VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INFORMAČNÍCH SYSTÉMŮ
DEPARTMENT OF INFORMATION SYSTEMS

DISTRIBUOVANÝ DOKUMENTOVÝ SERVER ZALOŽENÝ NA DATABÁZE COUCHDB

DISTRIBUTED DOCUMENT SERVER BASED ON COUCHDB DATABASE

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE **Bc. MARTIN KANIS**
AUTHOR

VEDOUCÍ PRÁCE **RNDr. MAREK RYCHLÝ, Ph.D.**
SUPERVISOR

BRNO 2017

Zadání diplomové práce/19246/2016/xkanis00

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav informačních systémů Akademický rok 2016/2017

Zadání diplomové práce

Řešitel: **Kanis Martin, Bc.**
Obor: Inteligentní systémy
Téma: **Distribuovaný dokumentový server založený na databázi CouchDB**
Distributed Document Server Based on CouchDB Database
Kategorie: Informační systémy

Pokyny:

1. Seznamte se s problematikou distribuovaného uložení dat a s databází CouchDB. Prozkoumejte, jakým způsobem CouchDB řeší synchronizaci a CAP problém.
2. Navrhněte distribuovaný dokumentový server s webovým rozhraním využívající pro dokumenty jako úložiště CouchDB databázi. Dokumentový server umožní nejen ukládání dokumentů do distribuovaného úložiště, ale bude také provádět řízení toku dokumentů v s nimi spojeném workflow vč. implementace systému oprávnění. Navrhněte také RESTful API pro přístup k dokumentům.
3. Po konzultaci s vedoucím navržený systém implementujte jako webovou aplikaci. Výsledek zveřejněte jako open-source.
4. Proveďte zhodnocení dosažených výsledků a diskutujte další možný vývoj projektu.

Literatura:

- Apache CouchDB(tm) 1.6.1 Documentation. [http://docs.couchdb.org/en/1.6.1/contents.html]
- Jiří Barbořík. Informační systém pro řízení toku dokumentů. Diplomová práce, FIT VUT v Brně, 2009. [http://www.fit.vutbr.cz/study/DP/DP.php?id=8597]

Při obhajobě semestrální části projektu je požadováno:

- bez požadavků

Podrobné závazné pokyny pro vypracování diplomové práce naleznete na adrese http://www.fit.vutbr.cz/info/szz/

Technická zpráva diplomové práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap, které byly vyřešeny v rámci dřívějších projektů (30 až 40% celkového rozsahu technické zprávy).

Student odevzdá v jednom výtisku technickou zprávu a v elektronické podobě zdrojový text technické zprávy, úplnou programovou dokumentaci a zdrojové texty programů. Informace v elektronické podobě budou uloženy na standardním nepřepisovatelném paměťovém médiu (CD-R, DVD-R, apod.), které bude vloženo do písemné zprávy tak, aby nemohlo dojít k jeho ztrátě při běžné manipulaci.

Vedoucí: **Rychlý Marek, RNDr., Ph.D.**, UIFS FIT VUT
Datum zadání: 1. listopadu 2016
Datum odevzdání: 24. května 2017

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
Fakulta informačních technologií
Ústav informačních systémů
612 66 Brno, Božetěchova 2

doc. Dr. Ing. Dušan Kolář
vedoucí ústavu

Abstrakt

Práca pojednáva o problematike distribuovaných databázových systémov, o ich výhodách a nevýhodách. Následne text oboznamuje s dokumentovou databázou *CouchDB*, spôsobom ukladania dokumentov, riešením synchronizácie a prístupu ku CAP teorému. Cieľom práce je implementovať distribuovaný databázový server pre správu a *workflow* dokumentov, ktorý je založený na *CouchDB*. Systém obsahuje *cluster* s troma *CouchDB* uzlami a *HAProxy* na popredí, ktorá rozdeľuje záťaž medzi uzlami. Systém umožňuje vytváranie ľubovoľných dokumentov na základe šablóny, spravuje ich životný cyklus a tok. Taktiež je možné vytvárať vlastné *workflow* za pomoci *BRMS* pravidiel. Implementované riešenie uľahčuje spravovanie a tok dokumentov a umožňuje vysokú mieru prispôsobenia potrebám organizácie.

Abstract

Thesis discusses distributed database systems and its advantages and disadvantages. Further, text informs about document database *CouchDB*, storage of documents, synchronization and CAP theorem. The aim of the thesis is to implement distributed document management system and workflow management system based on *CouchDB*. The system contains a cluster with three *CouchDB* nodes with *HAProxy* in front, which does load balancing. The system allows creation of any document based on the template, manages its life cycle and workflow. It is also possible to create a custom workflow using *BRMS* rules. The implemented solution simplifies document management and workflow and allows a high degree of customization for the organizations needs.

Kľúčové slová

Distribuované databázové systémy, CouchDB, Synchronizácia, CAP teorém, B-stromy, Správa dokumentov, Tok dokumentov

Keywords

Distributed database system, CouchDB, Synchronization, CAP theorem, B-trees, Document management, Workflow

Citácia

KANIS, Martin. *Distribuovaný dokumentový server založený na databáze CouchDB*. Brno, 2017. 55 s. Diplomová práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Rychlý Marek.

Distribuovaný dokumentový server založený na databáze CouchDB

Prehlásenie

Prehlasujem, že som túto diplomovú prácu vypracoval samostatne pod vedením pána RNDr. Marka Rychlého Ph.D. Uviedol som všetky literárne pramene a publikácie, z ktorých som čerpal.

.....
Martin Kanis
16. mája 2017

Poďakovanie

Chcel by som poďakovať vedúcemu práce RNDr. Marku Rychlému Ph.D. za rady, ktoré mi poskytol.

Obsah

Kapitola 1

Úvod

Distribuované databázové systémy patria medzi rýchlo rozvíjajúce sa odvetvia informatiky. Dôvodom je, že moderné aplikácie vyžadujú zvýšenú priepustnosť dát a transakcií, čo vedie k potrebe škálovateľných databázových systémov.

Pre tento typ aplikácii je potreba databázy, ktoré budú schopné efektívne uložiť a spracovávať veľké množstvo dát. Preto vznikli takzvané NoSQL databázy, ktoré majú iný pohľad na dáta, ako tradičné relačné databázy.

Problém spracovania dokumentov má veľký dopad na produktivitu v podnikoch. Spracovanie dokumentov je stále náročnejšie, pretože zahŕňa viacerých účastníkov potencionálne rozmiestnených na vzdialených pracoviskách s rôznymi rolami a úlohami. Mnoho každodenných úkonov s dokumentmi môže byť automatizovaných pomocou systému, ktorý bude riadiť tok týchto dokumentov. Preto je možné očakávať, že implementácia takého systému, bude mať priaznivý účinok na efektivitu podniku.

Táto diplomová práca naväzuje na semestrálny projekt, v ktorom bol popísaný CAP teorém, detailne predstavená dokumentová databáza CouchDB a vytvorený návrh programu.

Druhá kapitola pojednáva o distribuovaných databázových systémoch. Objasňuje, čo to sú distribuované databázové systémy, prečo sú tieto systémy potrebné, aké majú výhody a úskalia. Kapitola ďalej popisuje CAP teorém a alternatívny model nazývaný PACELC.

V rámci tretej kapitoly je detailne popísaná databáza CouchDB, ktorá je využitá v aplikácii. Popísaný je spôsob ukladania dokumentov, prístup ku CAP teorému a riešenie synchronizácie.

Štvrtá kapitola obsahuje popis a funkcionalitu systémov pre správu dokumentov (DMS). Popisuje, ako DMS uľahčuje organizáciám správu dokumentov a aké technológie sa využívajú v rámci DMS.

Piata kapitola popisuje základnú terminológiu *workflow*, ktorú vytvorila organizácia Workflow Management Coalition s cieľom zjednotiť terminológiu v tejto oblasti. V závere kapitoly sú popísané výhody, ktoré *workflow* prináša organizáciám.

Šiesta a siedma kapitola obsahujú návrh aplikácie a implementačné detaily. V rámci implementácie sú popísané použité technológie, architektúra programu a riešenie kľúčových aspektov aplikácie.

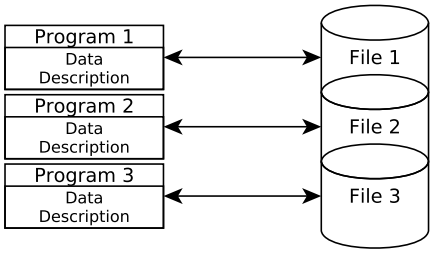
Ôsma kapitola obsahuje prípadovú štúdiu, ktorá oboznamuje s vytvorenou aplikáciou a ukazuje jej praktické využitie.

Deviata kapitola zhodnocuje dosiahnuté výsledky spolu s návrhmi na ďalšie vylepšenia.

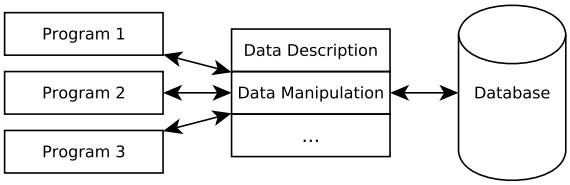
Kapitola 2

Distribuované databázové systémy

Distribuované databázové systémy vznikli spojením dvoch technológii s cieľom vytvoriť novú, lepšiu technológiu. Jedná sa o databázové systémy a počítačové siete. Databázové systémy popisujú a spracovávajú dáta centrálne, čo ústi v dátovú nezávislosť. Prostredníctvom čoho sú aplikačné programy odolné voči zmenám v logickej alebo fyzickej organizácii dát a naopak. Je to opak paradigmu, kde každá aplikácia definuje a spracováva svoje vlastné dáta. Oba prístupy sú zobrazené na obrázku 2.1 a 2.2.



Obr. 2.1: Tradičné spracovanie dát



Obr. 2.2: Databázové spracovanie dát

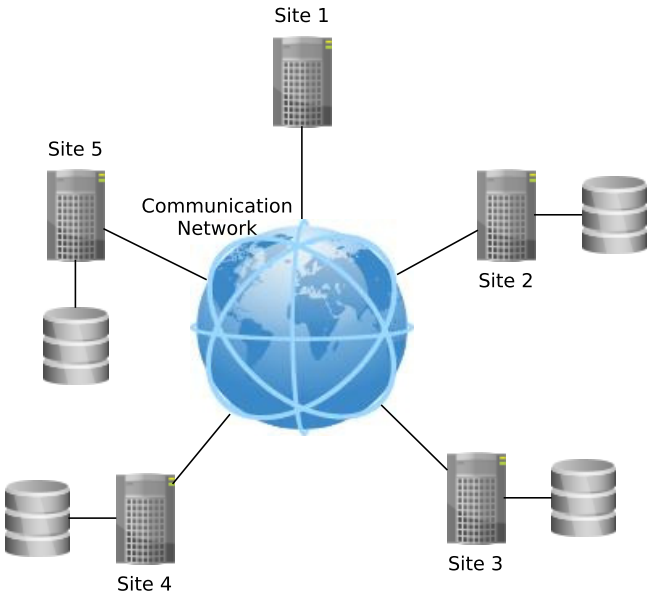
Hlavným cieľom databázových systémov je integrovať podnikové dáta a poskytnúť k ním centralizovaný prístup. Na druhú stranu, princíp počítačových siete ide proti centralizácii. Preto môže byť na prvý pohľad ťažšie pochopiť, ako môžu byť tieto dva prístupy spojené do jedného a vytvoriť tak sľubnú technológiu. Kľúčom k pochopeniu je uvedomenie si, že najdôležitejší cieľ databázovej technológie je integrácia a nie centralizácia. Dosiahnutie jed-

nej z týchto vlastností neimplikuje dosiahnutie druhej. Je však možné dosiahnuť integráciu aj bez centralizácie, čo je to, čo sa snažia distribuované databázové systémy dosiahnuť.

2.1 Distribuovaný databázový systém

S distribuovanými databázovými systémami súvisí pojem distribuované spracovanie dát. Je ho možné definovať, ako niekoľko autonómnych výpočetných elementov, ktoré sú prepojené počítačovou sieťou a spolupracujú pri riešení pridelenej úlohy. Distribuovanie je využívané z dôvodu, že lepšie vyhovuje štruktúre celosvetových podnikov a taký systém je potom viac spoľahlivý a dostupný. Mnoho súčasných využití počítačovej technológie je neodmysliteľne distribuovaných. Patria medzi ne napríklad webové aplikácie, elektronické internetové reklamy, multimediálne aplikácie a veľa ďalších. Základným dôvodom využívania distribuovaných systémov je, že sú lepšie schopné zvládnuť problémy s veľkým objemom dát, ktorým v súčasnosti čelíme. Je to za pomocí známeho princípu „rozdeľuj a panuj“.

Distribuované databázy môžeme definovať, ako množinu viacerých, logicky nesúvisiacich databáz, vzájomne prepojených počítačovou sieťou. Distribuovaný databázový systém je potom systém, ktorý dovoľuje spravovanie distribuovanej databázy tak, aby bola distribuovanosť transparentná pre užívateľov [28]. Na obrázku 2.3 je zobrazené prostredie distribuovaného databázového systému, na ktorom vidno databázy distribuované do fyzicky rozdielnych lokalít, prepojených počítačovou sieťou.



Obr. 2.3: Schéma distribuované databázového systému

2.2 Výhody

Distribuované databázové systémy majú mnoho výhod, ktoré môžeme zhrnúť do štyroch základných celkov: transparentné spravovanie distribuovaných a replikovaných dát, spoľahlivý

prístup k dátam, zlepšenie výkonu a jednoduchšie rozširovanie systému. V Ďalších častiach ich podrobnejšie rozoberieme.

2.2.1 Transparentné spravovanie distribuovaných a replikovaných dát

Transparencia v tomto kontexte znamená oddelenie vysoko-úrovňovej sémantiky systému od nízko-úrovňových implementačných detailov. Inak povedané, transparentný systém skrýva implementačné detaily pred užívateľom. Výhoda transparentného distribuovaného databázového systému spočíva vo vysokej úrovni podpory, ktorú poskytuje pre vývoj komplexných aplikácii.

Dátová nezávislosť

Dátová nezávislosť je základný prvok transparencie v kontexte distribuovaných databázových systémov. Referuje na odolnosť užívateľských aplikácii voči zmenám v definícii a organizácii dát. Existujú dve úrovne popisu definície dát. Prvý level je logická štruktúra, nazývaná tiež definícia schémy. Druhý level je fyzická štruktúra dát. Z tohto dôvodu hovoríme o logickej a fyzickej dátovej nezávislosti. Logická dátová nezávislosť znamená odolnosť užívateľských aplikácii voči zmenám v logickej štruktúre databázy, napríklad schémy databázy. Na druhú stranu, fyzická dátová nezávislosť znamená schovanie detailov ukladacej štruktúry pred užívateľskými programami. V takom prípade užívateľova aplikácia nemusí byť zmenená pri zmene organizácie dát.

Sieťová transparencia

V centralizovaných databázových systémoch, jediný zdroj, ktorý musí byť schovaný pred užívateľom, je systémový ukladací priestor. V distribuovanom prostredí pribúda navyše ďalší zdroj, ktorým je sieť. Užívateľ by mal byť chránený pred prevádzkovými detailmi siete. V ideálnom prípade by mala byť pred užívateľom schovaná existencia siete. V takom prípade by nebol rozdiel medzi databázovými aplikáciami, ktoré bežia na centralizovanom, alebo distribuovanom databázovom systéme. Tento typ transparencie sa nazýva sieťová transparencia. Na sieťovú transparenciu možno pozerať z dvoch pohľadov. Prvým je pohľad z poskytovaných služieb, pri ktorom je žiadané mať jednotný spôsob, ktorým sa pristupuje ku službám. Druhým je pohľad z perspektívy distribuovaného databázového systému, pri ktorom užívateľ nemusí špecifikovať, kde budú dáta uložené.

Replikačná transparencia

Z hľadiska výkonu, spoľahlivosti a dostupnosti je žiadané, aby boli dáta replikované do ďalších uzlov v sieti. Replikácia je výhodná k vzhľadom k výkonu, pretože dáta, ku ktorým užívateľ pravidelne pristupuje, môžu byť umiestnené bližšie, čím sa zvyšuje lokalitu odkazu.

V prípade, že jeden z uzlov v sieti zlyhá, kópia dát je stále dostupná v inom uzle. Rozhodnutie, kedy dáta replikovať a do koľkých uzlov, závisí na požiadavkách užívateľskej aplikácie.

Ak sú dáta replikované, nastáva problém, či by si mal byť užívateľ vedomý, že existuje viacero kópii dát. Z pohľadu užívateľa je lepšie, ak o replikácii nevie a zaobchádza zo systémom, akoby existovala iba jedna kópia dát. Z pohľadu systému je však problém zložitejší. Ak je zodpovednosť špecifikovať, že určitá akcia má byť vykonaná na viacerých kópiách, prenesená na užívateľa, tak je distribuovaný databázový systém jednoduchší. Na

druhú stranu systém stráca určitú mieru flexibility. Dôvodom je, že užívateľská aplikácia rozhoduje o počte kópii dát a nie systém. Akákoľvek zmena v rozhodnutiach ovplyvňuje užívateľskú aplikáciu a preto znižuje nezávislosť dát. Preto je žiadané, aby replikačná transparencia bola štandardnou súčasťou distribuovaných databázových systémov.

Fragmentačná transparencia

Fragmentačná transparencia je posledná forma transparencie v kontexte distribuovaných databázových systémov. Obecne je žiadané rozdeliť každý databázový vzťah do menších fragmentov a s každým fragmentom zaobchádzať, ako so samostatným databázovým objektom. Tento prístup je výhodný z ohľadom na výkon, dostupnosť a spoľahlivosť. Existujú dva typy fragmentácie. Prvý typ sa nazýva horizontálna fragmentácia. V tomto prípade je relácia rozdelená do množiny pod-relácii, z ktorých každá obsahuje podmnožinu n-tíc originálnej relácie. Druhý typ sa nazýva vertikálna fragmentácia, kde každá pod-relácia je definovaná na podmnožine atribútov originálnej relácie.

Keď sú databázové objekty fragmentované, je potreba sa vysporiadať s dotazmi, ktoré sú špecifikované na celých reláciách, ale musia byť vykonané na pod-reláciách. Typicky je požadovaný preklad z takzvaného globálneho dotazu na niekoľko fragmentových dotazov. Preklad dotazov je základným problémom v oblasti fragmentačnej transparencie.

2.2.2 Spoľahlivý prístup k dátam

Distribuované databázové systémy sú určené k zlepšeniu spoľahlivosti prostredníctvom replikácie dát. Vďaka tomu odstraňujú jediný bod zlyhania (single point of failure). Zlyhanie jedného uzla alebo komunikačného spojenia, čo spôsobí nedostupnosť jedného, alebo viacerých uzlov, neznamená zlyhanie celého systému.

2.2.3 Zlepšenie výkonu

Zlepšenie výkonu u distribuovaných databázových systémov má typicky dve príčiny.

Prvým dôvodom je, že dáta môžu byť uložené v blízkosti ich použitia, čo je tiež známe pod pojmom lokalizácia dát. Tento fakt má dve potencionálne výhody:

1. Keďže každý uzol spracováva iba časť databázy, záťaž procesoru a požiadavky na vstupno-výstupné operácie nebudú tak veľké, ako v prípade centralizovaných databáz.

2. Lokalizácia dát znižuje oneskorenie pri vzdialenom prístupe, ktoré sa väčšinou vyskytuje pri rozsiahlych sieťach.

Druhým dôvodom je využitie paralelizmu v dotazoch. Existujú dve možnosti, ako paralelizovať dotazy. Prvá možnosť je vykonávať viac dotazov zároveň a druhá spočíva v rozdelení dotazu na pod-dotazy, z ktorých každý je vykonaný na inom uzle, pričom pristupuje k inej časti databázy.

2.2.4 Jednoduchšie rozširovanie systému

Distribuované prostredie je omnoho jednoduchšie prispôsobiť narastajúcej veľkosti databázy. Rozšírenie systému môže byť obyčajne vykonané pridaním výpočtového výkonu a úložného priestoru do siete. Samozrejme nie je možné dosiahnuť lineárneho nárastu výkonu, pretože s pridaním ďalších zdrojov narastajú aj režijné náklady. Stále je však možné dosiahnuť značných vylepšení.

2.3 Komplikácie

Problémy vyskytujúce sa v databázových systémoch naberajú na zložitosti v distribuovanom prostredí. Vďaka vyššej komplexnosti systému vznikajú problémy ovplyvnené hlavne troma faktormi.

Prvým je možný výskyt replikácie v distribuovanom prostredí. Distribuovaná databáza môže byť navrhnutá tak, že celá alebo jej časť, leží v inom uzle v sieti. Možná duplikácia dát je z dôvodu spoľahlivosti a výkonnosti. Distribuovaný databázový systém je potom zodpovedný za výber jednej kópie dát v prípade požiadavku na dáta a za reflektovanie zmien na všetky kópie dát v prípade požiadavku na zmenu.

Druhým faktorom je situácia, keď nastane zlyhanie uzlu alebo spojenia v sieti v momente požiadavku na zmenu. Systém potom musí zabezpečiť, že zmena bude reflektovaná na dáta nachádzajúce sa v postihnutom uzle v momente, keď uzol bude znova dostupný.

Tretím faktorom je, že každý uzol nemôže mať okamžité informácie o akciách vykonaných na iných uzloch. Takže synchronizácia transakcii je o poznanie ťažšia, ako u centralizovaných systémoch.

2.4 CAP teorém

Kvôli možnosti definovať kompromis v distribuovaných systémoch medzi konzistenciou, dostupnosťou a toleranciou voči sieťovým partíciam vznikol takzvaný CAP teorém. Ako prvý predstavil myšlienku kompromisu medzi konzistenciou, dostupnosťou a toleranciou voči partíciam v distribuovaných systémoch Brewer [4] a neskôr Gilbert a Lynch ho formálne dokázali a prezentovali ako CAP teorém [7]. Brewer spočiatku prezentoval CAP teorém v kontexte webovej služby. Táto služba je implementovaná množinou serverov, potenciálne rozložených na geograficky vzdialených miestach. Klienti posielajú požiadavky na službu. Keď server obdrží požiadavok, odošle odpoveď späť. CAP teorém hovorí, že distribuovaný počítačový systém nemôže súčasne zabezpečiť následujúce tri požiadavky:

- konzistencia – požaduje konzistentný pohľad na všetky dáta vo všetkých uzloch distribuovaného systému. Systém musí zabezpečiť, že operácie sú atomické a všetky zmeny budú súčasne prenesené do všetkých uzlov.
- dostupnosť – požaduje, aby systém odpovedal na každý požiadavok, aj v prípade výskytu poruchy. Odpoveď musí prísť na každý typ požiadavku. V reálnych systémoch je výrazne pomalá odpoveď rovnako neprípustná, ako žiadna odpoveď.
- tolerancia voči sieťovým partíciam – požaduje, aby bol systém odolný voči stratám správ medzi uzlami. Partícia v sieti nastane, ak sa preruší spojenie medzi ľubovoľnými uzlami v sieti, čo má za následok stratu všetkých správ medzi uzlami.

Podľa CAP teorému je možné súčasne zabezpečiť maximálne dva z uvedených požiadavkov, pre ľubovoľný distribuovaný systém.

V reálnych systémoch sú všetky tri požiadavky žiadané. Každý distribuovaný systém však musí učiniť kompromis a vypustiť jeden požiadavok. Preto existujú následujúce typy systémov podľa požiadavkov, ktoré splňujú.

2.4.1 Konzistencia a dostupnosť (CA)

Distribuované systémy tohto typu poskytujú konzistentné a dostupné služby v prípade neparticiovanej siete. V prípade výskytu sieťovej partície, systémy môžu byť nekonzistentné. Sú tiež známe pod pojmom vysoko-dostupné konzistentné systémy (HA consistency).

Väčšina tradičných relačných databázových systémov využíva tento prístup. Tieto systémy využívajú replikáciu a transakčné protokoly ako napríklad *Two-phase commit protocol* na zabezpečenie dostupnosti a konzistencie. Ak vznikne sieťová partícia, môže nastať nekonzistencia. Táto situácia sa rieši pomocou konsenzusného protokolu. Do vyriešenia konfliktu nie sú uzly schopné odpovedať na požiadavky od klientov.

2.4.2 Konzistencia a tolerancia voči sieťovým partíciam (CP)

Distribuované systémy tohto typu poskytujú konzistentné služby. Konzistencia je zaručená aj v prípade výskytu sieťovej partície. Na druhú stranu uzly zasiahnuté sieťovou partíciou nemusia byť schopné odpovedať na požiadavky, pretože nie sú schopné dohodnúť sa s ostatnými uzlami na dohode. Preto tieto systémy nemusia byť vždy dostupné. Kombinácia týchto požiadavkov je tiež známa pod pojmom vynútená konzistencia.

Tento typ systémov sa využíva v situáciach, keď je nutné dosiahnuť úplnú konzistenciu a aplikácia musí byť distribuovaná. Príkladom sú napríklad bankové systémy. Často sú založené na relačnom modeli dát a využívajú sofistikované algoritmy, aby zaručili konzistenciu aj v prípade sieťovej chyby. Príkladom je protokol *Paxos* [10].

2.4.3 Dostupnosť a tolerancia voči sieťovým partíciam (AP)

Distribuované systémy tohto typu poskytujú svoje služby aj v prípade výskytu sieťovej partície. Dostupné uzly však v takom prípade môžu poskytovať dočasne nekonzistentné dáta. Táto kombinácia požiadavkov je tiež známa pod pojmom eventuálna konzistencia.

Pre radu aplikácii je výhodné tolerovať eventuálnu konzistenciu výmenou za vysokú dostupnosť aj v prípade výskytu sieťovej partície. Príkladom je dokumentová databáza CouchDB.

2.4.4 Alternatíva ku CAP teorému

S nárastom záujmu o *NoSQL* databázy sa objavila kritika na CAP teorém. Ako jeho hlavný nedostatok je považovaný zjednodušený pohľad na chyby v sieti, pretože uvažuje iba sieťové zlyhania. Konkrétne sa jedná o predčasnú stratu konzistencie, ako dôsledok chyby v sieti, na čo poukázal Stonebraker [14].

Abadi [1] poukázal na to, že CAP teorém rieši iba určité zlyhania v distribuovaných databázových systémoch a nevenuje pozornosť systému v normálnej prevádzke. Preto navrhol alternatívny model zvaný PACELC, ktorý lepšie postihuje kompromis medzi konzistenciou a dostupnosťou. Model je možné popísať následovne – systém sa v prípade výskytu sieťovej partície (P) zamerá na dostupnosť (A) alebo konzistenciu (C), inak (E) pri bežnej prevádzke sa zamerá na latenciu (L), alebo konzistenciu (C). Vo výsledku môžu byť systémy popísane s väčšou precíznosťou. Napríklad eventuálne konzistentné systémy (AP v kontexte CAP), môžu byť rozdelené do kategórii PA/EL alebo PA/EC. Oba typy fungujú rovnako v prípade výskytu sieťových partícii, kedy sa vzdajú konzistencie na úkor dostupnosti. V normálne prevádzke prvý typ uprednostní latenciu na úkor konzistencie, zatiaľ čo druhý typ opačne.

Kapitola 3

CouchDB

Existuje viacero prístupov v spôsobe ukladania dát v databázových systémoch. Najviac využívaný typ databázy sú relačné databázy. Tieto databázy sú založené na relačnom modeli dát. Systém, ktorý interaguje s užívateľmi, inými programami a manipuluje so samotnou databázou sa nazýva systém riadenia bázy dát (SRBD). Využíva jazyk SQL na dotazovanie a udržovanie databáze. SRBD manipuluje s databázou pomocou transakcií. Transakcia musí spĺňať následujúce vlastnosti:

- atomicita – transakcia buď uspeje celá alebo vôbec
- konzistencia – databáza je v správnom stave pred začiatkom aj po skončení každej transakcie
- izolovanosť – transakcia je nezávislá od inej transakcie, takže ak chcú dve transakcie súčasne modifikovať rovnaké dáta, jedna musí počkať, kým druhá skončí
- trvanlivosť – ak transakcia skončí úspešne, zmeny pretrvajú, aj keď nastane chyba systému

Druhý prístup k ukladaniu dát upúšťa od relačného modelu. Tento prístup je známy pod pojmom NoSQL [8]. V posledných rokoch sa NoSQL databázy tešia veľkej popularite. Ich spoločným rysom je, že postrádajú relačnú schému dát a ich dátový model je jednoduchý.

Časť NoSQL databázy pracuje na princípe asociatívneho poľa. Každej časti dát, ktoré sa ukladajú do databáze je pridelený kľúč. Pomocou tohto kľúča je možné pristúpiť do databázy a získať príslušné dáta. Tento prístup sa vyznačuje extrémne nízkou odozvou, aj v prípade vysokej záťaže. Medzi zástupcov tohto typu databáz patrí *Cassandra*, *Aerospike*, *Redis* a mnohé iné.

Medzi ďalšie druhy NoSQL databázy môžeme zaradiť dokumentové databázy. Dokumentom sa v tomto prípade myslí skupina dát, ktoré kódujú určité informácie. Každý dokument môže obsahovať rôzny počet polí rozličnej dĺžky. Dokumentové databázy sú vhodné pre veľké množstvo polo - štrukturovaných dát. Takisto sú vhodné pre ukladanie objektov z objektovo orientovaného programovacieho paradigmatu. Medzi zástupcov tohto typu databáz patrí *CouchDB*, *MongoDB* a iné.

3.1 Dokumentovo orientovaná databáza

CouchDB [17] je open source dokumentová databáza, napísaná vo funkcionálnom jazyku *Erlang*. Vznikla v roku 2005 a neskôr v roku 2008 bola zaradená medzi *Apache* projekty.

Jazyk *Erlang* vznikol za účelom umožniť vytvárať distribuované, vysoko dostupné aplikácie, ktoré sú tolerantné voči zlyhaniam. Preto je možné nasadiť CouchDB do distribuovaných systémov, ktoré musia byť vysoko škálovateľné. CouchDB je tiež dobre stavaná pre moderné webové a mobilné aplikácie.

CouchDB uchováva dáta ako *JSON* (JavaScript Object Notation) dokumenty. Poskytuje *RESTful HTTP API* pre prístup k dokumentom. Taktiež umožňuje dotazovanie a transformáciu dokumentov pomocou *JavaScriptu*. CouchDB umožňuje efektívne distribuovať dáta pomocou inkrementálnej replikácie. Podporuje *master - master* replikáciu s automatickou detekciou konfliktov.

Dokument je základná jednotka pre ukladanie dát v CouchDB. Obsahuje ľubovoľný počet polí, prílohy a metadáta, ktoré sú udržované databázovým systémom. Polia v dokumente musia mať unikátne názvy a ich hodnoty môžu byť rôznorodého typu. V CouchDB neexistuje žiadny limit na veľkosť textu alebo počet polí.

Každý dokument obsahuje jedinečnú položku v rámci databáze `_id`, ktorá identifikuje dokument. Táto hodnota môže byť ľubovoľný textový reťazec, ale doporučená je *UUID* (Universally Unique IDentifier), ktorú automaticky používa CouchDB.

Ďalšia položka, ktorú obsahuje každý dokument je položka `_rev`. Slúži na spravovanie verzií dokumentov. Keď CouchDB akceptuje zmenu dokumentu automaticky vygeneruje novú hodnotu `_rev`. V prípade, že chce užívateľ aktualizovať alebo zmazať dokument, CouchDB očakáva od užívateľa revíziu dokumentu. Ak medzitým iný užívateľ zmení tento dokument v databáze a prvý užívateľ neposkytne najnovšie `_rev`, CouchDB odmietne uložiť dokument a vráti chybovú hlášku o konflikte. Ak užívateľ poskytne hodnotu najnovšej revízie, CouchDB akceptuje zmenený dokument a vráti nové `_rev`.

Tento revizný systém, ktorý požíva CouchDB, sa nazýva *MVCC* (Multi-Version Concurrency Control). Ak užívateľ zmení obsah dokumentu, vytvorí sa nová verzia tohto dokumentu a uloží sa. Výsledok je, že existujú dve verzie rovnakého dokumentu v databáze. *MVCC* neuzamyká obsah databázy pri požiadavkoch od klientov. Preto môže viacero klientov súčasne pristupovať k dokumentu v databáze, zatiaľ čo ďalší klient môže uložiť novú verziu rovnakého dokumentu, bez toho, aby musel čakať na dokončenie predchádzajúcich požiadavkov. Súbežne zápisy do jedného dokumentu však nie sú povolené. Preto musí klient poskytnúť najnovšie `_rev` pri úprave dokumentu. Požiadavok na čítanie dokumentu vidí vždy najnovšiu snímku databázy v momente začiatku požiadavku. CouchDB využíva bezstavový *HTTP* protokol, vďaka čomu je databáza schopná obslúžiť veľké množstvo paralélnych požiadavkov.

CouchDB dokumenty môžu obsahovať prílohy. Príloha je identifikovaná názvom a obsahuje akého je typu a veľkosť, ktorú zaberá v bytoch. Môže obsahovať ľubovoľné dáta. Každej prílohe je pridelená vlastná *URL* adresa.

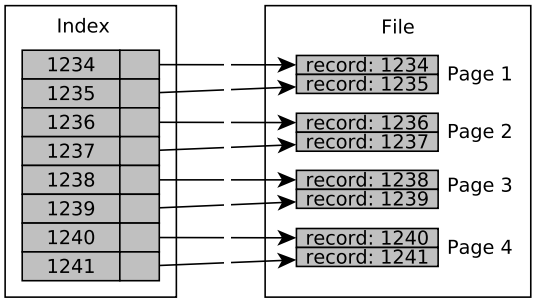
V CouchDB je možné vytvárať lokálne dokumenty, ktoré sa nereplikujú do iných databáz. Slúžia na uchovávanie konfigurácie alebo iných informácií, ktoré sú špecifické pre lokálnu CouchDB inštanciu. Pre prístup k lokálnemu dokumentu užívateľ potrebuje poznať jeho ID, pretože ho nie je možné inak získať.

3.2 B-stromy

CouchDB využíva stromovú dátovú štruktúru *B+ strom* [5] pre indexovanie dokumentov a pohľadov. Je to jedna z variant *B- stromov*. *B- strom* je ideálna dátová štruktúra pre ukladanie veľkého množstva dát, pričom k nim umožňuje veľmi rýchly prístup. Prístup k ľubovoľnému uzlu pre čítanie alebo zápis vyžaduje navštívenie iba pár uzlov.

3.2.1 Databázové indexy

Index [3] je dátová štruktúra, ktorá urýchľuje prístup k dátam. Každý index je vytvorený pre konkrétny vyhľadávací kľúč. Index sa skladá z dvoch položiek. Prvá položka obsahuje záznamy, ktoré sa nazývajú záznamy indexu. Každý záznam obsahuje hodnotu vyhľadávacieho kľúča. Záznamy indexu sú usporiadané podľa hodnoty vyhľadávacieho kľúča. Druhá položka obsahuje odkazy na bloky alebo stránky primárneho súboru pre odpovedajúce záznamy indexu. Na obrázku 3.1 je zobrazený jednoduchý index, kde záznamy indexu predstavujú čísla študentov.

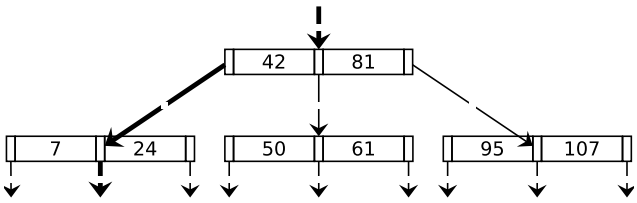


Obr. 3.1: Jednoduchý index

So zväčšovaním veľkosti indexu rastie čas potrebný na vyhľadávanie. Z toho dôvodu sa používajú viac-úrovňové indexy. *B- stromy* využívajú viac-úrovňové indexy usporiadané do stromu.

3.2.2 Operácie na B-stromoch

Vyhľadávanie v *B- stromoch* je generalizácia algoritmu pre vyhľadávanie v binárnych stromoch. Každý uzol v *B- strome* stupňa d obsahuje nanajvýš $2d$ kľúčov a $2d + 1$ ukazateľov. Počet kľúčov v uzloch sa môže líšiť, ale každý uzol musí obsahovať aspoň d kľúčov a $d + 1$ ukazateľov. Na obrázku 3.2 je ukázané vyhľadanie kľúča s hodnotou 15. Vyhľadávanie za-



Obr. 3.2: Vyhľadávanie kľúča s hodnotou 15. Cesta smerujúca k správnemu miestu je zvýraznená.

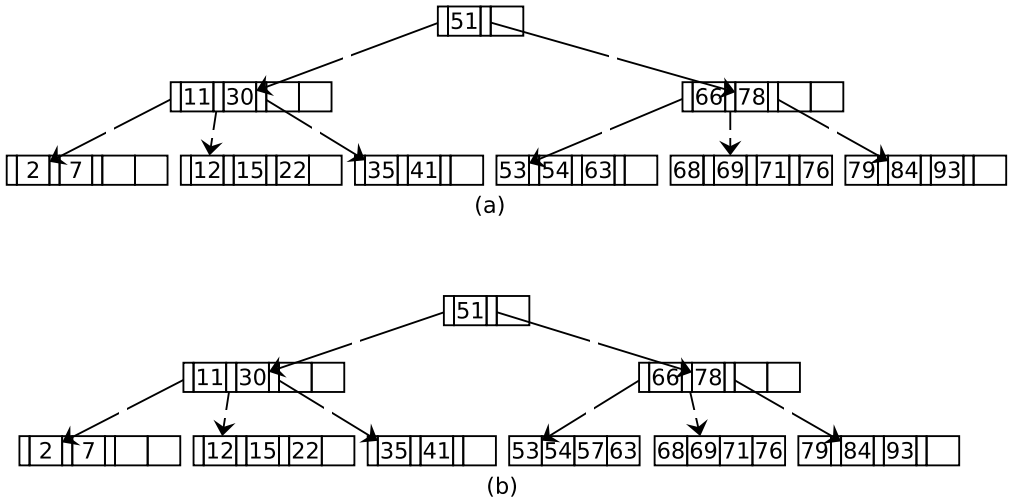
čína od koreňa stromu, zvolením jednej správnej cesty. Hodnota 15 je porovnaná s hodnotou kľúča v uzle. 15 je menej ako 42, preto je zvolená cesta vľavo. Pre hodnoty v rozmedzí 42 až 81 by bola zvolená stredná cesta a pre hodnoty väčšie, ako 81 by bola zvolená pravá cesta. Procedúra vyhľadávania sa opakuje pre každý uzol na ceste, až pokiaľ sa v prípade úspechu nájde zhoda, alebo v prípade neúspechu sa dosiahne listový uzol bez hľadaného kľúča.

B- strom je vždy vyvážený. To znamená, že všetky listy v strome, sú v rovnakej vzdialenosti od koreňa. Vkladanie a mazanie kľúčov je vykonané tak, aby strom ostal vyvážený. Najdlhšia cesta v *B- strome* s n kľúčmi obsahuje maximálne $log_d n$ uzlov, kde d je stupeň stromu. Udržovanie stromu vyváženým predstavuje vysoký potenciál pre úsporu času. Vyvažovanie stromu ale stojí procesorový čas, preto úspora času pri vyhľadávaní musí byť väčšia, ako čas potrebný na vyváženie stromu.

Vkladanie kľúča s hodnotou s hodnotou 57 ilustruje obrázok 3.3. Strom je stupňa $d = 2$, preto každý uzol obsahuje d až $2d$ kľúčov. V každom uzle sa musí nachádzať indikátor o aktuálnom počte kľúčov. Tento indikátor nie je kvôli prehľadnosti na obrázku zobrazený. Vkladanie nového kľúča prebieha v dvoch krokoch:

1. nájdenie správneho listu od koreňa pre vloženie kľúča
2. vloženie kľúča a vyváženie stromu od listu smerom ku koreňu

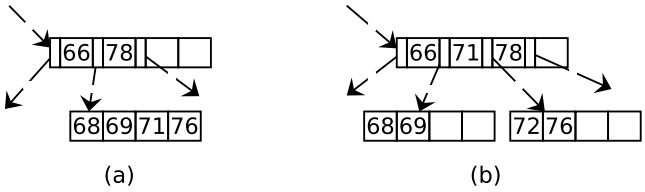
Na obrázku 3.3 a) je možné vidieť, že pri vkladaní hodnoty 57 vyhľadávanie skončí neúspešne na štvrtom liste zľava. V tomto liste je ešte voľné miesto a preto nová hodnota jednoducho vložená na odpovedajúce miesto v liste. Výsledok ukazuje obrázok 3.3 b).



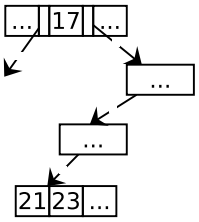
Obr. 3.3: (a) *B- strom* rádu 2. (b) Rovnaký *B- strom* po vložení kľúča 57.

Ak by sme chceli vložiť hodnotu 72, nastali by komplikácie, pretože odpovedajúci list je plný. Vždy, keď má byť kľúč vložený do obsadeného uzlu, nastane rozdelenie. Rozdelenie uzlu je ukázané na obrázku 3.4. Je potrebné rozmiestniť $2d + 1$ kľúčov. d najmenších kľúčov je umiestnených do jedného uzla, d najväčších kľúčov je umiestnených do druhého uzla. Ostávajúci kľúč je presunutý do rodičovského uzla, kde slúži ako rozdeľovač. Zvyčajne je v rodičovskom uzle voľné miesto, takže procedúra vkladania končí. V situácii, keď je rodičovský uzol plný, procedúra rozdelenia sa opakuje. V najhoršom prípade sa rozdelenie propaguje až ku koreňu a v tom prípade sa hĺbka stromu zvýši o jeden level.

Mazanie kľúča v *B- strome* zahŕňa nájdenie správneho uzla. Kľúč sa môže nachádzať v listovom alebo nelistovom uzle. Ak sa kľúč nachádza v nelistovom uzle, je potrebné nájsť následujúci kľúč a presunúť ho na uvoľnenú pozíciu. Následujúci kľúč sa nachádza v najľavejšom liste pravého podstromu z práve uvoľnenej pozície. Obrázok 3.5 znázorňuje postup mazania kľúča s hodnotou 17.

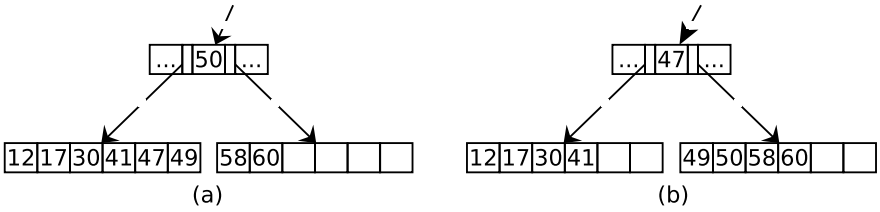


Obr. 3.4: (a) List a jeho predok v *B- strome*. (b) Rovnaký podstrom po vložení kľúča 72.



Obr. 3.5: Mazanie kľúča s hodnotou 17 vyžaduje, že následujúci kľúč s hodnotou 21 bude presunutý na uvoľnenú pozíciu.

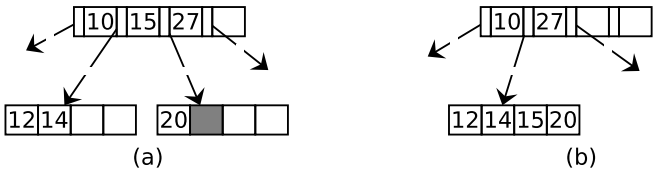
Po presunutí kľúča do rodičovského uzlu je potrebné skontrolovať, či sa v liste nachádza aspoň d kľúčov. Ak sa v liste nachádza menej, ako d kľúčov, je potrebné preusporiadať kľúče v uzloch. Pre nápravu stačí zobrať jeden kľúč zo susedného listu. Táto operácia by ale vyžadovala dva prístupy k sekundárnemu úložisku, a preto je výhodnejšie rozdeliť ostávajúce kľúče medzi dva susedné uzly. Tento postup zmierni cenu postupného mazania z rovnakého uzlu. Rozdelenie kľúčov medzi dva susedné uzly je možné, iba ak je k dispozícii aspoň $2d$ kľúčov na distribúciu. Obrázok 3.6 znázorňuje rozdelenie kľúčov.



Obr. 3.6: (a) Časť *B- stromu* pred rozdelením kľúčov. (b) Časť *B- stromu* po rozdelení kľúčov do susedných uzlov.

V prípade, že ostáva menej, ako $2d$ kľúčov, je potrebné vykonať spojenie uzlov. Počas spájania uzlov sú kľúče preskupené do jedného uzla, pričom druhý uzol je odstránený. Keďže ostáva iba jeden uzol, kľúč v rodičovskom uzle, ktorý oddeľoval dva uzly, je nepotrebný. Preto sa jednoducho presunie do ostávajúceho listu. Na obrázku 3.7 je znázornené spojenie uzlov.

Rovnako, ako v prípade rozdelenia uzlu, tak aj spojenie uzlov sa môže propagovať smerom ku koreňu. Propagácia nastáva v prípade, keď v uzle po odobratí oddeľovacieho kľúča v dôsledku spojenia dvoch uzlov ostane menej, ako d kľúčov. V tom prípade je nutné preu-



Obr. 3.7: (a) Mazanie spôsobujúce spojenie uzlov. (b) *B- strom* po vybalancovaní.

sporiadanie, prípadne ďalšie spojenie uzlov. V najhoršom prípade sa spojenie propaguje až ku potomkom koreňa, čo spôsobí vytvorenie nového koreňa a zníženie hĺbky stromu o jedna.

3.2.3 Ceny operácií

Okrem koreňa má každý uzol v *B- strome* aspoň d priamych potomkov, pretože každý uzol obsahuje d až $2d$ kľúčov. Koreň má aspoň dvoch priamych potomkov. Preto počet uzlov v hĺbkach 0, 1, 2,... musí byť aspoň 2, $2d$, $2d^2$,.... Všetky listy ležia v rovnakej hĺbke h, takže existuje

$$\sum_{i=0}^{h} d^i = \frac{d^h - 1}{2d - 1}$$

uzlov, z ktorých každý obsahuje aspoň d kľúčov. Výška stromu s n kľúčmi je preto obmedzená vzťahom

$$2d \cdot \frac{d^h - 1}{d - 1} \leq n$$

kde po úprave dostaneme

$$h \leq \log_d \frac{n+1}{2}$$

Takže cena vyhľadávania v *B- strome* rastie logaritmicky s veľkosťou súboru.

Vkladanie a mazanie v *B- strome* môže vyžadovať o jeden prístup k disku navyše oproti vyhľadávaniu. Celkovo sú ceny maximálne zdvojené, takže výška stromu stále dominuje. Preto v *B- strome* rádu d pre súbor s n záznamami je cena vkladania a mazania v najhoršom prípade úmerná $log_d n$.

3.2.4 B+ varianta

V tejto variante *B- stromov* sa všetky kľúče nachádzajú v listoch. Ostatné uzly obsahujú indexy, pomocou ktorých je možné veľmi rýchlo pristúpiť k požadovaným kľúčom. Indexy sú usporiadané do *B- stromu*. Listové uzly sú viazané do jednosmerného zoznamu, aby umožňovali sekvenčné vyhľadávanie. Vyhľadávanie postupuje od koreňa, cez indexovú časť smerom ku listom. Keďže všetky kľúče sa nachádzajú v listoch, na hodnotách v indexovej časti nezáleží, pokiaľ navigujú k správnym listom.

Mazanie kľúča v *B+ strome* je jednoduché, keďže všetky kľúče sa nachádzajú v listoch. Ak v liste ostane po zmazaní kľúča aspoň d kľúčov, nie je potrebné upravovať indexovú časť. V opačnom prípade nastane procedúra spojenia, alebo presunutia, ktorá spôsobí úpravu indexovej časti, prípadne aj listov.

Vkladanie a vyhľadávanie v *B+ stromoch* je takmer identické ako vkladanie a vyhľadávanie v *B stromoch*. Ak sa list rozdelí na dva z dôvodu preplnenia, namiesto presunutia prostredného kľúča do rodičovského uzla sa presunie iba kópia kľúča. Skutočný kľúč ostane na správnej pozícii v liste. V prípade vyhľadávania procedúra nezastaví, ak sa kľúč v indexe rovná hľadanej hodnote, ale pokračuje najbližším pravým ukazateľom smerom k listu, kde sa nachádza skutočný kľúč.

Ceny operácii v *B+ stromoch* ostávajú na logaritmickej zložitosti. Výhoda spočíva v operácii získania následujúceho kľúča, ktorá vyžaduje najviac jeden prístup k disku, oproti $log_d n$ pri *B- stromoch*. Takže *B+ stromy* uľahčujú sekvenčný prístup k súboru. Navyše pri sekvenčnom spracovaní žiadny uzol nebude sprístupnený viac než raz, takže iba jeden uzol musí byť dostupný v hlavnej pamäti.

3.3 Eventuálna konzistencia

CouchDB je flexibilná databáza, ktorá sa dobre prispôsobuje rastu dát a úpravám aplikácii, ktoré ju využívajú. Ďalšia silná stránka je, že umožňuje jednoduchú tvorbu škálovateľných, distribuovaných systémov. CouchDB sa líši od iných systémov tým, že akceptuje eventuálnu konzistenciu, narozdiel od uprednostnenia úplnej konzistencie na úkor dostupnosti.

V distribuovaných systémoch je potrebné synchronizovať jednotlivé databázy, aby obsahovali rovnaké dáta. V prípade, že je kritické, aby všetci klienti videli konzistentné dáta, užívatelia jedného uzlu budú musieť čakať na potvrdenie ľubovolného iného uzlu v sieti, aby mohli čítať alebo zapisovať dáta. V tomto prípade má konzistencia prednosť pred dostupnosťou.

Na druhú stranu, ak je priorita dostupnosť, môžeme dovoliť klientom zapisovať do jedného uzlu bez potreby čakať na potvrdenie od iného uzla. Ak databáza vie, ako sa vysporiadať s rozoslaním dát do ostatných uzlov, dosiahneme takzvanú eventuálnu konzistenciu zároveň s vysokou dostupnosťou. Pre mnohé aplikácie je toto žiadaná vlastnosť.

CouchDB využíva inkrementálnu replikáciu na propagovanie zmien do ostatných uzlov v sieti. Týmto spôsobom dosahuje eventuálnu konzistenciu. Je to fundamentálne odlišný prístup, ako algoritmy založené na konsenzuse, alebo spôsobe, akým fungujú relačné databázy.

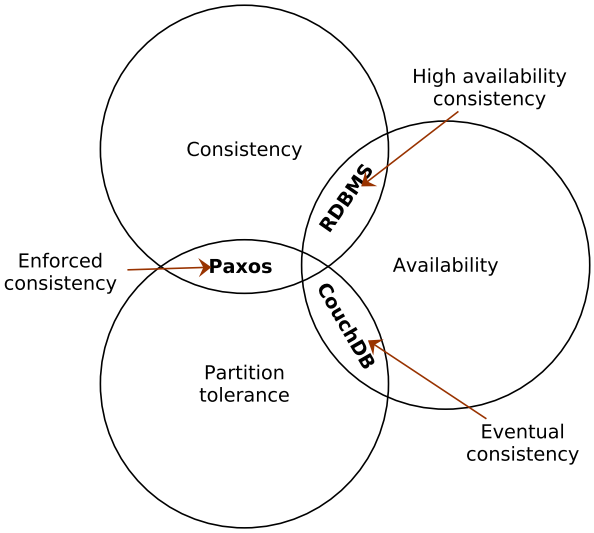
Na obrázku 3.8 je možné vidieť prístup, akým sa CouchDB a iné systémy stavajú ku CAP teorému.

Vďaka prístupu, ktorý CouchDB využíva, je možné jednoducho tvoriť aplikácie, ktoré obetujú okamžitú konzistenciu výmenou za vysokú rýchlosť a jednoduchú distribúciu dát.

3.4 Synchronizácia

Replikácia je ikrementálny spôsob na synchronizáciu dvoch databáz. Na konci procesu replikácie, sa všetky aktívne dokumenty zo zdrojovej databáze nachádzajú aj v cieľovej databáze. Rovnako všetky zmazané dokumenty v zdrojovej databáze sú zmazané aj v cieľovej databáze (ak v nej existujú). Proces replikácie kopíruje iba najnovšie revízie dokumentov. Replikácia umožňuje distribuovať dáta do viacerých uzlov, alebo datacentier, ale aj presunúť dáta bližšie ku klientom.

Replikácia sa spustí pridaním dokumentu do databáze zvanej *_replicator*, kde každý dokument predstavuje jeden replikačný proces. Je to databáza, ako každá iná, s tým rozdielom,



Obr. 3.8: CAP teorém

že uloženie dokumentu spustí replikáciu. Zrušiť replikáciu je možné zmazaním dokumentu alebo nastavením položky dokumentu *cancel* na hodnotu *true*.

CouchDB počas replikácie porovná zdrojovú a cieľovú databázu, aby sa zistilo, ktoré dokumenty sa líšia. Zmeny sú odoslané do cieľovej databáze v dávkach, pričom sa môžu objaviť konflikty. Keďže zmazanie dokumentu je reprezentované novou revíziou, zmena bude prenesená do cieľovej databáze. Proces replikácie skončí, keď sa prenesú všetky zmeny. Ak je nastavená položka *continuous*, CouchDB bude čakať na nové zmeny, až pokiaľ nebude proces replikácie zrušený. Počas replikácie sa v cieľovej databáze vytvárajú kontrolné dokumenty, ktoré slúžia na správnu obnovu procesu replikácie v prípade zlyhania. Vďaka ním bude reštartovaná replikácia pokračovať v bode, kde skončila.

Jeden replikačný proces prenáša dáta iba jedným smerom. Pre *master – master* replikáciu, je potrebné nastaviť dve replikácie v opačných smeroch.

V CouchDB je možné vytvoriť filter, ktorý určí, ktoré dokumenty sa majú replikovať. Tento filter sa následne vyhodnotí pre každý zmenený dokument. Dokument sa bude replikovať, iba ak daný filter vráti *true*. Ďalšia možnosť je využiť lokálne dokumenty, ktoré sa nikdy nereplikujú.

3.4.1 CouchDB replikačný protokol

CouchDB replikačný protokol slúži pre synchronizáciu *JSON* dokumentov medzi dvoma inštanciami prostredníctvom *HTTP/1.1* s využitím *CouchDB REST API* [16]. Referenčná implementácia protokolu je napísaná v *Erlangu* a nachádza sa v module *couch_replicator*. Na obrázku 3.9 je znázornený vývojový diagram replikačného algoritmu.

3.4.2 Overenie peerov

Replikátor musí zabezpečiť, že zdrojová a cieľová databáza existujú. Vykoná overenie a v prípade, že cieľová databáza neexistuje, replikátor sa ju môže pokúsiť vytvoriť. Vytvorenie môže zlyhať kvôli nedostatočným právam. V takom prípade je vrátená chyba `401`

`Unauthorized` alebo `403 Forbidden`. Ak neexistuje zdrojová, alebo cieľová databáza, replikácia by mala byť ukončená s chybovou hláškou.

3.4.3 Získanie informácii od peerov

Replikátor musí získať základné informácie z oboch databáz. Odpoveď musí obsahovať *JSON* objekt s dvoma povinnými položkami:

- `instance_start_time` – časová známka, kedy bola databáza vytvorená, vyjadrená v milisekundách
- `update_seq` – aktuálne sekvenčné ID, ktoré reprezentuje zmenu urobenú v databáze

3.4.4 Nájdenie spoločného predka

Replikátor musí pred začiatkom replikácie vygenerovať ID replikácie. Toto ID je použité na záznam histórie a obnovenie prerušenej replikácie.

Po vygenerovaní ID by mal replikátor získať replikačné záznamy zo zdrojovej a cieľovej databáze. Replikačný záznam obsahuje následujúce položky:

- `history` – história replikácie, ako pole objektov. Obsahuje mimo iné, povinné položky `recorded_seq`, `recorded_seq` využívané pri porovnávaní replikačných záznamov.
- `replication_id_version` – verzia replikačného protokolu
- `session_id` – unikátne ID poslednej relácie
- `source_last_seq` – posledný spracovaný kontrolný bod

Replikátor môže obdržať odpoveď `404 Not Found`. V takom prípade replikátor začne novú replikáciu, pretože neexistuje žiadna súčasná replikácia.

V prípade, že replikátor získa replikačné záznamy zo zdrojovej aj cieľovej databázy, určí ich spoločného predka pomocou nasledovného algoritmu:

- Porovnaj hodnoty `session_id` do chronologicky poslednej relácie – ak sa zhodujú, zdrojová a cieľová databáza majú spoločnú replikačnú históriu. Využi hodnotu `source_last_seq`, ako štartovný bod replikácie.
- V prípade nezhody v hodnotách `session_id`, iteruj cez položku `history`, pre nájdenie poslednej spoločnej hodnoty `session_id`. Využi hodnotu `recorded_seq`, ako štartovný bod replikácie.

Ak zdrojová a cieľová databáza nemajú spoločného predka, replikátor musí spustiť plnú replikáciu.

3.4.5 Nájdenie zmenených dokumentov

Po získaní štartovného bodu, replikátor získa zmeny v zdrojovej databáze. Požiadavok na získanie zmien musí obsahovať následujúce parametre:

- `feed` – definuje spôsob zmien (`continuous` pre inkrementálnu replikáciu, inak `normal`)

- `style=all_docs` – zdrojová databáza musí poskytnúť najnovšie revízie[1] pre každý dokument v odpovedi
- `heartbeat` – u inkrementálnej replikácie určuje periódu replikácie v milisekundách
- `since` – definuje štartovný bod replikácie (ak bol nájdený), v prípade plnej replikácie môže byť 0 alebo vynechaný

Požiadavok môže ďalej obsahovať parameter `filter`, na umožnenie filtrovania dokumentov na replikáciu.

Načítanie všetkých zmien naraz nemusí byť optimálne vzhľadom na využívanie prostriedkov. Preto je doporučené načítať zmeny postupne po malých častiach. Konkrétna veľkosť nie je definovaná, lebo závisí na dostupných zdrojoch. Veľké dávky dokumentov vyžadujú viac pamäte, ale na druhú stranu redukujú vstupno – výstupné operácie.

Po získaní dávky zmenených dokumentov, replikátor vytvorí *JSON* mapovanie medzi ID jednotlivých dokumentov a odpovedajúcimi revíziami. Výsledné mapovanie pošle do cieľovej databázy. V odpovedi replikátor obdrží mapovanie s revíziami, ktoré sa nenachádzajú v cieľovej databáze. Tieto revízie dokumentov sú požadované na prenesenie zo zdrojovej databázy.

Ak neostávajú žiadne zmeny na spracovanie a replikátor preniesol všetky požadované dokumenty, proces replikácie sa ukončí. Ak replikácia nebola inkrementálna, replikátor môže vrátiť klientovi odpoveď so štatistickými údajmi o replikácii.

3.4.6 Replikovanie zmien

V tomto momente musí replikátor načítať všetky listové revízie dokumentov, ktoré chýbajú v cieľovej databáze. Táto operácia je efektívna s využitím predchádzajúceho kroku, v ktorom sa zistili rozdiely v revíziách medzi zdrojovou a cieľovou databázou. Požiadavok na načítanie zmenených dokumentov obsahuje následujúce parametre:

- `revs=true` – zdrojová databáza musí zahrnúť zoznam všetkých revízii do položky `_revisions` v dokumente
- `open_revs` – obsahuje zoznam požadovaných listových revízii. Ak dokument v požadovanej revízii existuje, musí byť vrátený. V opačnom prípade zdrojová databáza vráti objekt s položkou `missing`, ktorá nadobúda hodnotu chýbajúcej revízie. Ak dokument obsahuje prílohy, zdrojová databáza vráti informácie iba pre tie prílohy, ktoré boli zmenené od požadovanej revízie.
- `latest=true` – zaistí, že zdrojová databáza vráti najnovšiu revíziu dokumentu bez ohľadu na revíziu špecifikovanú v `open_revs`. Pomocou tohto parametru je riešená časovo závislá chyba (race condition), keď požadovaný dokument môže byť zmenený počas načítania dokumentov.

Zdrojová databáza by mala vrátiť odpoveď typu `multipart/mixed`, pokiaľ položka `Accept` v hlavičke požiadavku nepožaduje iný typ. Typ `multipart/mixed` umožňuje zaobchádzať s dátami v odpovedi ako s prúdom, čo je výhodné, pretože odpoveď môže obsahovať viacero dokumentov a príloh. Navyše prílohy sú väčšinou binárne, čo neumožňuje efektívne spracovanie vo formáte *JSON*. Replikátor po obdržaní odpovede umiestni dáta do lokálneho zásobníku. Po zaplnení zásobníku odošle replikátor obsah zásobníku do cieľovej databázy.

[1] Dokument môže mať viacero konfliktných revízii kvôli súbežným zmenám.

Pre odoslanie viacerých dokumentov naraz, replikátor odošle požiadavok do cieľovej databáze obsahujúci *JSON* objekt s následujúcimi položkami:

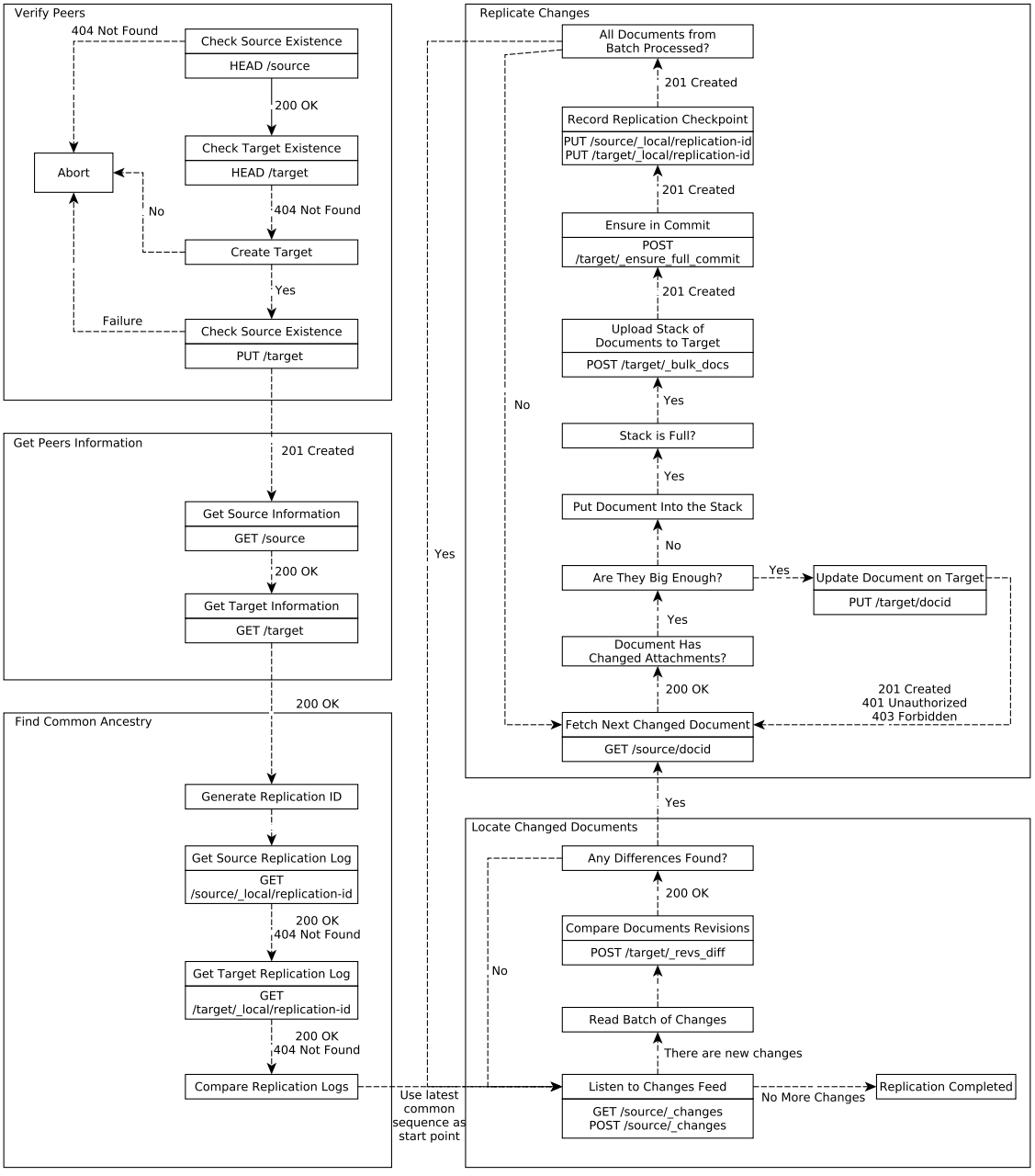
- `docs` – zoznam dokumentov, ktoré je potrebné aktualizovať v cieľovej databáze. Dokumenty musia obsahovať položku `_revisions`, obsahujúcu plnú replikačnú históriu.
- `new_edits` – príznak, ktorý určuje cieľovej databáze uložiť dokument v špecifikovanej revízii, bez generovania novej revízie. Príznak je vždy nastavený na *false*.

Cieľová databáza vráti zoznam dokumentov s odpovedajúcimi statusmi. Ak bolo uloženie dokumentu úspešné, odpovedajúci záznam musí obsahovať položku `ok` s hodnotou `true`. V opačnom prípade obsahuje položky `error` a `reason` s typom chyby a ľudsky čitateľným dôvodom zlyhania. Zlyhanie aktualizácie dokumentu nemusí byť fatálne, pretože cieľová databáza môže mať dôvod na odmietnutie aktualizácie. V takom prípade by sa replikátor nemal pokúšať odosielať dokument znovu, ak na to nemá dobrý dôvod (špeciálny chybový kód).

Po úspešnom odoslaní zmien do cieľovej databázy, replikátor odošle požiadavok, aby zaistil, že všetky odoslané dáta boli zapísané na disk. Cieľová databáza musí vrátiť odpoveď `201 Created` s *JSON* objektom obsahujúcim následujúce položky:

- `instance_start_time` – časová známka, kedy bola databáza vytvorená, vyjadrená v milisekundách
- `ok` – status (vždy `true`)

Po overení úspešnosti replikačného procesu, replikátor aktualizuje replikačné záznamy v zdrojovej aj cieľovej databáze podľa aktuálneho stavu replikácie. Replikátor môže ďalej čakať na nové zmeny. Ak sa nevyskytujú žiadne ďalšie zmeny, proces replikácie sa ukončí. V prípade inkrementálnej replikácie, replikátor musí ďalej čakať na nové zmeny v zdrojovej databáze.



Obr. 3.9: Vývojový diagram replikačného algoritmu.

Kapitola 4

Document management system

V súčasnej dobe organizácie zápasia so zvyšujúcim sa objemom informácii v podobe záznamov a dokumentov. Problém spracovania dokumentov je po dlhú dobu rozpoznaný ako kritický aspekt v efektivite podniku [9]. Spravovanie dokumentov začne byť ešte viac náročné, keď zahrnieme viacerých aktérov s rôznymi rolami, úlohami a zodpovednosťou. Situácia sa ešte viac zhorší, ak zahrnieme distribuované prostredie do systému.

Väčšina organizácii sa musí vysporiadať aj s papierovými dokumentmi, ako napríklad so zmluvami, zápismi a ďalšími. Tieto dokumenty sú väčšinou skladované v krabiciach a regáloch. V prípade, že niekto chce nájsť jeden z týchto dokumentov, ktorý bol vytvorený pred mnohými rokmi, môže to zabrať hodiny až dni. V najhoršom prípade nebude dokument nájdený vôbec. Ďalšou náročnou úlohou je zabezpečiť, aby vytvorený dokument bol v najaktuálnejšom stave. Tieto klasické metódy práce s informáciami sú neefektívne, náchylné na chybu a postrádajú dostatočnú kontrolu nad informáciami. To vedie k vyšším nákladom a predstavuje risk pre organizácie, ktoré pôsobia vo vysoko regulovaných odvetviach.

4.1 Základná funkcionalita

DMS pomáha organizáciám spracovávať veľké množstvo informácii. Dobre koncipovaný *DMS* môže zjednodušiť a v niektorých prípadoch automatizovať proces vytvárania, ukladania, vyhľadávania, sledovania a odstránenia dokumentov. Navyše môže zlepšiť produktivitu pracovníka a zvýšiť úroveň bezpečnosti a súkromia.

Spolupráca viacerých užívateľov na dokumentoch je kľúčovou súčasťou *DMS*. Môže mať rôzne podoby, od najjednoduchšej, kde systém blokuje prístup k dokumentu, zatiaľ čo na ňom pracuje iný užívateľ, cez umožnenie paralelnej práce na dokumente s detekciou konfliktov. Niektoré systémy umožňujú spoluprácu na dokumentoch v reálnom čase, kde užívatelia súčasne pracujú na dokumente a okamžite vidia zmeny ostatných užívateľov.

Ďalšou dôležitou funkcionalitou je správa revízií dokumentov, ktorá zaznamenáva všetky vykonané zmeny. Umožňuje užívateľom vidieť staršie verzie dokumentov a prípadne pokračovať v práci vo zvolenom bode. Správa revízií je užitočná pre dokumenty, ktoré sa menia v priebehu času a vyžadujú časté úpravy.

DMS zvyčajne ukladá metadáta v rámci každého dokumentu. Môže sa jednať o časové razítka, identifikáciu užívateľov, ktorí manipulovali s dokumentom a mnohé iné. Metadáta môžu byť získané systémom priamo z dokumentu, alebo môžu byť poskytnuté užívateľom. Systém ďalej môže extrahovať text z dokumentu za účelom identifikácie kľúčových slov

a pre zaradenie dokumentu do kategórii. Získané metadáta sú užitočné pri vyhľadávaní dokumentov.

4.2 Technológie DMS

Každý *DMS* by mal poskytovať základnú funkcionalitu pre spracovanie a spravovanie dokumentov. V následujúcej časti je popísaná množina technológii pre spracovanie dokumentov, organizovaná podľa krokov v životnom cykle dokumentu a funkcie správy dokumentov, ktoré spoločne tvoria infraštruktúru *DMS* [13].

4.2.1 Zachytenie a vytvorenie dokumentov

Jedná sa o technológie na digitalizáciu informácii. Hardware a software digitalizuje papierové dokumenty a následne ho elektronicky spracováva. Skener zachytí obraz, zatiaľ čo algoritmy ho prevedú do digitálnej formy, často s kompresiou, aby sa šetril ukladací priestor.

Následne môže byť digitalizovaný dokument analyzovaný na rozpoznávanie znakov. Súčasný software dokáže zachytiť text v rôznych fontoch, veľkostiach a formátoch. Rozpoznávacie techniky dokážu ďalej rozpoznať hlas, obrázky a vzory v grafike, animáciách a videu.

4.2.2 Ukladanie a organizácia dokumentov

Niekoľko technológii určuje, ako sú dokumenty ukladané a organizované. Primárne sú to architektúra zložených dokumentov, distribuovaný software pre správu úložisk, integrácia dokumentov a databáz a hypertextové odkazy. Veľký význam majú distribuované úložiska, ktoré umožňujú prístup celej organizácie k dokumentovým zdrojom. Dokumenty by mali byť uložené do prehľadnej štruktúry, aby bolo možné s nimi efektívne pracovať.

4.2.3 Architektúra zložených dokumentov

Táto architektúra je potrebná, aby rôzne objekty, ktoré tvoria zložený dokument, boli spracovávané spoločne. Zložený dokument pozostáva z objektov, ktoré môžu byť uložené na rôznych zariadeniach a logicky prepojené pomocou ukazateľov. Môže sa jednať napríklad o textové, grafické, tabuľkové a iné objekty.

4.2.4 Integrácia dokumentov a databáz

Vytváranie dokumentov je neoddeliteľnou súčasťou informačných zdrojov organizácie a vyžaduje integráciu s databázami. Jedným z prístupov je definovať *BLOB* (binary large object), ako časť n-tice relačnej databázy. Stĺpec definovaný ako *BLOB* môže obsahovať zložený dokument v binárnej podobe. V dokumente sa potom odkazuje na dátový záznam alebo entitu. Tento krížový odkaz sa používa v aplikácii na prepojenie dokumentu s dátovým záznamom. Takýto prístup je užitočný z krátkodobého hľadiska, nakoniec je však potreba prístup, ktorý integruje dáta a zdroje dokumentov podľa ich obsahu, namiesto prepojenia dokumentov a dátových záznamov. K tomu môže slúžiť iný typ databáz, napríklad dokumentové databázy.

4.2.5 Hypertext

Software, ktorý implementuje hypertextovú štruktúru umožňuje nelineárny prístup k logickej štruktúre textu v rámci dokumentu a viaceré krížové odkazy medzi dokumentmi. Rozširujúca technológia hypermedia poskytuje rovnakú funkcionalitu s multimediálnymi a zloženými dokumentmi.

4.2.6 Získavanie a syntéza dokumentov

Získavanie (retrieval) dokumentov slúži na výber dokumentov z kolekcie na základe prítomnosti alebo absencie kľúčových slov priradených zostavovateľom indexov. Získavanie textu využíva algoritmy, ktoré eliminujú potrebu priradeného indexu. Všetky slová, ktoré majú sémantický obsah, sú indexované.

Ďalšie vylepšenie nazývané konceptuálne vyhľadávanie využíva slovník a analýzu koexistencie slov, na výber dokumentov, ktoré používajú podobné, ale odlišné slová.

Dotazy môžu mať za následok zoznam vybraných dokumentov zoradených podľa pravdepodobnej relevancie. Rozšírenie tohto prístupu umožňuje automatickú syntézu alebo sumarizáciu dokumentov.

4.2.7 Prenos a smerovanie dokumentov

Email predstavuje primárny transportný mechanizmus elektronických dokumentov a formulárov [6]. Nezávislosť objektov umožňuje prenos zložených dokumentov pozostávajúcich z rôznych objektov (text, grafika, obrázok, zvuk, video). Ďalšie funkcie potrebné pre obchodný transport elektronických dokumentov zahŕňa:

- autorizáciu – zabezpečenie, aby správny užívateľ pristupoval k pracovnej stanici a k dokumentom
- autentifikáciu – zabezpečenie platnosti elektronického podpisu
- šifrovanie – kódovanie a dekódovanie dokumentov kvôli bezpečnosti
- filtrovanie – automatické smerovanie správ alebo dokumentov na základe ich obsahu

Ďalšie relevantné technológie pre smerovanie zahŕňajú systém pre správu *workflow*, mechanizmy kontroly prístupu a inteligentné dokumenty. Inteligentné dokumenty obsahujú mechanizmus, aby vedeli, kto by mal dokument obdržať a v akej forme.

4.2.8 Tlač a zobrazovanie dokumentov

Väčšina dokumentov bude počas ich životného cyklu vytlačená, takže dôležitou technológiou je široká škála digitálnych tlačiarní v sieti. Tieto tlačiarne spolu s programami na spracovanie textu, jazykmi pre rozloženie stránok a *WYSIWYG* editormi poskytujú vysoko-kvalitný tlačený výstup.

Alternatívou k tlačeniu dokumentov je ich elektronické zobrazovanie alebo publikovanie. Dokument musí byť publikovaný vo formáte, v ktorom nemôže byť ľahko modifikovateľný.

Kapitola 5

Workflow

Veľa každodenných operácii v organizácii môže byť chápaných, ako séria opakujúcich sa krokov, na ktorých participujú viacerí užívatelia. Preto je možné očakávať, že nasadenie efektívneho systému s podporou toku informácii, bude mať za následok zníženie nákladov v organizácii. Takýto systém sa nazýva Workflow Management System (WfMS).

5.1 Workflow Management Coalition

V oblasti *workflow* figuruje nezisková organizácia Workflow Management Coalition [25], ktorá vznikla za účelom zjednotenia terminológie. Všetky produkty v oblasti spravovania *workflow* majú spoločné charakteristiky, čo umožňuje dosiahnuť určitý level spolupráce skrz využitie všeobecných štandardov. WfM Coalition mala za úlohu identifikovať tieto spoločné rysy a vytvoriť adekvátnu špecifikáciu pre implementáciu *workflow* systémov. Takáto špecifikácia umožní spoluprácu medzi rôznorodými *workflow* produktami a vylepší integráciu *workflow* aplikácii s ostatnými IT produktmi, ako napríklad systém pre správu dokumentov.

5.2 Základná terminológia

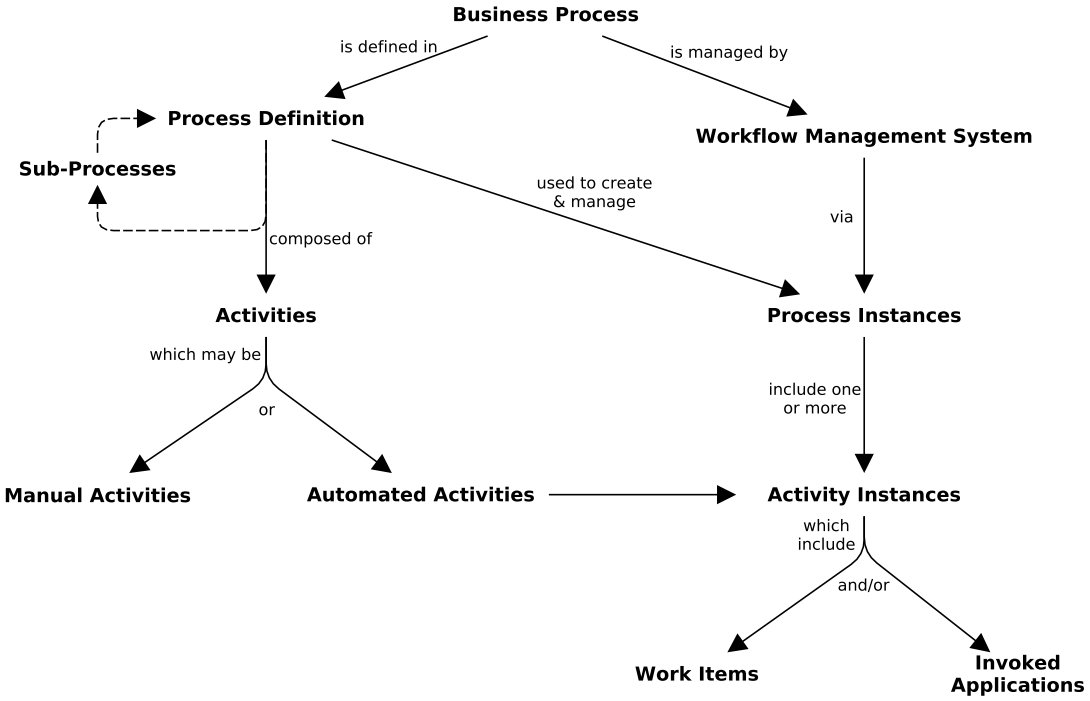
WfM Coalition vydala v roku 1996 terminologický slovnik, ktorý definuje základnú terminológiu *workflow* [15]. Obrázok 5.1 zachytáva vzťahy medzi základnou terminológiou. Jednotlivé súčasti sú popísané ďalej.

5.2.1 Workflow

Pod pojmom *workflow* rozumieme automatizáciu celého alebo časti podnikového procesu, počas ktorého sú dokumenty, informácie alebo úlohy predávané medzi účastníkmi procesu na základe procedurálnych pravidiel. Automatizácia podnikového procesu je definovaná v rámci procesu definície, ktorý identifikuje rôzne procesné aktivity, procedurálne pravidlá a súvisiace riadiace dáta využívané na riadenie *workflow* počas procesu prijatia.

5.2.2 Workflow management system

Riadenie *workflow* zaisťuje WfMS, ktorý definuje, vytvára a riadi priebeh procesu. Je schopný interpretovať definíciu procesu, komunikovať s účastníkmi *workflow* a v prípade potreby spustiť ďalšie aplikácie. WfM Coalition vytvorila referenčný model popisujúci jeho štruktúru a rozhranie. Systém obvykle poskytuje administratívne a monitorovacie funkcie,



Obr. 5.1: Vzťahy medzi terminológiou *workflow*

ako napríklad zmena účastníka, eskalácia, audit a riadiace informácie celého systému, prípadne jednotlivých procesov.

Workflow systémy obyčajne pokrývajú aj prípravnú fázu, ktorá slúži na nastavenie všetkých procesov. Obyčajne sa tu nastavujú parametre naviazania procesov, autorizácia, schvaľovanie a iné.

5.2.3 Podnikový proces

Je definovaný, ako množina jednej, alebo viacerých prepojených procedúr alebo aktivít, ktoré spoločne realizujú podnikový cieľ alebo stratégiu. Obyčajne v rámci kontextu organizačnej štruktúry definujúcej funkčné role a vzťahy.

Podnikový proces je typicky spojený s pracovnými cieľmi a podnikovými vzťahmi. Proces môže byť výlučne obsiahnutý v rámci jednej organizačnej jednotky, prípadne môže byť rozdelený do niekoľko rôznych organizácii.

5.2.4 Definícia procesu

Je reprezentácia podnikového procesu vo forme, v ktorej podporuje automatické spracovanie, ako modelovanie alebo prijatie systémom na spracovanie *workflow*. Definícia procesu pozostáva zo siete aktivít a ich vzťahov, kritérií na indikáciu začiatku a konca procesu a informácii o individuálnych aktivitách, ako účastníkov procesu a pridružených aplikácii.

5.2.5 Činnosť

Je popis časti práce, ktorá tvorí jeden logický krok procesu. Činnosť môže byť manuálna aktivita, ktorá nepodporuje automatizáciu počítačom, alebo *workflow* (automatizovaná) činnosť. *Workflow* vyžaduje ľudskú alebo strojovú podporu pre proces spracovania. V prípade, že je potrebný ľudský zdroj, činnosť je priradená účastníkovi *workflow*.

Definícia procesu obyčajne pozostáva z mnoho činností, ktoré sú logický prepojené tak, ako sa podieľajú na celkovej realizácii podnikového procesu. Činnosť typicky predstavuje najmenšiu jednotku práce, ktorá je naplánovaná počas prijatia procesu. Avšak jedna aktivita môže ústiť v priradení niekoľkých pracovných jednotiek účastníkom *workflow*.

Automatizovaná činnosť

Je činnosť, ktorú je možné počítačom automatizovať. WfMS spravuje činnosť počas vykonávania podnikového procesu, ktorého je súčasťou.

Manuálna činnosť

Je činnosť v rámci podnikového procesu, ktorú nie je možné automatizovať a preto leží mimo oblasť WfMS. Takéto činnosti môžu byť zahrnuté v rámci procesu definície, napríklad pre podporu modelovania procesu, ale nie sú súčasťou výsledného *workflow*.

5.2.6 Inštancia

Reprezentuje jedno spracovanie procesu alebo činnosti v rámci procesu vrátane použitých dát. Každá inštancia predstavuje samostatne vykonávané vlákno procesu alebo činnosti, ktoré môže byť riadené nezávisle. Každá inštancia má svoj interný stav a externe viditeľnú identitu, ktorá môže byť použitá k manipulácii, napríklad k získaniu údajov potrebných k auditu.

5.2.7 Účastník workflow

Zdroj, ktorý vykonáva prácu reprezentovanú činnosťou vo *workflow*. Táto práca sa obyčajne prejavuje, ako jedna, alebo viac pracovných položiek priradených účastníkovi *workflow*. Účastník *workflow* môže byť identifikovaný priamo v rámci definície podnikového procesu. Avšak častejšie je v rámci definície procesu priradený do role, ktorá môže byť vyplnená jedným, alebo viacerými dostupnými zdrojmi v systéme.

5.2.8 Pracovná položka

Je reprezentácia práce vykonaná účastníkom *workflow* v kontexte činnosti v rámci inštancie procesu. Činnosť typicky generuje jednu, alebo viac pracovných položiek, ktoré spolu formujú úlohu, ktorú prevezme účastník *workflow* v rámci tejto činnosti.

V niektorých prípadoch môže byť činnosť kompletne spracovaná aplikáciou, ktorá ju vyvolala. V takom prípade sa činnosť vykonáva bez účastníka *workflow* a nemusí byť priradená žiadna pracovná položka.

5.3 Výhody workflow

Cieľom WfMS je koordinácia všetkých zahrnutých entít vo vykonávaní podnikového procesu. Koordinácia môže byť definovaná, ako správa závislostí medzi aktivitami, pričom WfMS sa venuje dvom druhom koordinačným problémom [11].

Prvým je dátová nezávislosť medzi činnosťami, kde jedna činnosť závisí na výsledkoch jednej, alebo viacero iných činností. Činnosti sú riadené prostredníctvom kontrolných a dátových tokov.

Druhým problémom sú zdielané prostriedky, ako napríklad účastník *workflow*, ktorý môže vykonávať iba jednú úlohu v čase. Zdielané prostriedky sú riadené prostredníctvom mechanizmov plánovania a rozloženia zamestnancov. Automatizáciou týchto koordinačných problémov WfMS podporuje niekoľko cieľov efektivity podniku, ktoré sú zachytené v tabuľke 5.1 [2].

Cieľ	Popis	Podpora WfMS
Efektivita Procesu	Optimalizácia procesných kritérií, ako je doba spracovania (má byť minimalizovaná) alebo dodržiavanie termínov (má byť maximalizované).	Koordinácia činností skrz riadenie toku, termínov, atď.
Efektivita Zdrojov	Efektívne využitie zdrojov (ľudských zdrojov aj aplikačných systémov), ktoré sú dostupné pre vykonávanie procesov.	Rozloženie zamestnancov a pripomienka v prípade eskalácie.
Efektivita Trhu	Správne umiestnenie podniku vo vzťahu k obchodným partnerom. To zahŕňa spoľahlivé predpovedanie dodacích lehôt, transparentnú komunikáciu s dodávateľmi a zákazníkmi a optimalizované procesy dodávania a distribúcie.	Dobre definované rozhranie pre webové služby, predvídateľné vnútorné chovanie skrz štandardizované procesy.
Efektivita Delegácie	Primerané využívanie oprávnení nadriadeného a podriadených organizačných jednotiek.	Koordinácia priradenia zamestnancov a koncepcia rolí.
Efektivita Motivácie	Motivácia zamestnancov jednať spôsobom, ktorý odpovedá obchodným cieľom podniku.	Pokyny pre vykonávanie činností v rámci modelu *workflow*, sledovanie pokroku a vysvetlenie predošlých činností.

Tabuľka 5.1: Ciele efektivity podniku

Kapitola 6

Návrh riešenia

6.1 Use case diagram

Na modelovanie interakcie medzi účastníkom a systémom pre dosiahnutie cieľa je využitý diagram prípadov použitia. Dôraz je kladený na systém rolí, ktoré dovoľujú užívateľom vykonávať iba určité akcie, na základe pridelených práv k rolám. Na obrázku 6.1 sú zobrazení všetci aktéri. Detail na administrátora, ktorý zodpovedá za správu rolí, je zobrazený na obrázku 6.2.

V systéme sú preddefinované následujúce role: *app-admin*, *reader*, *writer*, *manager*.

App-admin je zodpovedný za vytváranie nových alebo úpravu existujúcich šablón. Šablóny sú ukladané ako *JSON schema* a určujú, akú majú mať dokumenty štruktúru a obsah. Šablóna je tiež využitá na vizualizáciu dokumentu.

Užívateľ s rolou *reader* má možnosť iba prezerať dokumenty. Nie je schopný ich upravovať alebo meniť ich stav. Môže si však zverejnený dokument exportovať do *PDF*. Tento typ užívateľov primárne prezerá zverejnené dokumenty.

Užívateľ s rolou *writer* vytvára nové dokumenty a upravuje ich. Na dokumente môže spolupracovať s ďalšími užívateľmi. Taktiež mení stav dokumentov a posiela ich na schválenie manažérovi. Môže vykonávať všetky bežné akcie s dokumentmi, ktoré nevyžadujú vyššie práva. Po dokončení môže podpísaný dokument zverejniť. Ďalej môže priradiť dokument užívateľom a nastaviť prístupové práva k danému dokumentu.

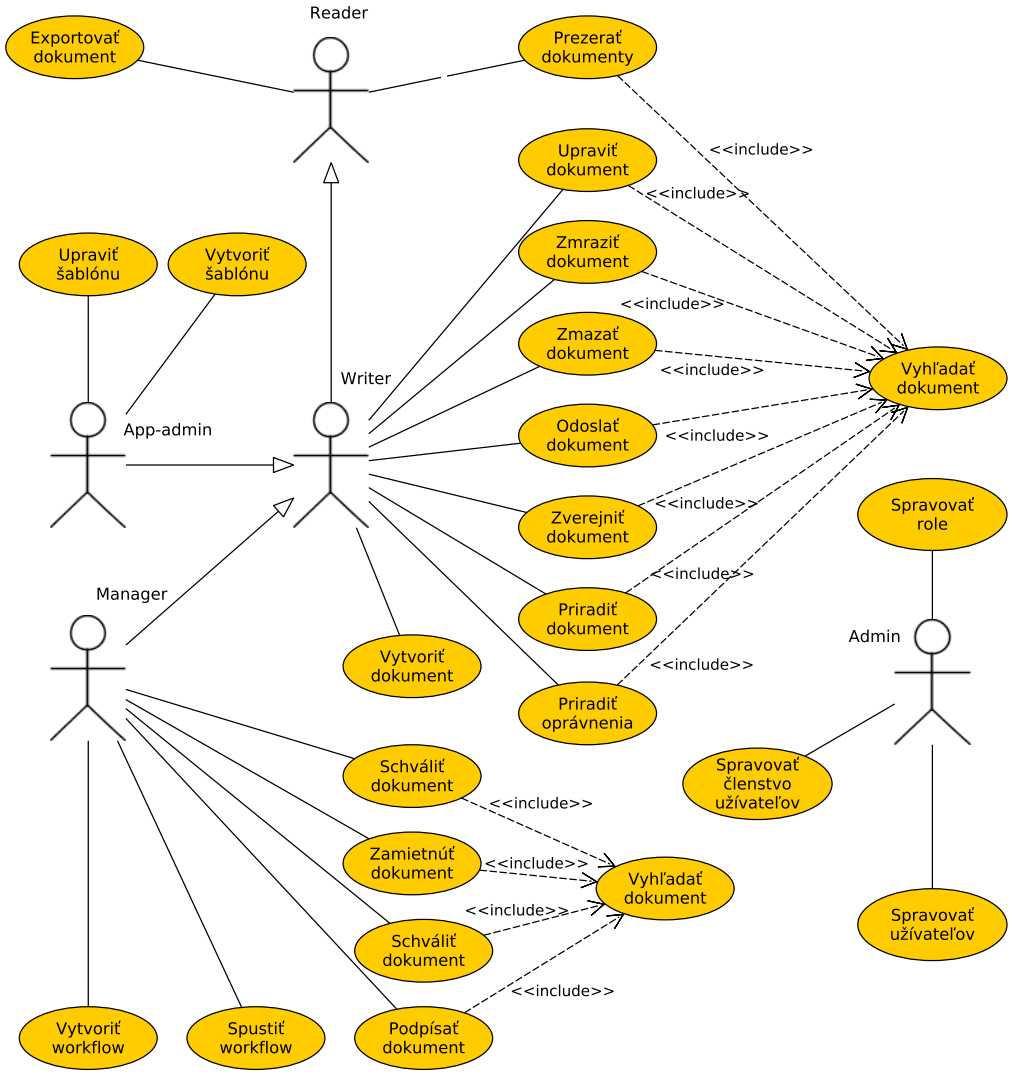
Užívateľ s rolou *manager* schvaľuje a podpisuje dokumenty. Dokument je po dokončení odoslaný jednému z manažérov na kontrolu. Ten dokument skontroluje, na základe čoho zhodnotí, či dokument schváli, alebo zamietne. Manažér ďalej môže spúšťať globálne *workflow* nad dokumentmi, prípadne vytvárať nové typy *workflow*.

V kontexte systému existuje užívateľ s rolou *admin*. Tento užívateľ nemusí byť nutne iná osoba, ako *app-admin*. Jedná sa o administrátora systému oprávnení, ktorý nie je súčasťou aplikácie. Administrátor môže spravovať užívateľov a role v systéme, čo zahŕňa vytváranie, menenie a mazanie užívateľov a rolí. Ďalej spravuje mapovanie medzi užívateľmi a rolami.

Role tvoria hierarchickú štruktúru, takže užívateľ s rolou *manager* dedí všetky práva od užívateľov s rolou *writer* a tak ďalej.

6.2 Systém oprávnení

Systém kladie dôraz na bezpečnosť, preto poskytuje správu rolí. Role spravuje administrátor, ktorý môže vytvárať nové role alebo mazať a upravovať stávajúce. Taktiež priraďuje a



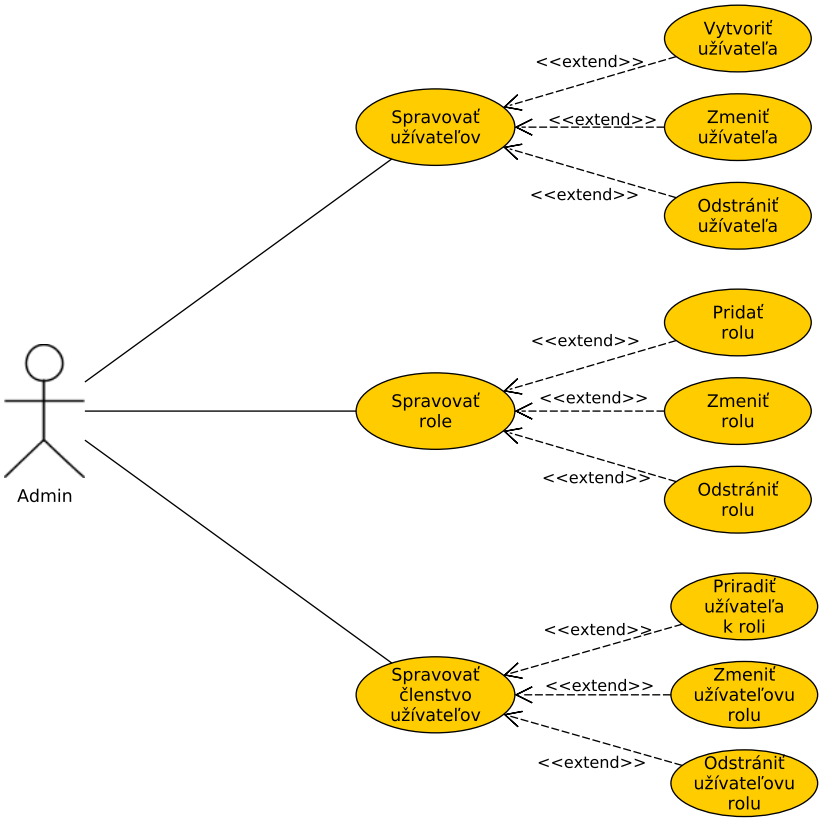
Obr. 6.1: Use case diagram – prehlad rolí v systéme.

odoberá užívateľov k rolám. Každá rola v sebe zahŕňa určité práva, na základe ktorých je užívateľom dovolené, alebo zakázané vykonávať konkrétne akcie.

Užívateľ sa autentizuje prihlásením do systému. Po úspešnom prihlásení si môže prezerať priradené role.

Aplikácia obsahuje dve vrstvy oprávnení. Prvá vychádza z rolí a ovplyvňuje úkony, ktoré môže daný užívateľ vykonávať s dokumentmi. Rozlišujeme právo zápisu, s ktorým môže užívateľ vytvárať a upravovať dokumenty. Právo čítania, s ktorým môže užívateľ iba prezerať dokumenty. A nakoniec práva manažéra, s ktorým môže užívateľ schvaľovať a podpisovať dokumenty.

Druhá vrstva oprávnení sa týka samotných dokumentov. Týmto spôsobom je možné vymedziť množinu užívateľov, pre ktorých je daný dokument viditeľný a rovnako obmedziť účastníkov toku dokumentu. V prípade, že dokument nemá špecifikovaných užívateľov, pre ktorých je viditeľný, tak je dostupný pre všetkých. Toto nastavenie je východiskové pre



Obr. 6.2: Use case diagram – detail na administrátora.

všetky nové dokumenty. Po pridaní prvého užívateľa do zoznamu oprávnených užívateľov sa začnú uplatňovať tieto práva. Štandardne je potom dokument viditeľný pre užívateľov v zozname, autora dokumentu a manažérov.

6.3 Tok dokumentov

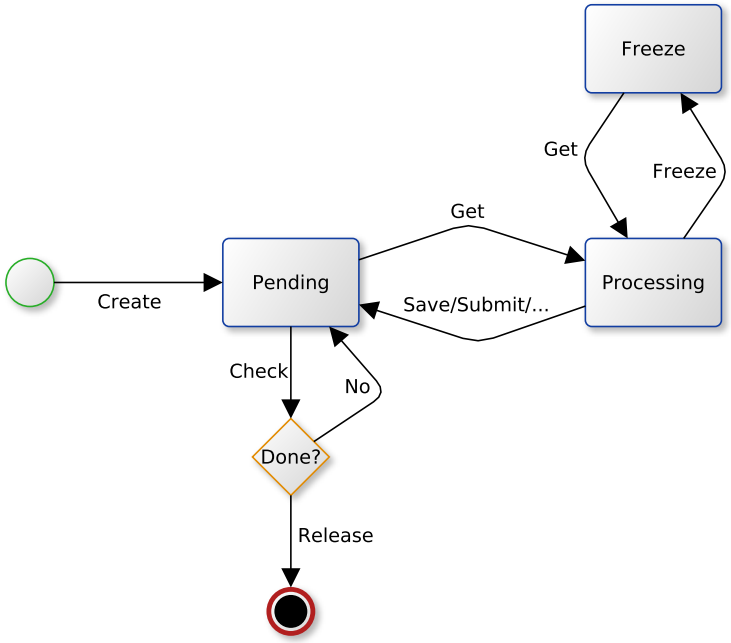
Systém je určený na vytváranie, správu a tok ľubovolných dokumentov. Dokument počas svojho životného cyklu prejde viacerými fázami, so začiatkom pri vytvorení dokumentu a koncom pri ukončení spracovania dokumentu. Stavový diagram na obrázku 6.3 popisuje rôzne stavy dokumentu.

Na počiatku je vytvorená inštancia dokumentu na základe jednej z dostupných šablón. Vzniknutý formulár má štruktúru definovanú šablónou, ktorá je definovaná pomocou JSON schémy. Užívateľ vyplní potrebné údaje a pri uložení sú k dokumentu pripojené metadáta a časť popisujúca *workflow*. Dokument prejde do čakajúceho stavu, kde čaká na ďalšie kroky.

Z čakajúceho stavu ho môžu jednotlivý užívatelia upravovať a spracovávať a to aj súčasne. V takom prípade však musia riešiť vzniknuté konflikty. Po vyzdvihnutí dokumentu užívateľom sa dokument nachádza v stave spracovávania. Tento stav zahŕňa následujúce úkony:

- *edit* – úprava položiek dokumentu

- *start progress* – začiatok práce na dokumente
- *close* – ukončenie práce na dokumente
- *reopen* – znovu začatie práce na dokumente
- *submit* – odoslanie dokumentu na schválenie manažérovi
- *approve* – schválenie dokumentu manažérom
- *decline* – zamietnutie dokumentu manažérom a vrátenie dokumentu na opravu
- *sign* – podpísanie schváleného dokumentu elektronickým podpisom manažéra



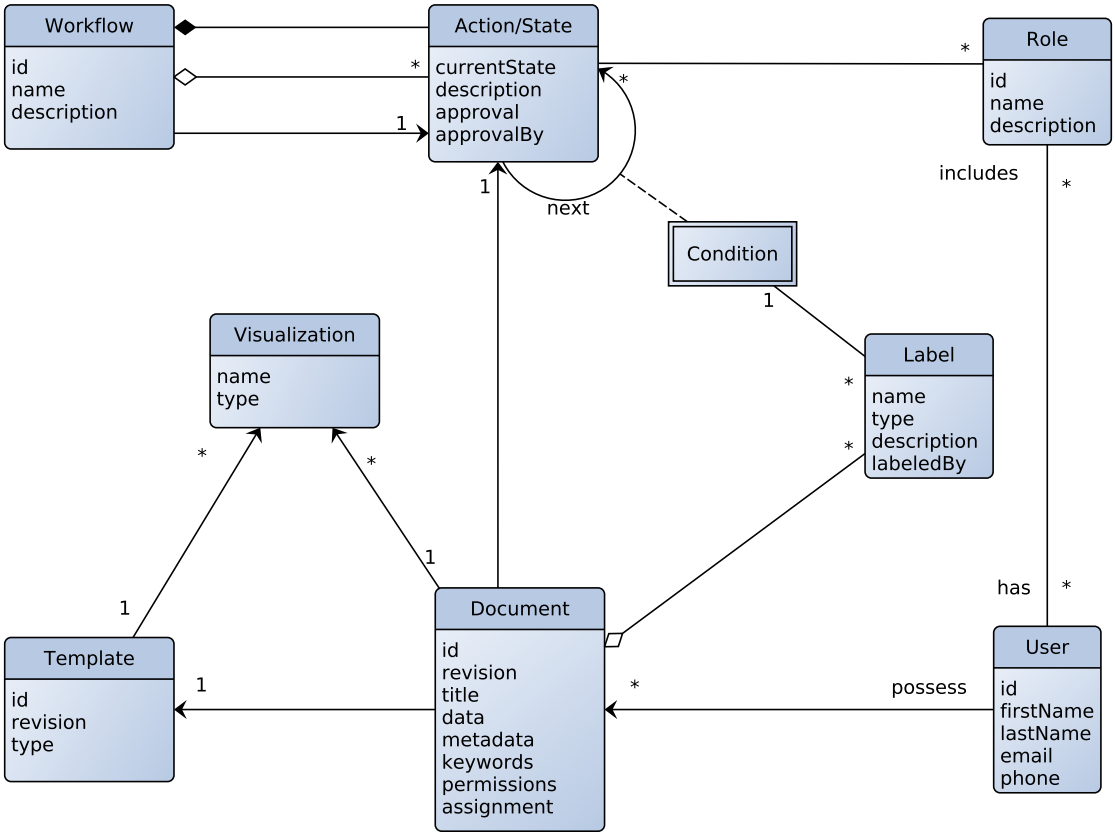
Obr. 6.3: Životný cyklus dokumentu

Po dokončení jednotlivých úkonov, je dokument vrátený späť do čakajúceho stavu. V prípade, že je dokument dokončený a podpísaný, môže byť zverejnený, čím končí jeho životný cyklus. Po zverejnení dokumentu si ho užívatelia môžu exportovať do *PDF*. V priebehu spracovávania môže byť dokument uvedený do stavu *freeze*, čo značí odloženie práce na dokumente na neskôr. Z tohto stavu je vyzdvihnutý ľubovoľnou zmenou vykonanou užívateľom.

Elektronický podpis dokumentu slúži na jednoznačnú identifikáciu pôvodu dokumentu a overenie obsahu, aby bol chránený pred neoprávnenými zmenami. Po zverejnení je dokument dostupný pre konzumentov, čiže pre interných, alebo externých účastníkov procesu, ktorí majú práva na čítanie dokumentu.

6.4 ER-diagram

ER diagram je využitý pre lepšie pochopenie vzťahov medzi jednotlivými entitami systému. Keďže systém postráda relačnú databázu, nebude ďalej využitý pre vytvorenie schémy databázy. ER diagram je zobrazený na obrázku 6.4.



Obr. 6.4: ER diagram

Každý dokument v systéme vlastní jeden z užívateľov. Každý užívateľ má priradenú rolu, ktorá mu umožňuje vykonávať iba určité akcie s dokumentmi. Systém rolí umožňuje administrátorovi spravovať role a užívateľov. Užívateľom môže pridávať nové role alebo meniť existujúce. Ďalej môže vytvárať nových užívateľov a odstraňovať existujúcich. Vďaka tomu je možné dosiahnuť rôzne úrovne oprávnení.

Každý dokument bol vytvorený na základe určitej šablóny. Dokument a šablóna môžu byť zobrazované rôznymi spôsobmi. Jedná sa napríklad o zobrazenie priamo vo webovom rozhraní systému v podobe formuláru alebo o export do *PDF*. Dokument sa vždy nachádza v určitom stave. Vo svojom životnom cykle sa dokument dostane z počiatočného stavu až do cieľového stavu v dôsledku vykonávania akcií, ktoré menia stav dokumentov. Každá akcia je vykonaná na základe splnenia určitej podmienky. Podmienky sú vyhodnocované podľa prítomnosti štítkov v dokumentoch a aktuálneho stavu. Štítky do dokumentov pridávajú užívatelia, prípadne samotný systém. Štítok môže predstavovať napríklad podpis alebo schválenie dokumentu užívateľom, ale aj ďalšie komplexnejšie úkony.

Tok dokumentov predstavuje postupnosť akcií, vykonávaných na základe splnenia určitých podmienok. Tok dokumentov je reprezentovaný počiatočným stavom, aktuálnym stavom a množinou akcií, ktoré je možné vykonať.

6.5 RESTful API

Systém poskytuje *RESTful API* pre prístup k dokumentom. Operácie vyžadujú autentifikáciu užívateľa. Po úspešnej autentifikácii sa skontroluje, či je užívateľ autorizovaný na daný úkon. Po úspešnej autorizácii sa skontroluje, či aktuálny stav dokumentu alebo systému dovoľuje vykovať požadovanú operáciu. Týmto sa zabráni neautorizovaným a nedovoleným zásahom do dokumentov a systému.

Tabuľka 6.1 zobrazuje *REST API* pre operácie s dokumentmi. Spoločný *URL* prefix pre všetky operácie je `/relax-dms-web/rest/document`. *REST API* umožňuje základné operácie, ako vytvorenie, zmazanie, upravenie a získanie dokumentu. Ďalej je možné získať ID všetkých dokumentov a prípadne aktuálnu alebo všetky revízie konkrétneho dokumentu. Taktiež je možné získať všetky šablóny v systéme. Posledná možnosť je vyzdvihnúť všetky hlavičky dokumentov, ktoré má autorizovaný užívateľ priradené alebo je ich autorom.

Metóda	URL	Poznámka
GET	/{docid}	získa dokument
GET	/ids	vráti ID všetkých dokumentov
GET	/rev/{docid}	vráti aktuálnu revíziu dokumentu
GET	/revs/{docid}	vráti všetky revízie dokumentu
POST	/store	vytvorí nový dokument
DELETE	/delete	zmaže dokument
POST	/update	upraví dokument
GET	/templates	vráti všetky šablóny
GET	/byAuthor	vráti hlavičky dokumentov podľa autora
GET	/byAssignee	vráti hlavičky dokumentov priradené užívateľovi

Tabuľka 6.1: REST API: Dokumenty

Tabuľka 6.2 zobrazuje *REST API* pre operácie s ovládacími prvkami nad dokumentmi. Spoločný *URL* prefix pre všetky operácie je `/relax-dms-web/rest/workflow/document`. Dostupné sú manažérske operácie, ako schválenie, zamietnutie a podpísanie dokumentu. Ďalej bežné operácie, ako odoslanie, priradenie, zmrazenie, zverejnenie, export dokumentu a iné. Rovnako je možné prideliť užívateľovi práva na dokument.

Metóda	URL	Poznámka
POST	/approve	schváli dokument
POST	/decline	zamietne dokument
POST	/submit/{manager}	odošle dokument manažérovi
POST	/assign/{assignee}	priradí dokument užívateľovi
POST	/sign	podpíše dokument
POST	/freeze	zmrazí dokument
POST	/release	zverejní dokument
POST	/export	exportuje dokument do *PDF*
POST	/startProgress	začne prácu na dokumente
POST	/close	zavrie dokument
POST	/reopen	znova otvorí dokument
POST	/addPermission/{user}	pridá užívateľovi práva na dokument

Tabuľka 6.2: REST API: Ovládacie prvky

Tabuľka 6.3 zobrazuje *REST API* pre operácie s globálnym *workflow*. Spoločný *URL* prefix pre všetky operácie je `/relax-dms-web/rest/workflow`. Pomocou týchto operácii môže administrátor spúšťať preddefinované *workflow*. Ďalej môže získať zoznam vytvorených pravidiel, pomocou ktorých sa spúšťa užívateľom vytvorený *workflow*.

Metóda	Cesta	Poznámka
POST	/fireUnfreeze	odmrazí zmrazené dokumenty
POST	/fireRelease	zverejní podpísané dokumenty
GET	/getCustomRules	získá vlastné pravidlá
POST	/fireCustom/rule	spustí vlastný *workflow*

Tabuľka 6.3: REST API: *Workflow*

Kapitola 7

Implementácia programu

7.1 Použité technológie

7.1.1 Java Enterprise Edition 7

Vývojári dnes stále viac rozpoznávajú potrebu distribuovaných, transakčných a prenosných aplikácii, ktoré využívajú rýchlosť, bezpečnosť a spoľahlivosť technológii na strane serveru. Podnikové aplikácie poskytujú podnikovú logiku pre organizácie, sú riadené centrálne a často komunikujú s ďalšími podnikovými aplikáciami. Podnikové aplikácie musia byť navrhnuté a vytvorené za menej peňazí, vyššou rýchlosťou a s menej zdrojmi.

Cieľom *Java EE* [21] platformy je poskytnúť vývojárom výkonnú sadu rozhraní *API* a zároveň skrátiť čas vývoja, znížiť zložitosť aplikácii a vylepšiť ich výkon. *Java EE* platforma využíva zjednodušený programovací model. Poskytuje štandardné systémové služby, takže vývojár sa sústredí iba na implementáciu obchodnej a prezentačnej logiky.

Verzia 7 prináša mnoho zjednodušení oproti predchádzajúcim verziám. Napríklad *XML* deskriptory sú nepovinné a vývojár môže využiť anotácie priamo v zdrojovom kóde a o konfiguráciu sa postará aplikačný server. V *Java EE* platforme môže byť injekcia závislostí aplikovaná na všetky vyžadované zdroje a komponenty, čo skrýva vytváranie a vyhľadávanie zdrojov z aplikačného kódu. S využitím anotácii, injekcia závislostí umožňuje *Java EE* kontajneru automaticky vkladať odkazy na požadované komponenty alebo zdroje.

Z uvedených dôvodov sme sa rozhodli implementovať aplikáciu s využitím *Java EE 7* technológii. Na implementáciu prezentačnej vrstvy sme zvolili *Wicket framework* [19].

7.1.2 JBoss Enterprise Application Platform

Jboss EAP [24] je *open source Java EE* aplikačný server využívaný pre vytváranie, nasadzovanie a hosťovanie vysoko-transakčných *Java* aplikácii a služieb. Keďže je založený na programovacom jazyku *Java*, pracuje naprieč platformami a je použiteľný na ľubovoľnom systéme podporujúcom *Javu*.

Jboss EAP 7 je certifikovaná implementácia *Java EE 7* špecifikácie. Poskytuje nakonfigurované funkcie, ako clustering s vysokou dostupnosťou, posielanie správ a distribuované ukladanie do vyrovnávacej pamäti. Taktiež obsahuje API a vývojové nástroje pre rýchly vývoj bezpečných a škálovateľných *Java EE* aplikácii. *Jboss EAP* obsahuje modulárnu štruktúru, ktorá umožňuje povolenie služieb, iba keď sú potreba, čo zrýchľuje zapnutie servera. Webová konzola a konzolové rozhranie pre správu (CLI) slúžia na konfiguráciu servera a uľahčujú automatizáciu úloh pomocou skriptov. *Jboss EAP* poskytuje samostatný prevádz-

kový režim serveru pre spravovanie diskrétnych inštancií a doménový prevádzkový režim pre správu skupín inštancií z jediného kontrolného bodu.

Mnoho rozhraní *API* a funkcií, ktoré sú dostupné aplikáciam nasadeným na *Jboss EAP*, je organizovaných do podsystémov. Tieto podsystémy môžu byť nakonfigurované administrátorom, aby poskytovali rôzne chovanie, v závislosti na potrebách aplikácie. Napríklad, ak aplikácia potrebuje databázu, tak v príslušnom podsystéme môže byť nakonfigurovaný dátový zdroj, ktorý bude sprístupnený aplikácii po jej nasadení na server.

Vysoká dostupnosť (HA) v *Jboss EAP* sa týka viacerých spolupracujúcich inštancií, ktoré poskytujú aplikácie odolnejšie voči kolísaniu v záťaži serveru a jeho zlyhaniu. HA zahŕňa koncepty, ako škálovateľnosť, rozdelenie záťaže a odolnosť voči chybám.

Aplikáciu sme nasadili na *Jboss EAP* server, ktorý beží v samostatnom prevádzkovom režime s preddefinovanou konfiguráciou podsystémov.

7.1.3 Jboss Business Rules Management System

Jboss BRMS [23] je otvorená platforma pre správu rozhodovania, ktorá kombinuje riadenie obchodných pravidiel a komplexné spracovanie udalostí. Automatizuje obchodné rozhodnutia a sprístupňuje túto logiku celej firme. V dnešnom svete, kde IT organizácie trvalo čelia zmenám z hľadiska stratégií, nových produktov a vládnych nariadení, *Jboss BRMS* uľahčuje oddelenie obchodnej logiky od zdrojového kódu aplikácie. *Jboss BRMS* zahŕňa *engine* s pravidlami, prostredie pre vývoj pravidiel, systém pre správu a centralizované úložisko, kde sú uložené všetky zdroje. Umožňuje vývojárom a obchodným analytikom prezerať, spravovať a overovať obchodné pravidlá, ktoré sú vykonávané v rámci infraštruktúry IT aplikácii. Obchodní analytici môžu meniť obchodnú logiku bez nutnosti programovania alebo asistencie od IT pracovníkov. *Jboss BRMS* sa skladá z nasledujúcich častí:

- *Drools Expert* – je *engine* s pravidlami založený na párovaní vzorov. Obsahuje interferenčný *engine*, produkčnú a pracovnú pamäť. Pravidlá sú uložené v produkčnej pamäti a fakty, voči ktorým interferenčný engine páruje pravidlá, sú uložené v pracovnej pamäti.

- *Business Central* – je webové rozhranie určené pre obchodných analytikov na vytváranie a udržiavanie obchodných pravidiel. Je navrhnuté tak, aby uľahčilo vytváranie a balenie pravidiel pre obchodných analytikov.

- *Drools Flow* – poskytuje možnosti pre vytváranie podnikových procesov pomocou vývojových diagramov.

- *Drools Fusion* – poskytuje možnosti pre spracovanie udalosti. Definuje radu cieľov, ktoré majú byť dosiahnuté.

Jboss BRMS môže byť spustené v ľubovoľnom aplikačnom serveri, ktorý implementuje *Java EE* špecifikáciu. V našom prípade spúšťame *Jboss BRMS* v *Jboss EAP*. V aplikácii využívame Drools Expert časť na definovanie pravidiel, ktoré riadia *workflow* dokumentov a Business Central rozhranie, ktoré využíva administrátor pre správu pravidiel.

Pre vytváranie obchodných pravidiel musí byť prítomný vhodný model faktov, na ktorom pravidlá operujú. Fakt je inštancia objektu aplikácie, reprezentovaného ako *POJO* (Plain old Java object). Pravidlo má následujúcu štruktúru:

Algoritmus 7.1: Štruktúra pravidla

```
rule "NAME"
when
    RULE CONDITIONS
then
    RULE CONSEQUENCES
end
```

Podmienky vo vnútri klauzule `when` sa dotazujú na kombináciu faktov, ktoré spĺňajú kritéria. Ak sa taká kombinácia faktov nájde, sú vykonané dôsledky špecifikované v klauzule `then`. Tieto akcie môžu vložiť, odstrániť alebo zmeniť fakt v pracovnej pamäti. Vo výsledku môžu byť vykonané ďalšie pravidlá. Postup spracovania pravidiel je následovný:

1. *BRMS* analyzuje všetky .drl súbory, ktoré obsahujú pravidlá a vloží ich do báze znalostí.
2. Každý fakt je vložený do pracovnej pamäte. *BRMS* využíva algoritmy *PHREAK* alebo *ReteOO* na odvodenie, ako sa fakty vzťahujú k pravidlám[1]. Pracovná pamäť potom obsahuje kópie pravidiel s odkazmi na fakty.
3. Zavolá sa metóda `fireAllRules()`. *Engine* vyhodnotí všetky pravidlá a fakty a páruje pravidlá s faktami tak, aby sa pravidlá zhodovali s danou množinou faktov.
4. Všetky spárované pravidlá s faktami sú zaradené do fronty nazývanej *agenda*.
5. Záznamy z fronty sú spracovávané jeden za druhým, čím sa vykonávajú dôsledky pravidiel. Vykonanie záznamu môže zmeniť obsah fronty predtým, než sa vykoná ďalší záznam. Využívané algoritmy sa dokážu vysporiadať s touto situáciou efektívne.

Objekt, ktorý reprezentuje dokument je vložený do pracovnej pamäte ako fakt. Ak sa už daný fakt nachádza v pracovnej pamäti, tak je pozmenený. Následne sa zavolá metóda `fireAllRules()`, čo spôsobí vykonanie odpovedajúceho pravidla, ktoré bolo spárované s faktom. Dôsledok pravidla upraví daný fakt, ktorý sa uloží späť do databázy. V prípade globálneho *workflow* sú do pracovnej pamäte vložené všetky dokumenty z databázy.

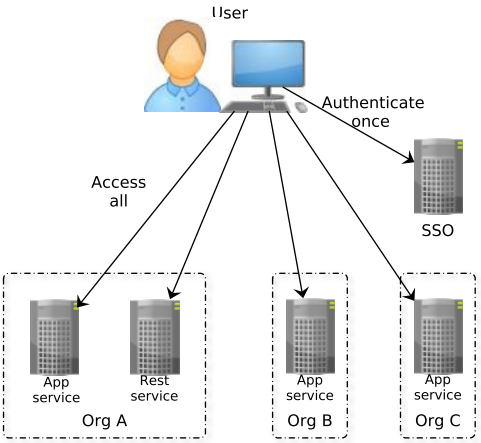
Pravidlá sú využívané v aplikácii na rôznych miestach na definovanie chovania. Sú pomocou nich implementované akcie v jednotlivých fázach toku dokumentu. Jedná sa napríklad o úkony, ako odoslanie, schválenie alebo zverejnenie dokumentu. Pre každý úkon existuje odpovedajúce pravidlo, ktoré implementuje žiadané chovanie. Implementácia úkonov nie je teda súčasťou zdrojového kódu aplikácie. Z toho vyplýva výhoda, že chovanie programu môže byť jednoduchšie pozmenené a aplikácia môže byť rozšírená o novú funkcionalitu za behu programu, bez nutnosti programovania. Obchodný analytik s právami administrátora má prístup do webovej konzole *BRMS*, kde môže spravovať pravidlá.

Ďalšia oblasť, kde sa využívajú pravidlá sú tzv. globálne *workflow*. Jedná sa o *workflow*, ktoré operujú nad množinou dokumentov, ktoré spĺňajú určité podmienky. Dostupných je niekoľko preddefinovaných *workflow* ako napríklad odmrazenie všetkých zmrazených dokumentov alebo podpísanie všetkých schválených dokumentov. Obchodný analytik má možnosť vytvoriť ďalšie vlastné *workflow* pomocou pravidiel. Pravidlo definuje chovanie nového *workflow*, ktorý sa objaví v aplikácii v záložke *Workflow* v rozbaľovacom zozname, odkiaľ môže byť spustený.

[1] Viac informácii je dostupných v [12]

7.1.4 Keycloak

Keycloak [22] je *open source* projekt pre správu identity a prístupu, zameraný na zabezpečenie moderných aplikácii a služieb. Poskytuje podporu jediného prihlásenia (Single Sign-On), čo znamená, že užívatelia sa autentizujú so serverom *Keycloak* a samotné aplikácie sa nemusia zaoberať prihlasovacím formulárom, autentifikáciou a ukladaním užívateľov. Po prvom prihlásení do *Keyloak* serveru sa užívatelia nemusia prihlasovať znova a môžu pristupovať k rôznym aplikáciam. Rovnako to platí aj pre odhlásenie. Užívatelia sa odhlásia iba raz a následne budú odhlásení zo všetkých aplikácii, ktoré využívajú *Keycloak*. V prípade, že sa užívatelia autentizujú na pracovnej stanici pomocou protokolu *Kerberos*, môžu byť automaticky autentifikovaní do *Keycloak* serveru bez nutnosti zadávať prihlasovacie meno a heslo znova. *Keycloak* tiež poskytuje prihlasovanie skrz sociálne siete.



Obr. 7.1: Single Sign-On

V aplikácii využívame *Keycloak* pre prihlasovanie užívateľov, zabezpečenie *REST API* a správu rolí. Preto nie je potreba ukladať užívateľov a role do databázy a implementovať prihlasovanie užívateľov. Pre prístup ku *REST* zdrojom užívateľ potrebuje prístupový token, ktorý môže získať dotazom na Keycloak server s platným prihlasovacím menom a heslom. *Keycloak* poskytuje administratívnu konzolu pre centrálnu správu serveru. Prihlasuje sa do nej užívateľ s rolou admin. V konzole môže spravovať užívateľov a role. Užívatelia sú združení do skupín, ktorým sú priradené právomoci pomocou rolí. Je možné vytvárať kompozitné role, vďaka čomu sa dá dosiahnuť dedičnosť a vytvoriť hierarchia rolí.

Prostredníctvom konzoly pre správu účtov si môžu užívatelia spravovať svoje vlastné účty. Môžu si upraviť profil, zmeniť heslo a nastaviť dvojfázovú autentifikáciu. Tiež môžu spravovať relácie (sessions) a pozerať históriu účtu. V prípade, že je umožnené prihlasovanie cez sociálne siete, užívatelia si tu môžu prepojiť účet s ďalšími poskytovateľmi a tým umožniť autentifikáciu do rovnakého účtu s odlišným poskytovateľom identity.

7.1.5 HAProxy

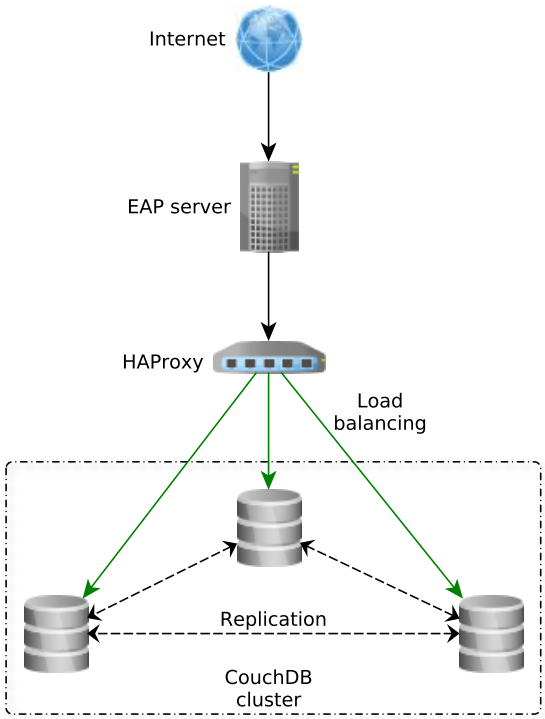
HAProxy [20] je open source software, ktorý poskytuje vyvažovanie záťaže s vysokou dostupnosťou a proxy server pre aplikácie založené na protokole *TCP* a *HTTP*, ktoré posielajú požiadavky na viacero serverov. Vyvažovanie záťaže zlepšuje distribúciu pracovnej záťaže

naprieč rôznymi výpočtovými zdrojmi, ako sú počítače, sieťové prepojenia alebo diskové jednotky. Cieľom je optimalizovať využitie zdrojov, maximalizovať priepustnosť, minimalizovať dobu odozvy a zamedziť preťaženiu niektorého zo zdrojov. Využitie viacerých komponent s vyvažovaním záťaže namiesto jednej komponenty môže zvýšiť spoľahlivosť a dostupnosť prostredníctvom redundancie.

HAProxy poskytuje viacero algoritmov pre rozloženie záťaže. V aplikácii sme zvolili východiskový algoritmus *round-robin*, ktorý využíva každý uzol striedavo v závislosti na váhe. Čas spracovania v každom uzle je rovnomerne rozdelený. Algoritmus je dynamický, čo znamená, že váhy môžu byť nastavované za behu.

7.2 Architektúra programu

Na obrázku 7.2 je zobrazená architektúra systému. *EAP* aplikačný server, na ktorom je nasadená aplikácia, prijíma požiadavky od užívateľov. Požiadavky na dokumenty idú ďalej skrz *HAProxy*, ktorá odpočúva na porte 5984. *HAProxy* rovnomerne rozdeľuje požiadavky medzi tri *CouchDB* uzly, ktoré spolu tvoria *cluster*. *CouchDB* uzly odpočúvajú na portoch 15984, 25984 a 35984. Jednotlivé uzly nie sú priamo dostupné, takže cluster je dostupný iba cez HAProxy. Pre potreby tejto práce postačuje, že všetky zúčastnené servery bežia lokálne na jednom serveri. Pri reálnom nasadení aplikácie, by boli uzly *CouchDB* distribuované na rôzne geografické polohy.



Obr. 7.2: Architektúra systému

7.2.1 CouchDB cluster

CouchDB vo verzii 2.0 predstavila podporu pre *clustering*, ktorý umožňuje lepšie škálovať aplikácie. Databáza je rovnomerne rozdelená do uzlov na totožné, ale separátne časti zvané *shards*. Čím viac častí databáza obsahuje, tým viac je možné databázu škálovať. Ľubovoľný dokument patrí do jednej časti, ktorá je určená z ID dokumentu. Vďaka tomu každý uzol vie, kde sa ľubovoľný dokument nachádza, čo umožňuje škálovanie požiadavkov na čítanie a zápis do databázy. *CouchDB* navyše uchováva viacero kópii z každej časti. *Cluster* má následovné parametre:

- q – Počet častí databázy (*shards*).
- r – Počet kópii dokumentu s rovnakou revíziou, ktoré musia musia byť prečítané než *CouchDB* vráti dokument. Ak je dostupný iba jeden dokument, tak je vrátený.
- w – Počet uzlov, ktoré musia uložiť dokument, než požiadavok zápisu vráti kód 201. Ak počet uzlov, ktoré uložia dokument menší ako w, ale väčší ako 0, je vrátený kód 202.
- n – Počet kópii každej časti (*shard*), tzv. repliky.

Repliky pridávajú odolnosť voči chybám, takže niektoré uzly môžu byť nedostupné, bez toho aby systém zlyhal. V systéme využívame východiskové nastavenie ($q = 8, r = 2, w = 2$, $n = 3$), ktoré je ideálne pre väčšinu prípadov. Uzol, ktorý spracováva *HTTP* požiadavok na dokument, vytvorí n paralelných procesov, aby vykonal žiadanú operáciu na všetkých kópiách dokumentu. Uzol počká na $n/(2 + 1)$ odpovedí, než spojí odpovede dohromady do jednej *HTTP* odpovedi. Tento postup pomáha zachovať konzistentný stav databázy, aj keď konzistencia nie je zaručená.

7.3 Usporiadanie dokumentu

Ako bolo spomenuté, *CouchDB* pracuje s dokumentmi vo formáte *JSON*. Každý dokument musí obsahovať položky `_id` a `_rev`. Zvyšná časť dokumentu nemá žiadne povinné kritéria zo strany databáze, ktoré je potreba splniť. Dokument je rozdelený do troch hlavných častí. Každá z častí predstavuje *JSON* objekt. Vo výpise 7.1 je ukážka usporiadania dokumentu.

Prvá časť sa nazýva *data* a obsahuje dáta, ktoré sú špecifikované šablónou, podľa ktorej bol dokument vytvorený. Táto sekcia obsahuje hlavnú časť informácií dokumentu. Dáta môžu mať ľubovolnú podobu, ktorá je ale vždy podmienená dostupnou šablónou. Aplikácia nijak nelimituje, ako majú dokumenty vyzerať. Štruktúra dokumentov prakticky závisí iba na potrebách organizácie. Preto môže aplikácia uspokojiť rôzne požiadavky organizácii.

Druhá časť sa nazýva *metadata* a obsahuje informácie o dokumente. Je tu uvedený autor dokumentu, autor poslednej zmeny a časové razítka. Ďalej je tu uvedený identifikátor a revízia šablóny, z ktorej bol dokument vytvorený. Tieto dáta majú hlavne informatívny charakter, ale môžu byť využité aj v pravidlách pre riadenie toku dokumentu.

Tretia časť sa nazýva *workflow* a obsahuje všetky informácie potrebné k toku dokumentov. Nachádza sa tu stav a oprávnenia dokumentov, ďalej štítky a užívateľ, ktorý ma aktuálne dokument priradený. K riadeniu toku dokumentov slúžia primárne štítky a aktuálny stav dokumentov. Nič ale nebráni tomu, využiť v pravidlách ostatné položky.

Posledná časť sa nazýva *_attachments*, kde sa ukladajú staršie verzie dokumentov ako binárne prílohy. Detaily sú popísané v ďalšej časti.

```
{
  "_id": "a6bfafde3436b5c0cb8a0ce4eb002607",
  "_rev": "5-6e464098ab8d42bdf9df06c3665eb662",
  "data": {
        "Title": "Feedback",
        ...
  },
  "metadata": {
    "author": "manager",
    ...
  },
  "workflow": {
    "state": {}
    "assignment": {}
    "labels": [],
    "permissions": []
  },
  "_attachments": {}
}
```

Výpis 7.1: JSON objekt dokumentu

7.4 História dokumentov

CouchDB nefunguje ako pravý systém na správu revízii, preto nie je možné spoliehať sa na databázu a je potreba implementovať vlastný systém revízii. Je to z toho dôvodu, že staršie revízie dokumentov sa nereplikujú a taktiež sú vymazané pri spustení *compaction*. Rovnako iba najnovšie revízie dokumentov sú zastúpené v databázových pohľadoch. Takže je možné sa spoliehať na dostupnosť iba jednej verzie dokumentu. Toto špeciálne platí v distribuovanom prostredí s viacerými uzlami databázy. V prípade, že dokument má viac revízii, je potreba staršie verzie ukladať, aby sme neprišli o históriu.

Existuje viac možností, ako implementovať správu revízii v CouchDB. Priamočiare riešenie je ukladať každú novú revíziu ako nový dokument. Tento prístup by však po určitej dobe vytvoril veľmi veľa dokumentov v databáze, keďže dokumenty sa často upravujú. Preto sme zvolili iný prístup. K ukladaniu starších revízii využívame položku *_attachments*, kde CouchDB ukladá prílohy dokumentu v binárnej podobe.

Pri každej zmene dokumentu je jeho predošlá verzia uložená ako binárna príloha. Položka *_rev* tejto revízie je využitá ako identifikátor prílohy. Príloha je potom dostupná na URL:

`http://localhost:5984/db/docid/3-5f26e2e86713b3204002d74cc6ae6069`,

kde posledná časť adresy je staršia revízia dokumentu, ktorú požadujeme.

CouchDB replikuje prílohy rovnako ako normálne dáta, takže týmto spôsobom neprídeme o žiadnu verziu dokumentu. Ďalšia výhoda je škálovateľnosť, keďže takto môžeme uložiť ľubovoľné množstvo dokumentov. Navyše uzly databáz nebudú plné previazaných dokumentov v rôznych verziách. Týmto jednoduchým a elegantným riešením sme implementovali správu revízii.

7.5 Riešenie konfliktov

Aplikácia podporuje spoluprácu a súbežné spracovanie dokumentov viacerými užívateľmi, pričom môžu vzniknúť konflikty. Konflikty sú detekované na strane *CouchDB*, pri pokuse upraviť zmenený dokument. Ako bolo spomenuté, každý dokument v *CouchDB* obsahuje položku `_rev`, v ktorej je uložená revízia dokumentu. V prípade, že viacerí užívatelia začnú pracovať na jednom dokumente s určitou revíziou a jeden z nich dokument upraví, *CouchDB* pridelí dokumentu novú revíziu. Druhý užívateľ avšak nemá dokument aktualizovaný a stále pracuje so starou revíziou. Pri pokuse uložiť zmenený dokument *CouchDB* odhalí konflikt a novú revíziu odmietne. *CouchDB* nemôže sama rozhodnúť, ktorá verzia dokumentu je správna a ako vyriešiť konflikt. Preto je riešenie konfliktov ponechané na užívateľoch.

Aplikácia uľahčuje užívateľom riešenie konfliktov. Po detekcii konfliktu je z databázy vyzdvihnutá najnovšia verzia dokumentu a tá je porovnaná s konfliktnou verziou dokumentu. Pri porovnávaní dvoch *JSON* dokumentov je odhalený ich rozdiel, čiže časti, ktoré boli zmenené prvým alebo druhým užívateľom. Tento rozdiel je potom graficky zobrazený užívateľovi vo formulári, v ktorom sa zobrazuje dokument. Časti, v ktorých je konflikt, sú zvýraznené červenou farbou a užívateľovi sú zobrazené obe konfliktné hodnoty. Po vyriešení konfliktov užívateľ dokument uloží a v prípade, že medzitým niekto ďalší dokument nezmenil, je dokument aktualizovaný v databáze. Vďaka tomuto systému detekcie konfliktov sa nikdy neuloží nesprávna verzia a neprídeme o žiadne dáta.

7.6 Export dokumentov

Systém poskytuje možnosť exportu dokumentov do formátu *PDF*. Exportovať je možné len zverejnený dokument. Môže tak urobiť každý, kto má práva vidieť daný dokument. Dokumenty sú ukladané vo formáte *JSON* a preto bolo potreba navrhnúť spôsob prevodu, keďže v tomto formáte sa neukladá štýl spolu s dátami.

K prevodu využívame knižnicu *Apache FOP* [18] (Formatting Objects Processor), ktorá prijíma vstup vo formáte *XSL-FO* a vyprodukuje výstup v požadovanom formáte. *XSL-FO* je značkovací jazyk pre formátovanie *XML* dokumentov. Najčastejší spôsob, ako previesť sémantiku *XML* dokumentu do *XSL-FO*, je *XSLT* transformácia.

Na export dokumentu do *PDF* sme navrhli postup, pri ktorom je dokument najprv prevedený do formátu *XML*. Následne je pridaný potrebný koreňový element. Kvôli prevodu do *XSL-FO* sme vytvorili obecnú *XSLT* šablónu so štýlom, ktorej aplikáciou na *XML* dokument vznikne naformátovaný *XSL-FO* objekt. Naformátovaný dokument je rozdelený do troch logických častí, kde každá odpovedá ekvivalentnej časti v pôvodnom *JSON* dokumente. Binárne prílohy sú odstránené pri prevode do *XML*.

V ďalšom kroku je *XSL-FO* objekt poslaný na vstup *Apache FOP*, ktorý *XSL-FO* objekt naformátuje do výsledného *PDF* dokumentu. Ten je napokon dostupný užívateľovi na stiahnutie.

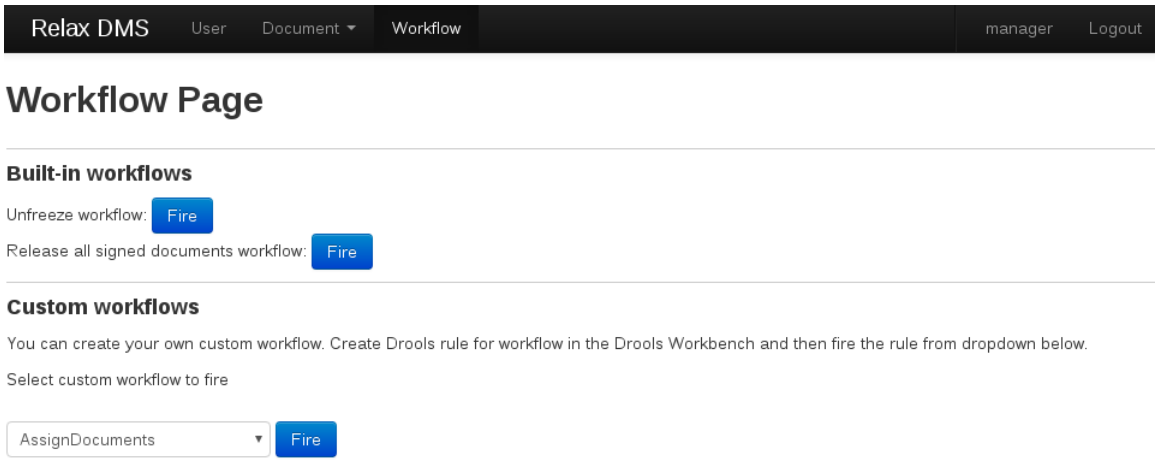
Kapitola 8

Prípadová štúdia

8.1 Vlastný workflow

Ako už bolo spomenuté, obchodný analytik má možnosť vytvoriť vlastný *workflow* za pomoci pravidiel a tým rozšíriť funkcionalitu aplikácie. V tejto časti to demonštrujeme na praktickom príklade.

Vytvoríme *workflow*, ktorý priradí všetky nepriradené dokumenty, ktoré sú v stave `Open` ich autorovi. Tento *workflow* je užitočný na to, aby dokumenty neboli v systéme nepovšimnuté, keďže sa automaticky nepriraďujú po ich vytvorení. Po priradení autor rozhodne, komu by mal byť dokument následne priradený.



Obr. 8.1: Stránka s globálnymi *workflow*

Workflow je definovaný pravidlom na ukážke 8.1, ktoré vložil do systému obchodný analytik napríklad prostredníctvom webovej konzoly *BRMS*. Pravidlo sa páruje s odpovedajúcimi faktami, pozmení ich a priradí do množiny zmenených dokumentov. Zmenené dokumenty sú následne aplikáciou aktualizované v databáze.

Algoritmus 8.1: Pravidlo pre vlastný *workflow*

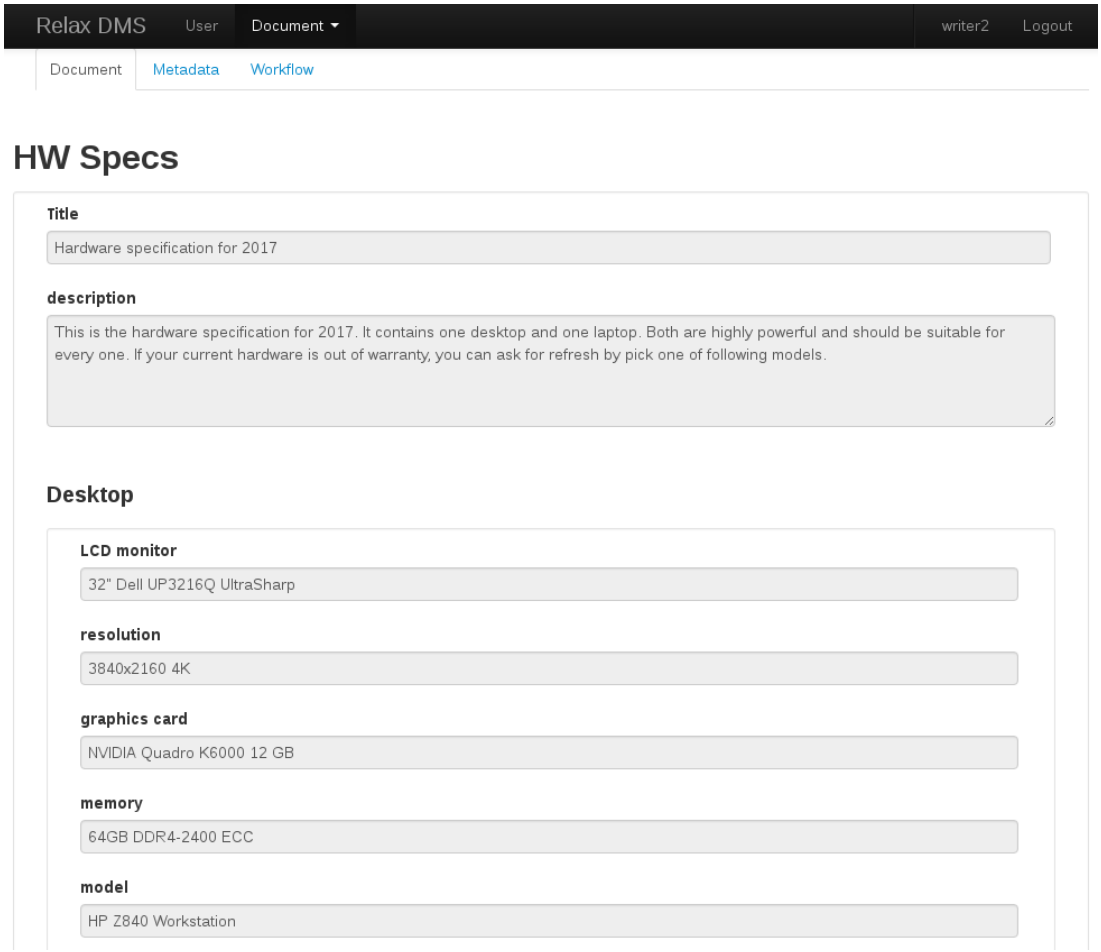
```
rule "AssignDocuments"
when
    $env : Environment(isTrue(), getRule() == "AssignDocuments"
    $document : Document(workflow.assignment.assignee == "Unassigned" &&
    workflow.state.currentState == StateEnum.OPEN)
then
    String author = $document.getMetadata().getAuthor();
    modify($document) {getWorkflow().getAssignment().setAssignee(author)};
    docSet.add($document);
end
```

Na obrázku 8.1 je zobrazená stránka s globálnymi *workflow*. V jej spodnej časti je možné vidieť rozbaľovací zoznam, ktorý obsahuje pridaný *workflow* s názvom *AssignDocuments*. Tlačidlom `Fire` je možné *workflow* spustiť.

8.2 Spolupráca a workflow

Na ilustráciu užitočnosti systému je vypracovaná prípadová štúdia, ktorá popisuje návrh výberu hardware do firmy. Dvaja pracovníci z oddelenia IT majú za úlohu urobiť návrh, aký hardware sa objedná na ďalší rok pre zamestnancov firmy. Návrh by mal obsahovať jeden laptop a jednú pracovnú stanicu. Zvolený hardware musí špecifikáciou čo najlepšie vyhovovať potrebám zamestnancov a zároveň sa musí vojsť do prideleného rozpočtu. Výsledný návrh bude odoslaný manažérovi na schválenie a podpísanie a následne bude zverejnený pre zamestnancov firmy. Po zverejnení môže byť vytvorený ďalší dokument, ktorý bude zachytávať spätnú väzbu od zamestnancov.

Táto štúdia je zameraná na ukážku spolupráce viacerých užívateľov na dokumente, na demonštrovanie toku dokumentu a jeho životného cyklu a na oboznámenie s užívateľským rozhraním aplikácie.

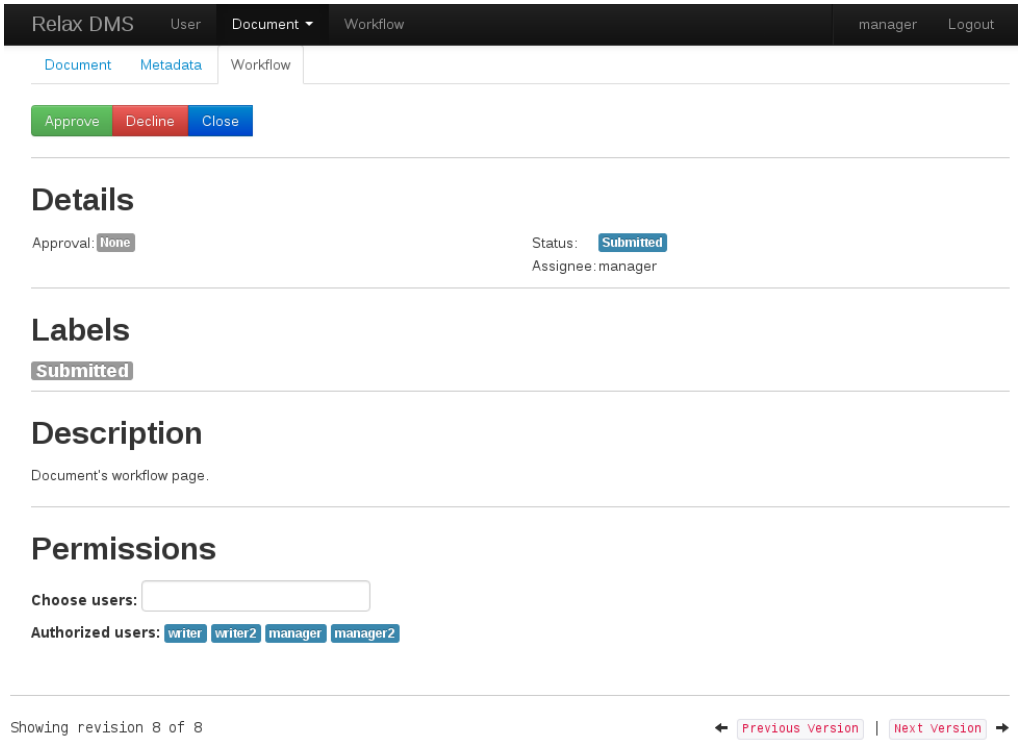


Obr. 8.2: Úvodný úsek dokumentu s detailom na dáta

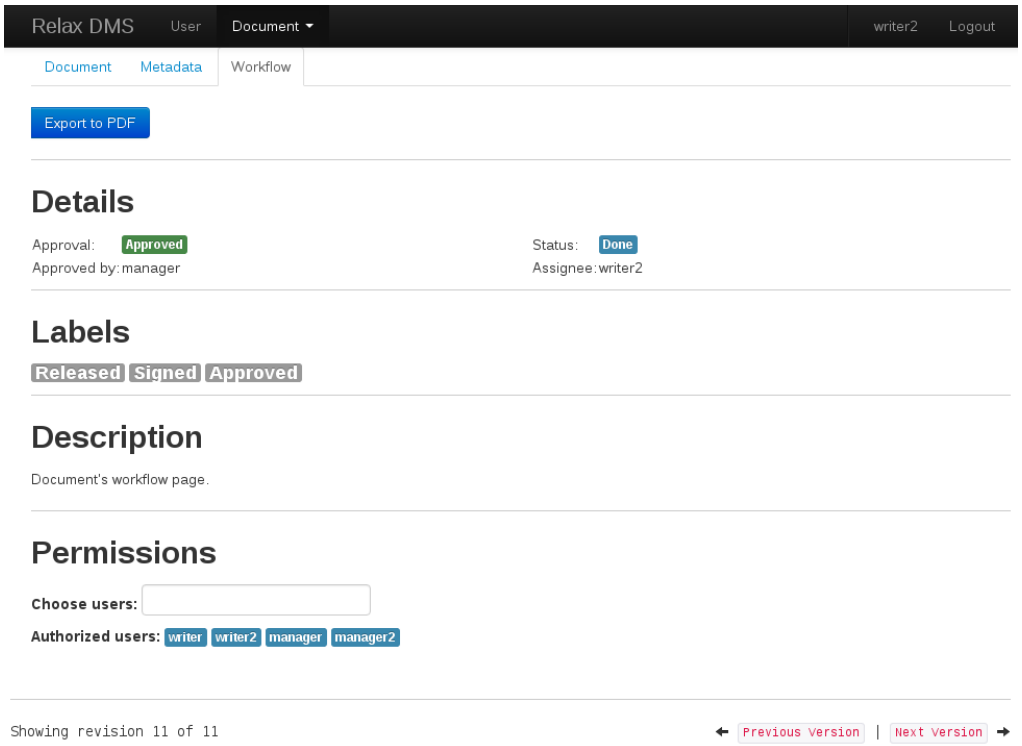
Na začiatku procesu je potreba, aby užívateľ s rolou *app-admin* vytvoril potrebnú šablónu pre dokument. Na dokumente budú spolupracovať dvaja užívatelia z IT oddelenia s menom *writer* a *writer2*. Užívateľ *writer* vytvorí dokument na základe šablóny, nastaví oprávnenia na dokument pre užívateľa *writer2*, priradí dokument sebe a zmení stav dokumentu na stav `In Progress`. Následne vyplní svoju časť dokumentu a upravený dokument uloží. Potom si užívateľ *writer2* dokument priradí a môže začať pracovať na svojej časti. Po dokončení dokument odošle manažérovi.

Na obrázku 8.2 je zobrazená úvodná časť dokumentu s detailom na záložku s dátami, ktorých podoba odpovedá šablóne. Bunky s dátami sú neaktívne, aby sa zabránilo nechceným zmenám. Editovať bunky je možné až po stlačení tlačidla `edit`.

Na obrázku 8.3 je zobrazený dokument po odoslaní na schválenie s detailom na záložku *workflow*. Manažér dokument skontroluje a prípade, že spĺňa požiadavky, tak ho schváli. V opačnom prípade ho zamietne a vráti na prerobenie. Po schválení je dokument elektronicky podpísaný a zverejnený medzi ostatných užívateľov. Na obrázku 8.4 je zobrazený dokument po dokončení. Automaticky prešiel do stavu `Done` a bol priradený späť užívateľovi, ktorý ho odoslal manažérovi. Za povšimnutie stoja aj priradené štítky, ktoré informujú o vykonaných úkonoch.



Obr. 8.3: Dokument po odoslaní manažérovi s detailom na *workflow* záložku



Obr. 8.4: Dokument po dokončení a zverejnení s detailom na *workflow* záložku

Kapitola 9

Zhodnotenie výsledkov

9.1 Dosiahnuté výsledky

V rámci implementácie bola vytvorená aplikácia Relax DMS [27], ktorá bola zverejnená ako *open source*. Súčasťou aplikácie je *cluster* o troch *CouchDB* uzloch s proxy na popredí, ktorá slúži na rozloženie záťaže. Aplikácia poskytuje následujúce kľúčové prvky:

- spravovanie ľubovoľných *JSON* dokumentov definovaných pomocou šablóny
- systém rolí a oprávnení
- webové rozhranie s *REST API*
- spoluprácu na dokumentoch s detekciou konfliktov
- tok a životný cyklus dokumentov
- vytváranie vlastných *workflow* za pomoci *BRMS* pravidiel
- vedenie histórie a export dokumentov do *PDF*
- vysokú dostupnosť a odolnosť voči chybám a preťaženiu

Vytvorená aplikácia slúži ako distribuovaný dokumentový server pre správu a *workflow* dokumentov. Založená je na *NoSQL* dokumentovej databáze *CouchDB*, čo vytvorené riešenie odlišuje od väčšiny dostupných riešení.

Z toho, že je aplikácia založená na *CouchDB*, vyplýva veľa výhod. Z podstaty fungovania *CouchDB clusteru* s *HAProxy* na popredí je aplikácia distribuovaná, odolná voči chybám a preťaženiu. Na druhú stranu môže nastať porušenie konzistencie a vzniku konfliktov, čo je vyriešené ich detekciou a zobrazením užívateľom, ktorý ich musia vyriešiť.

Ďalšou výhodou je ukladanie celých *JSON* dokumentov priamo do databázy, bez nutnosti riešiť spôsob ukladania, ako v prípade relačnej databázy.

Systém pre správu a *workflow* dokumentov umožňuje efektívne spravovať dokumenty počas celého ich životného cyklu. Nechýba riadenie toku dokumentov pomocou preddefinovaných úkonov a globálnych *workflow* pre všetky dokumenty. Do dokumentov sa pridávajú štítky, ktoré predstavujú vykonané akcie. Na základe štítkov, ale aj ďalších podmienok, je potom riadený tok dokumentov.

Systém umožňuje užívateľom súčasne spolupracovať na dokumentoch, čo má priaznivý vplyv na efektivitu podniku. Systém oprávnení a prístupu, ktorý je integrovaný do systému, poskytuje vysokú úroveň zabezpečenia.

Aplikácia je vysoko variabilná vďaka snahe o čo najobecnejšie riešenie. To je dosiahnuté možnosťou vytvárania ľubovoľných dokumentov na základe šablón, ktoré môže vytvárať administrátor. Ďalší z dôvodov je možnosť vytvárania vlastných *workflow* za pomoci *BRMS* pravidiel. Vďaka tomu je logika *workflow* oddelená od jadra aplikácie a môže byť ľubovoľne rozširovaná a prispôsobovaná potrebám organizácie.

9.2 Ďalší vývoj

Ďalší vývoj aplikácie je možné smerovať k zlepšeniu ovládateľnosti a užívateľského rozhrania, keďže to nepatrilo k prioritám v rámci diplomovej práce. Ako ďalšiu užitočnú funkcionalitu vidíme integráciu s *couchdb-lucene* [26], pre umožnenie fulltextového vyhľadávania.

V prípade veľkého zaťaženia aplikačného serveru, by bolo možné každému uzlu priradiť vlastnú inštanciu aplikačného serveru (*EAP*) a nastaviť tzv. *domain* mód, v rámci ktorého by sa inštancie spravovali centrálne. *HAProxy* by potom smeroval požiadavky rovnomerne medzi inštancie *EAP*.

Na umožnenie hlbšieho otestovania aplikácie, by bolo potreba aplikáciu nasadiť na viacero fyzických serverov, ideálne na rôznych geografických polohách a simulovať reálne nasadenie a prevádzku. Na základe získaných dát by bolo možné vytvoriť štatistiky a vykonať ďalšie optimalizácie a vylepšenia.

Kapitola 10

Závěr

Cieľom práce bolo oboznámiť čitateľa s problematikou distribuovaného uloženia dát a dokumentovou databázou *CouchDB*. Detailne naštudovať akým spôsobom *CouchDB* rieši ukladanie dokumentov, synchronizáciu a prístup ku CAP teorému.

V rámci druhej kapitoly sme definovali pojem distribuované databázové systémy. Popísali sme motiváciu pre ich využívanie, aké majú výhody a aké problémy sa u nich vyskytujú. Ďalej sme oboznámili s CAP teorémom a jeho alternatívou zvanou PACELC. Predstavili sme rôzne typy distribuovaných systémov podľa požiadavkov, ktoré splňujú.

V tretej kapitole sme porovnali tradičný relačný model dát s novým prístupom zvaným *NoSQL*. Ukázali sme sme aký má tento prístup výhody a aké má sú možnosti jeho využitia.

Ďalej sme predstavili dokumentovú databázu CouchDB, spôsob uloženia dokumentov prostredníctvom *B+ stromov*. Vysvetlili sme sme základné operácie na *B+ stromoch*, ceny operácii a ich výhody. Taktiež sme ukázali, ako sa CouchDB stavia ku CAP teorému. Nakoniec sme detailne vysvetlili algoritmus replikácie.

Ďalším cieľom práce bolo oboznámenie so systémami pre správu dokumentov (DMS) a systémami pre správu *workflow* (WfMS). Konkrétne, aká je ich funkcionalita v čom spočíva ich prínos pre organizácie.

Štvrtá kapitola pojednáva o systémoch pre správu dokumentov. Zamerali sme sa na motiváciu pre používanie, oboznámenie so základnou funkcionalitou a využívané technológie.

Piata kapitola zachytáva základnú terminológiu *workflow* v pojatí organizácie Workflow Management Coalition. V závere kapitoly sú diskutované výhody, ktoré *workflow* poskytuje organizáciám.

S pomocou nadobudnutých vedomostí bol navrhnutý a implementovaný distribuovaný databázový systém pre správu a *workflow* dokumentov založený na databáze *CouchDB*. Navrhnutá aplikácia poskytuje vytváranie ľubovoľných dokumentov, riadenie ich životného cyklu a toku. Mimo ďalšiu funkcionalitu, aplikácia obsahuje aj systém oprávnení a možnosť vytvárania vlastných *workflow*. Systém bol implementovaný ako webová aplikácia, ktorá poskytuje *RESTful API* pre prístup k dokumentom. Aplikácia bola zverejnená ako *open source* [27].

Návrh aplikácie je popísaný v rámci šiestej kapitoly. Návrh obsahuje diagramy použitia, ER diagram, popis toku dokumentov, systém oprávnení a popis *RESTful API*.

Siedma kapitola obsahuje implementačné detaily, riešenie zaujímavých problémov, popis architektúry programu a využitých technológii.

Ôsma kapitola obsahuje dve prípadové štúdie, ktoré majú za cieľ demonštrovať použiteľnosť aplikácie. Prvá je zameraná na vytváranie vlastných *workflow* a druhá na oboznámenie s vytváraním a tokom dokumentov.

V deviatej kapitole zhodnocujeme dosiahnuté výsledky a diskutujeme ďalší možný vývoj aplikácie.

Voľbu databázy *CouchDB* ako úložisko pre aplikáciu považujeme za vhodnú, keďže jej výhody oproti nevýhodám sú značné. Taktiež integráciu hotových riešení *Keycloak* a *BRMS*, zhodnocujeme ako prospešné, pretože prinášajú aplikácii veľké výhody v podobe zabezpečenia a rozšíriteľnosti.

Na základe dosiahnutých výsledkov považujeme aplikáciu za prospešnú pre organizácie, ktoré spracovávajú elektronické dokumenty. Jej nasadenie by mohlo viesť k zefektívneniu podnikových procesov a k zníženiu nákladov. S dosiahnutými výsledkami sme preto spokojní.

Literatúra

[1] Abadi, D. J.: Consistency Tradeoffs in Modern Distributed Database System Design: CAP is Only Part of the Story. *Computer*, ročník 45, č. 2, 2012: s. 37–42, ISSN 0018-9162.
URL http://doi.ieeecomputersociety.org/10.1109/MC.2012.33

[2] Becker, J.; zur Muehlen, M.; Gille, M.: Workflow application architectures: classification and characteristics of workflow-based information systems. *Workflow handbook*, ročník 2002, 2002: s. 39–50.

[3] Beynon-Davies, P.: *Database Systems*. PALGRAVE MACMILLAN, 2004, iSBN 1-4039-1601-2.

[4] Brewer, E. A.: Towards Robust Distributed Systems (Abstract). In *Proceedings of the Nineteenth Annual ACM Symposium on Principles of Distributed Computing*, PODC '00, New York, NY, USA: ACM, 2000, ISBN 1-58113-183-6, s. 7–.
URL http://doi.acm.org/10.1145/343477.343502

[5] Comer, D.: Ubiquitous B-Tree. *ACM Comput. Surv.*, ročník 11, č. 2, Červen 1979: s. 121–137, ISSN 0360-0300.
URL http://doi.acm.org/10.1145/356770.356776

[6] Foundation, B. C.: The Future of electronic mail. 1991, accompanied by management summary.

[7] Gilbert, S.; Lynch, N.: Brewer's Conjecture and the Feasibility of Consistent, Available, Partition-tolerant Web Services. *SIGACT News*, ročník 33, č. 2, Červen 2002: s. 51–59, ISSN 0163-5700.
URL http://doi.acm.org/10.1145/564585.564601

[8] Han, J.; Haihong, E.; Le, G.; aj.: Survey on NoSQL database. In *Pervasive Computing and Applications (ICPCA), 2011 6th International Conference on*, Oct 2011, s. 363–366, doi:10.1109/ICPCA.2011.6106531.

[9] Krishnan, R.; Munaga, L.; Karlapalem, K.: *XDoC-WFMS: A Framework for Document Centric Workflow Management System*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2002, ISBN 978-3-540-46140-1, s. 348–362, doi:10.1007/3-540-46140-X_27.
URL http://dx.doi.org/10.1007/3-540-46140-X_27

[10] Lamport, L.: The Part-time Parliament. *ACM Trans. Comput. Syst.*, ročník 16, č. 2, Květen 1998: s. 133–169, ISSN 0734-2071.
URL http://doi.acm.org/10.1145/279227.279229

[11] Malone, T. W.; Crowston, K.: The Interdisciplinary Study of Coordination. *ACM Comput. Surv.*, ročník 26, č. 1, Březen 1994: s. 87–119, ISSN 0360-0300, doi:10.1145/174666.174668.
URL http://doi.acm.org/10.1145/174666.174668

[12] Salatino, M.; De Maio, M.; Aliverti, E.: *Mastering JBoss Drools 6*. Community experience distilled, Packt Publishing, 2016, ISBN 9781783288632.
URL https://books.google.cz/books?id=CZnjCwAAQBAJ

[13] Sprague Jr, R. H.: Electronic document management: Challenges and opportunities for information systems managers. *MIS Quarterly*, 1995: s. 29–49.

[14] Stonebraker, M.: Errors in Database Systems, Eventual Consistency, and the CAP Theorem [online].
http://cacm.acm.org/blogs/blog-cacm/83396-errors-in-database-systems-eventual-consistency-and-the-cap-theorem/fulltext, April 5, 2010 [cit. 2016-01-04].

[15] WFMC: Workflow Management Coalition Terminology and Glossary. Technická zpráva, Workflow Management Coalition, Brussels, 1996.
URL http://www.wfmc.org/docs/TC-1011_term_glossary_v3.pdf

[16] WWW stránky: Apache CouchDB Release 2.0.0-git [online].
`https://media.readthedocs.org/pdf/couchdb/latest/couchdb.pdf`, 2015 [cit. 2016-01-04].

[17] WWW stránky: Apache CouchDB [online]. `http://couchdb.apache.org/`, [cit. 2016-01-04].

[18] WWW stránky: Apache FOP [online]. `https://xmlgraphics.apache.org/fop/`, [cit. 2017-05-02].

[19] WWW stránky: Apache Wicket [online]. `https://wicket.apache.org/`, [cit. 2017-05-02].

[20] WWW stránky: HAProxy [online]. `http://www.haproxy.org/`, [cit. 2017-05-02].

[21] WWW stránky: Java EE Overview [online].
`http://www.oracle.com/technetwork/java/javaee/overview/index.html`, [cit. 2017-05-02].

[22] WWW stránky: Keycloak [online]. `http://www.keycloak.org/`, [cit. 2017-05-02].

[23] WWW stránky: Red Hat Jboss BRMS [online].
`https://access.redhat.com/products/red-hat-jboss-brms/`, [cit. 2017-05-02].

[24] WWW stránky: Red Hat Jboss EAP [online].
`https://access.redhat.com/products/red-hat-jboss-enterprise-application-platform/`, [cit. 2017-05-02].

[25] WWW stránky: Workflow Management Coalition [online]. `http://www.wfmc.org/`, [cit. 2017-05-02].

[26] WWW stránky: couchdb-lucene github [online].
`https://github.com/rnewson/couchdb-lucene`, [cit. 2017-05-10].

[27] WWW stránky: Relax DMS github [online].
`https://github.com/martin-kanis/relax-dms`, [cit. 2017-05-10].

[28] Özsu, M. T.: *Principles of Distributed Database Systems*. Springer, 2011, iSBN 978-1-4419-8833-1.

Príloha A

Obsah CD

- Zdrojové súbory aplikácie.
- Technická správa vo formáte *PDF*
- Zdrojové súbory technickej správy vo formáte .tex.