Пловдивски Университет

Факултет
"Математика и Информатика"

Дипломна работа

Система за препоръки на мобилни приложения

Дипломант:

Венелин Вълков

фак. № 1201407012

Научен ръководител:

проф. д-р Станимир Стоянов

кат. „Компютърни технологии“

03. 07. 2013 г.

Съдържание

Въведение **3**

1 Създаване на системата **4**

1.1 Проучване 4

1.2 Моделиране 8

1.3 Използвани технологии 12

1.4 Реализация 14

2 Ръководство за потребителя **23**

Заключение **24**

Списък на фигурите **25**

А Използвани съкращения **27**

Б Библиография **28**

Резюме

Въведение

Глава 1

Създаване на системата

1.1 Проучване

Със създаване на Системи за препоръки (Recommendation systems) (СП) се занимава Машинно обучение (Machine learning) (МО) (отрасъл от Компютърни науки (Computer science) (КН)). СП са интересна алтернатива на алгоритми за търсене, тъй като те намират обекти, които може да нямат общо с термина за търсене. Тези системи, предоставят списък от препоръки, използвайки Кооперативно филтриране (Collaborative filtering) (КФ) или Филтриране, базирано на съдържание (Content-based filtering) (ФБС).

КФ подход, при който се изгражда модел на съществуваща история за потребителя и решения, взети от подобни потребители. Този модел се използва за предвиждане на какви нови обекти може да се харесат на потребителя. [Melville]

ФБС използва серии от абстрактни характеристики на обектите, за да препоръча подобни обекти. [Mooney]

Пример[1] за използване на двата подхода са *Last.fm*[2] и *Pandora Radio*[3]

[1] `http://en.wikipedia.org/wiki/Recommender_system`
[2] `http://www.last.fm/`
[3] `http://www.pandora.com/`

- Pandora използва характеристиките на песен или артист, за да избере радио станция, която предоставя песни с подобни характеристики. Мнението на потребителя се използва за определяне на тежест на различните характеристики на радио стацията. Pandora е пример за ФБС

- Last.fm създава виртуална радио станция на базата на историята на потребителя (какви песни и групи е слушал). Тя се сравнява с историята и предпочитанията на други потребители, като по този начин, системата предоставя нови песни. Last.fm е пример за КФ

Двата подхода имат слаби и силни страни. КФ се нуждае от голямо количество информация, за да направи качествени препоръки. ФБС има нужда от малко данни, за да започне работата си, но е лимитиран до първоначално подадени данни.

1.1.1 Преглед на КФ

КФ е подход за създаване на СП, който се налага в практическите имплементации на подобни системи. Основно предимство на метода е, че не разчита на анализиране на съдържание от машина и поради тази причина има възможност за точно препоръчване на сложни обекти (напр. филми), като не е необходимо разбиране на самия обект.

Използват се различни алгоритми за оценяване сходността на два обекта в СП. Два от най-използваните са к-близки съседи (k-nearest neighbours) (КБС) и Свързаност на Пиърсън (Pearson Correlation) (СП).

КБС е един от най-простите от всички МО алгоритми. Обектът е класифициран спрямо мажоратирен вот от неговите съседи, като той е поставен между най-близкия клас от възможните K.

СП е мярка за линейна свързаност между две променливи в интервала [-1:+1].

Проблеми с КФ

КФ има три основни проблема: "студен старт скалируемост и рядкост

- студен старт системата изисква много информация, за да предостави добри препоръки. В началото на нейното съществуване, обикновено, такава не е налична.

- скалируемост - много СП оперират върху милиони потребители и обекти. Нужна е голяма изчислителна мощ, за да се предоставят точни препоръки

- рядкост броят на обектите, обикновено, е в пъти по-голям от този на потребителите. Дори и най-активните потребители оценяват само малко подмножество от обектите. Това води до малък брой оценки за отделните обекти

1.1.2 Съществуващи СП

Amazon

Amazon[4] използва СП за препоръчване на нови продукти на потребителите си. Предоставя такива, които смята, че ще са интересни за тях. Използва се ФБС.

Интересно за Amazon е, че има над 29 милиона потребители и няколкомилионен каталог от продукти. Размер данни, който затруднява голяма част от алгоритмите за намиране на препоръки. Разработчиците на Amazon се справят с този проблем като използват офлайн създаване на таблици със сходни продукти. В резултат на това, алгоритъмът се грижи само да извлече данни от таблицата, когато те са нужни.

Youtube

Youtube[5] използва СП, за да предостави на потребителите си видео записи на интересни за тях теми. Системата се опитва да максимизира броя видеота, които потребителят гледа и времето, което прекарва в сайта. Използва се хибридна версия между КФ и ФБС.

Лимитиращи фактори са взети под предвид. Системата предоставя само определен брой видео клипове от същия потребител. Използват се уникални за

[4] `http://www.amazon.com/`
[5] `http://www.youtube.com/`

потребителя предпочитания, неговата история, брой изгледани видеота и време, по което са гледани, за да се увеличи възможността за добра препоръка.

1.1.3 Android и пазарът за мобилни приложения

Android[6] е мобилна Операционна система (Operating system) (ОС) базирана на Линукс, създадена предимно за мобилни устройства, като таблети и телефони. Тя се разработва от Google, които през 2005 я закупуват от малка компания [Elgin]. С появата на Android се основава и Open Handset Alliance[7], организация, която се грижи за развитието на мобилните технологии.

Нарастващ брой приложения

Android нараства бързо след 2009 г., когато представлява само 2.8%[8] от пазара за мобилни устройства. В края на 2010 г. представя 33%[9] от него. В края на 2012 г., Android е на челно място с 75% [IDC]. Активирани са над 900 милиона Android устройства [Google].

Броят приложения нараства заедно с популярността на Android. Техният брой за Февруари 2013 г. е 800,000 [10]. Броят сваляния е около 40 милиарда.

Нарастването на Android означава, че при създаване на СП посредством КФ ще има достатъчно информация за даване на добри препоръки. *Google*, разбира се, вече вграждат подобни методи в пазара на Android. Те са лимитирани до данни, които се предлагат само от техни продукти. Приятелите във *Facebook*[11], също биха допринесли за предоставяне на по-добри препоръки при създаване на подобна система. Това не се използва от *Google*.

[6] http://www.android.com

[7] http://www.openhandsetalliance.com/

[8] http://appleinsider.com/articles/09/08/21/canalys_iphone_outsold_all_windows_mobile_phones_in_q2_2009.html

[9] http://www.canalys.com/newsroom/google%E2%80%99s-android-becomes-world%E2%80%99s-leading-smart-phone-platform

[10] www.rssphone.com/google-play-store-800000-apps-and-overtake-apple-appstore/

[11] http://www.facebook.com

1.2 Моделиране

За създаване на гореописаната система са нужни два големи отделни компонента:

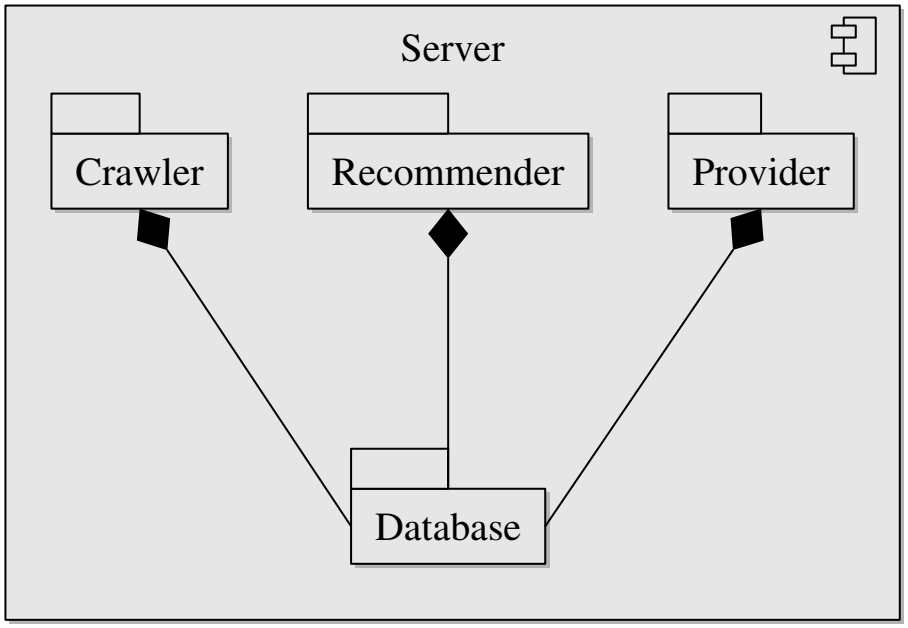
- *Сървър* - обработва данни за отделните приложения и потребители на системата. Грижи се за събирането, запазването, предоставянето и анализирането на препоръки за клиентската част. Предоставя Приложим интерфейс за програмиране (Application Programming Interface) (ПИП) за работа с данните, които съхранява. От своя страна, този компонент може да бъде разделен на следните, по-малки такива:

  - Database е компонент, който предоставя услуга за съхранение на данни. Той играе ролята на косвен(implicit) интерфейс, с който да работят останалите компоненти от сървърната част.
  - Spider се грижи за събиране на информация за различните приложения от уеб страници. Данните се запазват за по-нататъшен анализ.
  - Recommender е сърцето на системата. Подобно на примера на Amazon, този модул предварително създава препоръки и ги запазва за употреба от други части на системата. Използва данни събрани от предишния модул, както и такива предоставени от потребителите на системата.
  - Provider предоставя ПИП за комуникация със сървърната част. Той е единственият начин за обмяна на информация. Негова цел е да остане независим от клиенти и същевременно да предостави лесен и бърз начин за работа.

  Фигура 1.1 показва UML диаграма на сървърния компонент.

- *Клиент* предоставя, може би, най-важната част от системата - това, с което потребителя взаимодейства. Скрита за него остава комуникацията със сървъра. В този модул се разглеждат следните подмодули:

  - GUI се грижи за представяне на препоръките, инсталиране на приложения и оценяването им. Този модул трябва да предоставя идентично



Фигура 1.1: UML диаграма на сървърния компонент

изживяване за потребителя, независимо от устройството, с което той разполага, т.е. трябва да е високо адаптивен.

– Watcher събира допълнителна информация за потребителя, която да спомогне за създаване на по-точни препоръки. Инсталирани приложения, време, за което се посещават, брой посещения, са част от събираните данни. Модулът спомага за идентифициране на потребителя, което премахва нуждата от регистрация.

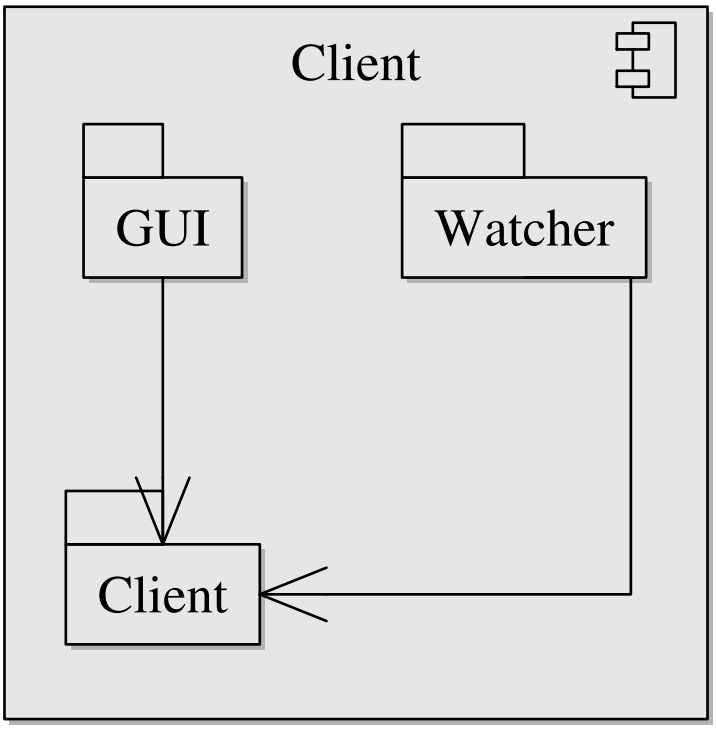
– Client комуникира със сървърната част, като предоставя информация за потребителя и извлича препоръки. Важно за модула е да извършва функциите си по напълно прозрачен начин за потребителя.

Естествен въпрос, който може да се породи в подобна ситуация е "Защо клиентът изпълнява толкова малка част от тежките изчислителни процеси?". При по-детайлно вглеждане в хардуера на днешните мобилни телефони от висок клас(напр. *Samsung Galaxy S4*[12]), става ясно, че съществува модел с 4-ядрен централен процесор и два гигабайта вътрешна памет. Това е съпоставима изчислителна мощ на преносим компютър от преди три

[12] `http://www.gsmarena.com/samsung_i9500_galaxy_s4-5125.php`

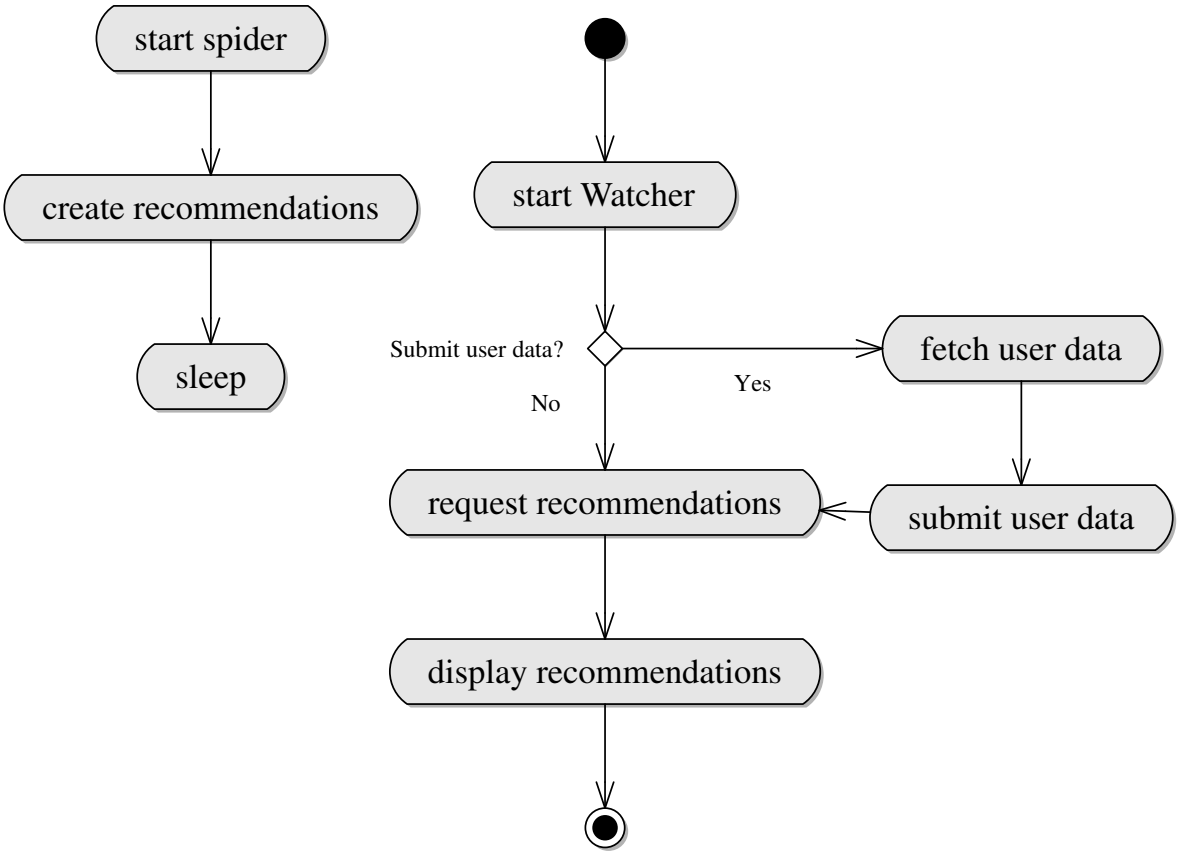
години. Разработчиците на мобилни приложения нямат достъп до пълния капацитет на устройството. Допълнителните ядра се грижит по-скоро за това телефонът да остане използваем и не предоставят пълен контрол върху хардуера [Gupta]. Остава възможността за създаване на хибридни системи, в които клиентът да допринася към по-тежките изчислителни процеси. При създаване на подобно решение, допълнителна процесорна мощ ще трябва да се отдели за разпределение на задачите, което може да намали осезаемо ефективността на системата.

Фигура 1.2 показва UML диаграма на клиента.



Фигура 1.2: UML диаграма на клиента

Фигура 1.3 показва дейностна(Activity) диаграма, която описва начина, по който предложенията за приложения се предоставят на потребителя. Ясно се вижда, че процесите, обвързани със събиране и анализиране на данни, са отделени от останалата част на системата.



Фигура 1.3: Activity диаграма на системата

1.2.1 Основен модел на системата

На фигура 1.4 е изобразена клас диаграма с основните класове от системата. От нея се виждат двата основни компонента - App и User. Всички останали, може да се разглеждат като спомагателни за тях и имат сравнително проста структура.

1.3 Използвани технологии

1.3.1 Сървърен модул

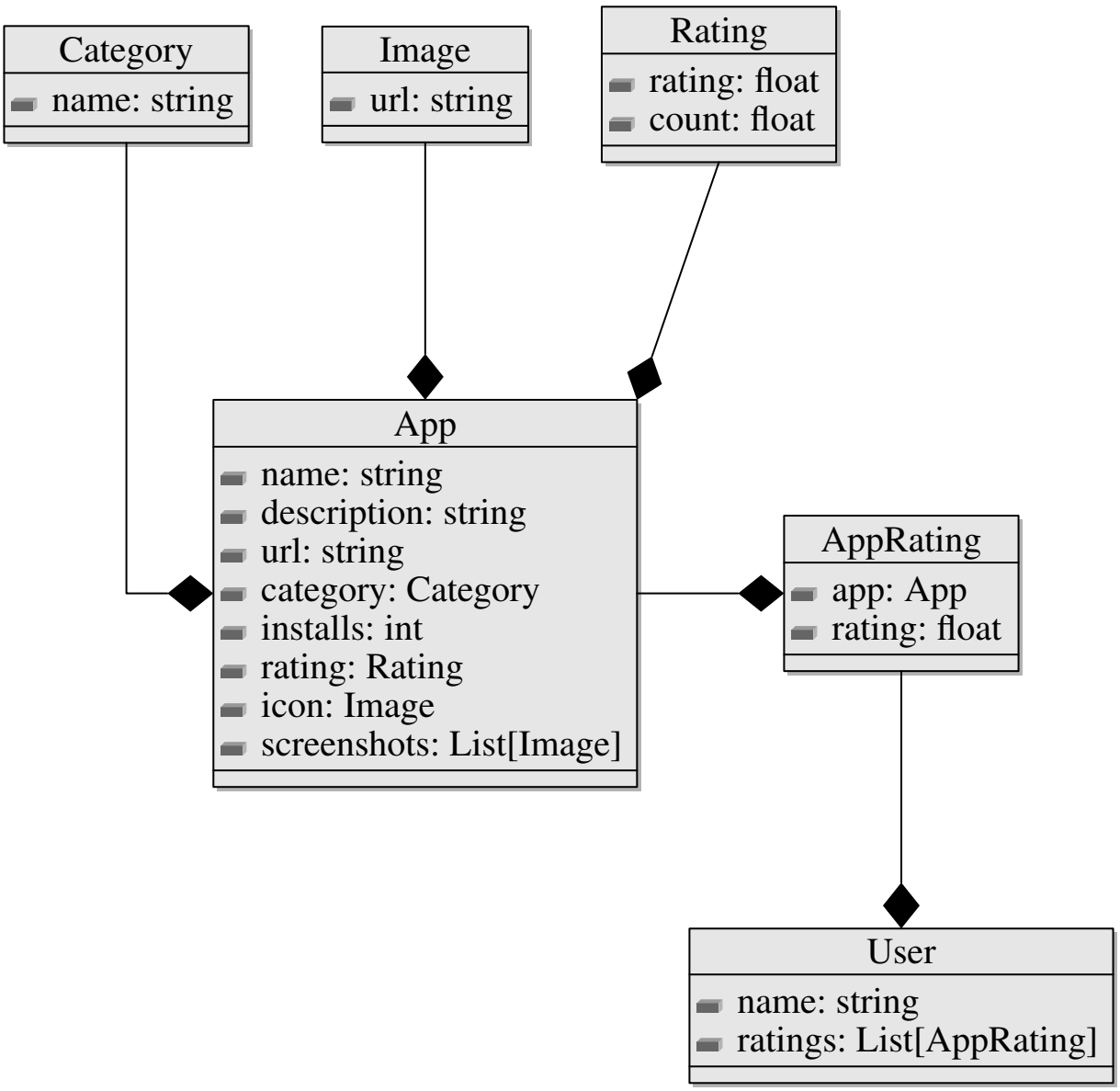
Сървърната част от системата е необходимо да бъде в постоянна готовност за обработка на нови данни, извличане на такива и предоставянето им на някой от клиентите. За извършване на своята функция, всяка от посочените операции, комуникира с услуга, която предоставя запазване и извличане на информация. Тя е критична точка в системата, тъй като всеки друг сървърен модул зависи от нейното правилно действие.

MongoDB

MongoDB[13] е документно-ориентирана база от данни, която не използва SQL[14]. Тя предоставя скалируемост, копиране(replication), индекси, извличане на информация и др. MongoDB запазва данни в JSON[15] подобен формат, като документи, вместо таблици.

Предоставени са множество библиотеки за комуникация с данните за популярните програмни езици[16]. Възможна е и връзка чрез REST и HTTP[17]. Системата предоставя бързо изграждане на прототипи и реални приложения като нуждата от създаване на таблици и схеми се избягва напълно. Негативен фактор е липсата на ясно изразен резултат от това дали даден запис е бил съхранен успешно [Sirer]

[13] `http://www.mongodb.org/`
[14] `http://www.w3schools.com/SQl/default.asp`
[15] `http://json.org/`
[16] `http://docs.mongodb.org/ecosystem/drivers/`
[17] `http://www.mongodb.org/display/DOCS/Http+Interface`



Фигура 1.4: Клас диаграма на модела на системата

MongoDB се използва в множество бизнес проекти и продължава да трупа популярност. Предоставя се и много добра интеграция от водещи доставчици на облъчни услуги: Amazon Web Services, Microsoft Windows Azure, Rackspace Cloud, Red Hat OpenShift and VMware Cloud Foundry и други [Marketwire].

Python

scrapy

mongoengine

bottle

1.3.2 Клиент

Android SDK

retrofit

1.4 Реализация

За постигане на целите на описаната система може да се подходи по различни начини (както и към повечето софтуерни проблеми), но естеството на проблема и незнанието от страна на автора за пълна спецификация на крайния продукт са предпоставки за използване на Гъвкава разработка на софтуер (Agile software development) (ГРС).

ГРС е процес, съвкупност от множество практики при разработка на софтуер. Той позволява изучаване на областта, за която се създава продукта, доставя частични работещи версии през определен период от време (най-често на всеки две седмици) и насърчава бързата реакция при настъпване на промени. Терминът е представен от *The Agile Manifesto*[18] през 2001 г.

ГРС предоставя начини за справяне с проблемите при създаване на софтуерни продукти, но оставя и свобода за персонализиране за съответната среда и разработчици.

[18] `http://agilemanifesto.org/`

1.4.1 Процес за реализация на системата

Предлага се следния персонализиран вариант на ГРС за създаване на системата. Необходимо е извършването на няколко повторения(iterations). За всяко от тях:

- Избиране на 2-3 свойства за добавяне към съществуващия продукт от общия списък
- За всяко от тях:
  - Създават се тестове, които да покажат, че част от свойството работи
  - Имплементира се логиката необходима за преминаване на тестовете
  - Кодът се преглежда за начини за промяна с цел улесняване за промяна и по-лесно разбиране
  - Повтаря се, докато свойството не е напълно имплементирано
- Пускат се всички тестове
- Преглед и обсъждане на възможности за подобряване на проекта
- Текущата версия е готова

1.4.2 Повторение I

За първото повторение от имплементацията на системата се избират най-важните свойства от нея:

- Извличане на информация за приложение и запазването ѝ
- Регистрация на потребители
- Създаване и запазване на препоръки

За извличане на информация за отделните приложения съществуват множество уеб страници, две от които са `http://play.google.com/` и `http://www.`

`appannie.com/`. Вторият предоставя допълнителна информация, като например ранкът на приложението в различните държави, което може да се окаже предимство при по-задълбочен анализ на всяко приложение.

Създават се тестове, които проверяват дали се взима правилната информация от AppAnnie, като заглавие, описание, категория и т.н. за всяко приложение. Оптимизация, която намаля значително времето за изпълнение на тестовете, е запазване на съдържанието на страницата, която се тества, върху диска на компютъра. Това има един сериозен недостатък - при промяна на страницата, тестовете ще продължат да преминават, тъй като имат стара версия на съдържанието. Този проблем се решава лесно като съдържанието се обновява често ръчно или автоматично.

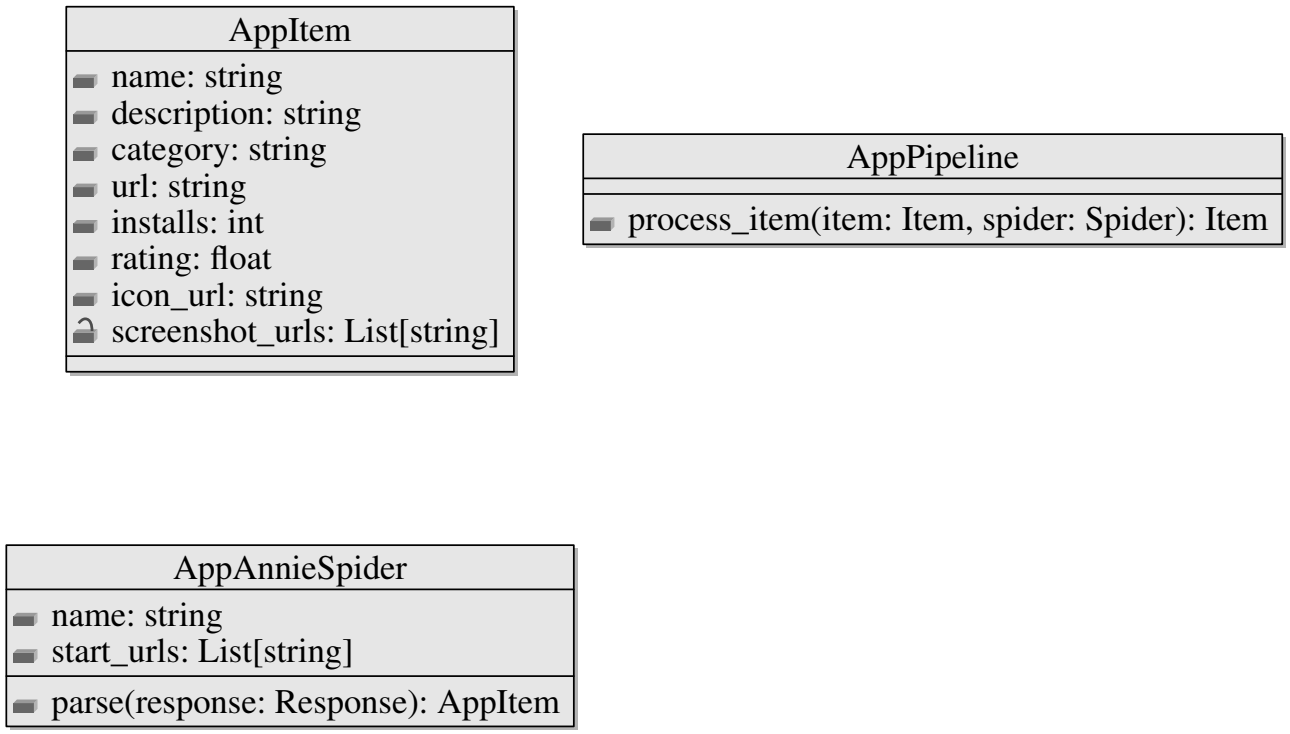
На фигура 1.5 е показана клас диаграма на модула *Spider*. Основният клас в него е *AppAnnieSpider*. Той наследява *BaseSpider* от библиотеката *Scrapy* и предоставя основната функционалност за обработка на данните от уеб страницата. Това става чрез задаване на конкретни *XPath*[19] изрази. В следващи версии на библиотеката ще е възможно и използването на *CSS*[20] изрази.

AppPipeline отново наследява базов клас от *Scrapy* и неговата цел е да предостави възможност за обработка на обектите, които са извлечени, паралелно. Тук се случва и самото запазване на информацията в базата от данни. *AppItem* има за цел да съхрани информация за приложение.

Регистрацията на потребител се състой в събиране и запазване на информация, подадена от клиента. По-ясен поглед върху класа, представляваш потребител, се намира в общия модел на фигура 1.4

Създаването на добри предложения е това, което показва колко успешна е имплементацията. КФ предоставя добър подход за решаването на такъв проблем, но оставя голяма част от детайлите да бъдат разгледани от разработчика. Това прави писането на тестове още по-важно, защото те позволяват налагане на бързи подобрения, ако бъде намерено ново, по-добро, решение.

[19] `http://www.w3.org/TR/xpath/`
[20] `http://www.w3.org/TR/CSS2/selector.html`



Фигура 1.5: Клас диаграма на Spider

Избор на метрика за сходство

След събиране на данни за това, какво харесва един потребител, трябва да се разбере колко сходен е той с другите. Това се прави чрез сравняване на всеки двама човека и изчислението на резултат за сходство. За тази цел съществуват няколко подхода. Два от по-известните са: СП и Евклидово разстояние (Euclidien distance) (EP).

EP е лесен начин за изчисляване на сходство между двама потребители. Формула (1.1) показва процеса за намиране сходността на общи приложения между потребителите p и q.

$$d(p,q) = \sqrt{\sum_{i=1}^{n}(q_i - p_i)^2} \tag{1.1}$$

Така зададена, формулата връща по-високи стойности за оценки, които се различават. Това е неинтуитивно, поради което се заменя връщаща по-високи

стойности за приличащи си оценки. Резултатът е в интервала *[0, 1]* [Segaran]. Формула (1.2) показва нормализирана версия на уравнението.

$$d(p,q) = \frac{1}{1+\sqrt{\sum_{i=1}^{n}(q_i - p_i)^2}} \tag{1.2}$$

Алгоритъм 1 е псевдокод за изчисляване на Евклидово сходство. Резултатът от извикване на тази функция е нормализирана стойност в интервала *[0, 1]*.

Algorithm 1.1: Евклидово сходство между двама потребители

Data: Two users p and q

Result: Euclidean similarity between the users

1 *As* ⟵ `Union(`*p.apps, q.apps*`)`;

2 *res* ⟵ 0;

3 **foreach** *app a of the apps As* **do** res += pow(pR[i] - qR[i], 2);

4 **return** $\frac{1}{1+res}$;

ЕР предоставя лесен и изчислително лек начин за оценка на сходство между два обекта. Методът има един сериозен недостатък - не взима в предвид, когато някои потребители дават високи или ниски оценки за всички приложения(grade inflation[21]).

Описаният проблем се решава от СП. Тази метрика е малко по-сложна, защото използва коефициент на сходство. Той показва колко добре две множества от данни може да се поставят на права линия. Формулата е по-сложна, но дава нормализирани резултати. Тя е показана на уравнение (1.3)

$$r(x,y) = \frac{\sum' xy - \frac{\sum' x \sum' y}{n}}{\sqrt{(\sum' x^2 - \frac{(\sum' x)^2}{n})(\sum' y^2 - \frac{(\sum' y)^2}{n})}} \tag{1.3}$$

Алгоритъм 2 представя псевдокод за изчисление на СП.

[21] `http://en.wikipedia.org/wiki/Grade_inflation`

Algorithm 1.2: Коефициент на Пиърсън за сходство между двама потребители

Data: Two users p and q

Result: Pearson correlation of the users

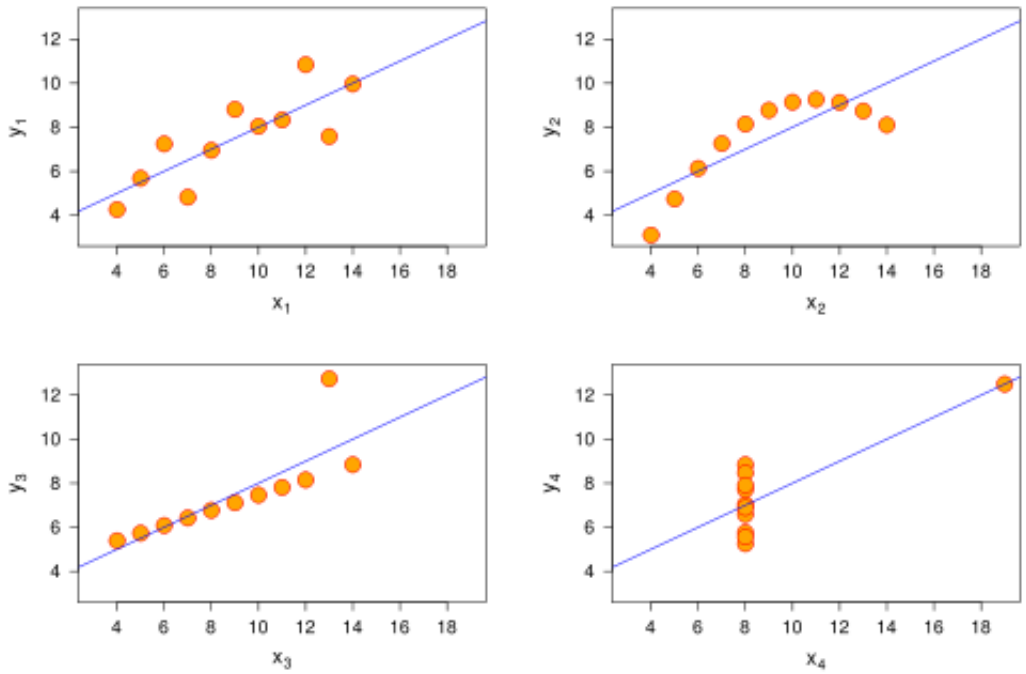
1 $As \longleftarrow$ `Union(`$p.apps,\ q.apps$`)`;
2 $n \longleftarrow$ `Length(`As`)`;
3 $sum1 \longleftarrow$ `Sum(`$As,\ p$`)`;
4 $sum2 \longleftarrow$ `Sum(`$As,\ q$`)`;
5 $sum1Sq \longleftarrow$ `SumSquares(`$As,\ p$`)`;
6 $sum2Sq \longleftarrow$ `SumSquares(`$As,\ q$`)`;
7 $sumProd \longleftarrow$ `SumProducts(`$As,\ p,\ q$`)`;
8 $num \longleftarrow sumProd - (sum1 * sum2/n)$;
9 $den \longleftarrow$ `Sqrt`$((sum1Sq - sum1^2/n)(sum2Sq - sum2^2/n))$;
10 **return** $\frac{num}{den}$;

СП е лоша метрика за свързаност, когато не съществува свързаност [Babenko]. Известен пример за това е показан на фигура 1.6. Примерът горе в ляво дава смислена стойност за свързаността между оценките за приложения. В другите графики стойността е същата, но значимостта ѝ не е голяма. Средната стойност и стандартното отклонение са еднакви.

В система с голям обем данни, вероятността за създаване на подобни проблемни ситуации е много малка. Поради това и сравнително високата ефективност на алгоритъма, той е избран за метрика за сходимост в системата. Имплементацията взима под предвид факта, че може да се наложи неговата замяна. Това прави решението ни лесно за промяна и оставя възможността за набиране на допълнителна информация за проблема.

Създаване на препоръки

Алгоритъм **??** създава препоръки за определен потребител. Той е лесен за имплементация, когато има функция, която отговаря на въпроса колко сходни вкусове имат двама потребители. Резултатът от изпълнението му е асоциативен масив подреден в низходящ ред спрямо предположението приложението да се



Фигура 1.6: Примери за проблем при използване на свързаност на Пиърсън

хареса на потребителя. Функцията *Similarity* може да бъде заменена с друга имплементация за сравнение на сходността на двама потребители. Необходимо е тя да предоставя същия интерфейс.

Запазването на предложения е лесно, благодарение на общия модел на системата. Тази задача се изпълнява от друг модул на системата.

1.4.3 Повторение II

Благодарение на изпълнението на предишното повторение, системата придобива по-ясен вид и доставя най-необходимата функционалност. Във второто повторение се обръща внимание и на клиента. Имплементацията включва:

- Предоставяне на препоръки от сървъра
- Вземане на препоръки от сървъра
- Изпращане информация за потребителя на сървъра

Algorithm 1.3: Създаване на препоръки за потребител

Data: All users Us and person p
Result: List of recommendations for person p

```
T ⟵ Dict;
S ⟵ Dict;
A ⟵ RelativeComplement(p.ratedApps(), u.ratedApps());
for u ∈ Us do
    sim ⟵ Similarity(u, p) ;
    for a ∈ A do
        T[a]+ = u.ratingFor(a) * sim;
        S[a]+ = sim;
    end
end
for t, a ∈ T do
    R[t/S[app]] = app ;
end
Sort(R);
Reverse(R);
return R;
```

- Приемане на информация за потребителя

Тези свойства се групират две по две и по този начин се разработват паралелно. Това е незадължително при разработка, използваща създаване на тестове преди самата имплементация. Те имитират ролята на сървър, както и тази на клиент, при изпълнение.

За даване на препоръки, сървърът предоставя ПИП, като използва *REST* протокол и кодира данните в *JSON* формат. Тази архитектура се изгражда на отворени и безплатни стандарти, които модерните уеб технологии предоставят. Липсата на единен стандарт, който задължително трябва да се спазва, за да може едно ПИП да бъде наречено *REST-пълно*, води до неясен подход за създаване на добра имплементация.

Някои чести проблеми при създаване на подобна архитектура са [Vitvar]

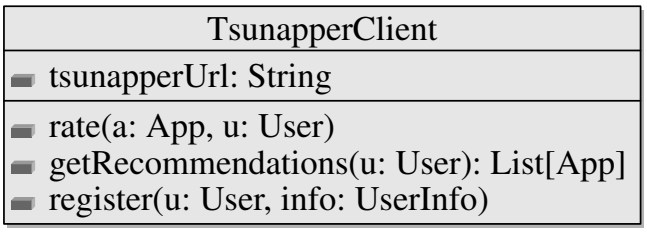
- Използване на *GET* за всички видове заявки
- Неспазване на кодовете за отговор

- Неспазване правилата за кеширане

Bottle предоставя лесен и скалируем начин за справяне с тези и други проблеми, като някои от решенията се взимат автоматично от библиотеката. Сървърът предоставя следните адреси за достъп до отделните ресурси

- */recommend/* : *userId [GET]* предоставя препоръки за потребител с ключ, подаден като параметър
- */users [POST]* добавяне на информация за потребител
- */apps/rate [POST]* добавяне на рейтинг за приложение от потребител

Клиентът използва класа *TsunapperClient*. Той е показан на диаграма 1.7. Методите отговарят на по-горе посочените. Моделът е този представен на фигура 1.4.



Фигура 1.7: Клас диаграма на TsunapperClient

Скоростта за имплементация на това повторение е около два пъти по-висока от предишното. Въпреки това, функционалността е съществена за изпълнение на системата.

Глава 2

Ръководство за потребителя

Заключение

Резултати

Приноси

Проблеми по време на разработка

Бъдещо развитие

Списък на фигурите

1.1 UML диаграма на сървърния компонент 9
1.2 UML диаграма на клиента 10
1.3 Activity диаграма на системата 11
1.4 Клас диаграма на модела на системата 13
1.5 Клас диаграма на Spider 17
1.6 Примери за проблем при използване на свързаност на Пиърсън . 20
1.7 Клас диаграма на TsunapperClient 22

LIST OF ALGORITHMS

1.1 Евклидово сходство между двама потребители 18
1.2 Коефициент на Пиърсън за сходство между двама потребители . . 19
1.3 Създаване на препоръки за потребител 21

Приложение А

Използвани съкращения

ОС Операционна система (Operating system)

СП Системи за препоръки (Recommendation systems)

МО Машинно обучение (Machine learning)

КН Компютърни науки (Computer science)

КФ Кооперативно филтриране (Collaborative filtering)

ФБС Филтриране, базирано на съдържание (Content-based filtering)

КБС к-близки съседи (k-nearest neighbours)

СП Свързаност на Пиърсън (Pearson Correlation)

ЕР Евклидово разстояние (Euclidien distance)

ПИП Приложим интерфейс за програмиране (Application Programming Interface)

ЦП Централен процесор (Central Processing Unit)

ГРС Гъвкава разработка на софтуер (Agile software development)

Приложение Б

Библиография

[Babenko] Haralambos Marmanis and Dmitry Babenko. *Algorithms of the Intelligent Web*. 2009.

[Elgin] Ben Elgin. Google buys android for its mobile arsenal. 2005.

[Google] Google. 900 million android activations to date, 48 billion app installs. 2013.

[Gupta] Tushar Gupta. Multi-threading android apps for multi-core processors. 2013.

[IDC] IDC. Android marks fourth anniversary since launch with 75.0in third quarter, according to idc. 2012.

[Marketwire] Marketwire. Mongodb extends leadership in nosql. 2011.

[Melville] Prem Melville and Vikas Sindhwani. Recommender systems, encyclopedia of machine learning. 2010.

[Mooney] Raymond J. Mooney and Loriene Roy. Content-based book recommending using learning for text categorization. 2000.

[Segaran] Toby Segaran. *Programming Collective Intelligence: Building Smart Web 2.0 Applications*. 2008.

[Sirer] Emin Gün Sirer. Broken by design: Mongodb fault tolerance. 2013.

[Vitvar] Tomas Vitvar. Api anti-patterns: How to avoid common rest mistakes. 2010.