Slovenská technická univerzita
Fakulta informatiky a informačných technológií
Ilkovičova 3, 842 16 Bratislava 4

Personalizované odporúčanie

Tímový projekt

Dokumentácia k inžinierskemu dielu

Tím číslo:	12	Bc. Michal Cádrik (SI)
Akad. rok:	2011/2012	Bc. Martin Detko (SI)
Predmet:	Tímový projekt	Bc. Igor Slotík (SI)
Kontakt:	tim12tp@googlegroups.com	Bc. Jakub Ševcech (SI)
Vedúci tímu:	Ing. Michal Kompan	Bc. Ľudovít Mydla (IS)
Téma:	Odporúčanie	Bc. Matej Mihalik (SI)
Názov tímu:	7 out of 10 Precision	Bc. Matej Červeňák (IS)

Obsah

Zoznam skratiek

Skratka	Plné znenie
p.p.	Používateľský príbeh
t.p.	Technický príbeh
p.z.	Požiadavka na zmenu

Tabuľka 1: Zoznam skratiek.

Slovník pojmov

Pojem	Význam
Konzument odporúčania	Systém, ktorý využíva našu webovú službu

Tabuľka 1: Slovník pojmov

1. Úvod

Témou tímového projektu je personalizované odporúčanie, ktoré je v posledných rokoch veľmi diskutovanou témou na vedeckých konferenciách. Úlohou tímu je vytvoriť webovú službu, ktorá bude môcť byť personalizovaná pre každý systém. Používateľ si bude môcť vybrať z viacerých typov a možností odporúčania. V neposlednom rade je tu funkcionalita pridávania nových metód odporúčania.

Projekt je riešený agilnou metodikou tvorby softvéru Scrum. Jeden šprint má určenú dĺžku dvoch týždňov. Dobrým zvykom je šprinty pomenovávať podľa nejakého radu slov ktoré spolu súvisia a dajú sa zoradiť. Naše šprinty sa nazývajú latinskými názvami zvierat zoradené podľa abecedy.

V dokumente sú popísané jednotlivé šprinty a ich používateľské príbehy. Používateľské príbehy sú opísané na úrovni celku a je rozdelený na časti analýza, návrh, opis implementácie a spôsobu testovania.

Na konci dokumentu je opísaný prototyp s dátovým modelom a kofiguráciou servera.

2. Šprinty

V tejto časti sú opísané jednotlivé šprinty a nim zodpovedajúce používateľské príbehy.

2.1. Šprint č. 1: „Anofeles“

ID p. príbehu	Ako	Chcem	Aby bolo možné
02	Používateľ	Získať odporúčanie kolaboratívnym prístupom	Porovnávať odporúčania
06	Používateľ	Vložiť dáta o používateľoch	Aplikovať odporúčanie na existujúci systém
07	Používateľ	Pristupovať k odporúčaniu pomocou webovej služby	Využiť odporúčanie v iných projektoch

Tabuľka 2.1: Používateľské príbehy v šprint backlogu.

ID p. príbehu	Hl. vedúci	ID úlohy	Podúloha	Zodpovedný
T01	Jakub Ševcech	T01.1	Objektový model	Ľudovít Mydla
		T01.2	Návrh – Spätná väzba	Jakub Ševcech
		T01.3	Podporné činnosti	Matej Červeňák
		T01.4	Vytvorenie reprezentácie dokumentov	Martin Detko
		T01.5	Vytvorenie reprezentácie používateľov	Matej Červeňák
02	Michal Cádrik	2.1	Pozrieť existujúce knižnice a riešenia	Igor Slotík
		2.2	Zistenie podobnosti používateľov	Matej Mihalik
		2.3	Implementovanie klastrovacieho algoritmu	Igor Slotík
		2.4	Implementácia algoritmu odporúčania	Michal Cádrik
07	Martin Detko	7.1	Získanie a uploadovanie testovacích dát	Matej Červeňák
		7.2	Proces pridávania	Martin Detko
		7.3	Špecifikácia formy dát	Martin Detko
		7.4	Rozhranie pre používateľa	Jakub Ševcech
06	Jakub Ševcech	6.1	Rozbehanie serveru	Ľudovít Mydla
		6.2	Autentifikácia	Jakub Ševcech
		6.3	Vytvorenie GUI	Jakub Ševcech

		6.4	Nasadzovanie na server	Ľudovít Mydla
		6.5	Vytvorenie API	Jakub Ševcech
T02	Matej Červeňák	T05.1	Databáza	Matej Mihalik
		T05.2	Konzultácia s externou osobou	Matej Červeňák

Tabuľka 2.2: Detailný opis úloh.

2.1.1. Vytvorenie dátového úložiska, t. p. T01

Cieľom tejto úlohy je vytvoriť objektový model, ktorý nám umožní pristupovať ku záznamom v databáze.

Analýza

Pre prístup ku záznamom v tabuľkách sa môžeme vydať dvoma smermi. Buď naprogramovaním vlastného objektového modelu s použitím JDBC, alebo využitím existujúceho nástroja alebo frameworku. Uprednostníme možnosť vygenerovania objektov z existujúcej databázy.

Návrh

Zvolili sme si Java framework Hibernate, ktorý poskytuje prístup a generovanie objektového modelu na základe už existujúcich tabuliek v databáze. Tento framework je veľmi rozšírený a dobre zdokumentovaný, čo uľahčuje jeho nasadzovanie.

Implementácia

Inštalácia frameworku prebieha podľa inštrukcií na wiki v Redmine, ktorá umožňuje Hibernate perspektívu v IDE Eclipse. Následne sa vytvorila databáza ktorej vytvárací skript bol použitý všetkými účastníkmi. Potom sa nastavili konfiguračné súbory pre prístup k databáze a pomocou nástroja sa vygenerovali objekty reprezentujúce záznamy v tabuľkách.

2.1.2. Kolaboratívne odporúčanie, p. p. 02

Cieľom tohto používateľského príbehu je implementovať základnú funkcionalitu kolaboratívneho odporúčania spojenú s určovaním podobnosti používateľov a implementáciou klastrovacieho algoritmu K-means.

Analýza

Odporúčanie založené na podobnosti používateľov, vypočítané pomocou niekoľkých metód, založených na podobných prečítaných dokumentoch. Odporúčajú sa používateľovi dokumenty, ktoré čítali jemu najpodobnejší používatelia z toho istého systému a pri tom ich on sám ešte nečítal. Problémy, ktoré môžu nastať sú pri novom používateľovi v systéme, kedy ešte nevieme určiť, s ktorými používateľmi je podobný. Tento problém nastáva aj pri odporúčaní založenom na obsahu a preto sa riešenie tejto situácie bude implementovať ako samostatná úloha.

Odporúčanie bude musieť byť v budúcnosti čo najlepšie konfigurovateľné pre potreby použitia v systéme a preto je vhodné na to myslieť dopredu. Lebo typ odporúčania by sa mal dať použiť v rôznych prostrediach s rôznymi vlastnosťami.

Systém bude pravdepodobne v budúcnosti pracovať s väčšími objemami dát, položiek. Preto je vhodné vykonávať čo najväčšie množstvo operácií priamo v databáze a nie v kóde programu.

Návrh

Dôležité informácie pre poskytnutie odporúčania sú informácie o používateľoch, identifikácia systému, v ktorom odporúčame a dokumenty, ktoré patria systému. K ostatným tabuľkám databázy nebudeme potrebovať prístup. Snaha je čo najviac operácií implementovať priamo nad databázov, aby sa urýchlili operácie, keďže systém bude bežať v reálnom čase.

Jednotlivé kroky algoritmu by mali nasledovať takto:

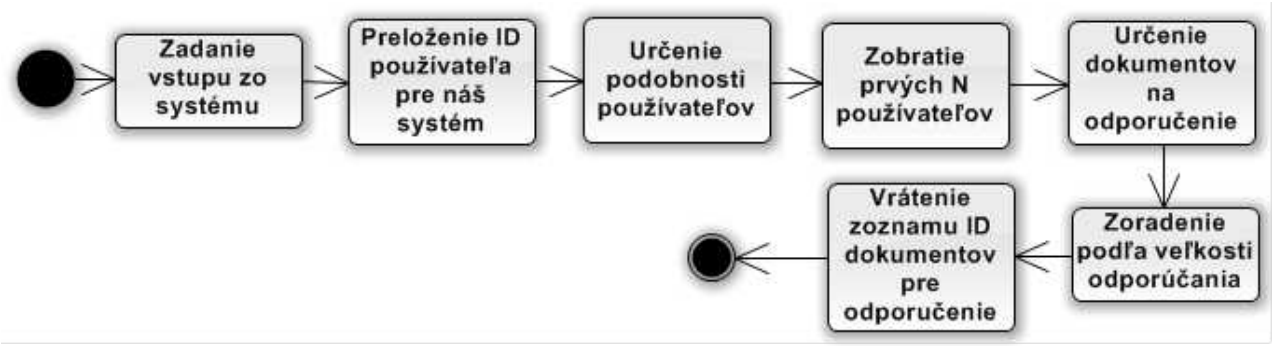
1. Zistenie používateľa, pre ktorého idem odporúčať.
2. Zistenie jemu podobných používateľov v systéme.
3. Získanie odporúčaných dokumentov od každého z podobných dokumentov.
4. Zoradenie dokumentov podľa miery odporúčania.
5. Vrátenie zoradeného zoznamu dokumentov.

Opis implementácie

Systém dostane požiadavku od používateľa s parametrami funkcie kolaboratívneho odporúčania. Hlavné parametre sú identifikácia používateľa v danom systéme, identifikácia systému a nasleduje metóda odporúčania s akou sa pracuje. Ďalšie jednotlivé parametre sú počet podobných používateľov, od ktorých sa má odporúčať, maximálny počet dokumentov, ktorý sa má odporúčať a zoradenie dokumentov. Je to vhodné pri rôznych chápaniach hodnotenia kde niekedy môže nižšia hodnota znamenať lepšie a niekedy to isté môže znamenať vyššia hodnota.

Vstupom do metódy je identifikácia používateľa v systéme, ktorému dodávame odporúčanie. Na začiatku odporúčania sa identifikuje používateľ v našej databáze používateľov pomocou identifikácie používateľa v hosťovskom systéme. Podľa tejto identifikácie a zvolenej metódy určovania podobných používateľov (korelačná, cosínusová, štandardná odchýlka) sa vypočítajú podobnosti používateľov pre daný systém. Následne sa vyberie prvých N najpodobnejších používateľov. Od každého používateľa sa zoberie prvých M dokumentov, z ktorých sa vyhodia dokumenty, ktoré už používateľ, pre ktorého chceme odporučiť, videl. Následne zoradíme zoznam podľa súčtu miest ako boli odporúčané (prvé miesto = najviac bodov). Ak sa dokument odporúča od viacerých používateľov, tak sa tieto hodnoty sčítavajú. Nakoniec vrátime systému zoznam zoradených ID dokumentov. Ak sa také dokumenty nenájdu, alebo je chybný vstup, tak vráti prázdny zoznam ID dokumentov.

Funkcionalita je implementovaná v triede *CollaborativeFiltering* v balíku *sk.stuba.fiit.tp1112.recommender.utils.* Hlavná metóda na volanie celého odporúčania sa nazýva *collaborativeFiltering(...).*



Obrázok 2.1: Diagram aktivít kolaboratívneho odporúčania.

Testovanie

Algoritmus bol otestovaný pomocou vzorky dát lokálnej databázy. Testované boli jednotlivé časti algoritmu ako preloženie ID používateľa z externého systému na ID používateľa nášho systému, jednotlivé metódy pre určenie podobnosti používateľov, vybratie prvých N podobných používateľov, určenie dokumentov pre odporučenie a ich zoradenie. Všetky testy prebehli úspešné.

Jediný problém by mohol nastať pri príkaze výberu dokumentov na odporučenie z databázy, kde je potrebné prediskutovať použitie atribútu *visited*.

2.1.3. Upload dát, p. p. 07

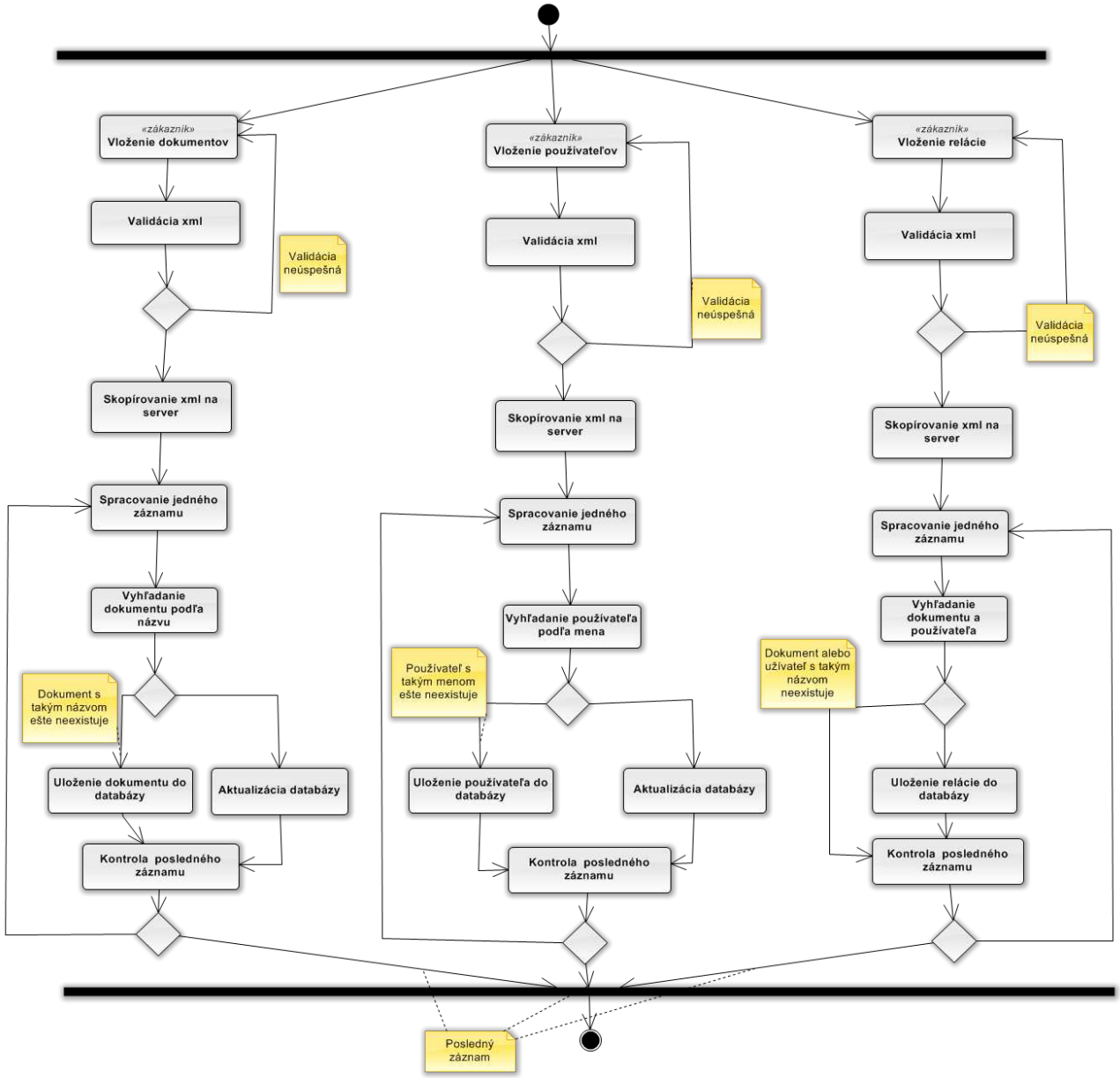
Cieľom tohto používateľského príbehu je umožniť používateľovi nahrať dáta z ich databázy do našej databázy.

Analýza

Keď sa zákazník rozhodne využiť náš systém, je potrebné, ak už nejakú databázu má, aby mohol bez väčšej námahy presunúť dáta do našej databázy. Nie je potrebné, aby sa presúvali všetky dáta. Podobnosti medzi používateľom alebo dokumentom bude schopný spočítať aj náš systém. Ďalšou dôležitou skutočnosťou je fakt, že údaje sa môžu časom meniť a bude potrebné ich aktualizovať. Z hľadiska časovej náročnosti spracovania veľkého počtu dát, je potrebné, aby spracovanie pracovalo v pozadí na serveri a zákazník mohol ďalej vykonávať inú činnosť.

Návrh

Obrázok 2.6 opisuje postupne ako systém spracováva súbor. Systém najprv skontroluje načítaný súbor, ak Xml nie je validné, tak si vyžiada ďalšie. Skopíruje ho na server a začne parsovať po jednom zázname. Ak sa jedná o dokumenty, tak súbor pred dokumentom spracuje najprv jeho kľúčové slová, ak sa jedná o používateľov, tak sú spracované a uložené najprv stereotypy. V prípade dokumentov a používateľov kontroluje, či už daný záznam neexistuje, ak hej tak ho aktualizuje podľa nových informácií. Ak sa jedná relácie medzi dokumentami a používateľmi, tak sa kontroluje, či taký používateľ a dokument vôbec existuje v systéme, ak hej tak sa uloží informácia do systému. V opačnom prípade sa pokračuje na ďalšie záznamy. Keď sú súbory spracované, tak sa vymažú ich dočasné kópie.



Obrázok 2.2: Aktivity diagram: Upload dát

Dôležitou súčasťou návrhu je práve návrh xml schém, ktoré sa nachádzajú v prílohe. Tieto schémy korešpondujú s dátovým modelom a jeho obsahom.

Okrem toho je nutné navrhnúť spôsob, ako sa bude spracovávať súbor na serveri. Toto sa bude riešiť tak, že navrhneme workera, ktorý bude bežať v druhom vlákne, teda na pozadí ostatných činností. Keďže je nutné, aby sa dokumenty a používatelia uložili skôr ako relácie, musí sa jednať o jediného workera. Okrem toho worker musí pracovať s frontou úloh. Je dobré, aby sa tento worker dal využiť aj na iné úlohy, preto bude všeobecný.

Opis implementácie

Worker bol implentovaný ako *singleton*, čím je zabezpečené, aby sa vyskytoval ako jediný. Okrem toho obsahuje frontu ľubovoľných objektov. Keď chce používať tohto *workera* musí jeho úloha byť implementovaná v triede, ktorá má implementovaný interface *Runnable* a samotný kód sa musí

nachádzať v prekonanej metóde *execute*. Pri spúšťaní úlohy sa vytvorí nový objekt tejto vytvorenej triedy a vloží sa metódou *put* do fronty. Následne si ju môže vlákno zobrať a spracovať.

Implementácia načítania, validácie a vytvárania dočasnej kópie bola implementovaná v jruby. Parsovanie xml bolo implementované v jave, na prístup bol využitý jazyk HQL, pomocou ktorého boli vytvorené selekty podľa kľúčového slova z tabuľky klúčových slov; názvu dokumentu a *id* systému z tabuľky dokumentov, *id* dokumentu a *id* kľúčového slova z tabuľky relácií týchto dvoch tabuliek; stereotypu a *id* systému v tabuľke stereotypov; identifikácie používateľa v danom systéme a *id* systému v tabuľke relácií dvoch tabuliek používateľov a systémov; *id* stereotypu a *id* používateľa v tabuľke relácií tabuliek užívateľa a stereotypu. Tieto selekty boli použité na zistenie, či sa v databáze už záznam nachádza. Rozdiel od návrhu spočíva v dvoch skutočnostiach. Najprv bolo nutné získať *id* používateľov a dokumentov a až potom sme boli schopný vytvárať relácie medzi stereotypmi a používateľmi a medzi kľúčovými slovami a dokumentami. Medzi tým, tieto nedokončené relácie boli ukladané do listu relácií. Až po spracovaní jedného záznamu z xml,či už dokumentu alebo používateľa boli do databázy pridaný aj ich relácie. Ďalšia zmena, ktorá vyplýva z dátového modelu, spôsobuje, že najprv sa uloží záznam o všeobecnom používateľovi, a až potom sa vytvorí relačná entita medzi používateľom a systémom. Ostatné časti boli implementované podľa návrhu.

Testovanie

Testovanie workera bolo zamerané na to, či sa úlohy spracovávajú naozaj podľa poradia. Jednotlivé časti importu dát boli testované Junit testami. Snahou bolo pokryť všetky možné prípady. Okrem toho testovanie kompletnej funckionality prebehlo najprv s načítaním malých komplexných vymyslených vstupoch a neskôr aj na skutočných dátach veľkého rozsahu.

2.1.4. Webová služba, p. p. 06

Analýza

Je potrebne vytvoriť rozhranie pre používateľa aby vedel:

- vykonávať všetky administratívne činnosti,
- aby vedel nastavovať, vymieňať, kombinovať rôzne odporúčacie algoritmy,
- aby mal jednoduchý a pohodlný prístup k najrôznejším potrebným informáciám,
- aby vedel automaticky spracovať odpoveď získanú z odporúčača.

Návrh

Rozhranie webovej služby bude zložené z dvoch častí:

- Grafické rozhranie v podobe webovej stránky na nastavovanie najrôznejších atribútov a na získavanie rôznych informácií o tom ako používať vytváranú aplikáciu.
- REST API na získavanie samotných odporúčaní. Služba bude na základe poskytnutých nastavení a apikľúču vracať XML dokument s vytvorenými odporúčaniami.

Opis implementácie

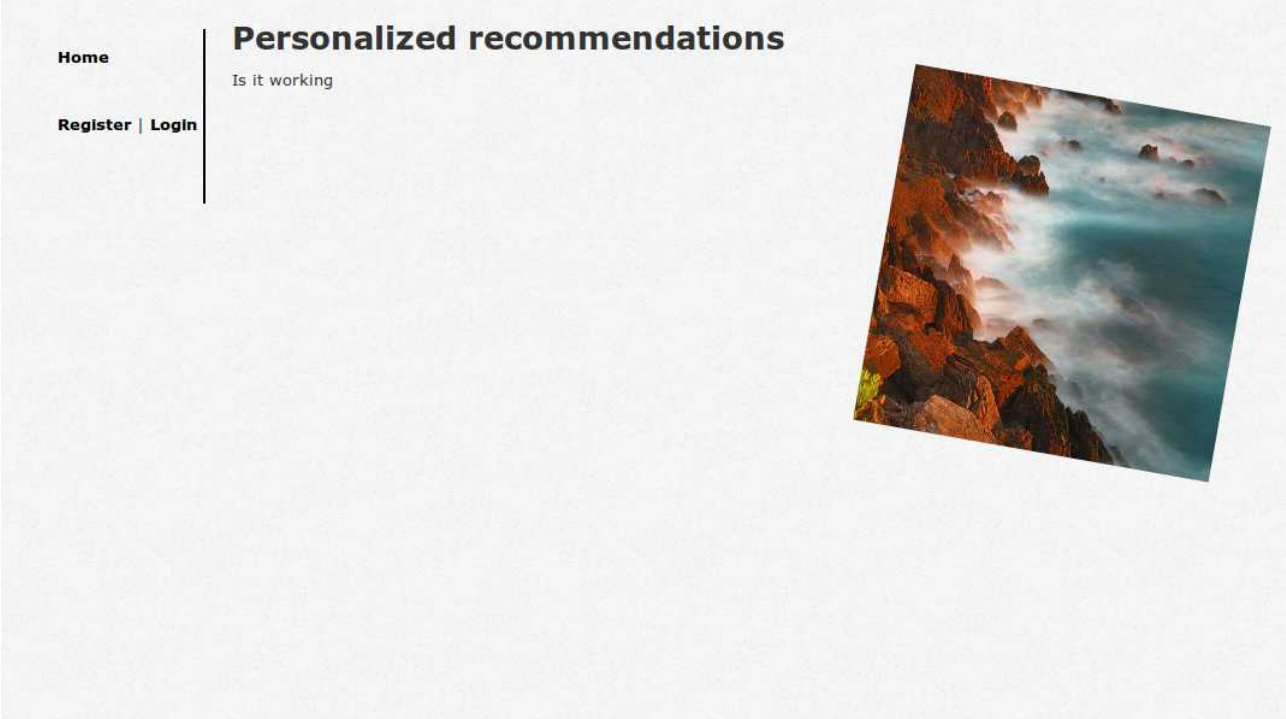
Pri implementácií sme používali rôzne technológie na vývoj. Na začiatku sme použili kombináciu

Jruby a Java. Dočasne sme zmenili použité technológie na samotnú Javu, s použitím frameworku Spring. Následne sme sa vrátili ku kombinácií Jruby a Java. Výsledná aplikácia používa na úrovni Jruby MVC framework Ruby on Rails. Samotná logika odporúčania a odporúčacie algoritmy sú implementované v jazyku Java. Implementované algoritmy odporúčania sú spúšťané a sprístupňovanie pomocou webovej služby volaním z jazyka Jruby.

Nasadená implementácia (Obrázok 2.5) je dostupná na adrese: *http://vm28.ucebne.fiit.stuba.sk/app/*.

Implementácia obsahuje rozhranie pre registrovanie nového systému (Obrázok 2.4), aby mohol využívať služby odporúčača.

Po zaregistrovaní je každému systému priradený jedinečný apikľúč (Obrázok 2.3), ktorý bude používať ako identifikáciu pri využívaní jednotlivý API volaní.



Obrázok 2.3. Úvodná stránka implementovanej webovej služby

Obrázok 2.4. Obrazovka registrácie nového systému, aby mohol mať prístup k API

Obrázok 2.5. Obrazovka zobrazujúca detail prihláseného systému, spolu s jeho API kľúčom

Testovanie

Implementácia bola otestovaná manuálnym preklikaním všetkých prechodov a stavov.

2.1.5. Analýza pridávania algoritmov, t. p. T02

Cieľom tohto používateľského príbehu je analýza jednotlivých pridávaných algoritmov. Jej úlohami je konzultácia s externou osobou a zistenie, či je možné dynamicky upravovať práva používateľa nad zvolenou databázou, alebo jej časťami.

Analýza

Keďže chceme poskytnúť používateľom možnosť pridávať do systému vlastné odporúčacie, či porovnávacie algoritmy, a tento proces sa snažíme čo najviac automatizovať, je potrebné zvážiť aké následky to môže mať na náš systém ako celok.

Jednou z oblastí ktoré treba ochrániť pred zlomyseľným konaním používateľov je databázový systém, zároveň však treba dbať na to aby neboli obmedzované práva štandardných používateľov.

Cieľom tohto používateľského príbehu je zistiť, aké sú možnosti pridávania odporúčacích algoritmov do nášho systému, aby sa dosiahlo zautomatizovanie. Ďalej je potrebné preskúmať, či a ako je možné dynamicky upravovať práva používateľov nad naším databázovým systémom tak, aby bol proces pridávania vlastných algoritmov do nášho systému bezpečný, a zároveň automatizovateľný.

Návrh

Konzultovať problematiku s expertom, ktorý má v oblasti pridávania odporúčacích algoritmov skúsenosti. Preštudovať online dokumentáciu k systému PostgreSQL, respektíve ďalšie vhodné online zdroje, a nájsť čo najvhodnejšie riešenie problému.

Implementácia

K implementácií tohto používateľského príbehu nakoniec neprišlo. Po odporúčaní nášho projektového vedúceho sa celý tím zhodol, že najlepšou ochranou pri poskytovaní funkcionality vkladania vlastných algoritmov do systému bude, keď sa nebudeme snažiť tento proces automatizovať. Bude existovať jedna, prípadne viac osôb, ktoré budú ručne kontrolovať dodávané algoritmy a ručiť tak za ich nezávadnosť a bezpečnú implementáciu.

2.2. Šprint č. 2: „Bubo Bubo“

ID p. príbehu	Ako	Chcem	Aby bolo možné
01	Používateľ	Získať odporúčanie prístupom založeným na obsahu	Porovnávať odporúčania
08	Používateľ	Pre konkrétneho používateľa získať odporúčanie definovaným spôsobom	Odporúčať
09	Používateľ	Po odporúčaní bolo o každom používateľovi možné získať spätnú väzbu a definovať či sa jedná o explicitnú alebo implicitnú	Zohľadniť ich pri odporúčaní
10	Používateľ	Rátať podobnosť používateľov a dokumentov rôznymi spôsobmi	Skúmať vplyv podobnosti na odporúčanie
13	Používateľ	Získavať vyhodnotenie, s akou úspešnosťou je generované odporúčanie	Porovnávať metódy

Tabuľka 2.3: Používateľské príbehy šprintu č. 2

ID p. príbehu	Hl. vedúci	ID úlohy	Podúloha	Zodpovedný
01	Ľudovít Mydla	1.1	API	Micha Cádrik
		1.2	Implementácia podobnosti	Ľudovít Mydla
08	Michal Cádrik	8.1	Správa konzumentov	Jakub Ševcech
		8.2	Poskytnutie odporúčania	Michal Cádrik
09	Jakub Ševcech	9.1	Možnosť spätnej väzby a otypovanie. API + GUI	Martin Detko
10	Jakub Ševcech	10.1	Spraviť API	Jakub Ševcech
		10.2	Implementačná logika používateľov	Matej Mihálik
		10.3	Implementačná logika dokumentov	Matej Mihálik
13	Igor Slotík	13.1	GUI	Matej Červeňák
		13.2	Metrika -Vyhodnotenie	Igor Slotík
		13.3	Logovanie	Martin Detko

Tabuľka 2.4: Detailný opis úloh

2.2.1. Odporúčanie založené na obsahu, p.p. 01

Ako používateľ chcem získať odporúčanie content-based prístupom, aby bolo možné porovnávať metódy.

Cieľom tejto úlohy je implementovať logiku odporúčania založeného na obsahu. Každý dokument je reprezentovaný metadátami, kľúčovými slovami, obsahom a nadpisom. Táto úloha využíva výsledky iných používateľakých príbehov, ktoré vyrátavajú podobnosť dokumentov ku používateľom. Ďalej je potrebné sprístupniť túto službu vonkajšiemu svetu pomocou API.

Analýza

Naštudovali sme si problematiku odporúčania založeného na obsahu, z ktorej sme pochopili nutnosť implementácie výpočtu obľúbenosti dokumentu pre používateľa, čo je obsahom iných úloh. Tie by sme mali využiť v tejto úlohe.

Návrh

Navrhujeme implementáciu, ktorá vyberie najvhodnejšie dokumenty podľa váh v tabuľke *user_to_document_relations* za predpokladu, že pre všetky dokumenty máme už určenú obľúbenosť u konkrétneho používateľa na základe vypracovaných metód, táto úloha spočíva vo výbere dokumentov s najväčšími váhami.. A usporiadané ich vráti ako výsledok odporúčania. Službu sprístupníme verejnosti pomocou API.

Opis implementácie

Odporúčanie na základe obsahu sa deje výberom dokumentov s najväčšími váhami z databázy. Predpokladáme, že už sú vyrátané vyššie spomenutými taskami. Výsledkom je usporiadaný zoznam dokumentov. API sa nakoniec neimplementovalo, keďže sme opäť prešli na platformu Ruby.

Testovanie

Zatiaľ funkcionalita nebola testovaná.

2.2.2. Získanie odporúčania, p.p. 08

Ako používateľ chcem pre konkrétneho používateľa získať odporúčanie definovaným spôsobom, aby mu bolo možné odporúčať.

2.2.3. Spätná väzba, p.p. 09

Ako používateľ chcem aby po odporúčaní bolo o každom používateľovi možné získať spätnú väzbu a definovať či sa jedná o explicitnú alebo implicitnú, aby ich bolo možné zohľadniť pri odporúčaní.

Analýza

- Je potrebné uchovávať všetky vypočítané odporúčania aby sme ich dokázali spätne spojiť so získanou spätnou väzbou.

- Je treba vytvoriť možnosť na vkladanie spätnej väzby do systému.
- Môže sa vyskytnúť problém s motiváciou pre poskytovanie aj spätnej väzby. To ale vyrieši fakt že každý, ktorý chce používať naše odporúčanie, bude chcieť aj vyhodnotenie jeho kvality a úspešnosti. Ak teda nebude poskytovať spätnú väzbu, bude nemožné spätne vyhodnotiť kvalitu odporúčania.
- Každé jedno odporúčanie musí byť jedinečne označené a túto značku musí dostať aj používateľ, aby nám vedel podľa toho poslať spätnú väzbu.
- Návrh dátového modelu musí byť spravený tak, aby sa v ňom dali reprezentovať rôzne formy spätnej väzby tj. implicitná/explicitná. Zároveň treba zohľadňovať rôzne formy hodnotenia, tj. bodovanie, áno/nie, pozitívne/negatívne hodnotenie, like …
- Treba zistiť kto vytvára nové typy spätnej väzby. 4I je to len nejaká na začiatku stanovená sada typov, alebo sa priebežne mení alebo si ju vytvára používateľ sám?

2.2.4. Výpočet podobnosti, p.p. 10

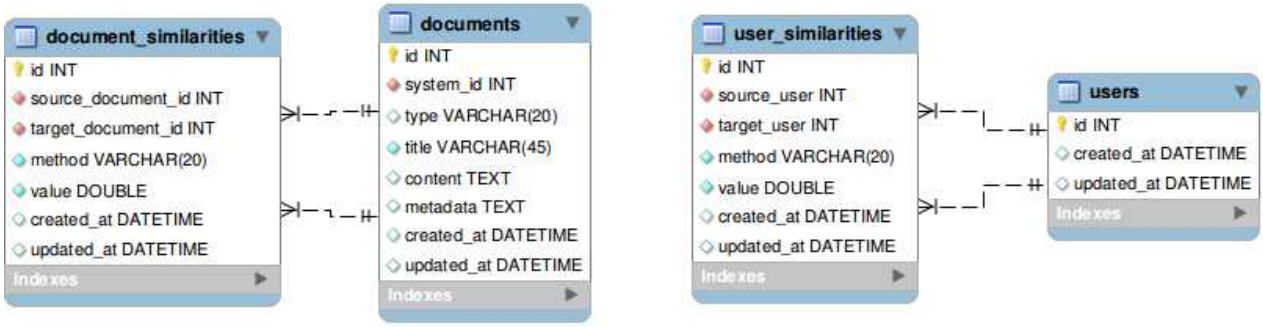
Ako používateľ chcem rátať podobnosť používateľov a dokumentov rôznymi spôsobmi, aby som mohol skúmať vplyv podobnosti na odporúčanie.

Analýza

- Je potrebné vytvoriť jednotný spôsob pre výpočet podobnosti. Úloha je zložená z dvoch častí: výpočet podobnosti dokumentov a výpočet podobnosti používateľov. Tieto časti sú do istej miery ekvivalentné, keďže v oboch ide o výpočet podobnosti medzi dvoma vektormi hodnôt.
- Je potrebné implementovať viacero metód na výpočet podobnosti, aby sa dali pri ich ďalšej aplikácií vymieňať
- Vypočítané hodnoty je potrebné uchovávať, aby nebolo potrebné ich opakovane prepočítavať. Toto by mohlo napríklad pri dokumentoch, ktoré majú veľký počet atribútov, spôsobiť problémy spojené s výkonnosťou riešenia.
- V tejto úlohe to nieje priamo spomenuté, ale v budúcnosti bude potrebné riešiť nejakú rovnováhu medzi opakovaným prepočítavaním podobností a uchovávaním hodnôt v databáze. Opakovaný výpočet bude s rastúcim množstvom údajom časovo náročný a uchovávanie všetkých prepojené bude tak isto nemožné.

Návrh

Už v predchádzajúcom šprinte sme navrhli dátový model na reprezentáciu používateľov a dokumentov. Podobnosť používateľov prípadne dokumentov sa uchováva v prepojovacej tabuľke *user_similarities* respektíve *document_similarities*. Záznam v tabuľke podobností je definovaný zdrojovým a cieľovým používateľom/dokumentom. Vzťah podobnosti je komutatívny, takže nezáleží na poradí, ktorý prvok je zdrojový a ktorý je cieľový. Každý záznam v tabuľke podobností je otypovaný atribútom *method*, na základe ktorého je možné určiť metódu použitú na výpočet podobnosti.



Obrázok 2.6. Časť dátového modelu, zachytávajúca podobnosti dokumentov a podobnosti používateľov

Opis implementácie

Po preštudovaní zdrojov som si zvolil cosínusový algoritmus, a algoritmus štandardnej odchýlky pre modelovanie podobnosti medzi používateľmi, a korelačný a cosínusový algoritmus pre modelovanie podobnosti dokumentov na základe vážených kľúčových slov, a potom jednoduchý algoritmus ktorý nebierie do úvahy váhu kľúčových slov. Implementácia pri podobnosti používateľov je riešená tak, že sa vytvorila jedna otcovská trieda *UserSimil*, ktorá združuje všetky metódy na výpočet podobnosti. To s ktorou metódou bude inštancia triedy pracovať sa stanoví pomocou argumentu pri jej inicializácií. Všetky metódy sa potom volajú nad touto otcovskou triedou, ktorá deleguje prácu triedam s konkrétnou implementáciou algoritmu. Všetky potrebné triedy sú združené v balíku *sk.stuba.fiit.tp1112.recommender.utils.userSimilarities*. Implementácia podobnosti medzi dokumentami je riešená analogicky, s tým rozdielom že otcovská trieda nesie názov *DocumentSimil* a všetky potrebné triedy sa nachádzajú v balíku *sk.stuba.fiit.tp1112.recommender.utils.documentSimilarities*.

Testovanie

2.2.5. Vyhodnotenie, p.p. 13

Ako používateľ chcem získavať vyhodnotenie, s akou úspešnosťou je generované odporúčanie, aby bolo možné porovnávať metódy.

2.3. Šprint č. 3: „Canis“

ID p. príbehu	Ako	Chcem	Aby bolo možné
05	Používateľ	Jednoducho pridávať ďalšie vlastné algoritmy	Experimentovať a vyhodnocovať ďalšie prístupy

Tabuľka 2.5: Používateľské príbehy šprintu č. 3

ID p. príbehu	Hl. vedúci	ID úlohy	Podúloha	Zodpovedný
05	Jakub Ševcech	5.1	Core mechanika systému	Ľudovít Mydla
		5.2	Návrh konvencie balíkovania, odporúčania	Igor Slotík
		5.3	Spísanie metodiky	Michal Cádrik
		5.4	Refaktoring štruktúry projektu	Jakub Ševcech
T03	Ľudovít Mydla	T03.1	Refaktoring dátového modelu	Matej Červeňák
		T03.2	Refaktoring databázy	Jakub Ševcech
		T03.3	Prepojenie uploadu a výpočtu podobnosti	Martin Detko
		T03.4	Testovacia databáza	Matej Mihalik
		T03.5	Kolaboratívny refaktoring	Ľudovít Mydla

Tabuľka 2.6: Detailný opis úloh

2.3.1. Jednoduché pridávanie ďalších algoritmov, p.p. 05

Ako používateľ chcem jednoducho pridávať ďalšie vlastné algoritmy, aby som mohol experimentovať a vyhodnocovať ďalšie prístupy.

Analýza

- Potrebujeme spôsob, ako jednoducho pridávať nové algoritmy a odporúčače, tak aby si mohol používateľ sám nejaké vytvoriť a pridať ich.
- Potrebujeme, aby mal používateľ prístup k základným funkciám, ako napríklad funkcie na prácu s databázou a podobne.
- Potrebujeme, aby pridávanie algoritmov bolo bezpečné, teda aby používateľ nemohol zhodiť celý náš systém tým, že vytvorí chybný kód alebo, že vytvorí kód, ktorý bude chcieť zhodiť náš systém.
- Potrebujeme používateľovi obmedziť prístup k databáze, tak aby mohol pracovať len so svojimi údajmi, a aby napríklad nemohol zmazať celú databázu alebo dáta ostatných používateľov.
- Celé pridávanie algoritmov musí byť postavené na nejakom API, ktoré budeme

používateľovi poskytovať, a ktoré môže vo svojich implementáciách použiť.

- Ruby je interpretovaný jazyk a dokáže dynamicky spúšťať kód, ktorý mu podsunieme. To znamená, že vieme za behu programu meniť kód a jazyk to dokáže spracovať. Toto by sme mohli využiť, aby bolo možné pridávať algoritmy aj za behu programu.

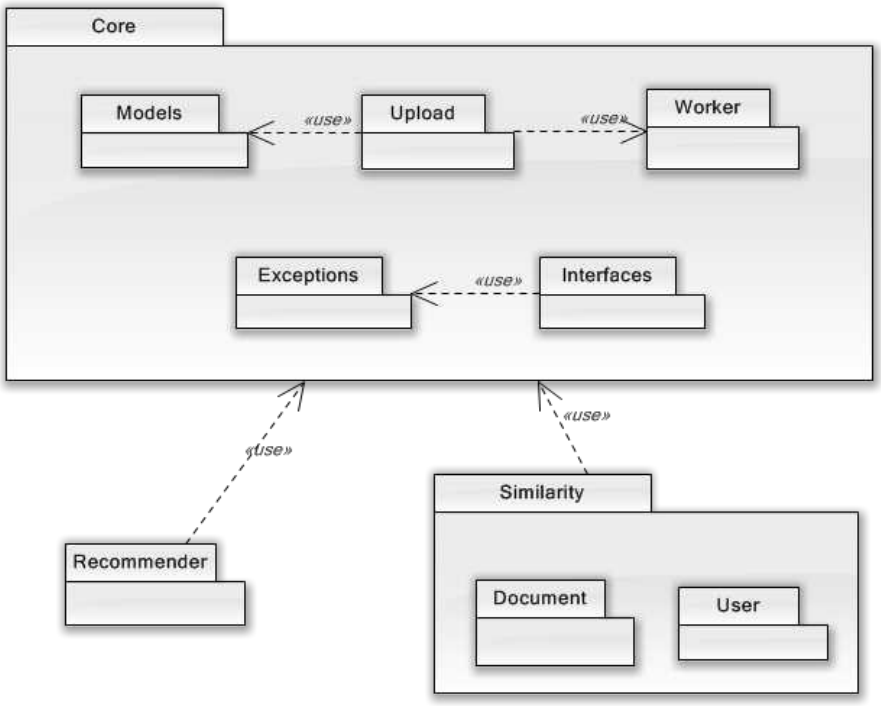
Návrh

Na poskytnutie možnosti jednoduchého pridávania nových funkcií použijeme možnosť pridávania nových funkcií v podobe pluginov za behu programu. Na to musíme zmeniť štruktúru projektu tak, aby bolo možné vytvárať nové funkcie ako pluginy a musíme implementovať funkcie na manažovanie a spúšťanie týchto pluginov.

Vytvoríme jadro systému v Jave, ktoré bude poskytovať rozhranie pre používateľa a ten ho môže používať vo svojich algoritmoch. Toto jadro bude obsahovať napríklad niektoré najzákladnejšie algoritmy, funkcie na prístup k databáze. Jadro bude možné zbaliť do jedného .jar súboru, a tak ho poskytnúť používateľovi.

Všetky algoritmu na výpočet odporúčania, výpočet podobnosti a podobne budeme ďalej implementovať vo forme pluginov, teda samostatných .jar súborov, ktoré budú využívať funkcie jadra.

Novú štruktúru projektu je potrebné navrhnúť tak, aby boli jednotlivé balíky dobre definované a každý zastrešoval iba jednu kľúčovú funkcionalitu systému, pričom sa treba snažiť eliminovať ich vzájomnú závislosť na čo najmenšiu možnú mieru. Samozrejme, túto závislosť nie je možné úplne odstrániť (napríklad, balíky ktoré potrebujú prístup k databáze budú závislé od balíka, ktorý bude zastrešovať funkcionalitu nad databázou). Balíky s odporúčaniami a výpočtami podobnosti nebudú zahrnuté v základnom core balíku, nakoľko takéto algoritmy budú môcť pridávať aj používatelia nášho systému, a teda by nemali byť súčasťou samotného jadra systému. Návrh našej novej štruktúry je zrejmý z obrázka číslo 2.7.



Obrázok 2.7: Architektúra balíkov v jadre systému

Časť aplikácie implementovaná v jazyku Java je rozdelená na dve základné časti:

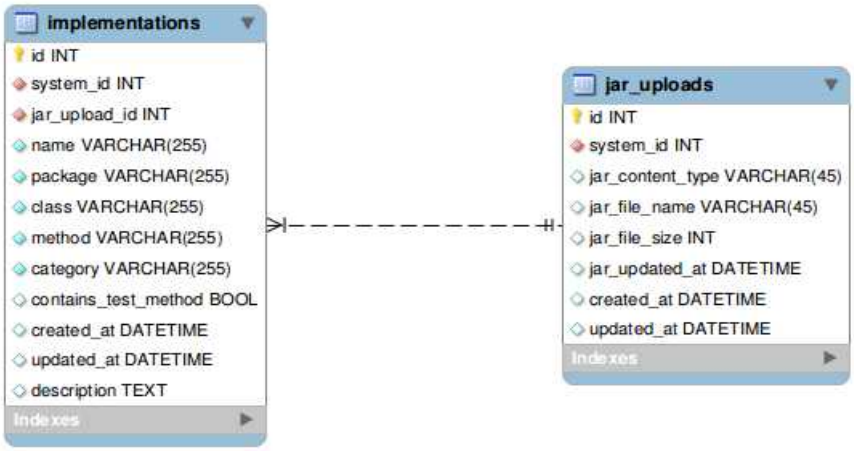
- jadro (Core) aplikácie a
- balíky ostatných častí, teda pluginov.

Jadro poskytuje všetky základné funkcie a pluginy tieto funkcie používajú.

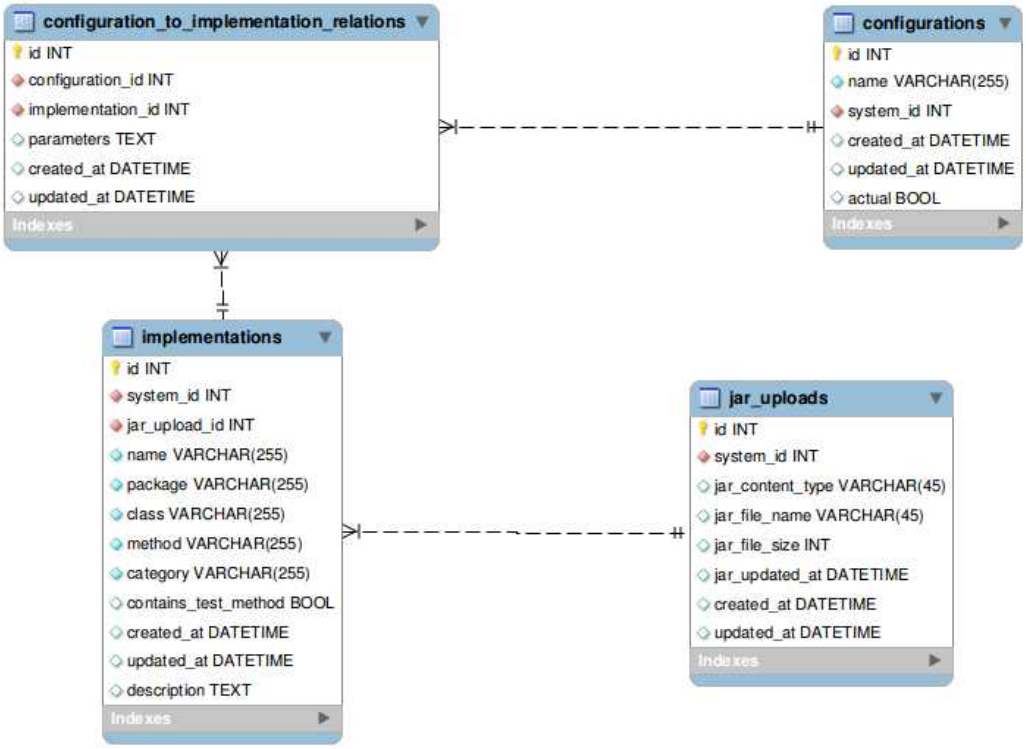
Ďalej vytvoríme funkcie, ktoré zabalia priamy prístup k databáze pre používateľov. Cieľom nášho nového návrhu pre prístup k databáze je limitovať prístup externého kódu k našej databáze z dôvodu zvýšenej bezpečnosti. Ďalším cieľom nového návrhu je schovať prácu s identifikáciou systému, tak aby sa nemusel pri každom volaní nejakej funkcie nad databázou explicitne zadávať ako argument id systému, pre ktoré sa daná funkcia volá. Systém musí teda transparentne dostávať len svoje údaje bez toho, aby si bol vedomí toho, že ich žiadal a nemohol získať prístup k údajom, ktoré mu nepatria. Dosiahnutie týchto cieľov sa skladá z niekoľkých krokov.

- Obmedziť prístup zvonka k objektovému modelu nad databázou.
- Ponechať však prístup k samotným objektom tak, aby bolo možné zachovať prácu s databázovými objektami (Teda napríklad trieda *document* bude stále prístupná, aby sme mohli pracovať s dokumentom v našom systéme ako s objektom, ale funkcie tejto triedy, ktoré ovplyvňujú databázu nebudú verejne dostupné).
- Vytvoriť triedu alebo triedy (zatiaľ však postačí jedna), ktoré budú poskytovať a ošetrovať verejne dostupné volania nad našou databázou.
- Vytvoriť factory objekt, ktorý bude vytvárať inštancie verejne dostupných tried pre prácu s databázou tak, že všetky tieto inštancie budú mať registrovaní systemId, s ktorým majú pracovať.
- Vytvoriť triedu, ktorú budú klientské triedy z externých kódov rozširovať a bude slúžiť na to, aby sme pomocou nej vedeli nastaviť systemId, pomocou ktorého bude factory objekt inicializovať inštancie verejne dostupných tried pre prácu s databázou. Hlavná trieda každého pluginu musí dediť od abstraktnej triedy. Vďaka tomu vieme túto triedu registrovať a tak jej dať prístup k časti databázy.

Používateľ bude implementovať svoje algoritmy tak, že vytvorí plugin, ktorý bude využívať funkcie jadra a nahrá ho do systému. V databáze budeme uchovávať informácie o vytvorenej implementácií, spolu s cestou k .jar súboru. Časť dátového modelu reprezentujúca tabuľku uchovávajúcu informácie o implementácii a informácie o zodpovedajúcom .jar súbore je na obrázku číslo 2.8. Tieto implementácie bude vedieť používateľ spájať a tak vytvoriť konfiguráciu, ktorá bude určovať algoritmy, ktoré sa budú používať na výpočet odporúčania. Časť dátového modelu reprezentujúca konfigurácie je na obrázku číslo 2.9.



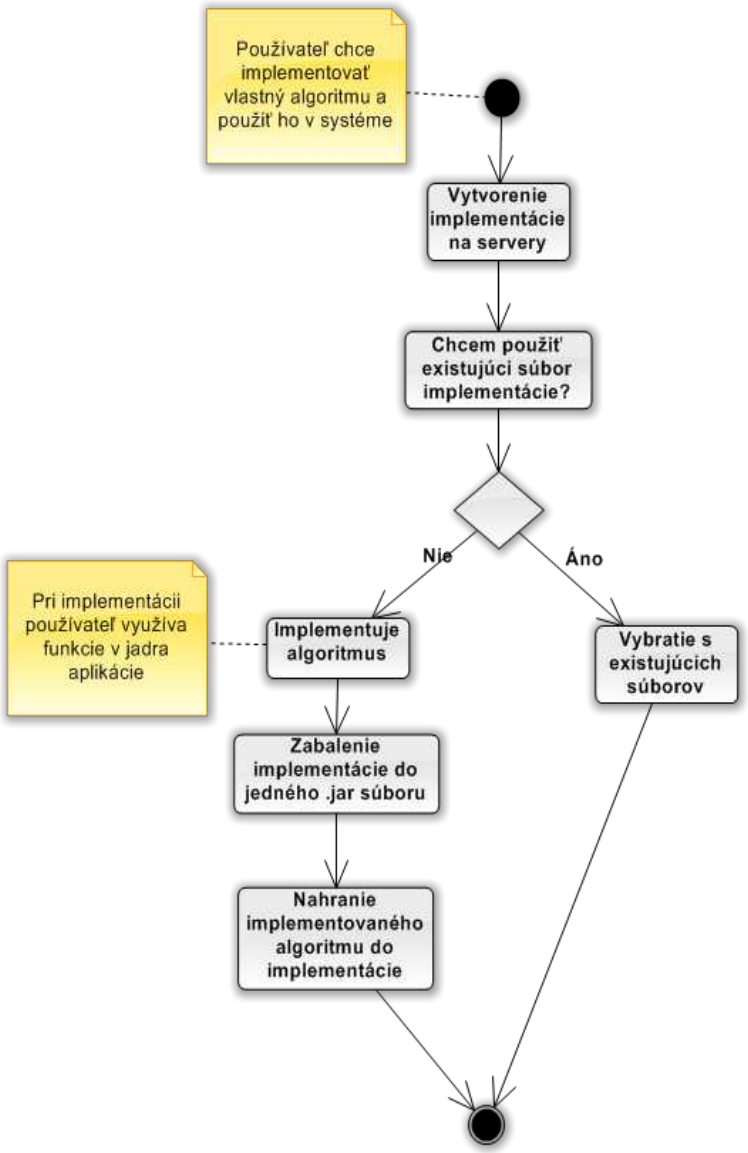
Obrázok 2.8: Časť dátového modelu zachytávajúca reprezentáciu implementácie a priradeného súboru



Obrázok 2.9: Časť dátového modelu zachytávajúca reprezentáciu konfigurácie

Každý používateľ (systém), ktorý bude chcieť získavať odporúčanie vytvorí konfiguráciu, v ktorej nastaví detaily použitých algoritmov na výpočet odporúčania. Túto konfiguráciu bude ďalej používať na získavanie odporúčania. Jeden systém môže mať vytvorených viacero konfigurácií, potom bude vedieť získavať odporúčanie podľa viacerých kombinácií algoritmov. Toto môže byť užitočné napríklad pri porovnávaní a vyhodnocovaní algoritmov.

Proces pridávania algoritmov do systému a vytváranie implementácie je opísaný na diagrame na obrázku číslo 2.10.

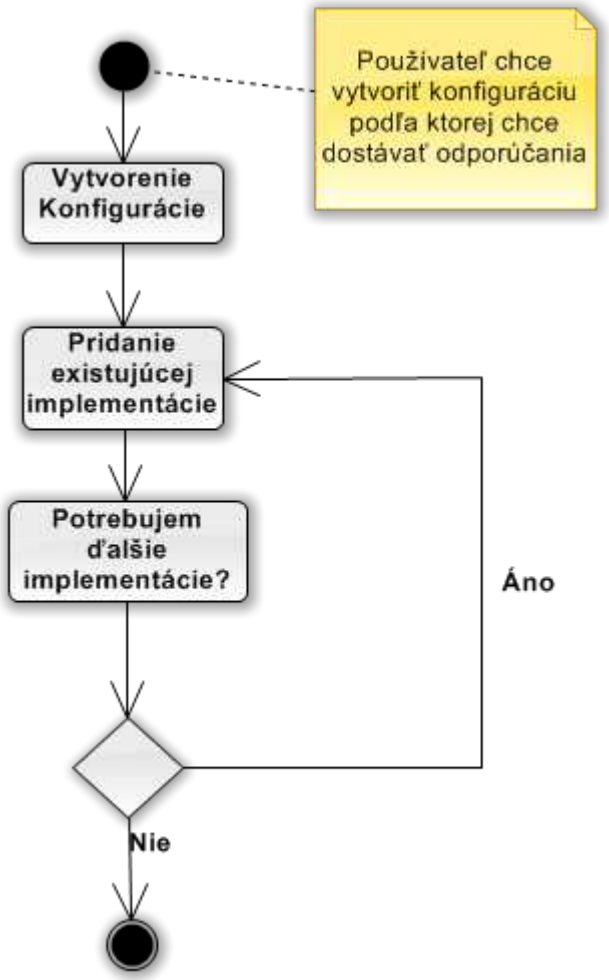


Obrázok 2.10: Proces implementovania algoritmu a vytvárania implementácie v systéme

Ak chce používateľ použiť nový algoritmus v systéme potrebuje na to vytvoriť reprezentáciu implementácie v systéme. Vytvorí teda pomocou grafického prostredia prázdnu reprezentáciu implementácie. Teraz sa musí rozhodnúť či chce použiť už raz nahraný súbor na servery, alebo chce použiť vlastný. Ak chce použiť existujúci súbor, tak si ho zvolí zo zoznamu a priradí k implementácii. Tu proces končí. Ak sa používateľ rozhodne použiť vlastný algoritmus, tak ho musí najskôr implementovať v jazyku Java. Pri implementácii môže použiť funkcie jadra aplikácie. Po implementovaní algoritmu zabalí výsledný program do jedného .jar súboru a nahrá ho do vytvorenej implementácie. Tu proces konči s vytvorenou implementáciou.

Táto implementácia sa dá následne použiť v niektorej z konfigurácii. Každý systém má viditeľné a môže použiť všetky implementácie, aj tie ktoré mu nepatria, avšak upravovať môže len tie, ktoré sám vytvoril.

Konfigurácia definuje kombináciu algoritmov, ktorú chce používateľ (systém) použiť, ak chce získavať odporúčanie. Proces vytvorenia konfigurácie je znázornený na diagrame aktivít na obrázku číslo 2.11. Proces začína vytvorením prázdnej konfigurácie. Ku každej konfigurácií je potrebné stanoviť meno, na základe ktorého ju bude používateľ identifikovať. Do vytvorenej konfigurácie je možné pridávať implementácie. Do konfigurácie je možné priradiť ľubovoľnú konfiguráciu, teda aj implementácie iných používateľov. Ku každej implementácií je možné špecifikovať dodatočné parametre.



Obrázok 2.11: Proces vytvárania konfigurácie

Opis implementácie

Navrhnuté funkcie sme implementovali v dvoch častiach:

Java

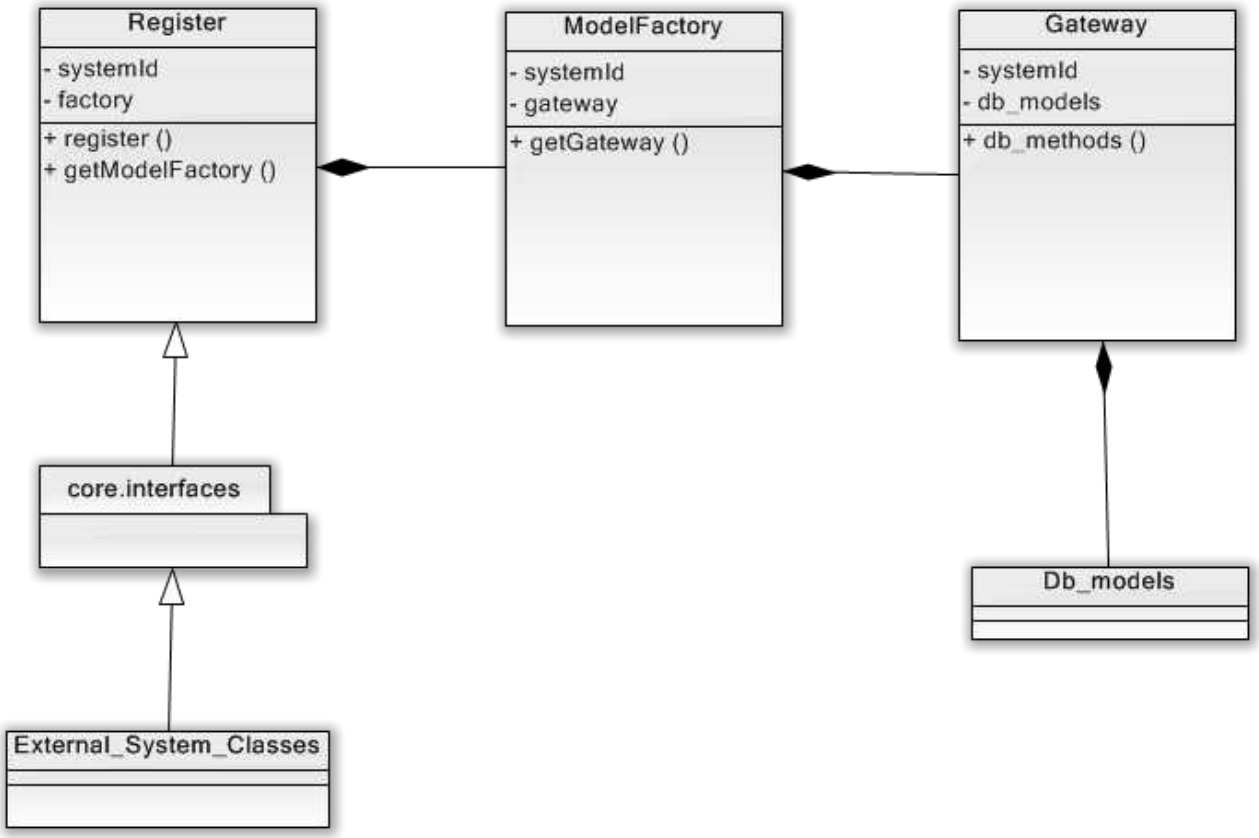
Táto časť obsahuje zmenu v štruktúre balíkov v časti aplikácie implementovanej v jazyku Java. Celý projekt sa nachádza v balíku *sk.stuba.fiit.tp1112.recommender* a obsahuje všetky balíky vyplývajúce z diagramu na obrázku číslo 2.7. Nasleduje stručný opis všetkých týchto balíkov:

- *Core* – Balík obsahujúce kľúčove funkcie nášeho systému, pre externí aplikácie je jeho

import nevyhnutný pre správne fungovanie našej služby.

- *Interfaces* – Obsahuje interface-i a abstraktné triedy využívané v našom systéme, ale aj tie, ktoré sú nevyhnutné pre správnu funkcionalitu externého klientského kódu.
- *Exceptions* – Vlastné výnimky využívané v našom sytéme.
- *Models* – Obsahuje všetky triedy nevyhnutné pre prácu s databázou.
- *Upload* – Balík poskytujúci funkcionalitu pre upload externých dát do našeho systému.
- *Worker* – Balík poskytujúci funkcionalitu pre plánovanie jednotlivých (časovo a výpočtovo náročnejších) úloh.
- *Recommender* – Obsahuje funkcionalitu jednotlivých odporúčaní.
- *Similarity* – Zastrešuje funkcie potrebné pre výpočet podobnosti.
- *Document* – Špecializuje sa na výpočty podobnosti medzi dokumentami.
- *User* – Obdobne ako balík *Document*, ale slúži pre výpočty podobnosti medzi používateľmi.

Pri implementácií sme vychádzali z návrhového diagramu na obrázku číslo 2.12.



Obrázok 2.12: Diagram tried zabezpečujúcich prístup pluginu k databáze

Register – Trieda ktorú musia klientske triedy rozširovať, aby nad nimi bolo možné realizovať správne volania na databázu.

ModelFactory – Factory trieda, ktorá inicializuje verejné triedy pre prístup k databáze.

Gateway – Verejná trieda poskytujúca prístup k databáze. Inicializuje si jednotlivé modely pre prácu s databázou, ktoré inak nie sú verejne dostupné. Tieto modely potom využíva pri poskytovaní svojich služieb. Takúto triedu máme zatiaľ jednu, pokiaľ zistíme, že by bolo vhodné mať ich viac, je tento systém jednoducho rozšíriteľný.

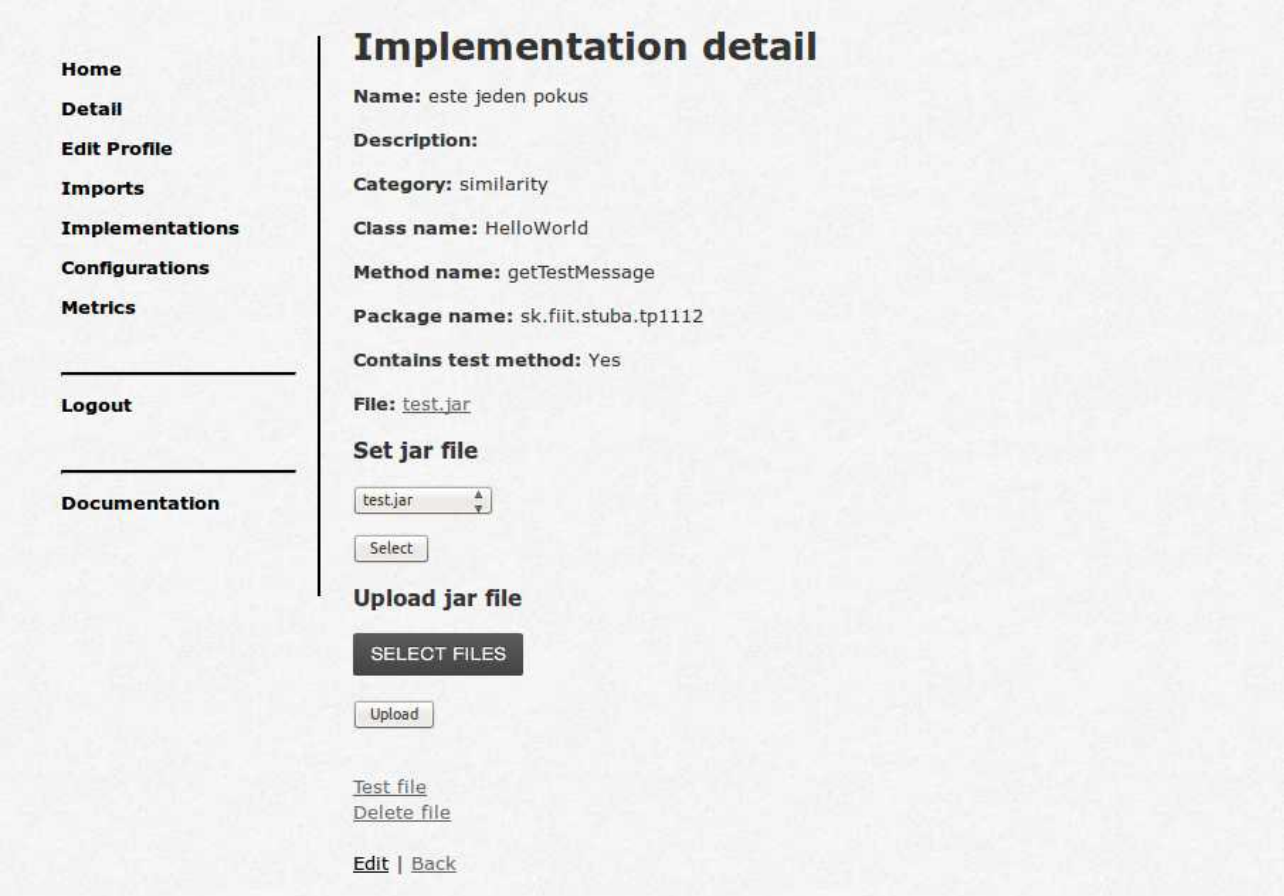
V praxi tento systém funguje tak, že z ruby sa nad inštanciou klientskej triedy zavolá metóda register, ktorá inicializuje systemId a podľa neho aj inštanciu ModelFactory. Ďalej už potom len stačí volať jednotlivé metódy cez Gateway triedu, ktorú poskytuje práve ModelFactory.

Čiastočné schovanie objektového modelu nad databázou sme zrealizovali tak, že jednotlivé modely sú verejne viditeľné, ale ich metódy pre prácu s databázou, rovnako ako set metódy sú nastavené ako protected, sú dostupné iba z balíka core.models. Samotné modely a ich get metódy sú však naďalej public, aby sa mohli využívať ako dátové entity pri práci s databázou.

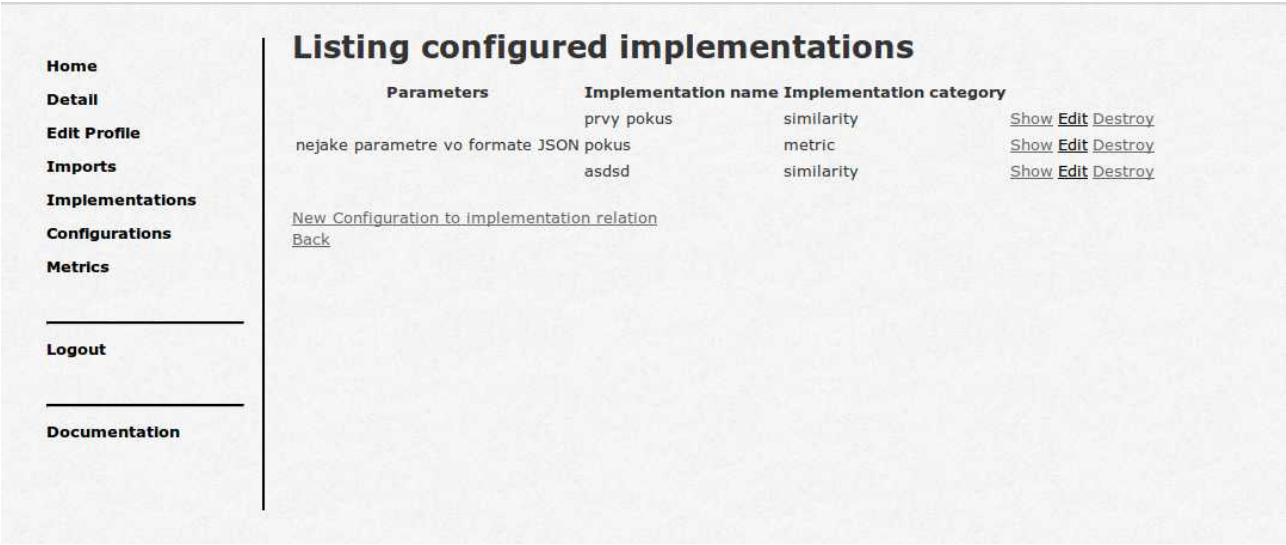
Po úprave štruktúry balíkov a prístupu k databáze sme boli nútení vykonať veľa zmien v skôr implementovaných častiach aplikácie. Preto sme vykonali úplný refactoring a previedli sme implementované algoritmy do podoby pluginov.

Ruby

V časti aplikácie implementovanej v jazyku Ruby sme vytvori rozhrania na vytváranie implementácií a konfigurácií. Časti rozhrania na vytváranie implementácií a konfigurácií sú na obrázkoch číslo 2.13 a 2.14. Implementovali sme nahrávanie súborov na server a funkcie na spúšťanie kódu zabaleného v nahraných .jar súboroch. Úsek kódu, ktorý slúži na spustenie .jar súboru je zobrazený na obrázku číslo 2.15.



Obrázok 2.13: Rozhranie na pripojenie a na nahrávanie súboru k implementácii



Obrázok 2.14: Rozhranie na zobrazovanie a pridávanie implementácií do konfigurácie

```
def run(*args)
  begin
    require jar_upload.jar.path
    full_name = package_name+"."+class_name
    import full_name
    classname = Object.const_get(class_name)
    return classname.java_send(method_name,args)
  rescue Exception => e
    return e.message
  end
end
```

Obrázok 2.15: Príklad úseku programu, ktorý spúšťa nahraný .jar súbor

Testovanie

V tomto používateľskom príbehu sa vykonával refactoring veľkej časti kódu v jazyku Java. Tento refactoring sa týkal aj testov, najmä kvôli zmenenému spôsobu prístupu k databáze. Museli sme teda prerobiť väčšinu testov doteraz implementovaných častí v jazyku Java a doplniť ďalšie.

V jazyku Ruby vzniklo rozhranie na vytváranie a spravovanie implementácií a konfigurácií. Všetky tieto rozhrania sme pokryli testami. Rovnako sme pokryli testami spúšťanie kódu, ktorý sme pripojili do aplikácie za behu programu pomocou implementácií. Na obrázku číslo 2.16 je príklad testu, ktorý overuje fungovanie pripájania uploadovaného súboru k implementácii.

```
it "should be able to attach already uploaded file" do
  system = System.first
  upload = jar_upload("#{Rails.root}/spec/test_data/test.jar")
  implementation = Factory(:implementation, :system => system)
  visit implementation_path(implementation)
  select upload.jar_file_name, :from => "implementation_jar_upload_id"
  click_button "Select"
  implementation = Implementation.find(implementation)
  implementation.jar_upload.should == upload
end
```

Obrázok 2.16: Príklad testu na otestovanie rozhrania na vytváranie implementácie

2.4. Šprint č. 4: „Delphinidae Delphis“

ID p. príbehu	Ako	Chcem	Aby bolo možné
14	Používateľ	Pravidelne prepočítavať metriky a podobnosti používateľov a dokumentov	Odporúčať podľa najnovších informácií a v reálnom čase
Z01	Používateľ	Mať možnosť poskladať kolaboratívny odporúčací systém z pluginov	Odporúčať na základe neho.
Z02	Používateľ	Mať možnosť poskladať odporúčací systém založený na obsahu z pluginov	Odporúčať na základe neho.
Z03	Používateľa	Určovať používateľov podľa cookies	Určiť ich jednoznačne.

Tabuľka 2.7: Používateľské príbehy šprintu č. 4

ID p. príbehu	Hl. vedúci	ID úlohy	Podúloha	Zodpovedný
14	Martin Detko	14.1	Vytvorenie dávkovača na spracovanie dát	Martin Detko
Z01	Michal Cádrik	Z01.1	Prerobenie kolaboratívneho odporúčania na spôsob pluginov	Michal Cádrik
Z02	Ľudovít Mydla	Z02.1	Prerobenie odporúčania založeného na obsahu na spôsob pluginov	Ľudovít Mydla
Z03	Jakub Ševcech	Z03.1	Identifikácia používateľa pomocou cookies a nie IP adresy.	Jakub Ševcech

Tabuľka 2.8: Detailný opis úloh

2.4.1. Vytvorenie dávkovača na spracovanie dát. p.p. 14

Ako používateľ chcem pravidelne prepočítavať hodnoty metrík a podobností užívateľov a dokumentov.

Analýza

Doteraz bolo nutné vždy pred odporúčaním prepočítať podobnosť. Ale to pri veľkom množstve dát trvá strašne dlho. To isté platí aj o okamžitom zobrazovaní metrík. Práve preto je nutné mať tieto informácie už v databáze uložené a vypočítané.

Návrh

Pri prepočítavaní podobnosti kvôli jednoduchosti bol interval prepočítavania nastavený na deň. Pri prepočítavaní metrík je nutné, aby si tento čas, od kedy do kedy sa má metrika prepočítať, mohol zvoliť sám používateľ, ale ako pravidelným intervalom ho môžeme obmedziť na deň, týždeň a mesiac. Presne takto často je potreba vytvoriť úlohu do zásobníka úloh, ktorá pre každú aktuálnu konfiguráciu spočíta, všetky pre ňu nastavené podobnosti a metriky.

Implementácia

Úloha bola implementovaná v ruby. Na jednej strane *cronjob* pravidelne generoval *rake tasky* a vkladal ich do fronty úloh, ktorá bola postupne spracovávaná. Na druhej strane tieto *rake tasky* boli dvoch typov. Spočítanie podobností a spočítanie metrík.

Pri počítaní podobností sa pre každú aktuálnu konfiguráciu, ktorá bola v databáze označená ako *actual*, spustili všetky implementácie podobností, ktoré mali v relačnej tabuľke *configuration_to_implementation_relations* záznam s aktuálnou konfiguráciou. Následne bol z danej relačnej tabuľky získaný atribút *parameters*, v ktorom bol uložený čas, kedy bolo naposledy robené prepočítavanie. S týmto parametrom je zavolaná funkcia z javy, ktorá prepočíta všetky podobnosti nových entít, pre ktoré podobnosť ešte nebola počítaná. Potom sa do atribútu *parameters* vloží aktuálny čas.

Pri počítaní metrík sa pre každú aktuálnu konfiguráciu, ktorá bola v databáze označená ako *actual*, spustili všetky implementácie metrík, ktoré mali v relačnej tabuľke *configuration_to_implementation_relations* záznam s aktuálnou konfiguráciou a v atribúte *parameters* mali poznačené, že sa majú vykonať s takou istou pravidelnosťou, ktorá je práve vykonávaná. Okrem toho sa z *parameters* získa aj čas posledného prepočítavania, čo je čas, od kedy sa ma počítať metrika. Parameter do kedy sa má počítať metrika je aktuálny čas systému. S týmito dvoma parametrami sa zavolá funkcia v Jave na prepočet metrík. Na konci sa minulý čas v atribúte *parameters* nahradí aktuálnym.

Testovanie

Testovanie plánovania a testovanie rake taskov prebiehalo zvlášť a nakoniec bolo otestované dokopy na menšej perióde plánovania.

2.4.2. Prerobenie kolaboratívneho odporúčania na spôsob pluginov, p.z. 01

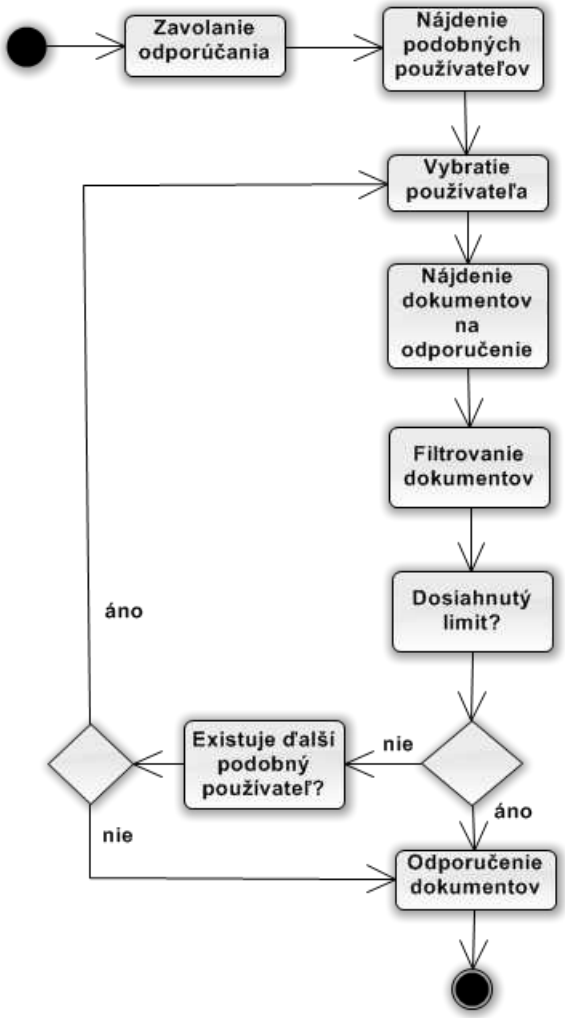
Ako používateľ chcem mať možnosť poskladať kolaboratívny odporúčací systém z pluginov, aby bolo možné na základe neho odporúčať.

Analýza

Ako sme uviedli v kapitole 2.1.2. Kolaboratívne odporúčanie, p. p. 02, tak je toto odporúčanie založené na podobnosti používateľov. Algoritmus ma za úlohu nájsť podobných používateľov daného systému a vrátiť zoznam dokumentov daného systému pre špecifikovaného používateľa. Pri tomto spôsobe odporúčania môžu vzniknúť určité problémy, ktoré v tomto štádiu nebudeme riešiť (Cold start problém, ...). Algoritmus by mal byť dostatočne svižný a poskytovať čo najviac prispôsobení, pretože bude musieť vykonávať počítania v reálnom čase vo veľkej množine dát. Preto by malo byť čo najviac možných filtrácií uskutočňovaných na databázovej vrstve.

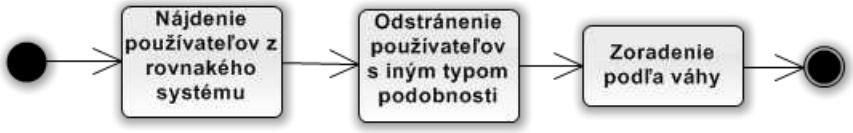
Návrh

Podľa návrhu systému musí naša trieda s odporúčaním dediť od abstraktnej triedy *sk.stuba.fiit.tp1112.recommender.core.interfaces.Recommender.java*. Postup algoritmu by mal byť podľa návrhu na obr. 2.17.

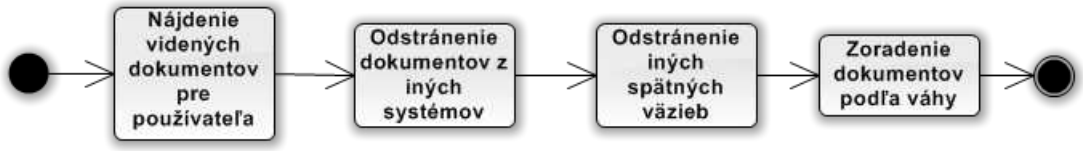


Obrázok 2.17: Zjednodušený diagram algoritmu kolaboratívneho odporúčania

Pre zrýchlenie behu celého algoritmu by mali byť časti algoritmu *Nájdenie podobných používateľov* a *Nájdenie dokumentov na odporučenie* realizované prevažne na databázovej vrstve.

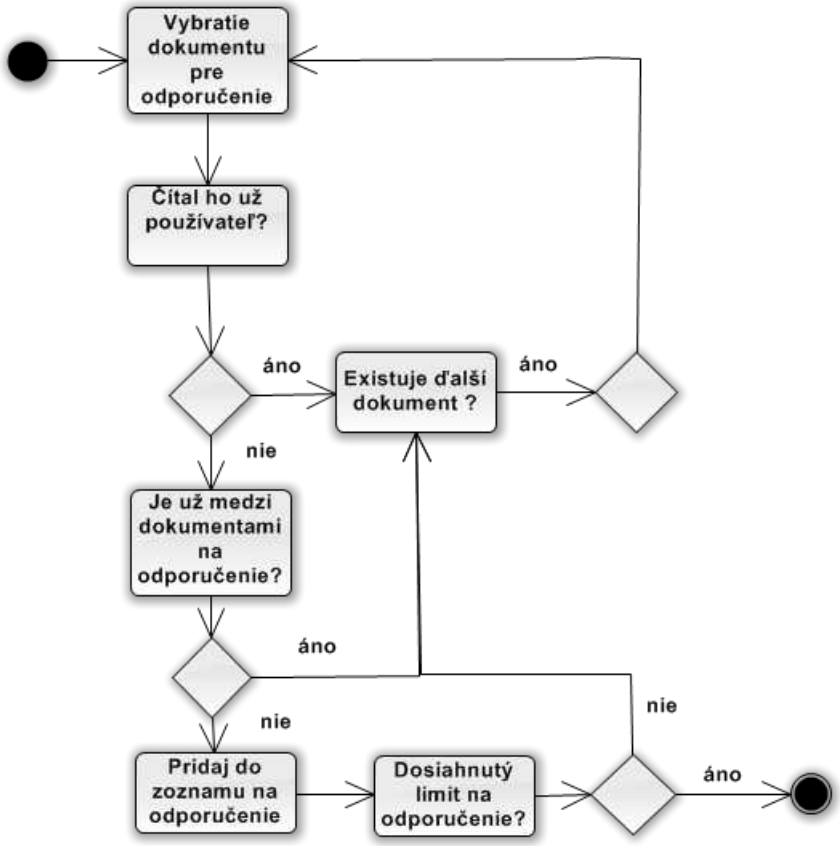


Obrázok 2.18: Postupné filtrovanie používateľov



Obrázok 2.19: Postupné filtrovanie dokumentov

Filtrovanie dokumentov znamená odstránenie všetkých dokumentov, ktoré už používateľ videl alebo už sú v zozname na odporučenie. Bližšie proces zobrazuje obr. 2.20.



Obrázok 2.20: Proces filtrovania dokumentov na množinu odporúčaných dokumentov

Implementácia

Navrhnutý algoritmus je implementovaný v triede CollaborativeRecommendation.java v balíku src\main\sk\stuba\fiit\tp1112\recommender\recommender. Predpis hlavnej metódy je zdedený z abstraktnej triedy Recommender z src\main\sk\stuba\fiit\tp1112\recommender\interfaces.

Algoritmus využíva metódy *getClosestNeighbours(), getReadDocsForUser()* a *getReadDocsForUserToRecommend()* na prístup do databázy, ktoré sú implementované v triede src\main\sk\stuba\fiit\tp1112\recommender\core\models\User.java. Každá prijíma ako parameter systémové id, id používateľa pre ktorého má niečo vyhľadať v databáze a typ zoradenia výsledku. Posledná obsahuje ešte naviac zoznam spätných väzieb, ktoré sa majú brať do úvahy. Keďže priamy prístup k metódam bol zakázaný, tak sa ku nim pristupuje pomocou triedy Gateway, ktorá sa stará o

doplnenie atribútov systémového id pre databázové funkcie. Metóda *getClosestNeighbours* vracia zoznam používateľov daného systému zoradených podľa zadaného typu pre jeden typ vypočítanej podobnosti pre nejakého používateľa. Metódy *getReadDocsForUser* a *getReadDocsForUserToRecommend* vracajú zoznam dokumentov s tým rozdielom, že *getReadDocsForUser* vráti všetky čítané dokumenty bez ohľadu na spôsob spätnej väzby.

Samotný odporúčací algoritmus spustíme pomocou metódy *getRecommendation(int userId, int resultLimit, int implementationId, String[] feedbackType, boolean orderDescDocs, boolean orderDescUsers)*, kde *userId* je id používateľa, pre ktorého ideme odporúčať, *resultLimit* je maximálny počet dokumentov, ktoré chceme odporučiť, *implementationId* je id implementácie podobnosti podľa ktorej máme vyberať podobných používateľov, *feedbackType* je pole typov spätných väzieb, ktoré máme akceptovať pri odporúčaní dokumentov. Ak je prázdne, tak akceptujeme všetky typy spätnej väzby. Posledné dva argumenty zodpovedajú typom zoradenia pre podobných používateľov a dokumenty, keďže nevieme povedať, ktorý spôsob podobnosti počíta akým spôsobom podobnos – či je podobnejší ten čo má vyššie číslo alebo nižšie. Metóda vracia pole reťazcov, alebo prázdne pole, ak sa nič nenájde.

Testovanie

Testovacie dáta boli vybrané vzhľadom na otestovanie všetkých možností, ktoré by sa mohli vyskytnúť pri odporúčaní. Otestovaná bola každá metóda, ktorá bola vytvorená pre daný kolaboratívny algoritmus. Testy sú implementované v balíčku src\test\sk\stuba\fiit\tp1112\recommender\recommender. Testy pre funkcie pristupujúce do databázy sa nachádzajú v triede RecommendationDbMethodsTest.java a testy pre celkový algoritmus odporúčania sú v CollaborativeRecommendationTest.java. Testy využívajú metódy *setupBeforeClass()* a *tearDownAfterClass()* na nastavenie sa na testovaciu databázu a vymazanie uložených dát v databáze. V metóde *setupBeforeClass()* sa ešte zavolá metóda *setTestingData()* z triedy testData.java, ktorá slúži na naplnenie databázy testovacími údajmi.

2.4.3. Identifikácia používateľa pomocou cookies a nie IP adresy, p.z. 03

V dátach na uploadovanie máme informácie o používateľoch, dokumentoch a interakcií používateľov s nimi. Je treba prerobiť skript na vytváranie uploadovacieho xml súboru tak, aby neboli používatelia identifikovaní IP adresou ale pomocou cookie.

Analýza

Nie je vhodné identifikovať používateľov pomocou IP adresy. Za jednou IP adresou sa môže nachádzať veľa počítačov a potom nieje identifikácia jednoznačná. Lepšie je použiť identifikátor ako cookie.

Okrem toho sa mierne zmenila štruktúra výsledných xml súborov, takže treba zmeniť aj šablónu na generovanie výsledných súborov.

Implementácia

V skripte na generovanie xml súboru som upravil súbory documents.xml.builder, users.xml.builder a user_to_document_relations.xml.builder tak, aby vyhovovali upravenej štruktúre xml súborov.

Testovanie

Testovanie prebehlo overením vygenerovaných súborov pomocou priložených xsd schém.

2.5. Šprint č. 5: „Eptesicus Fuscus“

ID p. príbehu	Ako	Chcem	Aby bolo možné
Z04	Používateľ	Prerobiť metriky podľa novej architektúry systému	Pridávať ďalšie metriky

Tabuľka 2.9: Používateľské príbehy šprintu č. 5

ID p. príbehu	Hl. vedúci	ID úlohy	Podúloha	Zodpovedný
Z04	Igor Slotík	Z04.1	Vytvorenie metriky systémom pluginov	Igor Slotík

Tabuľka 2.10: Detailný opis úloh

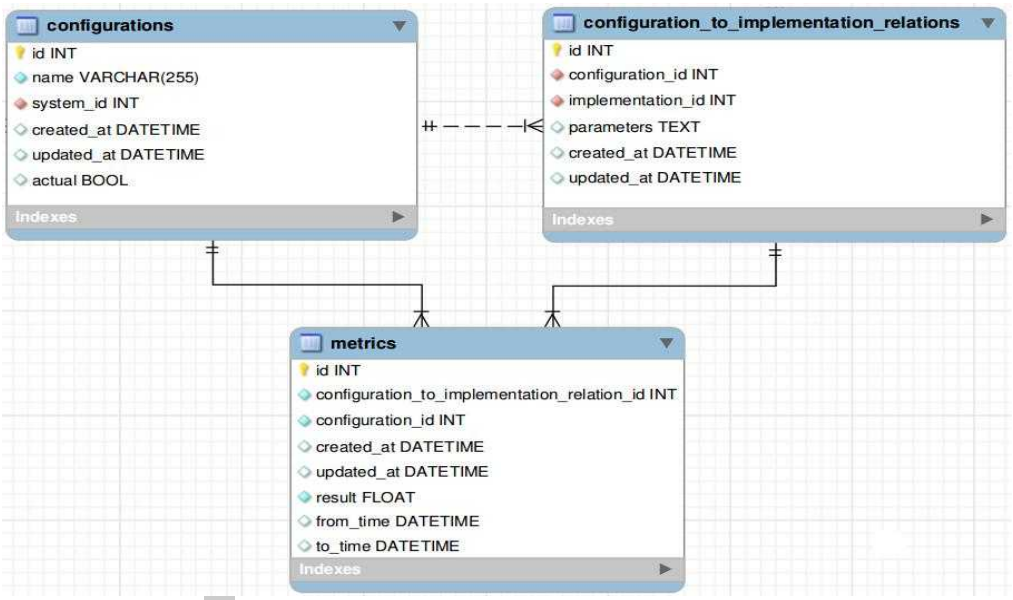
2.5.1. Vytvorenie metriky systémom pluginov, p.z. 04

Analýza

Metriky boli vytvorené za účelom vyhodnotenia odporúčaní pre používateľov. Keď sa odporúča určitý dokument, uloží sa poradie odporúčaného dokumentu. Používateľ má možnosť poslať ohodnotenie dokumentu. Porovnaním používateľovej spätnej väzby a uloženého odporúčaného dokumentu sa určí úspešnosť odporúčania. Problém je v zmene prístupe k databáze a možnosti jednoduchého používania metrík mimo projektu alebo inými vývojármi vrámci projektu. Vytvorenie metriky systémov pluginov rieši obe problémy.

Návrh

Na obr. 11 je časť dátového modelu, kde je tabuľka metrika a nevyhnutné tabuľky pre ukladanie metriky.



Obrázok 11: Dátový model tabuľky *metrics* a súvisiacich tabuliek

Tabuľka *metrics* má okrem svojho *id* ešte tri povinné parametre: dve cudzie kľúče, *configuration_id* a *configuration_to_implementation_relations* a *result,* ktorý výsledkom výpočtu konkrétnej metriky.

Implementácia

Pri návrhu metriky pomocou pluginov bolo nutné brať do úvahy zmenu prístupu k databáze. Bol vytvorený balík *src\main\sk\stuba\fiit\tp1112\recommender\metrics,* v ktorej sú dve triedy. Jedna z nich je abstraktná a obsahuje abstraktnú metódu. Abstraktná trieda dedí triedu *Register.* V klasickej triede je samotná implementácia metriky systémov pluginov. Klasická trieda dedí abstraktnú triedu za účelom prekonania metód triedy *Register.* Registráciou sa získa inštancia triedy *Gateway* a tým prístup k databáze a možnosť vytvorenia metriky.

Testovanie

Pre testovanie bol vytvorený balík *src\test\sk\stuba\fiit\tp1112\recommender\metrics*. Vygenerujú sa ideálne dáta na testovanie. Ideálne dáta sú také, že poradie odporúčaných dokumentov je totožné so spätnou väzbou používateľa. Výsledná metrika je v takom prípade nulová. Testovacia databáza sa vyčistí. Vygenerujú sa dáta s náhodným poradím odporúčania a aj dáta pre najhoršiu metriku. Pri najhoršej metrike je poradie odporúčania presne opačné s používateľovou spätnou väzbou. Testovanie je úspešné v tom prípade, ak je ideálna metrika nulová a náhodná metrika je menšia ako najhoršia metrika.

Príloha A - XML schéma používateľov pre upload dát

```
<?xml version="1.0" encoding="UTF-8"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema">
 <element name="FILE">
  <complexType>
   <sequence>
    <element name="USER" maxOccurs="unbounded" minOccurs="1">
     <complexType>
      <sequence>
       <element name="IDUSER">
        <simpleType>
         <restriction base="string">
          <maxLength value="255"></maxLength>
         </restriction>
        </simpleType>
       </element>
       <element name="STEREOTYPE" minOccurs="0" maxOccurs="unbounded">
        <complexType>
         <sequence>
          <element name="STEREOTYPENAME">
           <simpleType>
            <restriction base="string">
             <maxLength value="255"></maxLength>
            </restriction>
           </simpleType>
          </element>
          <element name="STEREOTYPEVALUE">
           <simpleType>
            <restriction base="string">
             <maxLength value="255"></maxLength>
            </restriction>
           </simpleType>
          </element>
         </sequence>
        </complexType>
       </element>
      </sequence>
     </complexType>
    </element>
   </sequence>
  </complexType>
 </element>
</schema>
```

Príloha B - XML schéma dokumentov pre upload dát

```
<?xml version="1.0" encoding="UTF-8"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema">

  <element name="FILE">
    <complexType>
      <sequence>
        <element name="DOCUMENT" maxOccurs="unbounded" minOccurs="1">
          <complexType>
            <sequence>
              <element name="IDENTIFICATION">
                <simpleType>
                  <restriction base="string">
                    <maxLength value="255"></maxLength>
                  </restriction>
                </simpleType>
              </element>

              <element name="TITLE">
                <simpleType>
                  <restriction base="string">
                    <maxLength value="255"></maxLength>
                  </restriction>
                </simpleType>
              </element>

              <element name="TYPE">
                <simpleType>
                  <restriction base="string">
                    <maxLength value="255"></maxLength>
                  </restriction>
                </simpleType>
              </element>

              <element name="CONTENT" minOccurs="0" maxOccurs="1" type="string"></element>

              <element name="METADATA" minOccurs="0" maxOccurs="1"
                              type="string"></element>

              <element name="KEYWORD" minOccurs="0" maxOccurs="unbounded">
                <complexType>
                  <sequence>
                    <element name="KEYWORDNAME">
                      <simpleType>
                        <restriction base="string">
                          <maxLength value="255"></maxLength>
                        </restriction>
```

```
                </simpleType>
               </element>

               <element name="KEYWORDWEIGHT" type="double"></element>

              </sequence>
             </complexType>
            </element>

           </sequence>
          </complexType>
         </element>
        </sequence>
       </complexType>
      </element>
</schema>
```

Príloha C - XML schéma relácií pre upload dát

```
<?xml version="1.0" encoding="UTF-8"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema">
<element name="FILE">
   <complexType>
    <sequence>
     <element name="RELATION" maxOccurs="unbounded" minOccurs="1">
      <complexType>
       <sequence>
        <element name="IDDOC">
         <simpleType>
          <restriction base="string">
           <maxLength value="255"></maxLength>
          </restriction>
         </simpleType>
        </element>

        <element name="IDUSER">
         <simpleType>
          <restriction base="string">
           <maxLength value="255"></maxLength>
          </restriction>
         </simpleType>
        </element>

        <element name="WEIGHT" type="double"></element>

        <element name="TYPE">
         <simpleType>
          <restriction base="string">
           <maxLength value="255"></maxLength>
          </restriction>
         </simpleType>
        </element>

       </sequence>
      </complexType>
     </element>
    </sequence>
   </complexType>
  </element>

</schema>
```

Príloha D- Prototyp

Architektúra systému

Prototyp je navrhnutý a implementovaný pomocou architektúry MVC (Model, View, Controller). Ako framework pre podporu implementácie aplikácie v MVC frameworku je použitý framework Ruby on Rails.

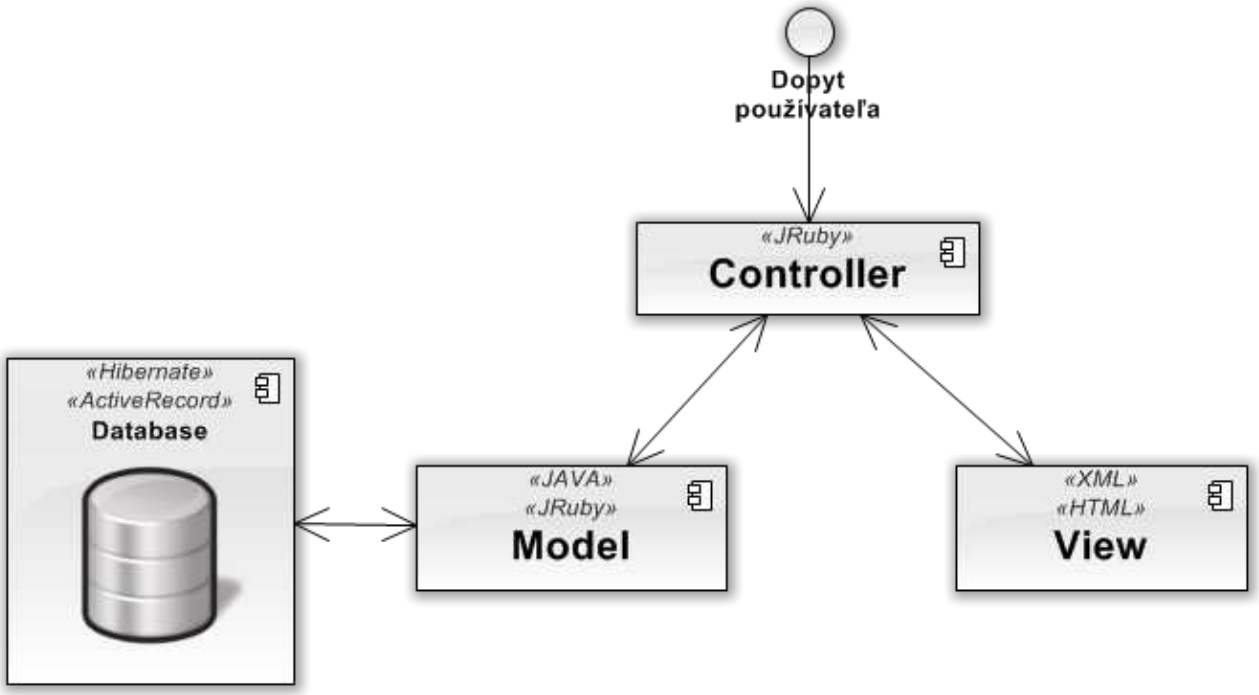
View reprezentuje vizualizáciu aplikácie pre používateľa. Táto je rozdelená na dve časti:

- Grafické prostredie v podobe webovej stránky
- API pre REST webovú službu, ktoré vracia XML súbory ako výsledok dopytu

Komponent *Controller* prepája implementovanú logiku v modeloch s požiadavkami používateľa a s vizualizáciou výsledkov. Tento komponent je implementovaný v jazyku JRuby.

Komponent model v sebe zahŕňa hlavnú implementačnú logiku webovej služby, algoritmy na odporúčanie a všetky pomocné funkcie, ako napríklad výpočet podobnosti vektorov a podobne.

Ako databázu sme použili PostgreSQL s ktorou komunikujú jednotlivé časti komponentu *Model* prostredníctvom objektovo relačných mapovačov. Pre jazyk Java je použitý mapovač Hibernate a pre jazyk JRuby je to ActiveRecord.



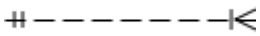
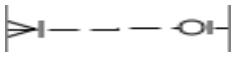
Obrázok D.1.Architektúra implementovaného prototypu

Dátový model

Kapitola obsahuje zdokumentovaný dátový model, v ktorom sú opísané najdôležitejšie atribúty a vzťahy.

Opis použitého značenia

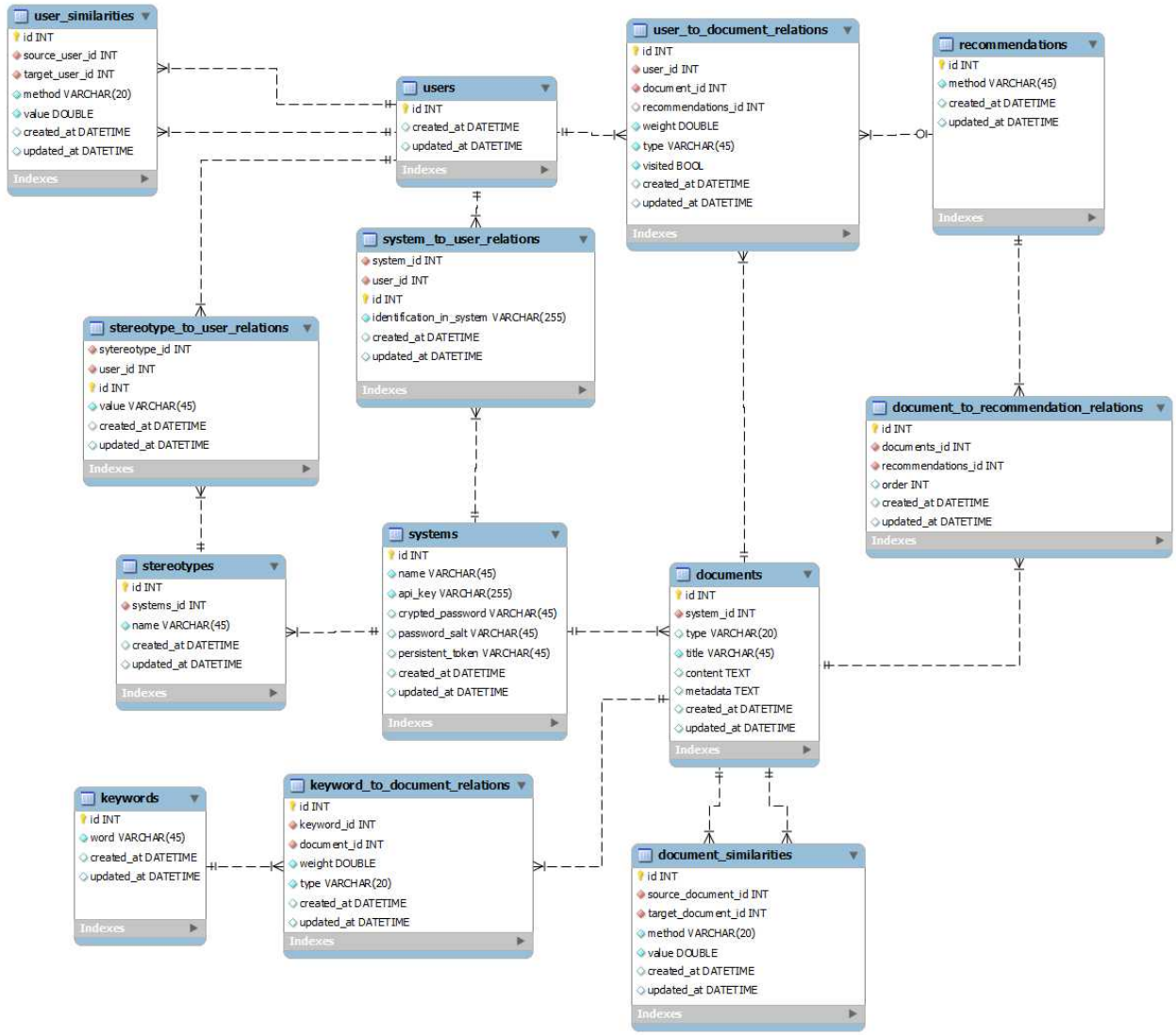
Tabuľka D.1 opisuje význam použitých značiek v diagramoch.

	Dátová entita
	Vzťah 1 ku n medzi dátovými entitami
	Vzťah 0 až 1 ku n medzi dátovými entitami
	Primárny kľúč entity
	Povinný cudzí kľúč entity
	Nepovinný cudzí kľúč
	Povinný atribút
	Nepovinný atribút

Tabuľka D.1: Význam použitých značiek v dátovom modely

Dátový model

Na obrázku D.2 sa nachádza dátový model. Opisuje obsah tabuliek v databáze a vzťahov medzi tabuľkami.



Obrázok D.2: Dátový model projektu

Opis základných použitých entít

users – Táto entita predstavuje človeka, ktorému chceme odporúčať.

documents – Dokument , ktorý chceme ľuďom odporúčať. Type hovorí o tom, akého typu je článok. V databáze bude uložený ako maximálne 20 znakový reťazec. Používať by ho mal odporúčací algoritmus a možné hodnoty definuje tvorca odporúčania. *Title* je názov článku, môže mať maximálne 45 znakov a je to povinný atribút. *Content* je atribút obsahujúci obsah článku. Môže byť ľubovoľne dlhý. *Metadata* sú dáta, ktoré hovoria o obsahu v dokumente. Jedná sa tiež nepovinný text ľubovoľnej dĺžky. Neskôr z neho môžu byť spracované kľúčové slová.

systems – Ak osoba chce vo svojom systéme používať našu službu musí sa vytvoriť nový záznam tohto systému u nás. Obsahuje atribúty ako *name*, čo by mal byť unikátny text maximálnej dĺžky 45 znakov, slúži tiež na prihlásenie sa do systému. Atribút *api_key* je reťazec dĺžky 255 znakov a slúži na sprístupnenie funkcií cez API. Ďalšie dva atribúty *crypted_password* a *password_salt* , sú dĺžky 45 znakov a slúžia na overenie identity pri prihlasovaní sa do systému. Atribút *persistent_token* má dĺžku 45 znakov a hovorí o tom, či je osoba práve prihlásená.

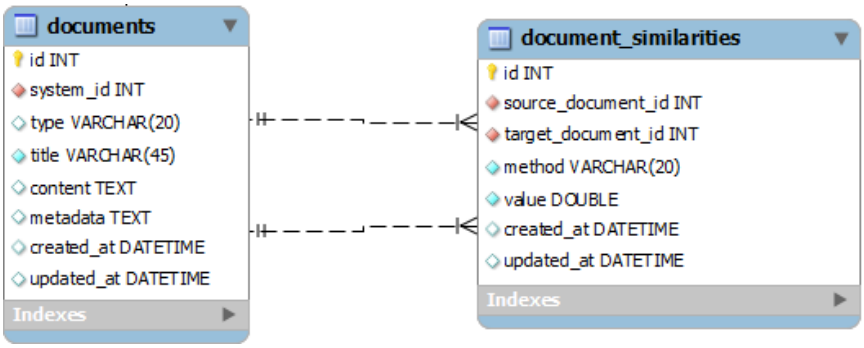
keywords – Kľúčové slovíčka dokumentov, podľa ktorého budeme zgrupovať dokumenty. Obsahuje jedine atribút *word*, čo je názov jedného kľúčového slova.

stereotypes – Opisuje odbornosť používateľa v danej oblasti v danom systéme. Obsahuje povinný atribút *name* ako názov oblasti v systéme. Reťazec má maximálnu dĺžku 45 znakov.

recommendations – Obsahuje všetky odporúčania, na základe ktorých boli používateľom odporúčané dokumenty. Obsahuje atribút *method*, ktorý hovorí o tom, akou metódou bolo používateľovi vytvorené odporúčanie.

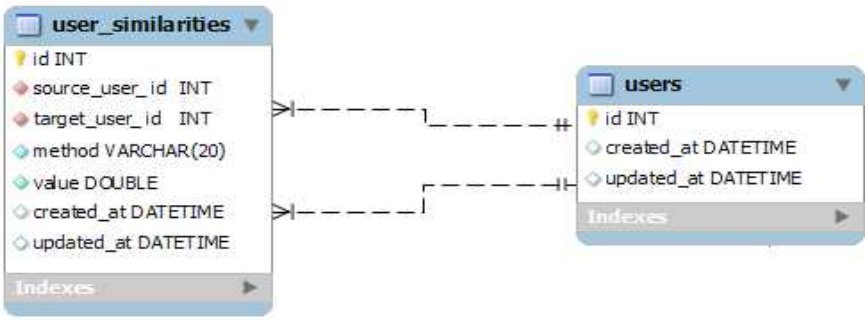
Opis vzťahov medzi entitami a relačných entít

document_similarities – Na obr. D.3 sa nachádza entita, ktorá hovorí o podobnosti dvoch dokumentov. Práve preto 2 atribúty sú cudzie kľúče dvoch spomínaných dokumentov. Ďalšími dvoma atribútmi sú *method* a *value*. *Method* je reťazec maximálne 20 znakov, ktorý hovorí o tom, akým spôsobom bola získaná podobnosť dokumentov. Túto hodnotu stanový človek, ktorý tvorí porovnávací algoritmus. *Value* je reálne číslo s presnosťou double. Ide o hodnotu, ktorá udáva ako veľmi sa dokumenty podobajú.



Obrázok D.3: Vzťah medzi documents a document_similarities

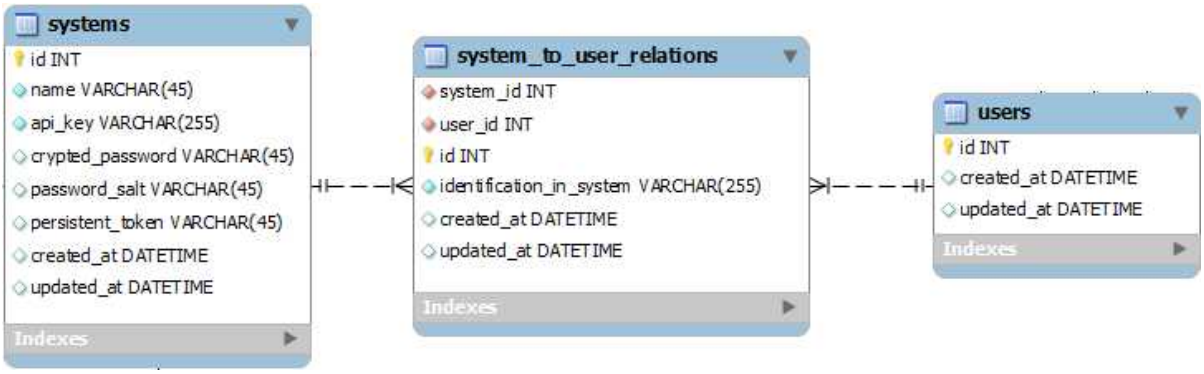
users_similarities – Na obr. D.4 sa nachádza entita, ktorá hovorí o podobnosti dvoch požívateľov. Práve preto 2 atribúty sú cudzie kľúče dvoch spomínaných používateľov. Ďalšími dvoma atribútmi sú *method* a *value*. *Method* je reťazec maximálne 20 znakov, ktorý hovorí o tom, akým spôsobom bola získaná podobnosť používateľov. Túto hodnotu stanový človek, ktorý tvorí porovnávací algoritmus. *Value* je reálne číslo s presnosťou double. Ide o hodnotu, ktorá udáva ako veľmi sa používatelia podobajú v ich správaní.



Obrázok D.4: Vzťah medzi users a users_similarities

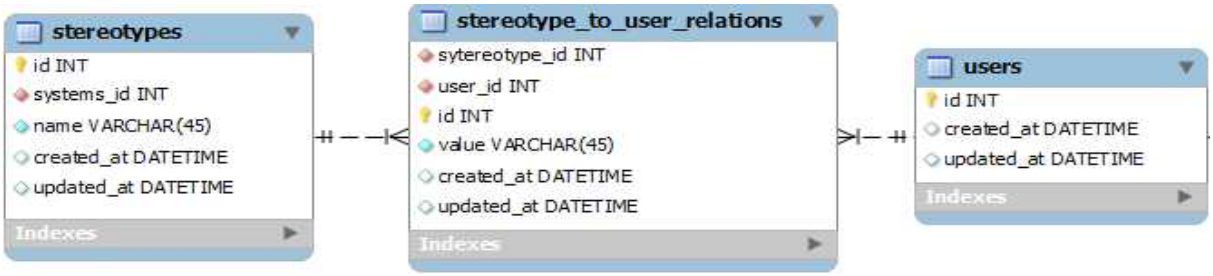
system_to_user_relations – Keďže v jednom systéme sa môže nachádzať viac ľudí a jeden človek

môže používať viac systémov, bola použitá relačná entita. Tento vzťah je zachytený na obr. D.5. Okrem toho táto relačná entita obsahuje atribút, *identification_in_system*, ktorý má dĺžku maximálne 255 znakov, a hovorí o tom, ako bol požívateľ identifikovaný v predchádzajúcom systéme.



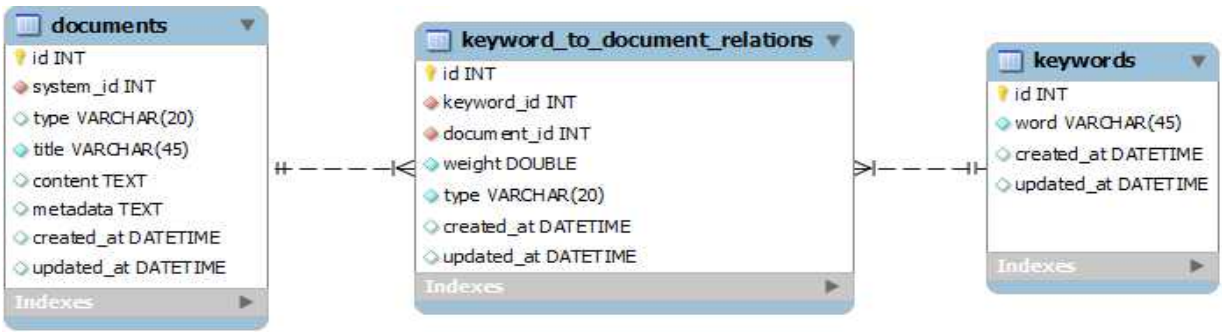
Obrázok D.5: Relačná entita medzi users a systems

stereotype_to_user_relations – Ako je vidieť na obr. D.6 táto relačná entita spája dve entity *users a stereotypes*. Opäť je to z toho dôvodu, že daný stereotyp môže mať viacero ľudí a jeden človek môže mať viac stereotypov. Pomocou reťazca *value*, ktorý má maximálne 45 znakov, určuje akú odbornosť ma používateľ v danom systéme.



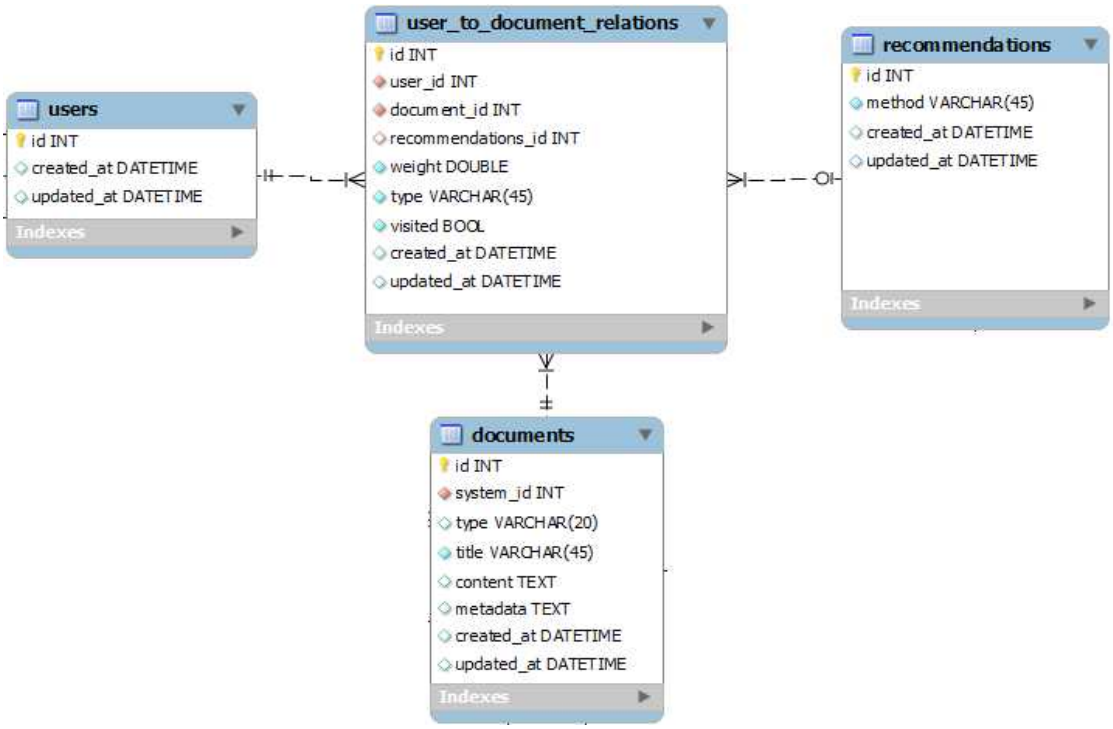
Obrázok D.6: Relačná entita medzi stereotypes a users

keyword_to_document_relations – Obrázok D.7 zachytáva vzťah medzi entitami *keywords* a *documents*. Táto entita obsahuje atribút *weight* ako celo-číselnú hodnotu hovoriacu o dôležitosti kľúčového slova v danom dokumente. Atribút *type* je reťazec maximálnej dĺžky 20 znakov a zachytáva typ tejto väzby.



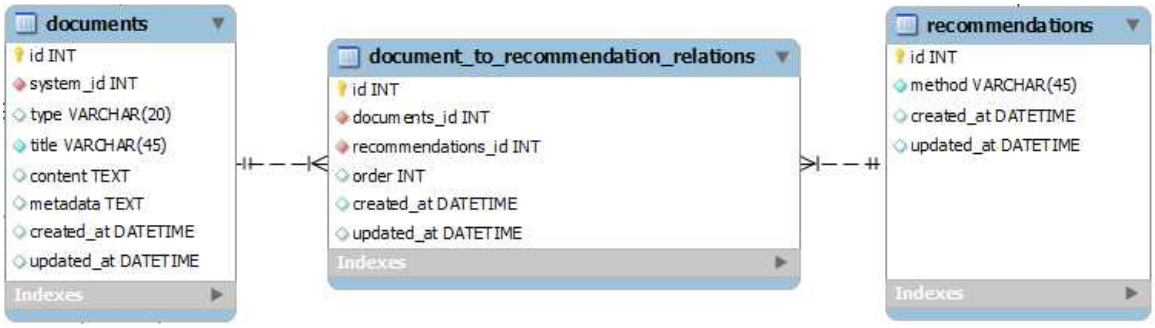
Obrázok D.7: Relačná entita medzi documents a keywords

user_to_document_relations – Relačná entita na obr. D.8 zachytáva práve vzťah hodnotenia dokumentov používateľmi. Okrem týchto dvoch cudzích kľúčov sa tam nachádza aj nepovinný cudzí kľúč entity *recommendations*. Tento vzťah zachytáva, pri akom odporúčaní používateľ dokument navštívil. Dokument však používateľ môže navštíviť nezávisle od odporúčania. Atribút *value* je reálne číslo s presnosťou double, ktorý hovorí o samotnej výške hodnotenia daného dokumentu používateľom. Ďalej je tu atribút *type*, ktorý je reťazec maximálnej dĺžky 45 znakov. Táto vlastnosť hovorí o tom, kedy, ako alebo prečo používateľ hodnotil dokument. Ďalší atribút *visited* je typu boolean a hovorí o tom, či už používateľ stránku navštívil alebo nie.



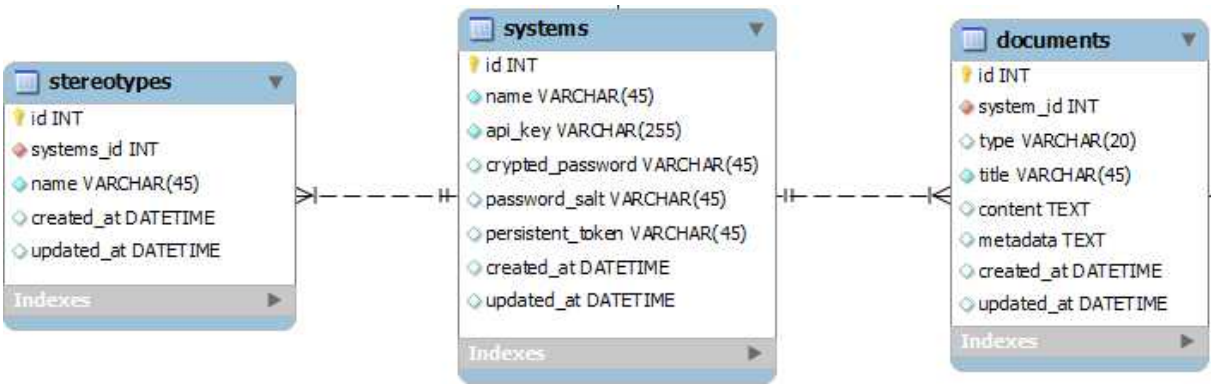
Obrázok D.8: Relačná entita medzi documents a users

document_to_recommendation_relations – Táto relačná entita zachytáva, že v ktorom odporúčaní bol odporučený aký dokument. Okrem toho, entita obsahuje atribút *order*, v ktorom je uložené miesto, na akom bol dokument odporučený v danom odporúčaní. Atribút nemusí byť zadaný, ak nás nezaujíma poradie. Tejto relácií zodpovedá obrázok D.9.



Obrázok D.9: Relačná entita medzi documents a users

Vzťah medzi *documents* a *systems* a vzťah medzi *stereotypes* a *systems* – Tieto dva vzťahy hovoria o tom, že dokumenty a stereotypy patria konkrétnemu systému. Táto skutočnosť je zachytená na obrázku D.10.



Obrázok D.10: Vzťah medzi dokumentmi a systémom a vzťah medzi systémom a stereotypmi

Okrem toho boli v každej entite použité atribúty *created_at* a *updated_at*. Prvý atribút hovorí o tom, kedy bol prvok záznam v tabuľke vytvorený a druhý, kedy bol zmenený. Obidva atribúty obsahujú informácie dátumu a času.

Konfigurácia serveru

Aplikácia je napísaná v jazyku JRuby, a v rámci nej sa volajú ostatné komponenty napísané v jazyku Java. Výber servera, kde by bola nasadená implementovaná výrazne ovplyvnili práve použité jazyky implementácie.

Počas nasadzovania aplikácie sme otestovali viacero rôznych serverov.

Mongrel

Veľmi jednoduchý server, z veľkej časti napísaný v jazyku Ruby. Je možné spustiť celý cluster procesov, ktoré budú spracovávať požiadavky.

Apache Tomcat v kombinácií s nástrojom Warbler

Warbler je nástroj na zabalenie Jruby aplikácie do jedného ‚war súboru. Tento súbor je následne jednoduché nasadiť na pripravený Tomcat server. Tomcat je veľký server s veľkým množstvom funkcií. Pre naše potreby boli však tieto len málo významné.

Trinidad

Podobne ako Mongrel je toto veľmi malý a kompaktný server. Inštalácia prebieha pomocou inštalovania gem knižnite.

Webrick

Je server, ktorý je nastavený ako pôvodný server pri vytvorení rails projektu. Tento server je však primárne určený pre účely vývoja aplikácie a jej testovanie, nie na produkcie, Tento server totiž napríklad dokáže pracovať len v jednom vlákne, čo pri produkčnom servery nepostačuje.

Glassfish

Nakoniec sme sa rozhodli pre použitie servera Glassfish. Je to pomerne rozšírený server, takže je

pomerne jednoduché nájsť pomoc pri riešení problémov na rôznych sprievodných fórach. Tento server je možné spustiť ako samostatný proces pre každú aplikáciu. Keďže sme už predtým používali server Apache, mohli sme nastaviť proxy a jednoducho preposielať dopyty na túto samostatne bežiacu aplikáciu.

Príloha E - Používateľská príručka

Aplikácia má dva druhy rozhrania:

- Grafické rozhranie prostredníctvom webovej aplikácie a
- REST Api.

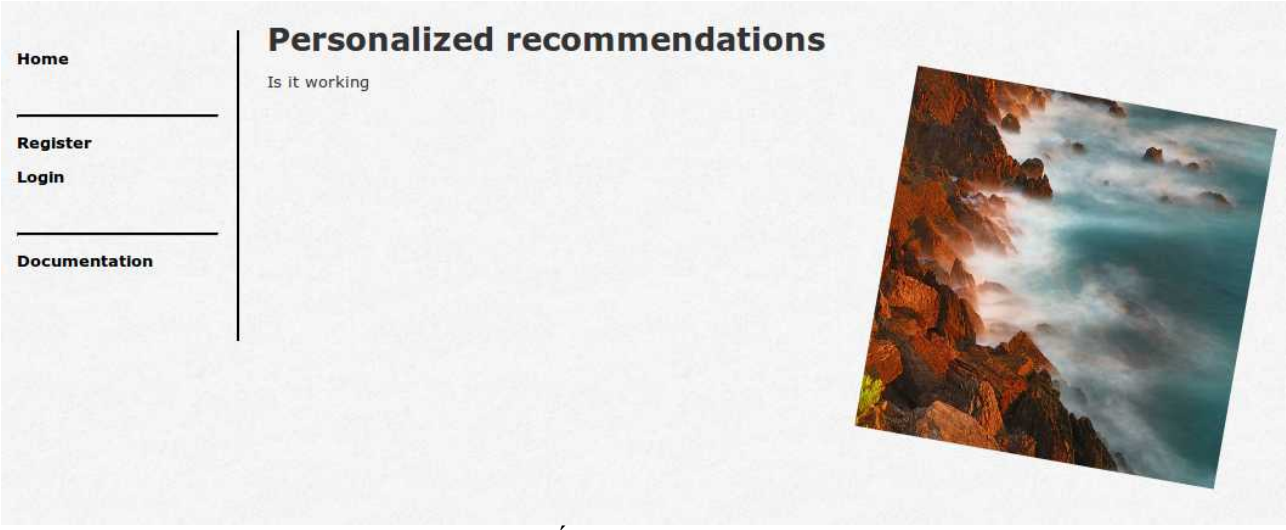
Grafické prostredie slúži na interaktívne manažovanie systému využívajúceho službu na odporúčanie. Api slúži na volanie funkcií služby automaticky priamo z konzumenta odporúčania.

Úvodná obrazovka

Aplikácia je dostupná na adrese:

http://vm28.ucebne.fiit.stuba.sk/app

Po zobrazení tejto stránky sa nachádzame ako neprihlásený systém na úvodnej obrazovke (Obrázok E.1).

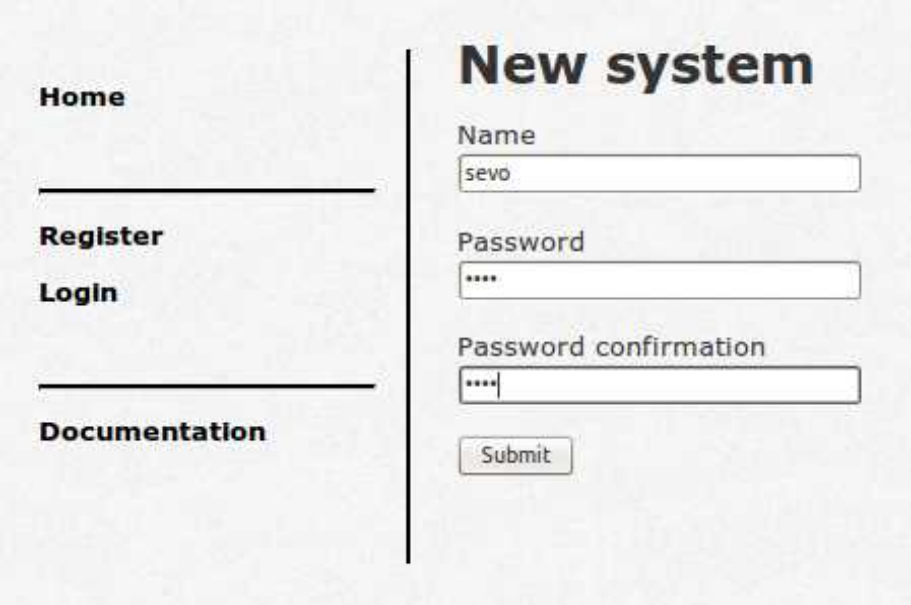


Obrázok E.1: Úvodná obrazovka aplikácie

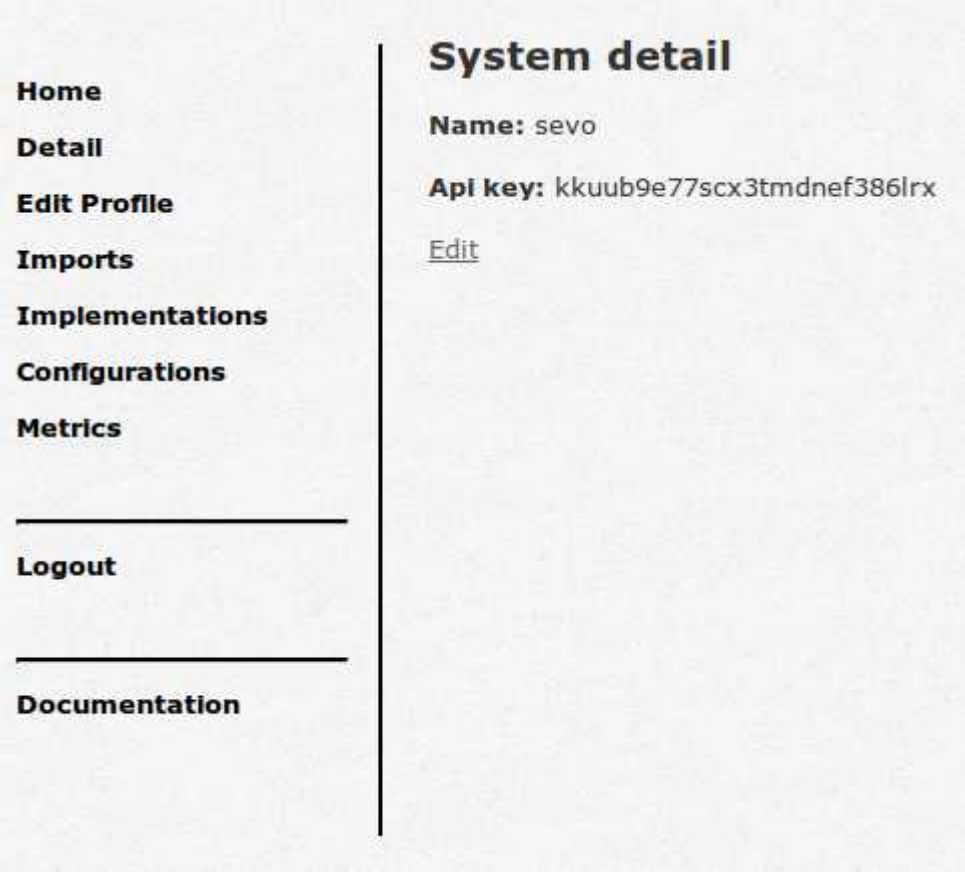
Na tejto obrazovke sú v ľavom menu dostupné možnosti na prihlásenie, registráciu a na zobrazenie dokumentácie k používaniu API funkcií.

Registrácia systému

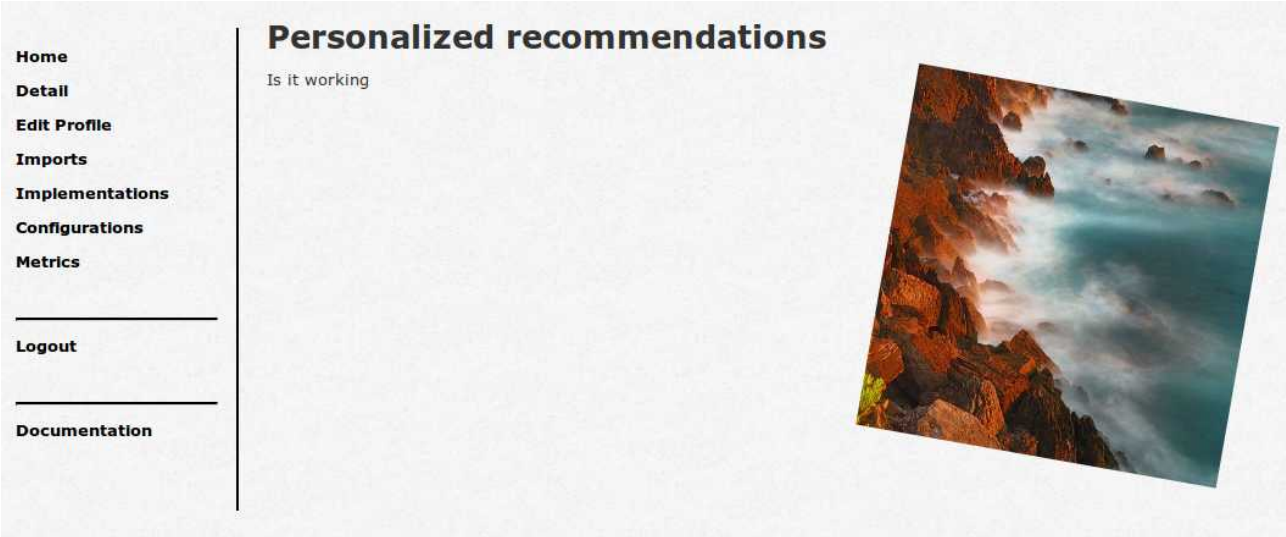
Pred začatím práce s aplikáciou sa systém, ktorý chce získavať odporúčanie pomocou tejto aplikácie, registrovať. Po kliknutí na odkaz *„Register“* na úvodnej obrazovke sa zobrazí registračný formulár (Obrázok E.2). Po vyplnení potrebných údajov a odoslaní formulára sa zobrazí detail nového systému aj s vygenerovaným apikľúčom (Obrázok E.3). Po kliknutí na odkaz *„Home“* sa opäť nachádzame na domovskej stránke, ale tentokrát už ako registrovaný systém máme dostupných oveľa viac funkcií (Obrázok E.4).



Obrázok E.2: Registračný formulár pre zaregistrovanie nového systému



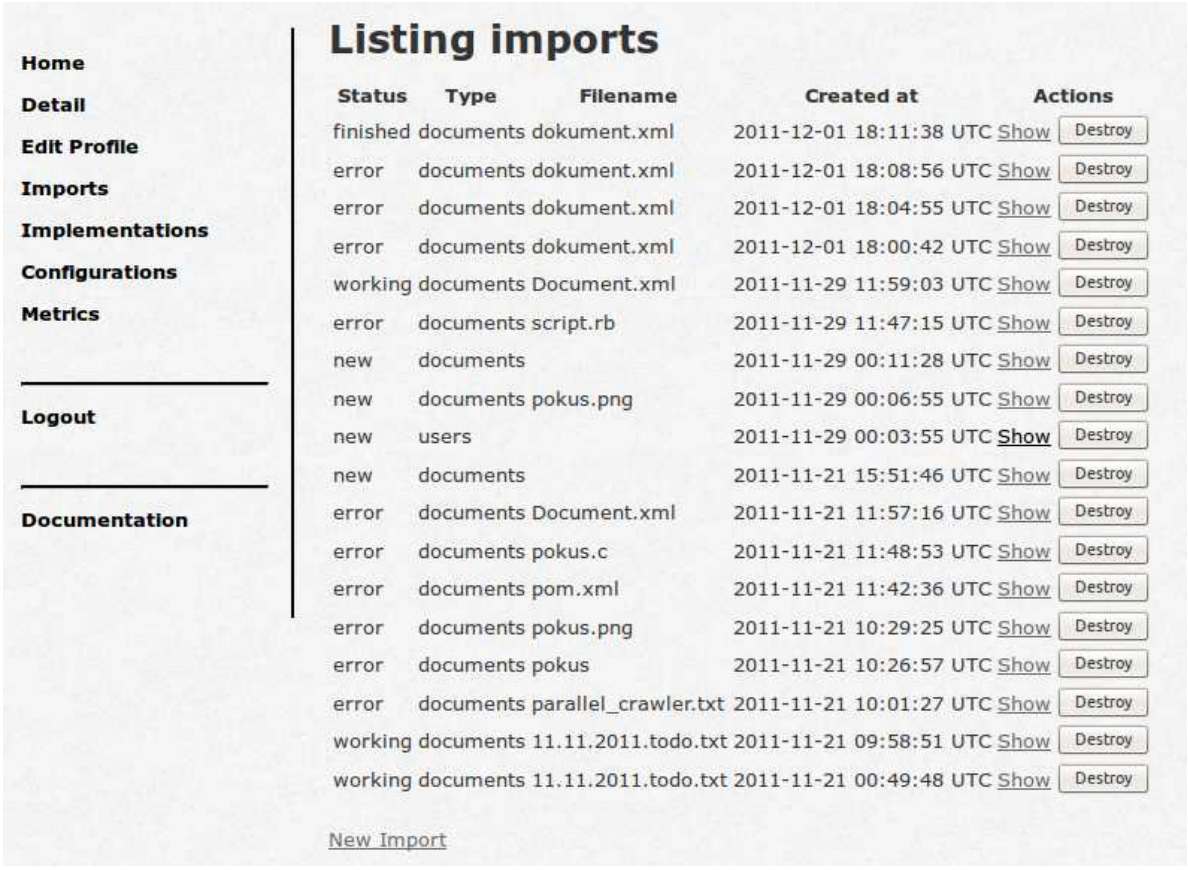
Obrázok E.3: Detail systému spolu s vygenerovaným apikľúčom



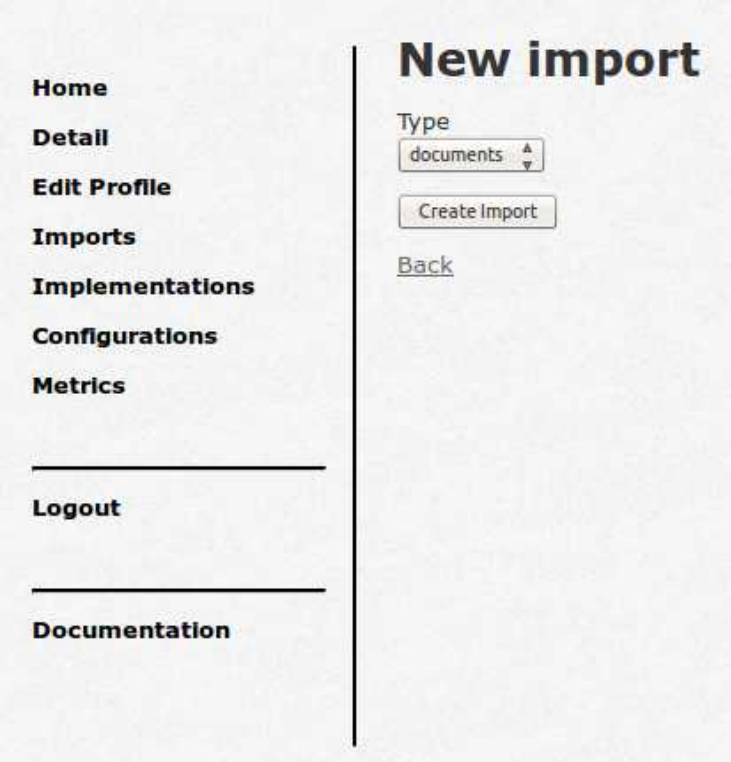
Obrázok E.4: Úvodná stránka registrovaného systému

Importovanie údajov

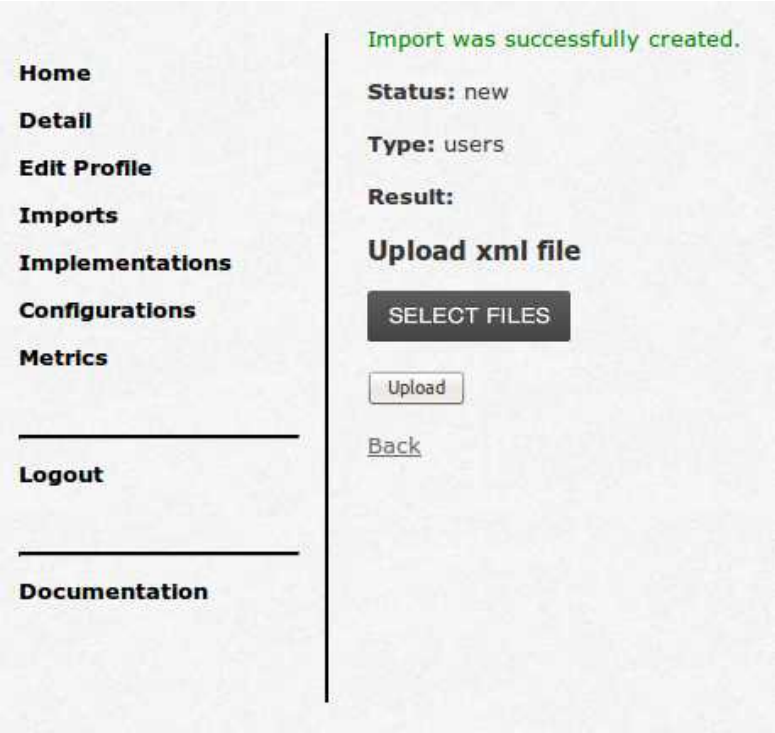
Ak sme prihlásený ako systém, tak sa v ľavom menu nachádza odkaz *„Imports“*. Po kliknutí na tento odkaz sa zobrazia všetky doteraz vykonané importy xml súborov (Obrázok E.5). Po kliknutí na odkaz *„New Import“* sa zobrazí okno na špecifikovanie nového importu (Obrázok E.6). V tomto okne vyberiem typ importu, ktorý chceme vytvoriť a klikneme na tlačidlo *„Create Import“*. Nachádzame sa na okne zobrazujúcom detail importu (Obrázok E.7). Na tejto obrazovke môžeme uploadovať súbor, ktorý sa bude importovať. Po spustení uploadovania máme možnosť sledovať postup pomocou progressbaru. Po nahraní súboru a nastavení sa na detail importu máme možnosť spustiť import kliknutím na odkaz *„Validate and run import“*. Import sa následne zaradí do zoznamu úloh na spracovanie a bude čakať, kým sa na neho dostane rad. Potom sa spustí a začne sa importovať. Výsledok importu sa opäť zobrazí na obrazovke detailu importu (Obrázok E.9). V prípade, že pri importe nastala chyba. Bude táto chyba opäť viditeľná na stránke detailu importu.



Obrázok E.5: Zoznam všetkých vykonaných importov

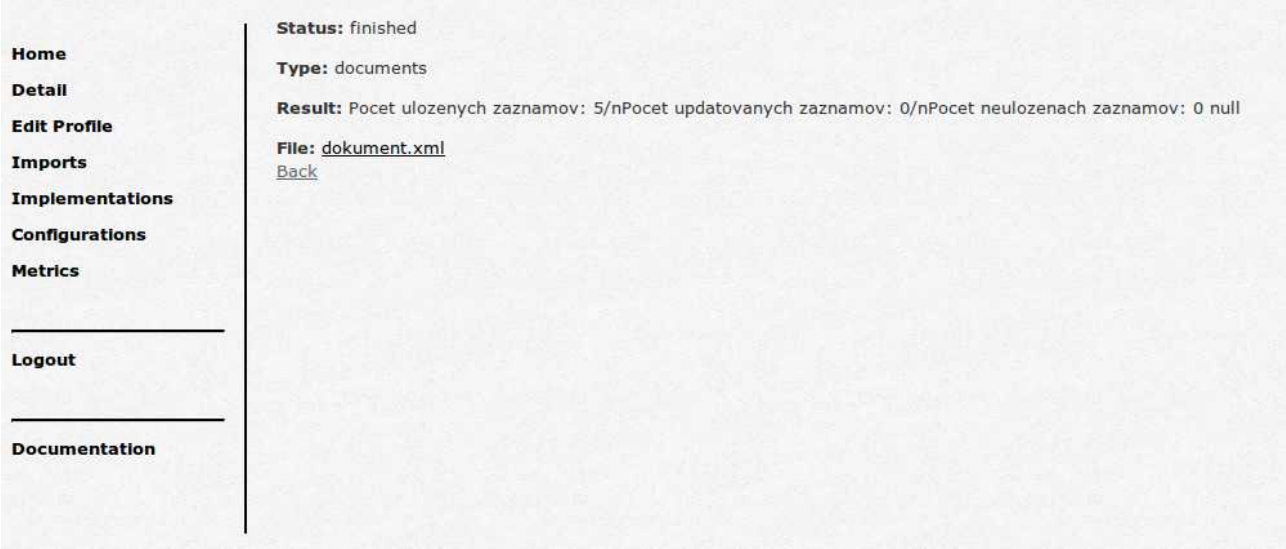


Obrázok E.6: Okno na špecifikovanie nového importu



Obrázok E.7: Detail novovytvoreného importu spolu s možnosťou nahrať súbor

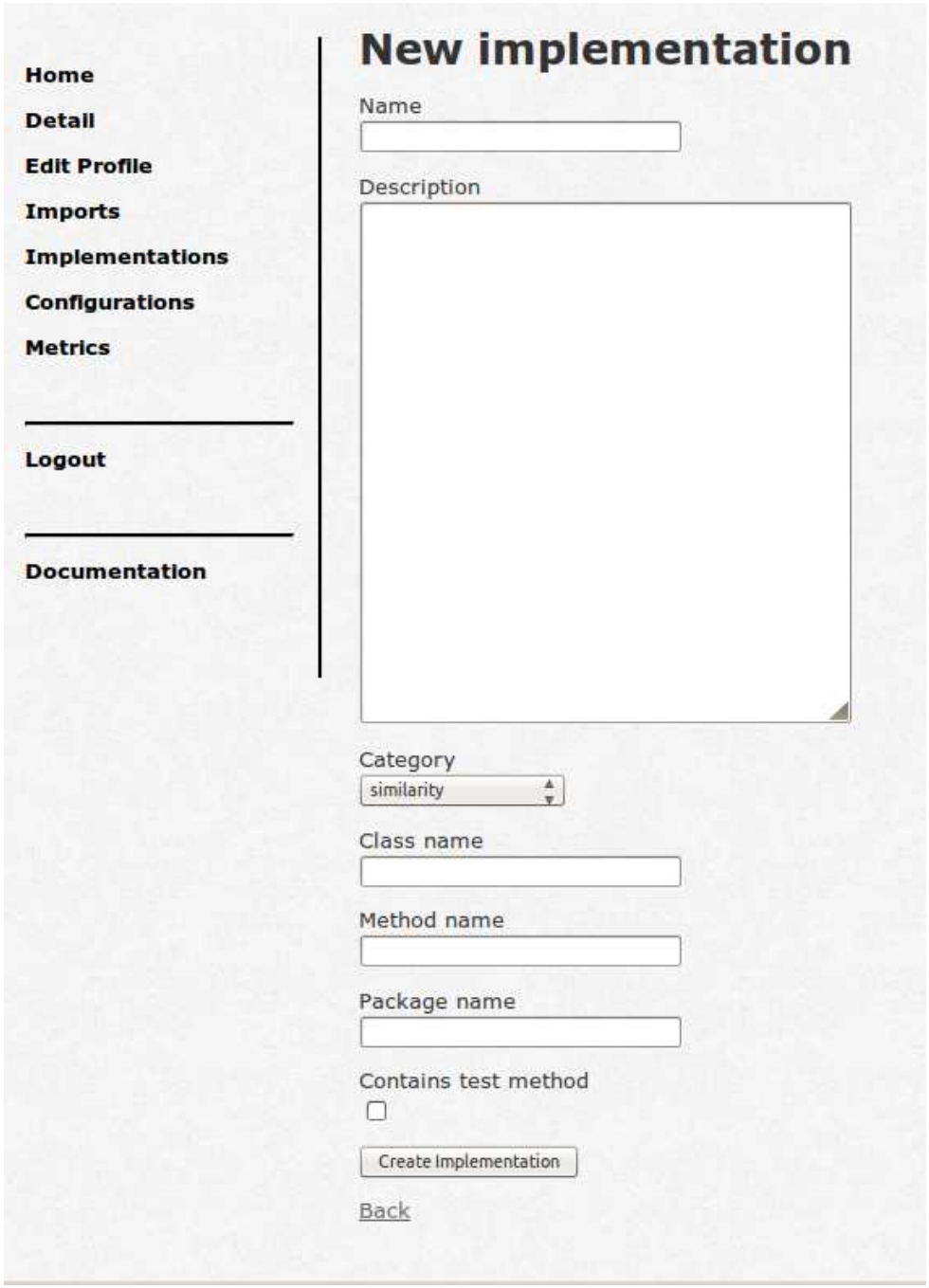
Obrázok E.8: Po nahraní súboru do importu máme možnosť spustiť import



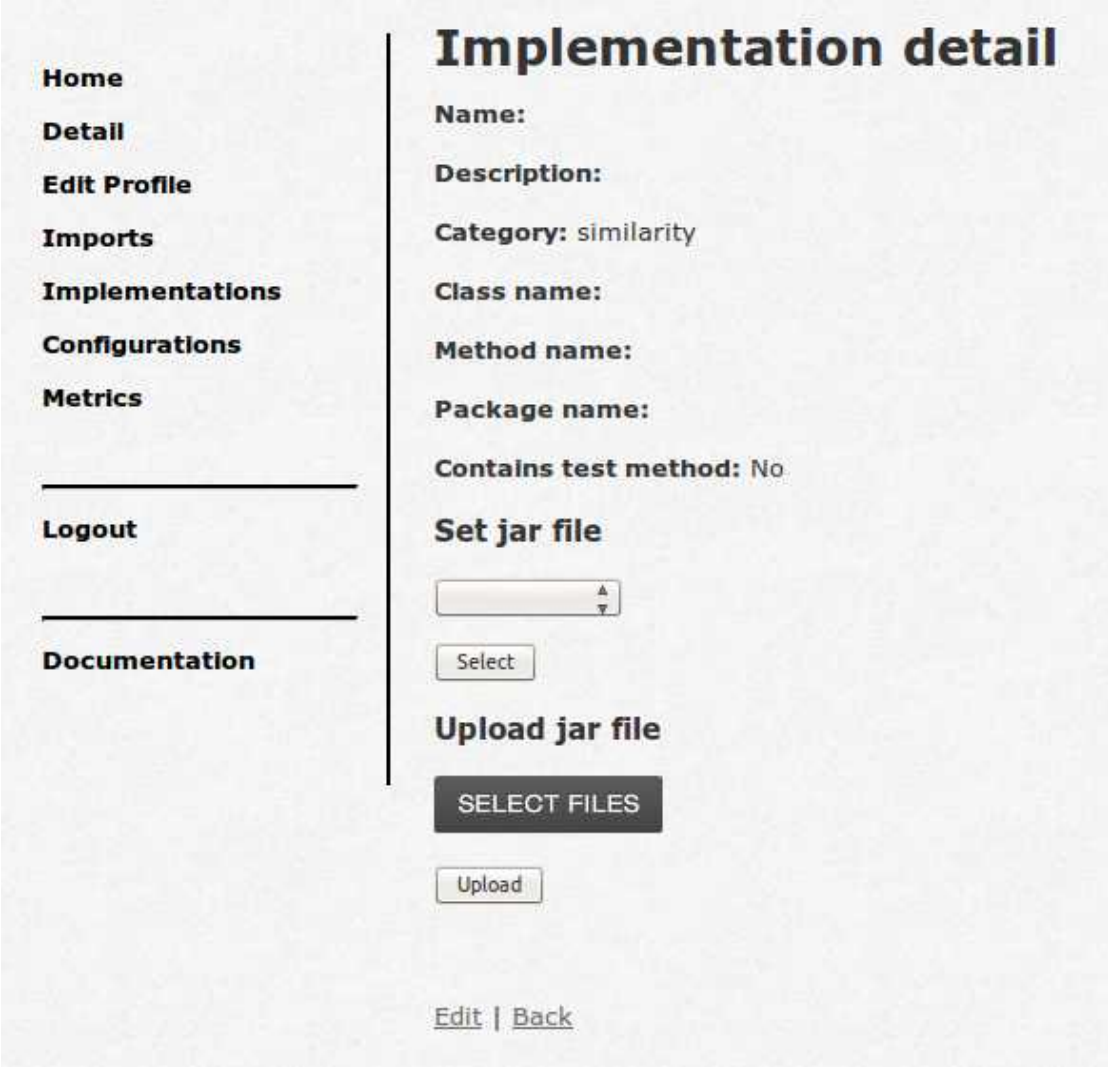
Obrázok E.9: Zobrazenie výsledku importu po úspešnom prebehnutí importovania

Správa implementácií

Ak sme prihlásený ako systém, tak sa v ľavom menu nachádza odkaz *„Implementations“*. Po kliknutí na tento odkaz sa zobrazia všetky doteraz vytvorené implementácie. Po kliknutí na odkaz *„New Implementation“* sa zobrazí obrazovka na nastavenie detailov novej implementácie (Obrázok E.10). Na tejto obrazovke je potrebné vyplniť najmä meno, kategóriu, meno triedy, názov balíčka a meno metódy, ktorá sa má spúšťať pri používaní danej implementácie. Kategória implementácie môže byť napríklad metrika, podobnosť, odporúčač alebo clustrovací algoritmus. Po kliknutí na tlačidlo *„Create Implementation“* sa zobrazí obrazovka (Obrázok E.11) s detailami novovytvorenej implementácie a s možnosťou pripojiť existujúci .jar súbor alebo nahrať nový.



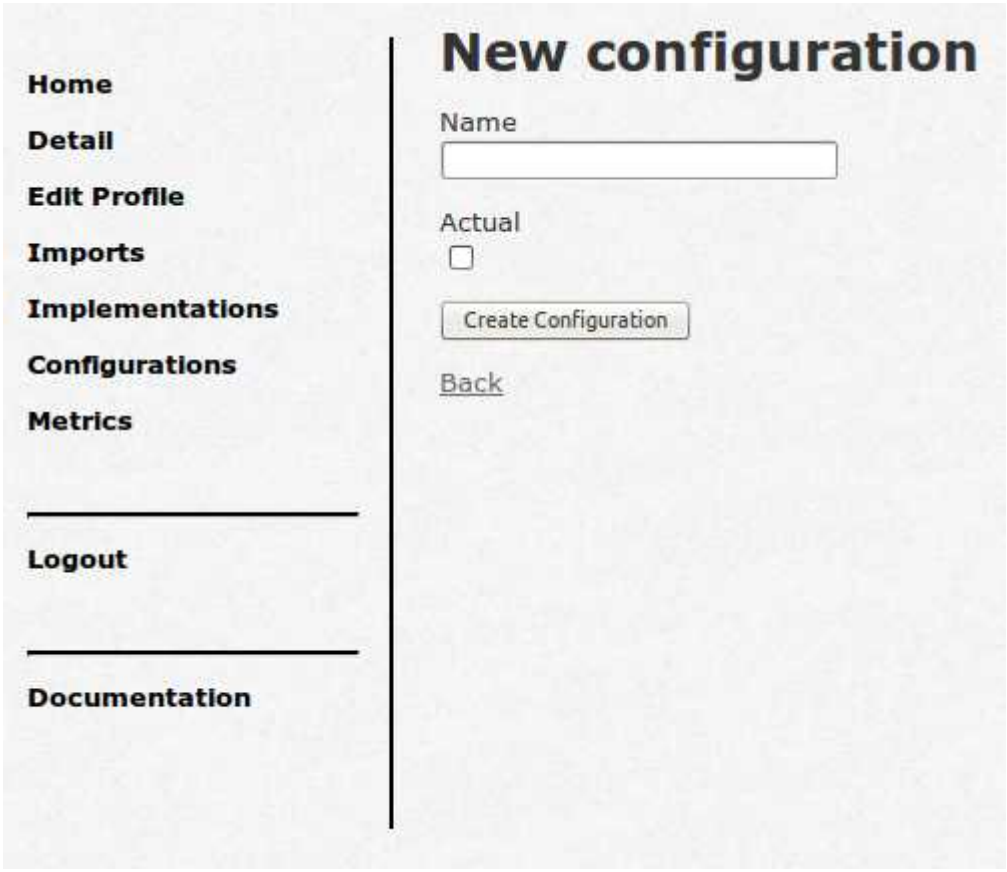
Obrázok E.10: Okno na nastavenie detailov novej implementácie



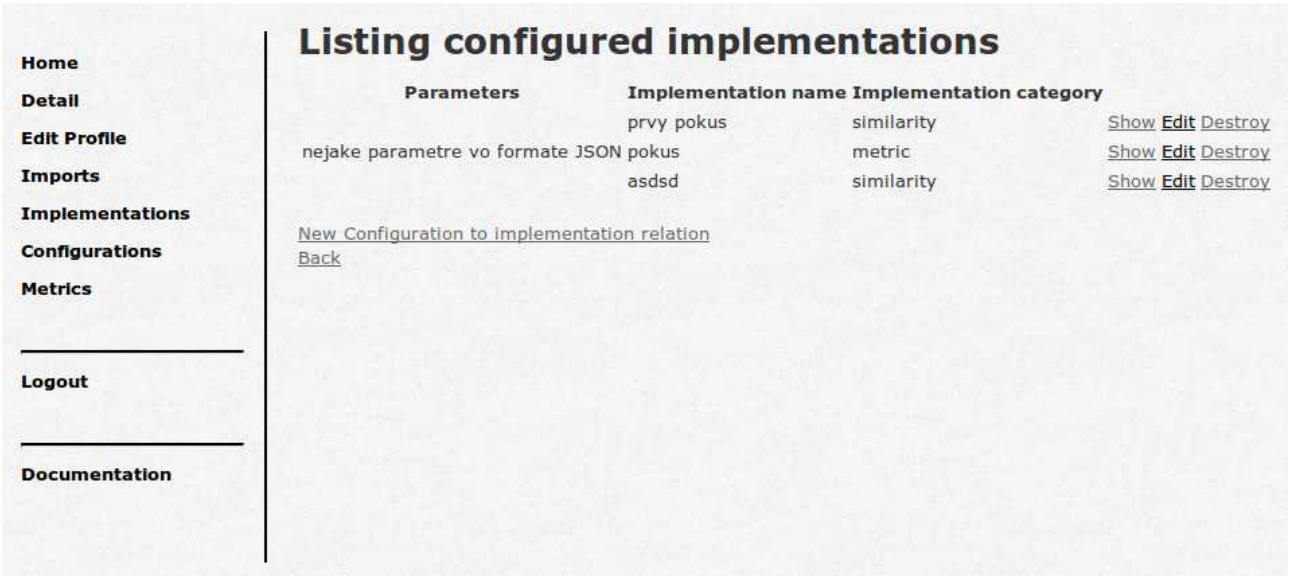
Obrázok E.11: Detail implementácie spolu s možnosťou pripojiť alebo nahrať .jar súbor

Správa konfigurácií

Ak sme prihlásený ako systém, tak sa v ľavom menu nachádza odkaz *„Configurations“*. Po kliknutí na tento odkaz sa zobrazia všetky doteraz vytvorené konfigurácie. Po kliknutí na odkaz *„New Configuration“* sa zobrazí obrazovka na nastavenie detailov novej konfigurácie (Obrázok E.12). Na tejto obrazovke je potrebné vyplniť najmä meno a informáciu o tom, či je konfigurácia aktuálna alebo nie. Následne po kliknutí na tlačidlo *„Create Configuration“* sa zobrazí detail konfigurácie. Po kliknutí na odkaz *„Configured implementations“* sa zobrazí zoznam (Obrázok E.13) implementácií priradených ku konfigurácii. Po kliknutí na odkaz *„New configuration to implementation relation“* je možné priradiť ďalšiu implementáciu ku konfigurácií.



Obrázok E.12: Obrazovka na vytvorenie novej konfigurácie



Obrázok E.13: Zoznam implementácii priradených k danej konfigurácii

Zobrazenie metrík

Ak sme prihlásený ako systém, tak sa v ľavom menu nachádza odkaz *„Metrics“*. Po kliknutí na tento odkaz sa zobrazia najnovšie hodnoty pre jednotlivé nakonfigurované metriky (Obrázok E.14).

Obrázok E.14: Tabuľka zobrazujúca výsledky všetkých metrík

API

Api poskytuje niekoľko druho volaní:

- volanie na uploadovanie súboru na import a na spustenie importu dát,
- volanie na poskytnutie spätnej väzby o vytvorenom odporúčaní a
- volanie na získanie odporúčania.

Import súboru

Url pre volanie funkcie na import súboru je:

http://vm28.ucebne.fiit.stuba.sk/app/api/import

volanie musí byť cez HTTP akciu POST a musí mať nasledujúce parametre:

Parameter	Popis
api_key:	Apikľúč systému, ktorý žiada o import dát.
type:	Typ importu, teda či ide o import dokumentov, používateľov alebo vzťahov.

	Podľa toho parameter nadobúda hodnoty „documents“, „users“ alebo „relations“.
file:	Súbor, ktorý sa má importovať.

Príklad volania pomocou príkazu curl môže vyzerať napríklad nasledovne:

```
curl -F file=@file.xml "http://localhost/app/api/import?
api_key=tralala&type=documents"
```

Výsledok volania je xml súbor obsahujúci správu o stave spracovania:

```
<?xml version="1.0" encoding="UTF-8"?>
  <result>working</result>
```

alebo xml súbor obsahujúci chybovú hlášku:

```
<?xml version="1.0" encoding="UTF-8"?>
<errors>
  <error>api_key parameter missing</error>
  <error>file parameter missing</error>
  <error>unauthorized access</error>
</errors>
```

Poskytnutie spätnej väzby

Túto funkciu využíva systém, ktorý poskytuje spätnú väzbu o interakcií používateľa odporúčaním, ktoré sme poskytli. Toto volanie je pre na poskytnutie spätnej väzby o jednom odporúčaní pre jedného používateľa. Ak systém potrebuje poskytnúť väčšie množstvo informácií o spätnej väzbe, tak by mal použiť funkciu na import dát.

Url pre volanie funkcie na poskytnutie spätnej väzby je:

http://vm28.ucebne.fiit.stuba.sk/app/api/feedback<?parameters>

volanie musí byť cez HTTP akciu GET a musí mať nasledujúce parametre:

Parameter	Popis
api_key:	Apikľúč systému, ktorý žiada o import dát.
type:	Typ importu, teda či ide o import dokumentov, používateľov alebo vzťahov. Podľa toho parameter nadobúda hodnoty „documents“, „users“ alebo „relations“.
weight:	Váha spätnej väzby
user_id:	Identifikácia používateľa

document_id:	Identifikácia dokumentu
recommendation_id:	Identifikácia odporúčania

Výsledok volania je xml súbor obsahujúci správu o stave spracovania:

```
<?xml version="1.0" encoding="UTF-8"?>
  <result>OK</result>
```

alebo xml súbor obsahujúci chybovú hlášku:

```
<?xml version="1.0" encoding="UTF-8"?>
<errors>
  <error>api_key parameter missing</error>
  <error>type parameter missing</error>
  <error>weight parameter missing</error>
  <error>user_id parameter missing</error>
  <error>document_id parameter missing</error>
  <error>recommendation_id parameter missing</error>
</errors>
```

Získanie odporúčania

Url pre volanie funkcie na získanie odporúčania je:

http://vm28.ucebne.fiit.stuba.sk/app/api/recommend<?parameters>

volanie musí byť cez HTTP akciu GET a musí mať nasledujúce parametre:

Parameter	Popis
api_key:	Apikľúč systému, ktorý žiada o import dát.
user:	Identifikácia používateľa v systéme, pre ktorého chceme odporúčanie.
result_limit:	Limit na počet výsledkov.
config:	Identifikácia configurácie odporúčania.
docord:	Poradie dokumentov, *„true“* ako zostupne, *„false“* ak vzostupne podľa hodnotenia.
userord:	Poradie používateľov, *„true“* ako zostupne, *„false“* ak vzostupne podľa hodnotenia.
feedback_type:	Zoznam typov spätnej väzby použitých pri odporúčaní oddelený čiarkami. Tento atribút nieje povinný, ak sa neuvedie, tak sa zohľadňujú všetky typy spätnej väzby.

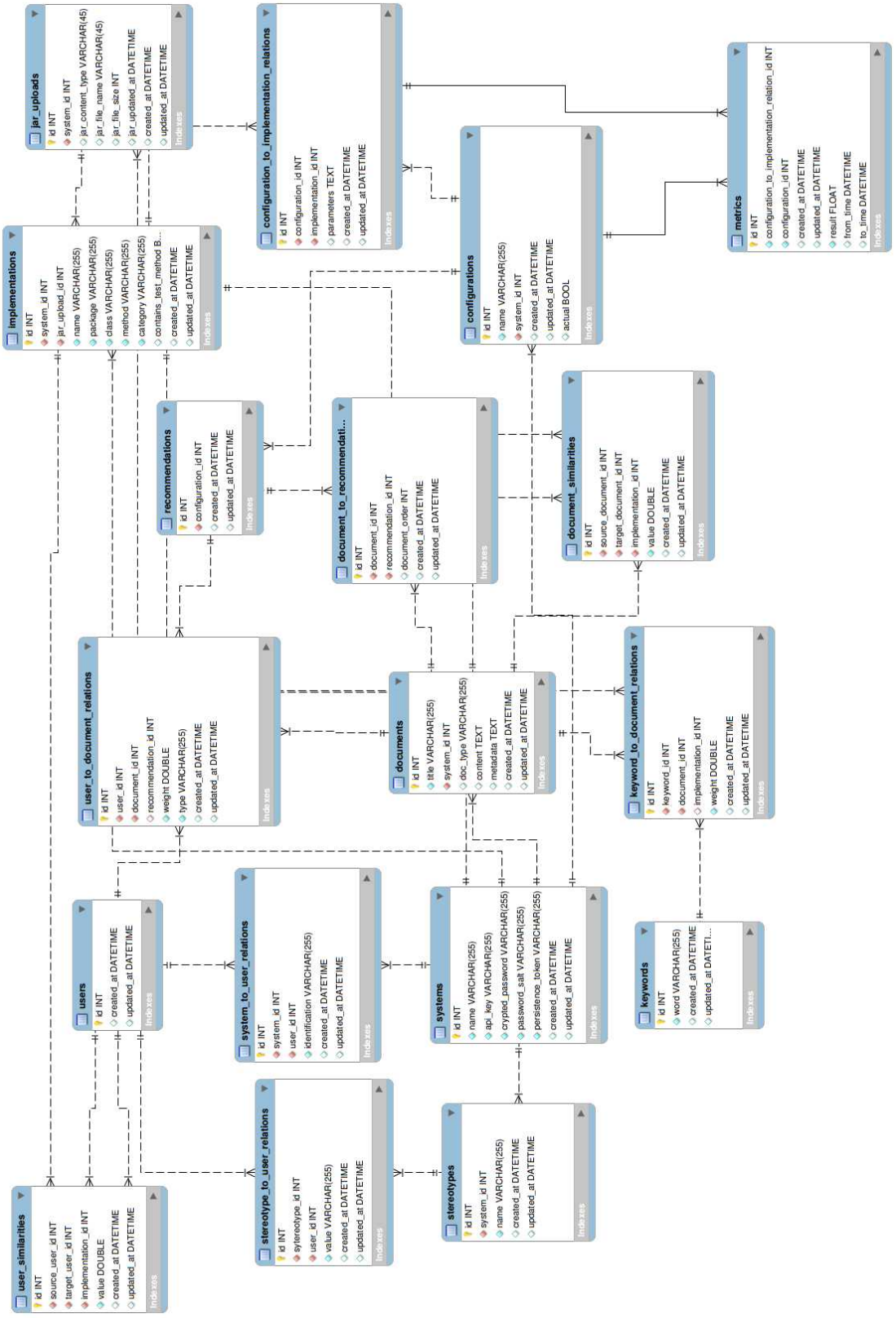
Výsledok volania je xml súbor obsahujúci zoznam odporučených dokumentov:

```
<?xml version="1.0" encoding="UTF-8"?>
<results>
  <result>2</result>
  <result>65</result>
</results>
```

alebo xml súbor obsahujúci chybovú hlášku:

```
<?xml version="1.0" encoding="UTF-8"?>
<errors>
  <error>api_key parameter missing</error>
  <error>user_identification parameter missing</error>
  <error>result_limit parameter missing</error>
  <error>configuration_id parameter missing</error>
  <error>order_desc_docs parameter missing</error>
  <error>order_desc_users parameter missing</error>
</errors>
```

Príloha F-Dátový model po prvom semestri



Obrázok F.1: Dátový model po prvom semestri

Opis nových dátových entít

implementations – Reprezentácia implementácie pluginu v systéme. Sú v nej uchované informácie potrebné na spustenie .jar súboru ako napríklad názov triedy, názov metódy alebo názov balíčku.

jar_uploads – Tabuľka, ktorá uchováva informácie ako meno súboru a cesta k súboru o nahraných jar súboroch.

configurations – Konfigurácia algoritmov použitých na výpočet odporúčania. Podľa informácií v konfigurácii sa spustí algoritmus na výpočet odporúčania.

configuration_to_implementation_relations – Väzobná entita medzi tabuľkou configurations a implementations. Umožňuje, aby mala jedna konfigurácia viacero implementácií a zároveň, aby mohla byť jedna implementácia vo viacerých konfiguráciách.

imports – Tabuľka importov, uchováva informácie o príkazoch na importovanie súborov. Rovnako uchováva aj výsledky importu.

xml_uploads – Tabuľka, ktorá uchováva informácie ako meno súboru a cesta k súboru o xml súboroch pripravených na importovanie.

metrics – Tabuľka, ktorá uchováva výsledky výpočtov metrík. Metriky slúžia na vyhodnocovanie úspešnosti algoritmov na odporúčanie.