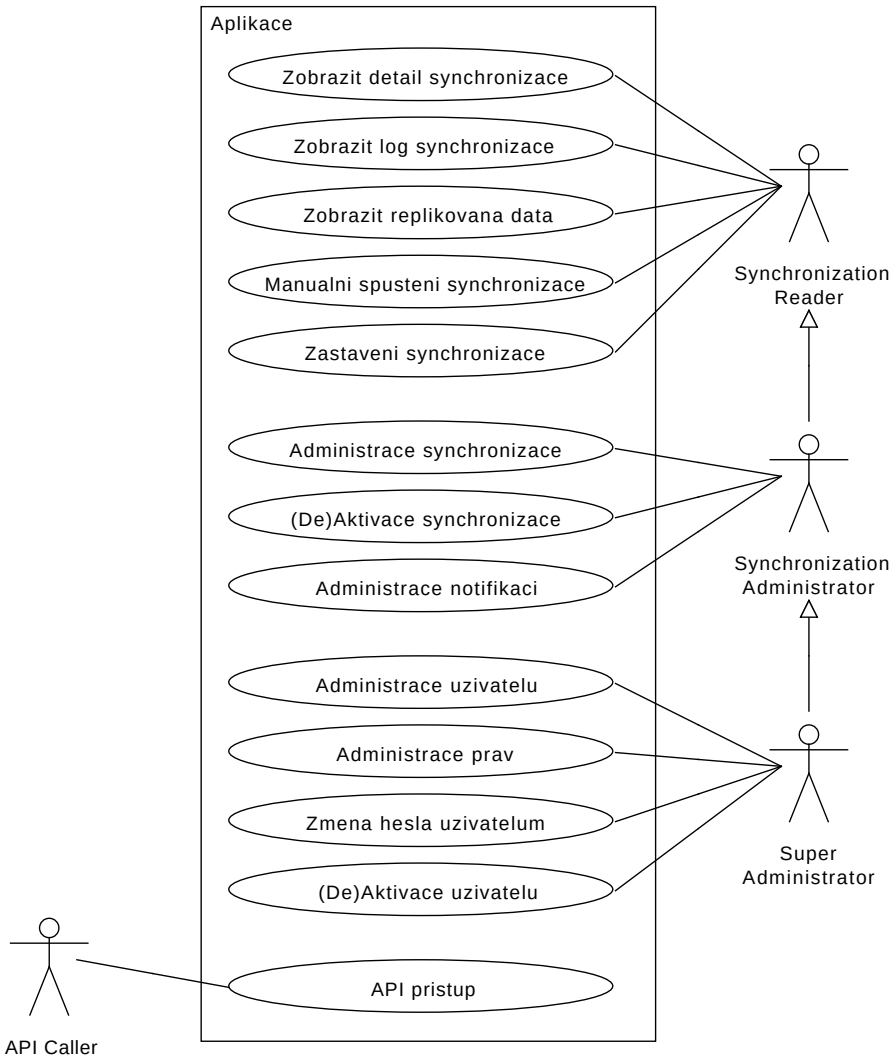
4.2 Uživatelské role

Aplikace umožňuje vytváření uživatelů a přiřazování rolí v relaci *1:N*. Seznam dostupných rolí:

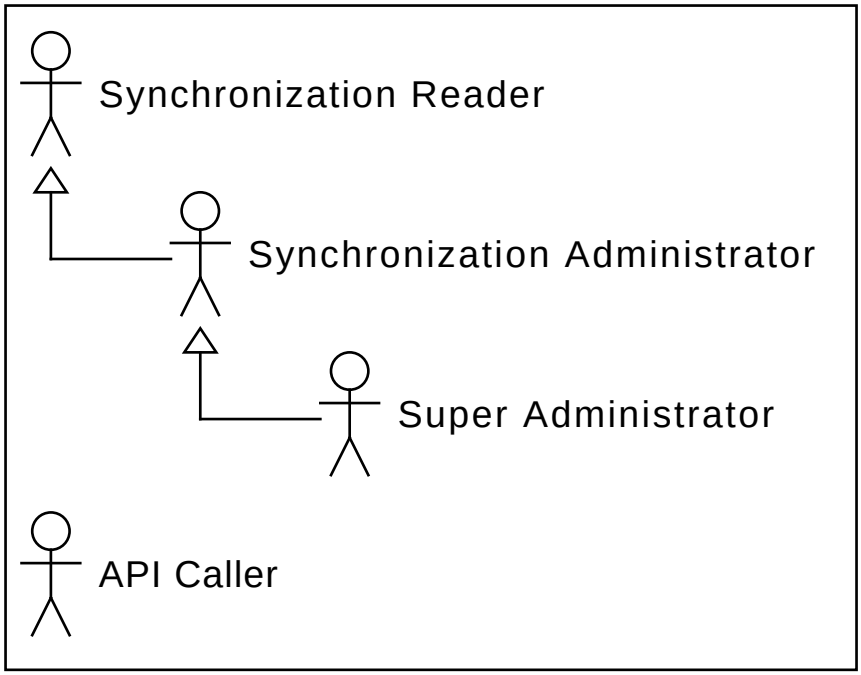
- Synchronization Reader - umožňuje zobrazení synchronizací, dat a také spuštění synchronizace
  - Zobrazit detail synchronizace
  - Zobrazení logu synchronizací
  - Zobrazení replikovaných dat
  - Manuální spuštění synchronizace
  - Zastavení právě probíhající synchronizace
- Synchronization Administrator - administrace synchronizací, obsahuje všechna oprávnění jako *Synchronization Reader* plus:
  - Vytváření, editace a smazání synchronizace
  - Aktivace a deaktivace synchronizace
  - Vytváření, editace a mazání emailových notifikací
- Super Administrator - přístup k administraci uživatelů, obsahuje všechna oprávnění jako *Synchronization Administrator* a k tomu navíc:
  - Vytváření, editace a mazání uživatelů
  - Editace práv uživatelů
  - Změna hesla uživatelům
  - Aktivace a deaktivace uživatelů
- API Caller - přístup k aplikaci přes REST API 5.5

4.3 Výběr vhodných technologií

Aplikaci jsem se rozhodl vytvořit v programovacím jazyce Java. Dále jsme se po konzultaci s vedoucím práce dohodli na využití frameworků Spring a Hibernate (pro ukládání dat na persistentní médium). Jako logovací framework je použit Log4J a jako sestavovací nástroj Maven.



Obrázek 4.1: UC diagram



Obrázek 4.2: UC - Přehled uživatelských rolí

4.3.1 Spring

Nejdůležitějším frameworkem použitým v aplikaci je framework Spring. Samotný framework se skládá z mnoha balíčků (projektů), ze kterých budu používat Spring Core a Spring Security. *Spring Core* [19] bude využit k řízení závislostí mezi třídami, jeho modul *Spring AOP* pro vytvoření aspektu, který bude kontrolovat licenci při každém pokusu o spuštění synchronizace a modul *Spring JDBC*, který mi poskytne implementaci základních funkcí (vytvoření *Statementu*, operace s *ResultSetem*, ...) v JDBC pro ukládání dat do replikační databáze. Dalším modulem, použitým ze Spring Core bude *Spring WebMVC*, který slouží jako implementace návrhového vzoru MVC pro webové aplikace. Pomáhá při rozlišování různých požadavků jejich směrováním na správné Controllery a také zajišťuje zobrazování správných Views. *Spring Security* [20] bude zajišťovat autorizaci (přihlašování do aplikace) a autentizaci (oprávnění vykonávat různé akce) uživatelů. Tento balíček mi poskytne nástroj pro zabezpečení aplikace před neoprávněným použitím.

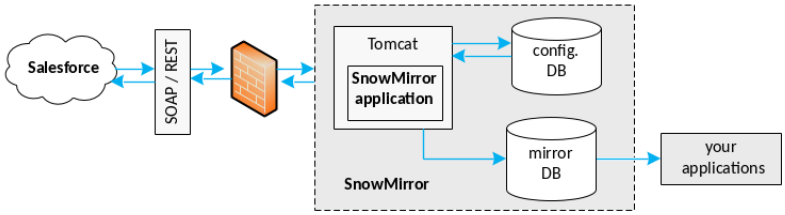
4.3.2 Verzování

Pro zajištění verzování zdrojových kódů je použit Subversion. Skutečnost, že máme zdrojové kódy aplikace pod sledováním verzovacího systému, nám umožňuje uchovávat verze veškerých souborů a případně se později vrátit k předchozí verzi.

4.3.3 Prostředí

Pro zajištění stability výsledné aplikace existují 3 prostředí. *Vývojové prostředí*, které má každý z vývojářů na svém lokálním počítači a spouští se zde jednotkové testy. *Integrační prostředí*, kde probíhá integrační testování a je zde nasazen CI Bamboo, který po úspěšném sestavení a otestování nasazuje war soubor na *produkční prostředí*. Zde se nad aplikací ještě spustí Selenium testy. Existence těchto prostředí umožňuje rychlejší vývoj, protože Selenium testy mohou běžet velmi dlouhou dobu a je nepřijatelné, aby pokaždé musel vývojář čekat na proběhnutí těchto testů. Ovšem je nutné zajistit, aby při selhání jakýchkoli testů se nesestavila výsledná aplikace, která by se potom mohla publikovat. Samotné Bamboo zařídí při úspěšném provedení integračních testů nasazení do produkčního prostředí a spuštění Selenium testů. Tímto workflow je zajištěna stabilita produkčního prostředí, jinými slovy se nikdy nestane to, že by výsledná aplikace nebyla sestavitelná nebo by byla narušena její základní funkčnost.

4.4 Architektura



Obrázek 4.3: Architektura aplikace

Jak vidno z obrázku, aplikace pomocí REST či SOAP webových služeb stahuje data ze Salesforce a ukládá je do tzv. mirror DB (replikační databáze). Samotná aplikace má ještě jednu databázi (na obrázku jako config. DB), která obsahuje konfigurační informace (přístup k Salesforce, nastavení samotné aplikace apod.). Klientské aplikace (na obrázku jako your applications) se připojují přímo na replikační (mirror) databázi, nad kterou provádí

veškeré (mnohdy i složité) operace. Aplikace bude typicky nasazena uvnitř firemní infrasktruktury, takže dotazy nad touto databází budou velmi rychlé.

4.5 Databáze

Aplikace používá ke svému fungování 2 oddělené databáze (konfigurační a replikační). U obou typů databází je podpora MySQL, PostgreSQL, MSSQL, Oracle a H2.

4.5.1 Konfigurační databáze

V této databázi jsou uložena veškerá nastavení samotné aplikace. Jedná se především o přístup k replikační databázi, přístup k Salesforce, nastavení uživatelů aplikace, nastavení upozornění apod. Samozřejmostí je, že všechny citlivé informace v konfigurační databázi jako přístupová hesla jsou šifrována algoritmem AES-256.

4.5.2 Replikační databáze

Do této databáze jsou replikována samotná data ze Salesforce. Každá synchronizace v aplikaci má vlastní tabulku v replikační databázi. Sloupce v tabulce této databáze si uživatel určuje sám při vytváření synchronizace. Jediné dva sloupce, které jsou přidány manuálně bez zásahu uživatele jsou *MirrorCreatedDate* a *MirrorLastModifiedDate* jenž určují, kdy byl záznam vytvořen (resp. naposledy upraven). Prefix *Mirror* u názvu sloupců je samozřejmě konfigurovatelný.

4.6 Activity log

Jedná se o log prováděných činností v rámci dané synchronizace, tudíž každá synchronizace má právě jeden vlastní Activity log. Obsahuje všechny důležité informace o provedené synchronizaci včetně případných chyb s návrhem řešení.

4.7 Strategie mazání záznamů

Aplikace nabízí vícero strategií mazání neplatných záznamů z replikační databáze.

- Audit
- Difference
- Truncate

- None

4.7.1 Audit

Nejběžnější strategie mazání. Vyžaduje, aby daný objekt v Salesforce byl auditovatelný (tj. obsahoval sloupce *CreatedDate* a *LastModifiedDate*). Bez těchto údajů není možné audit strategii použít a není ani nabízena při vytváření (resp. editaci) synchronizace.

4.7.2 Difference

Tato strategie funguje na principu získání všech ID záznamů z replikační databáze pro daný objekt a od těchto záznamů se odečtou ID všech záznamů, které se nacházejí v Salesforce pro daný objekt. Výsledná množina potom obsahuje ty záznamy, které se již nenachází v Salesforce a je třeba je odstranit z naší replikační databáze.

4.7.3 Truncate

Pokud se vybrána tato strategie, tak se před každou migrací dat vyprázdní tabulka pro daný objekt z replikační databáze.

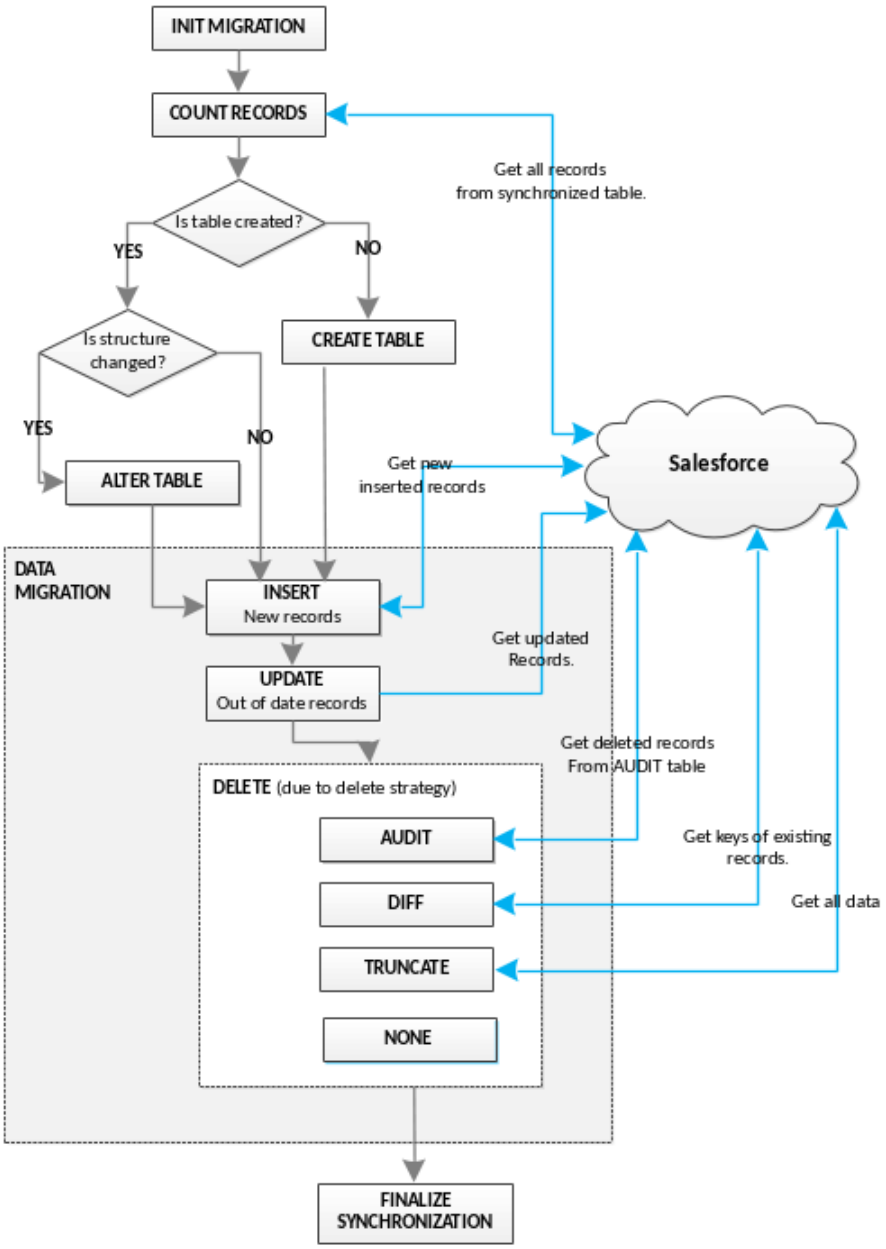
4.7.4 None

Tato strategie nemaže záznamy z replikační databáze. Hodí se například pro objekty, které nemohou své záznamy v Salesforce mazat (například log přihlášení apod.).

4.8 Proces replikace

Samotný proces replikace (též migrace) dat je rozdělen do následujících subprocesů:

- Inicializace migrace
- Získání počtu záznamů
- Vytvoření nebo úprava tabulky v replikační databázi
- Migrace nově vytvořených záznamů
- Migrace upravených záznamů
- Migrace smazaných záznamů
- Finalizace migrace



Obrázek 4.4: Proces replikace

4.8.1 Inicializace migrace

První subproces replikace dat. Z konfigurační databáze se načtou všechna data potřebná pro provedení replikace. Dále je inicializován tzv. Activity log (viz 4.6).

4.8.2 Získání počtu záznamů

Získá se dotazem na Salesforce počet nově vytvořených, upravených a smazaných záznamů od doby poslední synchronizace.

4.8.3 Vytvoření nebo úprava tabulky v replikační databázi

Pokud cílová tabulka v replikační databázi neexistuje, v tomto kroku je vytvořena. V opačném případě pokud je v nastavení synchronizace povolena volba Automatického obnovení schématu, pak se v tomto kroku provede kontrola na aktuálnost databázového schématu, případně jsou přidány nové sloupce nebo odebrány odmazané. Poslední kontrolou, která se zde provádí je kontrola datového typu a případně velikosti sloupců.

4.8.4 Migrace nově vytvořených záznamů

Pomocí SOAP API je proveden dotaz na Salesforce o ID záznamů, které byly nově vytvořeny od doby poslední synchronizace.

4.8.5 Migrace upravených záznamů

Pomocí SOAP API je proveden dotaz na Salesforce o ID záznamů, které byly upraveny od doby poslední synchronizace.

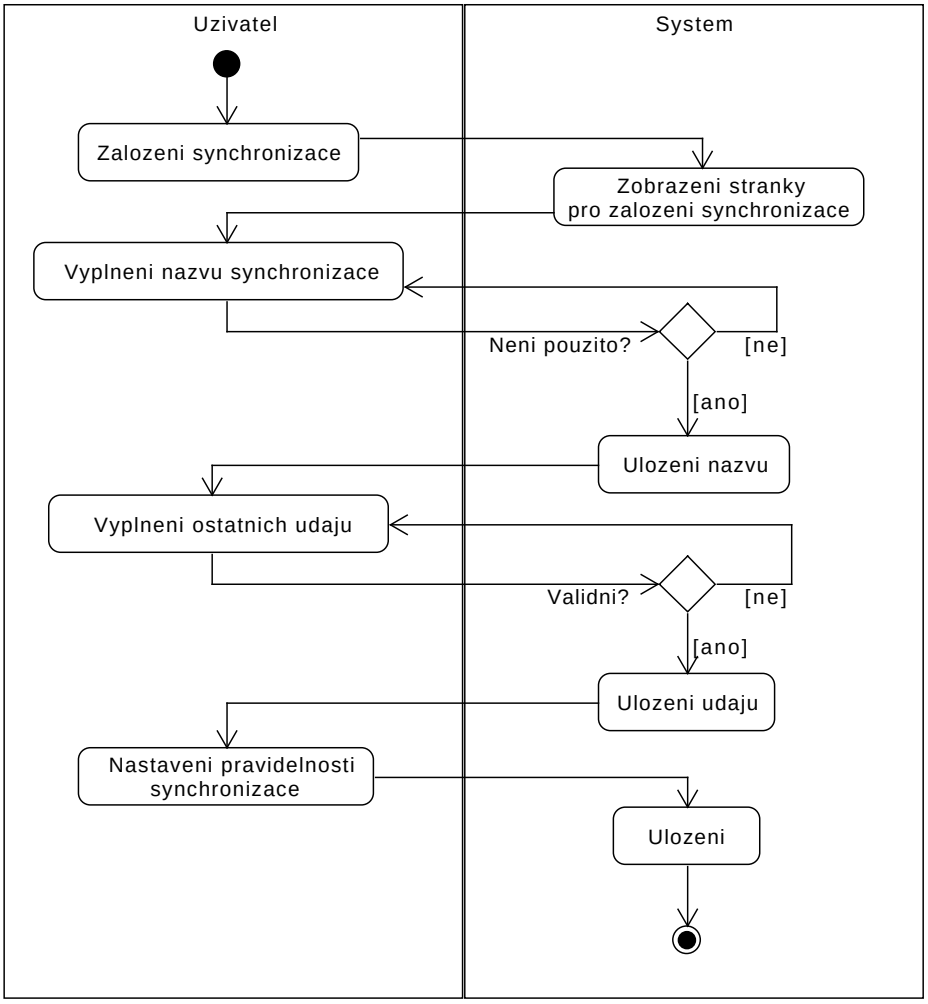
4.8.6 Migrace smazaných záznamů

Pomocí SOAP API je proveden dotaz na Salesforce o ID záznamů, které byly smazány od doby poslední synchronizace. Podle nastavení synchronizace je vybrána příslušná strategie mazání záznamů (viz. 4.7) a data jsou prostřednictvím vybrané strategie smazána.

4.8.7 Finalizace migrace

Poslední subproces replikace dat. Synchronizace je zde ukončena, activity log je uzavřen.

4.9 Activity diagram vytváření synchronizace



Obrázek 4.5: Aktivity diagram vytvoření synchronizace

Kapitola 5

Implementace zvoleného řešení

Aplikace je implementována v programovacím jazyce Java jako webová aplikace. Jako webový server je primárně použit Tomcat. Aplikace využívá framework Spring pro řízení vztahů mezi třídami a také Spring Security pro zajištění přístupu uživatelů k určitým akcím. Dále je využíván Hibernate framework pro persistenci dat do konfigurační databáze. Persistence dat v replikační databázi je z důvodu rychlosti, dynamického skládání dotazů a efektivity realizován pomocí JDBC. Pro plánování spuštění synchronizací v určitých okamžicích se používá framework Quartz. Pro tvorbu dokumentace je použit Asciidoctor.

5.1 Implementace replikačního algoritmu

V aplikaci je naimplementováno vícero migračních tasků, které zajišťují samotný průběh replikace dat. Efektivně je tohoto dosaženo s využitím Spring frameworku. Řídící třídou je zde ChainedMirrorStrategy.

```
public class ChainedMirrorStrategy implements MirrorStrategy {
   private List<MirrorTask> tasks;

   public void execute() {
      for (MirrorTask task : tasks) {
         if (task.shouldBeExecuted()) {
            task.execute(jobData);
         }
      }
   }
}
```

V uvedném příkladu je třída MirrorTask interface s metodami *shouldBeExecuted()*, která vrací **true**, pokud má být task spuštěn. Tato metoda se používá například u tasků, které není třeba spouštět každou jednotlivou synchronizaci. Jedná se například o kontrolu a migraci změněných záznamů od

poslední synchronizace při prvním spuštění (kde logicky nemohou být změněné záznamy). Druhou metodou je metoda *execute()*, která spuští daný task. V následující ukázce je definice Spring beany.

```
<bean id="mirrorService"
    class="com.solidservision.snowMirror.service.mirror.ChainedMirrorStrategy">
   <property name="tasks">
      <list>
         <ref bean="migrationInitializingTask" />
         <ref bean="autoSchemaUpdateTask" />
         <ref bean="prepareColumnsToSynchronizeTask" />
         <ref bean="sfoCountRecordsTask" />
         <ref bean="createOrAlterTableTask" />
         <ref bean="truncateBeforeInsertTask" />
         <ref bean="sfoMigrateCreatedTask" />
         <ref bean="sfoMigrateUpdatedTask" />
         <ref bean="sfoMigrateDeletedTask" />
         <ref bean="migrationFinalizingTask"/>
      </list>
   </property>
</bean>
```

Nyní si rozebereme jednotlivé tasky.

5.1.1 MigrationInitializingTask

Tento task je zodpovědný za správnou inicializaci Activity logu 4.6 a inicializace celého procesu synchronizace.

5.1.2 AutoSchemaUpdateTask

Pokud je v nastavení synchronizace zapnuta volba Auto Schema Update, tento task je spuštěn a způsobí kontrolu metadat tabulky. Kontrolují se jednotlivé sloupce, pokud některé přibyly, jsou zde přidány i do replikační tabulky.

5.1.3 PrepareColumnsToSynchronizeTask

Provádí překlad datových typů Salesforce na datové typy Javy, potažmo databázové datové typy.

5.1.4 SfoCountRecordsTask

Získává ze Salesforce pomocí SOAP API počet nově přidaných, upravených a na základě vybrané strategii mazání záznamů i počet smazaných záznamů od poslední synchronizace.

5.1.5 CreateOrAlterTableTask

Pokud není vytvořena replikační tabulka v replikační databázi, v tomto kroku je vytvořena. Pokud tabulka již existuje pouze se provede kontrola platnosti jejích sloupců. Pokud je definice již synchronizovaných sloupců změněna, stejná akce je provedena i nad synchronizovanou tabulkou. Kontrolují datové typy a jejich rozsahy pro jednotlivé sloupce.

5.1.6 TruncateBeforeInsertTask

Pokud je třeba tabulku před synchronizací vyprázdnit (je zvolena Truncate strategie pro mazání záznamů), pak je tabulka v replikační databázi v tomto kroku vyprázdněna.

5.1.7 SfoMigrateCreatedTask

Provede migraci nově vytvořených záznamů ze Salesforce od doby poslední synchronizace. Nejdříve se pomocí SOAP API vyžádají identifikátory záznamů, které byly přidány od poslední synchronizace a poté jsou k těmto identifikátorům přes BULK API stažena i příslušná data.

5.1.8 SfoMigrateUpdatedTask

Provede migraci upravených záznamů v Salesforce od doby poslední synchronizace. Nejdříve se pomocí SOAP API vyžádají identifikátory záznamů, které byly přidány od poslední synchronizace a poté jsou k těmto identifikátorům přes BULK API stažena i příslušná data.

5.1.9 SfoMigrateDeletedTask

Provede smazání stažených dat smazaných záznamů v Salesforce od doby poslední synchronizace. V závislosti na vybrané Delete strategii se postupuje různými způsoby.

- Audit
  Je dotázána tabulka deleting_log, ze které jsou získány identifikátory záznamů, které byly smazány od doby poslední synchronizace. Pomocí těchto identifikátorů jsou smazány i příslušné záznamy z replikační tabulky.

- Difference
  Pomocí SOAP API jsou získány identifikátory všech záznamů přítomných v Salesforce. Dále jsou získány identifikátory všech záznamů replikovaných v replikační tabulce. Provedením rozdílu (identifikátory z databáze - identifikátory ze Salesforce) jsou získány pouze ty identifikátory, které je třeba odmazat a následně jsou smazány.

- Truncate
  Tato strategie nedělá nic, protože pokud je vybrána tato stragie, je tabulka už vyprázdněna v *TruncateBeforeInsertTask*

- None
  Tato strategie nemaže žádné synchronizované záznamy. Vhodné například pro synchronizaci logovacích tabulek, kde se nepředpokládá mazání záznamů.

5.2 Změna Spring properties za běhu aplikace

Další ukázkou zdrojového kódu bude změna Spring properties za běhu aplikace bez nutnosti restartu. Tento požadavek je nutný z hlediska uživatele, kdy uživatel má možnost v nastavení aplikace změnit přístupové údaje k databázi, Salesforce atd. uložené ve Spring properties. Tohoto efektu je dosaženo rozdělením Springového kontextu na 2, v prvním kontextu jsou definovány tzv. nerestartovatelné beany a také je zde jedna beana, která tvoří Proxy do druhého kontextu, ve kterém jsou definovány všechny beany, které je možné restartovat (obsahují některou z měnitelných properties). Pokud je detekována změna některé ze Spring properties nutné pro běh aplikace, je kontext, ve kterém jsou definovány tzv. restartovatelné beany uzavřen a je vytvořen znovu se správnými propertami. Následuje ukázka zdrojového kódu.

```
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:context="http://www.springframework.org/schema/context"

    xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-3.2.xsd">

        <bean id="embeddedMirrorDbContextHolder"
    class="restart.EmbeddedContextHolder">
      <property name="contextLocation"
    value="classpath:mirror-database.xml" />
   </bean>

</beans>
```

V tomto úryvku zdrojového kódu je vidět definice Proxy beany v hlavním aplikačním kontextu. Beana s ID *embeddedMirrorDbContextHolder* slouží jako Proxy pro získávání Spring bean z tohoto kontextu. Pokud je změněna Spring property, která ovlivňuje chování některé beany obsažené v kontextu *embeddedMirrorDbContextHolder*, pak je před tuto Proxy beanu kontext uzavřen a znovu vytvořen se správnými hodnotami properties.

```
public class EmbeddedContextHolder implements
    ApplicationContextAware, InitializingBean {
   private ApplicationContext applicationContext;
   private ClassPathXmlApplicationContext embeddedContext;
   private String contextLocation;

   public Object getBean(String name) {
      return embeddedContext.getBean(name);
   }

   public void closeEmbeddedContext() {
      if (embeddedContext != null) {
         embeddedContext.close();
      }
   }

   public void createEmbeddedContext() {
      embeddedContext = new ClassPathXmlApplicationContext(new
    String[] { contextLocation }, false);

      embeddedContext.setParent(applicationContext);

      embeddedContext.refresh();
   }

   @Override
   public void setApplicationContext(ApplicationContext
    applicationContext) throws BeansException {
      this.applicationContext = applicationContext;
   }

   @Override
   public void afterPropertiesSet() throws Exception {
      Assert.notNull(contextLocation, "Context location must not be
    null.");
   }

}
```

Zde vidíme Java třídu, která zprostředkovává tzv. embedded kontext (kontext vytvořený uvnitř dalšího kontextu). Pomocí metod *createEmbeddedContext()* a *closeEmbeddedContext()* dochází k vytvoření, resp. uzavření daného kontextu. Metoda *getBean()* pouze deleguje získání Spring beany na vytvořený embedded kontext.

5.3 Šifrování citlivých informací

V konfigurační databázi je uloženo mnoho citlivých dat (heslo k replikační databázi, heslo k připojení do Salesforce, ...), proto je nutné tyto informace šifrovat, aby nebyly uloženy pouze ve své otevřené formě. Bohužel není možné hesla hashovat, protože je v otevřené podobě potřebujeme přečíst a použít pro připojení, což by při použití hashovací funkce bylo nemožné. Další problém, který byl potřeba vyřešit byl problém vytvoření šifrovacího klíče. Šifrovací klíč by neměl být uložen nikde na disku, protože by si ho mohl potenciální útočník lehce přečíst a také by to nemělo být lehce uhodnutelné. Dále bylo nutné zajistit, aby každá instalace aplikace vygenerovala pro stejná hesla jiné šifrové otisky, aby potenciální útočník neměl ulehčenou práci při odhalování šifrovacího klíče. To samé platí pro další požadavek na šifrovací klíč, a to aby pro každé heslo (heslo pro přístup do Salesforce, heslo do replikační databáze, ...) byl použit jiný šifrovací klíč. Abych byl schopen poznat již zašifrované heslo, je nutné k němu připojit jistý prefix, který mi heslo označí jako zašifrované. Řešení bylo nalezeno ve spolupráci s vedoucím práce postupným skládáním. Šifrovací klíč je rozdělen na 3 části, *Prefix*, *Hlavní část* a *Suffix*.

5.3.1 Prefix

Prefix klíče je jediná pseudonáhodná část klíče. S tvorbou této části mi pomohla Java, konkrétně její funkce *UUID.randomUUID()*. Výstup této funkce je pseudonáhodné vygenerované číslo [15]. Toto číslo je po vygenerování uloženo na disk do souboru.

5.3.2 Hlavní část

Hlavní část klíče tvoří název Spring property. Pokud máme heslo ke konfigurační databázi uloženo v property s názvem *mirror.jdbc.password*, pak řetězec *mirror.jdbc.password* je připojen k předchozí vygenerované hodnotě.

5.3.3 Suffix

Suffix je uložen přímo ve zdrojovém kódu aplikace v jedné z Java tříd.

5.3.4 Shrnutí

Spojením těchto výše popsaných 3 částí vzniká pseudonáhodný šifrovací klíč s dostatečnými vlastnostmi. Aby byl klíč pro každou instalaci aplikace jiný pro stejná hesla, tak to nám zařídí část *Prefix*. Požadavek, aby každé heslo mělo jiný šifrovací klíč nám zařídí *Hlavní část* klíče. Jako šifrovací funkce byla použita funkce AES s 256 bitovou délkou klíče. Java v základní verzi nepodporuje klíče delší než 128 bitů (pro šifru AES) [16], proto je nutný zásah uživatele, který si ve své JRE, resp. JDK musí doinstalovat rozšíření JCE

(Java Cryptography Extension). Toto rozšíření povoluje silnější šifry s delšími šifrovacími klíči.

```
private String doAction(String text, String password,
    EncryptionAction action) {
   MessageDigest sha256Digest = MessageDigest.getInstance("SHA-256");

   byte[] encryptionKey = ArrayUtils.addAll(installationUID,
    password.getBytes());
   encryptionKey = ArrayUtils.addAll(encryptionKey, passwordSuffix);

   Cipher cipher = Cipher.getInstance("AES/CBC/PKCS5Padding");
   SecretKeySpec key = new
    SecretKeySpec(sha256Digest.digest(encryptionKey), "AES");
   String ret = "";

   if (action == EncryptionAction.DECRYPT) {
      cipher.init(Cipher.DECRYPT_MODE, key, new IvParameterSpec(iv));

      // remove "enc:" prefix
      text = text.substring(ENC_PREFIX.length(), text.length());
      ret = new String(cipher.doFinal(Base64.decodeBase64(text)),
    "UTF-8");
   } else if (action == EncryptionAction.ENCRYPT) {
      cipher.init(Cipher.ENCRYPT_MODE, key, new IvParameterSpec(iv));

      ret = ENC_PREFIX;
      ret += new
    String(Base64.encodeBase64(cipher.doFinal(text.getBytes("UTF-8"))),
    "UTF-8");
   }

   return ret;
}
```

V této ukázce zdrojového kódu je vidět zašifrování, resp. odšifrování hesla. V proměnné *password* je uložen název Spring property (mirror.jdbc.password). Po spojení všech 3 částí klíče je tento klíč ještě zahashován hashovací funkcí SHA-256 pro dosažení přesně stanovené délky klíče (256 bitů). Poté je takto zahashovaný klíč předán jako šifrovací klíč pro šifrovací funkci AES.

5.4 Opakování dotazů na Salesforce v případě problémů

Při komunikaci na síti může nastat mnoho problémů. Služba je například dočasně nedostupná, zaneprázdněná nebo prostě neodpovídá. I na tyto možnosti je nutné v aplikaci myslet. Proto při dotazu na data byl implementován algo-

ritmus který zajišťuje, že se při selhání dotazu tento dotaz několikrát zopakuje než je rozhodnuto o zrušení celé synchronizace.

```
for (int i = 0; i < retryCountInt; i++) {
        updateLastRequestForData(mirrorJobData, new Date());
        waitForNextRetry(job.getId(), Long.valueOf(retries[i]) *
   1000);

        batch = bulkApi.getBatchInfo(job.getId(), batch.getId());

        if (batch.getState() == BatchStateEnum.Completed) {
           QueryResultList list =
   bulkApi.getQueryResultList(job.getId(), batch.getId());
           queryResults = list.getResult();
           break;
        } else if (batch.getState() == BatchStateEnum.Failed ||
   batch.getState() == BatchStateEnum.NotProcessed) {
           mirrorLoggerService.logMessage(String.format("Data
   preparation failed because of %s", batch.getStateMessage()),
                  MirrorLogger.MessageType.ERROR);
           bulkApi.closeJob(job.getId());
           throw new RuntimeException("Bulk API batch failed: " +
   batch.getStateMessage());
        }
     }

     if (batch.getState() != BatchStateEnum.Completed) {
        mirrorLoggerService.logMessage(String.format("Data not
   prepared after maximum retries. Error message: %s",
   batch.getStateMessage()),
              MirrorLogger.MessageType.ERROR);
     }
```

V tomto úryvku zdrojového kódu je použita proměnná *retryCountInt* která značí, kolikrát se má dotazovat a pole *retries*, které ve svých indexech obsahuje počet sekund, po jak dlouhou dobu počkat mezi dalším dotazem. Metoda *waitForNextRetry()* provádí pouze pozdržení dalšího dotazu po dobu aktuálních sekund. Dále získám do proměnné *batch* informace o tomto dotazu a kontroluji, zda byl již požadavek dokončen. Pokud ano, získám z tohoto požadavku výsledná data a končím algoritmus. Pokud byl požadavek označen stavem *Failed* nebo *NotProcessed* pak ukončuji celý pokus o získání dat, jelikož mi Salesforce tímto oznámil chybu v mém dotazu nebo z nějaké důvodu odmítl požadavek splnit, proto nemá cenu dále pokračovat v dotazování. Je také možné, že bude překročen limit počtu opakování a v takovém případě se ukončí *for* cyklus a pokud po ukončení tohoto cyklu není stav požadavku *Completed*, pak končím celou synchronizaci dat s chybou.

5.5 REST API

Aplikaci je možno ovládát i přes rozhraní REST API. Tato funkcionalita umožňuje vytvoření dalších klientských specifických aplikací ovládajících tuto aplikaci. Základní URL je *http://{hostname}:{port}/api/{api-version}/{resource-name}*, kde:

- hostname = adresa, kde aplikace beží
- port = port na kterém aplikace běží
- api-version = verze používaného API - aktuálně je pouze v1, do budoucna je zde otevřena možnost vylepšení
- resource-name = zdroj, se kterým budeme manipulovat

5.5.1 Seznam zdrojů

Každá žádost i případná následná odpověď na tuto žádost jsou kódovány v JSON formátu.

5.5.1.1 /synchronization

Provádí operace nad všemi synchronizacemi jako celkem.

- GET - Provede žádost o seznam všech synchronizací
  - 200 - seznam synchronizací úspěšně vygenerován a vrácen
  - 400 - chyba ve formátu žádosti nebo ve validaci
- POST - Vytvoření nové synchronizace
  - 200 - synchronizace byla úspěšně založena. V odpovědi je ID nově založené synchronizace
  - 400 - chyba v žádosti o vytvoření synchronizace nebo ve validaci
- DELETE - Smazání více synchronizací najednou
  - 200 - všechny synchronizace byly úspěšně smazány
  - 400 - chyba v žádosti o smazání synchronizací nebo při validaci
  - 404 - některá ze synchronizací neexistuje. Žádná akce není provedena.

5.5.1.2 /synchronization/action

Provede určité akce nad seznamem synchronizací (například hromadné zastavení synchronizací, smazání ...)

- GET - Provede žádost o seznam všech akcí nad seznamem synchronizací
  - 200 - seznam akcí úspěšně vrácen
  - 400 - chyba ve formátu žádosti nebo ve validaci
- POST - Provede danou akci nad seznamem synchronizací
  - 200 - akce byly úspěšně provedeny
  - 404 - některá synchronizace nebyla nalezena, žádná akce nebyla provedena

5.5.1.3 /synchronization/{id}

Provádí operace s určitou synchronizací.

- GET - Získá detail synchronizace se všemi údaji
  - 200 - detail synchronizace úspěšně vrácen
  - 404 - synchronizace nebyla nalezena
- PUT - Editace synchronizace
  - 200 - synchronizace byla úspěšně editována
  - 400 - chyba v žádosti o editaci synchronizace nebo ve validaci
  - 404 - synchronizace nebyla nalezena
- DELETE - Smazání dané synchronizace
  - 200 - synchronizace byla úspěšně smazána
  - 400 - chyba při mazání synchronizace
  - 404 - synchronizace nebyla nalezena

5.5.1.4 /synchronization/{id}/action

Provádí určité akce nad danou synchronizací.

- GET - Získá seznam všech proveditelných akcí nad danou synchronizací
  - 200 - seznam akcí nad synchronizací úspěšně vygenerován a vrácen
  - 404 - synchronizace nebyla nalezena
- POST - Provedení akce nad danou synchronizací

  - 200 - akce byla úspěšně provedena
  - 400 - chyba v žádosti o provedení akce nad synchronizací nebo ve validaci
  - 404 - synchronizace nebyla nalezena

5.5.1.5 /synchronization/history

Získává historii všech synchronizací založených v aplikaci.

- GET - Získání historie všech synchronizací
  - 200 - historie synchronizací úspěšně vygenerovány a odeslány

5.5.1.6 /synchronization/{id}/history/{id}

Získává informace o určitém průběhu určité synchronizace.

- GET - Získá informace o určitém průběhu určité synchronizace
  - 200 - historie synchronizace byla úspěšně vygenerována a odeslána
  - 404 - synchronizace nebo její historie nebyla nalezena

5.5.1.7 /synchronization/{id}/history/{id}/log

Získá Activity log 4.6 daného běhu dané synchronizace.

- GET - Provede žádost o log daného průběhu dané synchronizace
  - 200 - log úspěšně odeslán
  - 404 - synchronizace, historie nebo log nebyly nalezeny

5.5.2 Ukázka žádosti o vytvoření synchronizace

```
{
    "sync": {
        "name": "Lead",
        "table": "Lead",
        "mirrorTable": "Lead",
        "columns": [
            {
                "name": "Id"
            },
            {
                "name": "LastName"
            }
```

```
        ],
        "autoSchemaUpdate": false,
        "userQuery": "",
        "deleteStrategy": "DIFF",
        "active": true,
        "scheduler": {
            "type": "MANUALLY",
            "beginDate": "2014-08-07"
        },
        "fullLoadScheduler": {
            "type": "MANUALLY",
            "beginDate": "2014-08-07"
        }
    }
}
```

V přiložené ukázce žádosti o vytvoření synchronizace je vidět synchronizace Salesforce tabulky *Lead* do replikační tabulky s názvem *Lead*, synchronizovat se budou sloupce *Id* a *LastName*. Strategie mazání neplatných záznamů je nastavena na *DIFF* (viz 4.7). Synchronizace je aktivní a bude spouštěna manuálně.

5.5.3 Ukázka odpovědi pro žádost o seznam všech akcí nad danou synchronizací

```
{
    "actions": [
        "SYNCHRONIZE",
        "CLEAN_AND_SYNCHRONIZE",
        "STOP_SYNCHRONIZATION"
    ]
}
```

V přiložené ukázce odpovědi ve formátu JSON jsou vidět 3 možné akce nad danou synchronizací: Spuštění synchronizace (*SYNCHRONIZE*), vymazání dat a následné spuštění synchronizace (*CLEAN_AND_SYNCHRONIZE*) a zastavení synchronizace (*STOP_SYNCHRONIZATION*).

Kapitola 6

Testování a dokumentace

6.1 Testování

Aplikace obsahuje celkem 942 testů s celkovým pokrytím 50.6%. Aplikace obsahuje testy jednotkové, integrační, smoke testy a také SOAP UI testy pro testování REST API aplikace. Aplikace dále obsahuje 23 Selenium testů, které testují workflow jednotlivých obrazovek (založení nové synchronizace, úprava stávající synchronizace, deaktivace, ...).

6.1.0.1 Ukázka jednotkového testu

Tento test testuje mapování více *SynchronizationEntity* na *SynchronizationBulkGetResponseDto*. Jedná se o test správné funkčnosti Dozer mapperu, který se používá pro mapování jedné entity na jinou.

```
@Test
void SynchronizationEntitiesToSynchronizationBulkGetResponseDto() {
   def syncs = [new SynchronizationEntity(name: "sync 1"), new
    SynchronizationEntity(name: "sync 2")]
   def syncsDto = mapper.map(syncs,
    SynchronizationBulkGetResponseDto.class)
   assertEquals(2, syncsDto.getSyncs().size())
   assertEquals("sync 1", syncsDto.getSyncs()[0].getName())
   assertEquals("sync 2", syncsDto.getSyncs()[1].getName())
}
```

6.1.0.2 Ukázka integračního testu

Tento test testuje scénář získání podrobností o tabulce ze Salesforce. Pomocí WireMock frameworku je namockováno chování Salesforce. Pokud adresa, pro kterou volám o data je */services/Soap/u/32.0/00D24000000HMy0*, nadefinuji chování tak, že v odpovědi bude status 200 (OK) a samotná odpověď se vezme

ze souboru *successful_get_table_detail.xml*. Poté se zkontroluje, že název tabulky i její popisek je *Account* a první sloupec je *AccountNumber*. Poslední kontrolou je ta, že tabulka nemá žádnou nadřazenou tabulku.

```
@Test
public void testFindTable() {
   stubFor(
      post(
         urlEqualTo("/services/Soap/u/32.0/00D24000000HMy0")
      ).willReturn(
         aResponse()
            .withStatus(200)
            .withBody(loadFromFile("successful_get_table_detail.xml"))
      )
   );

   Table t = service.findTable("Account");
   assertNotNull(t);
   assertEquals("Account", t.getName());
   assertEquals("Account", t.getLabel());
   assertEquals("AccountNumber",
    t.getColumns().get(0).getColumnName());
   assertNull(t.getParentTable());
}
```

6.2 Dokumentace

Přílohou bakalářské práci je Administrátorská příručka, Uživatelská příručka a popis REST API.

6.2.1 Administrátorská příručka

Tato příručka je vhodná pro administrátory, kteří se chystají nasadit aplikaci na server ve firmě. Administrátor v této příručce nalezne:

- Požadavky na Salesforce - například nutná oprávnění uživatele, pod kterým se bude provádět replikaci, pokud se uživatel nerozhodne použít administrátorský účet)
- Hardwarové požadavky na PC
- Odhady využití disku - aplikace zakládá logy pro každou synchronizaci (viz 4.6)
- Požadavky na JRE resp. JDK
- Požadavky na aplikační server

- Seznam podporovaných databází
- Seznam podporovaných operačních systémů
- Popis instalace aplikace
- Popis upgrade aplikace
- Průvodce prvotním nastavením aplikace
- Návod na odinstalování aplikace
- Nastavení aplikace
- Návod na vytvoření uživatele s dostatečnými právy na Salesforce
- Pokročilá administrace - například nastavení komunikace aplikace se Salesforce přes HTTPS
- FAQ a Release notes

6.2.2 Uživatelská příručka

Tato příručka je vhodná pro koncové uživatele aplikace. Uživatel zde nalezne popis jednotlivých obrazovek a návod na celkové ovládání aplikace. Jedná se hlavně o:

- Popis rolí uživatele aplikace
- Popis hlavní stránky
- Ukázku detailu synchronizace
- Průvodce založením nové synchronizace
- Průvodce nastavením aplikace

6.2.3 Popis REST API

Jak již bylo zmíněno v 5.5 aplikaci je možno ovládat přes REST API. Tato dokumentace poskytuje detailní vysvětlení jednotlivých funkcí.

Závěr

V mojí bakalářské práci jsem měl za cíl popsat možné problémy při replikaci dat přes webové služby, dále navrhnout řešení těchto problémů a poté vytvořit aplikaci replikující data ze Salesforce do vlastní databáze s důrazem na řešení popsaných problémů. V kapitole 2 byly probrány jednotlivé problémy při synchronizaci dat přes webové služby a bylo navrženo adekvátní řešení, pokud to bylo možné. Existují problémy, které z hlediska klientské aplikace dotazující se na cloudový systém nelze vyřešit (například nedostupnost služby). V následující kapitole 3 byl představen cloudový systém Salesforce, dále funkční, nefunkční požadavky kladené na aplikaci, byla provedena rešerše stávajících řešení daného problému a zhodnocení a výběr řešení. Všechny funkční i nefunkční požadavky se podařilo splnit. V kapitole 4 byl proveden návrh řešení s použitím knihovny WSC od společnosti Salesforce, rozebrány jednotlivé uživatelské role v aplikaci a vybrány vhodné technologie pro vývoj aplikace. Poté byla představena architektura celé aplikace a zvoleny podporované typy databází, jak pro konfigurační, tak replikační databázi. Byly představeny jednotlivé typy mazání neplatných záznamů z replikační databáze a samotný proces replikace. Následující kapitola 5 již obsahovala praktické ukázky zdrojových kódů pro řešení vybraných částí aplikace. Byl zde ukázán hlavní algoritmus průběhu jedné synchronizace, dále bylo ukázáno jak za běhu aplikace je možné měnit její nastavení. Další ukázkou zdrojového kódu bylo šifrování citlivých informací z důvodu bezpečnosti aplikace. Poté bylo prakticky ukázáno řešení problému zahlcenosti cloudové služby algoritmem pro opakování dotazů v případě problémů a nakonec popis REST API aplikace. V poslední kapitole 6 byly ukázány některé druhy testů a popis jednotlivých druhů dokumentace.

Plány do budoucna

Do budoucna by se dalo uvažovat o možnosti ovládání pomocí mobilní aplikace. Mohl by vzniknout nativní klient pro Android, iOS a Windows Phone, který by využíval REST API pro ovládání aplikace přímo z telefonu. Uživatel

by tak mohl spouštět synchronizace manuálně, zobrazovat si poslední proběhnuté synchronizace či sledovat grafy přímo na svém telefonu. Dalším nápadem do budoucna je možnost replikace příloh k záznamům, které jsou uloženy zakódované v base64 přímo na Salesforce. Typicky k informacím o zaměstnancích v Salesforce jsou přiloženy některé dokumenty (pracovní smlouva, fotografie aj.), které by bylo vhodné také replikovat za účelem zálohy pro případ, že by cloudová služba ukončila činnost a rychlejšího přístupu k těmto přílohám přímo na disk.

Literatura

[1] SMITH, Gerry a Betsy ISAACSON. Websites Scramble As Hurricane Sandy Floods Data Centers. *The Huffington Post* [online]. 30. 10. 2012 [cit. 2015-04-08]. Dostupné z: http://www.huffingtonpost.com/2012/10/30/hurricane-sandy-websites-floods-data-centers_n_2046034.html

[2] NUSCA, Andrew. DDoS attack on UltraDNS affects Amazon.com, SalesForce.com, Petco.com. *ZDNet.com* [online]. 1. 4. 2009 [cit. 2015-04-08]. Dostupné z: http://www.zdnet.com/article/ddos-attack-on-ultradns-affects-amazon-com-salesforce-com-petco-com

[3] WEINS, KIM. Cloud Computing Trends: 2015 State of the Cloud Survey. *Right Scale* [online]. 2015 [cit. 2015-05-02]. Dostupné z: http://www.rightscale.com/blog/cloud-industry-insights/cloud-computing-trends-2015-state-cloud-survey

[4] Informatica cloud. *Informatica cloud* [online]. 2015 [cit. 2015-03-18]. Dostupné z: https://www.informatica.com

[5] Learn About Java Technology. *Java* [online]. 2015 [cit. 2015-03-18]. Dostupné z: https://java.com/en/about

[6] Cloud Data Replication for Salesforce. *DBSync* [online]. 2009 [cit. 2015-03-18]. Dostupné z: http://www.mydbsync.com/integration/cloud-replication-for-salesforce

[7] Features. *Snow Mirror* [online]. 2015 [cit. 2015-03-16]. Dostupné z: http://www.snow-mirror.com/features

[8] SnapLogic. *SnapLogic* [online]. 2015 [cit. 2015-03-18]. Dostupné z: http://www.snaplogic.com/

[9] WSC. In: *github* [online]. [cit. 2015-04-15]. Dostupné z ttps://github.com/forcedotcom/wsc

[10] SOAP API Developer's Guide. *Salesforce.com* [online]. 2015 [cit. 2015-04-15]. Dostupné z: https://www.salesforce.com/developer/docs/api

[11] Bulk API Developer's Guide. *Salesforce.com* [online]. 2015 [cit. 2015-04-15]. Dostupné z: https://www.salesforce.com/us/developer/docs/api_asynch

[12] Amazon EC2 SLA. *Amazon Web Services* [online]. 2013 [cit. 2015-04-15]. Dostupné z: http://aws.amazon.com/ec2/sla

[13] Google Cloud Storage, Google Prediction API, and Google BigQuery SLA. *Google Cloud Platform* [online]. 2015 [cit. 2015-04-15]. Dostupné z: https://cloud.google.com/storage/sla

[14] Service Level Agreements. *Microsoft Azure* [online]. 2015 [cit. 2015-04-15]. Dostupné z: http://azure.microsoft.com/en-us/support/legal/sla/

[15] Java Platform SE 7. *Oracle Java Documentation* [online]. 2014 [cit. 2015-05-02]. Dostupné z: http://docs.oracle.com/javase/7/docs/api/java/util/UUID.html#randomUUID()

[16] Java Platform SE 7. *Oracle Java Documentation* [online]. 2014 [cit. 2015-05-02]. Dostupné z: http://docs.oracle.com/javase/7/docs/api/javax/crypto/Cipher.html

[17] *Microsoft Technet* [online]. [cit. 2015-05-05]. Dostupné z: https://i-technet.sec.s-msft.com/dynimg/IC196811.gif

[18] My Starbucks Idea. *Starbucks Corporation* [online]. 2013 [cit. 2015-05-05]. Dostupné z: http://mystarbucksidea.force.com

[19] Spring Framework. *Spring Projects* [online]. 2015 [cit. 2015-05-07]. Dostupné z: http://projects.spring.io/spring-framework

[20] Spring Security. *Spring Projects* [online]. 2015 [cit. 2015-05-07]. Dostupné z: http://projects.spring.io/spring-security

Příloha **A**

Seznam použitých zkratek

API Application Programming Interface

BI Business Intelligence

CD Compact disc

CI Continuous Integration

CRM Customer relationship management

CRUD Create, Retrieve, Update, Delete

CSV Comma-separated values

FAQ Frequently Asked Questions

FB Facebook

GUI Graphical user interface

HTTP Hypertext Transfer Protocol

HTTPS Hypertext Transfer Protocol Secure

iPaaS integration Platform as a Service

JRE Java Runtime Environment

JDK Java Development Kit

MSSQL Microsoft SQL Server

MVC Model-View-Controller

PC Personal Computer

REST Representational State Transfer

SaaS Software as a Service

SOAP Simple Object Access Protocol

UC Use Case

VPN Virtual Private Network

WSDL Web Services Description Language

XML Extensible Markup Language

Příloha B

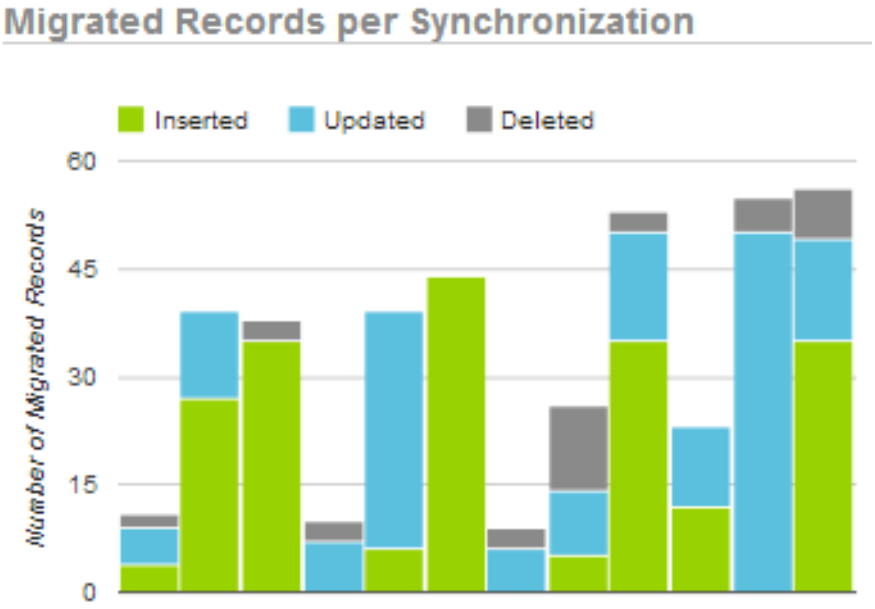
Obsah přiloženého CD

- `readme.txt` ..... stručný popis obsahu CD
- `src`
  - `thesis` ..... zdrojová forma práce ve formátu LaTeX
- `text` ..... text práce
  - `thesis.pdf` ..... text práce ve formátu PDF
  - `thesis.ps` ..... text práce ve formátu PS
- `documentation` ..... dokumentace
  - `administrator_guide.html` ..... administrátorská příručka
  - `usermanual.html` ..... uživatelská příručka
  - `RestApiGuide.html` ..... popis REST rozhraní aplikace

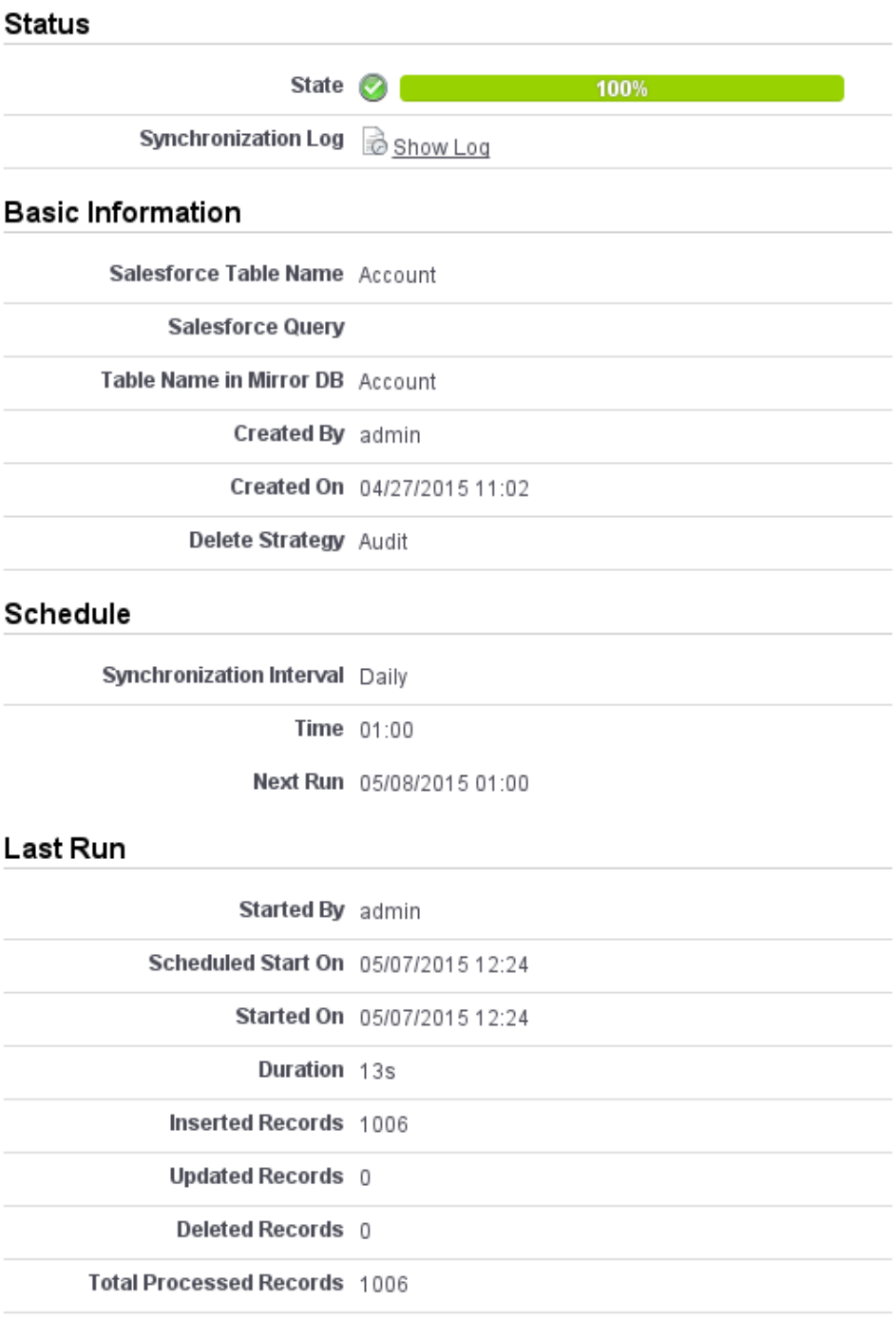
Příloha C

Ukázka aplikace

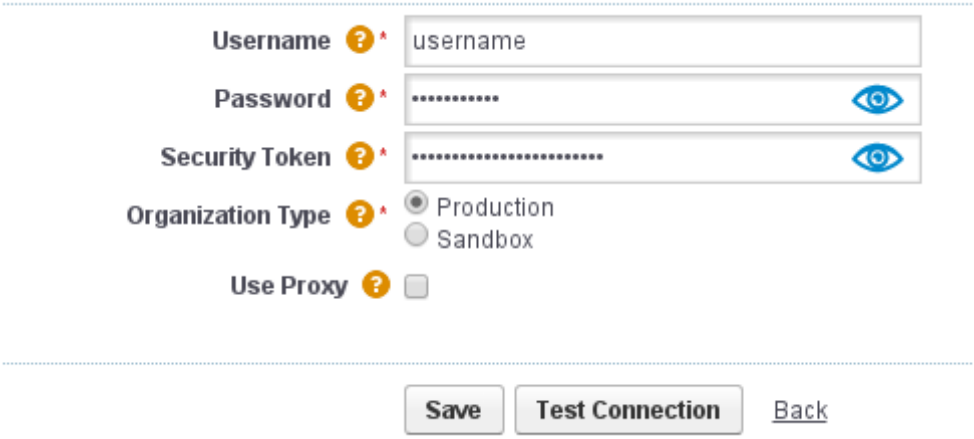
V této příloze naleznete jednotlivé ukázky výřezů obrazovek z aplikace. Obrázek C.1 ukazuje, kolik záznamů bylo synchronizováno (vytvořeno, upraveno a smazáno) v každé synchronizaci. Obrázek C.2 ukazuje přehled o nastavení synchronizace a o posledním průběhu. Obrázek C.3 ukazuje výřez obrazovky pro nastavení přihlašovacích údajů do Salesforce a obrázek C.4 ukazuje výřez obrazovky pro nastavení údajů do replikační databáze. Na obrázku C.5 je vidět nastavení uživatele a možnost přiřazování jednotlivých rolí. Na posledním obrázku (C.6) je ukázáno, jak si uživatel může aplikaci přizpůsobit podle svého uvážení.



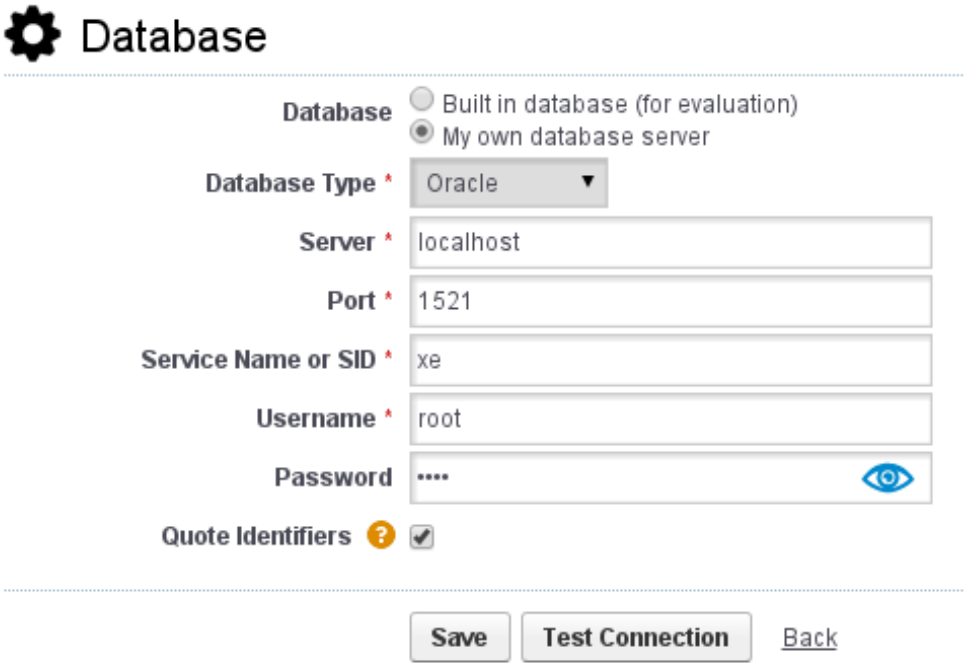
Obrázek C.1: Počet záznamů dle synchronizace



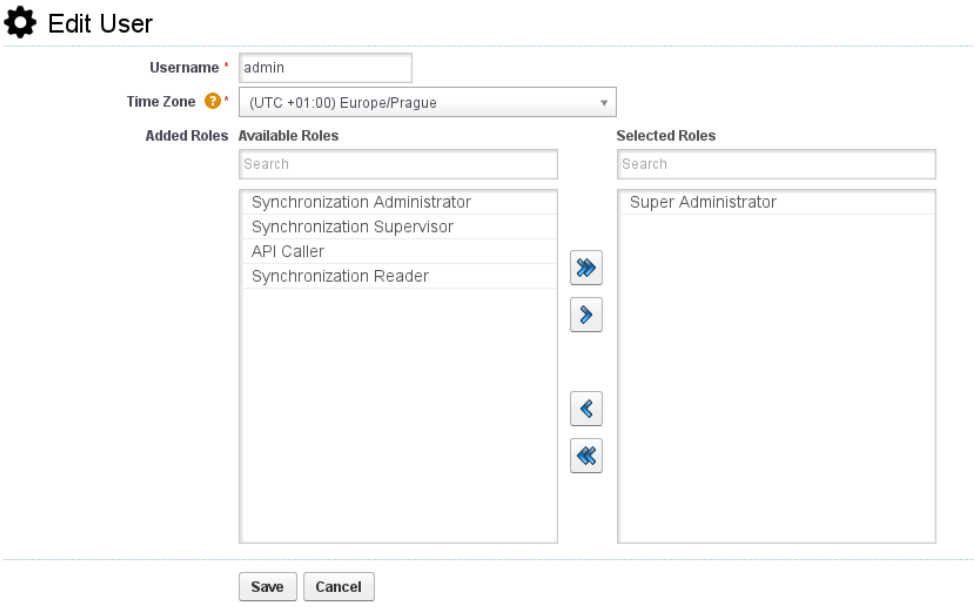
Obrázek C.2: Detail synchronizace



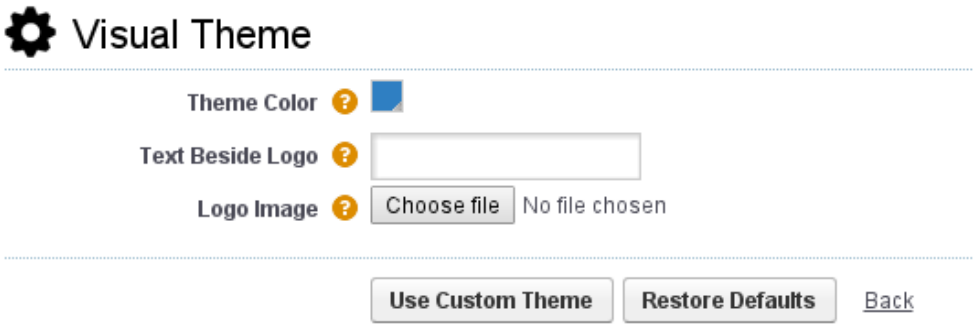
Obrázek C.3: Nastavení přihlášení do Salesforce



Obrázek C.4: Nastavení replikační databáze



Obrázek C.5: Editace uživatele a přiřazování rolí



Obrázek C.6: Volba barevného schématu aplikace