



Софийски университет “Св. Климент Охридски”
Факултет по математика и информатика
катедра “Компютърна Информатика”

ДИПЛОМНА РАБОТА

Система за препоръчване на музика

Мартин Асенов
факултетен номер 24464, Изкуствен Интелект
Софийски Университет “Св. Климент Охридски”

ръководител:
д-р Милен Чечев

София,
февруари 2015

Съдържание

1. Увод
 - 1.1 Цел и задачи на дипломната работа
 - 1.2 Структура на дипломната работа
2. Преглед на проблемната област
 - 2.1 История и приложения
 - 2.2 Методи на работа
 - 2.2.1 Препоръки основани на съдържание (content-based filtering)
 - 2.2.1.1 Препоръки основани на тагове
 - 2.2.2 Препоръки основани на сътрудничество (collaborative filtering)
 - 2.2.2.1 Memory-based сътрудничество
 - 2.2.2.1.1 Сътрудничество основано на потребители
 - 2.2.2.1.2 Изчисляване на сходство и предсказване на оценката на потребител при сътрудничество основано на потребители
 - 2.2.2.1.3 Сътрудничество основано на продукти
 - 2.2.2.1.4 Изчисляване на сходство и предсказване на оценката на потребител при сътрудничество основано на продукти
 - 2.3 Сравнение между препоръки основани на сътрудничество и препоръки основани на съдържание
 - 2.4 Хибридни препоръчващи подходи (hybrid recommendation systems)
 - 2.5 Оценяване работата на препоръчващи системи
 - 2.5.1 Офлайн оценяване
 - 2.5.2 Онлайн оценяване
 - 2.6 Метрики за оценяване точността на препоръчваща система
 - 2.6.1 Метрики за точност на предсказаниите оценки
 - 2.6.2 Метрики за коректност на препоръките
 - 2.6.3 Метрики за коректност на наредбата на препоръките
 - 2.6.4 Покритие на препоръките (Coverage)
 - 2.6.5 Сигурност на препоръките (Confidence)
 - 2.6.6 Развообразие
 - 2.6.7 Доверие
 - 2.6.8 Полезност
 - 2.6.9 Адаптивност
 - 2.6.10 Новост
 3. Анализ на изискванията
 - 3.1 Концептуален модел
 - 3.2 Функционални изисквания - основни случаи на употреба

- 3.2.1 Свързване на Facebook акаунт
 - 3.2.2 Свързване на Last.fm потребителско име
 - 3.2.3 Оценяване на предварително зададен списък с песни
 - 3.2.4 Настройка на параметрите на препоръчващия алгоритъм
 - 3.2.5 Настройка на типа на препоръчващия алгоритъм
 - 3.2.6 Настройка на типа на използваните при препоръка данни
 - 3.2.7 Преглед на персонализирани препоръки
 - 3.2.8 Генериране на персонализирани препоръки чрез сътрудничество
 - 3.2.9 Генериране на персонализирани препоръки чрез съдържание
 - 3.2.10 Онлайн оценяване качеството на направените препоръки
 - 3.2.11 Офлайн оценяване качеството на направените препоръки
 - 3.2.12 Агрегиране на потребителските оценки за работа на системата
 - 3.2.13 Събиране на данни от last.fm
- 3.3 Нефункционални изисквания
- 4. Проектиране, архитектура и реализация на препоръчващата система
 - 4.1 Модел на данните
 - 4.2 Декомпозиция на модулите
 - 4.3. Реализация на препоръчващата система
 - 4.3.1 Избор на препоръчващ алгоритъм
 - 4.3.2 Преглед на метода за събиране и съхранение на данни от last.fm
 - 4.4 Използвани технологии и услуги
 - 4.4.1 AngularJS
 - 4.4.2 Twitter Bootstrap
 - 4.4.3 Nodejs
 - 4.4.4 Node модули
 - 4.4.5 MongoDB
 - 4.4.6 Програмен интерфейс на Facebook
 - 4.4.7 Програмен интерфейс на Last.fm
5. Оценяване
- 5.1 Офлайн оценяване
 - 5.2 Онлайн оценяване
 - 5.3 Оценка на работата на системата спрямо стандартните критерии за оценяване
6. Заключение
- 6.1 Обобщение
 - 6.2 Насоки за усъвършенстване на системата и бъдещо развитие
- Литература
- Приложения
- Потребителски интерфейс

1. Увод

С увеличаването на информационните източници през последните години с все по-голяма тежест стои въпросът за филтрирането на информация. Потребителите имат на разположение огромни количества данни - към юли 2013г. търсачката *Google* е индексирала повече от 38 трилиона страници (TechNews, 2013) и са изправени пред проблема да изберат точно каква част от тези данни да прегледат, да преценят каква част от тях ще им бъдат полезни. Този проблем се решава от т.нар. препоръчващи системи – приложения, които подбират само този дял от множество обекти, който би бил интересен за потребителя.

В рамките на дипломната работа ще изследваме изграждането на препоръчваща услуга, която работи с музика. Подобни решения намират широко практическо приложение, тъй като в днешно време всеки слуша музика в електронен формат и често пъти използва за целта услуга достъпна по интернет - *Youtube*, *Spotify*, *last.fm* и т.н. Това води до натрупването на сериозни количества данни за интересите на потребителите и съответно възможност за изготвянето на качествени препоръки. Интерес от такива препоръки имат както ползвателите им - за да намират подходяща за тях музика, така и собствениците на онлайн магазини и сайтове за слушане на музика - за да увеличават приходите си.

1.1 Цел и задачи на дипломната работа

Основните цели на дипломната работа са свързани с изграждането на препоръчваща система за музика, с изследването на възможните подходи и проблеми в този процес и разглеждането на техните решения.

Задачите, които произтичат от тези цели са както следва:

- изследване на различни препоръчващи алгоритми с фокус върху откриването на подходящ, който да бъде ефективен при препоръчването на музика
- разработване на препоръчваща система, която да предлага подходяща музика на своите потребители, основавайки се на техния профил
- оценяване на точността на изградената система

Като подзадачи на така определените основни такива, може да отбележим:

- преглед на проблемната област
- събиране на данни, върху които ще се основават направените препоръки
- анализ на съществуващите алгоритми за препоръчване. Селектиране и разработка на тези от тях, които са подходящи за работа в контекста на музика.
- разработка на уеб приложение, което ползва препоръчващата система като услуга.
- оценка на точността на изградената система.

1.2 Структура на дипломната работа

Дипломната работа е условна разделена на шест глави, съдържанието на всяка, от които може да се обобщи както следва:

Втора глава се съсредоточава върху преглед на препоръчващите системи, техния начин на работа, използвани алгоритми и метрики. Дава се информация за различни съществуващи решения и тяхното развитие през годините.

Трета глава описва функционалните и нефункционални изисквания към системата и илюстрира възможните случаи на нейна употреба.

Четвърта глава илюстрира архитектурата на препоръчващата система и уеб интерфейса, който е изграден към нея. Подчертава как изградената архитектура отговаря на зададените нефункционални изисквания към системата. Прави се обзор на използваните при изработката на приложението технологии и услуги като се посочват техните предимства и недостатъци спрямо другите възможни решения. Дават се някой имплементационни детайли от процеса по реализация.

Пета глава описва използваните методи за оценяване работата на приложението, както и получените резултати.

Шеста глава прави обобщение на свършената в рамките на дипломната работа дейност и описва в каква степен са били постигнати поставените цели и задачи. Дават се насоки за бъдещо развитие.

2. Преглед на проблемната област

В настоящата глава ще направим обзор на препоръчващите системи - тяхната история, основни методи на работа и начини за тяхната оценка. Ще дадем и примери на успешни препоръчващи услуги и ползите от тях.

2.1 История и приложения

Идеята за препоръчващите системи идва от ежедневието, където хората често пъти разчитат на съвет от свои познати, когато трябва да вземат решение - каква стока да закупят, каква книга да прочетат, какви телевизионни програми да следят.

Първата дигитална препоръчваща система - *Tapestry* работи точно на този принцип и се появява в началото на 90-те години като научна разработка в лабораторията на *Xerox* в Пало Алто. Причина за създаването ѝ става увеличения обмен на електронни съобщения между служителите в лабораторията - голяма част от тях били различни обяви, които се пращали до всички служители. Първоначалният опит за справяне с този проблем бил посредством създаване на мейлинг листи - така всеки служител може да се запише в

листите, които го интересуват. На практика, обаче, не било възможно да се създаде подходящо множество от мейлинг листи и този подход се оказал неприложим. Вторият подход, който станал значително по-успешен, бил всеки потребител да създаде свои собствени филтри и да получава само мейлите, които отговарят на тях. Системата позволявала и *сътрудничество* между отделните потребители - всеки може да запише мнението си за даден мейл и да позволи на другите да го използват в своите филтри. Този процес е подобен на *tag-ване* - ако потребителят X , счита че мейлът A е забавен, той може да добави таг *забавен* за A . Потребителят Y може да създаде филтър "всички мейли, които X е отбелязал като забавни" и да получава само тях. Идеята в основата на системата е, че между текущия потребител и всички останали има сходство и, ако разгледаме обектите, които останалите потребители са харесали, може да добием представа какво би харесал и активният потребител. Основният недостатък идва от човешкия елемент - всеки трябва сам да създаде филтри, използвайки таговете на други. За да бъдат оптимални препоръките е необходимо и потребителите, чиито тагове се използват да бъдат подбрани оптимално - нещо, което не е възможно всеки да направи самостоятелно.

Следващите опити за създаване на препоръчващи системи отново идват от научните среди - през 1997г. лабораторията *GroupLens*¹ към университета в Минесота създава проекта *MovieLens*² - система за препоръчване на филми. Принципът, на който тя работи е подобен на *Tapestry* - потребителите получават като препоръки филми, които са харесани от други потребители, с които имат сходни вкусове. За да разбере предпочтенията на нови ползватели, системата ги кара да дадат оценки от 1 до 5 на няколко (около 15) филма. Подобрението се състои в начина на намиране на потребителите, които се ползват за препоръки - *MovieLens* ги намира автоматично вместо да трябва всеки да ги задава сам. Резултатите са положителни - въпреки че е научен проект, *MovieLens* събира над 160,000 потребители, които са оценили над 15млн. филма и продължава да работи и до днес.

След успеха на първите няколко препоръчващи системи в научните среди, бизнесът също започва да се обръща към употребата им. В края на 90-те години онлайн магазинът *Amazon.com* внедрява препоръчваща услуга в сайта си с цел да помогне на потребителите по-лесно да намерят подходящи за тях продукти. Системата позволява на всеки да даде оценка между 1 и 5 на продуктите, които закупи, а впоследствие препоръча обекти, които са били оценени високо от потребителите, с които имаме сходни вкусове. Само няколко години по-късно - през 2006г. - 35% от близо 800-милионните продажби на компанията идват от продукти, които са препоръчани. Препоръчващата система увеличава значително и печалбите на онлайн магазина - тя насочва потребителите към подходящи за тях продукти, които не винаги са нови или търсени и по този начин дава възможност на *Amazon* да продава непопулярни стоки, при които има висока надценка.

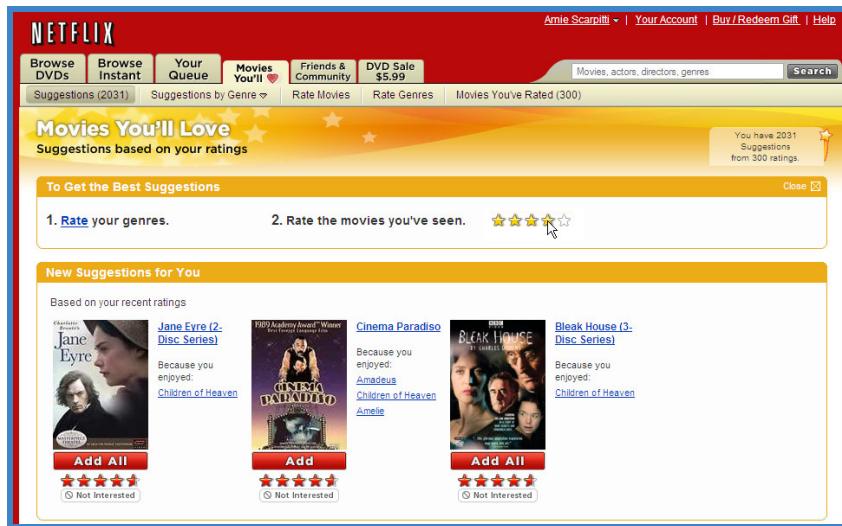
¹ <http://grouplens.org/>

² <https://movielens.org/>



Фигура 2.1. Препоръчани продукти в *Amazon.com*

През 2000г. препоръчваща система започва да използва и друга успешна компания - *Netflix*³. Дейността на *Netflix* е свързана с излъчване на филми и телевизионни сериали по интернет, а препоръчващата им услуга - *Cinematch* - използва оценки дадени от потребителите, за да предлага на зрителите нови предавания, съобразени с техните предпочитания. *Cinematch* постига висока точност на препоръките - над 75% от препоръчаните от системата филми се харесват на потребителите, а над 50% от тях получават максимална оценка от 5 звезди (Wilson, Tracey, 2007). Системата допринася значително и за печалбите на *Netflix* - към 2007г. над 60% от продадените филми са били препоръчани.



Фигура 2.2. Препоръчани филми от системата *Cinematch* в *Netflix*

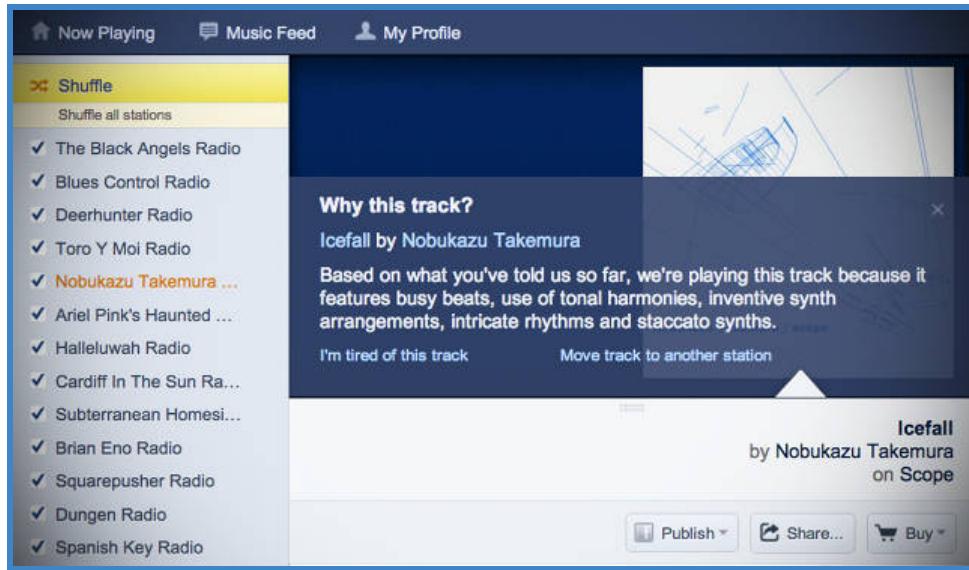
През 2006г. *Netflix* решават да заменят системата си *Cinematch* като обявят конкурс с

³ <https://www.netflix.com/global>

отворено участие, чиято цел е да се създаде система за препоръчване на филми, която има по-добра точност. За целите на конкурса от компанията осигуряват данни за оценките, които са дали над 480,000 потребители за над 18,000 филма. Първият участник, в конкурса, който успее да състави алгоритъм, предсказващ с 10% по-добра точност от *Cinematch* оценката, която потребител би дал на произволен филм, печели награда от 1,000,000 долара. С над 5000 участници (Netflixprize, 2009), конкурса предизвиква сериозен интерес и дава силен тласък на развитието на препоръчващите системи като цяло.

Почти по същото време, когато *Netflix* създават своята препоръчваща система започва работа и друга компания, чийто бизнес сериозно разчита на препоръки. *Pandora*⁴ е първото в света персонализирано онлайн радио. За разлика от разгледаните досега препоръчващи приложения, тя не разчита на оценки от други потребители, а анализира само оценките, които конкретния потребител дава на песните, които слуша. Подходящите за препоръка песни се откриват от системата на база сходство в музикалните качества между харесана песен и останалите такива в библиотеката на *Pandora*. Този подход води до много висока точност на препоръките, но е и силно ограничаващ - необходима е експертна човешка намеса, за да се оценят музикалните свойства на множество (*хиляди*) песни. В рамките на известния като *Music Genome* проект създателите на *Pandora* заедно с над 30 музикални експерти успяват да оценят над 400,000 песни по над 450 различни атрибути (Layton, Julia, 2006). Именно тези песни формират музикалната библиотека на *Pandora*. Макар този метод да дава добри резултати, той е доста времеемък - оценяването на една песен отнема средно около 30 минути (Joyce, John, 2006), поради което препоръките се дават в рамките на относително малко множество продукти. Загубена е и възможността за препоръчване на качествено нови песни - системата може да предложи само мелодии, които са сходни с вече харесани от потребителя такива.

⁴ <http://www.pandora.com/>



Фигура 2.3. Препоръчана песен в онлайн радиото *Pandora*. На потребителя се дава и обяснение защо е получил такава препоръка.

Няколко години след *Pandora* започва работа и друга сходна услуга, която добива сериозна популярност - *last.fm*⁵. Тя позволява на потребителите си да свалят приложение, което автоматично записва информация за всяка прослушана от тях песен, а впоследствие им препоръчва нови песни въз основа на сходство в музикалните им предпочтения с други потребители на услугата. Тук информацията за харесваните песни се дава неявно - потребителят не поставя оценки, а системата си прави извод за неговите предпочтения на база брой прослушвания на конкретна песен.

2.2 Методи на работа

Съществуват голям брой алгоритми, които се ползват при изграждане на препоръчващи системи, сред които се открояват два големи подтипа - препоръчващи алгоритми, основани на *съдържание* (*content-based filtering*) и алгоритми, основани на *сътрудничество* (*collaborative filtering*). Огромната част от съществуващите днес препоръчващи системи работят именно с тяхна помощ. Освен тях, през последните години приложение намират и алгоритми, които се възползват от допълнителна информация за потребителите - географско положение, демографска информация, информация за средата, в която се извършват препоръките и други.

⁵ <http://www.last.fm/>

2.2.1 Препоръки основани на съдържание (content-based filtering)

При препоръки, основани на *съдържание*, препоръчващите системи работят с профили на потребителите и описание на препоръчваните продукти. Обикновено след като потребител се регистрира, системата му задава поредица от въпроси (*какви продукти харесва*), за да добие първоначална информация за него. В течение на времето, потребителят оценява продуктите, които са му предложени и по този начин дава допълнителна информация за интересите си и допълва първоначалния профил. В процеса на работа, системата препоръчва продукти, които са близки до вече харесани от потребителя такива, като пресмята сходство между продуктите и профила му. За целта се използват предварително дефинирани за всички продукти стойности на атрибути (*описание на продукта*).

Таблица 2.1 показва примерен списък с оценки (*по скалата от 1 до 10*), които потребител X е дал на множество кинозаглавия в система за препоръчване на филми. Таблица 2.2 показва предварително създаден списък с филми и атрибут-стойности за тях, които определят степента им на принадлежността (*между 0 и 1*) към различни жанрове (*описание на филмите*). В таблица 2.3 е илюстриран профил на потребител X като за целта въз основа жанровата принадлежност на филмите, които е гледал и оценката, която им е поставил, на всеки жанр е зададено тегло между 0 и 1. Колкото по-голямо е теглото, толкова по-определен е профила на потребител X за този жанр.

Име на филм	Interstellar	The Matrix	Gladiator	The Lion King
Оценка	2	8.5	10	8

Таблица 2.1. Оценени от потребител заглавия в система за препоръчване на филми.

Име на филм \ Жанр	Комедия	Фантастика	Исторически	Драма
Gladiator	0.2	0	1	0.8
Braveheart	0.2	0	1	0.7
Lawrence of Arabia	0	0	1	0.8
WALL·E	0.8	0.3	0	0.1

Таблица 2.2. Предварително създадена информация за всички филми в системата.

Жанр	Комедия	Фантастика	Исторически	Драма
Тегло	0.07	0.01	0.87	0.21

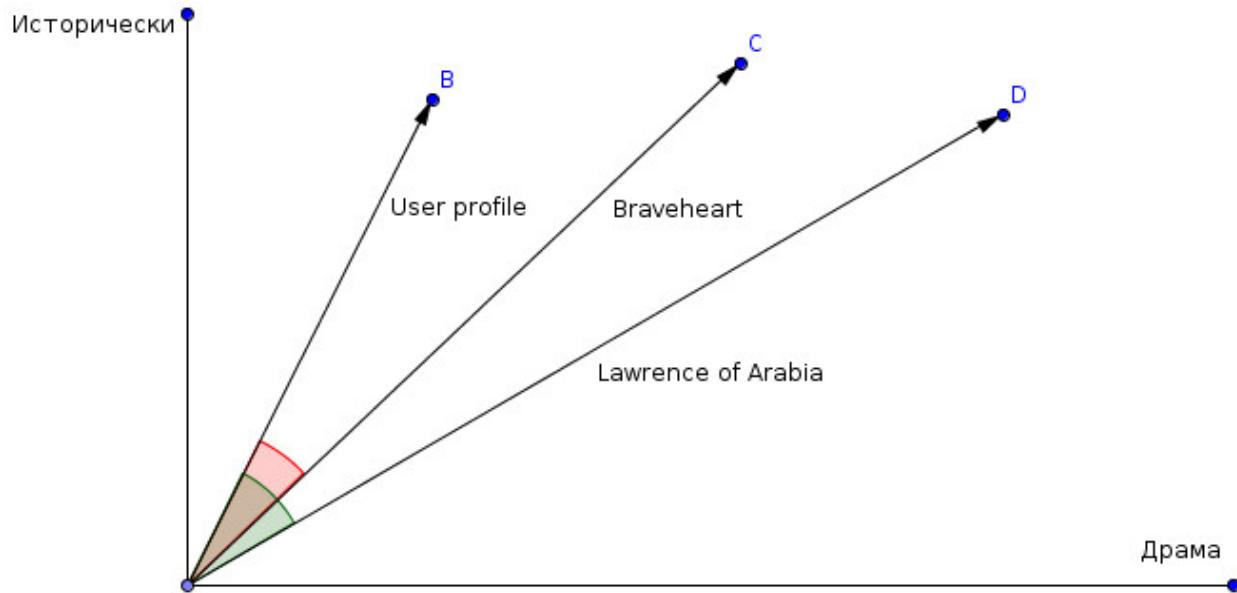
Таблица 2.3. Потребителски профил в система за препоръчване на филми.

Препоръчваща система, основана на съдържание ще изгражда инкрементално профил на потребителя, като за целта обновява теглата, асоциирани с всеки жанр след добавяне на оценка за нов филм. След това ще разгледа списъка с всички филми (*таблица 2.2*), за да намери такива, които отговарят на профила. Именно тази част от последните филми, които все още не са оценени от потребителя ще му бъдат препоръчани. В конкретния пример от профила се вижда, че потребителят харесва исторически филми, а в списъка с всички филми *Lawrence of Arabia* и *Braveheart* принадлежат към този жанр. Тъй като потребителят не е оценил *Lawrence of Arabia* и *Braveheart*, те ще му бъдат препоръчани.

Съществуват различни алгоритми за измерване на сходството между потребителски профил и отделните продукти, които може да се препоръчват. Най-разпространеният сред тях е т.нар. косинусова прилика (*cosine similarity*). При нея потребителският профил и всички продукти се представляват като n-мерни вектори от тегла - (w_1, w_2, \dots, w_n) и се пресметя ъгъла (*чрез неговия косинус*), който сключват вектора на профила и векторите на всеки един от продуктите.

$$\text{similarity}(\text{profile}, \text{item}) = \cos(\vec{w}_p, \vec{w}_i) = \frac{\vec{w}_p \cdot \vec{w}_i}{|\vec{w}_p| \cdot |\vec{w}_i|} = \frac{\sum_{j=1}^n w_{jp} w_{ji}}{\sqrt{\sum_{j=1}^n w_{jp}^2} \sqrt{\sum_{j=1}^n w_{ji}^2}}$$

Колкото по-малки са ъглите между векторите, толкова по-голямо е сходството между профил и продукт.



Фигура 2.4. Двумерна проекция на векторите на потребителския профил и филмите *Braveheart* и *Lawrence of Arabia*

2.2.1.1 Препоръки основани на тагове

През последните години все по-популярни стават интернет услуги, които позволяват на потребителите си да качват съдържание, свободно да го анотират с произволен текст и след това да го споделят с останалите. От своя страна другите потребители също могат да добавят свои анотации и т.н. Множеството от тези тагове задава един вид класификация на въпросните обекти и би могла да се използва при изготвянето на препоръки. Един от начините това да се направи е следният:

- за всеки от потребителите на системата се изготвя профил на основа таговете (*анотациите*) на обектите, които е харесал
- за всеки от обектите в системата се изготвя профил, базиран на таговете (*анотациите*), които е получил от потребителите
- препоръчват се тези обекти, чийто профил е най-близък до профила на разглеждания потребител (съгласно разгледаните в глава 2.2.1 метрики и подходи)

2.2.2 Препоръки основани на сътрудничество (collaborative filtering)

Препоръчването, основано на сътрудничество е най-разпространеният и изучаван препоръчващ подход от създаването му от *Paul Resnick* и *Hal Varian* (Resnick, Paul and Varian, Hal R., 1997) през 1997г. насам. От своя страна то се дели на два подтипа - *memory-based* (основано на памет) и *model-based* (създаващи модели). Първият тип е значително по-широко разпространен и от своя страна се дели на два подтипа - *user-based* и *item-based* системи, основани на сътрудничество.

2.2.2.1 *Memory-based* сътрудничество

При този тип сътрудничество е характерно, че системата запомня оценките, които потребителите дават на продуктите, а впоследствие изчислява степента на сходство между отделните потребители или продукти с помощта на тези оценки. Препоръчват се продукти, сходни с такива, които текущият потребител е оценил високо или такива, които са получили висока оценка от сходни потребители. Използват се два основни алгоритъма за *memory-based* сътрудничество - *user-based* (основано на намиране най-сходни потребители) и *item-based* (основано на намиране на най-сходни продукти)

2.2.2.1.1 Сътрудничество основано на потребители

Идеята зад сътрудничеството, основано на потребители (*user-based collaborative filtering*) е да се намерят сходни потребители - ако двама души са дали приблизително еднакви оценки на едни и същи продукти, то системата счита, че те имат еднакъв вкус. Продуктите, които се препоръчват са такива, които не са оценени от текущия потребител, но са били високо оценени от други потребители, с които той има сходство. В същността си този подход е еквивалентен на намиране на N -те най-близки съседи на потребител въз основа на предварително зададена метрика. Изборът на N е ключов за точността на препоръчващата система - прекалено голяма стойност би довела до нерелевантни препоръки, а прекалено ниска - към тесен кръг от препоръчани продукти. Повечето комерсиални системи ползват стойности в диапазона 15 - 50.

Таблица 2.4 показва примерен списък с оценките, които потребителите в система за препоръчване на филми са дали. Потребителите *B* и *C* са дали сходни оценки на всички филми, но *B* не е оценил *The Matrix*. Система, работеща чрез сътрудничество, основано на потребителите би препоръчала на *B* филма *The Matrix*, тъй като ще предположи, че потребителите *B* и *C* имат еднакви вкусове.

потребител \ фильм	The Matrix	Gladiator	The Lion King	Wall-E
A	6	7	7	8
B	?	10	4	1
C	10	10	3	1
D	5	5	6	6

Таблица 2.4. Потребителски профил в система за препоръчване на филми.

2.2.2.1.2 Изчисляване на сходство и предсказване на оценката на потребител при сътрудничество основано на потребители

Общийят метод на работа на система, работеща чрез *user-based* сътрудничество е, както следва:

- изчислява сходството между текущият потребител и всички останали
- формира списък от N най-близки съседи на текущия потребител
- въз основа степента на сходство с N -те съседи и оценките, които те са дали формира предсказания за оценките, които текущият потребител би дал на продуктите, които все още не е оценил
- препоръчва K -те продукта, които имат най-високи предсказани оценки, сортирани в низходящ ред по стойностите на тези оценки

Следвайки стъпките на работа на такъв алгоритъм възниква въпросът как да пресметнем сходството $s(u, v)$ между текущия потребител u и произволен друг потребител v в системата. Метриките, които се ползват в повечето случаи за тази цел са следните:

Корелация на Пирсън. Този подход пресмята сходството $s(u, v)$ като използва корелацията на Пирсън между оценките, които потребителите са дали за продукти, които и двамата са оценили. Корелацията се изчислява, както следва (Resnick, Paul and Varian, Hal, 1997):

$$s(u, v) = \frac{\sum_{i \in I_u \cap I_v} (r_{u,i} - \bar{r}_u)(r_{v,i} - \bar{r}_v)}{\sqrt{\sum_{i \in I_u \cap I_v} (r_{u,i} - \bar{r}_u)^2} \sqrt{\sum_{i \in I_u \cap I_v} (r_{v,i} - \bar{r}_v)^2}}$$

където $r_{x,i}$ е оценката, която потребителят x е дал на продукта i , \bar{r}_x е средната оценка дадена от потребителя x , а I_x е множеството от всички продукти, които са оценени от x . Възможните стойности за така пресметнатото s са между -1 (обратна зависимост между u и v - пълна липса на сходство) и 1 (пълно сходство). Изваждането на средната оценка, която потребителят е дал при пресмятане на корелацията решава често срещан при препоръчващите системи проблем - различните скали на оценяване, които ползват потребителите. Корелацията на Пирсън ще бъде коректна дори единият потребител да използва целия диапазон на наличната оценъчна скала (1-10), а другият само подмножество от нея (например 4-7). Като основен недостатък на тази метрика може да посочим, че пресмятана високо сходство между потребителите, които имат малък брой общи продукти (например, ако потребителите имат само един общ продукт, но са го оценили еднакво, то корелацията им би била 1). Един от начините да се преодолее този проблем е като се постави праг K на минималния общ брой продукти, които трябва да имат двама души и да се умножи получената по гореизложените изчисления корелация с:

$$\min\left(\frac{|I_u \cap I_v|}{K}, 1\right)$$

Рангова корелация на Спирман. Този подход пресмята сходството $s(u, v)$ като намира рангова корелация на Спирман между оценките, които потребителите са дали. Тя се изчислява както следва (Spearman, Charles, 1904):

$$s(u, v) = \frac{6 \sum_{i \in I_u \cap I_v} (rank(r_{u,i}) - rank(r_{v,i}))^2}{n(n^2 - 1)}$$

където $n = |I_u \cap I_v|$ е броят на продуктите, които са били оценени както от u , така и от v , а $rank(r_{u,i})$ и $rank(r_{v,i})$ изчисляваме по следната процедура:

- съставяме таблица с две колони $r_{u,i}$ и $r_{v,i}$ - съответните оценки, които са дали потребителите u и v за всеки тяхен общ продукт.
- сортираме в нарастващ ред по стойностите на $r_{u,i}$ редовете на таблицата и добавяме трета колона $rank(r_{u,i})$, стойностите, в която съответстват на позицията на $r_{u,i}$ в сортирания ред.
- сортираме в нарастващ ред по $r_{v,i}$ стойностите на редовете на таблицата и добавяме четвърта колона $rank(r_{v,i})$, стойностите, в която съответстват на позицията на $r_{v,i}$ в сортирания ред.
- при равни стойности на оценките $r_{x,i}$ и $r_{x,i+1}$ за $rank(r_{x,i}) = rank(r_{x,i+1})$ се определя средно-аритметичната стойност на позициите на двете в сортирания ред на оценките.

Косинусова прилика. При този подход всеки потребител се представя като вектор с размерност $|I|$, където I е множеството от всички продукти в системата. На позиция j в този вектор стои оценката, която потребителят е дал на j -тия продукт или 0, ако такава оценка няма. Сходството $s(u, v)$ намираме, като пресметнем ъгъла, който векторите на двата потребителя сключват (на по-малък ъгъл съответства по-голямо сходство). Този ъгъл намираме чрез неговия косинус и скаларното произведение на двета вектора (Resnick, Paul and Varian, Hal R., 1997):

$$s(u, v) = \frac{\vec{r}_u \cdot \vec{r}_v}{|\vec{r}_u| |\vec{r}_v|} = \frac{\sum_{i \in I} r_{u,i} r_{v,i}}{\sqrt{\sum_{i \in I} r_{u,i}^2} \sqrt{\sum_{i \in I} r_{v,i}^2}}$$

При тази метрика не се отчита възможна разлика в скалите за оценяване, които потребителите използват. Възможно е това да се коригира като от стойностите $r_{u,i}$ и $r_{v,i}$ се извадят средните оценки, които u и v са дали на всички продукти. В този случай косинусовата прилика би дала същия резултат като корелацията на Пирсън.

След изчисляване на сходството между текущия потребител и всички останали остава да се разгледат N -те негови съседа с най-висока степен на сходство и на база техните оценки да се предскажат стойностите на оценките $P_{u,i}$, които текущия потребител u би дал на

неоценените от него продукти. За целта обикновено се ползва формулата:

$$p_{u,i} = \bar{r}_u + \frac{\sum_{u' \in N} s(u, u')(r_{u',i} - \bar{r}_{u'})}{\sum_{u' \in N} |s(u, u')|}$$

Препоръчват се продуктите с най-висока стойност на $p_{u,i}$ сортирани в низходящ ред по тази стойност.

2.2.2.1.3 Сътрудничество основано на продукти

Сътрудничеството основано на продукти (*item-based collaborative filtering*) е създадено от Amazon (Linden, Greg, Smith, Brent and York, Jeremy, 2003), които първоначално са ползвали *user-based* алгоритъма, но се е оказало, че той е неприложим в машабите на техния онлайн магазин - с над 240 miliona потребители не е възможно да се пресметне сходството между един потребител и всички останали в реално време. Решението било то да се изчислява предварително, но *user-based* подходът е подчертано динамичен - какво ще бъде препоръчано зависи не само от оценките, дадени от текущия потребител, но и от тези, дадени от всички останали - неща, които постоянно се изменят. От друга страна, профилите на продукти се променят значително по-бавно, особено, ако продуктът има множество оценки.

Препоръчваща система работеща с *item-based collaborative filtering* се основава на предположението, че продукти, които са получили еднакви оценки от едни и същи потребители са сходни и, че нов потребител също би им дал еднакви оценки. Такава система изчислява сходство между всички продукти и препоръчва на потребителите такива, които са сходни с тези, които вече са харесали / оценили високо.

В таблица 2.5 се вижда, че филмите *Gladiator* и *The Matrix* получават близки оценки от потребителите. Потребителят *B* е оценил с максимална оценка *Gladiator*, но не е оценил *The Matrix*, поради което системата ще предположи, че *B* би оценил и него с максимална оценка и ще му го препоръча.

потребител \ фильм	The Matrix	Gladiator	The Lion King	Wall-E
A	6	7	7	8
B	?	10	4	1
C	10	10	3	1
D	5	5	6	6

Таблица 2.5. Потребителски профил в система за препоръчване на филми.

2.2.2.1.4 Изчисляване на сходство и предсказване на оценката на потребител при сътрудничество основано на продукти

Общийят метод на работа на система, работеща чрез *item-based* сътрудничество е както следва:

- изчислява сходството между всички двойки продукти
- за всеки продукт, който е получил висока оценка от текущия потребител се намира списък с неговите N най-близки съседи. Те се дават като препоръка сортирани в низходящ ред по степента на сходство.

Първата стъпка на алгоритъма изисква да се пресметне близост между двойка продукти i и j - $s(i, j)$. За целта би могло да се ползват някои от следните метрики:

Косинусова прилика. Косинусовата прилика е най-разпространеният подход при *item-based collaborative filtering*. При него продуктите се представят като n -мерни вектори от оценките, които са получили и се пресмята тъгълът между векторите. По-малък тъгъл съответства на по-висока степен на сходство.

$$s(i, j) = \frac{\vec{r}_i \cdot \vec{r}_j}{|\vec{r}_i| |\vec{r}_j|} = \frac{\sum_k r_{ik} r_{jk}}{\sqrt{\sum_k r_{ik}^2} \sqrt{\sum_k r_{jk}^2}}$$

Условна вероятност. В някои системи за пресмятане степента на сходство $s(i, j)$ се ползва условна вероятност

$$s(i, j) = P(j \in B | i \in B) = \frac{P(i \cap j)}{P(i)}$$

За целта се пресмята броя на потребителите дали висока оценка на i и на тези дали високи оценки, както на i , така и на j . Разделяйки така получените стойности на общият брой потребители оценили i и j получаваме и търсената степен на сходство (*условна вероятност*).

2.3 Сравнение между препоръки основани на сътрудничество и препоръки основани на съдържание

Системите препоръчващи на основа съдържание и тези основани на сътрудничество имат добри и лоши страни. Ползите от употребата на препоръчващи услуги, работещи чрез съдържание включват (Burke, Robin, 2002):

- независимост от потребител - тези препоръчващи системи работят само с оценките, които текущият потребител е дал, т.е. дори системата да има единствен потребител, той би получил релевантни препоръки.
- прозрачност на работата - препоръките дадени от такъв тип системи могат да бъдат

обяснени - тъй като всеки продукт е бил предложен, защото има точно определени качества (зададени в описанието му) е възможно на потребителя да се обясни точно защо получава дадена препоръка.

- работа с нови продукти - препоръчващите системи, основани на съдържание могат да препоръчат, както отдавна съществуващи продукти, така и новодобавени, които все още никой потребител не е оценил.

Недостатъците на препоръчването основано на съдържание са:

- ограничено съдържание - за да може един продукт да бъде препоръчен е необходимо той да има подходящо описание. Често пъти такива описания се създават ръчно, поради което не е възможно системата да работи с голям брой продукти.
- ограничени препоръки - такива препоръчващите системи не могат да дават неочеквани препоръки. Системата препоръча само продукти, които отговарят на зададен профил, поради което на потребителя ще се препоръчат само обекти, подобни на тези, които вече е оценил. Например, ако потребителят е оценил само филми на *Франсис Форд Копола*, то ще му се предлагат само филми от този режисьор.
- нови потребители - ако нов потребител се регистрира в системата, тя няма да има никаква информация за неговия профил и не би могла да му препоръча нищо.

От друга страна препоръчването основано на сътрудничество е по-добро в следните направления:

- изненадващи препоръки - *collaborative filtering* приложенията могат да ни препоръчат продукти, които нямат нищо общо с досегашната ни потребителска история, но които да ни харесат, т.е. може да получим качествено нови препоръки. Това качеството на препоръчващите системи се нарича *serendipity*.
- богато съдържание - *collaborative filtering* приложенията могат да дават препоръки в рамките на огромен брой продукти, стига системата да има достатъчно потребители, които да оценяват.

Недостатъците при този вид препоръки се изразяват преди всичко в:

- работа с нови продукти - тъй като новите продукти са оценени от много малък брой хора те ще се препоръчат значително по-рядко от стари продукти, които имат много оценки
- работа с нови потребители - ако в системата се появи нов потребител, той не би могъл да получи никакви препоръки, тъй като не може да се прецени сходство със съществуващите потребители.
- скалируемост - бързодействието при този алгоритмичен подход пряко зависи от

броя на потребителите в системата. Тъй като при всяка заявка за препоръки е необходимо да се изчисли сходството между текущия потребител и всички други, то при голям брой потребители препоръчващата услуга би работила незадоволително.

- “*черни овце*” - този проблем се състои в наличието на потребител, чийто вкус не би могъл да се причисли към този на нито един от останалите ползватели на системата. Такива потребители не биха могли да получат никакви препоръки.
- рейтинг атаки - тъй като препоръките, които получава даден потребител пряко зависят от оценките, които всички останали са давали, то е възможно някои потребители да се опитат да промотират продукти в системата, с които те са обвързани по някакъв начин (например търговци да рекламират продавани от тях стоки). За целта такива хора обикновено поставят високи оценки на техните продукти и ниски на тези на техни конкуренти.

2.4 Хибридни препоръчващи подходи (*hybrid recommendation systems*)

С цел постигане на по-добри резултати някои препоръчващи системи комбинират различни подходи от препоръчването основано на съдържание и основано на сътрудничество като по този начин избягват някои ограничения и проблеми на обикновените препоръчващи системи. Тези приложения се наричат хибридни и обикновено работят по някой от следните начини (Burke, Robin, 2002):

Хибридизация чрез претегляне. При този алгоритъм предсказаната оценка на даден продукт се формира на база отделните оценки, които са предсказали всички препоръчващи подходи, които се ползват в приложението. Най-широко ползваният метод е да се вземе линейна комбинация от тези оценки и в процеса на работа на услугата да се подобрят теглата, които се дават на отделните алгоритми. Такава схема се ползва в препоръчващата система *P-Tango* (Burke, Robin, 2002), която ползва хибриден препоръчващ подход, комбиниращ *content-based* и *collaborative filtering*. Всеки от тези подходи пресмята оценката, която текущият потребител би дал на определен продукт, след което се взима средно-претеглено от стойностите на оценките. В началото на работата на системата теглата, които се дават и на двата подхода са по $\frac{1}{2}$, но с течение на времето се променят въз основа обратната връзка (оценки), които потребителите дават на препоръчаните продукти.

Хибридизация чрез редуване. Системата отново комбинира няколко препоръчващи подхода, но за всяка препоръка се избира точно един от всички подходи и препоръките се правят единствено от него. Подобен алгоритъм се ползва в препоръчващото приложение *DailyLearner* (Burke, Robin, 2002), което използва комбинация от *content-based* и *collaborative filtering*. Системата се опитва винаги да прави препоръки въз основа на

съдържание, но ако няма достатъчно данни за потребителя или счита, че препоръките ще бъдат с ниска точност, прави препоръките си въз основа на сътрудничество.

Хибридизация чрез смесване. При този алгоритъм системата комбинира няколко препоръчващи подхода, всеки от които дава независими препоръки. На потребителя се показват заедно комбинираните резултати, препоръчани от всеки от ползваните подходи. Такъв алгоритъм се ползва в приложението *PTV* (Smyth, Barry and Cotter, Paul, 2000), което препоръчва телевизионни предавания. То ползва *content-based* препоръчващ метод въз основа описанията на предаванията и *collaborative* препоръки, работещи с оценките, които отделните потребители дават на програмите, които са гледали.

Хибридизация чрез комбиниране на атрибути. Този тип препоръчващи системи работят с препоръчване, основано на съдържание като използват данни, извлечени чрез *collaborative* алгоритми за характеристики на продуктите. Услугата *Ripper* (Burke, Robin, 2002), която препоръчва филми използва този начин и чрез него е постигнала значително по-добри резултати от чист *collaborative* подход. В нея към описанието на всеки филм е била добавена и информация “*Подобен на*”, която е извлечена от предпочитанията на отделните потребители.

Каскадна хибридизация. При този подход препоръчването се осъществява на етапи. През първия етап се използва *collaborative* стратегия, която дава първоначално множество от препоръчани продукти. Тези продуктите се подават като вход на *content-based* алгоритъм от втория етап на препоръчване - той филтрира неподходящите и генерира крайното множество препоръчани обекти. Система, която ползва този подход е *EntreeC* (Burke, Robin, 2002), която препоръчва ресторани.

2.5 Оценяване работата на препоръчващи системи

За да бъдат полезни препоръчващите системи е важно те да дават релевантни съвети на потребителите, тъй като в противен случай употребата им би имала негативен ефект. Разработени са множество методи, които да оценяват различните аспекти в работата на едно препоръчващо приложение. Те се разделят на два големи подтипа - *offline* оценяване (без интеракция с потребител) и *online* оценяване (оценяване на системата в реални условия с участието на истински потребители).

2.5.1 Офлайн оценяване

Офлайн оценяването се извършва без реалното участие на потребител, само с помощта на вече налични данни - информация за потребител и оценки, които те са дали на различните препоръчвани от системата продукти. Това множество данни се разделя на две части - обучаващо и тестово. Системата използва данните от обучаващото множество и се опитва да предскаже оценките, които потребителите биха дали за продуктите в

теството множество. Точността се измерва въз основа отклонението на предсказанията спрямо реалните данни от теството множество.

Често използван подход при онлайн оценяване е данните, които имаме да се разделят на K подмножества (т.нар. K -fold кросвалидация). От тези K множества се избира едно, което служи за тествово, а останалите $K - 1$ са обучаващи. Точността на системата се измерва при тази конфигурация, след което процесът се повтаря още $K - 1$ пъти като всеки път се избира различно тествово множество. След като са пресметнати съответните метрики за всеки от опитите обикновено техните стойности се комбинират в единствена крайна оценка посредством взимане на (претеглено) средно-аритметично.

Онлайн оценяването е полезно, защото дава относително релевантни резултати и в същото време е бързо и лесно за извършване. Недостатъците му са, че не включва интеракция с реалните потребители на системата и съответно не можем да сме напълно уверени в качеството на резултатите, които ни дава.

2.5.2 Онлайн оценяване

При онлайн оценяването системата дава препоръки на реални потребители на основа тяхната потребителската история и/или профил. Веднъж получили препоръките, потребителите дават обратна връзка (чрез оценки) за тяхното качество. Приложението записва информация за направените препоръки и получената обратна връзка и сравнява тези данни с предсказаните от него. Получените разлики се ползват за изчисляване точността на препоръките.

Онлайн оценяването е доста по-скъпо и бавно, тъй като изиска работа с потребители, но е задължително за оценяване работата на услугата в реални условия.

2.6 Метрики за оценяване точността на препоръчваща система

Както при онлайн, така и при офлайн оценяването системата получава информация за разликата между предсказаниите от нея стойности за оценки на продукти и тези, които реално са получени. Остава въпросът как да превърнем тези разлики в оценка за качеството на работа на системата. За целта се използват няколко типа метрики.

2.6.1 Метрики за точност на предсказаниите оценки

Този тип метрики измерват до колко оценките, предсказани от приложението съвпадат с тези, които реално са били дадени от потребителите. Те показват както в каква степен направените от системата препоръки са релевантни, така и до колко те са били представени на потребителя в правилния ред (продуктите, които би харесал най-много да се покажат първи). Метрики, които се ползват с такава цел са:

Средна абсолютна грешка (Mean Absolute Error - MAE). Средната абсолютна грешка се изчислява въз основа разликите между предсказани и реални оценки на продуктите по следната формула:

$$MAE = \frac{1}{|B_i|} \sum_{b_k \in B_i} |r_i(b_k) - p_i(b_k)|$$

където с B_i сме означили множеството от всички продукти, оценени от потребителя α_i , $r_i(b_k)$ е оценката поставена от този потребител на продукта b_k , а $p_i(b_k)$ е оценката, която е била предсказана от системата за продукта b_k и потребител α_i .

Средна квадратична грешка (Mean Squared Error - MSE). Средната квадратична грешка е вариация на средната абсолютна грешка, при която по-големите отклонения в предсказана и реална оценка се наказват по-тежко от останалите. Тя се изчислява със следната формула::

$$MSE = \frac{1}{|B_i|} \sum_{b_k \in B_i} (r_i(b_k) - p_i(b_k))^2$$

където означенията са както при пресмятане на средна абсолютна грешка.

2.6.2 Метрики за коректност на препоръките

Този тип метрики игнорират точната разлика между реална и предсказана оценка на продуктите, а се фокусират само върху релевантността на препоръките. Т.е. дали те действително отразяват интересите на потребителя, дали продукт, за който е предсказана максимална оценка е харесан (макар и не оценен максимално) или е напълно отхвърлен. Най-често ползваните метрики за тази цел са точност (*precision*) и откриване (*recall*).

Точността измерва каква част от всички продукти, които са били препоръчани са релевантни (харесани от потребителя).

$$Precision = \frac{|B_{rs}|}{|B_s|}$$

В горната формула B_{rs} е подмножеството от тези препоръчани продукти, които потребителят е харесал, а B_s е множеството от всички препоръчани продукти.

Откриването (*recall*) показва каква част от всички релевантни продукти са били препоръчани.

$$Recall = \frac{|B_{rs}|}{|B_r|}$$

Тук B_{rs} е множеството от всички препоръчани продукти, които потребителят счита за релевантни, а B_r е множеството от всички релевантни продукти.

Използването на тези метрики изисква всички продукти, които са препоръчани да се

класифицират като релевантни и нерелевантни. Обикновено това се прави като се постави някакъв праг на оценките (например 5 при оценъчна скала между 1 и 10) и всички продукти с оценка над този праг се считат за релевантни, а всички с по-ниска оценка се считат за нерелевантни.

Точността и откриването са взаимно свързани (повишената точност води до ниско откриване), поради което в практиката често се ползват метрики, които отчитат и двете. Най-популярната сред тях е т. нар. F-метрика

$$F_{\alpha} = (1 + \alpha) \frac{Precision \cdot Recall}{\alpha \cdot Precision + Recall}$$

която обикновено се ползва със стойност $\alpha = 1$. Нейната стойност съвпада с претегленото средно хармонично от *precision* и *recall*.

Друг подход е да се изследва стойността на средната точност (*Mean Average Precision - MAP*). Тя се пресмята по формулата:

$$MAP = \frac{1}{|B_{rs}|} \sum_{b \in B_{rs}} Precision(b)$$

Недостатък на така описаните метрики е, че се пресмятат върху всички препоръчани на потребителя обекти. Често пъти в практиката се налага да се оцени качеството само на подмножество от направените препоръки - първите N такива. За целта се използват стойностите на *Precision@n* и *Recall@n*, които се пресмятат, както следва:

$$Precision@n = \frac{|B_{rs}@n|}{n}$$

$$Recall@n = \frac{|B_{rs}@n|}{|B_r|}$$

2.6.3 Метрики за коректност на наредбата на препоръките

Обикновено при правене на препоръки се цели продуктите, които системата счита, че ще бъдат харесани в най-голяма степен от потребителя да му бъдат предложени най-напред. За да се следи този аспект от работата на препоръчващите системи обикновено се ползва някоя от следните метрики:

Разстояние между наредбите (Distance Performance Measure - DPM). Тази метрика е била предложена за първи път от авторите (Zhou, Bing and Yao, Yiyu, 2009) и измерва разстоянието между наредбата \succ_u , която потребителят би дал на препоръчаните продукти и реалната наредба \succ_s зададена от системата. Разстоянието се измерва в термините на подреждането на отделни двойки продукти. За да изследваме начина на работата на тази метрика ще въведем следните дефиниции:

- ще казваме, че две наредби се *съгласяват* върху продуктите d и d' , ако и двете

подреждат d и d' в един и същи ред.

- ще считаме, че две наредби си **противоречат**, ако едната слага d преди d' , а другата d' преди d .
- две наредби ще наричаме **съвместими**, ако едната дава на d и d' една и съща позиция, а другата подрежда единия продукт преди другия.

За всяко множество от препоръчани продукти означаваме броят на продуктите, върху които наредбата на потребителя и наредбата дадена от системата се *съгласяват*, броят на продуктите, върху които двете си *противоречат* и броят на продуктите, върху които са *съвместими*, както следва:

$$\begin{aligned} C^+ &= |\succ_u \cap \succ_s| \\ C^- &= |\succ_u \cap \succ_s^c| = |\succ_u^c \cap \succ_s| \\ C^0 &= |\succ_u \cap \tilde{s}| + |\tilde{u} \cap \succ_s| \end{aligned}$$

Ще считаме, че разстоянието между две наредби по отношение на двойка продукти е 0, ако те се съгласяват върху продуктите, 1, ако са съвместими по отношение тези продукти и 2, ако си противоречат. Тогава разстоянието между наредбата дадена от потребителя и тази дадена от системата се измерва, както следва:

$$\beta(\succ_u, \succ_s) = 2C^- + C^0$$

Обезценена кумулативна печалба (Discounted Cumulative Gain). Тази метрика оценява качеството на подредбата на списък с продукти и е функция от полезнотта на всеки продукт за потребителя и неговата позиция в списъка. Обезценената кумулативна печалба на списък, състоящ се от P продукта се пресмята, както следва:

$$DCG_p = rel_1 + \sum_{i=1}^P \frac{rel_i}{log_2 i}$$

където rel_i е релевантността на i -тия продукт за потребителя, т.е. оценката, която той е получил.

Алтернативна формула за пресмятане, която поставя по-голяма тежест върху връщането на релевантни резултати е:

$$DCG_p = \sum_{i=1}^P \frac{2^{rel_i} - 1}{log_2(i + 1)}$$

Тъй като стойността на тази метрика пряко зависи от броя на препоръчаните продукти P , то в практиката често пъти се ползва нейн нормализиран вариант, което позволява да се сравняват стойностите получени при препоръки, дадени на различни потребители.

Стойността на тази нормализирана версия се пресмята по формулата:

$$nDCG_p = \frac{DCG_p}{IDCG_p}$$

където $IDCG$ (*Ideal Discounted Cumulative Gain*) представлява стойността на DCG пресметната върху списък с P продукта, който е сортиран в намаляващ ред по тяхната релевантност.

2.6.4 Покритие на препоръките (Coverage)

Освен точност на работата на системата обикновено се следят и група от други критерии. Един от тях е покритие на препоръките. Покритието измерва частта от всички продукти, за които системата може да предположи каква оценка би дал даден потребител. Това е необходимо, тъй като не винаги приложението може да препоръча даден продукт - например, когато такъв не е бил оценен нито веднъж и препоръките ползват сътрудничество, то не е възможно да се предскаже оценката, която текущият потребител би му дал.

Измерват се два типа покритие - покритие на предсказанията (*prediction coverage*) и покритие на каталога (*catalogue coverage*). Първото измерва частта от всички продукти, за които системата може да предскаже оценката на даден потребител. *Catalogue coverage* измерва каква част от всички продукти в системата никога са били препоръчвани на даден потребител. Високо ниво на покритие на препоръките означава, че препоръчващата услуга би работила успешно в повече ситуации. То обаче е пряко свързано и с точността на им - би могло да се работи с висока степен на покритие, ако се дават случаини предложения, но това значително би понижило точността.

Обикновено *coverage* се измерва като се вземе случаинно множество от потребители и продукти от базата данни на приложението и се поискат препоръки за тях. По този начин може да се пресметне както *prediction coverage*, така и *catalogue coverage*.

2.6.5 Сигурност на препоръките (Confidence)

Препоръките направени от една система имат две направления - сила (*strength*) и сигурност (*confidence*). Силата характеризира в каква степен потребителят ще хареса препоръчания продукт. Сигурността показва до колко система е уверена в точността на направената препоръка.

Съществуват различни подходи, чрез които препоръчващите услуги се справят с нивата на сигурност в препоръките. Някои системи не дават на потребителите препоръки, чиито нива на сигурност са под някакъв prag. Други показват ниво на сигурност заедно с всяка направена препоръка. И в двата случая, обаче, е важно тази метрика да се измерва и да се анализира нейната корелация с точността на препоръките и количеството данни в приложението.

2.6.6 Разнообразие

Разнообразието от продукти в списък с препоръчани такива е друг аспект, който е важен за потребителите и затова трябва да се изследва. Например, потребител, който е дал висока оценка на филма *The Godfather* може да получи като препоръки други филми от режисьора *Франсис Форд Копола* и да хареса и тях. Въпреки това, потребителят може да не е доволен от цялостната работа на препоръчващата услуга, тъй като получава еднообразни препоръки. Разнообразието в направените от системата предложения може да се измерва посредством вътрешно-списъчна метрика за близост (*ILS - intra-list similarity metric*), която се изчислява, както следва:

$$ILS(L_{\alpha_i}) = \frac{\sum_{x \in L_{\alpha_i}} \sum_{y \in L_{\alpha_i}, x \neq y} sim(x, y)}{|L_{\alpha_i}|(|L_{\alpha_i}| - 1)}$$

където L_{α_i} е множеството от всички продукти, които са били препоръчани на потребителя α_i , а $sim(x, y) : B \times B \rightarrow [-1, 1]$ е функция, която на всяка двойка продукти съпоставя оценка на сходството помежду им.

2.6.7 Доверие

Друга важна характеристика на работата на една препоръчваща система е доверието, с което тя се ползва сред своите потребители, т.е. в каква степен потребителите вярват в релевантността на препоръките, които получават. Тази характеристика е съществена, тъй като приложението може да дава напълно теоретично и алгоритмично обосновани предложения, които обаче да изглеждат странно на потребителите и те да не са доволни от тях. Един начин за справяне с този проблем е за всички препоръчани обекти да се дават съответни обяснения защо са били предпочетени от системата.

Този показател обикновено се измерва като се проведе анкета с реални потребители на препоръчващата услуга и се разгледат техните отговори.

2.6.8 Полезност

Полезнота на направените предложения е друг съществен критерий за оценка на работата на препоръчващо приложение. Той се отнася до бизнес стойността, която препоръчващата услуга носи на организацията, която я е създала, т.е. дали направените от нея предложения водят до повишени продажби и печалби. Всяка система трябва да се стреми към постигане на баланс между стойностите на този параметър и тези на точността и доверието на потребителите. Това е така, тъй като е възможно да се препоръчват само продукти, които имат висока търговска стойност за собствениците на препоръчващата услуга, но това може значително да понижи потребителско доверие и интерес към

системата.

2.6.9 Адаптивност

Важна точка от работата на всяко препоръчващо приложение е и неговата адаптивност към промени в потребителските интереси или в глобалните тенденции в дадена област. Типичен пример за важността на този критерий се проявява при препоръчването на новини - новинарските статии са информационни обекти, към които потребителите обикновено проявяват интерес само в кратък интервал от време (в общия случай непосредствено около деня, в който се е случило дадено събитие). Възможно е обаче отдавна написани публикации да станат внезапно интересни във връзка с друга новина - например, ако някъде по света има случай на цунами би било подходящо системата да се адаптира към популярността на тази новина и да започне да препоръчва на потребителите статии, обясняващи този природен феномен (макар и написани отдавна).

Съществено е и препоръчващата услуга да може да се приспособява към промени в интересите на потребителя - възможно е някой дълго време да е давал високи оценки на определен тип продукти, но вече по някаква причина да не е доволен от тях. В този случай приложението трябва да засече тази промяна и да направи съответна корекция в даваните предложения.

Работата на системата по този показател обикновено се измерва в термините на разлика между препоръчаните обекти преди и след добавяне на допълнителна информация за потребители.

2.6.10 Новост

Новостта е мярка за това каква част от предложените обекти не са били познати на потребителя преди интеракцията му с препоръчващата система.

Един начин да се измери този параметър е като се проведе анкета с потребителите, в която те да кажат каква част от предложените им продукти са им били непознати. Друга възможност е да се подхodi посредством онлайн метод като наличните данни се разделят на две части по отношение на някакъв момент във времето. В такъв случай на системата се забранява да използва всички данни получени след него, както и случайна малка част от данните, събрани преди това. При тестване се дава висока оценка, когато се препоръчват обекти, които са били харесани от потребителя и са оценени след разделящия момент и ниска, когато се предлагат обекти, харесани от потребителя преди този момент.

И при двата разгледани подхода, обаче, е важно освен новост в препоръките да се изследва и точността, тъй като високи нива на новост могат да бъдат лесно постигнати за сметка на ниска точност.

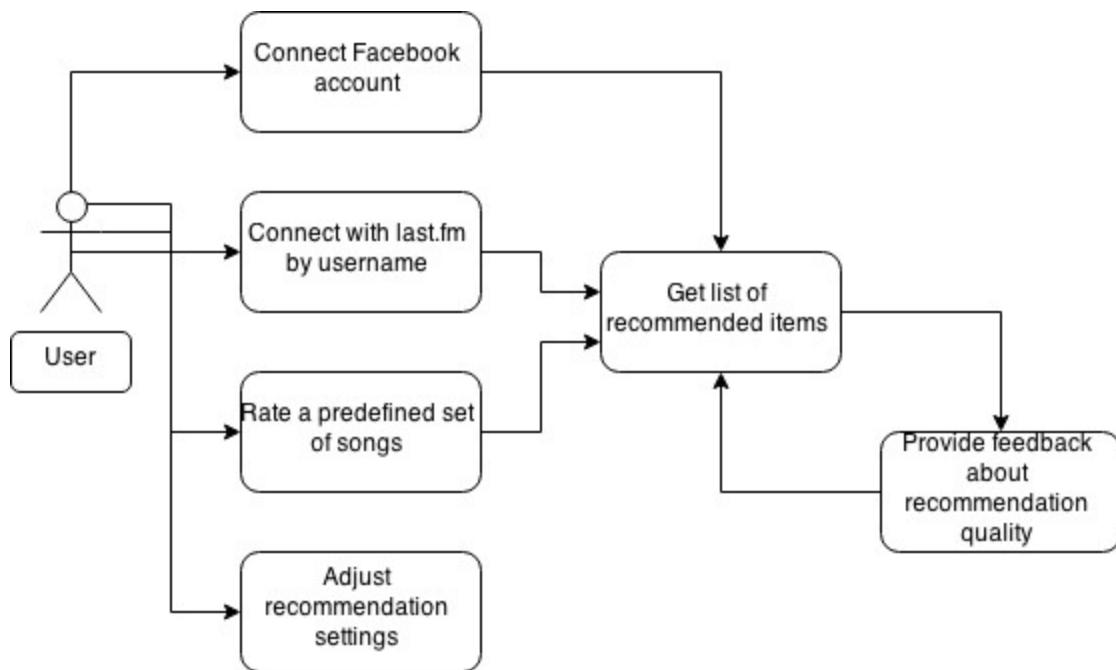
3. Анализ на изискванията

В тази глава ще разгледаме подробно изискванията към изгражданата препоръчваща система за музика. Ще разделим тези изисквания най-общо на две части - функционални и нефункционални. Първите са свързани с определянето на свойствата на системата, които са видими за потребителя, докато вторите се отнасят към нейните качествени характеристики - т.е. свойства, които са важни за нейната цялостната работа, макар и потребителят да не разчита на тях директно.

3.1 Концептуален модел

Основното действащо лице в модела на препоръчващата система е потребителят.

Фигура 3.1 илюстрира главните случаи на употреба на приложението от неговата гледна точка, както и основните функции изпълнявани от препоръчващата система.



Фигура 3.1. Начини на употреба на приложението от страна на потребителя

За да получат персонализирани препоръки, потребителите трябва да въведат по един или друг начин данни за музикалните си предпочитания. Поддържат се следните източници на такива данни:

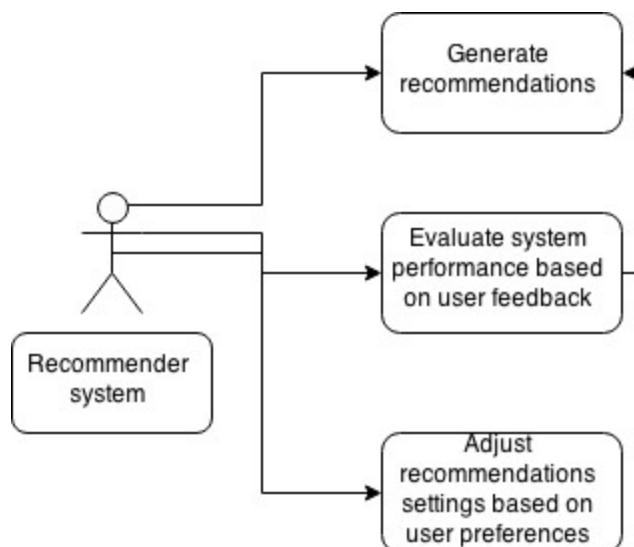
- *Facebook* - потребителите могат да позволят на препоръчващата услуга да използва данните от техния *Facebook* профил. Тогава приложението ще изтегли информация за харесаните от тях музикални изпълнителни и те ще послужат като основа за

работата на препоръчващия алгоритъм.

- *Last.fm* - потребителят може да въведе името на своя акаунт в *last.fm*, при което системата ще извлече списък с неговите прослушани песни. На базата на тези песни ще бъдат изгответи съответните препоръки.
- оценяване на набор от предефинирани песни - ако потребителят не желае да свърже своите *Facebook* и/или *Last.fm* профили или пък в тях няма достатъчно информация, то ще му бъде предложено да оцени (по скалата от 1 до 5) предварително зададен списък от популярни песни. Дадените оценки ще послужат като основа за даване на персонализирани музикални предложения.

След като получат препоръки потребителите могат да дадат оценка на тяхната точност, а благодарение на тази обратна връзка системата ще генерира нови, по-добри такива.

Освен потребителя, другият основен актьор в приложението е самата препоръчваща система. От нейна гледна точка сценариите за работа са както следва:

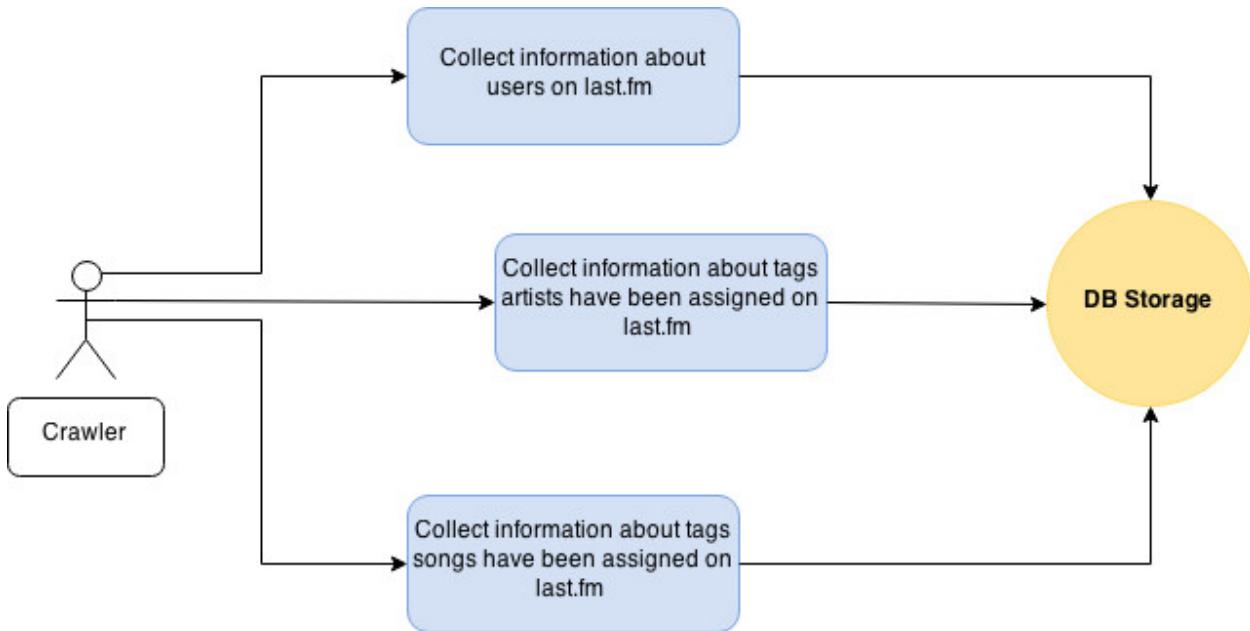


Фигура 3.2. Сценарии за работа на препоръчващата система

Накратко, основните действия на системата са следните:

- настройва се спрямо зададените от потребителя преференции за работата на алгоритъма. Ако такива не са зададени изрично, то се ползват стойности по подразбиране.
- прави препоръки, базирани на предоставените от потребителя данни.
- запазва данните за оценките дадени от потребителя на препоръчените песни и ги използва при даване на нови, по-оптимални препоръки, а също и при оценяване работата на приложението.
- запазва информация за вече дадените на потребителя препоръки, за да се избегнат повторения.

Третата част от приложението е т.нр. обхождаща (*crawling*) част, която отговаря за събирането на подходящи данни, които да се ползват като база за работата на препоръчващите алгоритми. Работата на тази част от приложението може да се обобщи чрез следната диаграма:



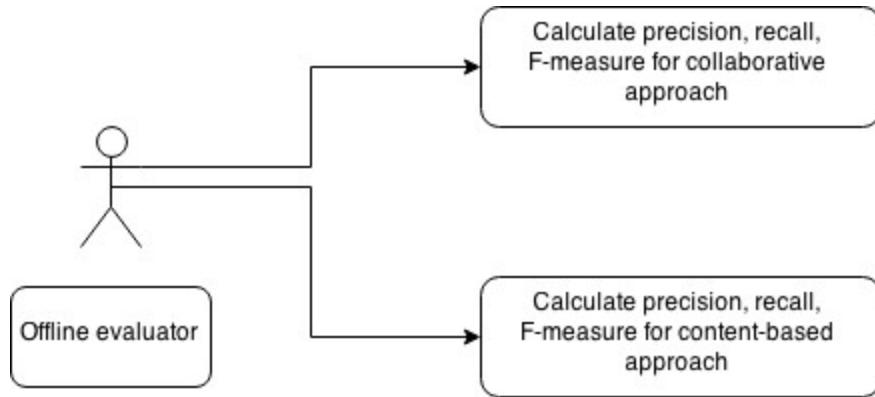
Фигура 3.3. Сценарии за работа на crawler-а за събиране на данни за работата на системата

Действията на *crawler*-а се изчерпват със следните стъпки:

- събиране на информация за музикалните предпочтения на множество потребители на сайта *last.fm*.
- събиране на информация за анотациите, с които различни изпълнители са били обозначени от потребители на *last.fm*.
- събиране на данни за анотациите, с които множество от песни са били обозначени от потребителите на *last.fm*

Основното ограничение при неговата работа е свързано с необходимостта да се събере достатъчно количество данни, така че даваните препоръки да бъдат максимално релевантни и представителни. Като допълнително изискване може да се постави и бързото събиране на информация - тъй като ще се работи с голям брой обекти е необходимо всеки да се извлича за максимално кратко време, като се прави и паралелно извличане на данни за множество обекти едновременно.

Последният модул от препоръчващата система отговаря за нейното оценяване в офлайн условия - т.е. без интерфейсът към нея да работи и в отсъствието на реални потребители. Неговото функциониране се представя чрез следната фигура:



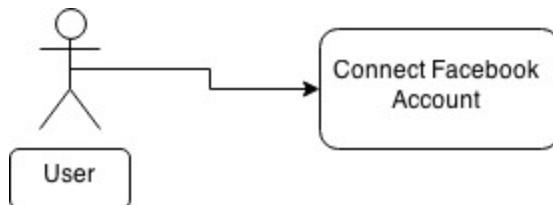
Фигура 3.4. Сценарии за работа на модула за онлайн оценяване на системата

Като изискване към работата на този модул отново може да поставим той да работи оптимално от гледна точка на време - тъй като оценяването изискава да се дадат препоръки за голям брой потребители би било непрактично модулът да не обработва техните данни паралелно. Освен това е необходимо изчислените по време на неговата работа стойности да бъдат показани в удобен вид на администраторите на системата.

3.2 Функционални изисквания - основни случаи на употреба

В рамките на тази част ще разгледаме основните случаи на употреба (т.нар. *use cases*) на препоръчващата система. Към всеки такъв случай ще приложим списък с последователност от асоциирани действия, както и възможни извънредни ситуации при изпълнение.

3.2.1 Свързване на Facebook акаунт



Фигура 3.5. Свързване на потребителски профил във Facebook

Описание: Потребител, който има профил във Facebook свързва този профил като източник на информация за работата на препоръчващата услуга

Предусловие: Потребителят е регистриран като потребител във Facebook.

Постъпково описание:

1. Потребителят отваря началната страница на приложението
2. Потребителят натиска бутона “Login with Facebook”
3. Потребителят попълва детайлите за своя Facebook акаунт в появилия се прозорец.

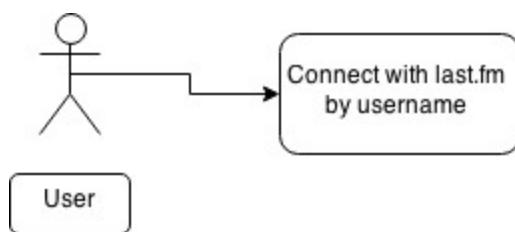
4. *Facebook* валидира въведените данни.
5. *Facebook* показва на потребителя диалог обясняващ позволенията, които приложението изисква за достъп до неговите данни.
6. Потребителят оторизира препоръчвашата услуга да получи достъп до информацията за харесаните от него обекти във *Facebook*.
7. Системата получава код за достъп до данните на потребителя
8. Използвайки кода за достъп се събира информация за всички харесани от потребителя музикални обекти.
9. Системата дава персонализирани препоръки

Алтернативен поток на изпълнение:

- В стъпка 4 *Facebook* сървърите установяват, че потребителят е предоставил грешни данни за вход. Показва се съобщение за грешка и се дава възможност за повторно въвеждане на данни.
- В стъпка 6 потребителят отказва да оторизира приложението за достъп до своите данни. Той остава на началната страница на сайта и не получава препоръки.
- В стъпка 8 се оказва, че потребителят не е харесал във *Facebook* никакви музикални изпълнители. Тогава той получава списък с предварително зададени препоръчани песни.

Резултат: Системата получава данни за музикалните предпочитания на потребителя и му показва препоръки.

3.2.2 Свързване на Last.fm потребителско име



Фигура 3.6. Свързване на потребителски профил във Last.fm

Описание: Потребител, който има профил в *last.fm* свързва този профил като източник на информация за работата на препоръчвашата услуга

Предусловие: Потребителят е регистриран като потребител в *last.fm*.

Постъпково описание:

1. Потребителят отваря началната страница на приложението
2. Потребителят натиска “Link *last.fm*” бутона
3. Потребителят въвежда името на своя *last.fm* акаунт в появилото се поле
4. Системата изтегля информация за прослушаните от него песни в *last.fm*
5. Посредством получената в предната стъпка информация се изготвят

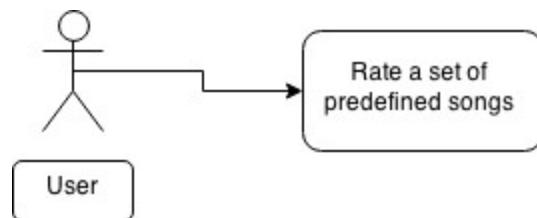
персонализирани препоръки

Алтернативен поток на изпълнение:

- В стъпка 3 потребителят въвежда несъществуващо потребителско име. Системата не може да изтегли данни и прави неперсонализирана препоръка - предлага списък с предварително зададени популярни песни
- В стъпка 4 се оказва, че потребителят не ползва активно *last.fm* и профилът му там не предоставя необходимата за работата на приложението информация. В този случай отново се прави неперсонализирана препоръка - предлага се списък с предварително зададени популярни песни

Резултат: Системата изтегля данни за потребителя от *last.fm* и ги използва, за да прави препоръки.

3.2.3 Оценяване на предварително зададен списък с песни



Фигура 3.7. Оценяване на предварително зададен списък с песни

Описание: Потребител оценява списък с предварително зададени песни, за да даде на препоръчващата система начална информация за предпочитанията си.

Предусловие: Няма.

Постъпково описание:

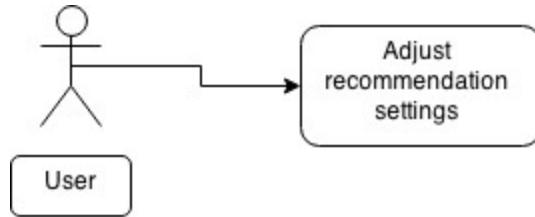
1. Потребителят отваря началната страница на приложението.
2. Потребителят натиска "Rate" бутона.
3. Потребителят вижда списък с 15 предварително зададени популярни песни.
4. Потребителят оценява по скалата от 1 до 5 поне 5 от тези песни.
5. Потребителят натиска "Submit" бутона.
6. Потребителят получава персонализирани препоръки.

Алтернативен поток на изпълнение:

- В стъпка 4 потребителят оценява по-малко от 5 песни и натиска "Submit" бутона. На екрана се извежда съобщение за грешка - потребителят трябва да оцени най-малко 5 песни, за да бъдат дадени смислени препоръки.

Резултат: Системата дава на потребителя препоръки, основани на информацията, която е получила от оценяването на списъка с песни.

3.2.4 Настройка на параметрите на препоръчващия алгоритъм



Фигура 3.8. Настройка на параметрите на препоръчващия алгоритъм

Описание: Потребител променя стойностите на параметрите, с които работи препоръчващия алгоритъм с цел получаване на по-добри препоръки.

Предусловие: Няма.

Постъпково описание:

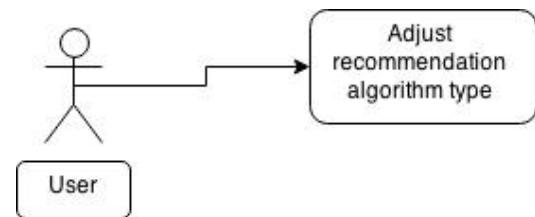
1. Потребителят отваря началната страница на препоръчващото приложение
2. Потребителят натиска бутона “*Settings*” от навигационното меню
3. Потребителят въвежда нови данни за стойността на броя на съседите, които се разглеждат.
4. Потребителят натиска бутона “*Save*”, за да изпрати на сървъра обновените настройки

Алтернативен поток на изпълнение:

- В стъпка 3 потребителят въвежда некоректни данни - например текст вместо число за стойност на броя на най-близките съседи. В този случай ще се използва стойността по подразбиране за този параметър при изготвяне на препоръки.

Резултат: Системата обновява стойността на съответния параметър на препоръчващия алгоритъм и ще я използва при последващи обръщения към него.

3.2.5 Настройка на типа на препоръчващия алгоритъм



Фигура 3.9. Настройка на типа на използвания препоръчващ алгоритъм

Описание: Потребител променя типа на използвания препоръчващ алгоритъм с цел получаване на по-добри препоръки.

Предусловие: Няма.

Постъпково описание:

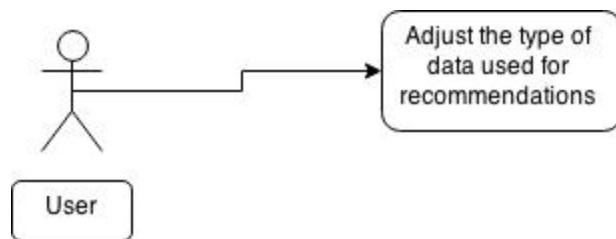
1. Потребителят отваря началната страница на препоръчващото приложение
2. Потребителят натиска бутона “*Settings*” от навигационното меню

3. Потребителят избира типа на използвания препоръчващ подход чрез съответния радио бутон.

Алтернативен поток на изпълнение: Няма.

Резултат: Приложението обновява типа на препоръчващия алгоритъм и ще го използва при изготвяне на следващи препоръки за съответния потребител.

3.2.6 Настройка на типа на използваните при препоръка данни



Фигура 3.10. Настройка на типа на данните, които се ползват при препоръки

Описание: Потребител променя типа на данните (изпълнители / песни), които да се ползват при търсене на сходство между него и останалите потребители, за които системата има информация.

Предусловие: Потребителят е изbral препоръчващата услуга да работи посредством *collaborative filtering*.

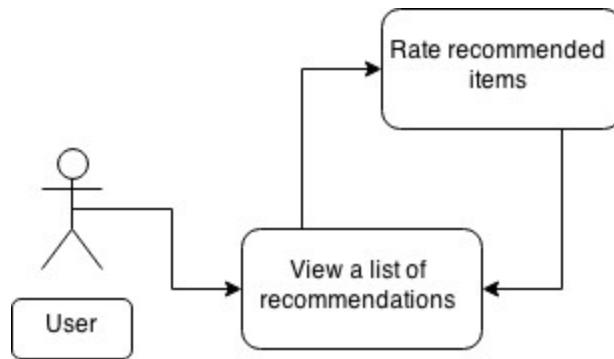
Постъпково описание:

1. Потребителят отваря началната страница на препоръчващото приложение
2. Потребителят натиска бутона “*Settings*” от навигационното меню
3. Потребителят избира типа използваните при препоръка данни посредством радио бутон. Възможните стойности са две - песни (*tracks*) и изпълнители (*artists*).

Алтернативен поток на изпълнение: Няма.

Резултат: Системата обновява типа на данните, които ще се ползват от препоръчващия алгоритъм и ще го използва при изготвяне на следващи препоръки за съответния потребител.

3.2.7 Преглед на персонализирани препоръки



Фигура 3.11. Преглед на получени от системата музикални препоръки

Описание: Потребител преглежда списък с песни, които са му били препоръчани.

Предусловие: Потребителят е позволил на системата да се свърже с неговия *Facebook* или *last.fm* профил или е оценил предварително зададен списък с песни.

Постъпково описание:

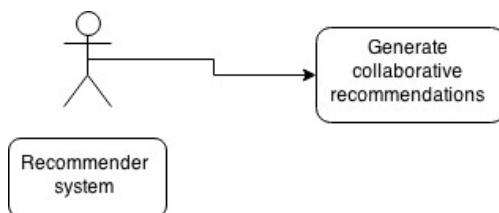
1. Потребителят свързва своя *Facebook* или *last.fm* акаунт или оценява предварително зададен списък с песни
2. Потребителят получава списък с персонализирани препоръки
3. Потребителят оценява песните от този списък
4. Потребителят натиска бутона “*Submit*”
5. Потребителя получава за разглеждане нов списък с персонализирани препоръки

Алтернативен поток на изпълнение:

- В стъпка 3, потребителят оценява по-малко от 5 от предложените му песни и натиска бутона “*Submit*”. Получава съобщение за грешка - за да получи нови препоръки е необходимо да оцени поне 5 от вече препоръчаните му песни.
- След стъпка 3, потребителят не натиска бутона “*Submit*”. При това положение той не получава нов списък с препоръки и остава на текущата страница с такива.

Резултат: Системата прави препоръки на потребителя, той ги разглежда и потенциално дава обратна връзка за тяхното качество.

3.2.8 Генериране на персонализирани препоръки чрез сътрудничество



Фигура 3.12. Генериране на препоръки, базирани на сътрудничество от системата

Описание: Системата намира списък от песни, които са подходящи за текущия потребител след като той е направил заявка за това.

Предусловие: Потребител е направил заявка към приложението да му се препоръчат подходящи песни. От настройките е избрано да се ползва препоръчване чрез сътрудничество.

Постъпково описание:

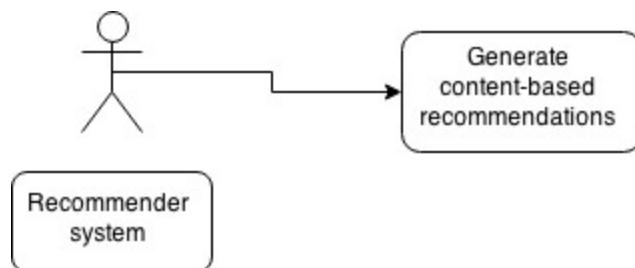
1. Потребителят подава данни за себе си към системата (посредством оценяване на песни или *Facebook/last.fm* интеграция)
2. Системата получава данните и въз основа на тях пресмята сходство между текущия потребител и потребителите, за които е запазила информация в своята база
3. Намира се списък от K на брой потребителя, с които активният има положителна степен на сходство
4. Избират се само първите N потребителя от тези K , за които коефициентът на сходство е най-висок.
5. Сред всички песни на тези N съседа системата намира тези, които потребителят не е оценил и пресмята каква оценка би им дал. Приложението избира 25-te песни с най-висока изчислена оценка и ги препоръчва.
6. Клиентската част получава като отговор списъка с препоръчани песни

Алтернативен поток на изпълнение:

- В стъпка 3 се оказва, че данните получени от потребителя не са достатъчни, за да се намерят сходни потребители. В този случай препоръчващата система прави препоръка по подразбиране, която се състои от 15 предварително зададени популярни песни.

Резултат: Клиентската част на препоръчващата система получава списък с препоръчани за потребителя песни и му ги показва.

3.2.9 Генериране на персонализирани препоръки чрез съдържание



Фигура 3.13. Генериране на препоръки, базирани на сътрудничество от системата

Описание: Системата намира списък от песни, които са подходящи за текущия потребител след като той е направил заявка за това.

Предусловие: Потребител е направил заявка към приложението да му се препоръчат подходящи песни. От настройките е избрано да се ползва препоръчване, основано на

съдържание.

Постъпково описание:

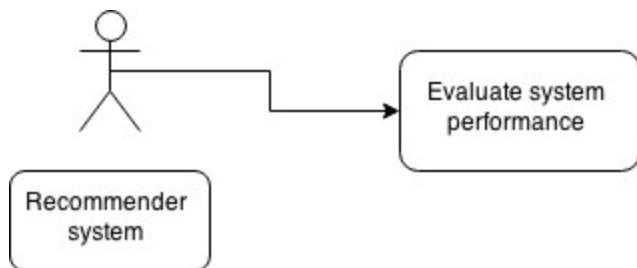
1. Потребителят подава данни за себе си към системата (посредством оценяване на песни или *Facebook/last.fm* интеграция).
2. Системата получава данните и въз основа на тях създава профил на потребителя.
3. Намира се сходство между профила на потребителя и профилите на всички песни, за които е събрана информация.
4. Използвайки това сходство, препоръчващата услуга пресмята оценките, които потребителят би дал на всяка от неоценените от него песни.
5. Избират се 25-те песни с най-висока изчислена оценка и се препоръчват.
6. Клиентската част получава като отговор списъка с препоръчани песни.

Алтернативен поток на изпълнение:

- В стъпка 3 се оказва, че данните получени от потребителя не са достатъчни, за да се намерят подходящи песни. В този случай препоръчващата услуга прави препоръка по подразбиране, която се състои от 15 предварително зададени популярни песни.

Резултат: Клиентската част на приложението получава списък с препоръчани за потребителя песни и му ги показва.

3.2.10 Онлайн оценяване качеството на направените препоръки



Фигура 3.14. Онлайн оценяване точността на работата на системата

Описание: Потребителят дава обратна връзка на системата за точността на направените препоръки като ги оценява. Системата пресмята метрики, характеризиращи тази точност и ги запазва за съхранение.

Предусловие: Потребител дава обратна връзка за работата на препоръчващата система като оценява препоръчаните обекти.

Постъпково описание:

1. Потребителят получава списък с препоръчани обекти.
2. Потребителят дава оценка по скалата между 1 и 5 на всички или поне част от тези обекти.
3. Системата получава дадените от потребителя оценки.
4. Пресмятат се метриките *precision@25*, *precision@10*, *precision@5* за релевантността на направените препоръки и *discounted cumulative gain* за точността на наредбата

им.

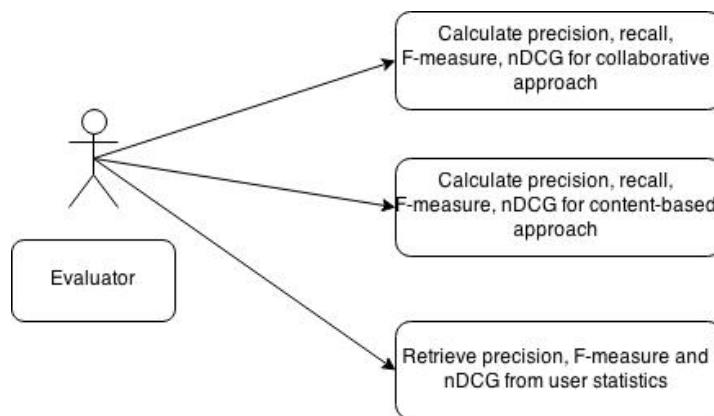
5. Пресметнатите стойности се съхраняват за бъдещи справки.

Алтернативен поток на изпълнение:

- в стъпка 5 възниква грешка - не е възможно в базата данни да се запише исканата информация - пресметнатите метрики за този потребител не се запазват.

Резултат: В базата се записва информация за качеството на правените от системата препоръки. Тази информация може да се използва, за да се анализира работата на различните препоръчващи подходи и да се правят подобрения по приложението.

3.2.11 Офлайн оценяване качеството на направените препоръки



Фигура 3.15. Офлайн оценяване точността на работата на системата

Описание: Администратор на системата стартира модула за оценяване. Той събира данните за всички потребители от базата и ги разделя на обучаващо и тестово множество посредством *3-fold* кросвалидация. Използва тези данни, за да пресметне различни метрики, характеризиращи работата на системата. Като опция от командния ред се подават типа и параметрите на препоръчващия алгоритъм, който да се оцени.

Предусловие: Няма.

Постъпково описание:

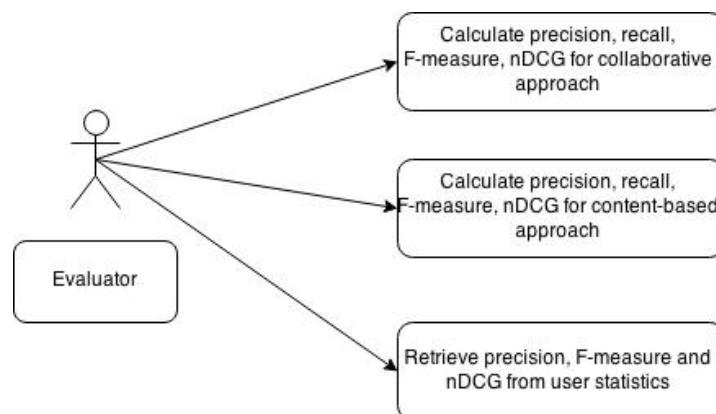
1. Администратор на системата стартира модула за оценяване като подава нула или повече от следните опции:
 - a. *--collaborative* или *-cl*
 - b. *--content-based* или *-cb*
 - c. *--tracks* или *-t*
 - d. *--neighbours* или *-n*
2. Модулът извлича данните за всички потребители, изпълнители и песни в системата
3. Получените данни се разделят на обучаващо и тестово множество
4. Данните от обучаващите множество се използват за получаване на препоръки.
Пресмята се разликата между реалните данни от тестовото множество и

предложените от системата обекти, като при изготвянето на препоръките са използвани подадените от командния ред опции.

Алтернативен поток на изпълнение: Няма.

Резултат: В текстов файл се записва информация за качеството на правените препоръки. Тази информация може да се използва, за да се анализира работата на различните препоръчващи подходи и да се правят подобрения по приложението.

3.2.12 Агрегиране на потребителските оценки за работа на системата



Фигура 3.16. Офлайн оценяване точността на работата на системата

Описание: Администратор в системата стартира модула за оценяване. Той извлича от базата данни вече пресметнатите стойности за метриките *precision* и *nDCG* за отделните потребители, които са ползвали приложението. Модулът агрегира тези данни и записва получените резултати в текстов файл.

Предусловие: Няма.

Постъпково описание:

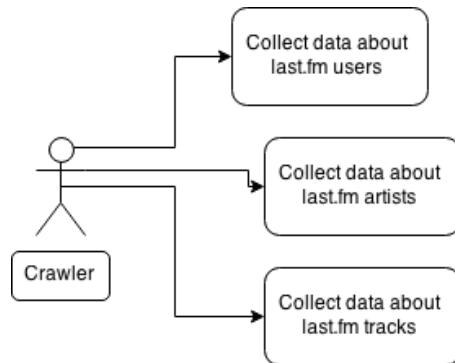
1. Администратор стартира модула за оценяване. При неговото стартиране се подава като опция от командния ред *-o* или *--online*
2. Модулът извлича от базата данни информация за стойностите на метриките *precision@25*, *precision@10*, *precision@5* и *nDCG*, които са били пресметнати в резултат от интеракцията на потребителите с препоръчващото приложение
3. Извлеченните резултати се агрегират като се намират средно-аритметичните стойности на съответните метрики върху всички потребители, за които са събрани данни.
4. Така получените резултати се записват в текстов файл за по-нататъшна обработка и анализ

Алтернативен поток на изпълнение: Няма.

Резултат: В текстов файл се записва информация за качеството на правените препоръки. Тази информация може да се използва, за да се анализира работата на различните

препоръчващи подходи и да се правят подобрения по приложението.

3.2.13 Събиране на данни от last.fm



Фигура 3.17. Събиране на данни от last.fm

Описание: Администратор в системата стартира модула за събиране на данни от *last.fm*. В зависимост от опцията, която е била подадена при стартиране от командния ред модулът започва да извлича информация за потребители, песни или изпълнители.

Предусловие: Няма.

Постъпково описание:

1. Администратор в системата стартира модула за събиране на данни, използвайки за целта командния ред. При стартиране подава една от следните опции:
 - a. `--users` или `-u`
 - b. `--tracks` или `-t`
 - c. `--artists` или `-a`
2. В зависимост от подадената опция системата обхожда множество потребители/песни/изпълнители и събира информация за тях
3. Процесът по събиране на данни продължава безкрайно, докато не бъде принудително прекратен от администратор.

Алтернативен поток на изпълнение:

- в стъпка 1 не се подава опция при стартиране на модула. В такъв случай изпълнението продължава по същия начин, както когато е подадена опцията `--users`.

Резултат: В базата се събират данни за съответните потребители/изпълнители/песни от *last.fm*

3.3 Нефункционални изисквания

Изискванията от този тип се отнасят не до това какво може да прави системата, а по-скоро към това какви качества притежава тя, как трябва да изпълнява задачите си.

Основните ограничения от тази гледна точка към препоръчващото приложение за музика са следните:

Време за реакция/бързодействие. Препоръките, които генерира системата се правят в отговор на потребителска заявка и това означава, че те трябва да бъдат изготвяни максимално бързо, т.ч. да не е необходимо потребителят да чака. Допустимо време за отговор на заявка за препоръки е рамките на няколко секунди.

Използваемост. Ключов фактор за успеха на всяка софтуерно приложение е наличието на добре изглеждащ и удобен за ползване потребителски интерфейс. Интерфейсът трябва да използва компоненти, които са познати на потребителя от работата му с други подобни системи и уеб сайтове като цяло. Компонентите, които стапират изпълнението на важни за работата на приложението задачи (бутони/меню-та) трябва да бъдат максимално видими и лесни за достъп от страна на потребителя.

Лесна поддръжка/изменяемост. Важно за успеха на системата е да може да се добавят лесно нови източници на данни за нейната работа. Към днешна дата огромна част от потенциалните ѝ потребители имат *Facebook* и/или *last.fm* профили, но е възможно в бъдеще това да се промени или пък да се появят социални мрежи, които предоставят по-добра информация за потребителите си от гореизброените. Необходимо е да има и слой на абстракция над използваните при препоръчване алгоритми и метрики, така че да бъде максимално лесно те да бъдат променени или изцяло заменени. По-този начин ще е възможно, ако се установи, че качеството на работа на приложението не е достатъчно добро да се реагира максимално бързо.

Преносимост. Интернет потребителите използват множество от браузъри, почти всеки от които работи със свой собствен *rendering engine*. Това води до разлики в начините, по които едни и същи визуални компоненти изглеждат под различните браузъри, което от своя страна влошава цялостното потребителско изживяване. Затова е важно да се подсигури, че графичния интерфейс към препоръчващата система изглежда еднакво под възможно най-голям брой уеб клиенти и операционни системи. Сходно изискване е в сила и към сървърната част - необходимо е тя да може да работи под различни операционни системи - Windows, различни Linux дистрибуции и MacOS.

Скалируемост. Предвид експерименталните цели на изготвяния прототип се предполага, че той ще работи с малък брой реални потребители. Все пак е важно той да бъде изграден по начин, който би позволил употребата му от голям брой хора с възможно най-минимални промени.

Зашита на личните данни. Препоръките, които системата дава се основават на сходство в интересите на текущия потребител и други потребители и/или продукти, за които е събрана информация. В този ред на мисли е необходимо потребителите на препоръчващата услуга да не получават никаква информация за другите потребители, от които са дошли техните препоръки. Също така, не бива да е възможно те да извлечат информация за музикалните предпочитания на някои от останалите потребители.

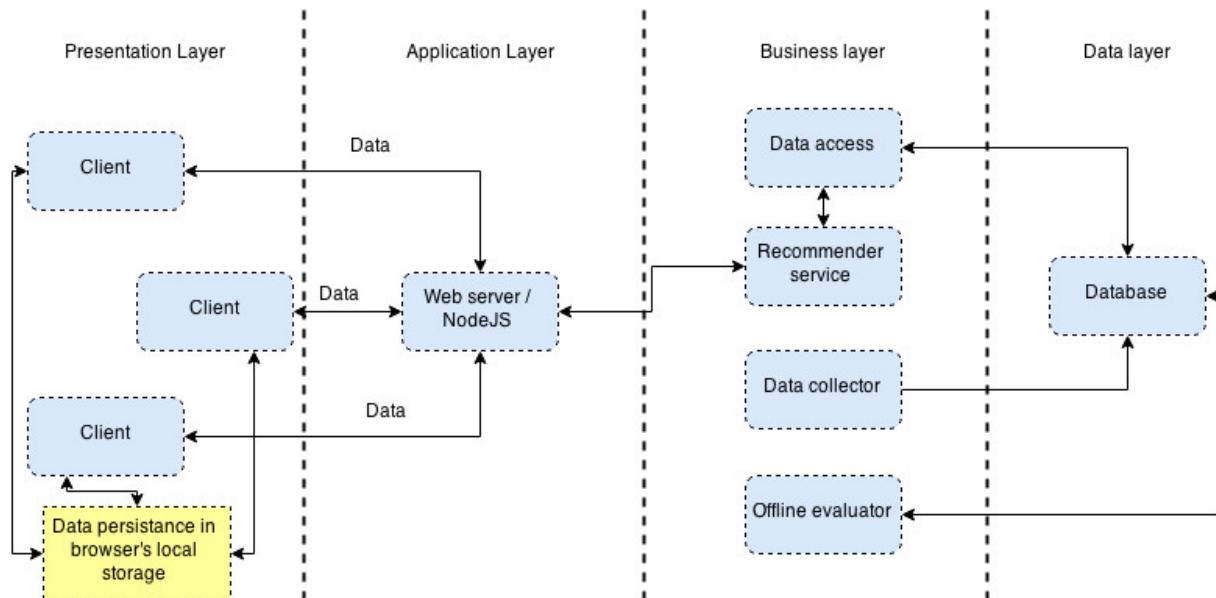
Отворен код. В днешно време все повече проекти се пускат с отворен код, особено в платформи, които се радват на широк потребителски интерес като *Github*, *GoogleCode* и *BitBucket*. Отварянето на кода на проекта позволява привличането както на нови разработчици по него, така и на нови потребители, които да дадат важна обратна връзка.

4. Проектиране, архитектура и реализация на препоръчващата система

Системата работи с често използваната в обектно-ориентираните системи 4-слойна архитектура (Papagelis, Manos, 2012), като отделните елементи са както следва:

- клиентски слой - още наричан *Presentation layer* - частта от системата, която потребителят вижда и, с която той работи директно.
- сървърен слой - още наричан *Application layer* - този дял от приложението, който получава клиентските заявки.
- препоръчващ слой - още познат като *Business layer* - слойт от системата, който извършва действителните препоръки.
- база данни - още наричан *Data-Access layer* - елементът от приложението, който отговаря за съхраняване на събраната информация.

Фигура 4.1 показва общия вид на архитектурата на изградената препоръчваща услуга за музика, както и информация за работата на всеки слой.



Фигура 4.1. Обща архитектура на препоръчващата система

Клиентът показва графичния интерейс на потребителите и се свързва с уеб сървисите в сървърната част, когато това е необходимо. Тази връзка е двупосочна - от една страна

клиентът изпраща на сървъра данните и заявките, получени от потребителя, а от друга - сървърът връща желаната от клиента информация.

Сървърната част служи като медиатор между клиентите и препоръчващата система. Тя приема заявките от потребителите, обработва ги и от своя страна прави заявки към препоръчващата услуга. Когато препоръките са готови, сървърната част получава отговор от препоръчващата система и изпраща готовите данни на потребителя.

Препоръчващата услуга получава данните за заявката, която потребителят е направил в подходящ, обработен формат. Прави заявка към услугата за достъп до данни, за да събере информация за вече запомнени потребители, които са имали сходни музикални предпочитания и/или за анотираните песни и изпълнители. Когато получи тези данни, прави необходимите изчисления, за да намери сходството между текущия потребител и други такива или между профила на потребителя и профилите на различни песни. Използвайки това сходство предсказва оценката, която неоценени до момента от потребителя песни биха получили и препоръчва тези, които имат най-високо предсказание.

Модулът за събиране на данни отговаря за първоначалното събиране на информация от *last.fm*, която да се запомни и да се ползва при по-нататъшни препоръки. Този модул не работи постоянно, а се включва от администратор в системата в случаите, когато се прецени, че информацията, която е налична в базата данни не е достатъчна за направата на качествени препоръки.

Модулът за оценяване се занимава с изследване на точността на предложенията, които се дават от системата. Този модул също не работи постоянно, а се включва периодично от администратор. Има два режима на работа - в единия агрегира вече пресметнати метрики от работата на потребителите с приложението, а в другия сам оценява различните аспекти от работата на препоръчващата услуга.

Модулът за съхранение (*база данни*) отговаря за запазване на събраната от *data collector-a* информация, за запазването на информацията за обратната връзка от страна на потребителите, както и за последващо извлечане на тази информация.

4.1 Модел на данните

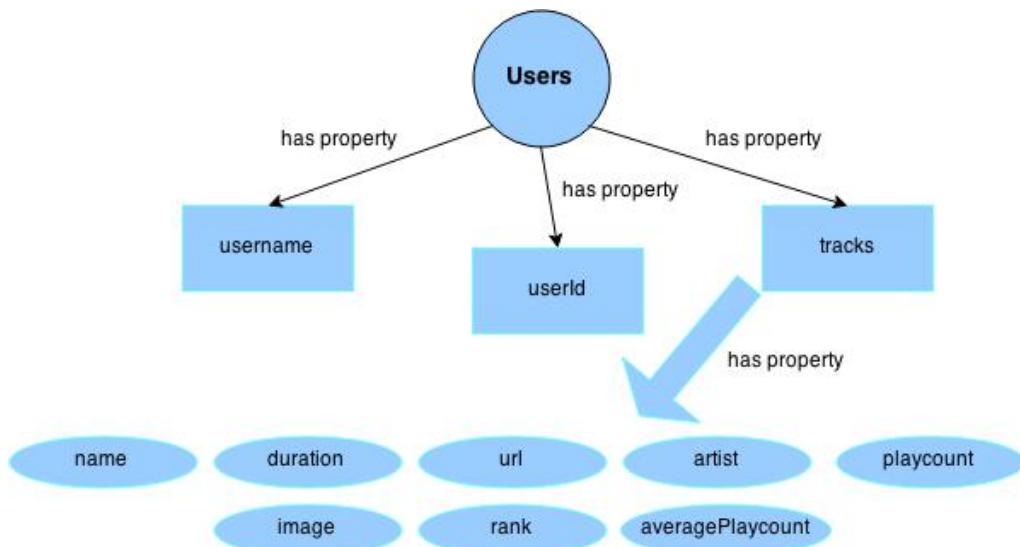
В тази част ще разгледаме главните същности, които приложението използва, както и връзките между тях. Основната същност в системата са *last.fm* потребителите (*Users*), за които са били събрани данни първоначално. За всеки такъв потребител се пази следната информация (*показана на фигура 4.2*):

- потребителско име от *last.fm*
- потребителски идентификатор от *last.fm*
- списък с всички прослушани в *last.fm* песни. За всяка песен, от своя страна се съхраняват нейното име, продължителност, линк за информация, изпълнител, брой

прослушвания, свързано изображение и поредност в списъка с песните, сортиран по брой слушания

- агрегиран среден брой прослушвания за песен. Това поле не е част от извлечаните от *last.fm* данни, а се пресмята допълнително с цел оптимизация работата на препоръчващата услуга, тъй като в противен случай ще трябва да се пресмята при всяка заявка към нея.

Друга същност в системата са песните, които се дават като първоначална препоръка на потребителя в случай, че нямаме достатъчно информация за него. Тези данни се съхраняват в колекцията *TracksToRate*, като структурата е същата като при списъка с песни за всеки потребител от *last.fm (Users)*. На теория наличието на такава колекция води до излишество, тъй като бихме могли да направим справка за най-популярните песни, които потребителите в колекцията *Users* са слушали. Това обаче би било неефективно от изчислителна гледна точка, тъй като ще трябва при всяка потребителска заявка да се обхождат всички песни в базата (над 112000 на брой), за да се намерят най-популярните такива.



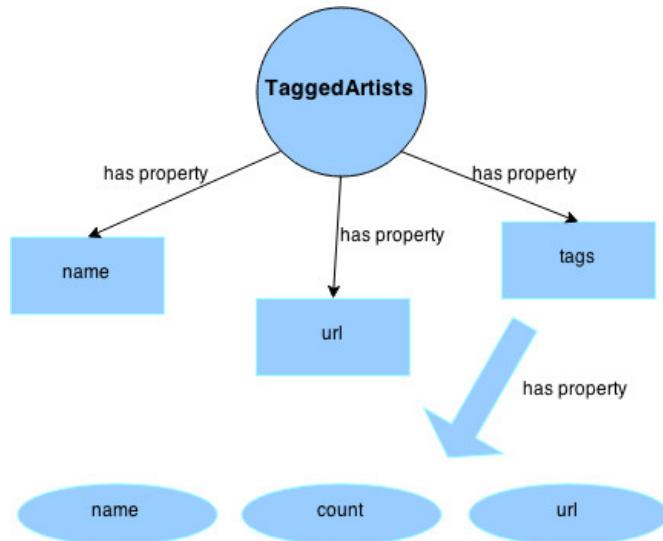
Фигура 4.2. Модел на данните за потребителите в препоръчващата система

Други две същности (*TaggedArtists* и *TaggedTracks*) предоставят информация за:

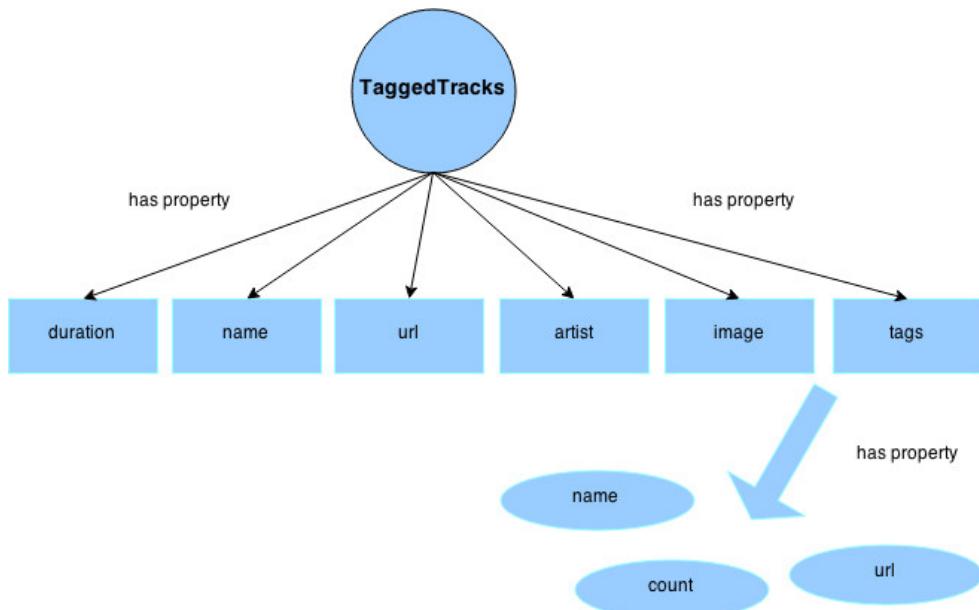
- изпълнител на дадена песен. За всеки такъв изпълнител се пази информация за име, снимки, свързани линкове, както и за множество от анотации, с които е бил обозначен от потребителите на *last.fm*, както и броят на тези анотации.
- конкретна песен. За всяка песен се пазят име, изпълнител, свързани изображения и линкове, както и всички анотации, с които е била обозначена от потребителите на *last.fm* и броят на тези тагвания.

Фигури 4.3 и 4.4 дават визуална представа за структурата на документите в

горепосочените две колекции:

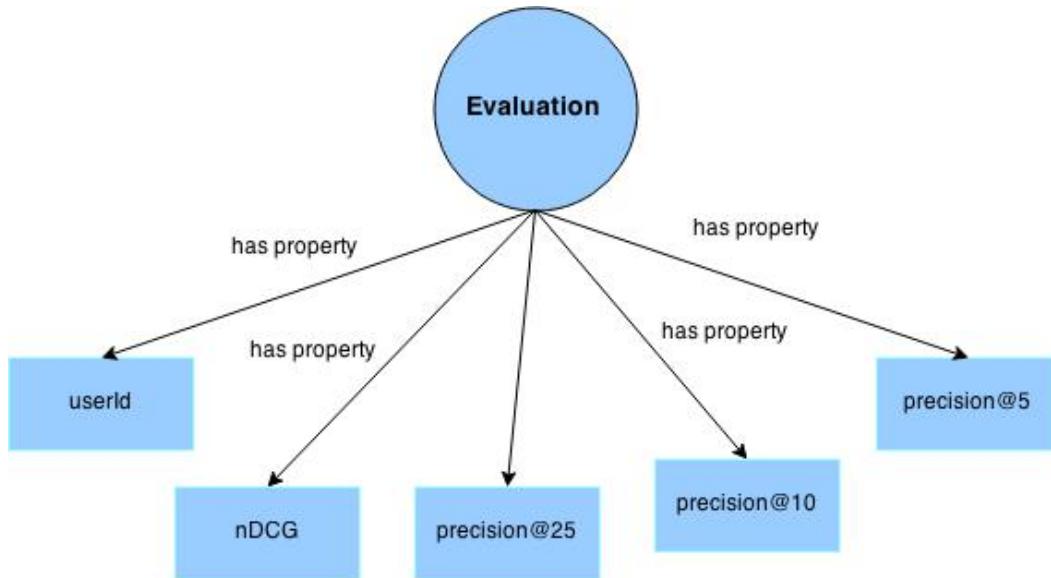


Фигура 4.3. Модел на данните за изпълнители в препоръчващата система



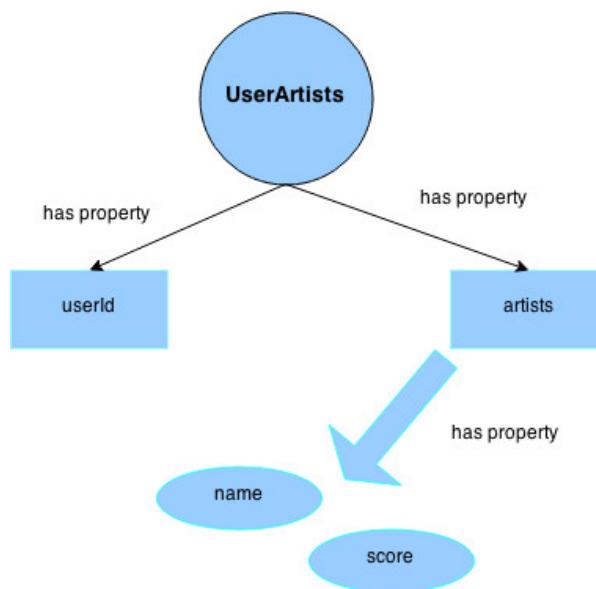
Фигура 4.4. Модел на данните за песни в препоръчващата система

Друга от същностите в системата (*Evaluation*) пази информация за обратната връзка дадена от всеки потребител за качеството на препоръките. Нейната структура е показана на *фигура 4.5* и съдържа стойностите на метриките, оценяващи точността на направените препоръки, които даден потребител (с идентификатор *userId*) е получил.



Фигура 4.5. Модел на данните за оценяване в препоръчваща система

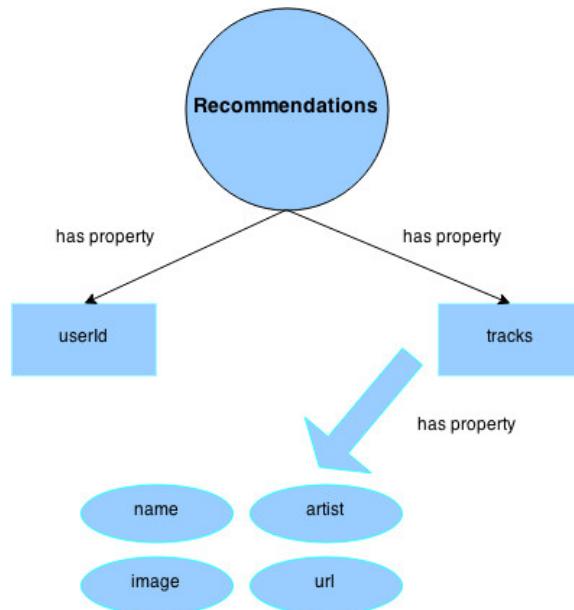
Друга същност, за която се пази информация се нарича *UserArtists*. Тя се ползва за съхранение на данните за това какви оценки са поставили потребителите на различни изпълнители като основната цел е да се натрупа потребителска история и да се подобри качеството на даваните препоръки с времето. Отделните полета са илюстрирани на *фигура 4.6*:



Фигура 4.6. Модел на данните за оценяване в препоръчваща система

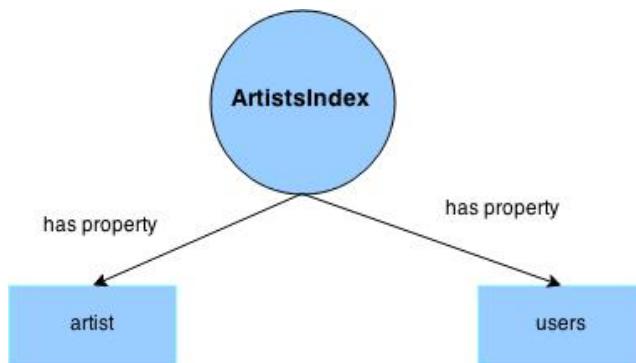
Една от останалите същности, за които се пази информация в базата е *Recommendations*. В нея, за всеки потребител на системата се записват данни за

получените от него препоръки, както и оценката, която те са получили. Тези данни се използват, за да се избегне повторно препоръчване на едни и същи обекти, а също и за да се поддържа потребителска история.

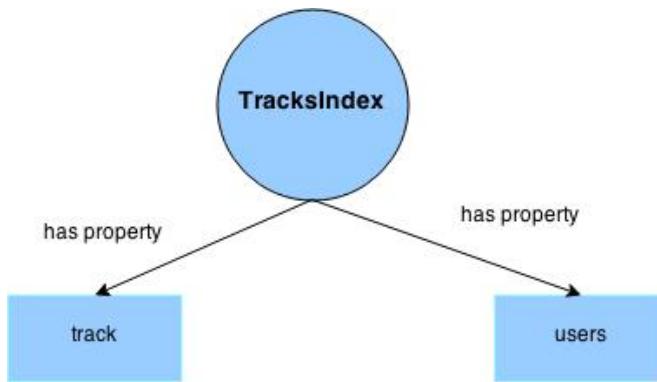


Фигура 4.7. Модел на данните за оценяване в препоръчващата система

Последните две колекции в базата са *ArtistsIndex* и *TracksIndex*. От семантична гледна точка, те не са носители на някакви нови данни, а се ползват единствено с цел ускоряване на търсенето при потребителски заявки за препоръки. В тях се пазят съответствия респективно между изпълнители и уникални идентификатори на потребители, които са слушали техни произведения (за *ArtistsIndex*) и между песни и идентификатори на потребители, които са ги слушали (*TracksIndex*). Тъй като *MongoDB* поддържа индекс по *id*-тата на потребителите, търсенето по тях е изключително бързо и позволява ефективната работа на услугата.



Фигура 4.8. Модел на данните за съответствие между изпълнители и потребители, които са ги слушали в препоръчващата система

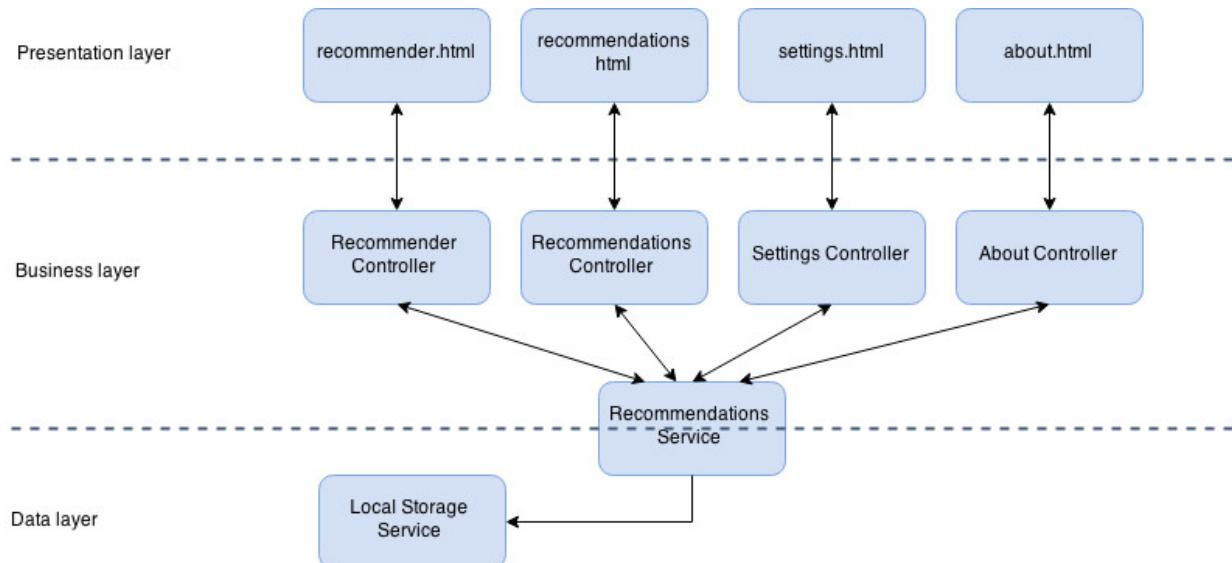


Фигура 4.9. Модел на данните за съответствие между песни и потребители, които са ги слушали в препоръчващата система

4.2 Декомпозиция на модулите

В най-общи линии приложението се състои от 5 модула - клиентски, сървърен, препоръчващ, модул за събиране на данни и модул за оценяване.

Задачата на клиентския модул е да показва съответните графични елементи на потребителя в зависимост от неговите действия и да комуникира със сървърния компонент, когато това се налага. В него има ясно разделение на логиката на 3 слоя - презентационен, бизнес и слой за съхранение. Комуникацията между отделните компоненти се осъществява по следната схема:



Фигура 4.10. Обща архитектура на клиентската част на препоръчващата система.

При първоначалното зареждане на приложението на потребителя се показва изгледа *recommender.html*, който му позволява да въведе източници на данни за музикалните си

предпочитания. Когато потребителят поиска списък с препоръки, действието му извиква метод на асоциирания контролер - *RecommenderController*, който от своя страна се обръща към компонента, който води комуникацията със сървъра - *RecommendationsService*.

След получаването на препоръките от сървъра, *RecommenderController-a* навигира потребителя към следващия еcran - този, който ги визуализира. Той е представен графично посредством изгледа *recommendations.html* като специфичните за потребителя данни, които се показват в него се взимат автоматично (посредством *two-way data binding*) от контролера. Когато потребителят оценява направените му препоръки отново се извикват съответни методи в този контролер, а те се обръщат към *RecommendationsService-a*, който от своя страна изпраща оценките на сървърната част.

Изгледът *settings.html* дава възможност на потребителя да настрои параметрите на работата на приложението. С него е свързан *SettingsController-a* и при промяна на стойностите на препоръчващите параметри се извикват съответни негови методи. От своя страна той се обръща към *RecommendationsService-a*, който запазва новите стойности в т.нар. *local storage* на уеб браузъра. Това е хранилище за информация, представена във вида ключ-стойност, която е достъпна от клиентската част на уеб приложенията. При последващи обръщения от страна на *RecommendationsService-a* към сървърната част се включват обновените стойности на параметрите.

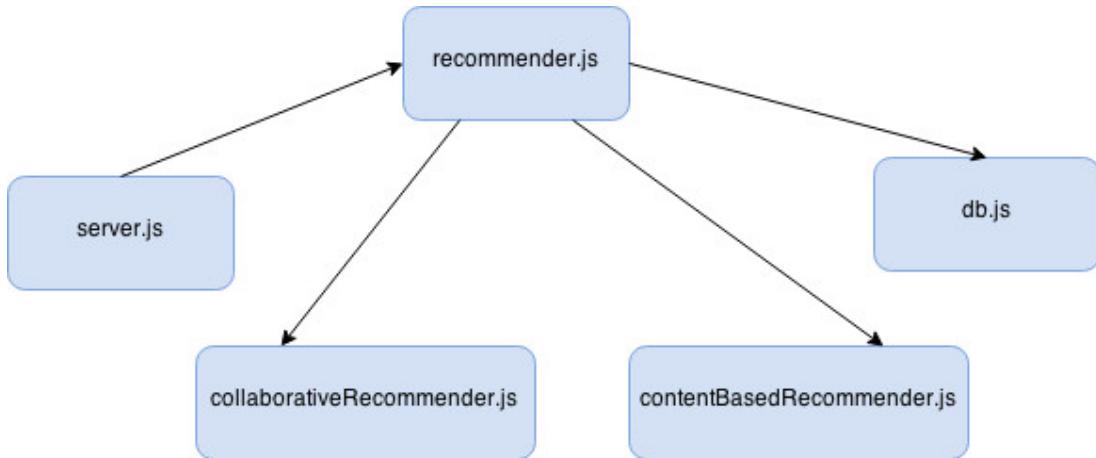
Последният еcran, който може да види потребителя е *about.html*. Той е изцяло статичен и неинтерактивен и съдържа обща информация за играденото приложение, неговите принципи на работа и цели.

Целите на сървърния модул са да приема клиентските заявки, да ги обработва и да прави съответните извиквания за резултати към препоръчващия модул. Той е съставен от единственный файл - *server.js*, който отговаря за създаването и стартирането на уеб сървър, изпълняващ следните дейности:

- сервиране на статичните клиентски файлове на потребителите
- предоставяне на съответните уеб услуги, чрез които клиентите да получават препоръки и да дават обратна връзка

За да се изпълняват тези задачи се ползват 2 помощни библиотеки (*NodeJS* модули) - *node-static* и *expressjs*, информация, за които е предоставена в предходните глави.

Отговорностите на препоръчващия модул са свързани с генерирането на списък с препоръчана музика на базата на съответни входни данни. Той е организиран в 3 файла - *recommender.js*, *collaborativeRecommender.js*, *contentBasedRecommender.js* и комуникира както със сървърната част (получаване на входни данни и връщане на отговор), така и с модула за съхранение на данни (извлечане на информация за потребители).



Фигура 4.11. Комуникация между препоръчващата част и останалите компоненти на системата

Модулът за събиране на данни също се състои от единствен файл - *crawler.js*. Неговата дейност е свързана със събирането на първоначално множество от информация за потребителите на *last.fm*. За да осигури съхраняването на така събраната информация, този модул комуникира с модула за запазване на данни (*db.js*).

Модулът за оценяване е събран в рамките на един файл - *evaluator.js*. Той се занимава с оценяване работата на системата като за целта пресмята метриките *precision*, *recall*, *F₁-measure* и *nDCG*. За да може да направи това се нуждае от достъп до информацията, събрана от работата на приложението и съответно комуникира с модула за запазване на данни.

4.3. Реализация на препоръчваща система

В тази част ще се спрем на конкретните алгоритми, метрики и технологични решения, които са взети в процеса на създаване на препоръчващата система. Ще дадем детайли относно причините за техния избор и предимствата и недостатъците им спрямо останалите възможни решения.

4.3.1 Избор на препоръчващ алгоритъм

Музикалните препоръки са по-особени от останалите преди всичко с това, че могат бързо да бъдат проверени. При най-често срещаните употреби на препоръчващи системи - за препоръка на филми, книги и битови продукти, потребителят не може да даде непосредствена обратна връзка за степента на удовлетвореност от направената препоръка. Това се дължи на дългия период между момента на даване на препоръката и момента, в който потребителят може да оцени нейният субект - при закупуване на продукт от онлайн магазин обикновено са необходими дни за доставка, прочитането на книга би могло да

отнеме седмици и т.н. От друга страна средната продължителност на песен е около 3 минути и потребителят би могъл веднага да оцени качеството на направените препоръки, по този начин подобрявайки работата на системата.

Изследването на вече съществуващи приложения, препоръчващи музика и начините, по които те работят показва, че както подходът базиран на сътрудничество, така и този основан на съдържание са подходящи. Нещо повече - най-популярното от разгледаните решения - проектът MuSe⁶ позволява на потребителите си сами да настройват както типа на препоръчващия алгоритъм, така и различните параметри, с които той работи. Този подход е добър от гледна точка на експерименталния характер на MuSe, тъй като дава възможност за събиране на информация за качеството на музикалните препоръки при различни условия и тяхното анализиране. В разработения прототип също е подходящо да се позволят подобни настройки на системата, тъй като по този начин потребителите могат да извлекат максимална полза от нея, а именно:

- използвайки колаборативния препоръчващ подход да получат като препоръки качествено нови за тях песни, макар и рискувайки те да бъдат не толкова релевантни.
- използвайки препоръчващия подход, основан на съдържание да получат като препоръки песни от сходни на вече харесани от тях изпълнители, които макар и не толкова неочаквани са точни.

Макар и двете основни алгоритмични схеми за препоръчване да са подходящи за предлагане на музика, все пак, остава отворен въпросът как да намираме сходни потребители и/или песни, които отговарят на потребителския профил. Възприетият в текущата работа подход действа, както следва:

- **при препоръки чрез сътрудничество**
 - потребителят на системата трябва да предостави информация за предпочитани от него песни, техните автори, както и за степента, в която ги харесва.
 - в зависимост от настройките на приложението се извлича списък с всички *last.fm* потребители, за които първоначално са били събрани данни, които са слушали музика от въведените от потребителя изпълнители (по подразбиране) или, които са слушали точно същите като оценените от потребителя песни (ако от панела с настройките е била избрана опцията *tracks* като тип на използваните при препоръки данни).
 - за активния потребител и всеки от потребителите, получени в предходната стъпка се пресмята *корелация на Пирсън*, като се използват оценките и/или броя на слушанията на всеки изпълнител / песен.
 - на база на така пресметната близост между потребителите се определят

⁶ <https://muse.informatik.uni-freiburg.de/MuSe/#>

N-me най-близки до текущия ползвател на системата съседи и за всяка от изслушаните от тях песни се пресмята оценката, която текущия потребител би й дал (съгласно формулата, описана в глава 2).

- препоръчват се *25-me* песни с най-висока предсказана оценка, които не са слушани от активния потребител, сортирани в низходящ ред по тази оценка.
- при **препоръки, основани на съдържание**
 - потребителят на препоръчващата система отново предоставя информация за степента, в която харесва дадени изпълнители.
 - от базата данни се извлича информация за *таговете / анотациите*, с които *last.fm* потребителите са означили въведените изпълнители. Извлича се информацията за всички налични в базата песни, които отговарят на поне един от тези тагове.
 - Чрез горната стъпка за ползвателя на системата се изготвя *профил* - вектор от числа, представляващи общия брой на означаванията на харесваните от него изпълнители с определени тагове. По аналогичен начин се изготвя и профил на всяка от песните, за които разполагаме с информация.
 - между вектора на потребителския профил и векторите на всички песни се изчислява косинусова прилика. Избират се *25-me* песни, за които стойността на тази прилика е най-висока и те се дават като препоръка.

4.3.2 Преглед на метода за събиране и съхранение на данни от *last.fm*

Един от ключовите за работата на системата компоненти е този за събиране на началното множество данни, върху които ще се основават направените препоръки. Този компонент извлича информация за 3 основни същности - потребители от *last.fm*, изпълнители и песни.

Действието му при събиране на данни за потребители се състои в следните стъпки:

- обхождането на потребители започва от предварително зададен в конфигурационен файл такъв (подбран, така че да има голям брой приятели). Системата събира данни за *50-te* най-слушани от него песни.
- Системата обхожда в дълбочина социалния граф с корен този потребител и върхове неговите приятели, приятелите на неговите приятели и т.н. За всеки потребител от това дърво се събира информация за *50-te* най-слушани от него песни.

По този начин в рамките на около 3 часа бяха събрани данни за най-слушаните песни от повече от 25000 потребители като общото количество информация надвишава *3GB*. Нека отбележим, че с цел постигане на по-голямо разнообразие в събранныте за дипломната работа данни *crawler*-а беше пуснат неколкократно (*над 10 пъти*) с различни начални за обхождането потребители. Тези потребители бяха подбрани, така че да бъдат от различни географски локации и да имат различен профил (пол, възраст). Като допълнителна мярка

за увеличаване покритието на препоръчващата система и разнообразието на даваните препоръки може да отбележим и използването на обхождане в дълбочина вместо обхождане в ширина. При подход в ширина щяхме да имаме много данни за сходни потребители (за всеки потребител щяхме да разполагаме с данните за всички негови приятели), докато обхождането в дълбочина подсигурява, че ще посетим хора, които имат минимална връзка помежду си.

Освен за *last.fm* потребители този компонент събира данни и за различни изпълнители и песни, както и таговете, с които те са били означени от общността. Подходът при извличането на тази информация е сходен с този при потребителите и може да се опише със следните стъпки:

- обхождането започва от предварително създаден списък от изпълнители (тези, които се предлагат на потребителите в случая на неперсонализирана препоръка). За всеки такъв изпълнител се извличат *50-te* най-популярни тага, с които е бил описан
- за всеки такъв таг се изличат *50-te* най-популярни изпълнителя, означени с този таг
- за всеки от тези изпълнители се извлича списък с тагове, отново се повтарят действията от предната точка т.н.

Използвайки този алгоритъм, за целите на приложението беше събрана информация за повече от *12000* изпълнителя и *11000* песни.

4.4 Използвани технологии и услуги

Прототипът на препоръчващата система е изграден по модела клиент-сървър.

Клиентската част е тази, с която потребителят работи директно, въвеждайки данни за себе си и получавайки препоръки. Основната част от нея се базира на *AngularJS* - една от най-популярните към момента *Javascript* библиотеки. Причините да бъде предпочетена именно тя са нейната висока популярност, голяма общност от разработчици и наличието на множество помощни материали и добра документация.

Сървърната част е тази, която съхранява данните въз основа, на които се правят препоръки, данните на потребителите и освен това прави самите препоръки. Основната технология, която се ползва за целта е *NodeJS*. Тя е подходяща за целите на системата, тъй като е лесна за работа и позволява бързо прототипиране. За нея съществуват множество готови модули и добра документация.

Съхранението на данните се осъществява посредством *MongoDB*. Това е документно-ориентирана база данни и е особено удобна за съхранение на данни, получени в *JSON* формат. Освен това се интегрира без затруднения с израната сървърна технология - *NodeJS*.

4.4.1 AngularJS

AngularJS е *Javascript* фреймуърк с отворен код, който понастоящем се поддържа и развива основно от *Google*. Неговата цел е да улесни създаването и тестването на уеб приложения като предостави множество готови компоненти и лесен начин за изграждане на *Model-View-Controller(MVC)* архитектура. По-интересните особености на фреймуърка включват:

- начин за дефиниране на собствени *HTML* таг-ове (наричани *директиви*).
Директивите позволяват отделянето на бизнес логиката на приложението от логиката за манипулиране на *DOM* елементите.
- двупосочко свързване на данни (*two-way data binding*) между модела и изгледите (*views*) в приложението. Води до силно намаляване на броя на извършваните *DOM* манипулации, тъй като показваните на потребителя стойности се взимат директно от модела и при тяхна промяна (както в модела, така и в изгледа - от страна на потребителя) стойностите се променят и на двете места.
- предоставя начин за преизползване на изгледи (*views*). Може да създаваме визуални компоненти, които да отделим в собствен файл (наричани още *partial-u*). Такива компоненти може да се реферират на множество места в кода чрез "включване" на файла и по този начин да се избегне дупликация на код.
- дефинира точно определена структура на приложението, която е изградена върху принципа за разделяне на бизнес логиката от тази за съхранение на данните и показване на информация на потребителя (*Model-View-Controller*).
- разполага с множество готови за ползване компоненти и библиотеки, които решават често срещани при разработката на уеб приложения задачи и проблеми.

Като алтернативни технологии, които биха могли да се ползват в реализирания прототип бихме могли да посочим библиотеката *Ember.js*, а също и *vanilla Javascript*, т.е. клиентската част да се имплементира без помощта на допълнителни фреймуърци. Първият вариант беше отхвърлен, тъй като въпросната библиотека разполага с много ограничена документация и е значително по-трудна за разучаване и работа от *Angular*. Не считаме, че и опцията да не се ползва фреймуърк е подходяща, тъй като по този начин ще трябва да се напише голямо количество несъществен (*спомагателен/boilerplate*) код, който ще дублира вече съществуващи функционалности в *Angular*.

4.4.2 Twitter Bootstrap

Bootstrap е колекция от инструменти за създаване на уеб страници. Съдържа множество *HTML* и *CSS* темплейти за създаването на различни форми, бутони, навигация, дефиниране на стилна и консистентна типография. Пренаписва стиловете по подразбиране за почти всички *HTML* елементи като по този начин подсигурява, че те изглеждат еднакво

и ефектно под всички браузъри. Предоставя множество готови визуални компоненти - списъци за избор на елемент (*dropdowns*), менюта, диалози и други.

Bootstrap е разработен от група програмисти, работещи за *Twitter*, за да се улесни работата по някои от вътрешните за компанията проекти, но впоследствие се пуска за широка употреба и с отворен код. Понастоящем се разработва основно от доброволци и е най-популярният проект в платформата за споделяне на код *Github*.

4.4.3 Nodejs

NodeJS е *Javascript* фреймуърк с отворен код за създаване на сървърни и мрежови приложения. В основата му е залегнала идеята за асинхронна комуникация, базирана на събития и неблокиращи извиквания на методи, което го прави особено подходящ за приложения работещи в реално време.

За изпълнението на код, написан на *NodeJS* се използва *Javascript engine-a V8*, създаден от *Google* за *Chrome*. За разлика от традиционните *engine*-и, които интерпретират *Javascript* кода в реално време, *V8* го компилира използвайки т.нр. *Just In Time(JIT)* компилация директно до машинен код, което значително ускорява изпълнението.

Като част от стандартната дистрибуция на *Node* се разпространява и т.нр. *node package manager* - инструмент, който дава достъп до хранилища, в които се съхраняват множество готови модули, огромна част, от които са разработени от доброволци. Тези модули улесняват значително разработката на приложения като предоставят наготово редица често търсени функционалности.

Като алтернативно решение за реализация на сървърната част беше разгледано тя да се напише на *Java*. Една такава имплементация, обаче би била значително по-бавна, тежка и тромава от текущата. Имплементацията на уеб сървиси на *Java* добавя ненужно ниво на сложност в системата, тъй като уеб приложениета написани за тази платформа не са *self-contained*, а е необходим специален *application* сървър, върху който да се изпълняват. Изборът и настройките на такъв сървър биха отнели ненужно време. Като друг сериозен минус бихме посочили и трудностите при паралелна обработка на данни - изпълняването на паралелен код в *Java* изисква да се положат специални грижи за синхронизацията между отделните нишки, които изпълняват този код, докато в *NodeJS* по подразбиране всичко се изпълнява асинхронно.

4.4.4 Node модули

Беше отбелязано, че един от основните плюсове на избраната сървърна технология е наличието на множество помощни модули, които значително улесняват работата с нея. В рамките на тази секция ще се спрем на тези от тях, които са използвани при изграждането на прототипа на препоръчващата система.

На първо място трябва да се спомене модулът *express*. Той е един от най-популярните сред потребителите на *Node* и представлява уеб фреймуърк, който надгражда вградения модул *http* като добавя функционалност за лесно парсване на заявките към сървъра, поддържане на потребителска сесия, рутиране на заявките към съответните хендъри, интеграция с *template engine*-и и други. Като негови алтернативи може да отбележим модулите *Meteor.js*, *Derby.js*, *Sails.js* и *Flatiron.js*, които обаче са значително по-непопулярни и слабо развити.

Node-static е друг широко използван модул, който се използва в изградения прототип. Той представлява минималистичен уеб сървър, който позволява сервирането на статично съдържание. Предимствата му пред други подобни модули са основно в лесното използване и добра документация.

Библиотеката *argparse* се ползва за лесно парсване на аргументи, които са били подадени от командния ред към дадено приложение. Освен възможност за парсване, тази библиотека дава наготово и допълнителни функционалности при конзолна работа с изпълними файлове - показване на списък с възможните опции за стартиране, помощ, *alias*-и/*shortcuts* за някои команди и други.

Async е друг от използванието модули и се различава от останалите вече обсъдени такива по това, че за разлика от тях не добавя никаква нова функционалност, а ползата от употребата му е само стилистична и води до подобряване четимостта и качеството на написания код. С негова помощ се решава един от най-разпространените в рамките на *Node* приложенията проблеми, а именно т. нар. *callback hell*. Поради характера на работата на *NodeJS* е необходимо при изпълнението на всяка асинхронна функция да се подаде като аргумент друга функция (*callback*), която да се извика, когато асинхронната работа приключи. Често пъти това води до многоократно влагане на извиквания на функции, които са зависими една от друга и трябва да се изпълнят последователно. Разгледания модул решава този проблем по елегантен начин и води до значително по-компактен и четим код.

Модул, който се ползва със сходни цели в рамките на приложението е и *underscore*. Той представлява порт за *Node* на популярната клиентска *Javascript* библиотека, носеща същото име. Употребата на тази библиотека ни дава възможност да ползваме множество, познати ни от функционалните езици за програмиране функции - map, filter, fold, zip, flatten, както и много други, които предоставят наготово често ползвани функционалности и по този начин ни позволяват да пишем по-малко и по-четим код.

За да е напълно изчерпателен списъка с използванието в прототипа модули остава да споменем и тези, които се ползват като абстракция върху комуникацията със свързани услуги - *mongodb* и *lastfmapi*.

4.4.5 MongoDB

MongoDB е най-популярната към момента нерелационна база данни. За разлика от традиционните релационни бази, където обектите се моделират посредством таблици, в *Mongo* те се представят в *JSON (Javascript Object Notation)* формат. Това значително улеснява пазенето на информация, тъй като вместо да разпръсваме отделните данни за един и същи обект в множество таблици ги съхраняваме в единна структура (наречена *документ*) в *JSON* формат. Допълнителна полза е и, че *JSON* се ползва широко при обмен на данни между клиент и сървър и употребата му и в базата решава проблема за конвертиране на получената от сървъра информация, т.ч. да може да се съхранява.

Както всяка нерелационна база данни, така и *Mongo* работи с динамична схема, т.е. не е необходимо при създаването на базата да се уточнява типа и формата на данните в нея. Това е от голяма полза при прототипиране, тъй като решава проблема с промени по схемата, които обикновено изискват много време.

По подобие на релационните бази, *Mongo* индексира данните и по този начин осигурява бърз отговор на потребителски заявки. Освен това поддържа лесни възможности за репликация и разделяне на базата между множество сървъри (т.нар. *sharding*), което би било ключово за бъдещото развитие на изграждания прототип. Тъй като *MongoDB* е проект с отворен код за него съществува богата документация и потребителска общност, а също и множество библиотеки за различни езици и платформи, които улесняват работата с него.

В процеса на разработка на системата беше обсъдена и друга технология за съхранение на данните - MySQL / MariaDB. Двете представляват съответно една от най-популярните релационни бази данни и нейният най-използван към момента *fork*. Считаме, обаче, че употребата на нерелационна база е значително по-подходяща за изгражданото препоръчващо приложение за музика по следните причини:

- динамичния характер на нерелационните бази позволява тяхната схема да не се специфицира предварително, а да може да се променя по всяко време. Това е от голямо полза при създаването на прототип.
- нерелационните бази дават лесен начин за съхранението, записване и извлечане на информация. *JSON* форматът, който се ползва от *Mongo* се ползва и за комуникация между клиента и сървъра, което пък от своя страна значително улеснява комуникацията между сървъра и базата.
- нерелационните бази имат ефективност и бързодействие, които са сходни с тези на релационните такива.

4.4.6 Програмен интерфейс на Facebook

От 2010г. насам (Facebook Developers, 2015) *Facebook* предоставят програматичен достъп до информацията си посредством т.нр. *Graph API*. За неговото ползване е необходимо предварителното създаване на *Facebook* приложение, което потребителите да оторизират за достъп до техните данни. След процеса по оторизация е възможно да се направят заявки към следните крайни точки в *API-то* даващи почти пълната информация, с която *Facebook* разполага за всички действия на потребителя. Ако оторизация бъде отказана е възможно да се извършват заявки само за силно ограничена информация, която е достъпна публично.

Адрес	Описание
<i>/achievement/{achievement-id}</i>	дава информация за постижението с идентификатор <i>achievement-id</i> , отключено в някоя игра
<i>/achievement-type/{achievement-type-id}</i>	дава информация за типа на постижение с идентификатор <i>achievement-type-id</i>
<i>/album/{album-id}</i>	дава информация за албум със снимки с идентификатор <i>album-id</i> (брой снимки, коментари, харесвания на албума и т.н.)
<i>/app/{app-id}</i>	дава информация за <i>Facebook</i> приложението с идентификатор <i>app-id</i>
<i>/app-link-host/{app-id}</i>	дава информация за адреса на сървъра, на който върви <i>Facebook</i> приложението с идентификатор <i>app-id</i>
<i>/comment/{comment-id}</i>	дава информация за коментара с идентификатор <i>comment-id</i>
<i>/conversation</i>	дава информация за потребителски пост на стената на <i>Facebook</i> страница
<i>/debug_token?input_token={token-id}</i>	дава информация за <i>access token-a</i> с дадения идентификатор
<i>/domain/{domain-id}</i>	дава информация за домейна на дадено <i>Facebook</i> приложение
<i>/event/{event-id}</i>	дава информация за <i>Facebook</i> събитието с идентификатор <i>event-id</i>

<i>/friendlist/{friendlist-id}</i>	дава информация за списък с приятели, който потребителят е направил
<i>/group/{group-id}</i>	дава информация за дадена <i>Facebook</i> група
<i>/group-doc/{group-doc-id}</i>	дава информация за <i>Facebook</i> документ, споделен с всички потребители на дадена група
<i>/link/{link-id}</i>	дава информация за линк, който е бил споделен от даден потребител
<i>/message/{message-id}</i>	дава информация за дадено съобщение
<i>/milestone/{milestone-id}</i>	дава информация за събитие, създадено от <i>Facebook</i> страница
<i>/notification/{notification-id}</i>	дава информация за известяване, получено от даден потребител
<i>/{object-id}/comments</i>	дава информация за всички коментари, свързани с произволен обект (снимка, статус, линк, албум, други)
<i>/{object-id}/insights/{metric-name}</i>	дава статистика за произволен обект (страница, статус, линк, снимка)
<i>/{object-id}/likes</i>	дава информация за всички харесвания на даден обект (снимка, статус, албум, т.н.)
<i>/object/sharedposts</i>	дава информация за споделянията на даден обект (снимка, статус, албум, други)
<i>/offer/{offer-id}</i>	дава информация за оферта, публикувана от <i>Facebook</i> страница
<i>/page/{page-id}</i>	дава информация за дадена <i>Facebook</i> страница
<i>/photo/{photo-id}</i>	дава информация за дадена снимка, споделена от потребител
<i>/{place-tag-id}</i>	дава информация за geo-тагване, извършено от потребител
<i>/{post-id}</i>	дава информация за потребителски пост (статус, линк, снимка, албум, т.н.)

<i>/profile/{profile-id}</i>	дава информация за профила на потребител/страница/група/събитие
<i>/request/{request-id}</i>	дава информация за предложение за ползване на някакво приложение или за харесване на страница
<i>/review/{review-id}</i>	дава информация за ревю на място или приложение, направено от потребител
<i>/status/{status-id}</i>	дава информация за статус, пуснат от даден потребител
<i>/test-user/{test-user-id}</i>	дава информация за потребител, който е бил създаден като тестови за дадено приложение
<i>/thread/{thread-id}</i>	дава информация за множество съобщения (разговор) обменени между двама потребители
<i>/user/{user-id}</i>	дава информация за даден потребител
<i>/video/{video-id}</i>	дава информация за видео, публикувано от даден потребител

Таблица 4.1. Списък с точки за достъп от Graph API.

Резултатите от заявки към някой от гореизброените адреси се получават в *JSON*⁷ (*Javascript Object Notation*) формат (Crockford, Douglas, 2006). Макар създаден и стандартизиран едва през 2006г. той е широко разпространен и лесен за генериране и парсване. Примерен резултат от заявка към *Graph API*, използваща този формат е показан на *фигура 4.12*.

⁷ <http://json.org/>

```
{
  "data": [
    {
      "category": "Health/beauty",
      "name": "Спортни лагери на Софийския университет",
      "created_time": "2015-01-19T17:40:06+0000",
      "id": "1404726009825292"
    },
    {
      "category": "Internet/software",
      "category_list": [
        {
          "id": "2256",
          "name": "Internet/Software"
        }
      ],
      "name": "Heroku",
      "created_time": "2015-01-11T17:14:00+0000",
      "id": "110151569043864"
    }
  ],
  "paging": {
    "cursors": {
      "after": "MTEwMTUxNTY5MDQzODY0",
      "before": "MTQwMDcyNjAwOTgyNTI5Mg=="
    },
    "next": "https://graph.facebook.com/v2.2/812914452/likes?limit=2&after=MTEwMTUxNTY5MDQzODY0"
  }
}
```

Фигура 4.12. Примерен резултат от заявка към Graph API

4.4.7 Програмен интерфейс на Last.fm

През 2005г. (Last.fm, 2005) музикалният сайт *Last.fm* също прави достъпна информацията, с която разполага за потребителите си. Тази информация може да се ползва по два начина - свободно (публично), както и след регистрация.

При достъпът без регистрация ползвателят на *API*-то има пълен достъп до данните на всички потребители без да е необходима някаква форма на оторизация. Ограниченията са свързани с невъзможността да се правят заявки, които променят данни (например отбелязване на песен като слушана и т.н.) - т.е. предоставя се само достъп за четене. Друг недостатък е налагането на максимален брой на заявките, които може да се пращат - те не трябва да надвишават 1000 за един час, което означава, че този подход е подходящ само за тестови цели.

При достъпът с регистрация е необходимо ползвателят на услугата да се регистрира в *last.fm* и да създаде приложение там. След като направи това получава уникален ключ за ползване на *API*-то, който го идентифицира и му дава възможност да прави неограничен брой заявки за информация. Освен това, след съответна оторизация на приложението от потребителя, то може да изменя неговите данни в рамките на системата (т.нар. достъп за писане).

Списък с по-важните точки от *API*-то е описан в таблица 4.2.

Адрес	Описание
<i>/user/getTopArtists</i>	връща списък с информация за всички изпълнители прослушани от даден потребител сортиран в низходящ ред по брой на слушанията
<i>/user/getTopTracks</i>	връща списък с информация за всички прослушани от даден потребител песни сортиран в низходящ ред по брой на слушанията.
<i>/user/getLovedTracks</i>	връща списък с информация за всички харесани от даден потребител песни
<i>/user/getFriends</i>	връща списък с всички приятели на даден потребител
<i>/user getInfo</i>	връща информация за даден потребител (име, възраст, местоположение и т.н.)
<i>/artist/getInfo</i>	връща информация за даден изпълнител - име, песни, снимки, местоположение и т.н.
<i>/artist/getTopFans</i>	връща списък с информация за потребителите, които са слушали най-голям брой песни от този изпълнител
<i>/artist/getTopTags</i>	връща списък с информация за всички тагове, които са били дадени от потребители за този изпълнител
<i>/artist/getTopTracks</i>	връща списък с информация за всички песни на даден изпълнител сортиран в низходящ ред по броя на прослушванията
<i>/artist/getTopAlbums</i>	връща информация за всички албуми, които са дело на даден изпълнител сортиран в низходящ ред по брой слушания на песните от албумите
<i>/artist/getEvents</i>	връща списък с информация за предстоящи събития (концерти) на даден изпълнител
<i>/tag/getInfo</i>	дава информация за направен от потребител таг

<code>/tag/getSimilar</code>	дава информация за сходни на даден таг такива
<code>/tag/getTopTracks</code>	дава списък с всички песни означени с даден таг, сортиран по брой прослушвания
<code>/tag/getTopArtists</code>	дава списък с всички изпълнители, които са били означени с даден таг, сортиран по брой означавания
<code>/track/getInfo</code>	връща информация за дадена песен - изпълнител, тагове, свързани изображения и т.н.
<code>/track/getTopTags</code>	връща списък с всички тагове за дадена песен сортиран по техния брой
<code>/track/getTopFans</code>	връща списък с информация за всички потребители, прослушали дадена песен сортиран по броя на слушанията.
<code>/event/getInfo</code>	връща информация за дадено събитие (концерт на изпълнител)
<code>/geo/getTopTracks</code>	връща списък с всички песни, прослушани в даден географски регион сортирани в низходящ ред по брой слушания
<code>/geo/getTopArtists</code>	връща списък с всички изпълнители, прослушани в даден географски регион сортирани в низходящ ред по брой слушания.
<code>/geo/getEvents</code>	връща списък с информация за всички събития в даден географски регион

Таблица 4.2. Списък с по-важните точки за достъп от Last.fm API.

Информацията от *API*-то се предоставя за достъп в два формата - *XML* и *JSON* като конкретния формат, в който искаме да се върне резултата се задава като аргумент при всяко обръщение към сървъра. Съществуват множество библиотеки за различни езици и платформи, които улесняват достъпа до *API*-то, правейки редици параметри на извиквания към него конфигурационни опции.

Примерен резултат от обръщение към `/user/topTracks`, който връща резултат в *XML* формат е показан на *фигура 4.13*.

```

<toptracks user="RJ" type="overall">
  <track rank="1">
    <name>Learning to Live</name>
    <playcount>42</playcount>
    <mbid/>
    <url>
      http://www.last.fm/music/Dream+Theater/_/Learning+to+Live
    </url>
    <streamable fulltrack="0">1</streamable>
    <artist>
      <name>Dream Theater</name>
      <mbid>28503ab7-8bf2-4666-a7bd-2644bfc7cb1d</mbid>
      <url>http://www.last.fm/music/Dream+Theater</url>
    </artist>
    <image size="small">...</image>
    <image size="medium">...</image>
    <image size="large">...</image>
  </track>
  ...
</toptracks>

```

Фигура 4.13. Примерен резултат от заявка към Last.fm API

5. Оценяване

Оценяването на музикални препоръки отново се отличава от това на препоръки в други области, тъй като харесването на музика е силно субективно. За да бъдат резултатите представителни е необходимо те да са получени чрез тестване с максимално голям брой потребители (в случая на *онлайн* оценяване) и/или с достатъчно обемно множество от данни (за *офлайн* оценяване).

Оценяването на работата на изготвения прототип включва:

- офлайн оценяване на точността на препоръчването на музика, което се прави както посредством колаборативен подход, така и чрез подход, базиран на съдържание.
- онлайн оценяване на препоръчващата система с реални потребители

5.1 Офлайн оценяване

Както отбелязахме във втора глава, офлайн оценяването на препоръчваща услуга се извършва върху исторически данни на реални потребители, които обикновено се разделят на две части - обучаващо и тестово множество. Данните от обучаващото множество се ползват свободно от системата, за да може тя да генерира някакви резултати, а оценяването се състои в сравнение между тези резултати и данните от тестовото множество.

Нека напомним, че в процеса на изграждане на препоръчващата система за музика бяха събрани данни за музикалните интереси на над 25000 потребители на сайта *last.fm*. За всеки от тези потребители беше извлечена информация за 50-те най-слушани от тях песни или общо над 1250000 песни. Сред тези песни, уникални са малко над 112000.

Освен с данни за потребителите, системата разполага и с информация за таговете, с които са били означени различни песни и изпълнители. Събрани са данни за повече от 12000 изпълнители като за всеки от тях се съхранява списък с 50-те най-популярни тага (общо над 600000 тагвания). Приложението разполага и с информация за повече от 11000 песни, заедно с придвижаващите ги потребителски анотации. Поради огромното разнообразие от песни и невъзможността да бъдат включени достатъчно голям брой анотирани такива (предимно от гледна точка бързодействие на системата), подбраните песни са най-популярните такива, за всеки от извлечените тагове.

От така събранныте данни бяха взети последно извлечените 1000 потребителя и информацията за тях беше разделена на групи за тестване чрез 3-fold кросвалидация по следния начин:

- извлечените за всеки потребител 50 най-слушани песни са разделени на 3 групи от по 16.
- В рамките на три итерации се избира поредна група от споменатите 3 и се използва за тестово (валидиращо) множество, а останалите 2 групи се ползват за обучение на системата. За всяка така изградена опитна постановка се пресмятат стойностите на метриките *precision*, *recall*, F_1 и *nDCG* (*обезценена кумулативна печалба*).
- за всяка получена резултантна четворка стойности на метриките в рамките на трите итерации се взима тяхното средно-аритметично

Използвайки гореописаната схема беше оценена работата на препоръчващия алгоритъм, основан на сътрудничество в случая, когато близостта между потребителите се определя на база предпочтения към определени изпълнители. Получените резултати са както следва:

N \ метрика	Precision@25	Precision@10	Precision@5	Recall
12	0.043	0.059	0.072	0.068
17	0.041	0.056	0.070	0.065
22	0.040	0.053	0.067	0.063
27	0.039	0.052	0.065	0.061

Таблица 5.1. Резултати от проведеното онлайн оценяване на системата при колаборативен препоръчващ подход

N \ метрика	F1@25	F1@10	F1@5	nDCG
12	0.076	0.075	0.068	0.292
17	0.072	0.071	0.065	0.283
22	0.069	0.067	0.061	0.282
27	0.067	0.066	0.060	0.279

Таблица 5.2. Резултати от проведеното офлайн оценяване на системата при колаборативен препоръчващ подход

където с $Precision@K$ сме означили стойността на точността, пресметната само върху първите K на брой препоръчани обекта. Числото N , показва броят на съседите на текущия потребител, които са били взети предвид при изготвяне на препоръките.

Резултатите показват относително ниска зависимост на алгоритъма от броя на разглежданите съседи, като все пак най-високи стойности за разглежданите показатели се получават при $N = 12$. В тази връзка именно това е броят на съседите, който се ползва по подразбиране в реализирания прототип.

Както беше отбелоязано в предходните глави е възможно препоръчващата услуга да търси сходни потребители на база предпочитания към музикални изпълнители или въз основа на предпочитания към определени песни. В тази връзка беше проведено оценяване и на тази конфигурация на приложението, резултатите, от което са представени в *таблица 5.3* и *таблица 5.4*:

N \ метрика	Precision@25	Precision@10	Precision@5	Recall
12	0.036	0.050	0.064	0.056
17	0.034	0.046	0.055	0.053
22	0.032	0.042	0.051	0.050
27	0.030	0.038	0.046	0.047

Таблица 5.3. Резултати от проведеното офлайн оценяване на системата при колаборативен препоръчващ подход и сходство между песни

N \ метрика	F1@25	F1@10	F1@5	nDCG
12	0.062	0.063	0.058	0.265
17	0.058	0.058	0.052	0.241
22	0.055	0.053	0.047	0.223
27	0.052	0.048	0.043	0.209

Таблица 5.4. Резултати от проведеното офлайн оценяване на системата при колаборативен препоръчващ подход и сходство между песни

Вижда се, че резултатите получени при използването на песни за изчисляване на степента на сходство между потребителите са малко по-ниски от тези при ползването на изпълнители. От една страна това е малко неочеквано, тъй като слушането на едни и същи песни от двама души е доста по-показателно за сходство в музикалните им предпочитания от слушането на еднакви изпълнители. От друга, обаче, е обяснимо, тъй като ползването на изпълнители води до намирането на много повече съседи, а оттук и по-добри възможности за даването на добри препоръки.

При използване на алгоритъма, основан на съдържание (*потребителски анатации*) бяха получени малко по-високи стойности на изследваните параметри, отколкото при сътрудничество:

Precision@25	Precision@10	Precision@5	Recall
0.035	0.065	0.107	0.054

Таблица 5.5. Резултати от проведеното офлайн оценяване на системата при препоръчване, основано на съдържание

F1@25	F1@10	F1@5	nDCG
0.059	0.070	0.073	0.441

Таблица 5.6. Резултати от проведеното офлайн оценяване на системата при препоръчване, основано на съдържание

Такива резултати са очаквани, тъй като по принцип *content-based* препоръките са с по-висока точност, макар и незадължително по-добри като цяло.

За да анализираме получените резултати нека направим сравнение на работата на системата спрямо препоръчването на случайно избрано множество от песни. Както отбелязахме в предходните глави, информацията, с която боравим се състои от 112000 уникални песни в рамките, на които се извършват препоръките. Тъй като разглеждаме

точността измежду първите 5 препоръчани обекта, то нека разгледаме какъв е броят на различните петорки измежду множеството на препоръчвани обекти:

$$C_{112000}^5 = \frac{112000 \times 111999 \times 111998 \times 111997 \times 111996}{5 \times 4 \times 3 \times 2} \approx 10^{23}$$

От друга страна при оценяването разглеждаме сценарии, при които потребителят подава 32 песни на системата и тя му препоръчва 25. Тъй като получената от нас точност е от порядъка на 10% искаме да проверим какъв е броят на петорките, при които се постига такава или по-висока точност. Да отбележим, че $0.1 \times 5 \approx 1$, така че ни интересува броят на петорките измежду всички онези 10^{25} , в които има поне 1 обект измежду избрани 16. Различните начини, по които можем да изберем един, два, три, четири или пет обекта от множество с 16 са както следва:

$$\begin{aligned} C_{16}^1 &= 16 \\ C_{16}^2 &= 120 \\ C_{16}^3 &= 560 \\ C_{16}^4 &= 1820 \\ C_{16}^5 &= 4368 \end{aligned}$$

Тъй като тестването е проведено с 1000 потребителя, то “*dobrите*” случаи са общо $1000 \times (C_{16}^1 + C_{16}^2 + C_{16}^3 + C_{16}^4 + C_{16}^5) \approx 10^6$

В такъв случай вероятността да получим по-добра или сходна на текущата точност посредством препоръчване на случайни обекти е:

$$P = \frac{10^6}{10^{23}} = 10^{-17}$$

или с други думи значително по-малко от 1%.

Стойностите на получените в рамките на дипломната работа резултати се потвърждават допълнително и от сходно проучване (Cantador, Ivan, Belogin, Alejandro and Vallet, David, 2010), извършено от изследователи в университета в Мадрид. Използвайки подобно множество от данни от *last.fm* те успяват да постигнат точност между 0.04 и 0.16 в зависимост от метриката за сходство, която се използва.

5.2 Онлайн оценяване

Създаденият прототип беше оценен и в реални условия като за целта група от 27 различни потребители дададоха обратна връзка за предложените им от системата песни. За така получените данни бяха пресметнати стойностите за *Precision* и *Normalized Discounted Cumulative Gain* (*nDCG*), резултатите, за които са обобщени по-долу:

Precision@25	Precision@10	Precision@5	nDCG
0.112	0.134	0.167	0.381

Таблица 5.7. Резултати от проведеното онлайн оценяване при препоръчване, основано на сътрудничество

По-добрите резултатите при този тип оценяване спрямо получените при онлайн замерванията бихме могли да си обясним преди всичко с възможността потребителите да дадат оценка на всички препоръчани им песни. Това е важно, тъй като при онлайн експеримента проверяваме единствено каква част от препоръчаните обекти вече са били харесани (в конкретния случай - слушани в *last.fm*) и считаме, че всички останали не са релевантни. Макар и да е възможно това наистина да е така, в много случаи потребителят би харесал някаква част от тези “нерелевантни” препоръки, но просто данните, с които разполагаме към момента за него са прекалено ограничени. Напълно възможно е потребителят да е почитател на още много песни извън извлечените от профила му *50*, или пък да не ползва услугата на *last.fm* толкова активно и да има малък брой изслушани там песни, при все че слуша и много други. При онлайн оценяването имаме възможност да получим неговото мнение конкретно за всички песни, които сме му препоръчали. Благоприятстващ по-високите резултати фактор е и възможността за прослушване на направените предложения - по този начин е възможно потребителят да хареса мелодии, които досега не е чувал и да ги отбележи като “успешни” препоръки, при все че без тази възможност те биха му били напълно непознати.

Двадесет от горепосочените потребители дадоха отговори и на кратък въпросник, засягащ степента им на удовлетвореност от създадената система и мнението им за нейната работа. Въпросникът, както и получените отговори са описани в рамките на таблица 5.2.

Въпрос	Да	По-скоро да	Колебая се	По-скоро не	Не
Доволни ли сте от направените от системата препоръки след вход с <i>Facebook</i> акаунт?	20%	55%	20%	5%	0%
Доволни ли сте от направените от системата препоръки след вход с <i>last.fm</i> акаунт?	25%	60%	15%	0%	0%
Доволни ли сте от направените от системата	15%	55%	20%	10%	0%

препоръки след оценяване на предварително зададеното множество от песни?					
Доволни ли сте като цяло направените от системата препоръки (независимо от използванието източници на данни)?	20%	65%	10%	5%	0%
Получавахте ли по-добри препоръки с оценяването на повече на брой песни?	20%	55%	25%	0%	0%
Доволни ли сте от цялостната работа на изграденото приложение?	35%	60%	5%	0%	0%
Бихте ли използвали подобно приложение?	20%	70%	10%	0%	0%
Изслушахте ли някоя от препоръчените песни?	35%	0%	0%	0%	65%
Използвахте ли възможността за избор на използван препоръчващ подход?	25%	0%	0%	0%	75%
Използвахте ли възможността за настройване параметрите на използвания препоръчващ подход?	20%	0%	0%	0%	80%

Таблица 5.8. Отговори на потребителската анкета относно работата на препоръчващата система

Като цяло бихме могли да обобщим, че потребителите по-скоро харесват системата и са доволни от нейната дейност. Немалка част от хората споделят, че биха използвали отново подобна услуга, което валидира идеята на дипломната работа.

Резултатите относно броя на хората, които са ползвали настройките на системата ни навеждат на мисълта, че подобни опции явно не са чак толкова необходими. Оценяваме, обаче, че все пак те ни дадоха възможност да направим сравнителен преглед на работата на различни възможни алгоритми. Получените резултати за частта от хората, които са

изслушали някои от направените им препоръки потвърждава обяснението ни за получените при онлайн оценяването резултати.

5.3 Оценка на работата на системата спрямо стандартните критерии за оценяване

В рамките на тази част от дипломната работа ще анализираме работата на препоръчващата система спрямо критериите, които описахме в глава 2, а именно:

Точност. Получените при онлайн и офлайн оценяването резултати показват задоволителна степен на точност, която е сравнима с тази получена при сходни експерименти.

Покритие. Предвид огромното множество от съществуващи песни е невъзможно да се постигне пълно покритие на всички от тях при препоръчване. Системата може да предлага само песни в рамките на извлеченията от *last.fm* 112000 такива. Все пак считаме, че подобна извадка е достатъчно голяма и представителна.

Увереност. Предвид на факта, че при препоръчване системата работи с данни за над 25000 потребители и 112000 песни бихме могли да кажем, че тя работи с висока увереност в направените препоръки.

Разнообразие. Разнообразието в направените от системата препоръки, съвпада с това в слушаните от потребителите на *last.fm* песни. Към момента не се подсигурява, че препоръчваните мелодии са от различни стилове и/или автори, но текущата имплементация би позволила лесно налагането на подобно ограничение. Трябва да отбележим и, че събранныте от *last.fm* данни би трябвало да са разнородни, тъй като са събираны от потребители, които живеят в различни географски региони и нямат много общо помежду си.

Скалируемост. Сървърната част от системата е изградена с помощта на *NodeJS* и *MongoDB*. Асинхронния характер на работата на *NodeJS* подсигурява, че системата би могла да се справи и с огромен брой заявки без да блокира. От гледна точка на достъпа до данни - *MongoDB* дава лесни възможности за т.нар. *sharding* (разпределение на данните от базата върху множество сървъри) както и репликация (копиране на едни и същи данни на множество машини). Посредством тези два метода базата може да отговаря на милиони потребителски заявки в минута.

Зашита на данните. При изготвянето на препоръки на потребителите не се разкрива информация за другите потребители, от които реално са получили препоръки. Системата не съхранява никакви лични данни за своите ползватели, освен списък с оценените от тях песни в рамките на системата и/или *last.fm* и *facebook*. Този списък не може да бъде достъпен по никакъв начин през наличния уеб интерфейс.

Изненадващи препоръки. Тъй като по подразбиране системата работи с препоръчване чрез сътрудничество е напълно възможно потребителят да получи т.нар. изненадващи

препоръки - обекти, които той ще хареса, но нямат общо с досегашната му потребителска история.

6. Заключение

В рамките на настоящата глава ще разгледаме в каква степен изградената препоръчваща система отговаря на първоначалните изисквания, поставени към нея, резултатите от нейната работа и ще дадем някои насоки за бъдещото й развитие. Ще направим обзор на постигнатото в дипломната работа.

6.1 Обобщение

Дипломната работа представя препоръчваща система, чиято архитектура е съобразена с технологичните изисквания. Създаден е компонент за събиране на данни, с помощта на който е извлечена значителна информация за потребители, изпълнители и песни от сайта *last.fm*. Изградена е клиентска, сървърна и препоръчваща част, във всяка, от които има ясно разделение на отговорностите. Клиентската част има стилен и удобен за ползване потребителски интерфейс, а уеб услугите, предоставяни от сървъра са достатъчно универсални, за да позволяват използването им от клиенти, написани от трети страни. Препоръчващата услуга може да работи с двата най-разпространени алгоритмични подхода и да агрегира резултатите от тяхната работа, за да бъдат удобни за бъдещ анализ. Налични са и възможности за настройване на нейната дейност.

Изпълнени са всички функционални изисквания, изложени в *трета глава*, като считаме, че и нефункционалните такива също са налице. Изчисленията, които се извършват при потребителска заявка са достатъчно минимални и оптимизирани, т.ч. да се получи отговор в рамките на няколко секунди. Използваните технологии - *MongoDB*, *NodeJS*, *Twitter Bootstrap* и *AngularJS* осигуряват лесни възможности за бъдещо скалиране на системата, като същевременно подсигуряват и нейната преносимост. Целият код на изграденото приложение е свободно достъпен в рамките на *Github* като към него е приложена и начална версия на документация, която цели привличането на нови разработчици.

От гледна точка на теоретичната обосновка на използваните алгоритми, метрики и решения е направен подробен обзор на всички по-популярни такива като е включен и сравнителен преглед в случаите, когато това е приложимо. Направена е детайлна оценка на работата на изградения прототип с помощта на метрики, чиято употреба е предварително описана и обоснована. Налично е сравнение между създадената препоръчваща услуга и други подобни такива.

6.2 Насоки за усъвършенстване на системата и бъдещо развитие

В този вид препоръчващата система изпълнява основните изисквания, които са отправени към нея, но все пак оставя място за желани подобрения. Една от посоките, в които може да се търсят такива подобрения е постигането на по-високо качество на дадените препоръки. На първо време това може да се постигне по следните начини:

- събиране на по-голямо начално множество данни - това може да стане като се разгледат музикалните интереси на повече на брой потребители на *last.fm*. Този подход би довел до подобреие както на точността на правените препоръки, така и на тяхното разнообразие.
- интегриране на хибриден препоръчващ подход. Към момента системата може да работи както на основа съдържание, така и чрез сътрудничество, но контролът кой от двата подхода да се ползва е в потребителя. Макар и това да е подходящо за експериментални цели, бихме постигнали оптималност на препоръките, ако това решение се взима автоматично от системата. На първо четене най-подходяща изглежда т.нар. хибридизация чрез претегляне, при която на двата препоръчващи подхода се дават равни начални тегла, които се оптимизират в процеса на работа на приложението.

Друг аспект, в който може да се желае повече от приложението е неговото бързодействие. Към момента даването на препоръки става в рамките на няколко секунди, като огромната част от това време се изразходва за извлечение и обновяване на съответни записи от базата данни. Това време може да се минимизира чрез промените в схемата и колекциите, които се съхраняват в *Mongo*, като е възможно това да трябва да стане с цената на допускане на излишество.

Допълнително подобреие в работата на препоръчващата система би могло да настъпи и в резултат на добавяне на нови източници за потребителски данни. Понастоящем външните източници на такива данни са единствено *Facebook* и *last.fm*, но в първият сайт обикновено потребителите не оставят особена информация за музикалните си преподпочитания, а *last.fm* става все по-слабо използвана услуга. В същото време сайтове като *Youtube* и *Spotify* се радват на огромна популярност и предоставят интерфейс, през който да се извлече информация за потребителите им. В този ред на мисли бъдеща интеграция с гореизброените системи би била от полза.

В по-далечна перспектива работата на препоръчващата услуга би могла да се подобри допълнително като се утилизира по-пълноценно и информацията, която се извлича от *Facebook* профила на потребителя. Допълнителните данни, от които работата ѝ би спечелила са както демографски (възраст, пол, местонахождение), така и социални (професия, приятели).

Литература

1. Adomavicius, Gediminas and Tuzhilin, Alexander, 2005, Toward the next generation of recommender systems: a survey of the state-of-the-art and possible extensions. *IEEE Transactions on Knowledge and Data Engineering*. 2005. Vol. 17, no. 6, p. 734-749. DOI 10.1109/tkde.2005.99. Institute of Electrical & Electronics Engineers (IEEE)
2. Angularjs.org, 2015, Introduction to AngularJS. [online]. 2015. [Accessed 8 February 2015]. Available from: <https://docs.angularjs.org/guide/introduction>
3. Angularjs.org, 2015, Conceptual overview of AngularJS. [online]. 2015. [Accessed 8 February 2015]. Available from: <https://docs.angularjs.org/guide/concepts>
4. Asanov, Danair, 2011, *Algorithms and methods in Recommender systems*. Undergraduate. Berlin Institute of Technology.
5. Basu, Chumki, Hirsh, Haym and Cohen, William, 1998, Recommendation as Classification: Using Social and Content-Based Information in Recommendation. In : *Proceedings of the 15th National Conference on Artificial Intelligence*. Madison, WI. 1998. p. 714-720
6. Burke, Robin, 2002, Hybrid Recommender systems, Survey and Experiments. *User modeling and User-adAPTER interaction*. 2002. Vol. 12, no. 4, p. 331-370.
7. Cantador, Ivan, Belogin, Alejandro and Vallet, David, 2010, Content-based recommendation in social-tagging systems. In : *Recsys*. Barcelona, Spain. 2010. p. 237-240.
8. Crockford, Douglas, 2006, The application/json Media Type for Javascript Object Notation (JSON). *RFC 4627*. 1.
9. De Wit, Joost, 2008, *Evaluating recommender systems*. Undergraduate. University of Twente.
10. Facebook Developers, 2015, Graph API Changelog. [online]. 2015. [Accessed 8 February 2015]. Available from: <https://developers.facebook.com/docs/apps/changelog>
11. Goldberg, David, Nichols, David, Oki, Brian and Terry, Douglas, 1992, Using collaborative filtering to weave an information tapestry. *Commun. ACM*. 1992. Vol. 35, no. 12, p. 61-70. DOI 10.1145/138859.138867. Association for Computing Machinery (ACM)
12. Hernández del Olmo, Félix and Gaudioso, Elena, 2008, Evaluation of recommender systems: A new approach. *Expert Systems with Applications*. 2008. Vol. 35, no. 3, p. 790-804. DOI 10.1016/j.eswa.2007.07.047. Elsevier BV
13. Joyce, John, 2006, The music genome project. *Scientific Computing*. 2006. Vol. 23, no. 10, p. 40-41.
14. Kwon, Hyeong-Joon, Lee, Tae-Hoon and Hong, Kwang-Seok, 2009, Improved memory-based collaborative filtering using entropy-based similarity measures. In : *Proceedings of the 2009 International Symposium on Web Information Systems and*

- Applications*. Nanchang, China : Academy Publisher. 2009.
15. Last.fm, 2005, API – Last.fm. [online]. 2005. [Accessed 8 February 2015]. Available from: <http://www.last.fm/api>
 16. Layton, Julia, 2006, How Pandora Radio Works. *HowStuffWorks* [online]. 2006. [Accessed 8 February 2015]. Available from: <http://computer.howstuffworks.com/internet/basics/pandora.htm>
 17. Linden, Greg, Smith, Brent and York, Jeremy, 2003, Amazon.com recommendations: item-to-item collaborative filtering. *IEEE Internet Comput.* 2003. Vol. 7, no. 1, p. 76-80. DOI 10.1109/mic.2003.1167344. Institute of Electrical & Electronics Engineers (IEEE)
 18. Mateo, Sergio and Assent, Ira, 2010, *Collaborative filtering in social networks*. Undergraduate. Aalborg University.
 19. Mongdb.org, 2015, Introduction to MongoDB. [online]. 2015. [Accessed 8 February 2015]. Available from: <http://www.mongodb.org/about/introduction/>
 20. Netflix, 2015, History of Netflix. [online]. 2015. [Accessed 8 February 2015]. Available from: <https://pr.netflix.com/WebClient/loginPageSalesNetWorksAction.do?contentGroupId=10477>
 21. Netflixprize website, 2009, Netflix Prize: Leaderboard. [online]. 2009. [Accessed 8 February 2015]. Available from: <http://www.netflixprize.com/leaderboard?showtest=t>
 22. Nodejs.org, 2015, About | Node.js. [online]. 2015. [Accessed 8 February 2015]. Available from: <http://nodejs.org/about/>
 23. O'Connor, Brendan, 2012, Cosine similarity, Pearson correlation and OLS coefficients. *Brenocon*[online]. 2012. [Accessed 8 February 2015]. Available from: <http://brenocon.com/blog/2012/03/cosine-similarity-pearson-correlation-and-ols-coefficients>
 24. Papagelis, Manos, 2012, WEB APP ARCHITECTURES: MULTI-TIER (2-TIER, 3-TIER) & MVC. . Presentation. 2012.
 25. Resnick, Paul and Varian, Hal Ronald, 1997, Recommender systems. *Commun. ACM*. 1997. Vol. 40, no. 3, p. 56-58. DOI 10.1145/245108.245121. Association for Computing Machinery (ACM)
 26. Sarwar, Badrul, Karypis, George, Konstan, Joseph and Riedl, John, 2010, Item-Based Collaborative Filtering Recommendation Algorithms. . Presentation. 2010.
 27. Smyth, Barry and Cotter, Paul, 2000, A personalized television listings service. *Commun. ACM*. 2000. Vol. 43, no. 8, p. 107-111. DOI 10.1145/345124.345161. Association for Computing Machinery (ACM)
 28. Song, Yading, Dixon, Simon and Pearce, Marcus, 2012, A survey of music recommendation

- Systems and future perspectives. In : *CMMR*. London. 2012.
29. Spearman, Charles, 1904, The Proof and Measurement of Association between Two Things. *The American Journal of Psychology*. 1904. Vol. 15, no. 1, p. 72. DOI 10.2307/1412159. JSTOR
30. Tech-tech-news.blogspot.com, 2013, How many pages on the internet? - Tech News. [online]. 2013. [Accessed 8 February 2015]. Available from: <http://tech-tech-news.blogspot.com/2013/03/how-many-pages-on-internet.html>
31. van Meteren, Robin and van Someren, Maarten, 2000, Using content-based filtering for recommendation. In : *Machine Learning in the New Information Age: MLnet/ECML2000*. Barcelona, Spain. 2000. p. 47-56.
32. Wikipedia, 2015, GroupLens Research. [online]. 2015. [Accessed 8 February 2015]. Available from: http://en.wikipedia.org/wiki/GroupLens_Research
33. Wikipedia, 2015, Last.fm. [online]. 2015. [Accessed 8 February 2015]. Available from: <http://en.wikipedia.org/wiki/Last.fm>
34. Wikipedia, 2015, Bootstrap (front-end framework). [online]. 2015. [Accessed 8 February 2015]. Available from: http://en.wikipedia.org/wiki/Bootstrap_%28front-end_framework%29
35. Wikipedia, 2015, Multilayered architecture. [online]. 2015. [Accessed 8 February 2015]. Available from: http://en.wikipedia.org/wiki/Multilayered_architecture
36. Wilson, Tracey, 2007, Netflix recommendation explained. *HowStuffWorks* [online]. 2007. [Accessed 8 February 2015]. Available from: <http://electronics.howstuffworks.com/netflix2.htm>
37. Yin, Hongzhi, Cui, Bin, Li, Jing, Yao, Junjie and Chen, Chen, 2012, Challenging the long tail recommendation. *Proceedings of the VLDB Endowment*. 2012. Vol. 5, no. 9, p. 896-907. DOI 10.14778/2311906.2311916. VLDB Endowment
38. Zhou, Bing and Yao, Yiyu, 2009, Evaluating information retrieval system performance based on user preference. *Journal of Intelligent Information Systems*. 2009. Vol. 34, no. 3, p. 227-248. DOI 10.1007/s10844-009-0096-5. Springer Science + Business Media
39. Zhou, Ke, Zha, Hongyuan, Chang, Yi and Xue, Gui-Rong, 2008, Learning the gain values and discount factors of discounted cumulative gains. In : *SIGIR*. Singapore. 2008. p. 227-248.

Приложения

Потребителски интерфейс

The screenshot shows the homepage of the 'Pick me a song' music recommender system. At the top, there is a navigation bar with links for 'Home', 'About', and 'Settings'. Below the navigation bar, a large heading reads 'Find new music. Get personalized recommendations.' A subtext message states: 'We need some info about your music tastes in order to make recommendations. We can get it only if you would:'. Three options are listed: 'Rate' (with a star icon), 'Link' (with a Last.fm icon), and 'Log In' (with a Facebook icon). At the bottom of the page, a footer bar contains the text 'Pick me a song - music recommender system'.

Приложение 1. Начална страница на изградената препоръчваща система

The screenshot shows the 'About' page of the 'Pick me a song' music recommender system. At the top, there is a navigation bar with links for 'Home', 'About', and 'Settings'. Below the navigation bar, a section titled 'What it is?' provides a description of the system: 'Pick me a song is a music recommender system that allows you to enter information about your favourite musicians and afterwards provides you with personalized suggestions based on that information. You can enter your music tastes either manually - by typing in a list of artists along with their importance for you or, conversely, we can pull this info from your Facebook profile.' Another section titled 'How it works?' describes the algorithm: 'The system is based on a Collaborative Filtering algorithm that tries to find a list of users that have similar tastes to you and recommend you those of their favourite songs that you still haven't listened. The algorithm is using data from last.fm in order to find similar users. Additionally, you can change the algorithm parameters from the settings page or even change the algorithm type altogether. The other option is for the system to use a version of Content-based Filtering which is based on the tags users have given to the objects in the system.' At the bottom of the page, a footer bar contains the text 'Pick me a song - music recommender system'.

Приложение 2. Страница с обща информация за препоръчвашата система

Settings

You can control the algorithm parameters that are used when making recommendations.

Number of neighbours

Each time we make a recommendation we consider a number of users that are similar to you. Feel free to change this having in mind that lower count will result in a limited set of recommendations and considerably higher will result in broader, but less relevant ones. The default value is **17**.

Recommendation algorithm type

The work of the system can be based on 2 different algorithmic approaches - [Collaborative filtering](#) and [Content-based filtering](#). The first one finds suitable music for you by comparing your taste to the thousands of other users have and recommends you their favourite songs. The latter uses meta information about the music you like and the music we have information about in order to find tracks that match your taste.

Used algorithm type:

[Collaborative filtering](#)[Content-based filtering](#)

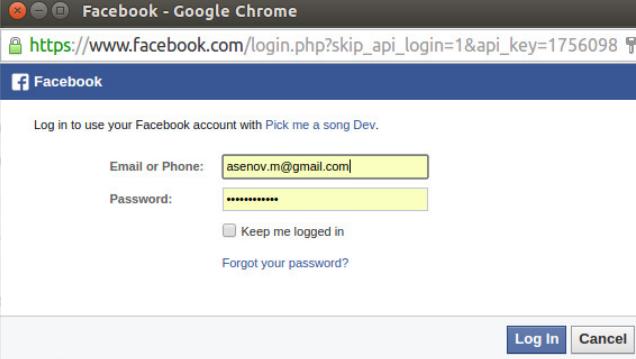
Pick me a song - music recommender system

Приложение 3. Страница с настройки на препоръчващата система

Pick me a song

Home About Settings

Find new music. Get personalized recommendations.



We need some info about your music tastes in order to make recommendations.
We can get it only if you would:

Rate a couple of songs to allow us get a better taste of what music you enjoy.

Link your Last.fm profile in order to allow us gather what tracks you've listened.

OS simoalttime Go

Fetch f info about your music likes from your Facebook.

Pick me a song - music recommender system

Приложение 4. Извличане на потребителски данни от Facebook

Pick me a song

Home About Settings

Find new music. Get personalized recommendations.

We need some info about your music tastes in order to make recommendations.
We can get it only if you would:

Rate a couple of songs to allow us get a better taste of what music you enjoy.

Link your Last.fm profile in order to allow us gather what tracks you've listened.

OS simoalttime Go

Fetch f info about your music likes from your Facebook.

Pick me a song - music recommender system

Приложение 5. Извличане на потребителски данни от *last.fm*

Pick me a song

Home About Settings

Rate some items.

This is a list of the most listened tracks in the system. Let us know which ones you like and once you are ready feel free to

Artist Name	Song Name	Listen	Album Cover	Rate
Lady Gaga	Bad Romance	URL		Please, give this song a score between 1 and 5
Kings of Leon	Use Somebody	URL		Please, give this song a score between 1 and 5
Red Hot Chili Peppers	Californication	URL		Please, give this song a score between 1 and 5

Приложение 6. Списък с неперсонализирани препоръки, които потребителят може да оцени

Pick me a song

[Home](#)[About](#)[Settings](#)

Your recommended items.

This is a list of tracks we think you may like. If you happen to listen to any of them please rate them and [Send](#) us some feedback.

Artist Name	Song Name	Listen	Album Cover	Rate
Lady Gaga	Poker Face	URL		Please, give this song a score between 1 and 5 
The Beatles	Lucy in the Sky with Diamonds	URL		Please, give this song a score between 1 and 5 
Selena Gomez	Come & Get It	URL		Please, give this song a score between 1 and 5 

Приложение 7. Списък с персонализирани препоръки някои, от които са били оценени от потребителя.