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
Akad. rok: 2011/2012
Predmet: Tímový projekt
Kontakt: tim12tp@googlegroups.com
Vedúci tímu: Ing. Michal Kompan
Téma: Odporúčanie
Názov tímu: 7 out of 10 Precision

Bc. Michal Cádrik (SI)
Bc. Martin Detko (SI)
Bc. Igor Slotík (SI)
Bc. Jakub Ševcech (SI)
Bc. Ľudovít Mydla (IS)
Bc. Matej Mihalik (SI)

Obsah

Zoznam skratiek

Skratka	Plné znenie
p.p.	Používateľský príbeh
T, t.p.	Technický príbeh
Z, p.z.	Požiadavka na zmenu
DB	Databáza
TZ, t.z.	Technická zmena

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:

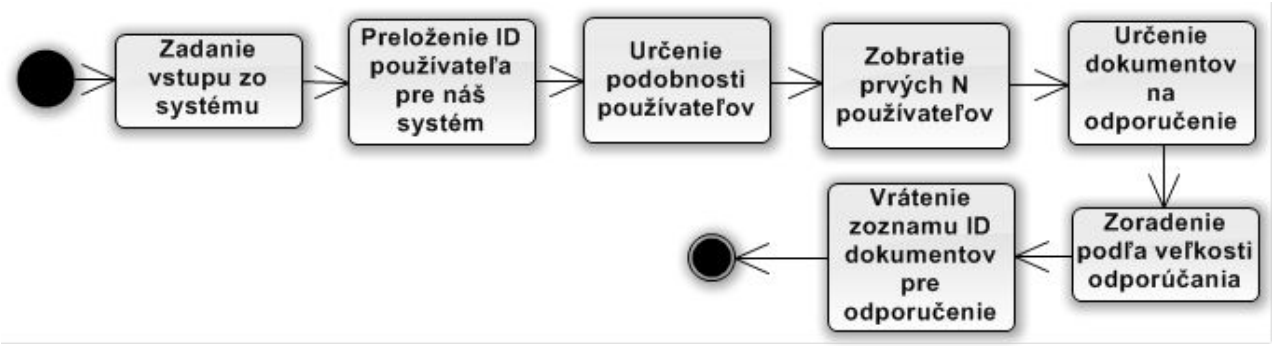
1. Zistenie používateľa, pre ktorého idem odporúčať.
2. Zistenie jemu podobných používateľov v systéme.
3. Získanie odporúčaných dokumentov od každého z podobných dokumentov.
4. Zoradenie dokumentov podľa miery odporúčania.
5. Vrátenie zoradeného zoznamu dokumentov.

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

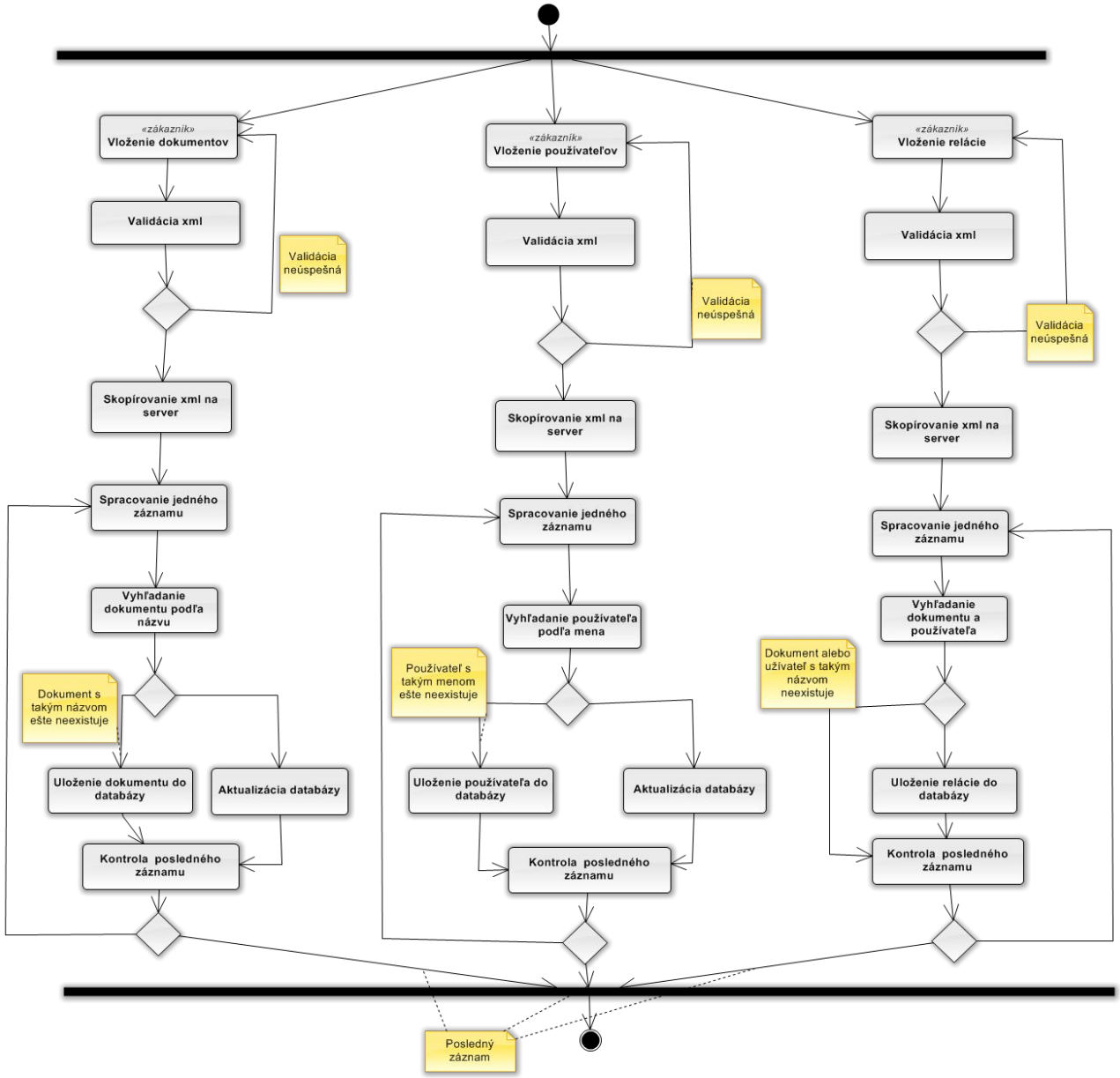
Cieľom tohto používateľského príbehu je umožniť používateľovi nahrať dáta z ich databázy do našej databázy.

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

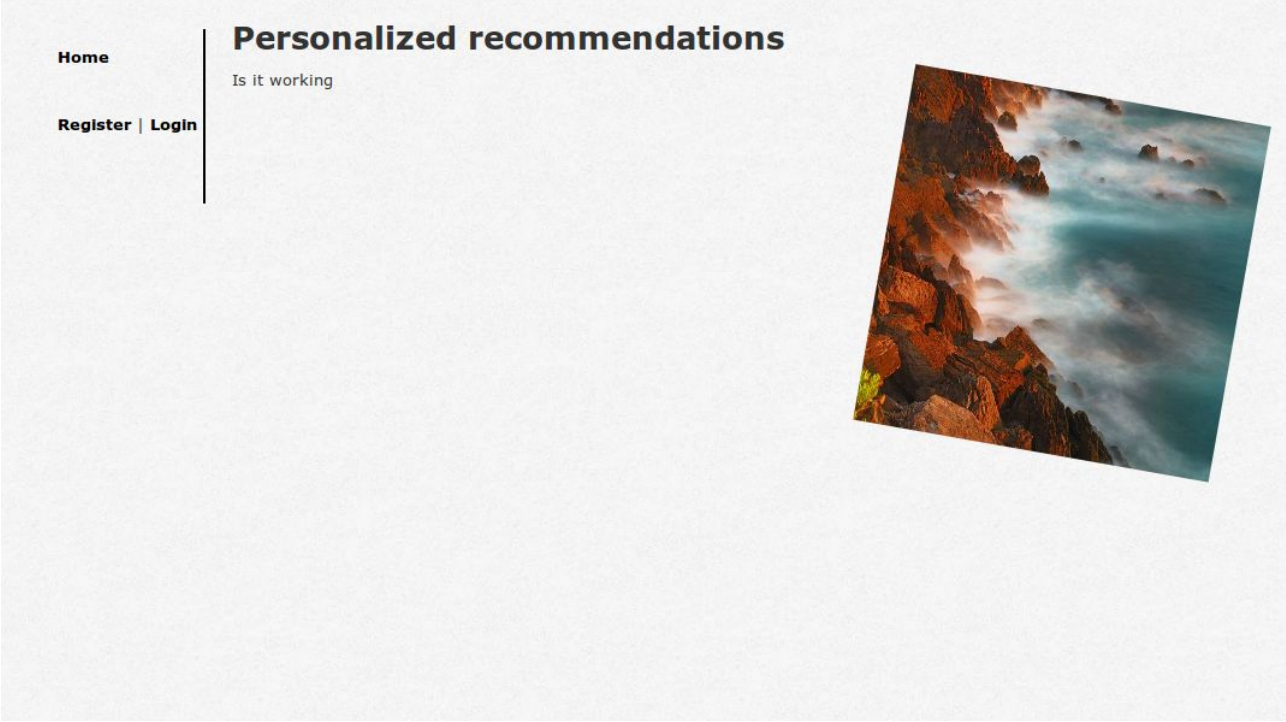
Pri implementácií sme používali rôzne technológie na vývoj. Na začiatku sme použili kombináciu

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.

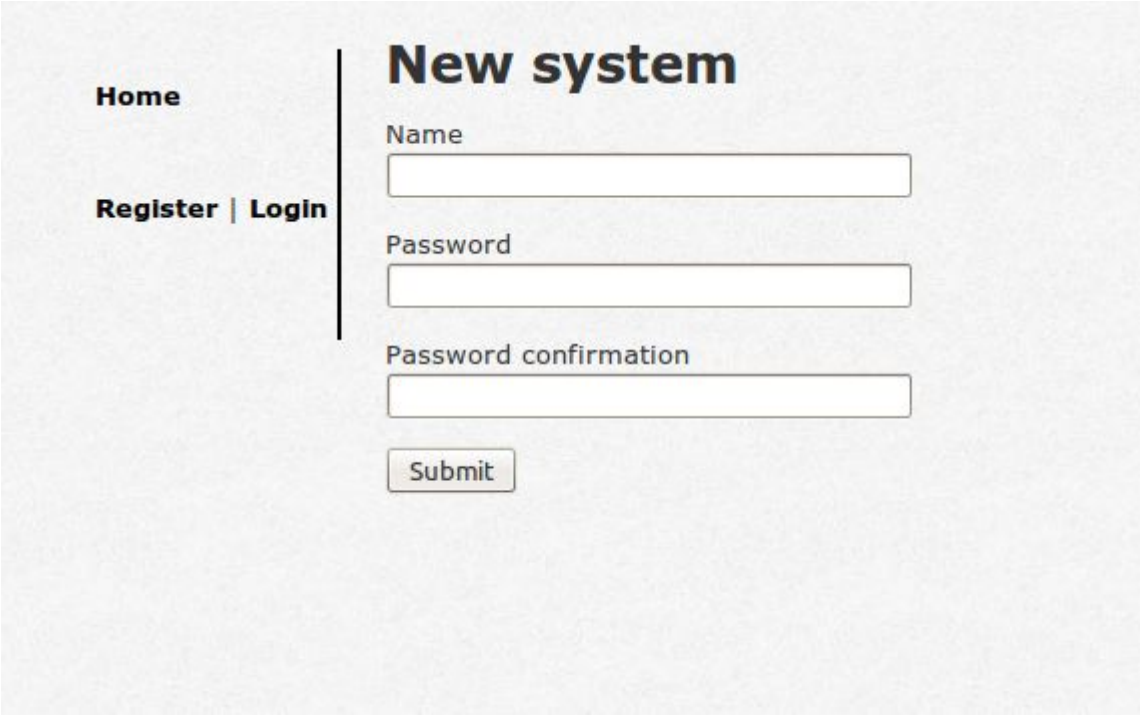
Nasadená implementácia (Obrázok 2.5) je dostupná na adrese: *http://vm28.ucebne.fiit.stuba.sk/app/*.

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

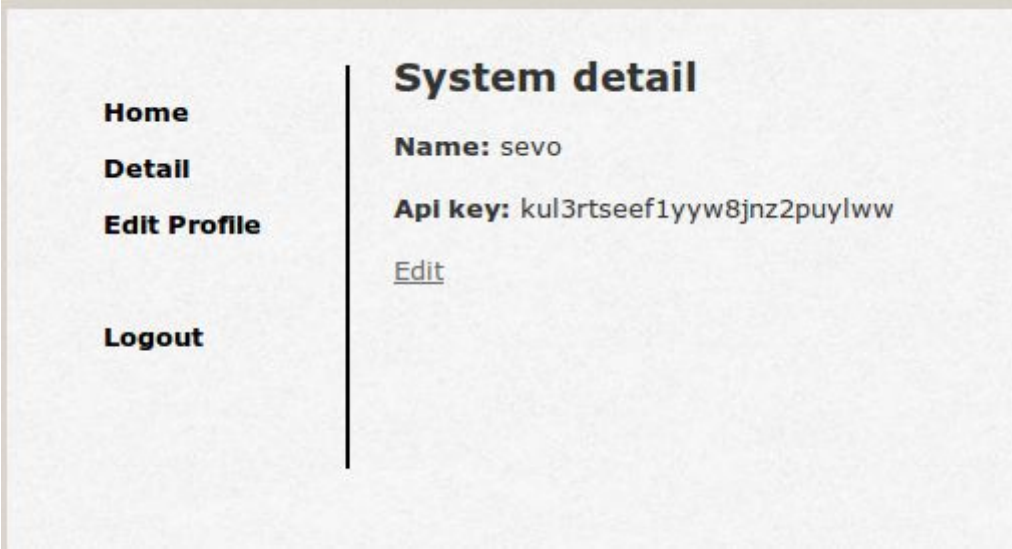
Po zaregistrovaní je každému systému priradený jedinečný apikľúč (Obrázok 2.3), ktorý bude používať ako identifikáciu pri využívaní jednotlivý API volaní.



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

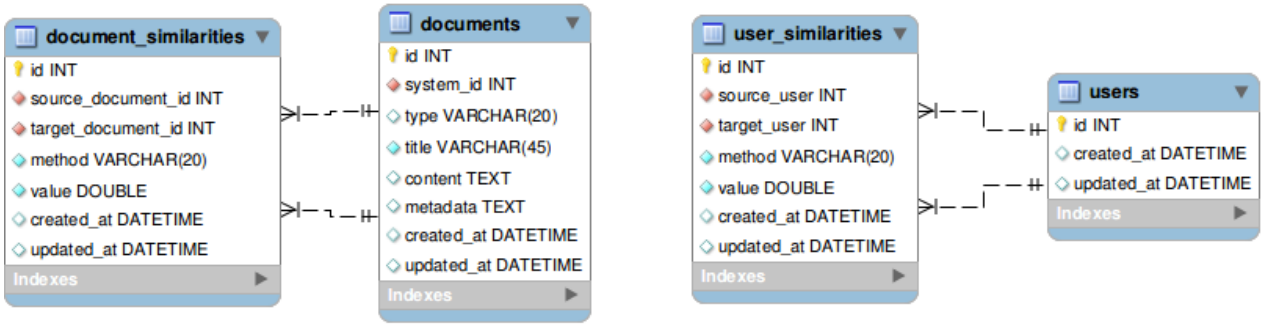
Ako používateľ chcem rátať podobnosť používateľov a dokumentov rôznymi spôsobmi, aby som mohol skúmať vplyv podobnosti na odporúčanie.

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.

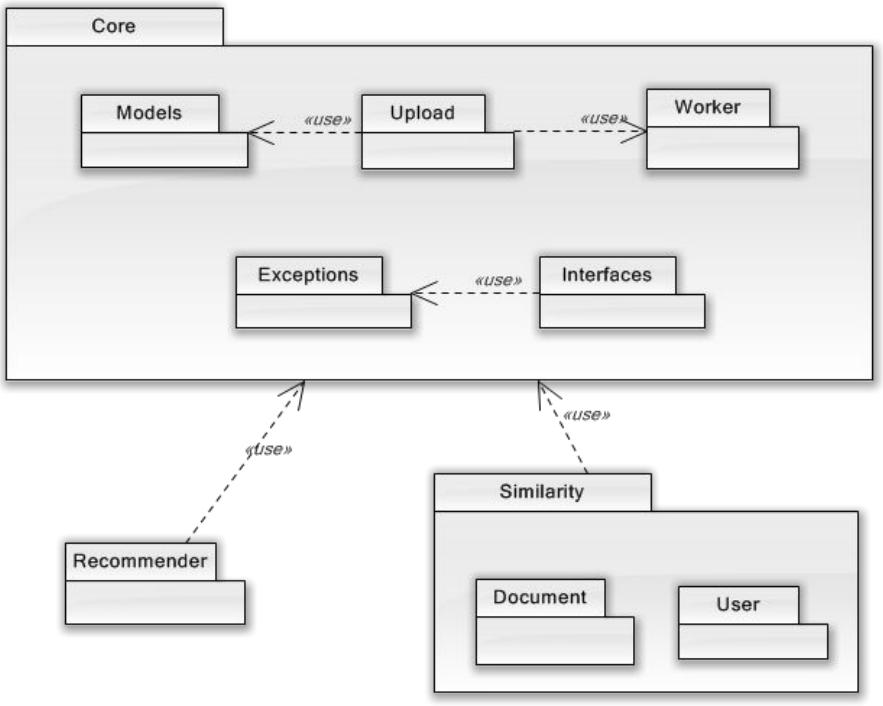
Návrh

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:

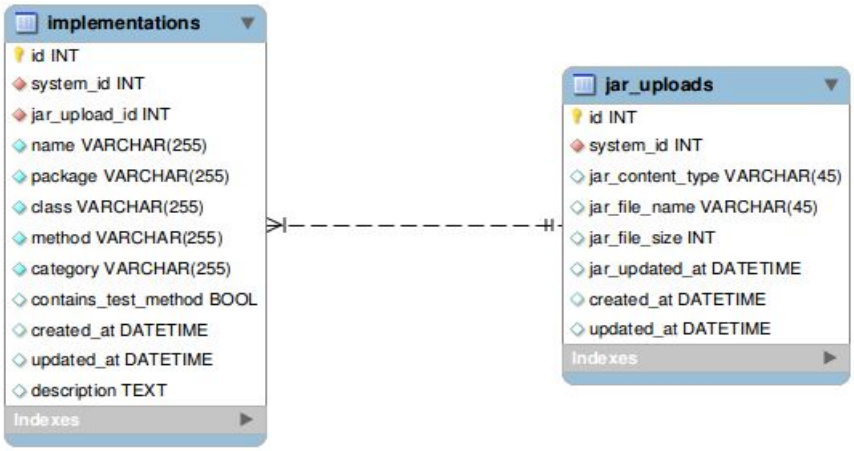
- jadro (Core) aplikácie a
- balíky ostatných častí, teda pluginov.

Jadro poskytuje všetky základné funkcie a pluginy tieto funkcie používajú.

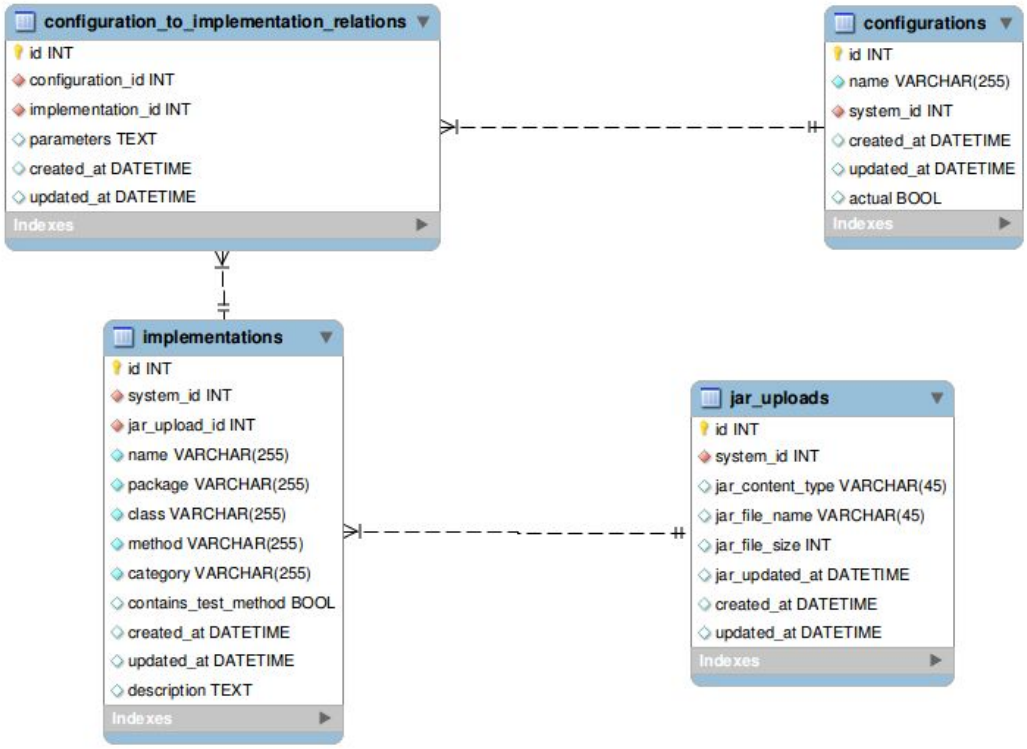
Ďalej vytvoríme funkcie, ktoré zabalia priamy prístup k databáze pre používateľov. Cieľom nášho nového návrhu pre prístup k databáze je limitovať prístup externého kódu k našej databáze z dôvodu zvýšenej bezpečnosti. Ďalším cieľom nového návrhu je schovať prácu s identifikáciou systému, tak aby sa nemusel 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.



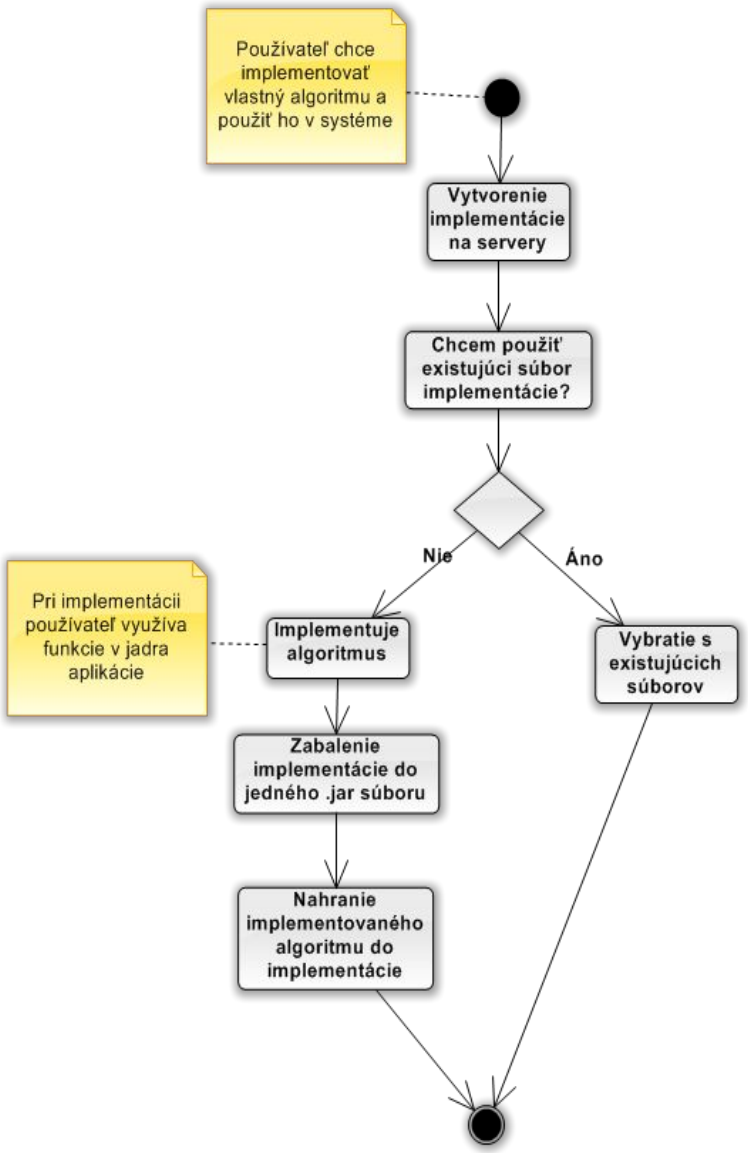
Obrázok 2.8: Časť dátového modelu zachytávajúca reprezentáciu implementácie a priradeného súboru



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.

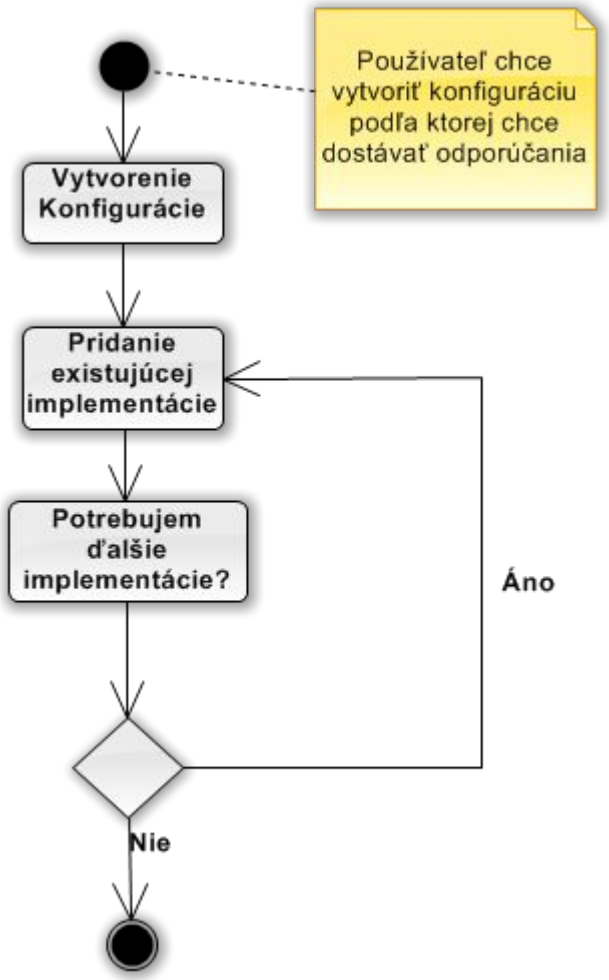


Obrázok 2.10: Proces implementovania algoritmu a vytvárania implementácie v systéme

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:

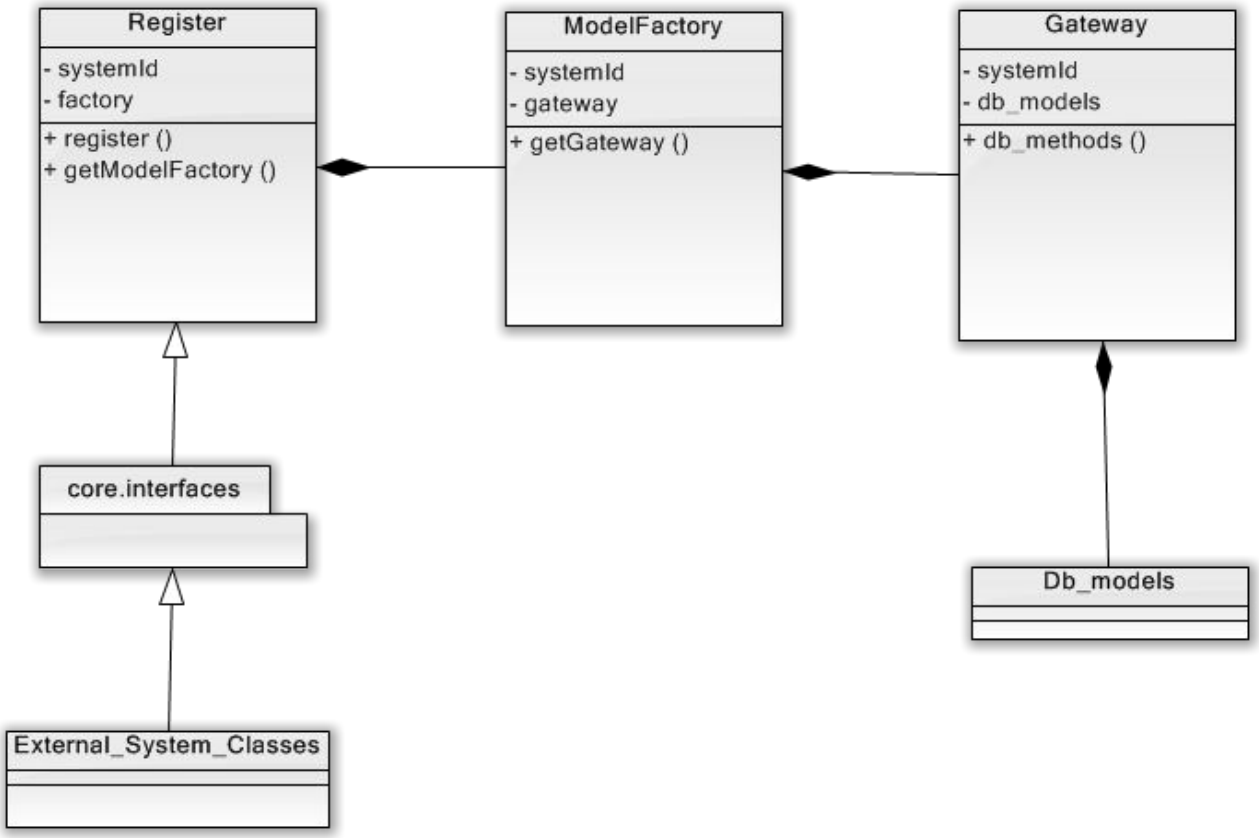
Java
Táto časť obsahuje zmenu v štruktúre balíkov v časti aplikácie implementovanej v jazyku Java. Celý projekt sa nachádza v balíku *sk.stuba.fiit.tp1112.recommender* a obsahuje všetky balíky vyplývajúce z diagramu na obrázku číslo 2.7. Nasleduje stručný opis všetkých týchto balíkov:

- *Core* – Balík obsahujúce kľúčove funkcie našeho 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.

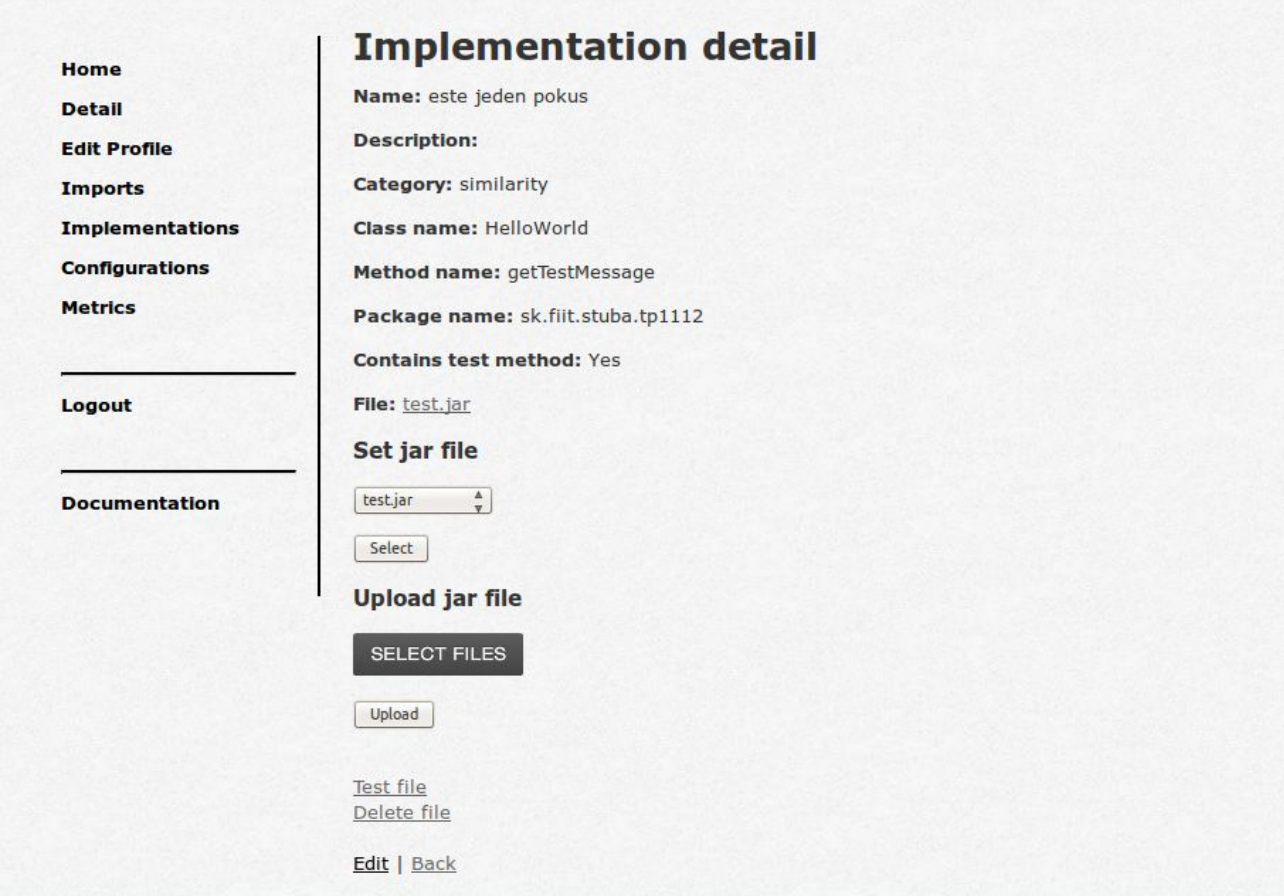
V praxi tento systém funguje tak, že z ruby sa nad inštanciou klientskej triedy zavolá metóda register, ktorá inicializuje systemId a podľa neho aj inštanciu ModelFactory. Ďalej už potom len stačí volať jednotlivé metódy cez Gateway triedu, ktorú poskytuje práve ModelFactory.

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.

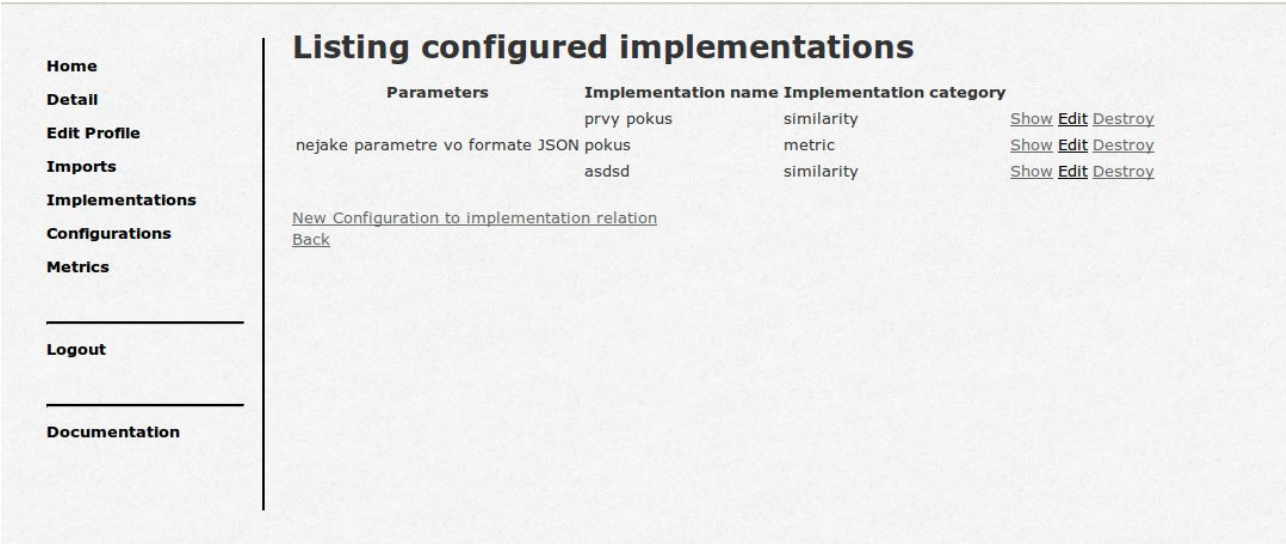
Po úprave štruktúry balíkov a prístupu k databáze sme boli nútení vykonať veľa zmien v skôr implementovaných častiach aplikácie. Preto sme vykonali úplný refactoring a previedli sme implementované algoritmy do podoby pluginov.

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.

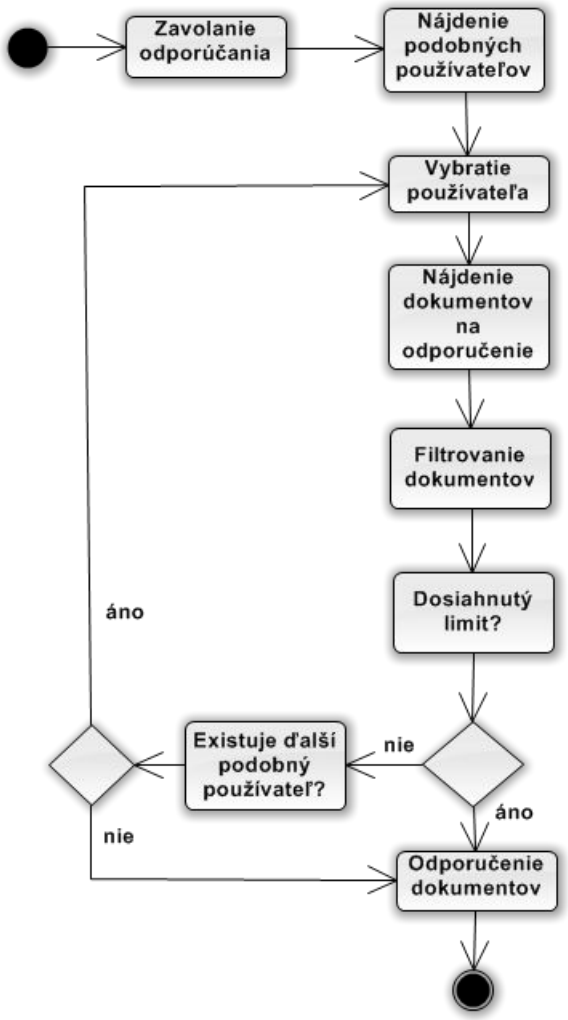
Analýza

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

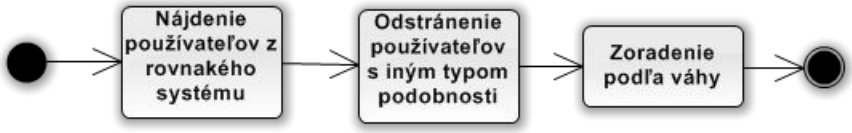
Podľa návrhu systému musí naša trieda s odporúčaním dediť od abstraktnej triedy

sk.stuba.fiit.tp1112.recommender.core.interfaces.Recommender.java. Postup algoritmu by mal byť podľa návrhu na obr. 2.17.

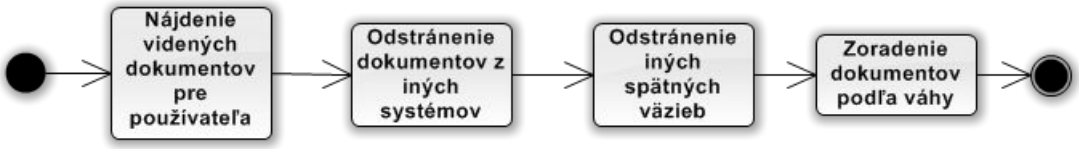


Obrázok 2.17: Zjednodušený diagram algoritmu kolaboratívneho odporúčania

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

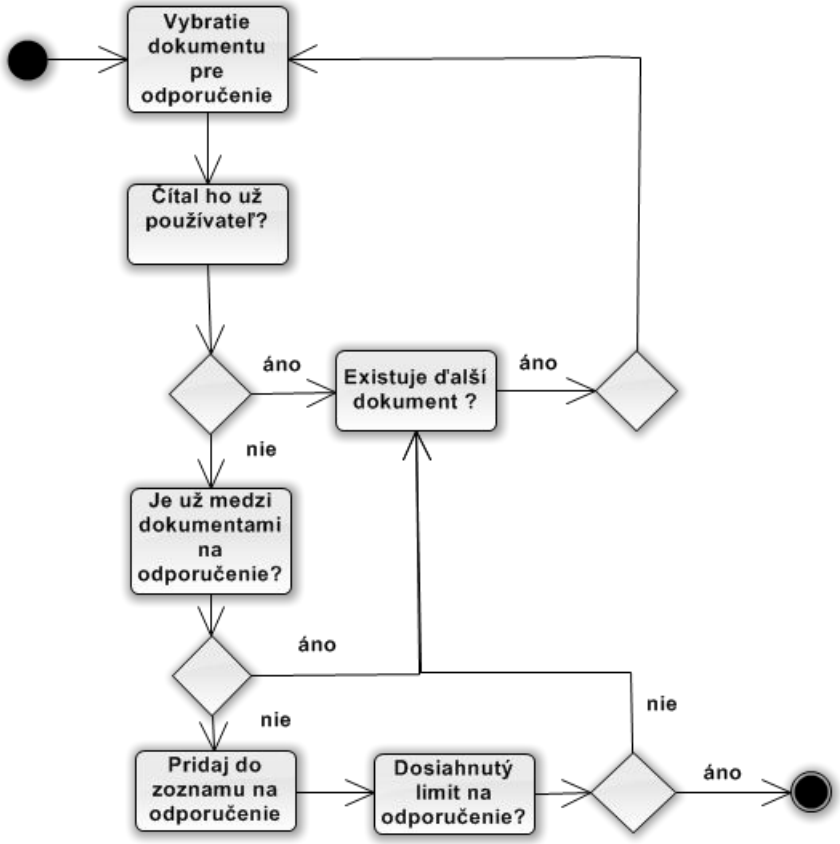


Obrázok 2.18: Postupné filtrovanie používateľov



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

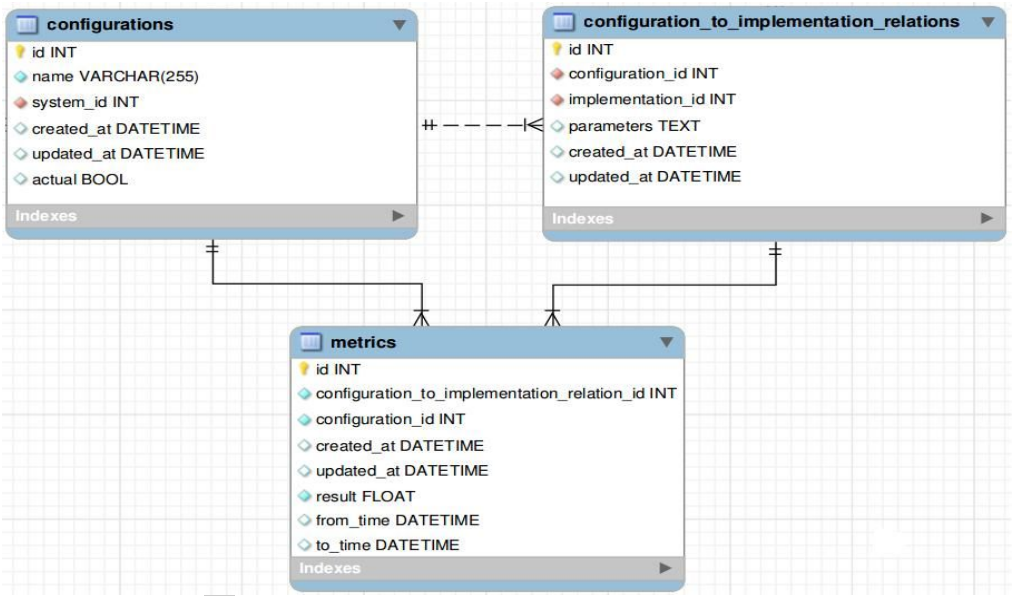
2.5.1. Vytvorenie metriky systémom pluginov, p.z. 04

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.

2.6. Šprint č. 6: „Felis Catus“

ID p. príbehu	Ako	Chcem	Aby bolo možné
15	Používateľ	Kombinovať rôzne odporúčače	Dosahovať kvalitnejšie výsledky

Tabuľka 2.11: Používateľské príbehy šprintu č. 6

ID p. príbehu	Hl. vedúci	ID úlohy	Podúloha	Zodpovedný
15	Martin Detko	15.1	Prepínací kombinovač	Ľudovít Mydla
		15.2	Kaskádový kombinovač	Michal Cádrik
		15.3	Váhový kombinovač	Igor Slotík
		15.4	Všeobecný kombinovač	Ľudovít Mydla
T04	Jakub Ševcech	T04.1	Nasadiť New Relic	Jakub Ševcech
		T04.2	Capistrano	Jakub Ševcech
		T04.3	Validácia XML	Jakub Ševcech
T05	Matej Mihalik	T05.1	Prístup do DB v jave z ruby	Matej Mihalik
		T05.2	Vkladanie testovacích dát do DB	Matej Mihalik

Tabuľka 2.12: Detailný opis úloh

2.6.1. Vytvorenie kombinovačov systémom pluginov, p. p. 15

Analýza

Kombinovače slúžia na skombinovanie viacerých druhov odporúčačov tak, aby sa dosiahlo lepšie odporučenie. Používateľ by mal mať možnosť vytvoriť si vlastný kombinovač, a preto aj táto časť našej služby bude vytvorená pomocou pluginov.

Kaskádový kombinovač

Tento typ kombinovača zavolá prvý odporúčač zo zoznamu, ktorý vráti zoznam dokumentov s váhami, s ktorými ich odporúča. Potom sa zavolá druhý odporúčač, ktorý už pracuje len nad množinou týchto odporučených dokumentov. Teda výsledok druhého odporúčača je preusporiadaný zoznam tých dokumentov, ktoré vrátil prvý odporúčač a s nanovo prepočítanými váhami odporučenia, prípadne ešte aj odrezanými poslednými dokumentami.

Váhový kombinovač

Môže volať ľubovoľne veľa odporúčačov. Pre každý volaný odporúčač má váhový kombinovač pridelenú váhu, ktorá vyjadruje vážnosť vráteného odporúčania. Získané odporučenia všetkých volaných odporúčačov sa skombinujú. Používateľovi sa vrátia dokumenty s najvyššou celkovou váhou, pričom si používateľ určuje, koľko dokumentov sa mu má vrátiť.

Prepínací kombinovač

Prepínací kombinovač ktorý kombinuje dve odporúčania sleduje počet výsledkov prvého

odporúčača. Ak nevráti žiadne odporúčanie, použije sa odporúčanie druhé. V prípade viacerých odporúčaní by mal fungovať podobne. Ak niektorý odporúčač nevráti žiadne, alebo žiadaný počet dokumentov, preskočí (prepne sa) sa na ďalší odporúčač v poradí.

Návrh

Abstraktná trieda, od ktorej bude zdedený každý kombinovač, je definovaná v *recommender/lib/jrecommender/src/main/*sk/stuba/fiit/tp1112/recommender/core/interfaces/Combinator.java. Definuje rozhranie, ktoré bude slúžiť na volanie každého kombinovača.

```
public abstract Object[][] combine(
                    int userId,
                    Recommender[] recommendersList,
                    float[] weightsList,
                    int[] limitsList,
                    int[] implementationsList,
                    String[][] feedbacksList,
                    String[] docTypeList,
                    int combineResultLimit);
```

Každý kombinovač v sebe obsahuje parameter *userId*, ktorý je ID používateľa v našom systéme, *recommenderList* je pole odporúčačov, ktoré sa majú zavolať, *weightsList* je pole váh s akými sa majú jednotlivé výsledky odporúčačov kombinovať, *limitList* je pole limitov výsledkov pre každý odporúčač, *implementetionsList* je pole implementácií použitých odporúčačov, *feedbackList* je dvojrozmerné pole typov feedbackov, medzi ktorými sa má hľadať, *docType* je pole typov dokumentov (toto je pre každý odporúčač rovnaké) a posledný parameter *cimbineResultLimit* je celkový počet výsledkov, ktorý má vrátiť daný kombinovač.

Kaskádový kombinovač

Algoritmus musí dodržať definovanú abstraktnú triedu pre kombinovače. Samozrejme je potrebné brať do úvahy prípad, keď nám nejaký odporúčač nič nevráti. Ak nám bol vrátený prázdny zoznam v niektorej iterácií algoritmu, tak algoritmus vráti prázdne pole výsledkov. Keďže sa kombinovač bude volať vždy cez ruby vrstvu, tak sa nemusíme starať o korektnosť posielaných argumentov.

Váhový kombinovač

Algoritmus musí dodržať definovanú abstraktnú triedu pre kombinovače. Treba brať do úvahy, že používateľ nezadal vhodné parametre. Napr. v prípade, že pre každý odporúčač neexistuje váha alebo nie je definovaný počet odporúčaní, koľko môže odporúčač maximálne vrátiť, tak používateľovi je vrátená hodnota *null* s chybovou hláškou.

Prepínací kombinovač

Navrhovaný kombinovač by mal v cykle prechádzať zadanými odporúčačmi. Ak celkový počet dokumentov nedosiahne žiadaný počet, ktorý by mal kombinovač vrátiť, vezme sa do úvahy výsledok ďalšieho odporúčača a doplní sa k aktuálnym výsledkom. Ak sa dosiahne žiadaný počet odporúčaných dokumentov vráti sa výsledok aj napriek tomu, že neboli použité všetky odporúčače.

Implementácia

Kaskádový kombinovač

recommender/lib/jrecommender/src/main/sk/stuba/fiit/tp1112/recommender/combinator/CascadeCombinator.java

Implementácia algoritmu je veľmi jednoduchá. Zavolá sa prvé odporúčanie pred hlavným cyklom. V cykle sa volajú ostatné odporúčania za sebou, tak ako boli definované v zozname. Na konci každej iterácie sa kontroluje, či nie je množina výsledkov prázdny zoznam. Ak je to prázdny zoznam, tak algoritmus končí.

Váhový kombinovač

recommender/lib/jrecommender/src/main/sk/stuba/fiit/tp1112/recommender/combinator/WeightCombinator.java

Na začiatku je kontrola správnosti parametrov. Druhým krokom je zistenie maximálneho počtu rôznych vrátených odporúčaní. Potom v cykle sú postupne volané všetky odporúčače. Vrátené odporúčania sú násobené zodpovedajúcou váhou. Potom sa skontroluje, či sa momentálne spracovávané odporúčania už predtým nezískali iným odporúčačom. Ak nie, tak sa vypočítaná váha odporúčania nemení. Ak áno, tak sa aktualizuje váha odporúčania o novú váhu. Tento postup sa robí pre všetky odporúčače a všetky vrátené odporúčania. Nakoniec sa celkové váhy odporúčania zoradia. Používateľovi sa vrátia najlepšie odporúčania, pričom ich počet si určuje používateľ.

Prepínací kombinovač

recommender/lib/jrecommender/src/main/sk/stuba/fiit/tp1112/recommender/combinator/SwitchingCombinator.java

Kombinovač v cykle pre každý odporúčač získa jeho odporúčanie. Spája postupne tieto odporúčania a ak množstvo odporúčaných dokumentov dosiahne určitý počet, ukončí cyklus a vráti tieto dokumenty. Za účelom spájania dvoch zoznamov odporúčaní sa implementovala metóda, ktorá zachováva aj poradie podľa váh. Takisto aj kontroluje a vynecháva duplikáty dokumentov. Pre zachovanie poradia sa implementovala aj vnorená trieda *CompareByWeight* implementujúca rozhranie *Comparator*.

Testovanie

Všetky testy pre kombinovače sa nachádzajú v *recommender/lib*/jrecommender/src/test/sk/stuba/fiit/tp1112/recommender/combinator/.

Kaskádový kombinovač

Kombinovač bol otestovaný pomocou knižnice JMock na jednoduchom prípade, kedy sa použijú 4 rôzne odporúčače. Test prebehol úspešne.

Váhový kombinovač

Kombinovač bol otestovaný pomocou knižnice JMock na jednoduchom prípade, kedy sa použijú 4 rôzne odporúčače. Test prebehol úspešne.

Prepínací kombinovač

Kombinovač bol otestovaný pomocou knižnice JMock na jednoduchom prípade, kedy sa použijú 4 rôzne odporúčače. Test prebehol úspešne.

2.6.2. Nasadiť New Relic, t. p. T04.1

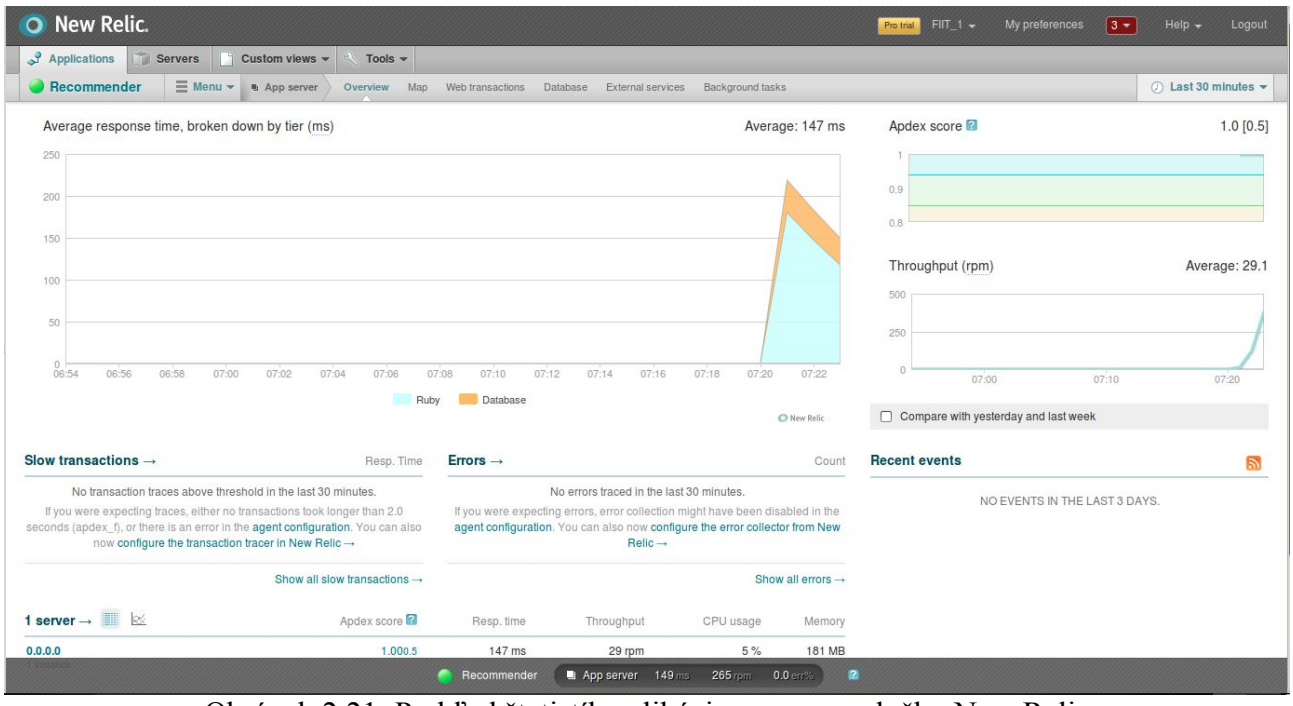
Analýza

Projekt sa blíži do stavu, keď ho potrebujeme nasadiť na server a testovať v reálnej prevádzke.

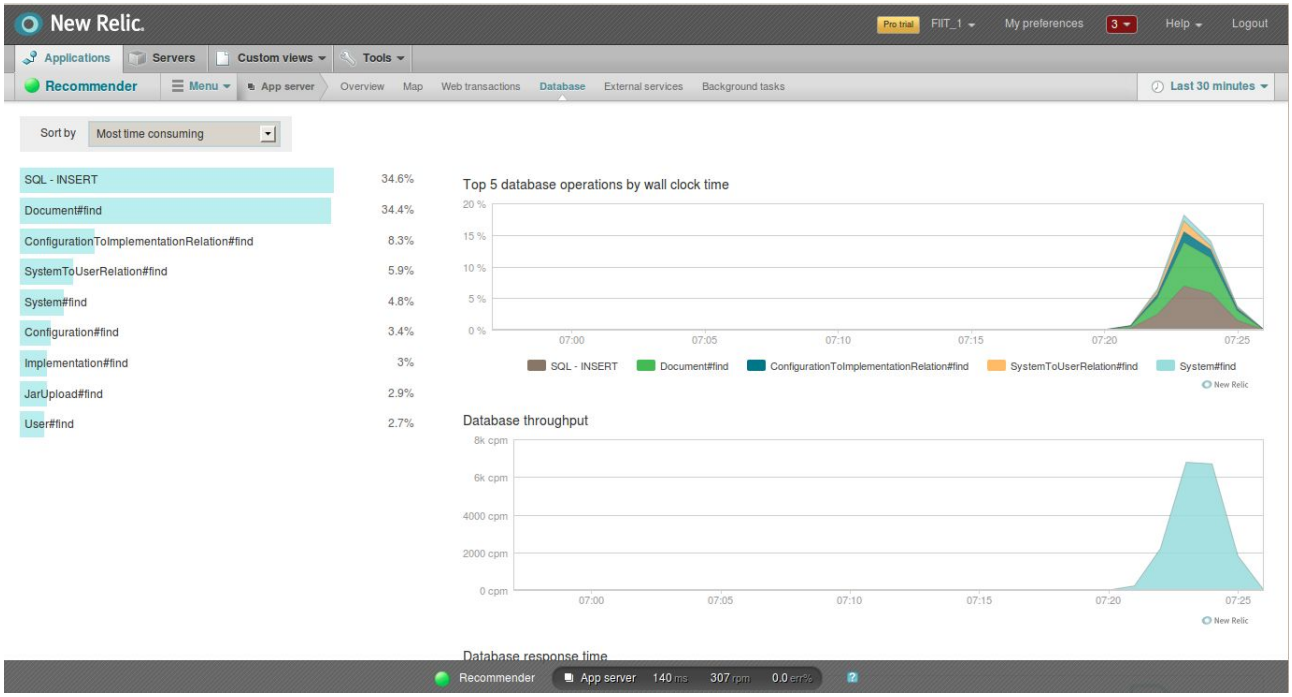
V reálnej prevádzke je potrebné sledovať, ako riešenie odpovedá na používateľské požiadavky, sledovať výkon aplikácie, zachytávať a riadiť prípadné vzniknuté chyby. Existuje niekoľko služieb, ktoré umožňujú sledovať aplikáciu v reálnom čase a umožňujú napríklad riadenie vzniknutých chýb. Niektoré z nich sú napríklad New Relic (http://newrelic.com/) alebo Airbrake (http://airbrake.io). Rozhodli sme sa v našom projekte použiť službu New Relic, pretože poskytuje veľké množstvo funkcií a poskytuje sadu funkcií, ktoré sú dostupné zadarmo. Okrem toho na prvých 15 dní používania sú dostupné všetky funkcie, ktoré služba poskytuje. Služba nám umožňuje sledovať výkon aplikácie, rozlišuje koľko času z odpovede spotrebovala databáza, aplikácia, ako dlho sa stránka zobrazovala a podobne (Obrázok 2.21). Umožňuje odhaľovať pomalé časti aplikácie na úrovni jednotlivých dopytov na databázu (Obrázok 2.22) a na úroveň stránok a controllerov, ktoré ich obsluhujú. Aplikácia monitoruje chyby, ktoré vznikli pri behu aplikácie a upozorňuje na ne pomocou emailu. Aplikácia poskytuje základný prehľad o používateľoch, ktorý aplikáciu používajú: počet návštev, prehliadač, ktorý používatelia používajú, ich operačný systém a miesto z kadiaľ sa pripájajú.

Implementácia

Služba priamo podporuje aplikácie napísané v jazyku Ruby s použitím frameworku Ruby on Rails, takže implementácia je pomerne jednoduchá. Stačí pridať do zoznamu použitých gemov pripravený gem, vytvoriť konfiguračný súbor, nastaviť v ňom názov aplikácie prípadne ďalšie hodnoty, ktoré sa nezhodujú s prednastavenými hodnotami. Okamžite po spustení aplikácie v produkčnom nastavení sa začnú odosielať informácie do služby New Relic a je možné ich sledovať pomocou webového rozhrania.



Obrázok 2.21: Prehľad štatistík aplikácie pomocou služby New Relic



Obrázok 2.22: Prehľad prístupov k databáze pomocou služby New Relic

2.6.3. Capistrano, t. p. T04.2

Capistrano je nástroj na automatizovanie nasadzovania aplikácie na server. Nasadzovanie je riadené pomcou jedného konfiguračného súboru (/config/deploy.rb), v ktorom sú definované úlohy, ktoré sa dajú spúšťať pomocou príkazu „cap“ vyvolaného terminálu. Pomocou tejto aplikácie sa dá nasadiť aplikácia, nainštalovať chýbajúce knižnice, zmigrovať databáza a podobne jedným príkazom bez toho, aby bolo potrebné sa prihlasovať na server, kde má aplikácia bežať.

Definovali sme kroky na:

- nahranie aktuálnej verzie aplikácie na server
- spustenie, zastavenie, reštartovanie aplikácie
- zmigrovanie databázy
- preloženie časti aplikácie napísanej v jazyku Java
- nahranie chýbajúcich knižníc
- a spustenie úloh, ktoré sa majú vykonávať na pozadí
- ...

2.6.4. Validácia XML, t. p. T04.3

Pri importovaní súborov do aplikácie je potrebné kontrolovať či tieto súbory zodpovedajú predpísanej štruktúre. Máme vytvorenú sadu XSD súborov, ktoré popisujú XML súbory, ktoré je možné importovať do aplikácie. Vyskytli sa problémy s validáciou súborov ak sme definovali maximálny počet prvkov, ktoré môže xml súbor obsahovať. Po prerobení importovania XML

súborov cez SAX parser a po vyriešení problémov s nahrávaním veľkých súborov na server bolo možné odstrániť tieto obmedzenia, a teda aj problémy s validáciou súborov.

Problémy s nahrávaním veľkých súborov spôsoboval server, pod ktorým bežala aplikácia. Server Glassfish totiž všetky nahrávané súbory držal v pamäti a až keď bol celý nahraný ho uložil z pamäte na pevný disk. Pri nahrávaní väčšieho súboru jednoducho nestačila pridelená pamäť a aplikácia spadla. Problém sme vyriešili výmenou serveru za server Mongrel, ktorý súbory ukladá do dočasných súborov a po skončení nahrávania ich premiestni na ich konečné miesto.

2.6.5. Prístup do DB v jave z ruby, t. p. T05.1

Už dlhšie existoval v našom systéme problém, a to síce prepínanie pre prácu s testovacou databázou v Java kóde volanom cez Jruby prostredie. Prepínanie medzi reálnou a testovacou databázou fungovalo na základe statických metód, ktoré menili premenné, v ktorých sa spojenie na databázu uchovávalo, a následne sa toto spojenie ťahalo z triedy *HibernateUtil*. Tento prístup síce fungoval v prostredí Eclipse, ale cez volanie java kódu cez prostredie Jruby tento prístup z neznámych príčin nefungoval.

Nakoľko sme predpokladali, že chyba bude niekde v prepínaní premennej cez statické metódy, zvolil sa nový prístup, a to síce taký, že premenná uchovávajúca spojenie s databázou je definovaná na začiatku inicializácie java objektu, s ktorým sa bude pracovať (pomocou boolean premennej určujúcej, či sa jedná o testovaciu alebo štandardnú databázu), a táto premenná sa uchováva ako atribút objektu, teda máme istotu, že tento atribút sa viac už nezmení, a bude sa pracovať so správnym spojením. Toto riešenie náš problém úspešne vyriešilo.

2.6.6. Vkladanie testovacích dát do DB, t. p. T05.2

Analýza

Počas toho, ako sme robili rôzne zmeny v systéme, sme museli meniť aj testy a občas sa stalo, že sme museli zmeniť aj množinu testovacích dát. Nakoľko sme (správne) predpokladali, že zmien ešte bude prebiehať relatívne dosť veľké množstvo, rozhodli sme sa zjednodušiť proces vkladania testovacích dát do databázy. Doteraz boli tieto dáta priamo zaznamenané v kóde, čo nebolo veľmi príjemné, keď bolo treba nad týmito dátami vykonať nejakú zmenu.

Návrh

Rozhodli sme sa tieto testovacie dáta abstrahovať a ako dostatočne jednoduché a elegantné riešenie plne postačujúce našim potrebám sa nám zdalo riešenie tieto testovacie dáta vyextrahovať priamo z kódu, a miesto toho ich načítavať zo zvlášť súborov v CSV formáte.

Implementácia

V prvom rade sme sa rozhodli vytvoriť abstraktnú triedu *TestCase,* ktorú budú všetky naše testy rozširovať. Táto trieda bude mať na starosti samotné načítavanie a ukladanie testovacích dát tak, aby sme túto časť logiky vyextrahovali zo samotných testov a nakoľko bude táto logika spoločná pre všetky testy, vytvorenie abstraktnej triedy bolo jednoznačne najlepšie riešenie.

Ďalej, všetky CSV súbory sa budú nachádzať v priečinku *recommender/lib/jrecommender/csv* nášho projektu a každý CSV súbor bude mať názov triedy, na ktorú sa testovacie dáta vzťahujú. Napríklad testovacia trieda *CorrelationTest* bude svoje testovacie dáta ukladať v súbore *lib/jrecommender/csv/CorrelationTest.csv*.

Samotný formát CSV súborov je nasledovný:

```
@TABLE;documents
id;system_id;identification
1;'1';'1_1'
2;'1';'1_2'
3;'1';'1_3'
4;'1';'1_4'
21;'2';'2_1'
25;'2';'2_5'
28;'2';'2_8'
```

@TABLE je identifikátor, ktorý označuje, že sa začína definovať nová tabuľka. Na tom istom riadku sa potom nachádza názov tabuľky. Ďalší riadok obsahuje vymenované stĺpce tabuľky, do ktorých sa bude ukladať. Ostatné riadky obsahujú samotné dáta, ktoré sa budú ukladať. Jeden riadok zodpovedá jednému riadku tabuľky. Prázdne riadky sa preskakujú, nepredstavujú žiaden nový záznam. Na oddeľovanie jednotlivých hodnôt sa používa bodkočiarka. Reťazce (stringy) a znaky sa vymedzujú jednoduchými apostrofmi.

Samotná testovacia logika jednotlivých testov sa potom musí nachádzať v metóde s názvom *test*, preťažujúc tým metódu z abstraktnej triedy *TestCase*.

Testovanie

Testovanie prebiehalo jednoducho skontrolovaním databázy po dokončení načítavania údajov či obsahuje dáta, ktoré by mala obsahovať. Taktiež, kontinuálne fungovanie všetkých testov v systéme ďalej svedčí o fungovaní nami navrhnutého nového postupu.

2.7. Šprint č. 7: „Glaucomys Sabrinus“

ID p. príbehu	Ako	Chcem	Aby bolo možné
TZ01	Vedúci projektu	Zmeniť algoritmy podľa nového DB modelu	Používať odporúčanie podľa aktuálneho modelu DB

Tabuľka 2.13: Používateľské príbehy šprintu č. 7

ID p. príbehu	Hl. vedúci	ID úlohy	Podúloha	Zodpovedný
TZ01	Michal Cádrik	TZ01.1	Zmena kolaboatívneho odporúčača	Michal Cádrik
		TZ01.2	Zmena odporúčača založeného na obsahu	Ľudovít Mydla
		TZ01.3	Zmena kaskádového kombinovača	Michal Cádrik
		TZ01.4	Zmena prepínacieho kombinovača	Igor Slotík
		TZ01.5	Zmena váhového kombinovača	Ľudovít Mydla

Tabuľka 2.14: Detailný opis úloh

2.7.1. Zmena kombinovačov a odporúčačov, t. z. TZ01

Zmena nastala v pristupovaní do databázy a v tvorbe testov. Aby všetko opäť správne fungovalo bolo potrebné zmeniť doposiaľ naprogramované odporúčače a kombinovače a kompletne prerobiť všetky testy. Taktiež je potrebné, aby bolo odporúčanie vykonávané čo najrýchlejšie, a preto je potrebné optimalizovať tieto algoritmy čo najlepšie ako sa len dá.

Optimalizácia sa dá vykonávať hlavne pomocou databázových operácií, ktoré sú rýchlejšie ako spracovanie priamo v programe. Treba však dávať pozor na to, aby sme nepoužívali počas jedného behu odporúčania veľa prístupov do databázy, keďže naraz bude musieť prebiehať viacero odporúčaní a keby sa stále volali požiadavky na databázu, tak by sa systém značne spomalil.

Taktiež namiesto vyberania celej databázovej tabuľky do zoznamu v programe nie je efektívne a dá sa nahradiť databázovým volaním v iteráciach s nastavenou hodnotou *offset*. Hodnota tohto offsetu by nemala byť veľmi malá a ani nie príliš veľká. Momentálne je v kolaboratívnom odporúčaní a v riešičoch problému studeného štartu nastavená na 100 položiek.

2.8. Šprint č. 8: „Haliaeetus Leucocephalus“

ID p. príbehu	Ako	Chcem	Aby bolo možné
16	Používateľ	Mať spôsoby odporučenia aj pre nových používateľov	Eliminovať problém studeného štartu
17	Používateľ	Bezpečné pridávanie nových algoritmov	Prevádzkovať službu bezpečne a mať dáta v bezpečí
18	Používateľ	Získať kľúčové slová z dokumentov	Vytvárať si vlastné algoritmy v podobe pluginov na získavanie kľúčových slov z textu a ich následné používanie

Tabuľka 2.15: Používateľské príbehy šprintu č. 8

ID p. príbehu	Hl. vedúci	ID úlohy	Podúloha	Zodpovedný
16	Michal Cádrik	16.1	Random riešič studeného štartu	Michal Cádrik
		16.2	Najčítanejšie dokumenty	Michal Cádrik
		16.3	Najviac ohodnocované dokumenty	Michal Cádrik
17	Jakub Ševcech	17.1	Vytvoriť bezpečné vkladanie modulov do projektu	Jakub Ševcech
18	Matej Mihalik	18.1	Predspracovanie dokumentu	Ľudovít Mydla
		18.2	Vytvorenie základného pluginu na získavanie keywordov	Matej Mihalik

Tabuľka 2.16: Detailný opis úloh

2.8.1. Riešiče problému studeného štartu, p. p. 16

Analýza

Riešiče studeného štartu sú určené na odporúčanie používateľom, ktorí sú v systéme noví a nevieme mu korektne prideliť podobnosť s inými používateľmi pomocou kolaboratívneho odporúčania a ani pomocou odporúčania na základe obsahu, lebo si používateľ ešte nič nepozrel. Rozhodli sme sa pre tri základné techniky.

Návrh

Random dokumenty

Tento riešič bude odporúčať z množiny všetkých dokumentov, ktoré sú v danom systéme. Teda aj už čítané nejakým používateľom aj nečítané, teda jeho veľkou výhodou je, že odporúča aj nečítané dokumenty. Do výsledku nezahŕňa dokumenty, ktoré už prečítal používateľ, ktorému odporúčame. Všetky dokumenty majú váhu odporučenia nastavenú na 1. Nemusí vrátiť zadaný počet

dokumentov.

Najčítannejšie dokumenty

Riešič studeného štartu, ktorý odporúča len z množiny už čítaných dokumentov v systéme, okrem čítaných daným používateľom. Odporúča najčítanejšie dokumenty tak, že každý používateľ mohol daný dokument čítať maximálne raz. Teda počíta počet používateľov, ktorý dokument videli. Jeho slabou stránkou je neodporúčanie dokumentov, ktoré ešte nikto nevidel, alebo sú v systéme krátko a videlo ich málo používateľov. Dokumenty majú nastavenú váhu odporučenia nastavenú podľa vzorca (*aktuálna váha*)/(*váha odporučenia prvého prvku*), kde *aktuálna váha* je počet všetkých používateľov, ktorí videli daný dokument v systéme okrem používateľa, ktorému odporúčame.

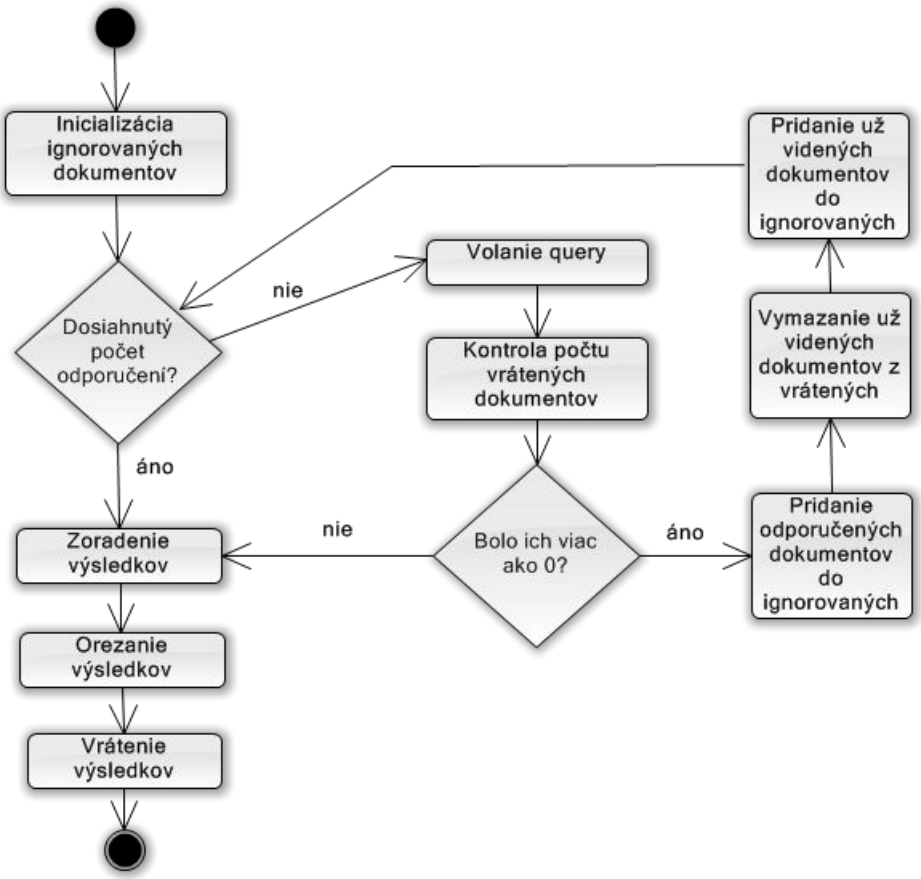
Najviac hodnotené dokumenty

Taktiež odporúča len z množiny videných dokumentov. Odporúča dokumenty, ktoré sú najviac hodnotené. Tu treba dat pozor, lebo do úvahy sa berie suma hodnotení všetkých feedbackov, ktoré sú zadané, aby sa v nich hľadalo. Takže ak dáva stále jeden používateľ na jeden dokument veľa feedbackov, tak sú tam všetky sčítané. Výsledná váha odporúčania je počítaná rovnako ako pre najčítanejšie dokumenty s rozdielom v tom, že *aktuálna váha* je súčet všetkých feedbackov pre daný dokument.

Implementácia

Abstraktná trieda pre riešič problému studeného štartu je implementovaná v *recommender/lib/jrecommender/src/main/sk/stuba/fiit/tp1112/recommender/core/interfaces/ColdStart.java.*

```
public abstract Object[][] getAdditionalDocs(int userId, Object[] docIdentifications,
int resultLimit, String[] feedbackType, String[] docType);
```



Obrázok 2.23: Diagram všeobecného riešiča problému studeného štartu

Podľa rozhrania dokážu všetky typy odporučiť iba podľa definovaných typov feedbacku a typu dokumentu. Ak tieto ostanú prázdne, tak sa berú do úvahy všetky typy. Taktiež je možné zadať množinu dokumentov, ktoré už nemá brať do úvahy ako parameter *docIdentifications*. Parameter *resultLimit* označuje maximálny počet dokumentov, ktoré môže riešič vrátiť. Každý algoritmus prehľadáva výsledky po offsete veľkosti 100.

Algoritmus končí svoje vykonávanie, ak už má dostatočný počet dokumentov, alebo v istom kroku sa z databázy nevrátil žiadny dokument.

Random dokumenty

recommender/lib/jrecommender/src/main/sk/stuba/fiit/tp1112/recommender/coldStart/RandomColdStart.java

Pri tomto type riešiča problému studeného štartu sa nerieši zoraďovanie dokumentov, keďže dokumenty nemáme podľa čoho zoradiť.

Najčítannejšie dokumenty

recommender/lib/jrecommender/src/main/sk/stuba/fiit/tp1112/recommender/coldStart/MostReadedColdStart.java

Najviac hodnotené dokumenty

recommender/lib/jrecommender/src/main/sk/stuba/fiit/tp1112/recommender/coldStart/MostRecomm

endedColdStart.java

Testovanie

Každý typ riešiča studeného štartu bol samostatne otestovaný. Boli otestovaný na dátach, v ktorých sa nachádzali viaceré záznamy pre jeden dokument, viacero typov feedbackov pre jeden dokument, viacero používateľov, ktoré čítali daný dokument, viacero druhov dokumentov. Všetky testy prebehli úspešne.

2.8.2. Predspracovanie dokumentu, p. p. 18

Analýza

Text musí byť upravený tak, aby mohol slúžiť ako vstup pre lemmatizér slovenského jazyka. Na lemmatizáciu máme k dispozícii slovník, v ktorom sú tvary slov malými písmenami. Požiadavkou pre metódy na vypočítanie kľúčových slov je, aby boli zachované značky konca vety (.!?) Ďalšou požiadavkou je odstránenie HTML značiek v prípade importu html stránok ako dokumentov. Lemmatizácia slovenského textu znamená upraviť každé slovo na jeho základný tvar – lemmu. Keďže slovenčina je dosť komplexný jazyk, neexistuje štatisticky úspešný algoritmus na lemmatizovanie slovenských slov.

Návrh

Navrhujeme, aby každá úprava textu bola implementovaná ako samostatná metóda a nie ako jedna veľká. Bude potrebná metóda pre odstránenie HTML značiek, zmenu na malé písmená, odstránenie zbytočných interpunkčných znamienok. Kvôli úspešnému získavaniu kľúčových slov je dôležité, aby sa v texte nenachádzali stop slová. Preto je potrebné vykonať aj túto úpravu pred lemmatizáciou. Lemmatizáciu navrhujeme vykonať za pomoci slovníka. Kde ku každému slovu bude priradený aj základný tvar. Ako vstup do lemmatizácie očakávame text v malých písmenách a bez stop slov.

Implementácia

Metódy odstránenia HTML značiek a odstránenia interpunkčných znamienok boli implementované s použitím regulárnych výrazov tak, že nechcené znaky sa nahradia medzerou. Zmena na malé písmená je zasa veľmi základná operácia na reťazci znakov už implementovaná v Jave. Odstránenie stop slov sa implementovalo s pomocou súboru, v ktorom je na každom riadku stop slovo z daného jazyka. Súbor sa najskôr celý načíta a stop slová sa uložia do poľa. Následne je každé stopslovo regulárnym výrazom odstránené. Aby sa zamedzilo nevhodným odstráneniam, napríklad zmena slova „lakomec“ na „lmec“ (odstránenie „ako“), je každé slovo ešte obalené regulárnymi značkami pre biele znaky „\\s“. Takto sa odstránia iba celé stopslová.

Čo sa týka lemmatizácie, tak sa nám podarilo získať slovník 600 tisíc tvarov slov a naimportovali sme ho do databázy. Lematizácia prebieha dopytom nad danou tabuľkou, kde získavame lemmy asociované s danými tvarmi slov.

Testovanie

Na náhodnom texte v slovenčine dĺžky asi 200 slov, bolo každé slovo upravené vo svojom základnom tvare. Pokrytie rôznych tvarov a foriem slov slovníkom nevieme odhadnúť, ale je veľmi

veľké, v našich testoch sme nenašli slovo, ktorému sme nevedeli prisúdiť základný tvar.

2.8.3. Vytvorenie základného pluginu na získavanie keywordov, p. p. 18.2

Ako pri väčšine funkcionality nášho systému, aj pri funkcionalite získavania keywordov z textu sme sa rozhodli zvoliť pluginový prístup. Funkcionalita získavania keywordov je koncentrovaná v balíku *sk.stuba.fiit.tp1112.recommender.keywordExtractor*, pričom všetky algoritmy na získavanie keywordov z už predspracovaného textu rozširujú triedu *KeywordExtractor* z balíka *sk.stuba.fiit.tp1112.recommender.core.interfaces*. Vlastné algoritmy na získavanie keywordov potom musia preťažiť metódu *fillMap*, ktorá napĺňa *LinkedHashMap* objekt obsahujúcu jednotlivé keywordy a im priradené váhy. Sami sme naimplementovali dve metódy na získavanie keywordov, takzvanú naivnú a query metódu.

2.9. Šprint č. 9: „Indicator Exilis“

ID p. príbehu	Ako	Chcem	Aby bolo možné
19	Používateľ	Mať metriky Precission a Recall	Pomocou nich vyhodnocovať odporúčače

Tabuľka 2.17: Používateľské príbehy šprintu č. 9

ID p. príbehu	Hl. vedúci	ID úlohy	Úloha	Zodpovedný
19	Igor Slotík	19.1	Precision a Recall metriky	Igor Slotík
		19.2	JMOCK metrika	Igor Slotík
T06	Jakub Ševcech	T06.1	Spúšťanie metrík	Jakub Ševcech
		T06.2	Indexácia a migrácia DB	Jakub Ševcech
		T06.3	Spúšťanie podobnosti	Jakub Ševcech
		T06.4	Testovanie výkonu servera	Jakub Ševcech
TZ02	Michal Cádrik	TZ02.1	Úprava collaborative odporúčača	Michal Cádrik
		TZ02.2	Zmena content-based odporúčača podľa novej špecifikácie	Ľudovít Mydla
TZ03	Martin Detko	TZ03.1	Implementacia keyword algoritmov do procesu importovania	Martin Detko

Tabuľka 2.18: Úlohy šprintu č. 9

2.9.1. Precision a Recall metriky, p. p. 19

Analýza

Úlohou metrík je porovnávanie odporúčania so spätnou väzbou používateľa. Takýmto spôsobom sa vyhodnocuje úspešnosť odporúčania. Implementovaná metrika je jednoduchšia v tom zmysle, že sa nekontroluje, o aký typ spätnej väzby ide a ani s akou váhou používateľ ohodnotil odporúčanie ani s akou váhou bolo odporúčanie vytvorené. Ak pre odporúčanie exituje spätná väzba, odporúčanie bolo úspešné, pretože používateľ na ňu reagoval pozitívne. Používateľ si môže definovať, pre aké obdobie a pre akú konfiguráciu chce získať metriku. Boli vytvorené dve jednoduché implementácie metrík. Uvediem rozdiel medzi *Precision* a *Recall* vo výpočte metrík.

Precision

Táto metrika sa počíta pomocou vzťahu:

$$Precision = \frac{|\{dokumenty\ spätnej\ väzby\} \cap \{odporučené\ dokumenty\}|}{|\{odporučené\ dokumenty\}|}$$

Recall

Táto metrika sa počíta pomocou vzťahu:

$$Recall = \frac{|\{dokumenty\ spätnej\ väzby\} \cap \{odporučené\ dokumenty\}|}{|\{dokumenty\ spätnej\ väzby\}|}$$

Návrh

Algoritmus musí dodržať definovanú abstraktnú triedu pre metriky. Používateľ má možnosť zadať namiesto začiatočného a konečného dátumu výpočtu metrík hodnotu *null*. V takom prípade bude metrika počítaná pre všetky dátumy. Začiatočný a konečný dátum, zadaný používateľom, je súčasťou *SQL query,* pomocou ktorého sa získajú potrebné informácie z daného obdobia. Problém je však v tom, že hodnota *null* nie je validná forma dátumu a algoritmus by skončil s chybou. Riešením je definovať validný začiatočný a konečný dátum, ak by používateľ zadal *null*. V takom prípade musí byť interval medzi začiatočným a konečným dátumom aj dostatočne široký. 1. Január 1970 bol určený ako začiatočný dátum a 31. December 2050 ako konečný prednastavený dátum.

Implementácia

Implementácia *Precision* a *Recall* metrík je do veľkej miery podobná.

Precision

- Získanie prístupu k databáze.
- Ak používateľ zadal *null* ako hodnotu pre dátumy, tak použi prednastavené dátumy.
- Inicializácia posunutia na 0 a maximálneho počtu získaných záznamov z databázy na 1000.
- V cykle:
  - Pýtam sa, koľko prienikov spätnej väzby používateľa a odporúčaní by sa mohlo vybrať z databázy, pre ktoré platí:
    - Také isté identifikačné číslo odporúčania sa nachádza v tabuľkách *recommendations*, *user_to_document_relations* a *document_to_recommendation_relations*
    - *id_configuration* v tabuľke *recommendations* zodpovedá identifikačnému číslu konfigurácie zadanej používateľom.
    - Dátum aktualizácie v tabuľkách *user_to_document_relations* a *document_to_recommendation_relations* musí byť v intervale medzi špecifikovanými dátumami.
    - Spätné väzby používateľa na ten istý dokument sú zlúčené podľa *recommendation_id*.
    - Záznamy sa vyberajú s posunutím.
    - Maximálny počet získaných záznamov je 1000.
  - Aktualizácia celkového prieniku medzi spätnou väzbou a odporúčaním.
  - Ak by sa získalo menej ako 1000 záznamov, tak sa cyklus ukončí.
  - Ak by sa získal maximálny počet záznamov, tak sa posunutie aktualizuje o 1000.
- Posunutie inicializuj na 0.

Potiaľto je implementácia *Precision* a *Recall* rovnaká. Nižšie je uvedená implementácia *Precision*. Pri *Recall* uvediem len rozdiel oproti *Precision*.

- V cykle:
  - Pýtam sa, koľko odporúčaní by sa mohlo vybrať z databázy, pre ktoré platí:
    - Také isté identifikačné číslo odporúčania sa nachádza v tabuľkách *recommendations*, a *document_to_recommendation_relation.*
    - *id_configuration* v tabuľke *recommendations* zodpovedá identifikačnému číslu konfigurácie zadanej používateľom.
    - Dátum aktualizácie v tabuľke *document_to_recommendation_relations* musí byť v intervale medzi špecifikovanými dátumami.
    - Odporúčania sú zlúčené podľa *recommendation_id* v tabuľke *document_to_recommendation_relations*.
    - Záznamy sa vyberajú s posunutím.
    - Maximálny počet získaných záznamov je 1000.
  - Aktualizácia celkového počtu odporúčaní.
  - Ak by sa získalo menej ako 1000 záznamov, tak sa cyklus ukončí.
  - Ak by sa získal maximálny počet záznamov, tak sa posunutie aktualizuje o 1000.
- Výpočet metriky *Precision* podľa vzťahu uvedeného v analýze.
- Zápis do tabuľky *metrics*.

Recall

- V cykle:
  - Pýtam sa, koľko spätných väzieb používateľa by sa mohlo vybrať z databázy, pre ktoré platí:
    - Také isté identifikačné číslo odporúčania sa nachádza v tabuľkách *recommendations*, a *user_to_document_relations.*
    - *id_configuration* v tabuľke *recommendations* zodpovedá identifikačnému číslu konfigurácie zadanej používateľom.
    - Dátum aktualizácie v tabuľke *user_to_document_relations* musí byť v intervale medzi špecifikovanými dátumami.
    - Spätné väzby sú zlúčené podľa *recommendation_id* v tabuľke *user_to_document_relations*.
    - Záznamy sa vyberajú s posunutím.
    - Maximálny počet získaných záznamov je 1000.
  - Aktualizácia celkového počtu spätných väzieb.
  - Ak by sa dalo získať len menej ako 1000 záznamov, tak sa cyklus ukončí.
  - Ak by sa dal získať maximálny počet záznamov, tak sa posunutie aktualizuje o 1000.
- Výpočet metriky *Recall* podľa vzťahu uvedeného v analýze.
- Zápis do tabuľky *metrics*.

Testovanie

Metriky *Precision* a *Recall* boli zatiaľ testované na takýchto dátach:

- 10000 záznamov v tabuľke *recommendations*,
- 3000 záznamov v tabuľke *user_to_document_relations*,
- 5000 záznamov v tabuľke *document_to_recommendation_relations*.

Trvanie výpočtu bolo 663 milisekúnd pre implementáciu *Precision*. Efektívnosť implementácií *Precision* a *Recall* je rovnaká.

2.9.2. Spúšťanie metrík, t. p. T06.1

Spúšťanie prepočtu metrík funguje obdobne s prepočtom podobnosti (kapitola 2.9.4, Spúšťanie podobností, t. p. T06.3), len sa do radu úloh zaraďujú úlohy na prepočet metriky nakonfigurovanej v konfiguráciách systému.

2.9.3. Indexácia a migrácia DB, t. p. T06.2

Pri stálom zvyšovaní údajov uložených v databáze by v budúcnosti vznikol problém s klesajúcim výkonom databázy. Preto je treba pridať indexy na niektoré atribúty v tabuľkách. Indexy sú potrebné na tie tabuľky, ktoré budú obsahovať najviac záznamov a ktoré sa budú najčastejšie používať. Určite treba pridať indexy napríklad na tabuľky *user_similarities, users*, *documents*, *document_similarities*, ale aj na niektoré ďalšie. Indexy je potrebné mať aj z dôvodu kontroly unikátnosti niektorých záznamov. Nesmie byť napríklad možné vytvoriť dva systémy, ktoré by mali rovnaký api kľúč alebo meno. Takéto obmedzenia sú pri viacerých atribútoch, ktoré sú podrobnejšie opísané v priloženej dokumentácii k dátovému modelu.

2.9.4. Spúšťanie podobností, t. p. T06.3

Analýza

Na ponúkanie odporúčania je potrebné poznať podobnosť medzi používateľmi a medzi dokumentami. Je nereálne, aby sa táto podobnosť počítala pri požiadavke na poskytnutie odporúčania, ale musí sa predpripraviť. Je potrebné, aby si mohol používateľ určiť, ako často sa má táto podobnosť predpočítavať, aby vedel nastaviť aktuálnosť odporúčania.

Implementácia

Implementovali sme funkciu, ktorá predpočítava podobnosť pre jednotlivé systémy podľa ich konfigurácie. Predpočítavanie prebieha na pozadí, takže priamo nespomaľuje odozvu samotného systému. Je možné nastaviť počet vlákien, v ktorých sa bude podobnosť predpočítavať, takže pri zvýšenej záťaži je možné zrýchliť predpočítavanie tým, že sa bude predpočítavať podobnosť pre viacero systémov súčasne. Samotné predpočítavanie funguje nasledovne:

- Do rady úloh na spracovanie sa zaradí úloha na predpočítavanie podobností pre jeden systém.
- Spustí sa funkcia na predpočítavanie podobnosti pre jeden systém.
- Táto funkcia vyberie všetky konfigurácie systému a postupne ich prechádza.

- Pre každú konfiguráciu vyberie nakonfigurovanú implementáciu podobnosti a prepočíta podobnosti medzi všetkými používateľmi, prípadne dokumentami.
- Po prepočítaní všetkých konfigurácií zaradí táto funkcia úlohu na predpočítavanie pre nasledujúci systém do rady úloh na spracovanie.

Pre radu úloh na spracovanie je pridelený počet vláken, ktoré úlohy postupne spracovávajú. Ďalší systém na spracovanie sa berie podľa nastavenej frekvencie prepočítavania. Po dokončení prepočítavania sa pre systém nastaví čas, kedy bol naposledy prepočítaný. Pri hľadaní systému, ktorý sa má najskôr prepočítať sa zoberie dátum posledného prepočítania, pripočíta sa k nemu frekvencia predpočítavania a nájde sa systém, ktorý má najnižší výsledný dátum. Nieje teda zaručené, že do stanoveného času sa systém prepočíta, ale dajú sa tak určiť priority s akými sa budú systémy prepočítavať.

2.9.5. Testovanie výkonu servera, t. p. T06.4

Analýza

Pre úspešné používanie odporúčacieho systému je potrebné sa uistiť, že poskytuje dostatočne vysoký výkon pre reálne použitie. Je teda potrebné overiť, že dokáže poskytovať odporúčania primerane rýchlo. Na otestovanie výkonu sme sa rozhodli použiť nástroj Apache JMeter, ktorý umožňuje opakovane vykonávať dopyty na aplikáciu a pri tom meria čas odozvy. Pomocou tohto nástroja je možné definovať rôzne postupnosti krokov a je pomocou neho možné otestovať rýchlosť odporúčania ako aj rýchlosť rozhrania na nastavenie odporúčača.

Implementácia

Vytvorili sme tri rôzne postupnosti krokov pomocou ktorých sme otestovali rýchlosť grafického prostredia ako aj rýchlosť odporúčania pri rôznej záťaži.

Prvý najjednoduchší test overoval odozvu pri načítaní úvodnej stránky. Pri tejto stránke aplikácia nepotrebuje pristupovať do databázy a nemusí spúšťať ani žiadne vložené java súbory. Pri teste s dvadsiatimi používateľmi, ktorí bez prestávky dookola pristupovali na úvodnú stránku sme dostali výsledky zobrazené v tabuľke 2.18. Tabuľka zobrazuje maximálny, minimálny a priemerný čas odozvy aplikácie pri zobrazovaní úvodnej stránky. Medián časov odozvy má hodnotu 367 milisekúnd, to znamená, že polovica používateľov bola obslúžená do 367 milisekúnd. 90% používateľov bolo obslúžených do 441 milisekúnd.

Dopyt	Počet dopytov	Priemer (ms)	Minimum (ms)	Maximum (ms)
/app	200	352	89	3133

Tabuľka 2.19: Odozva aplikácie pri zobrazovaní hlavnej stránky

Druhý test overuje rýchlosť grafického rozhrania. V tomto teste osem rôznych používateľov vykonáva nasledujúce kroky stále dookola.

1. Prihlásia sa.
2. Zobrazia detail používateľa (systému).
3. Odhlásia sa.

Výsledky tohto testu sú zhrnuté v tabuľke číslo 2.20. Cyklus, ktorý každý používateľ vykonával sa skladal zo šiestich volaní aplikácie, z ktorých každé sa vykonalo v priemere za 257 milisekúnd.

Dopyt	Počet dopytov	Priemer (ms)	Minimum (ms)	Maximum (ms)
/app	1200	196	91	3354
/app/log-in	400	203	102	580
/ app/system_sessions	400	372	182	715
/app/log-out	400	378	175	3319
TOTAL	2400	257	91	3350

Tabuľka 2.20: Odozva aplikácie pri prihlasovaní sa

Tretí test mal za úlohu otestovať rýchlosť odporúčania pri rôznom počte používateľov. Testovali sme dve situácie:

- Jeden používateľ samostatne žiada o odporúčanie.
- Dvadsať používateľov dookola bez prestávky žiada o odporúčanie.

Výsledky týchto testov sú zhrnuté v tabuľke číslo 2.21. Podľa tejto tabuľky vidíme, že rýchlosť odporúčania pre samostatného používateľa je v priemere necelých 200 milisekúnd a aj pre 20 používateľov, ktorí bez prestávky dookola žiadajú o odporúčanie je čas prijateľných 2,4 sekundy v priemere.

Počet dopytov	Počet používateľov	Priemer (ms)	Minimum (ms)	Maximum (ms)
1000	20	2413	176	5608
50	1	189	84	265

Tabuľka 2.21: Rýchlosť odporúčania v závislosti od počtu používateľov

2.9.6. Zmena odporúčačov podľa novej špecifikácie, t. z. TZ02

Zmena v odporúčačoch nastala po zmene rozhrania, v ktorom vypadlo niekoľko parametrov z hlavnej funkcie *getRecommendation*. Dohodli sme sa, že usporiadanie dokumentov a používateľov bude vždy s váhou od najväčšej po najmenšiu, teda väčšia váha značí väčšiu podobnosť vždy. Okrem tejto zmeny sa vykonali aj menšie zmeny týkajúce sa odporúčania zo zoznamu dokumentov, ale tieto zmeny sa týkajú len odporúčania na základe podobnosti dokumentov.

2.9.7. Implementácia keyword algoritmov do procesu importovania, t. z. TZ03

V prvom rade bola vytvorená nová trieda s podobným obsahom, ako trieda pre import dokumentov. Rozdiel je práve v konštruktore, ktorý obsahuje nastavenia pre získavanie kľúčových slov. V ruby vo view alebo pomocou volania API sa dá nastaviť, či je potrebné odobrať z obsahu dokumentu HTML tagy, či treba zmeniť všetko na malé písmená, či treba odobrať znamienka a iné znaky, či treba odobrať stopwordy, v akom jazyku sa má robiť lematizovanie a aký plugin sa má použiť na počítanie kľúčových slov. Všetky tieto nastavenia sa pošlú konštruktorom do novej triedy v jave. V rovnakom poradí ako je vyššie uvedené sa vykonávajú v jave na koncovom elemente dokumentu. Kľúčové slová sa počítajú z atribútu *content.* Po získaní listu kľúčových slov, sa uložia do databázy aj s dokumentom a vzájomnými prepojeniami.

2.10. Šprint č. 10: „Jynx Torquilla“

ID p. príbehu	Ako	Chcem	Aby bolo možné
20	Používateľ	Používať Porterov algoritmus	Spracovať anglické texty
21	Používateľ	Podpora AB testovania	V rámci jedného systému odporúčať rôznymi odporúčačmi pre rôznych konzumentov

Tabuľka 2.22: Používateľské príbehy šprintu č. 10

ID p. príbehu	Hl. vedúci	ID úlohy	Podúloha	Zodpovedný
20	Ľudovít Mydla	20.01	Implementácia Porterovho algoritmu	Ľudovít Mydla
21	Martin Detko	21.01	Vyrtvoriť AB testovanie	Martin Detko
TZ04	Matej Mihalik	TZ04.1	Prepočet z viacerých feedbackov na jeden	Matej Mihalik
		TZ04.2	Feedbacky s ováhovaním	Matej Mihalik
TZ05	Jakub Ševcech	TZ05.1	Zmena počítania váh v kolaboratívnom odporúčaní a riešičoch problému studeného štartu	Michal Cádrik
		TZ05.2	Zmena v prepočte podobností	Jakub Ševcech
		TZ05.3	Zmena v metrikách	Jakub Ševcech

Tabuľka 2.23: Detailný opis úloh

2.10.1. Implementácia Porterovho algoritmu, p. p. 20

Analýza

Pre angličtinu existuje algoritmus pre získavanie koreňov slova – Porterov algoritmus. Hoci je angličtina pravidelnejšia ako slovenčina, algoritmus na lemmatizáciu sme nemali čas hľadať alebo implementovať. Podobne i slovník pre lemmatizáciu ako v slovenčine. Rozhodli sme sa použiť Porterov algoritmus, ktorý je vlastne stemmer – získava koreň slova. Jeho implementácia je voľne dostupná aj pre jazyk Java. Použili sme implementáciu zo stránky ttp://www.tartarus.org/~martin/PorterStemmer

Testovanie

Na náhodnom anglickom texte (vygenerovaný za pomoci stránky http://randomtextgenerator.com/) sme testovali získaný algoritmus. Podľa našich znalostí angličtiny sme usúdili, že stemmovanie prebehol korektne.

2.10.2. Podpora AB testovania, p. p. 21

Analýza

Pre daný stav, kde bolo možné mať aktívnych viac konfigurácií, pomocou ktorých sme odporúčali, je nutné dorobiť to, aby systém presne určil konfiguráciu, podľa ktorej odporučíme konzumentovi. Je dôležité, aby tento konzument dostával odporúčania vždy na základe tej istej konfigurácie. Ďalej je potrebné, aby sme vedeli nastaviť konfigurácie, akej veľkej časti konzumentov majú odporúčať.

Návrh

Prvou možnosťou bolo vytvoriť nad konfiguráciami ďalšiu tabuľku AB testovanie, ktorej by jednotlivé konfigurácie patrili. Druhou možnosťou bolo na základe atribútov *activ* a *weight* v tabuľke *configurations* vybrať konfiguráciu, ktorou ideme odporúčať. Nevýhodou druhého riešenia je, že v jednom čase pre jeden systém môžeme mať aktívne len jedno AB testovanie. Pre jednoduchosť a po konzultácií so zákazníkom bola zvolená práve druhá možnosť.

Problém, aby každý konzument dostával odporúčanie tou istou konfiguráciou, bol vyriešený tak, že *id* konzumenta sme najprv vždy zahashovali pomocou MD5 a potom sme získali zvyšok po delení číslom rovným súčtu váh aktívnych konfigurácií. Vďaka tomu, ak sa konfigurácie nemenili konzument dostane odporúčanie, vždy tou istou konfiguráciou.

Implementácia

Najprv bol do tabuľky *configurations* doplnený atribút *weight*. Následne bol do view v ruby doplnená možnosť nastavovať tento argument. Potom pri volaní API funkcie pre odporúčanie bol z konfigurácie spravený nepovinný argument. Ak sa vo volaní argument nenachádzal, malo sa jednať o AB testovanie. Po získaní konkrétnej konfigurácie môže odporúčanie pokračovať rovnakým spôsobom ako s argumentom konfigurácie.

Funkcia pre určenie konfigurácie. Pomocou algoritmu rulety určíme, ktorú konfiguráciu použijeme. Ak všetky aktívne konfigurácie nemajú nastavenú váhu alebo majú nastavenú 0, tak pravdepodobnosť pre všetky konfigurácie bude rovnaká. Vstupom do algoritmu rulety bude práve *id* konzumenta z tabuľky *system_to_user_relations* zahashované pomocou algoritmu MD5. Algoritmus ruleta by mala vrátiť víťaznú konfiguráciu.

2.10.3. Zmena kombinovačov a odporúčačov, t. z. TZ05.1

Zmena v kolaboratívnom algoritme a v riešičoch problému studeného štartu vyplýva z možnosti uchovania viacerých feedbackov rovnakého druhu od jedného používateľa na jeden dokument v rámci jedného systému. Keďže takéto ukladanie nebolo uvažované, tak bolo potrebné zmeniť všetky použité metódy, ktoré pristupovali k databáze.

Taktiež bolo potrebné riešiť prepočet váhy odporúčania tak, aby výsledná váha bola v rozmedzí vždy od 0 po1. Dosiahnuté je to tým, že ku sume rovnakých feedbackov sa pripočíta číslo 1, čo zabráni, aby sme dostali číslo menšie ako jedna a dosadíme do vzorca *1- 1 / (suma * počet)*. *Počet* je počet zhlukovaných feedbackov a *suma* je výsledná suma týchto feedbackov.

2.11. Šprint č. 11: „Kakamega Poliothorax“

ID p. príbehu	Hl. vedúci	ID úlohy	Podúloha	Zodpovedný
T07	Jakub Ševcech	21.01	Pridanie tabuľky lemmas do migrácii	Jakub Ševcech
TZ06	Martin Detko	TZ06.1	Zmena importu relácií - doplnenie argumentu	Martin Detko
		TZ06.2	Prerobiť API pre feedback	Jakub Ševcech

Tabuľka 2.24: Úlohy šprintu č. 11

2.11.1. Pridanie tabuľky lemmas do migrácií, t. p. T07

Pri lemmatizácii slovenského textu sa používa databáza lém. Na to, aby sme ju vedeli používať v našej aplikácií je potrebné vytvoriť tabuľku *lemmas* a naplniť ju údajmi. Údaje sú vo forme SQL súboru. Aby bola úprava databázy v súlade s doterajšími migráciami, bolo potrebné vytvoriť migráciu, ktorá naimportuje tento SQL súbor do databázy.

2.11.2. Zmena importu relácií - doplnenie argumentu, t. z. TZ06.1

Do XML schémy bol doplnený argument *appended_at*. Nová schéma sa nachádza v prílohe K. Okrem toho bolo upravené aj parsovanie XML v jave.

2.11.3. Prerobiť API pre feedback, t. z. TZ06.2

Volanie API funkcie na poskytnutie spätnej väzby o videní dokumentu vyžaduje pridanie dátumu, kedy sa táto akcia vykonala. Bolo potrebné pridať atribút *appended_at* do tabuľky *user_to_document_relations*, bolo potrebné zmeniť rozhranie na volanie api funkcie a bolo potrebné upraviť návod na použitie na stránke tak, aby vyhovoval zmenenému rozhraniu.

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.

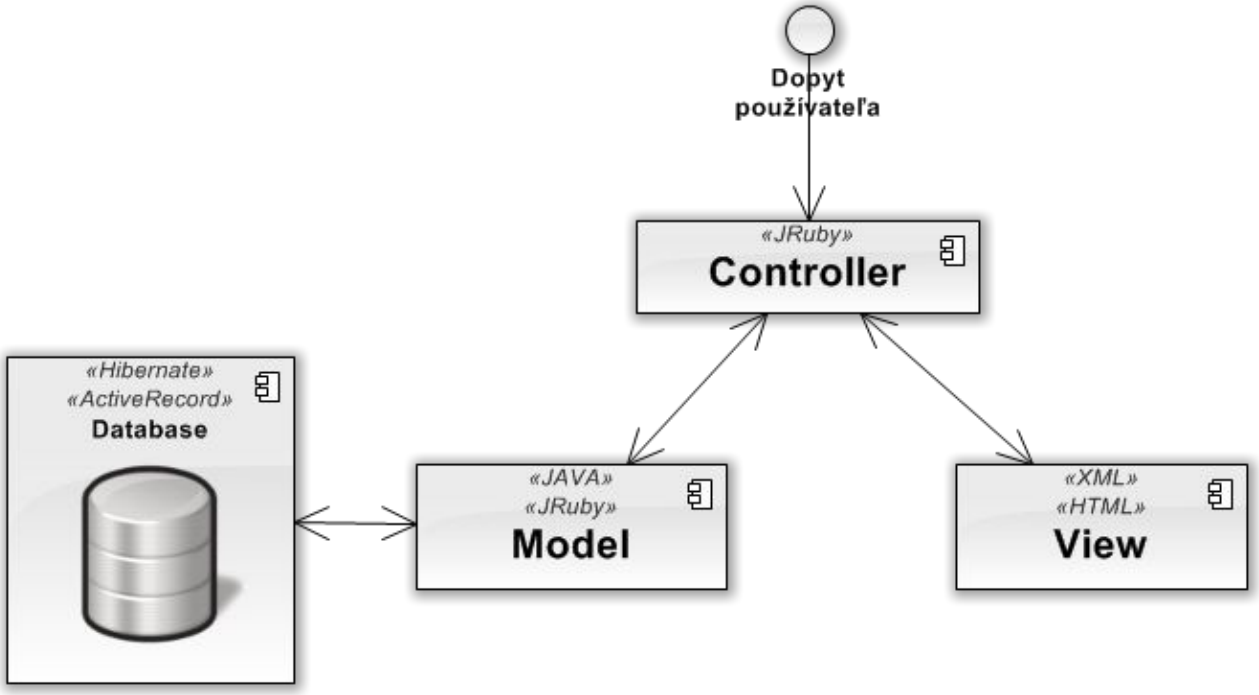
View reprezentuje vizualizáciu aplikácie pre používateľa. Táto je rozdelená na dve časti:

- 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.



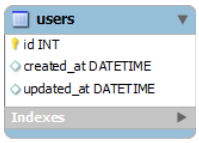
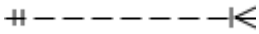
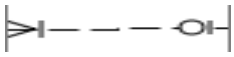
Obrázok D.1.Architektúra implementovaného prototypu

Dátový model

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

Opis použitého značenia

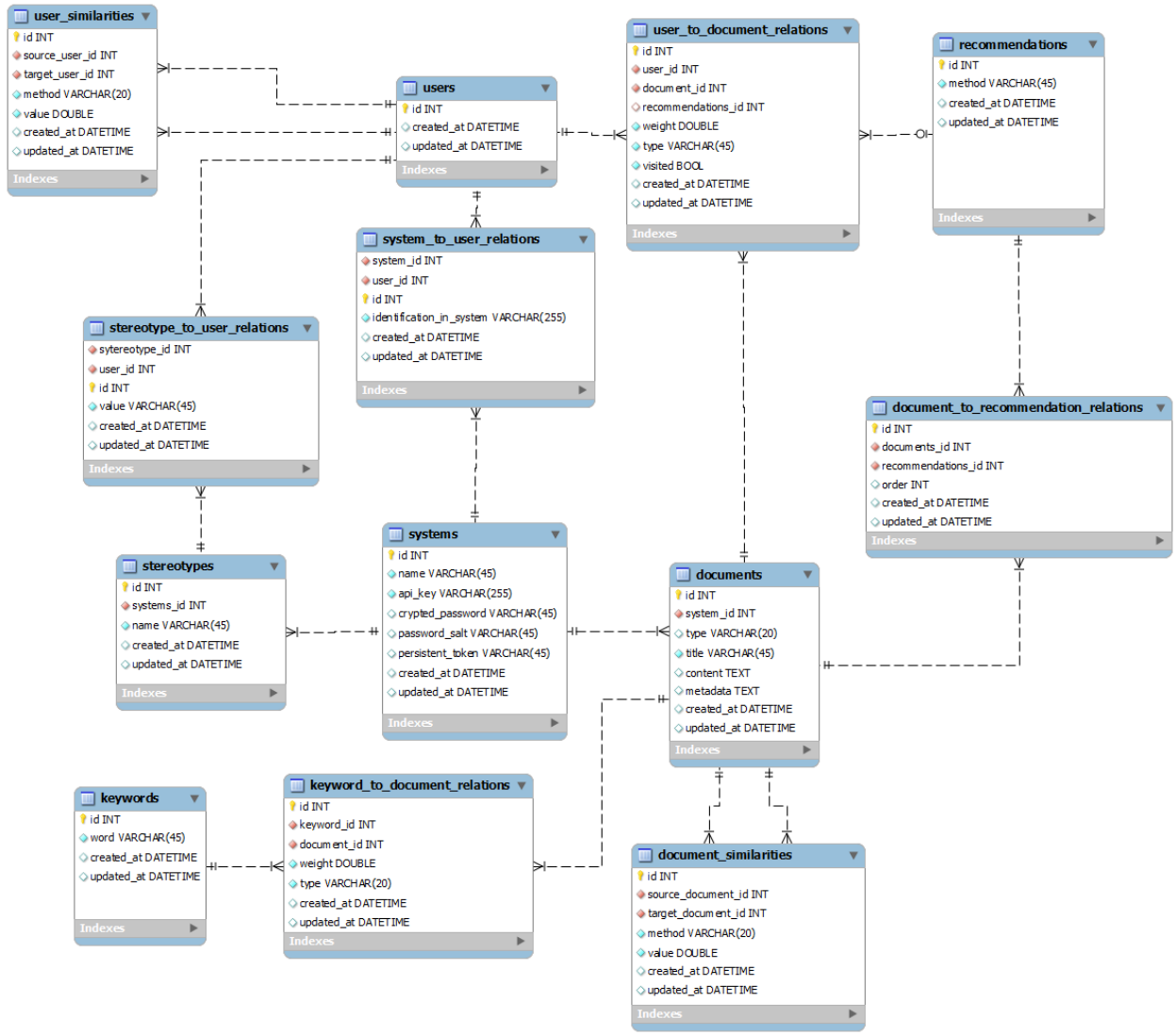
Tabuľka D.1 opisuje význam použitých značiek v diagramoch.

	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.

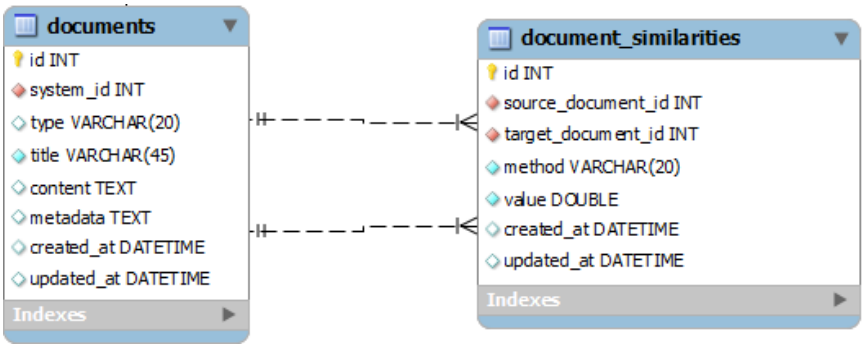
keywords – Kľúčové slovíčka dokumentov, podľa ktorého budeme zgrupovať dokumenty. Obsahuje jedine atribút *word*, čo je názov jedného kľúčového slova.

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.

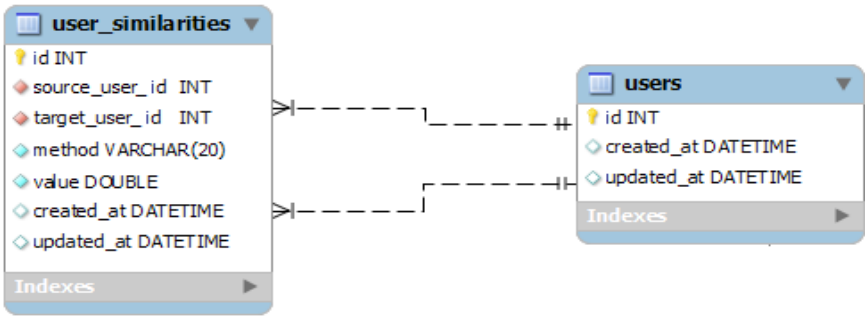
Opis vzťahov medzi entitami a relačných entít

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

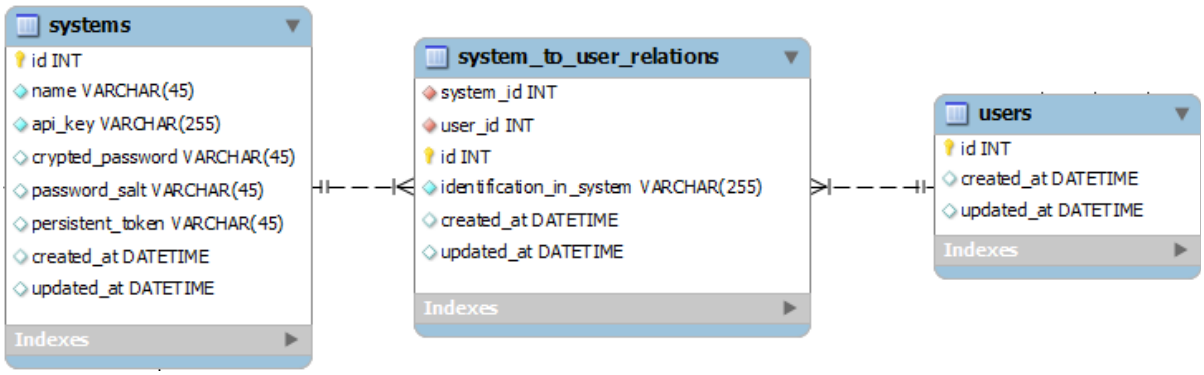
users_similarities – Na obr. D.4 sa nachádza entita, ktorá hovorí o podobnosti dvoch požívateľov. Práve preto 2 atribúty sú cudzie kľúče dvoch spomínaných používateľov. Ďalšími dvoma atribútmi sú *method* a *value*. *Method* je reťazec maximálne 20 znakov, ktorý hovorí o tom, akým spôsobom bola získaná 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

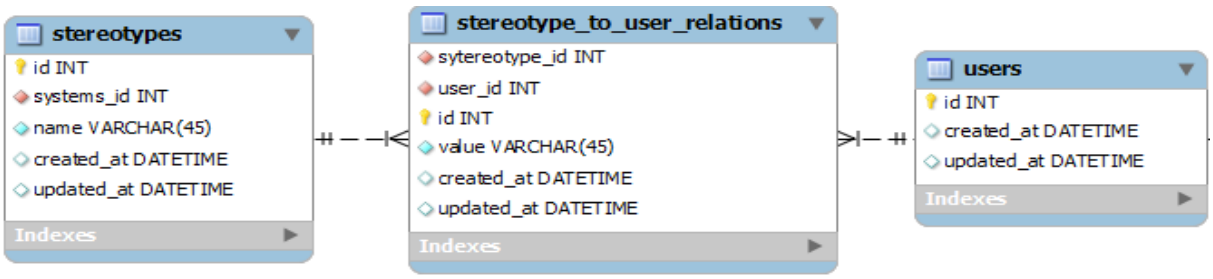
system_to_user_relations – Keďže v jednom systéme sa môže nachádzať viac ľudí a jeden človek môže používať viac systémov, bola použitá relačná entita. Tento vzťah je zachytený na obr. D.5.

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.



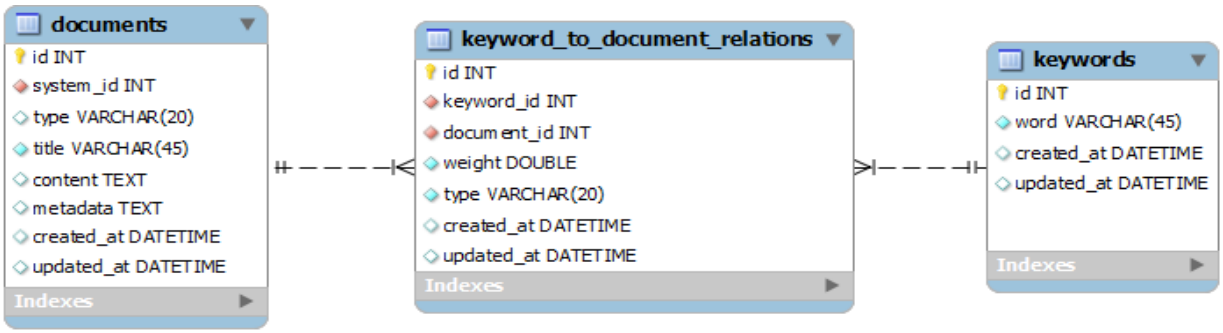
Obrázok D.5: Relačná entita medzi users a systems

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

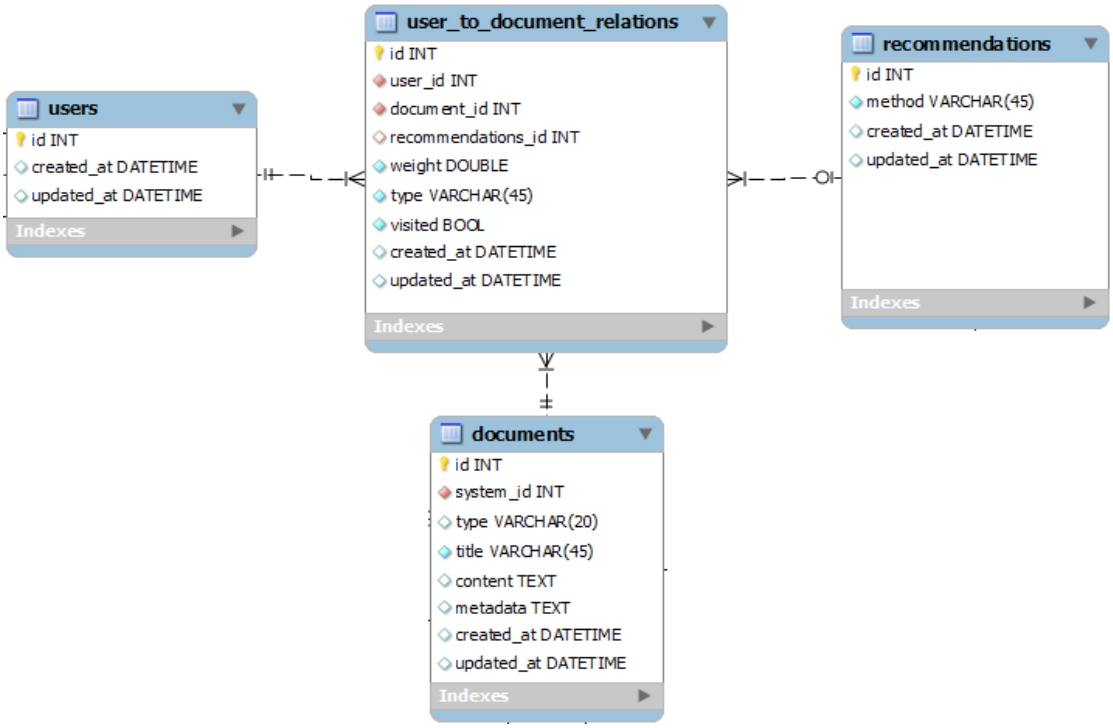
keyword_to_document_relations – Obrázok D.7 zachytáva vzťah medzi entitami *keywords* a *documents*. Táto entita obsahuje atribút *weight* ako celo-číselnú hodnotu hovoriacu o dôležitosti kľúčového slova v danom dokumente. Atribút *type* je reťazec maximálnej dĺžky 20 znakov a zachytáva typ tejto väzby.



Obrázok D.7: Relačná entita medzi documents a keywords

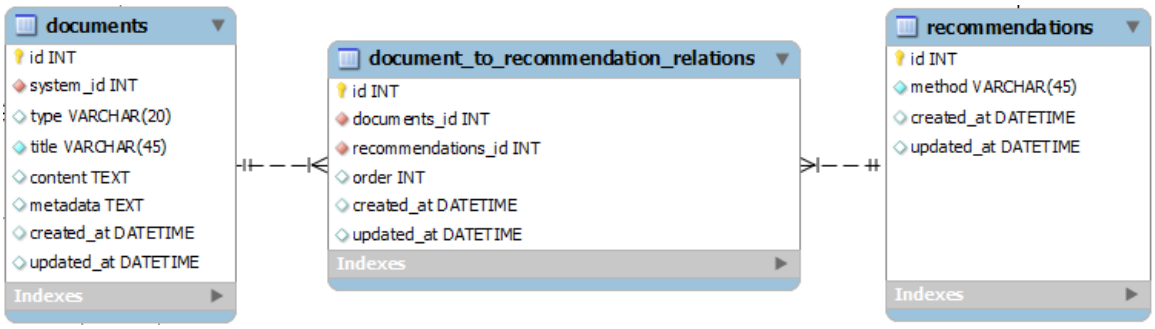
user_to_document_relations – Relačná entita na obr. D.8 zachytáva práve vzťah hodnotenia

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.



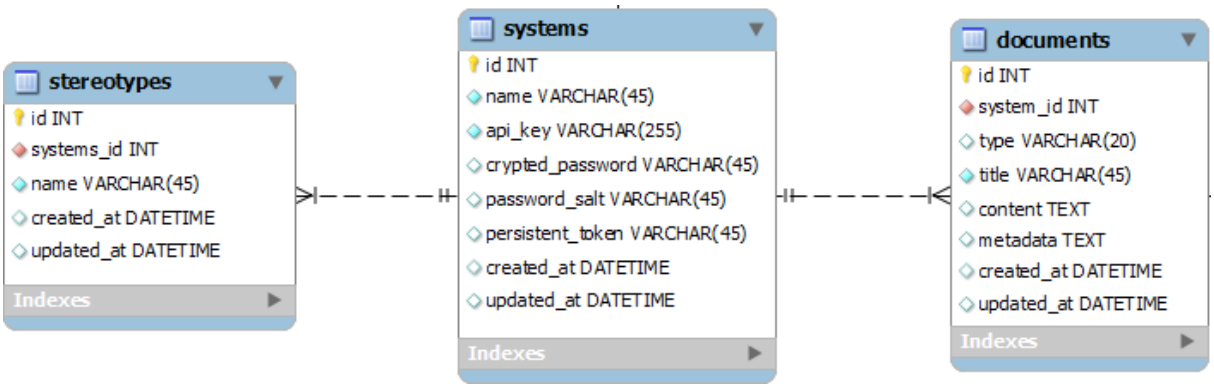
Obrázok D.8: Relačná entita medzi documents a users

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 knižnice.

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.

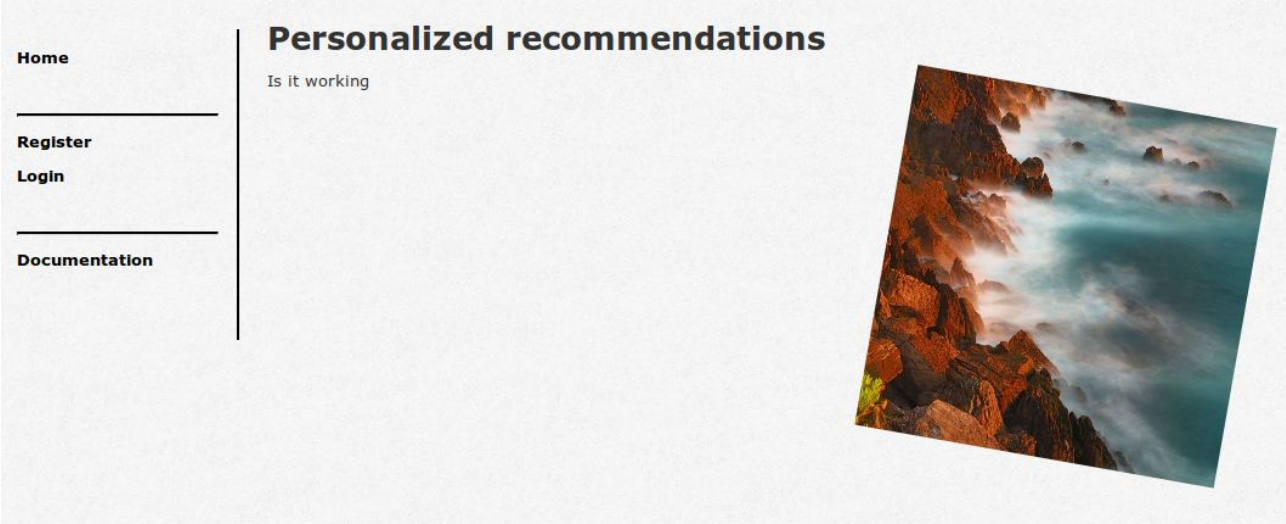
Grafické prostredie slúži na interaktívne manažovanie systému využívajúceho službu na odporúčanie. Api slúži na volanie funkcií služby automaticky priamo z konzumenta odporúčania.

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).

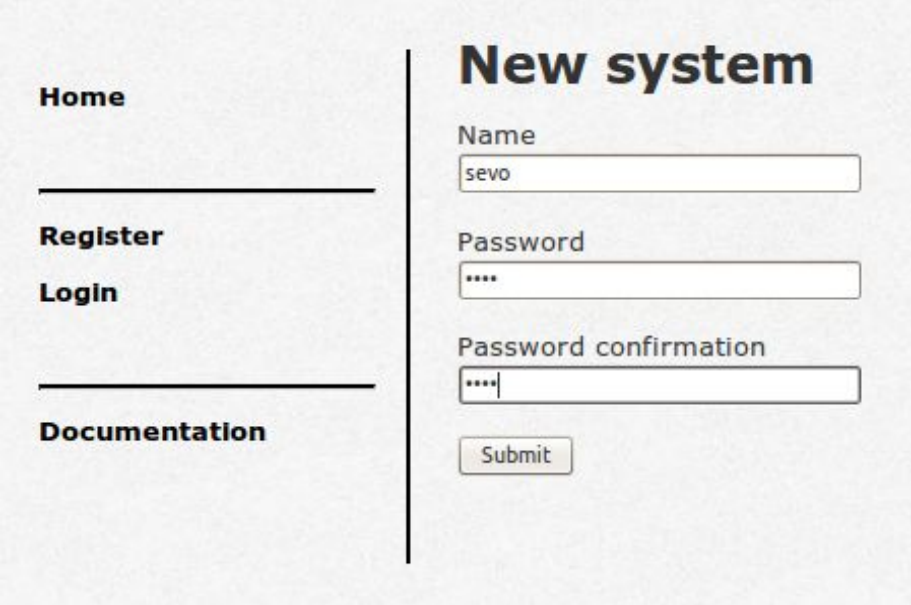


Obrázok E.1: Úvodná obrazovka aplikácie

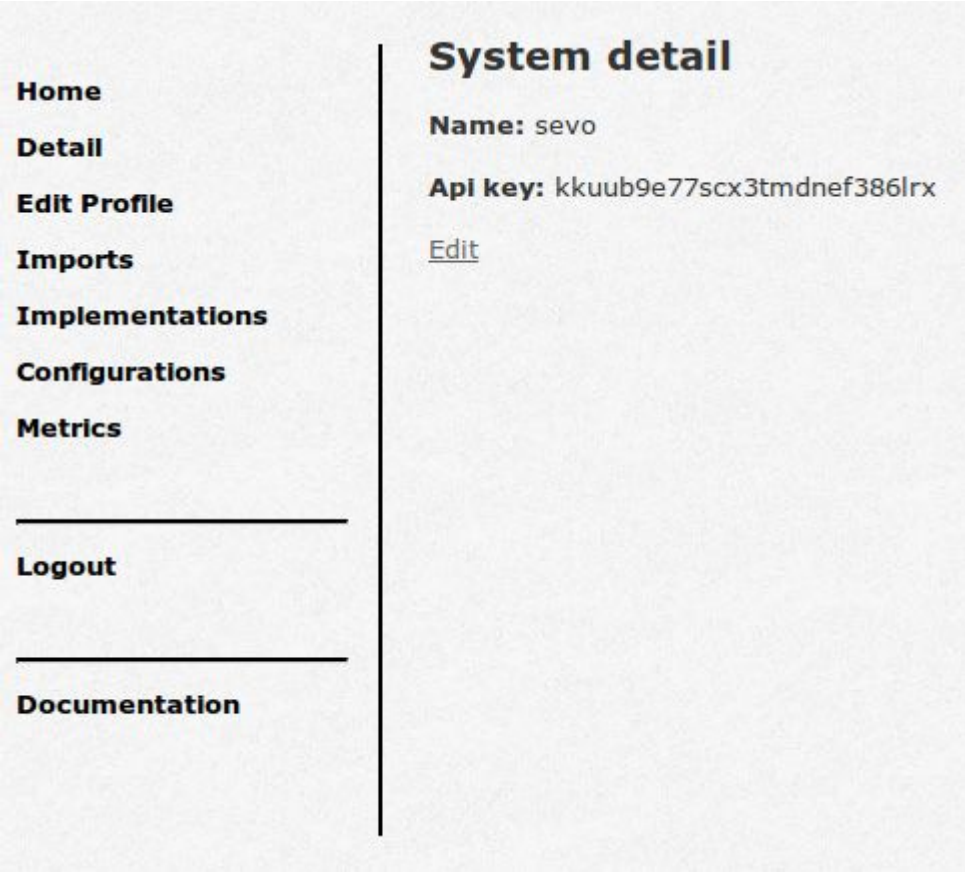
Na tejto obrazovke sú v ľavom menu dostupné možnosti na prihlásenie, registráciu a na zobrazenie dokumentácie k používaniu API funkcií.

Registrácia systému

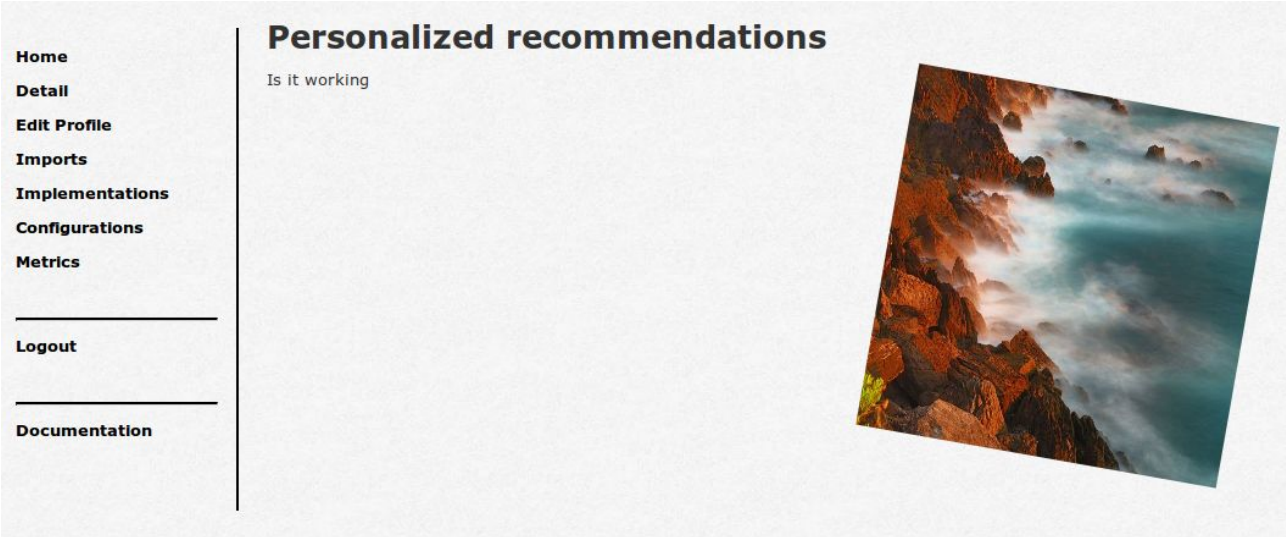
Pred začatím práce s aplikáciou sa systém, ktorý chce získavať odporúčanie pomocou tejto aplikácie, registrovať. Po kliknutí na odkaz *„Register“* na úvodnej obrazovke sa zobrazí registračný formulár (Obrázok E.2). Po vyplnení potrebných údajov a odoslaní formulára sa zobrazí detail nového systému aj 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



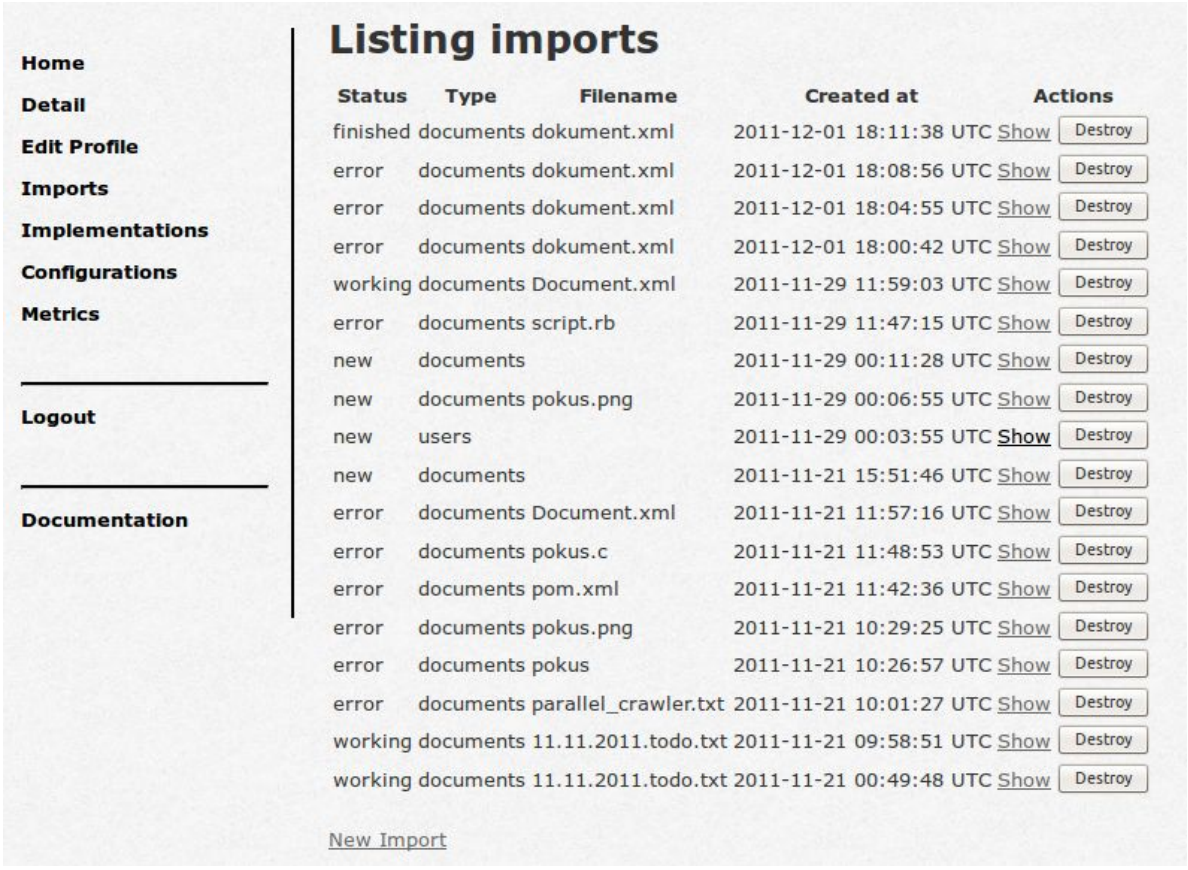
Obrázok E.3: Detail systému spolu s vygenerovaným apikľúčom



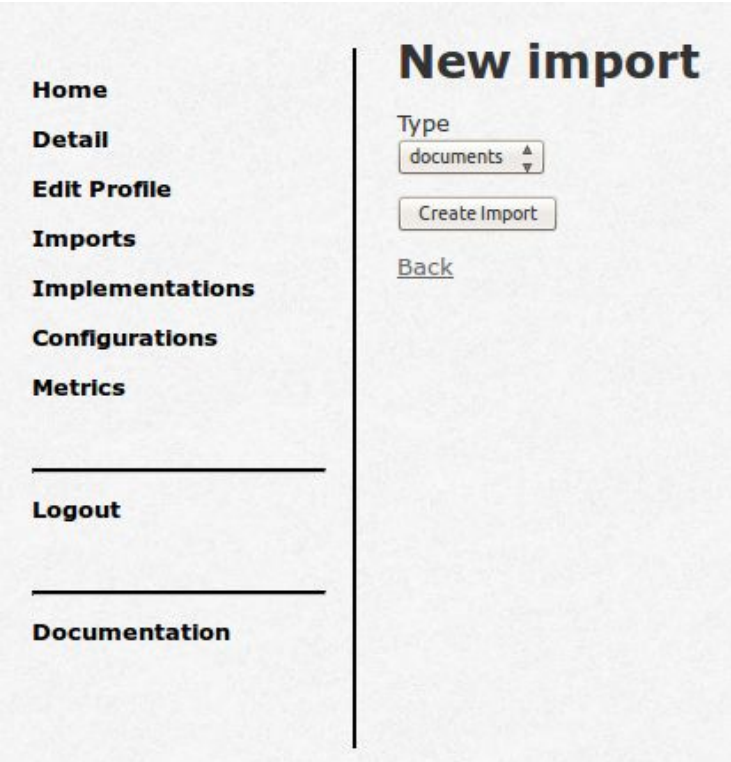
Obrázok E.4: Úvodná stránka registrovaného systému

Importovanie údajov

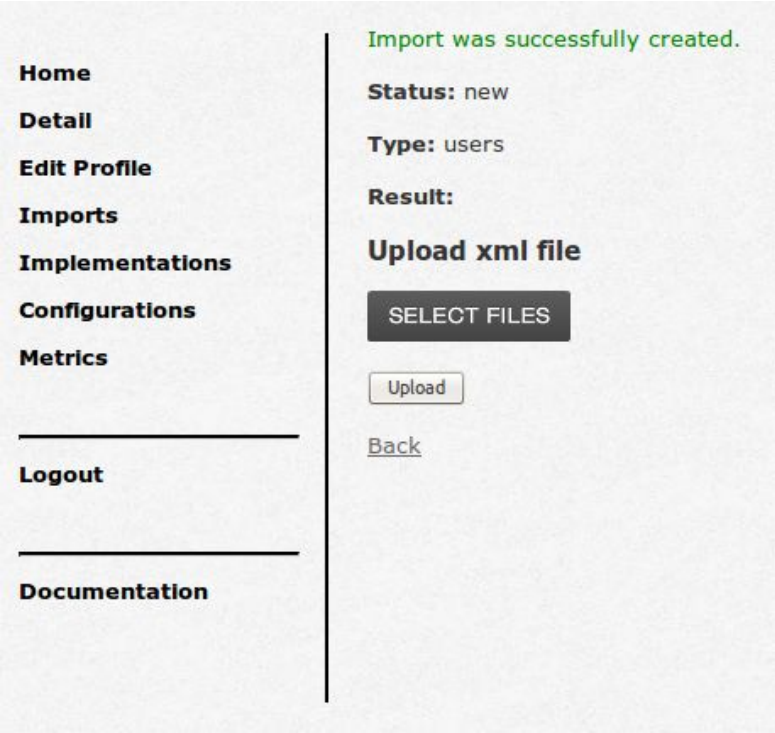
Ak sme prihlásený ako systém, tak sa v ľavom menu nachádza odkaz *„Imports“*. Po kliknutí na tento odkaz sa zobrazia všetky doteraz vykonané importy xml súborov (Obrázok E.5). Po kliknutí na odkaz *„New Import“* sa zobrazí okno na špecifikovanie nového importu (Obrázok E.6). V tomto okne vyberieme typ 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.



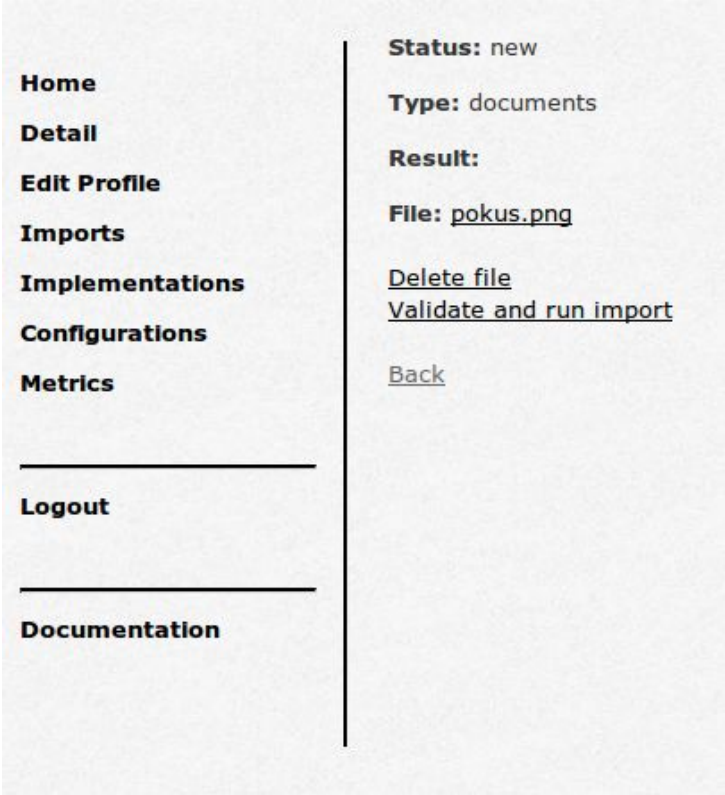
Obrázok E.5: Zoznam všetkých vykonaných importov



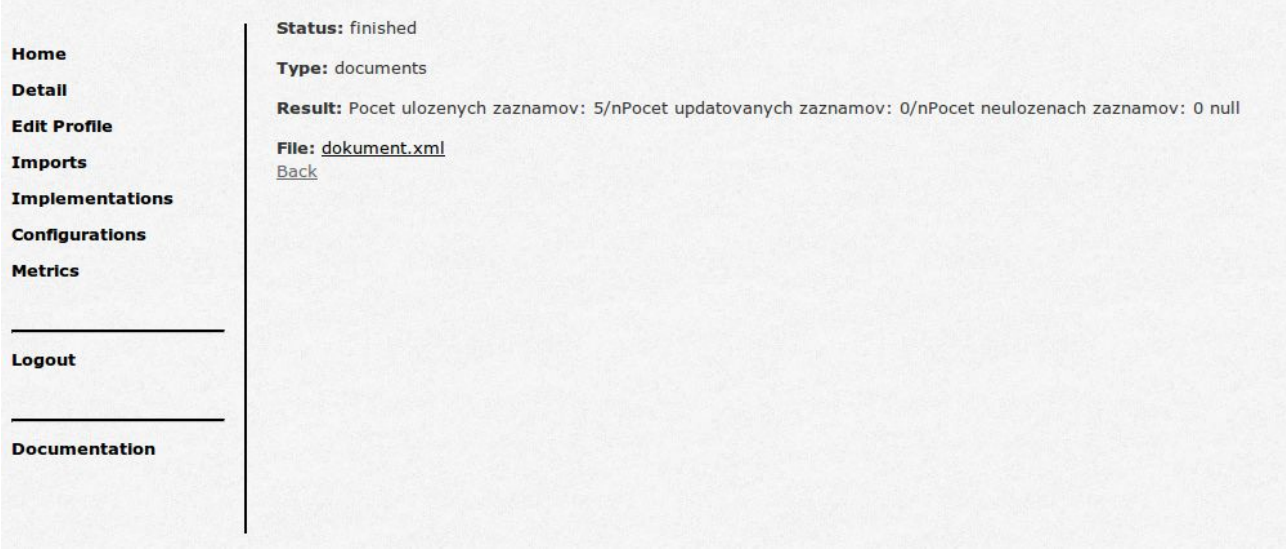
Obrázok E.6: Okno na špecifikovanie nového importu



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



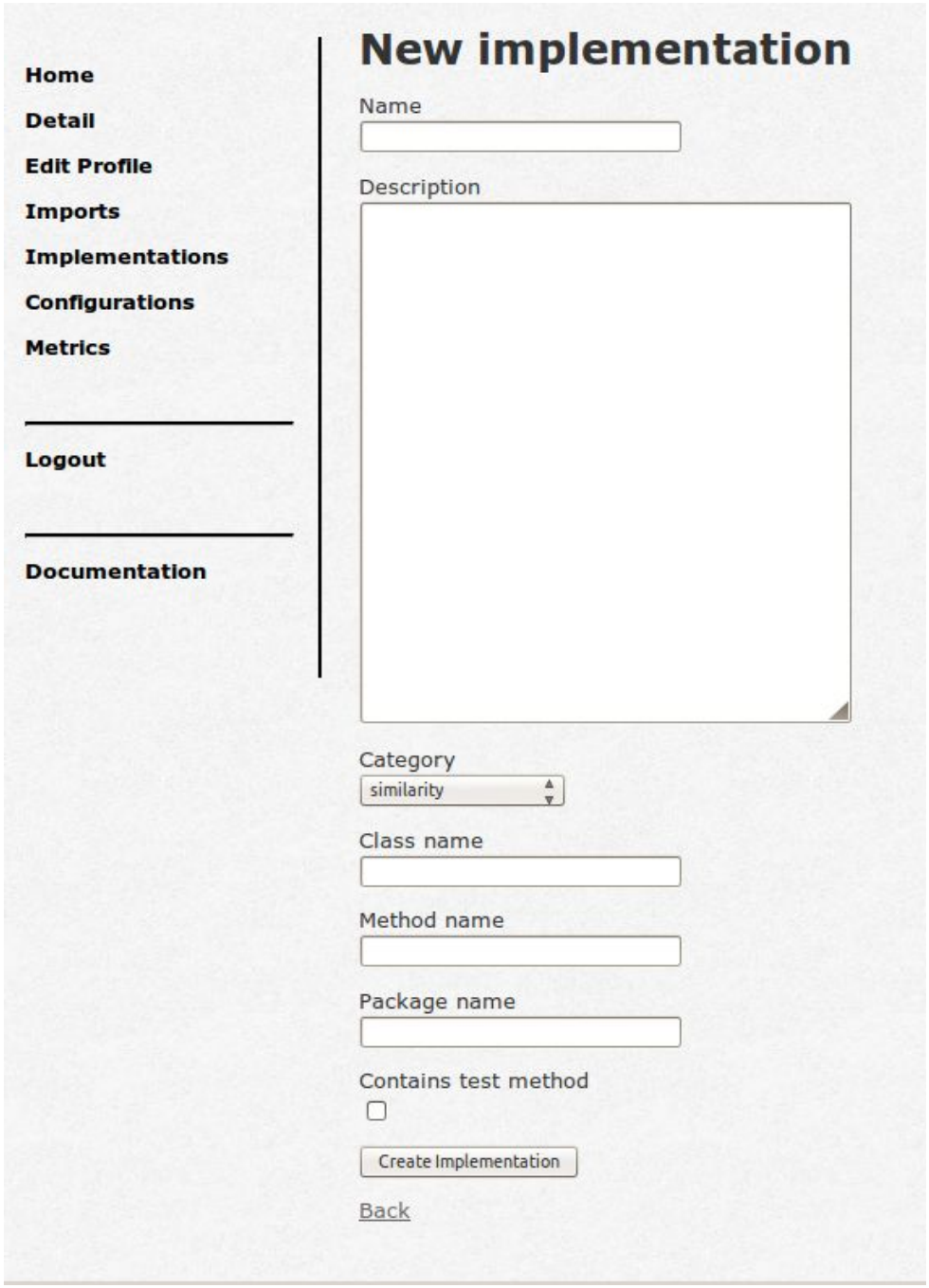
Obrázok E.8: Po nahraní súboru do importu máme možnosť spustiť import



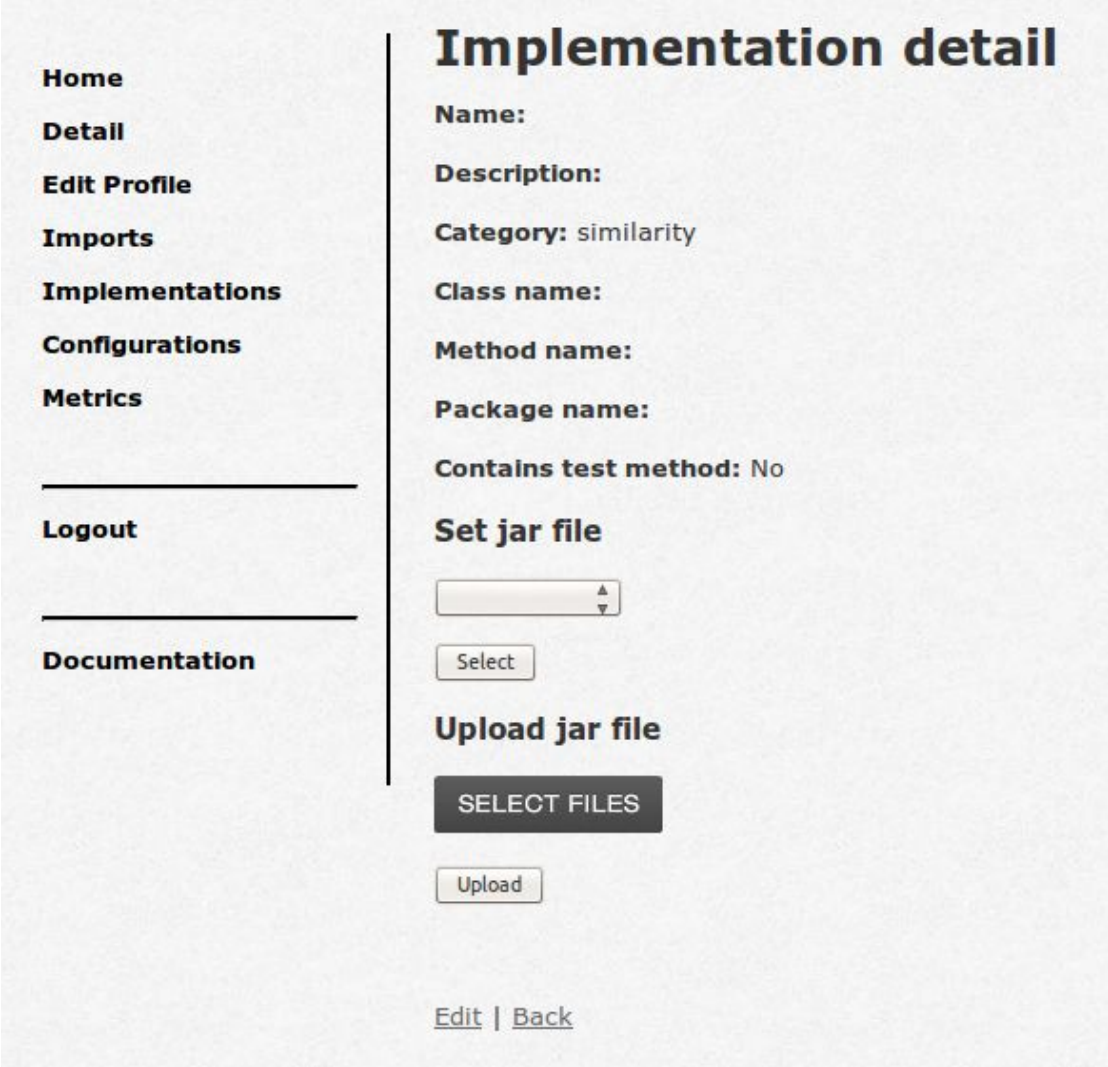
Obrázok E.9: Zobrazenie výsledku importu po úspešnom prebehnutí importovania

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.



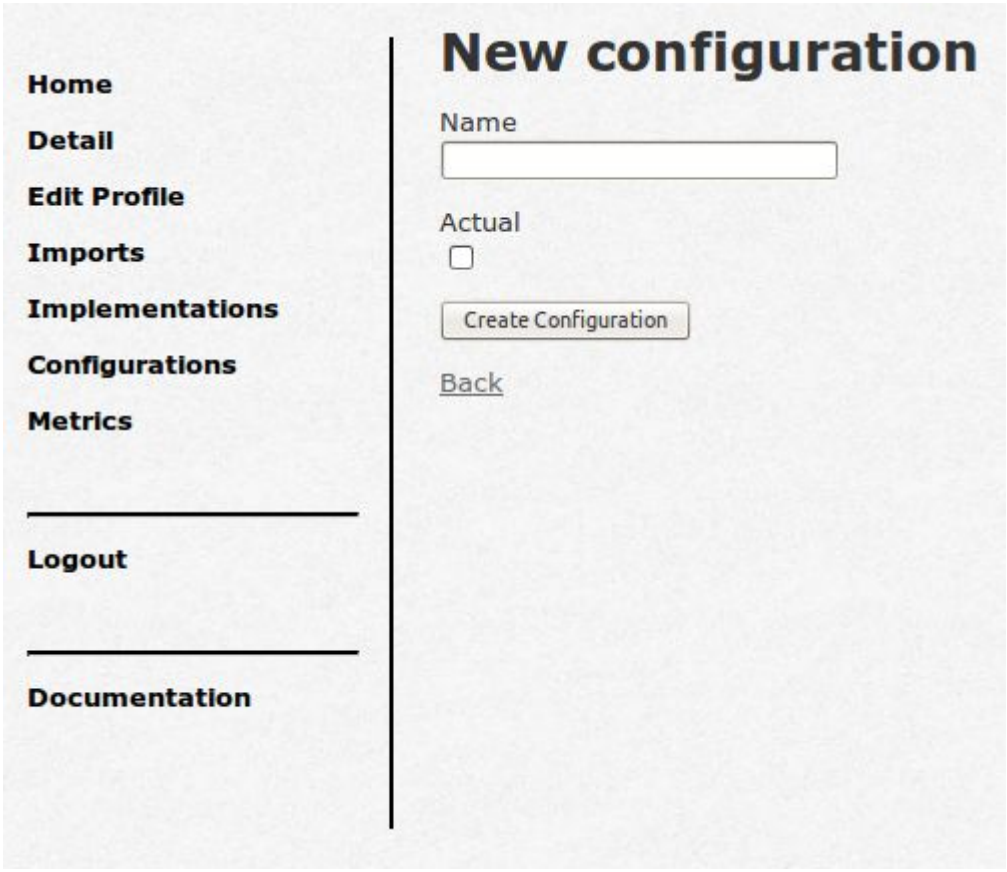
Obrázok E.10: Okno na nastavenie detailov novej implementácie



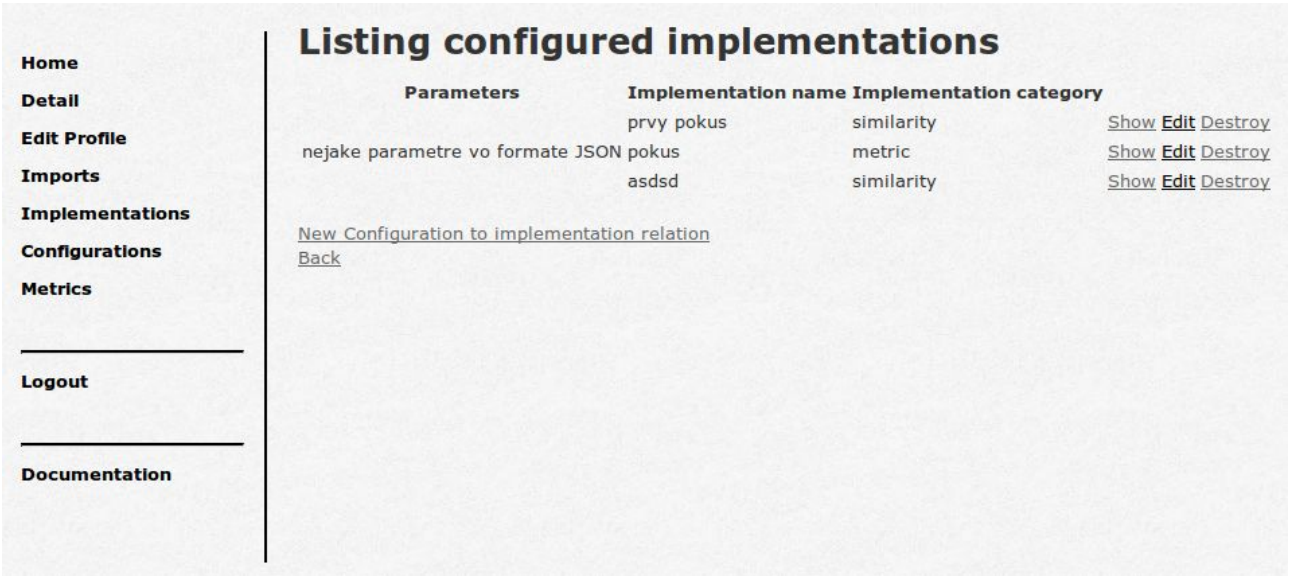
Obrázok E.11: Detail implementácie spolu s možnosťou pripojiť alebo nahrať .jar súbor

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.

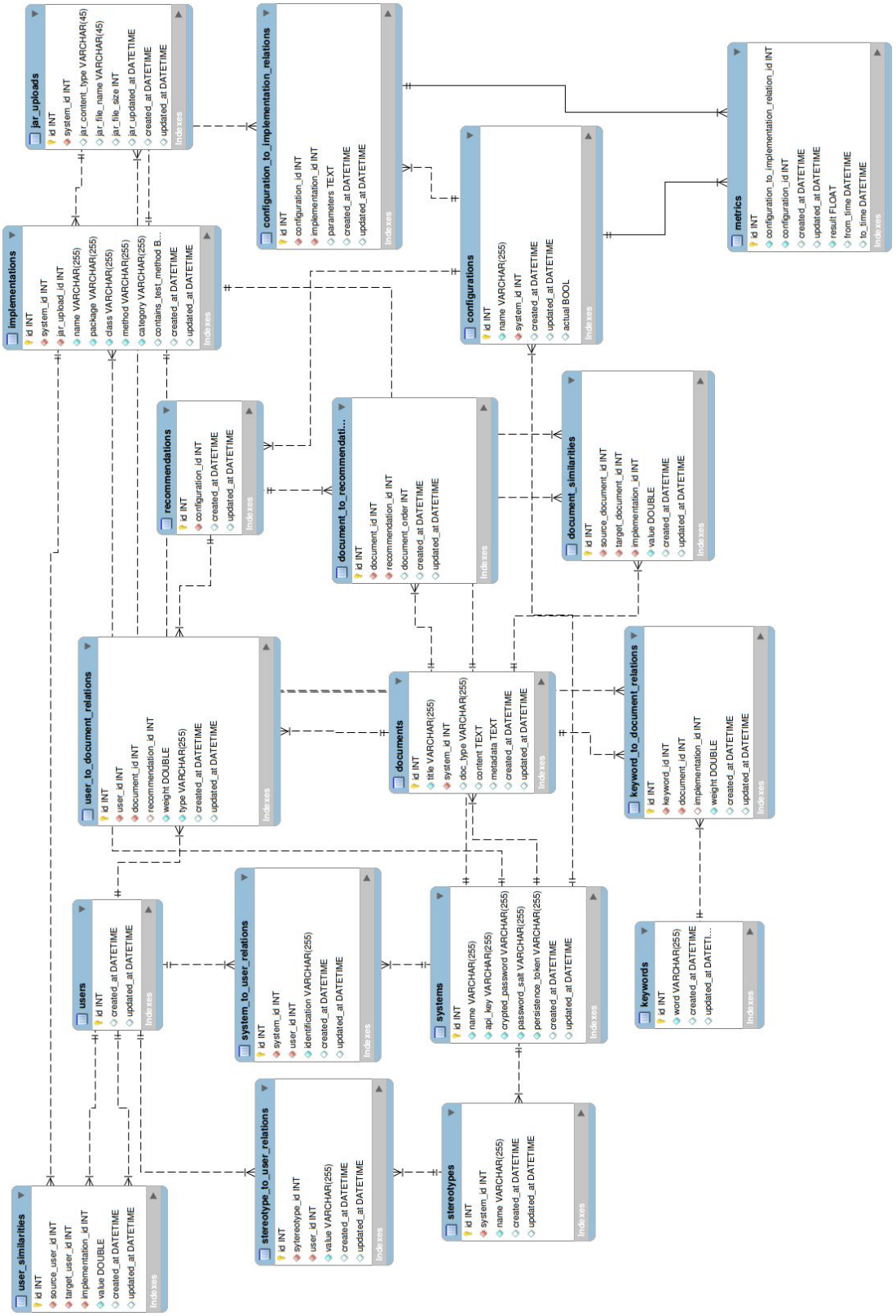
Výsledok volania je xml súbor obsahujúci zoznam odporučených dokumentov:

```
<?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

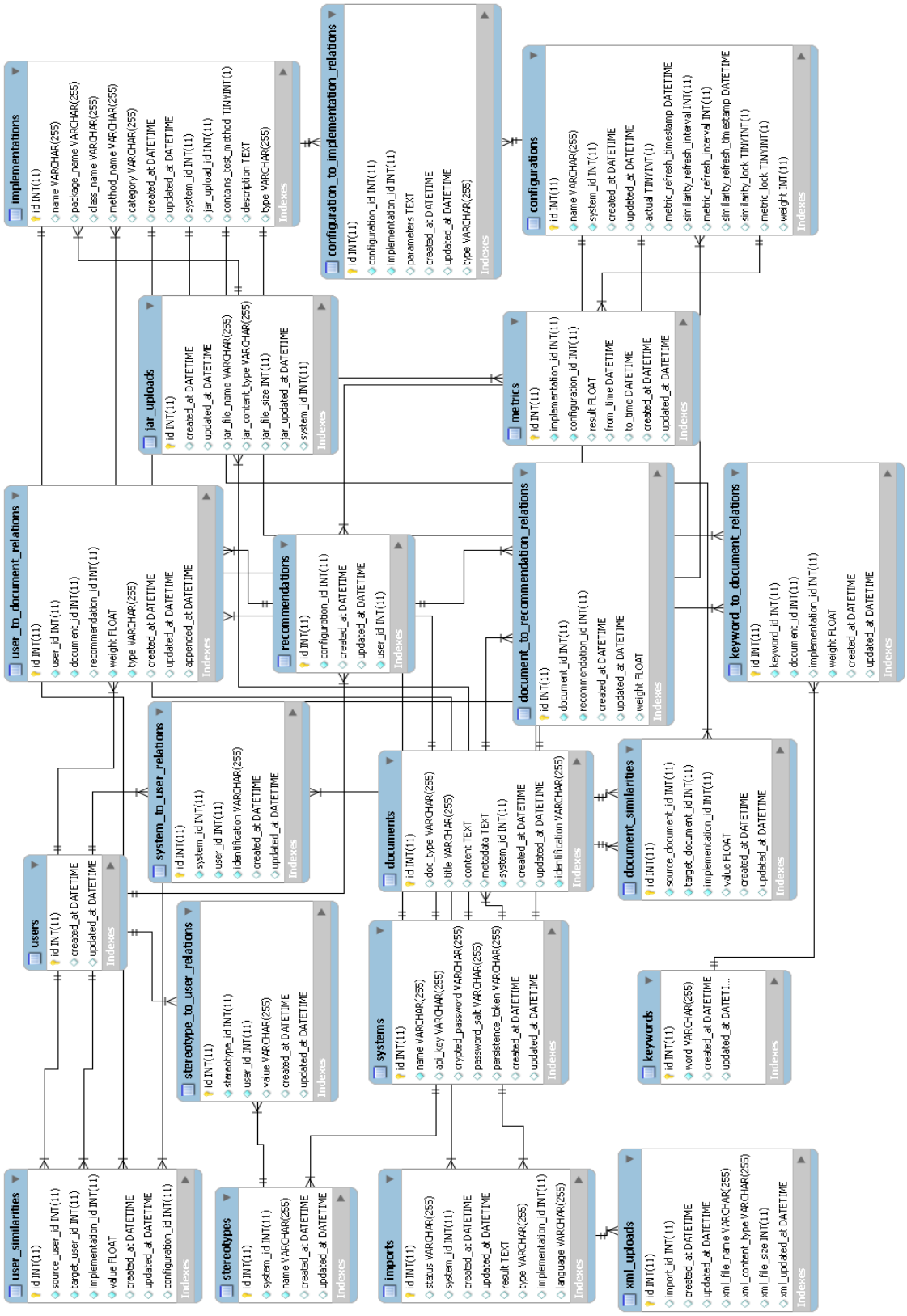


Obrázok F.1: Dátový model po prvom semestri

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.

Príloha G - Dátový model po druhom semestri



Obrázok G.1: Dátový model po druhom semestri

Použité skratky

K – kľúč, P – primárny, F – cudzí

NN – nemôže byť nulové

U – unikátna hodnota pre každý záznam

Tabuľka systems

Obsahuje zaregistrované externé systémy.

Atribút	Dátový typ	K	NN	U	Opis
id	int(11)	P	x		
name	varchar(255)		x	x	Názov systému
api_key	varchar(255)			x	Kľúč, ktorý používa systém na identifikáciu pri používaní API volaní
crypted_password	varchar(255)				Šifrované heslo pre prihlásenie systému do aplikácie
password_salt	varchar(255)				Salt na doplnenie šifrovaného hesla, znemožňuje slovníkový útok pri lámaní hesla.
persistence_token	varchar(255)				Pre potreby bezpečného prihlásenia
created_at	datetime				Čas a dátum vytvorenia záznamu
updated_at	datetime				Čas a dátum zmeny záznamu

Tabuľka users

Obsahuje konzumentov externých systémov, kde jeden človek môže navštevovať viac systémov.

Atribút	Dátový typ	K	NN	U	Opis
id	int(11)	P	x		
created_at	datetime				Čas a dátum vytvorenia záznamu
updated_at	datetime				Čas a dátum zmeny záznamu

Tabuľka system_to_user_relations

Obsahuje väzbu medzi konzumentami a systémami.

Atribút	Dátový typ	K	NN	U	Opis
id	int(11)	P	x		
system_id	int(11)	F	x		Id systému z tabuľky systems
user_id	int(11)	F	x		Id konzumenta z tabuľky users
identification	varchar(255)		x		Identifikácia konzumenta v externom systéme
created_at	datetime				Čas a dátum vytvorenia záznamu
updated_at	datetime				Čas a dátum zmeny záznamu

Ďalšie unikátne indexy:

unique_user_in_system: na atribútoch *system_id* a *identification*

Tabuľka stereotypes

Obsahuje názvy stereotypov pre konkrétny externý systém.

Atribút	Dátový typ	K	NN	U	Opis
id	int(11)	P	x		
system_id	int(11)	F	x		Id systému z tabuľky systems
name	varchar(255)		x		Názov stereotypu pre konkrétny externý systém
created_at	datetime				Čas a dátum vytvorenie záznamu
updated_at	datetime				Čas a dátum zmeny záznamu

Tabuľka stereotype_to_user_relations

Obsahuje ohodnotenie konzumenta pre daný stereotyp.

Atribút	Dátový typ	K	NN	U	Opis
id	int(11)	P	x		
stereotype_id	int(11)	F	x		Id stereotypu z tabuľky stereotypes
user_id	int(11)	F	x		Id konzumenta z tabuľky users
value	varchar(255)				Slovné ohodnotenie konzumenta
created_at	datetime				Čas a dátum vytvorenia záznamu
updated_at	datetime				Čas a dátum zmeny záznamu

Tabuľka documents

Obsahuje dokumenty z externých systémov, ktoré môžeme odporúčať konzumentom.

Atribút	Dátový typ	K	NN	U	Opis
id	int(11)	P	x		
system_id	int(11)	F	x		Id systému z tabuľky systems
identification	varchar(255)		x		Identifikácia dokumentu v externom systéme
title	varchar(255)				Názov dokumentu
doc_type	varchar(255)				Typ dokumentu
content	text				Obsah dokumentu
metadata	text				Metadáta na opis dokumentu
created_at	datetime				Čas a dátum vytvorenia záznamu
updated_at	datetime				Čas a dátum zmeny záznamu

Ďalšie unikátne indexy:

uniq_identification_in_system: na atribútoch *system_id* a *identification*

Tabuľka keywords

Obsahuje kľúčové slová, podľa ktorých sa počíta podobnosť dokumentov.

Atribút	Dátový typ	K	NN	U	Opis
id	int(11)	P	x		
word	varchar(255)		x		Názov kľúčového slova
created_at	datetime				Čas a dátum vytvorenia záznamu
updated_at	datetime				Čas a dátum zmeny záznamu

Tabuľka keyword_to_document_relations.

Obsahuje väzbu kľúčového slova na dokument a hodnotu tejto väzby.

Atribút	Dátový typ	K	NN	U	Opis
id	int(11)	P	x		
keyword_id	int(11)	F	x		Id kľúčového slova z tabuľky keywords
document_id	int(11)	F	x		Id dokumentu z tabuľky documents
implementation_id	int(11)	F			Id implementácie, ktorou bola váha vypočítaná
weight	float		x		Váha akú má kľúčové slovo v dokumente
created_at	datetime				Čas a dátum vytvorenia záznamu
updated_at	datetime				Čas a dátum zmeny záznamu

Tabuľka jar_uploads

Obsahuje zoznam všetkých importovaných jar súborov.

Atribút	Dátový typ	K	NN	U	Opis
id	int(11)	P	x		
jar_file_name	varchar(11)				Názov súboru
jar_content_type	varchar(11)				Typ uloženého súboru
jar_file_size	int(11)				Veľkosť súboru
jar_updated_at	datetime				Dátum kedy bol naposledy zmenený nahraný súbor
created_at	datetime				Čas a dátum vytvorenia záznamu
updated_at	datetime				Čas a dátum zmeny záznamu

Tabuľka implementations

Obsahuje zoznam všetkých implementácií prídavných pluginov (kombinovače, odporúčače, počítanie podobností, počítanie kľúčových slov, metriky).

Atribút	Dátový typ	K	NN	U	Opis
id	int(11)	P	x		
system_id	int(11)	F			Id systému z tabuľky systems
jar_upload_id	int(11)	F			Id jar súboru v tabuľke jar_uploads
name	varchar(255)				Názov implementácie
package_name	varchar(255)				Názov packagu
class_name	varchar(255)				Názov triedy
method_name	varchar(255)				Názov metódy
type	varchar(255)				Typ implementácie (kombinovač/odporúčač/...)
category	varchar(255)				Kategória odporúčačov a počítania podobností (content/colaborativ a users/documents)
description	text				Opis implementácie
contains_test_method	tinyint(1)				Informácia o tom či implementácia obsahuje testovaciu metódu na otestovanie jej funkčnosti
created_at	datetime				Čas a dátum vytvorenia záznamu
updated_at	datetime				Čas a dátum zmeny záznamu

Tabuľka configurations

Obsahuje konfigurácie, na základe ktorej možno odporúčať.

Atribút	Dátový typ	K	NN	U	Opis
id	int(11)	P	x		
system_id	int(11)	F	x		Id systému z tabuľky systems
name	varchar(255)		x	x	Názov konfigurácie
actual	tinyint(1)				Označuje, či je možné použiť konfiguráciu na odporúčanie
metric_refresh_timestamp	datetime				Časová pečiatka kedy boli naposledy prepočítané metriky pre danú konfiguráciu
metric_refresh_interval	int(11)				Interval v sekundách ako často sa majú prepočítavať metriky
similarity_refresh_timestamp	datetime				Časová pečiatka kedy boli naposledy prepočítané podobnosti pre danú konfiguráciu
similarity_refresh_interval	int(11)				Interval v sekundách ako často sa majú prepočítavať podobnosti
similarity_lock	tinyint(1)				Uzamknutie konfigurácie pri prepočítavaní podobností. Umožňuje paralelné prepočítavanie podobností viacerými procesmi.
metric_lock	tinyint(1)				Uzamknutie konfigurácie pri prepočítavaní metrík.

					Umožňuje paralelné prepočítavanie metrík viacerými procesmi.
weight	int(11)				Označuje podiel konzumentov, ktorý pri AB testovaní dostanú odporúčanie práve od tejto konfigurácie
created_at	datetime				Čas a dátum vytvorenia záznamu
updated_at	datetime				Čas a dátum zmeny záznamu

Tabuľka configuration_to_implementation_relations

Obsahuje nastavenie implementácie v danej konfigurácií.

Atribút	Dátový typ	K	NN	U	Opis
id	int(11)	P	x		
configuration_id	int(11)	F	x		Id konfigurácie v tabuľke configurations
implementation_id	int(11)	F	x		Id implementácie v tabuľke implementations
parameters	text				Nastavenia uložené ako json
type	varchar(255)				Typ implementácie (odporúčač, kombinovač, ...)
created_at	datetime				Čas a dátum vytvorenia záznamu
updated_at	datetime				Čas a dátum zmeny záznamu

Tabuľka imports

Obsahuje údaje o stave importu.

Atribút	Dátový typ	K	NN	U	Opis
id	int(11)	P	x		
system_id	int(11)	F			Id systému v tabuľke systems
implementation_id	int(11)	F			Id implementácie v tabuľke implementations
status	varchar(255)				Stav importu (new, working, finished, error)
result	text				Krátky sumár spracovania, prípadne chyby, ktorá nastala
type	varchar(255)				Typ importu (users, documents, relations, docsCountKeyword)
language	varchar(255)				Jazyk v ktorom sú importované dokumenty
created_at	datetime				Čas a dátum vytvorenia záznamu
updated_at	datetime				Čas a dátum zmeny záznamu

Tabuľka xml_uploads

Obsahuje informácie o importovanom XML súbore.

Atribút	Dátový typ	K	NN	U	Opis
id	int(11)	P	x		
import_id	int(11)	F	x		Id importu v tabuľke imports
xml_file_name	varchar(255)				Názov XML súboru
xml_content_type	varchar(255)				Typ nahraného súboru
xml_file_size	int(11)				Veľkosť XML súboru
xml_updated_at	datetime				Čas, kedy bol nahraný súbor naposledy upravený
created_at	datetime				Čas a dátum vytvorenia záznamu
updated_at	datetime				Čas a dátum zmeny záznamu

Tabuľka document_similarities

Obsahuje mieru podobnosti dvoch dokumentov.

Atribút	Dátový typ	K	NN	U	Opis
id	int(11)	P	x		
source_document_id	int(11)	F	x		Id prvého dokumentu
target_document_id	int(11)	F	x		Id druhého dokumentu
implementation_id	int(11)	F	x		Id implementácie, ktorou bola hodnota vypočítaná
value	float				Hodnota ako sa dokumenty podobajú
created_at	datetime				Čas a dátum vytvorenia záznamu
updated_at	datetime				Čas a dátum zmeny záznamu

Tabuľka user_similarities

Obsahuje mieru podobnosti dvoch konzumentov.

Atribút	Dátový typ	K	NN	U	Opis
id	int(11)	P	x		
source_user_id	int(11)	F	x		Id prvého dokumentu
target_user_id	int(11)	F	x		Id druhého dokumentu
implementation_id	int(11)	F	x		Id implementácie, ktorou bola hodnota vypočítaná
configuration_id	int(11)	F			Id konfigurácie
value	float				Hodnota ako sa návštevníci podobajú
created_at	datetime				Čas a dátum vytvorenia záznamu
updated_at	datetime				Čas a dátum zmeny záznamu

Tabuľka metrics

Obsahuje hodnoty vypočítané metrikami hovoriace o úspešnosti odporúčania.

Atribút	Dátový typ	K	NN	U	Opis
id	int(11)	P	x		
implementation_id	int(11)	F	x		Id implementácie, ktorou bola hodnota vypočítaná
configuration_id	int(11)	F	x		Id konfigurácie, pre ktorú počítame úspešnosť
result	float				Výsledok ako bola metrika úspešná
from_time	datetime				Od kedy sa úspešnosť má počítať
to_time	datetime				Do kedy sa úspešnosť má počítať
created_at	datetime				Čas a dátum vytvorenia záznamu
updated_at	datetime				Čas a dátum zmeny záznamu

Tabuľka recommendations

Obsahuje odporúčania odporučené konzumentom.

Atribút	Dátový typ	K	NN	U	Opis
id	int(11)	P	x		
configuration_id	int(11)	F	x		Id konfigurácie, ktorou sme odporúčali
user_id	int(11)	F	x		Id konzumenta, ktorému sme odporúčali
created_at	datetime				Čas a dátum vytvorenia záznamu
updated_at	datetime				Čas a dátum zmeny záznamu

Tabuľka document_to_recommendation_relations

Obsahuje väzbu medzi dokumentom a odporúčaním s váhou, akou sme daný dokument odporučili.

Atribút	Dátový typ	K	NN	U	Opis
id	int(11)	P	x		
document_id	int(11)	F	x		Id dokumntu, ktorý sme odporúčali
recommendation_id	int(11)	F	x		Id odporúčania, v ktorom sme odporúčali
weight	float				Váha, ako sme dokument odporučili
created_at	datetime				Čas a dátum vytvorenia záznamu
updated_at	datetime				Čas a dátum zmeny záznamu

Príloha H - Používateľská príručka po druhom semestri

Obsah

Grafické používateľské rozhranie

Úvodná obrazovka

Aplikácia je dostupná na adrese:

http://team12-11.ucebne.fiit.stuba.sk/app

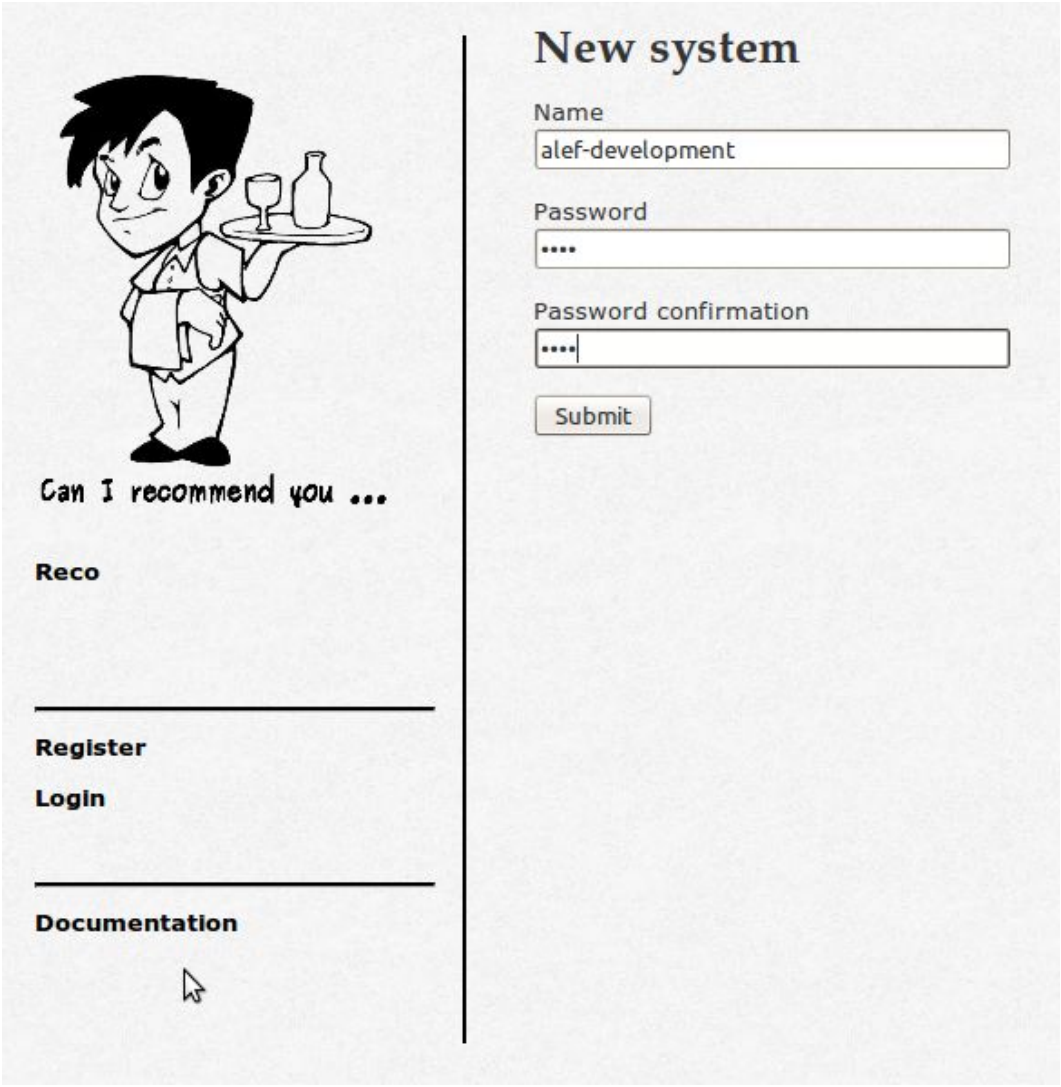
Po zobrazení tejto stránky sa nachádzame ako neprihlásený systém na úvodnej obrazovke (Obrázok H.1). Obrazovka má dve hlavné časti: Menu v ľavej časti (ovál 1) a hlavný obsah v pravej časti obrazovky (ovál 2). V menu má používateľ možnosť zaregistrovať nový systém kliknutím na voľbu „*Register*“. V prípade, že už systém registrovaný je, má používateľ možnosť prihlásiť sa kliknutím na voľbu „*Login*“. Kliknutím na voľbu „*Documentation*“ v ľavom menu (ovál 4) sa zobrazí popis ako je potrebné volať api funkcie a aké sú ich návratové hodnoty.

Obrázok H.1: Úvodná obrazovka pre neprihláseného používateľa

Registrácia systému

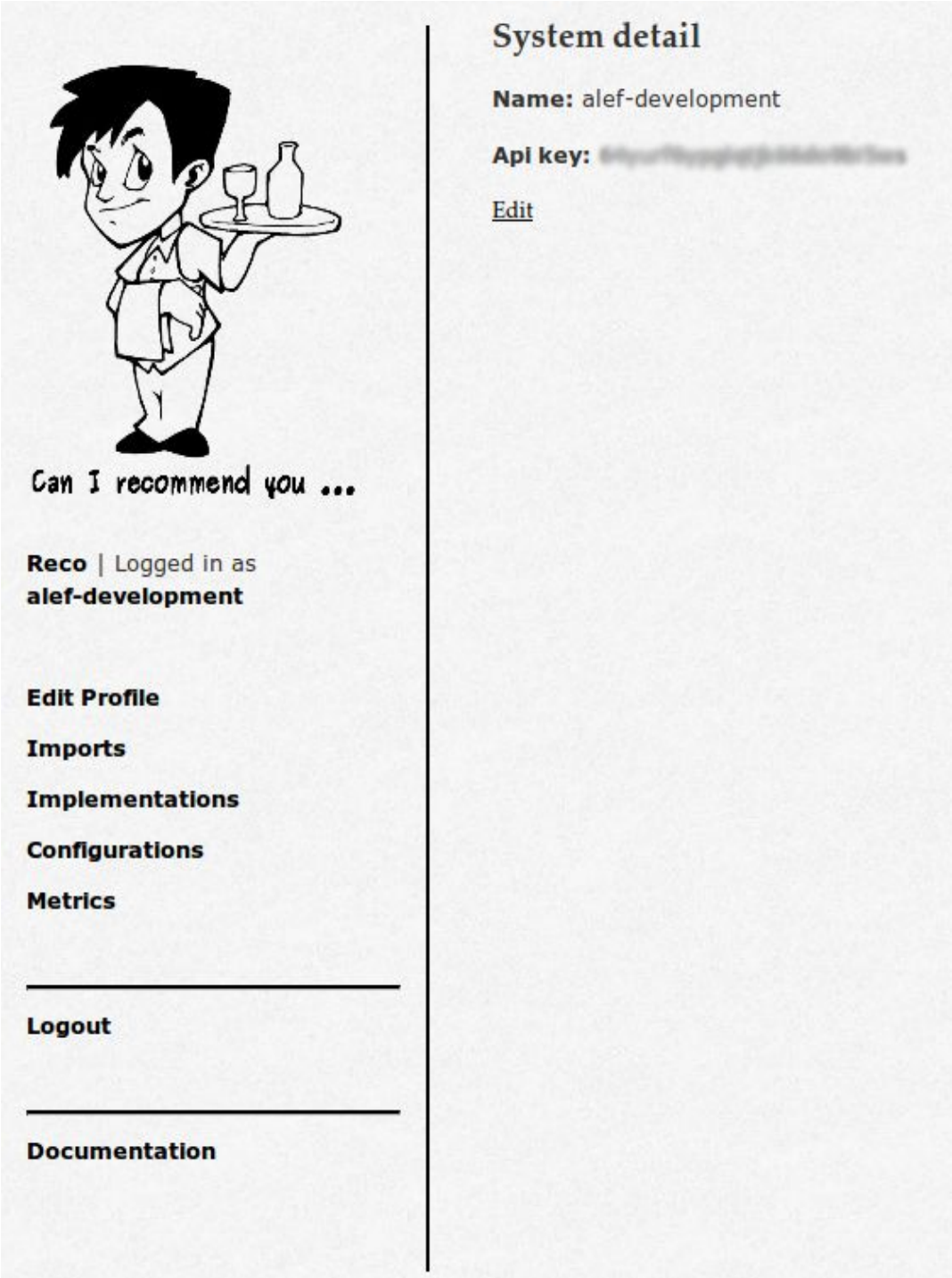
V tejto časti opíšeme postup pri registrácií nového systému, ktorý by chcel používať aplikáciu na získavanie odporúčania.

Po zadaní adresy *http://team12-11.ucebne.fiit.stuba.sk/app* do internetového prehliadača sa zobrazí úvodná stránka zobrazená na obrázku H.1. Kliknutím na voľbu „*Register*“ sa zobrazí obrazovka, kde je potrebné vyplniť prístupové údaje pre nový systém. Táto obrazovka je zobrazená na obrázku H.2. Po správnom vyplnení všetkých zobrazených polí a kliknutí na tlačidlo „*Submit*“ sa vytvorí nový systém, a zobrazí sa detail tohto systému spolu s vygenerovaným api kľúčom (Obrázok H.3). V tejto chvíli je systém prihlásený a má prístup k ďalším funkciám. Vygenerovaný api kľúč slúži na identifikáciu systému pri používaní api funkcií na získavanie opdorúčania, posielanie spätnej väzby a pod.

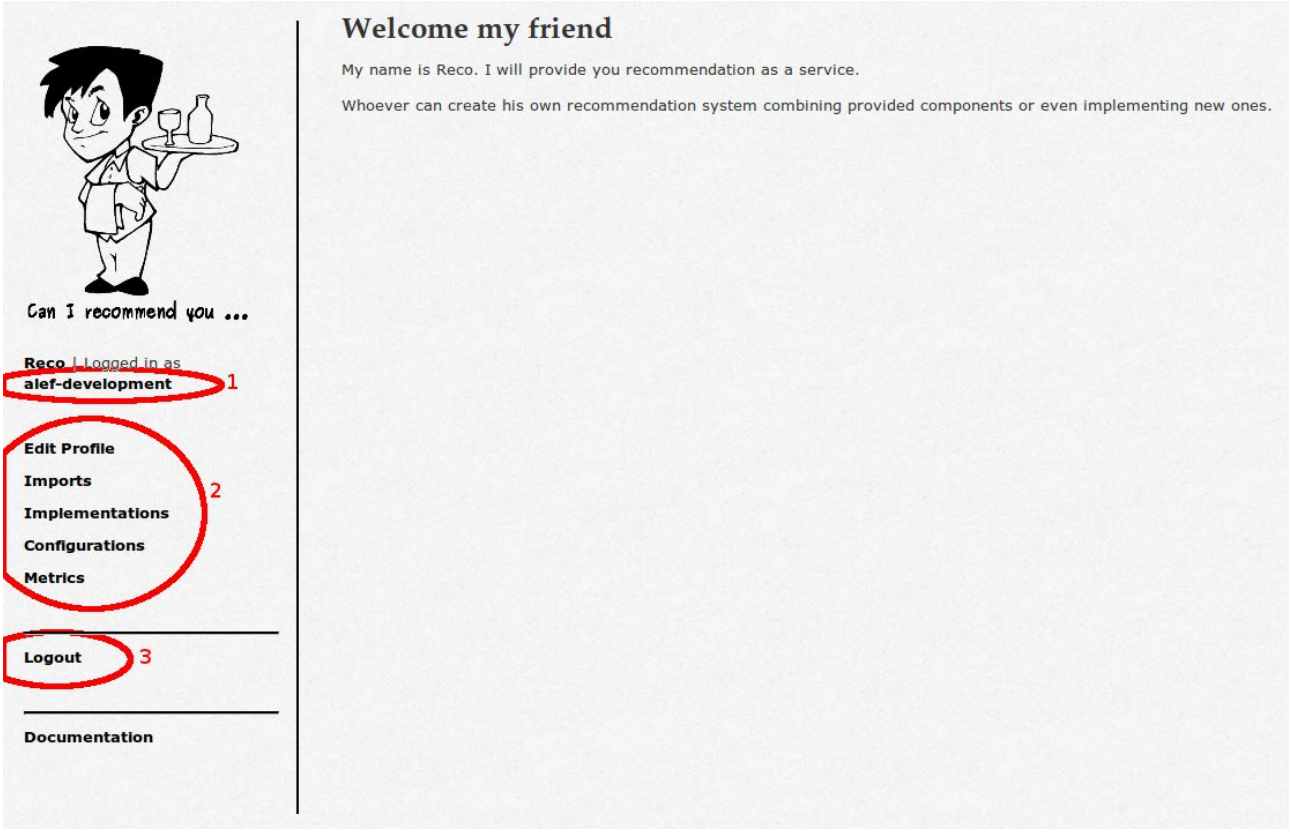


Obrázok H.2: Obrazovka na vyplnenie prístupových údajov systému pri jeho registrácii

Zmenené ľavé menu pre prihlásený systém je možné vidieť na obrázku H.4. Zostala tu možnosť zobraziť dokumentáciu pre volanie API funkcií, ale pribudli ďalšie funkcie systému. Pribudla možnosť zobraziť detail prihláseného systému (ovál 1). Pribudol celý zoznam volieb (ovál 2), ktoré slúžia na konfigurovanie odporúčačov, pridávanie vlastných implementácií algoritmov, importovanie údajov a zobrazovanie výsledkov pre overenie úspešnosti odporúčania. Poslednou možnosťou, ktorá pribudla v ľavom menu je možnosť pre odhlásenie systému (ovál 3).



Obrázok H.3: Detail prihláseného systému



Obrázok H.4: Hlavná obrazovka prihláseného systému

Vytvorenie konfigurácie

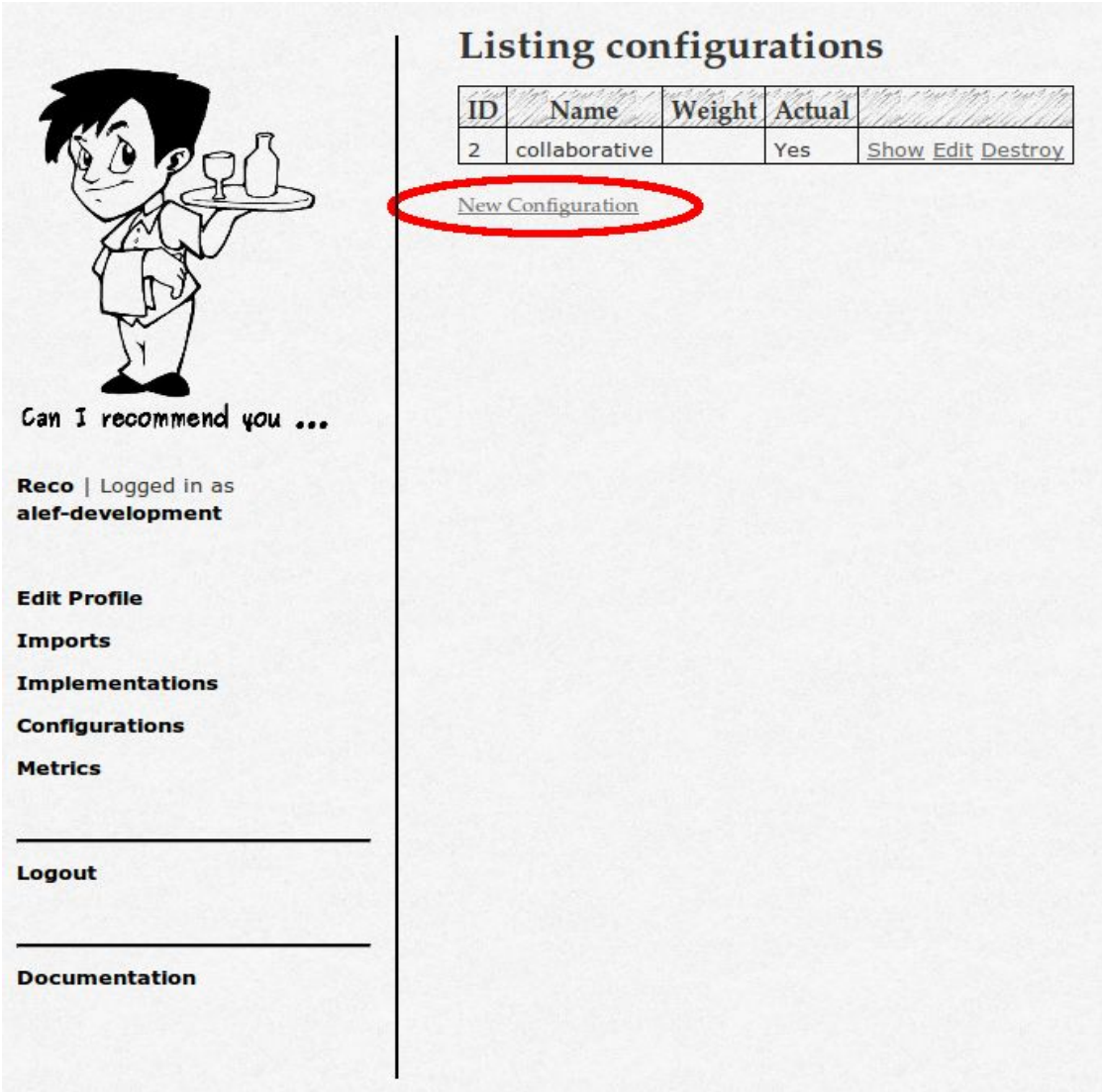
Na to, aby vedel systém získavať odporúčania, musí si najskôr nakonfigurovať vlastný odporúčač. Konfigurácia prebieha spájaním a konfigurovaním jednotlivých častí odporúčača. Vytváranie konfigurácie sa začína kliknutím na voľbu „*Configurations*“ v ľavom menu. Následne sa zobrazí zoznam všetkých konfigurácií, ktoré má používateľ zatiaľ vytvorené (Obrázok H.5). Pod spomínaným zoznamom konfigurácií sa nachádza možnosť pridať novú konfiguráciu. Po kliknutí na túto možnosť sa zobrazí formulár na počiatočné nastavenie novej konfigurácie (Obrázok H.6). V tomto formuláre je možné nastaviť meno, podľa ktorého bude možné identifikovať konfiguráciu a pomocou ktorého bude systém špecifikovať konfiguráciu pri získavaní odporúčania.

Po vytvorení novej konfigurácie sa zobrazí detail konfigurácie (8Obrázok H.7), v ktorom je okrem nastavených atribútov možné vidieť časy, kedy boli naposledy prepočítané metriky a podobnosti. Pre novú konfiguráciu sú tieto časy nastavené na aktuálny čas. Na tejto obrazovke je tiež zobrazený zoznam implementácií použitých v danej konfigurácií. Pridať záznam do zoznamu implementácií je možné zvolením typu implementácie a stlačením tlačidla „*New Implementation*“. Je možné pridať implementácie odporúčačov, funkcií na prepočet podobnosti, metrík, kombinovačov odporúčačov a nástrojov na riešenie problému studeného štartu. Rozhrania na pridanie jednotlivých implementácií sa mierne líšia, preto v tejto časti bližšie popíšeme najzákladnejšie z nich.

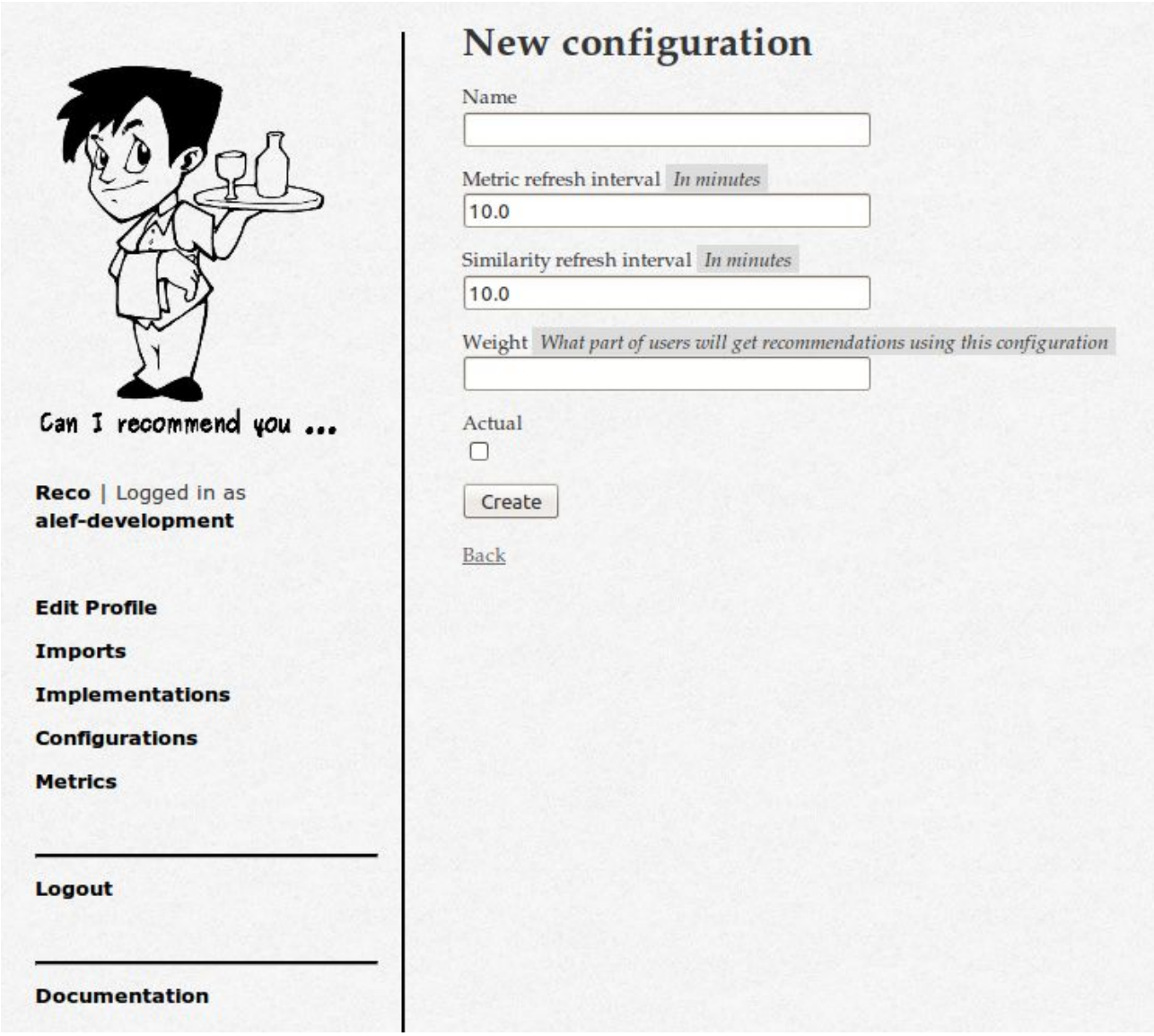
Rozhranie na nastavenie prepočtu podobnosti (Obrázok H.8) obsahuje pole na výber konkrétnej implementácie funkcie na prepočet podobnosti a v závislosti od toho, či ide o podobnosť

používateľov alebo dokumentov, zoznam typov spätnej väzby alebo zoznam implementácií algoritmov na nájdenie kľúčových slov spolu s ich váhami. Ďalší typ spätnej väzby alebo implementácie algoritmu na nájdenie kľúčových slov je možné pridať kliknutím na tlačidlo „*Add next*“.

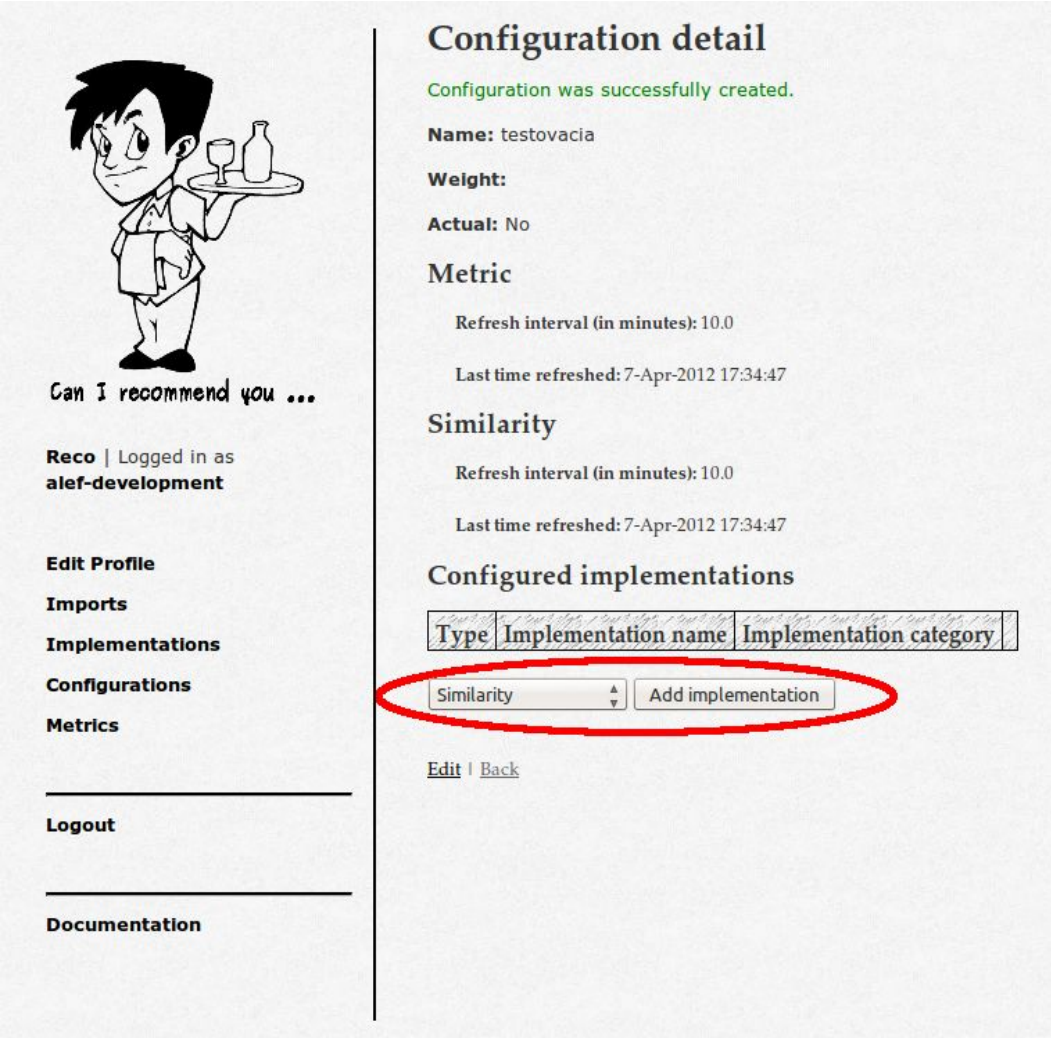
Po pridaní konfigurácie pre výpočet podobnosti stačí pridať konfiguráciu odporúčacieho algoritmu a môžeme začať odporúčať. Konfigurácia odporúčacieho algoritmu je znázornená na obrázku H.9. Pri konfigurácií odporúčacieho algoritmu je potrebné nastaviť implementáciu, ktorú chceme použiť, implementáciu na výpočet podobnosti spomedzi tých, ktoré sme si už nakonfigurovali, Počet výsledkov, ktoré chceme dostávať. Ďalej je to poradie a váha pri kombinovaní viacerých algoritmov a nakoniec typy súborov, ktoré sa majú odporúčať. Takto pripravená konfigurácia stačí na získanie odporúčania.



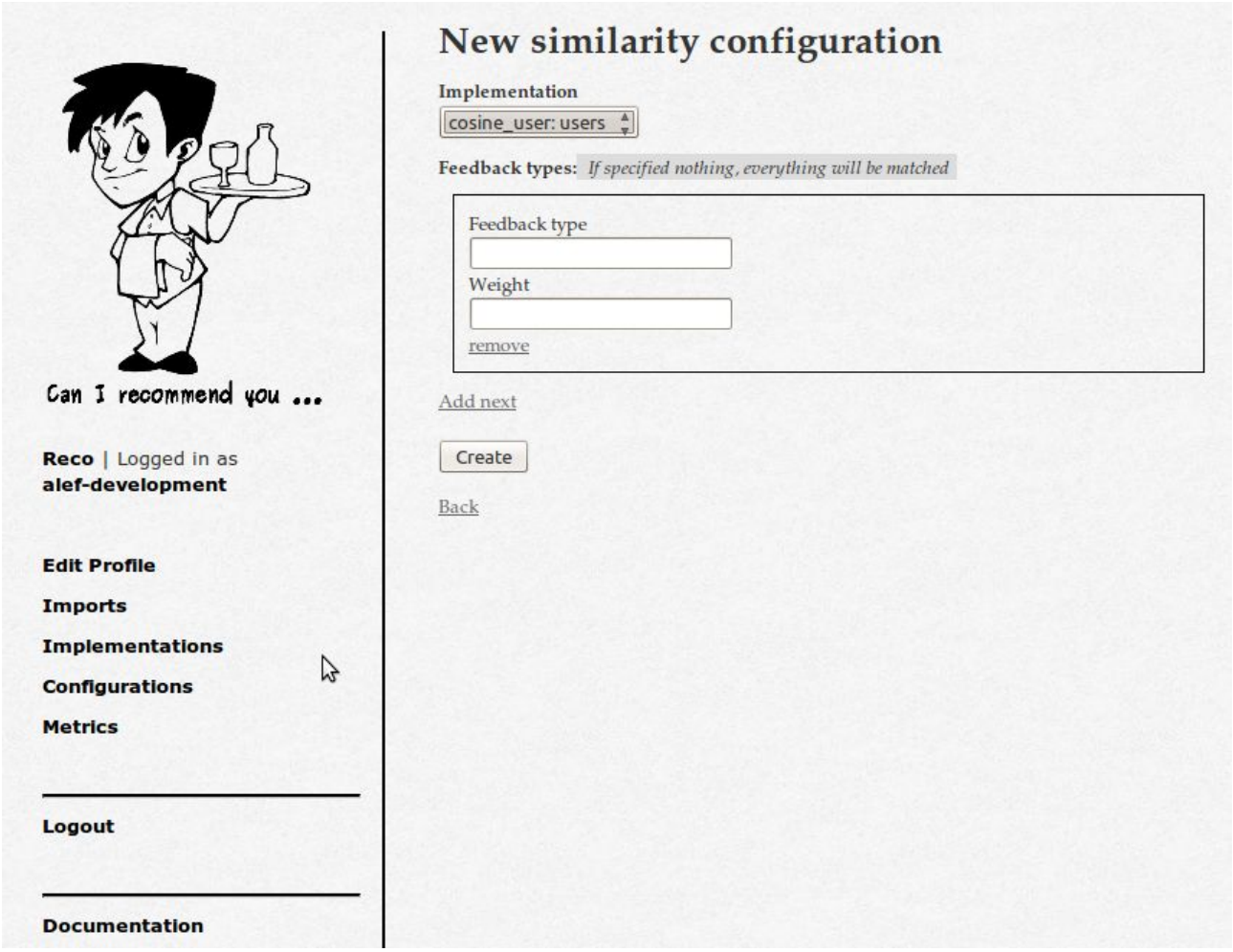
Obrázok H.5: Zoznam všetkých konfigurácií systému



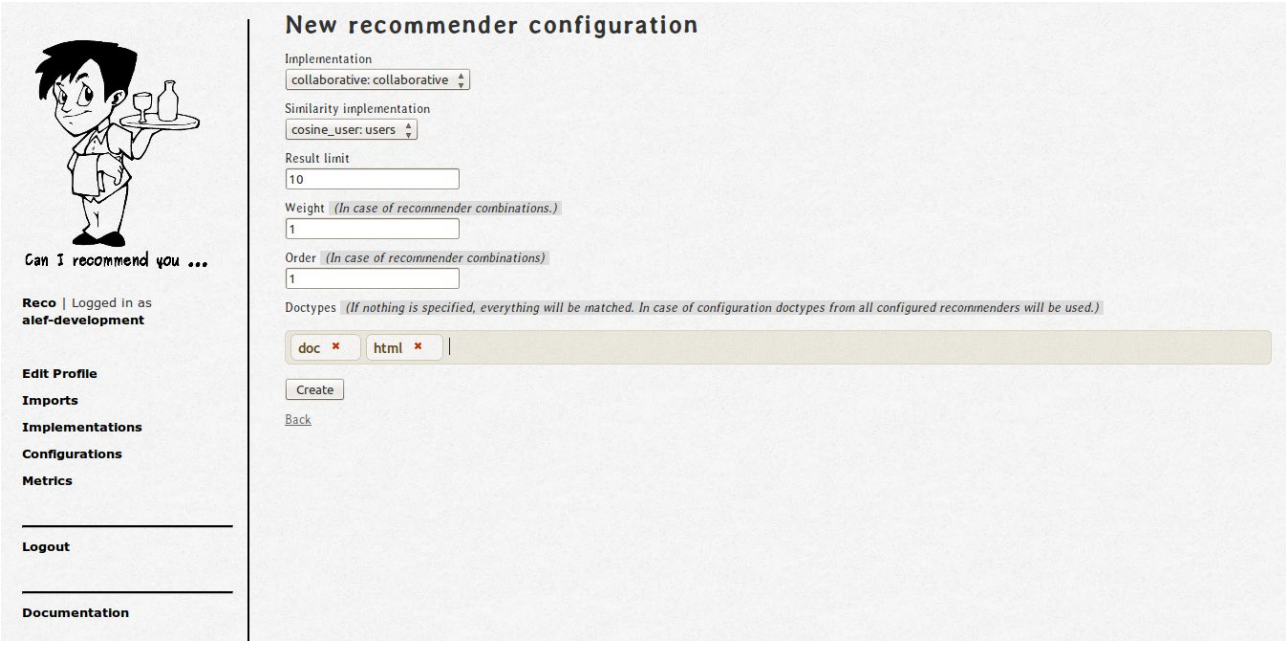
Obrázok H.6: Formulár na nastavenie novej konfigurácie



Obrázok H.7: Detail konfigurácie



Obrázok H.8: Rozhranie na pridanie implementácie funkcie na prepočet podobnosti do konfigurácie

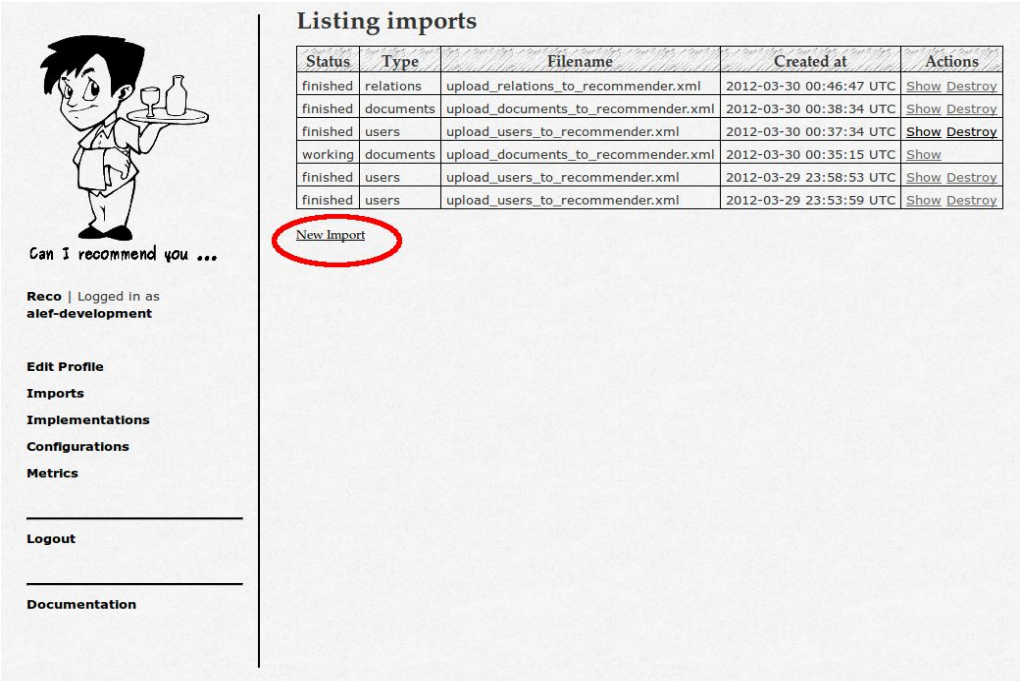


Obrázok H.9: Konfigurácia odporúčacieho algoritmu

Importovanie údajov

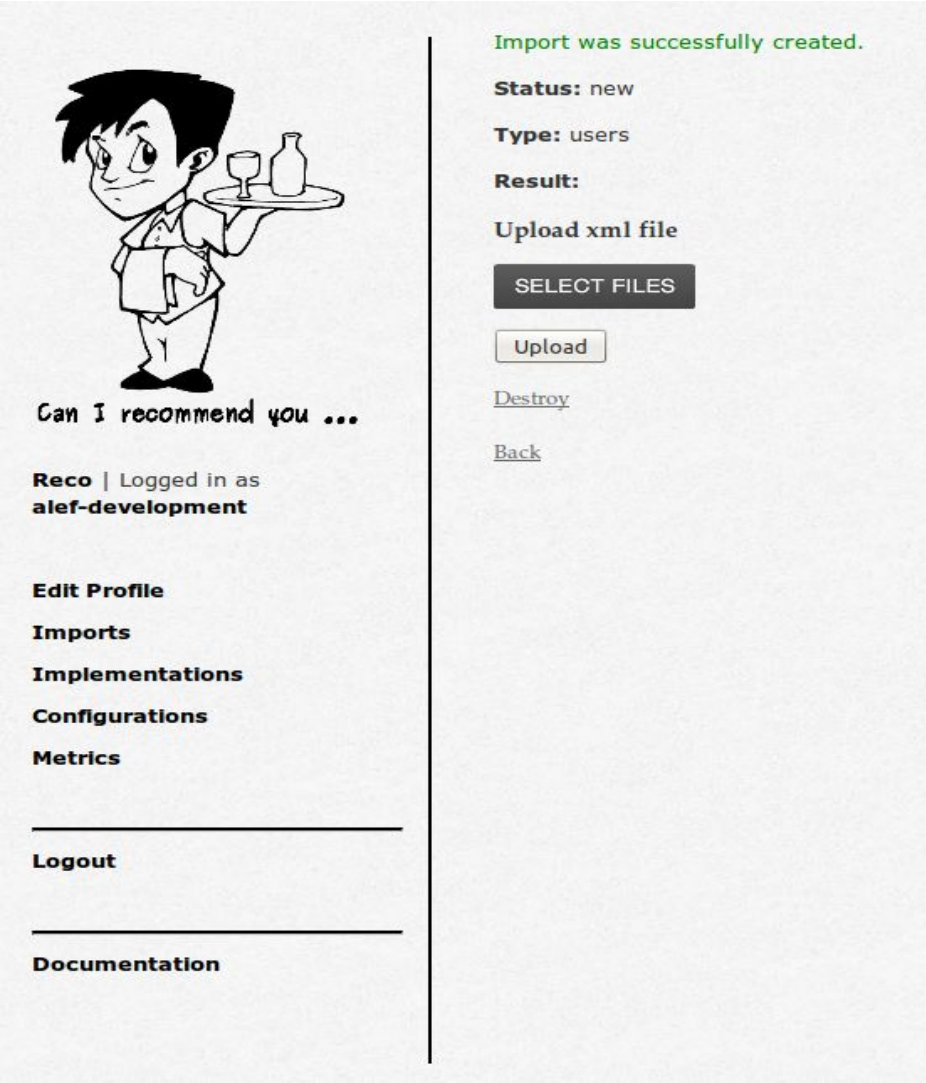
Po vytvorení konfigurácie je potrebné vložiť údaje, na základe ktorých sa má vypočítavať odporúčanie. Na vkladanie týchto údajov existuje API volanie, ale aj grafické rozhranie. Grafické rozhranie je dostupné po kliknutí na voľbu „*Imports*“ v ľavom menu. Zobrazí sa zoznam importov spolu s možnosťou vytvoriť nový (Obrázok H.10). Po kliknutí na voľbu „*New Import*“ sa zobrazí formulár na nastavenie importu, kde je potrebné zvoliť si typ importu (Obrázok H.11). Na výber sú dostupné typy *documents*, *documents with keyword search*, *user* a *relations* podľa toho, či chceme importovať dokumenty, dokumenty s vyhľadávaním kľúčových slov v texte, používateľov alebo relácie medzi používateľmi a dokumentami. Pri importe dokumentov a používateľov sa staré záznamy upravujú, zatiaľ čo pri importe relácií sa stále pridávajú nové záznamy.

Po vytvorení importu sa zobrazí obrazovka s detailom importu a s možnosťou nahrania súboru na importovanie (Obrázok H.12). Pri nahrávaní súboru sa zobrazuje progress bar spolu s percentuálnym hodnotením stavu nahrávania súboru na server. Po nahraní súboru sa zobrazí detail importu s možnosťou spustiť samotný import (Obrázok H.13). Pri ďalšom zobrazovaní detailu importu sa bude zobrazovať status importu spolu s krátkym sumárom vykonaných úloh, prípadne s popisom chyby, ktorá nastala (Obrázok H.14).

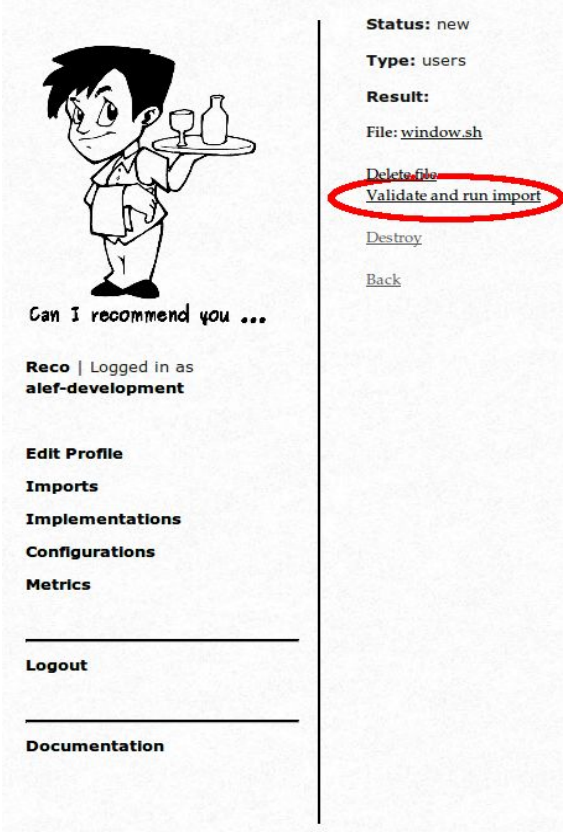


Obrázok H.10: Zobrazenie zoznamu importov

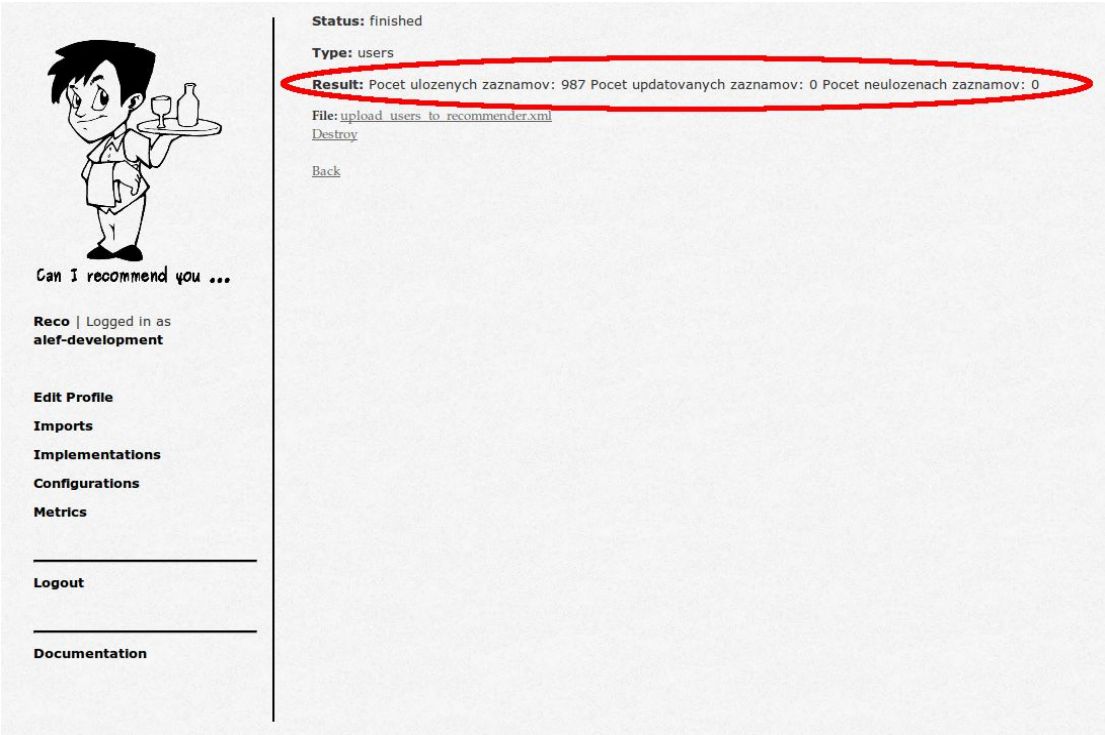
Obrázok H.11: Rozhranie na výber typu importu



Obrázok H.12: Detail importu a možnosť nahrať súbor na import



Obrázok H.13: Možnosť spustenia importu v detaile importu

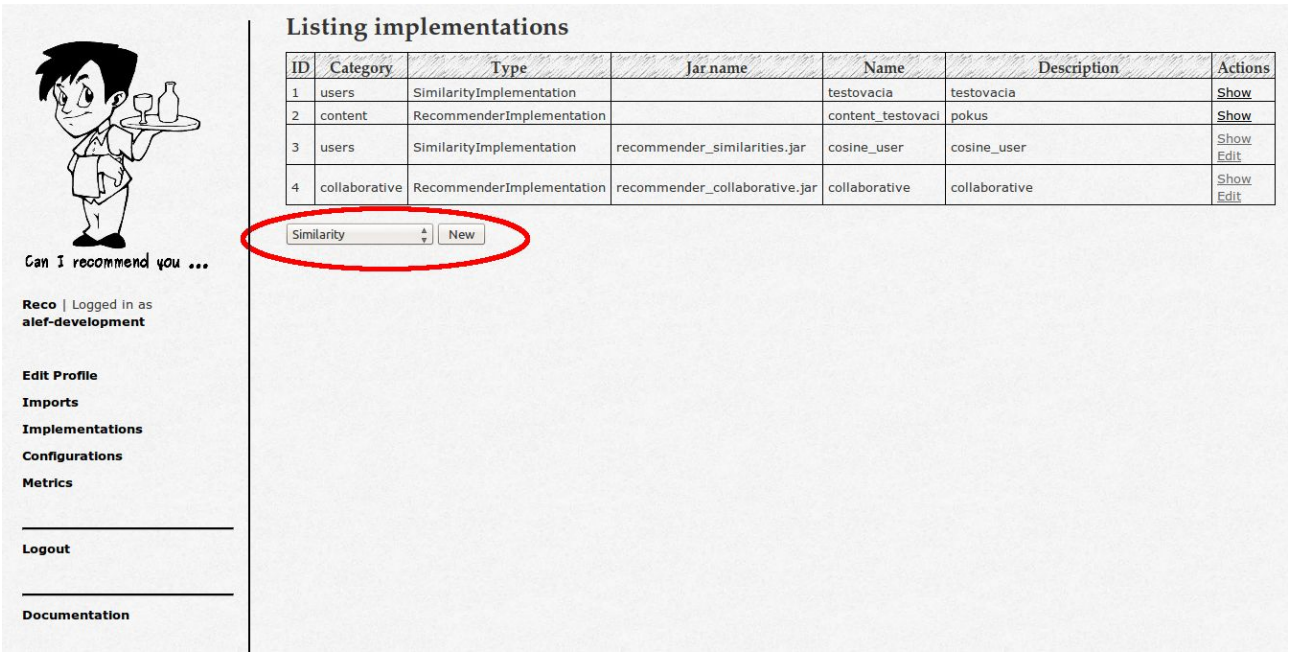


Obrázok H.14: Výsledok dokončeného importu

Pridanie implementácie

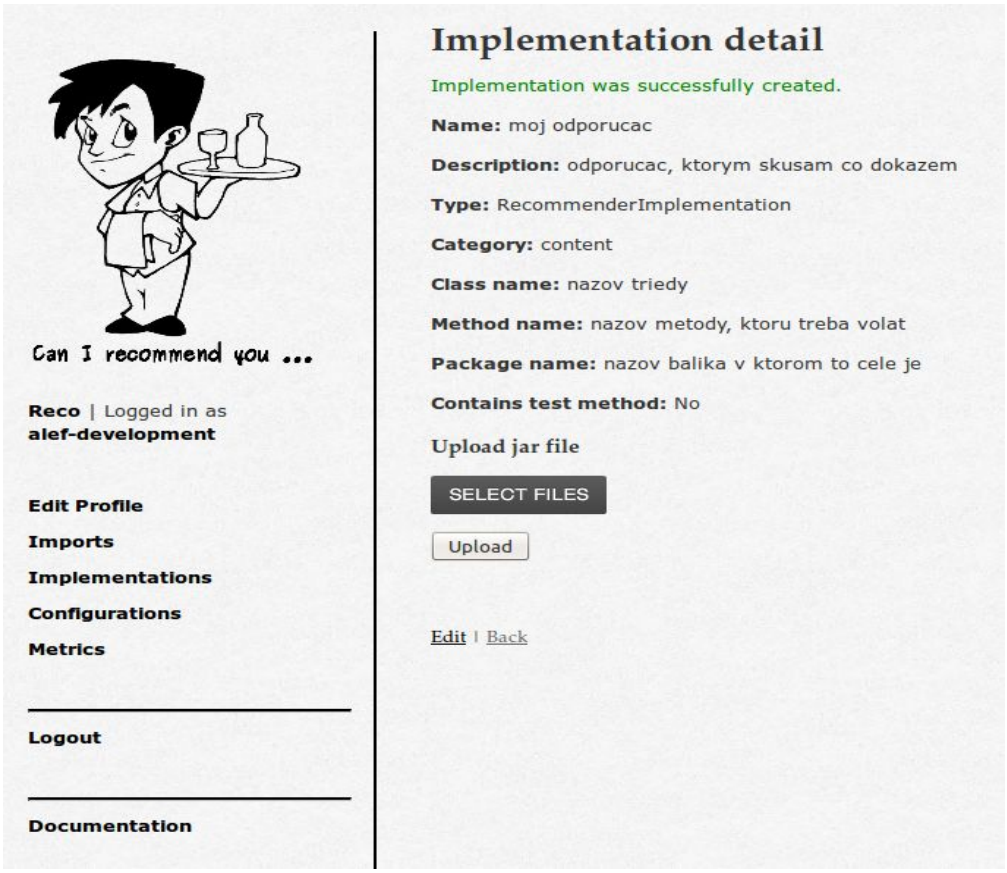
Okrem konfigurovania odporúčačov existujúcich implementácií algoritmov je možné pridávať vlastné implementácie algoritmov pomocou na to určeného rozhrania. Je možné pridávať implementácie v jazyku Java, ktoré používajú definované rozhrania. Na pridanie implementácie slúži zohranie, ktoré sa objaví po kliknutí na odkaz „*Implementations*“ v ľavej časti stránky. Zobrazí sa zoznam všetkých existujúcich implementácií, upravovať je však možné len tie, ktoré patria danému systému (Obrázok H.15).

Pod týmto zoznamom implementácií sa nachádza voľba na pridanie vlastnej implementácie. Po Vybratí typu implementácie zo zoznamu a kliknutí na tlačidlo „New“ sa zobrazí obrazovka na nastavenie detailu implementácie (Obrázok H.16). Po nastavení a vytvorení novej implementácie sa zobrazí okno na nahranie súboru implementácie (Obrázok H.17). V tomto rozhraní je potrebné nahrať JAR súbor, v ktorom je zabalená celá implementácia. Po nahraní implementácie je možné použiť implementáciu v ľubovolnej konfigurácii.



Obrázok H.15: Zoznam všetkých implementácií

Obrázok H.16: Rozhranie na nastavenie novej implementácie



Obrázok H.17: Rozhranie na nahranie jar súboru implementácie

Aplikačné rozhranie (API)

API poskytuje niekoľko druhov 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://team12-11.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, či ide o import používateľov, vzťahov, dokumentov alebo dokumentov s hľadaním kľúčových slov. Podľa toho parameter nadobúda hodnoty „users“, „relations“, „documents“ alebo „docsCountKeyword“.
file	Súbor, ktorý sa má importovať. Súbor je XML súbor, zodpovedajúci XSD schémam opísaným v častiach Error: Reference source not found, Príloha J a Príloha K

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 s dokumentami v systéme a s odporúčaním, ktoré sme poskytli. Toto volanie slúži na poskytnutie spätnej väzby o jednej interakcií používateľa s dokumentom, na ktorý sa dostal vďaka odporúčaniu, ale aj inak. 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://team12-11.ucebne.fiit.stuba.sk/app/feedback<?parameters>

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

Parameter	Popis
api_key	Apikľúč systému, ktorý posiela spätnú väzbu.
type	Typ spätnej väzby. Nedefinujeme žiadne typy, je teda len na používateľovi čo tu uvedie. Tento typ sa používa napríklad pri prepočte podobnosti.
weight	Váha spätnej väzby
user_id	Identifikácia používateľa
document_id	Identifikácia dokumentu
recommendation_id	Identifikácia odporúčania. Tento parameter nieje povinný, ale bez neho nebude možné vyhodnocovať úspešnosť odporúčača.
appended_at	Dátum, kedy používateľ vykonal akciu, ktorej sa týka táto spätná väzba. Ak tento čas nieje špecifikovaný, použije sa súčasný čas na servery.

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 parameter missing</error>
  <error>document parameter missing</error>
</errors>
```

Získanie odporúčania

Url pre volanie funkcie na získanie odporúčania je:

http://team12-11.ucebne.fiit.stuba.sk/app/api/recommend<*.format*><?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.
config	Identifikácia configurácie odporúčania. Tento parameter nieje povinný, ak nieje špecifikovaný vyberie sa configurácia na základe v nich definovaných priorít (AB-testovanie).

Podporované formáty, v ktorých môže aplikácia poskytovať odporúčenie“

Formát	Popis
Json	Výsledok je vo formáte json v podobe asociatívneho poľa
XML	Výsledok je xml súbor so zoznamom odporúčaní prípadne zoznamom chýb
	Ak nieje špecifikovaný žiadny typ výsledok je vo formáte Json

Výsledok volania pre formát json je serializované asociatívne pole obsahujúce odporúčané dokumenty a identifikáciu odporúčania:

```
{"results":
["article1","article2","article400","article33"],"recommendation":46}
```

alebo chybové hlášky:

```
{"errors":["api_key parameter missing","user_identification parameter
missing"]}
```

Výsledok volania pre formát XML je súbor obsahujúci identifikáciu odporúčania a zoznam odporučených dokumentov:

```
<?xml version="1.0" encoding="UTF-8"?>
<results>
  <recommendation>46</recommendation>
  <result>article1</result>
  <result>article2</result>
  <result>article400</result>
  <result>article33</result>
</results>
```

alebo XML súbor obsahujúci chybové hlášky:

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

Opis algoritmov

Riešiče problému studeného štartu

Používateľ má na výber z 3 predprogramovaných riešičov problémov studeného štartu.

1. Random riešič problému studeného štartu
2. Najhodnotenejšie dokumenty
3. Najčítanejšie dokumenty

všetky riešiče problému studeného štartu obsahujú nasledovnú metódu pretože musia všetky dediť od triedy *recommender/lib/jrecommender/src/main/sk/stuba/fiit/tp1112/recommender/core/interfaces/ColdStart.java*:

```
Object[][] getAdditionalDocs(
                    int userId,
                    Object[] docIdentifications,
                    int resultLimit,
                    String[] feedbackType,
                    String[] docType);
```

Každý riešič problému studené štartu môže mať definované dokumenty, ktoré už používateľ videl v parametri *docIdentifications*. Tieto sa potom nezahrnú do výsledkov. Taktiež máme ohraničený maximálny počet výsledkov, ktorý nám má algoritmus vrátiť v parametri *resultLimit.* Ak chceme vyberať z pomedzi všetkých typov dokumentov a všetkých typov feedbackov, tak na mieste parametrov *feedbackType* a *docType* dáme prázdne polia.

Algoritmus najprv vyhľadá prvú sériu dokumentov, okrem tých ktoré boli zadané na ignorovanie, ak boli zadané. Potom z týchto dokumentov odstráni tie, ktoré používateľ už videl. Tie ktoré sa odstránili sa pridajú do ignorovaných a ostatné čo zostali sa pridajú do výsledných. Takto sa to opakuje kým nedosiahneme požadovaný počet, alebo sa nám z databázového volania nevráti žiadny dokument.

Každý algoritmus má rovnaký priebeh, len volá iné metódy pracujúce z databázou. Najviac odlišný od ostatných je random riešič problému studeného štartu.

Ďalej popíšeme jednotlivé rozdiely medzi algoritmami

Random

Vráti dokumenty, ktoré daný používateľ nečítal, ale pritom nemuseli byť čítané inými používateľmi. Algoritmus môže vrátiť aj menej dokumentov, ako si používateľ vyžiadal a to v prípade, že v systéme nie je dostatočný počet dokumentov alebo už všetky dokumenty používateľ čítal. Váha odporučenia všetkých dokumentov je 1, pretože nemáme podľa čoho spočítať silu odporučenia, keďže vyberáme dokumenty náhodne. Pri tomto riešiči nefunguje parameter *feedbackType,* pretože berieme dokumenty celého systému a nie len videné.

Najhodnotenejšie dokumenty

Tento riešič vyberie ako prvé dokumenty tie, ktoré majú najviac záznamov feedbackov, Počíta sa

každý jeden feedback, aj feedback rovnakého typu, teda od jedného používateľa ich môže byť viacero.. Váha doporučenia sa počíta ako (suma váh týchto feedbackov / maximálna váha z vrátených súm).

Najčítanejšie dokumenty

Tento riešič berie ako váhu počet používateľov, ktorí ho čítali. Každá dvojica dokument používateľ sa započíta iba raz. Výsledná váha je (počet čitateľov daného dokumentu / maximálny počet čitateľov z vrátených dokumentov).

Odporúčacie algoritmy

2.12. Všetky odporúčacie algoritmy musia dediť od triedy *recommender/lib/jrecommender/src/main/sk/stuba/fiit/tp1112/recommender/core/interfaces/Recommender.java.*

Odporúčanie založené na obsahu

Odporúčanie založené na obsahu je základný odporúčací algoritmus, ktorý je súčasťou nášho systému. V krátkosti, odporúčanie založené na obsahu berie do úvahy, aké dokumenty zaujímajú konkrétneho konzumenta služby. Na základe toho odhadne, ktoré iné dokumenty by ho mohli zaujímať. V implementácii nášho odporúčacieho algoritmu sa najskôr z databázy získa určité množstvo dokumentov (množstvo určené argumentom *resultLimit*). Sú to dokumenty, ktoré konzument s ID *userId* už videl a pomocou spätnej väzby ich ohodnotil ako najviac podobné tomu, čo hľadal. Keďže konzument môže vrátiť nie len jeden typ spätnej väzby, to ktoré typy sa vezmú do úvahy pri tomto dopyte určuje parameter *feedbackTypes*. Je to pole reťazcov, kde každý reťazec je názov typu spätnej väzby. Sú to vlastne hodnoty z tabuľky *user_to_document_relations* stĺpec *type.* Čiže už vieme povedať, čo konzumenta služby zaujíma.

Nasledujúci dopyt nad databázou vyberie dokumenty iné ako dokumenty, ktoré videl, ale sú im čo najviac podobné. Podobnosť dokumentov je vypočítaná a uložená v tabuľke *document_similarities* a pri dopyte sa dokumenty usporiadajú podľa váhy, ktorá je zaznamenaná v tejto tabuľke. To, ktorá metóda výpočtu podobnosti sa použije, je určené parametrom *similarityIimplementationId.* Tento parameter je ID použitej implementácie metódy podobnosti. Čiže konzumentovi sa nakoniec odporučí *resultLimit* dokumentov, ktoré ešte nevidel a ktoré sú najviac podobné tým, ktoré už videl podľa danej metódu výpočtu podobnosti.

Metóda odporúčača je deklarovaná takto:

```
Object[][] getRecommendation(
      int userId,
      String[] fromDocuments,
      int resultLimit,
      int similarityIimplementationId,
      String[] feedbackType,
      String[] docTypes)
```

Metóda odporúčania je upravená aj pre prípad, že potrebujeme odporúčať nie zo všetkých

dokumentov, ale iba z určitej množiny. Vtedy sa ako parameter odporúčača dá použiť zoznam dokumentov – *fromDocuments,* z ktorých chceme odporúčať. Tento parameter je pole reťazcov a každý reťazec je identifikátor (nie ID!) dokumentu.

Posledný parameter *docTypes* je pole reťazcov, kde každý reťazec označuje typ dokumentu, ktorý akceptujeme pri odporúčaní. Tento parameter sa používa pri dopyte na dokumenty, ktoré konzument videl aj pri dopyte na získanie odporúčania. Typ dokumentu môže byť ľubovoľné označenie (napríklad *html*, *video*, *page*, ale aj MIME typ napríklad application/pdf, application/vnd.oasis.opendocument.text, atď.)

Ešte pre úplnosť treba dodať, že všetky dopyty na databázu usporiadavajú získané dokumenty podľa váhy od najvyššej po najnižšiu (ORDER BY … DESC).

Kolaboratívne odporúčanie

Tento typ odporúčania je zameraný na podobnosť medzi používateľmi jedného systému a odporúča dokumenty na základe tejto podobnosti.

Algoritmus funguje takto:

Najprv sa nájde skupinka 100 najpodobnejších používateľov danému používateľovi, potom sa pre každého používateľa vráti 100 najodporúčanejších dokumentov a z nich sa odstránia tie, ktoré už používateľ videl. Ak sa stále nedosiahol želaný počet dokumentov na odporučenie, tak sa vyberie ďalšia skupinka najpodobnejších používateľov a proces sa opakuje, kým nám dokumenty pribúdajú, alebo kým nedosiahneme želaný počet.

Váha sa počíta nasledovne: Pre každého používateľa sa spočítajú váhy feedbackov podľa vzorca (1-1/(suma váh * počet feedbackov)) a tie sa vynásobia váhou podobnosti s používateľom, ktorému odporúčame. Ak aj iný používateľ odporúča rovnaký dokument, tak sa všetky takéto hodnoty spoja opäť podľa prvého vzorca (1-1/(suma váh * počet rôznych používateľov)). Vždy dostaneme číslo v rozmedzí 0 až 1, ktoré bude vo výslednom zozname.

Funkcia na volanie kolaboratívneho odporúčania je rovnaká ako na volanie odporúčania založeného na obsahu, ale parameter *fromDocumets* nemá pri kolaboratívnom prístupe zmysel, a preto nie je podporovaný v tomto odporúčaní. Aj po definovaní nejakého zoznamu dokumentov sa bude odporúčať zo všetkých dokumentov, ktoré už boli videné. Význam ostatných parametrov je rovnaký až na parameter *similarityIimplementationId,* ktorý značí typ prepočítavania podobnosti medzi používateľmi a nie medzi dokumentami.

Spracovanie textu

Pri importovaní textových dokumentov sa zároveň zisťujú i kľúčové slová. Základom pre získanie kľúčových slov je upravený text, kde slová sú v základnom tvare, bez zbytočných interpunkčných znamienok a bez stop slov. V prípade importu HTML súborov, je potrebné aj odstrániť HTML značky. Spracovanie textu spočíva v niekoľkých krokoch:

1. Odstránenie HTML značiek – vykoná sa na začiatku, pretože odstráni hneď signifikantné množstvo textu. Je vykonané regulárnym výrazom nahradzujúcom všetko medzi otváracou a zatváracou značkou (vrátane daných značiek „<“ a „>“) jednou medzerou.
2. Konverzia na malé písmená – celý text sa prevedie na malé písmená podľa použitého jazykového nastavenia. Je to potrebné kvôli ďalšej úprave, najmä pri použití slovníkov stop slov a pri premene na základný tvar (tieto úpravy sú case-sensitive).

3. Odstránenie špeciálnych znamienok – regulárnym výrazom sa všetky znaky „<([{}])>\/^-=$|*+;:,@#%&_“ nahradia medzerou. Zachovajú sa iba ukončenia viet (.?!) a apostrofy (') kvôli niektorým anglickým slovám.
4. Odstránenie stop slov – stop slová sú slová, ktoré sa v danom jazyku vyskytujú veľmi často, ale nie sú dôležitými nositeľmi významu. Tieto slová sú odstránené z textu za pomoci slovníkov stop slov, ktoré máme pre anglický a slovenský jazyk.
5. Transformácia textu do základného tvaru – pre obe jazyky (angličtina i slovenčina) je text spracovaný rozdielne. Pre angličtinu sa každé slovo textu zmení na svoj koreň – stem. Pre to sa používa Porterov stemovací algoritmus, ktorý sme prebrali z http://www.tartarus.org/~martin/PorterStemmer. Pre slovenčinu používame slovník s vyše 600 tisíc slovami a ich základnými formami – lemmami. Takto vieme ku drvivej väčšine slov v rôznych tvaroch priradiť základný tvar.
6. Úprava – transformovaný text v základnom tvare po predošlých úpravách obsahuje zvyčajne veľa zbytočných medzier. Preto je regulárnym výrazom každý výskyt 2 a viacerých medzier nahradený jednou medzerou.

Získavanie kľúčových slov

Po úspešnom predspracovaní textu prichádza na rad samotná extrakcia kľúčových slov. Algoritmy na extrakciu kľúčových slov musia rozširovať abstraktnú triedu triedu *KeywordExtractor* z balíka *sk.stuba.fiit.tp1112.recommender.core.interfaces.* Samotná logika extrakcie kľúčových slov sa potom bude nachádzať v preťaženej metóde:

```
protected LinkedHashMap<String, Integer> fillMap(String text)
```

Táto metóda dostáva ako argument predspracovaný text a vracia objekt typu *LinkedHashMap,* ktorý obsahuje nájdené kľúčové slová spolu s ich vypočítanou váhou / relevanciou. Maximálny počet kľúčových slov, ktorý sa pre jeden dokument uloží do databázy je 20 slov.

V našom systéme existujú zatiaľ dva algoritmy na extrakciu kľúčových slov, a to síce naivná a query metóda.

Naive

Jednoduchá metóda, ktorá vypočíta váhu kľúčových slov na základe toho, koľkokrát sa v danom dokumente slovo vyskytlo.

Query

- Zvolí sa okno dĺžky K a postupne po jednom slove sa posúva cez celý dokument
- Predpokladá sa, že všetky slová v okne majú spoločný výskyt s prvým slovom v okne.
- Určí sa n(w1, k, w2) označujúce počet, koľkokrát sa dve slová (w1 a w2) v okne spolu-vyskytujú, pričom *k* označuje vzdialenosť medzi nimi.
- Sila tohto spolu-výskytu sa určuje ako W(k) = K – k + 1
- Vypočíta sa relevantný stupeň dvoch slov *w1* a *w2* ako súčin sily a počtu spolu-výskytu pre každú vzdialenosť *k*, v ktorej sa dve slová spolu-vyskytujú. Tieto výsledky sa potom pre každé také k spočítajú.

- Výsledná váha slova sa potom vypočíta ako súčet všetkých relevantných stupňov slova.

Metriky

Boli implementované dva algoritmy výpočtu metrík, *Precision* a *Recall.* Obe sú v balíku *sk.stuba.fiit.tp1112.recommender.metrics*. Implementácie výpočtu metrík musia rozširovať triedu *Metrics* v balíku *sk.stuba.fiit.tp1112.recommender.core.interfaces.* V tejto triede je jediná abstraktná metóda, ktorú je potrebné preťažiť:

```
public abstract void calculateMetrics(int configuration_id, Date from, Date to)
```

Prvým argumentov je identifikačné číslo konfigurácie, pre ktorú sa počíta metrika. Spätné väzby používateľa a odporučenia dokumentov ovplyvňujúce výpočet musia patriť do tejto konfigurácie. Druhý argument je začiatočný dátum výpočtu metriky. Tretí argument je konečný dátum výpočtu metriky. Tieto dátumy tvoria časový interval. V databáze ovplyvňujú výpočet metriky len tie záznamy, ktoré patria do časového intervalu. Používateľ má možnosť zadať hodnotu *null* ako začiatočný aj konečný dátum. Ak začiatočný dátum je *null*, výpočet metriky nie je obmedzený začiatočným dátumom. Podobne, ak konečný dátum je *null*, výpočet metriky nie je obmedzený konečným dátumom.

Výpočet metriky podľa *Precision* algoritmu je postavený na nasledujúcom matematickom vzťahu:

$$Precision = \frac{|\{spätná\ väzba\} \cap \{odporúčanie\}|}{|\{odporúčanie\}|}$$

Výpočet metriky podľa *Recall* algoritmu je postavený na nasledujúcom matematickom vzťahu:

$$Recall = \frac{|\{spätná\ väzba\} \cap \{odporúčanie\}|}{|\{spätná\ väzba\}|}$$

Parameter *odporúčanie* v hore-uvedených matematických vzťahoch sú všetky odporúčania, ktoré boli odporučené pre konkrétneho používateľa na konkrétny dokument. Inak povedané, sú to záznamy v tabuľkách *recommendations* a *document_to_recommendation_relations* s totožným identifikačným číslom odporúčania.

Parameter *spätná väzba* sú všetky spätné väzby, pre ktoré existuje odporúčanie. Sú to záznamy v tabuľkách *recommendations* a *user_to_document_relations* s totožným identifikačným číslom odporúčania.

Keďže sa metrika vždy počíta len pre jednu konfiguráciu, na konci ľubovolného algoritmu metrík sa pridá jeden záznam do tabuľky *metrics*. Záznam obsahuje identifikačné číslo záznamu, dva cudzie kľúče, konkrétne identifikačné číslo konfigurácie a implementácie, výsledok algoritmu metriky, začiatočný a konečný dátum pre počítanie metriky, kedy bol záznam vytvorený a aktualizovaný. Výsledkom algoritmu metriky je *double* hodnota z intervalu <0,1>. 1 znamená ideálnu zhodu medzi spätnou väzbou používateľa a odporučením. 0 vyjadruje, že používateľ pozitívne nereagoval ani na jedno odporučenie.

Počítanie podobnosti

Počítanie podobnosti dokumentov

Všetky algoritmy počítajúce podobnosť dokumentov v systéme musia rozširovať triedu *SimilarityDocument* z balíka *sk.stuba.fiit.tp1112.recommender.core.interfaces.* Táto trieda obsahuje dve metódy ktoré je nutné preťažiť.

- `public boolean calculate(Document sourceDoc, List <Document> otherDocs)`
- `public boolean calculate(Document sourceDoc, Document targetDoc)`

Prvá metóda dostáva ako argumenty zdrojový dokument a list cieľových dokumentov. Pointou tejto metódy je prednačítať si údaje pre zdrojový dokument a následne vypočítať podobnosť medzi týmto zdrojovým dokumentom a každým z cieľových dokumentov pomocou druhej metódy.

Druhá metóda potom akurát vypočíta podobnosť medzi zdrojovým a cieľovým dokumentom, a výsledok následne uloží do databázy.

Následne je ešte potrebné doplniť konštruktor pre metódu počítania podobnosti, pričom konštruktor musí byť v predpísanom tvare. Pre lepšiu ilustráciu uvádzame príklad konštruktora z korelačnej metódy počítania podobnosti:

```
public Correlation(int configurationId, int similarityImplementationId, Object[]
                        [] weightList) {
            this.configurationId = configurationId;
            this.similarityImplementationId = similarityImplementationId;
            this.weightList = weightList;
      }
```

Konštruktor musí obsahovať 3 argumenty, prvé dva sa využijú pri ukladaní vypočítanej podobnosti do databázy, nakoľko sa jedná o polia, ktoré sú povinné pre každú vypočítanú podobnosť.

Tretí argument je takzvaný zoznam váh. Výpočty kľúčových slov pre dokumenty (na základe ktorých sa potom počíta podobnosť dokumentov) môžu prebehnúť na základe rôznych algoritmov. Tento zoznam váh potom určuje, z ktorých algoritmov majú byť kľúčové slová brané do úvahy a s akou váhou (súčet týchto váh je vždy jedna). Zoznam váh môže mať aj hodnotu *null*, v tomto prípade sa berú do úvahy výsledky všetkých dostupných algoritmov. V prípade, že je zoznam váh naplnený hodnotami, tak prvé pole zoznamu obsahuje iba jednoducho indexy jednotlivých váh. Druhé pole zoznamu má potom dva prvky, Prvý prvok určuje *id* implementácie (teda algoritmu), ktorého výsledky sa majú brať do úvahy, a druhý prvok poľa určuje váhu s akou sa tieto výsledky berú do úvahy. Dôležité je ešte dodať, že vypočítané hodnoty podobnosti sa môžu nachádzať iba v rozmedzí 0 – 1.

Systém ponúka tri predprogramované algoritmy na výpočet podobnosti, kde X a Y sú vektory obsahujúce hodnotenia kľúčových slov spoločných pre oba dokumenty, a n je počet týchto spoločných kľúčových slov. *MAX* a *MIN* sú potom maximálna a minimálna hodnota hodnotení kľúčových slov.

- Correlation - $\sum_{i=1}^{n}(x_i * y_i) - \sum_{i=1}^{n}(x_i) * \sum_{i=1}^{n}(y_i) / \sqrt{(\sum_{i=1}^{n}(x_i^2) - \sum_{i=1}^{n}(x_i)^2 / n) * (\sum_{i=1}^{n}(y_i^2) - \sum_{i=1}^{n}(y_i)^2 / n)}$

- Cosine-based - $\sum_{i=1}^{n}(x_i * y_i) / \sqrt{\sum_{i=1}^{n}(x_i^2)} * \sqrt{\sum_{i=1}^{n}(y_i^2)}$
- Mean difference - $\frac{\sqrt{\sum_{i=1}^{n}(x_i - y_i)^2 / n}}{MAX - MIN}$

Počítanie podobnosti používateľov

Všetky algoritmy počítajúce podobnosť používateľov v systéme musia rozširovať triedu *SimilarityUser* z balíka *sk.stuba.fiit.tp1112.recommender.core.interfaces*. Táto trieda obsahuje dve metódy ktoré je nutné preťažiť.

- `public boolean calculate(SystemToUserRelation sourceUser, List <SystemToUserRelation> otherUsers)`
- `public boolean calculate(SystemToUserRelation sourceUser, SystemToUserRelation targetUser)`

Prvá metóda dostáva ako argumenty zdrojového používateľa a list cieľových používateľov. Pointou tejto metódy je prednačítať si údaje pre zdrojového používateľa a následne vypočítať podobnosť medzi týmto zdrojovým používateľom a každým z cieľových používateľov pomocou druhej metódy.

Druhá metóda potom akurát vypočíta podobnosť medzi zdrojovým a cieľovým používateľom a výsledok následne uloží do databázy.

Následne je ešte potrebné doplniť konštruktor pre metódu počítania podobnosti, pričom konštruktor musí byť v predpísanom tvare. Pre lepšiu ilustráciu uvádzame príklad konštruktora z korelačnej metódy počítania podobnosti:

```
public Correlation(int configurationId, int similarityImplementationId, Object[]
                        [] weightList) {
            this.configurationId = configurationId;
            this.similarityImplementationId = similarityImplementationId;
            this.weightList = weightList;
      }
```

Konštruktor musí obsahovať 3 argumenty, prvé dva sa využijú pri ukladaní vypočítanej podobnosti do databázy, nakoľko sa jedná o polia ktoré sú povinné pre každú vypočítanú podobnosť.

Tretí argument je takzvaný zoznam váh. Hodnotenia jednotlivých dokumentov používateľmi (na základe ktorých sa potom počíta podobnosť používateľov) môžu prebehnúť na základe rôznych interakcií používateľov s dokumentami. Každá takáto interakcia má potom priradený v databáze nejaký typ, napríklad (videl, ohodnotil, stiahol, čítal, atď...). Tento zoznam váh potom určuje, z ktorých hodnotení majú byť záznamy brané do úvahy, a s akou váhou (súčet týchto váh je vždy jedna). Zoznam váh môže mať aj hodnotu *null*, v tomto prípade sa berú do úvahy výsledky všetkých dostupných typov hodnotení. V prípade, že je zoznam váh naplnený hodnotami, tak prvé pole zoznamu obsahuje iba jednoducho indexy jednotlivých váh. Druhé pole zoznamu má potom dva prvky, Prvý prvok obsahuje stringovú premennú a určuje aký typ hodnotenia sa berie do úvahy a druhý prvok poľa určuje váhu s akou sa tieto výsledky berú do úvahy. Dôležité je ešte dodať, že vypočítané hodnoty podobnosti sa môžu nachádzať iba v rozmedzí 0 – 1.

Systém ponúka tri predprogramované algoritmy na výpočet podobnosti, kde X a Y sú vektory obsahujúce hodnotenia dokumentov spoločných pre oboch používateľov a n je počet týchto

spoločných dokumentov. *MAX* a *MIN* sú potom maximálna a minimálna hodnota hodnotení dokumentov.

- Correlation - $\sum_{i=1}^{n}(x_i * y_i) - \sum_{i=1}^{n}(x_i) * \sum_{i=1}^{n}(y_i) / \sqrt{(\sum_{i=1}^{n}(x_i^2) - \sum_{i=1}^{n}(x_i)^2 / n) * (\sum_{i=1}^{n}(y_i^2) - \sum_{i=1}^{n}(y_i)^2 / n)}$
- Cosine-based - $\sum_{i=1}^{n}(x_i * y_i) / \sqrt{\sum_{i=1}^{n}(x_i^2)} * \sqrt{\sum_{i=1}^{n}(y_i^2)}$
- Mean difference - $\frac{\sqrt{\sum_{i=1}^{n}(x_i - y_i)^2 / n}}{MAX - MIN}$

Príloha I - Upravená XML schéma pre import používateľov

```
<?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 J - Upravená XML schéma pre import dokumentov

```
<?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 K - Upravená XML schéma pre import relácií

```
<?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>

                            <element name="DATETIME" type="dateTime"></element>

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

</schema>
```

Príloha L - Obsah elektronického média

/application - aplikácia

/lib/jrecommender - java časť aplikácie

/documentation

/rdoc - dokumentácia pre Ruby časť aplikácie

/javadoc - dokumentácia pre Java časť aplikácie

/Dokumentácia k inžinierskemu dielu.pdf

- dokumentácia k dielu aj s používateľskou príručkou (pdf)

/Dokumentácia k inžinierskemu dielu.odt

- dokumentácia k dielu aj s používateľskou príručkou (odt)

/Dokumentácia k riadeniu.pdf

- dokumentácia k riadeniu (pdf)

/Dokumentácia k riadeniu.odt

- dokumentácia k riadeniu (odt)

/README

- pokyny pre sprevádzkovanie aplikácie