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 системи“

2013 г.

Съдържание

Увод

Играете ли *Angry Birds*[1]? Повече от 260 милиона[2] потребители го правят редовно! Може би игрите не ви допадат? Може спокойно да изберете някое от другите 900,000+ приложения за мобилни устройства. Кои от тях са подходящи?

Апликации и игри за мобилни устройства варират спрямо категории, целеви групи и др. Над един милиард[3] от земното население има ежедневен достъп до тях. Децата се забавляват с увлекателни игри, докато по-възрастните използват телефоните си за банкиране, запазване на самолетни билети, планиране на екскурзии и др.

От другата страна на този феномен са разработчиците на мобилни приложения. Те се опитват да слеят границите между истинския свят и виртуалната реалност. Явен пример за това е *Ingress*[4]. Способността да имаме различна, виртуална идентичност, кара милиони потребители да се състезават, помагат, следят и обсъждат с непознати и приятели.

Голямото разнообразие на мобилни приложения поставя сериозен въпрос - Как да намеря най-доброто приложение за мен? В ерата на *Google* търсенето, този въпрос може да изглежда безмислен, но това съвсем не е така. Текстът на едно приложение е само неговото заглавие и описание. Той може да е неточен и

[1] `http://www.angrybirds.com/`

[2] `http://mashable.com/2013/01/11/rovio-angry-birds-260-million/`

[3] `http://cuttinglet.com/world-smartphone-population-crosses-1-billion/`

[4] `http://www.ingress.com/`

непълен. Още повече, функционалността и интерфейса може да са неправилно създадени.

Системи за препоръки (Recommendation systems) (СП) предоставят разрешение на този проблем. Те създават предложения, които използват данни от други потребители и техните оценки. Фактори като общ рейтинг, брой сваляния, категория и класиране, също се взимат под предвид.

Цел

Целта на дипломната работа е да изгради система за препоръки на мобилни приложения за *Android*.

Изисквания

- Лесна за употреба.
- Предоставя добри препоръки.
- Ниска цена за поддръжка.
- Лесна за разширение и подобрение.

Постигане на целта

За постигане на целите на дипломната работа, спрямо поставените изисквания се разглеждат:

- Модерни технологични решения в подобни ситуации.
- Проучване на съществуващи системи.
- Създаване на система за препоръки на мобилни приложения.
- Получените резултати се анализират, за да се получи частично или пълно решение на проблема.

Целеви групи

Потребители, които желаят по-добри приложения, максимално приближаващи се до техните вкусове и изисквания. Важно за тях е бързото намиране и инсталиране на голям брой апликации, които да ползват за по-дълго време.

Желан резултат

Дипломната работа може да се сметне за успешна, когато бъде създаден прототип на система, който да позволява предоставянето на препоръки за мобилни приложения. Системата трябва да бъде лесна за промяна и разширяване, така че да предоставя начини за добавяне на нови източници на информация и създаване на клиенти.

Задачи

За постигане на желания резултат е необходимо следните задачи да бъдат изпълнени:

- Проучване на видове СП
- Съществуващи примери
- Избор на подходящи технологии
- Моделиране на отделните компоненти
- Създаване на тестове
- Създаване на системата
- Провеждане на тестове върху системата
- Привеждане в употреба
- Препоръки за подобрения

Структура

Настоящата дипломна работа се състои от увод, създаване на система за препоръки, ръководство за програмиста и заключение.

В уводът се описват проблема и целта на дипломната работа. За постигане на целта са посочени задачи, които трябва да бъдат изпълнени.

Основната задача е създаване на система за препоръки на мобилни приложения. Тя съдържа четири части:

- *Проучване* - кратко разглеждане на съществуващи решения и проблеми, които решават
- *Моделиране* - запознаване с процесът на моделиране на системата
- *Използвани технологии* - преглед на използваните технологии за изграждане на решението
- *Реализация* - описва процеса за създаване на системата

Използване на системата съдържа описание за инсталиране и стартиране на двата основни компонента. Това е допълнено с екрани от имплементацията.

В заключението се предоставят резултати от системата, решаваща поставените проблеми. Описват се приносите на автора и проблемите, срещнати по време на разработка на програмата. Дават се препоръки за възможни подобрения и разширения.

Предоставят списъци от библиография, използвани алгоритми, фигури и използвани съкращения.

Глава 1

Създаване на системата

1.1 Проучване

Събиране на препоръки от доверени източници е важен компонент от взимането на решение. С навлизане на уеб технологиите, пред купувачите на стоки се представя нарастващ избор. Продавачите са принудени да персонализират предложенията си, за да останат конкурентноспособни. В същото време, големите компании събират огромен брой информация за потребителите си. Тя се анализира и предоставя поглед върху предпочитанията на потребителите. СП удовлетворяват нуждите на купувачи и продавачи, като автоматизират създаването на препоръки, базирани на анализ на данни.

Най-общ поглед върху СП представя матрица от n потребителя и m обекта и всяка клетка $r_{u,i}$ съпоставя рейтинг за обект i от потребител u. Матрицата е почти празна, защото повечето потребители поставят рейтинг на малък брой обекти. Задачата е да се предвиди оценката на обект, която потребителят би дал. Препоръчват се тези продукти, които имат най-висока предвидена оценка. Потребителят, за който се създават препоръки, се нарича *активен потребител*.

Със създаване на 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]

- 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

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

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 К.

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.

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

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 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

[5] http://www.youtube.com/

[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

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*.

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:

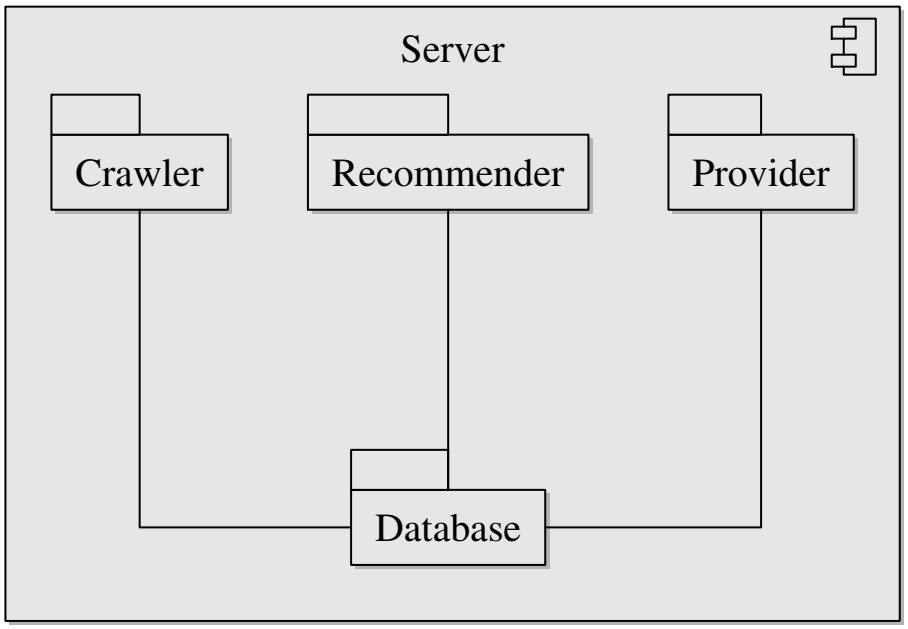
  - Database е компонент, който предоставя услуга за съхранение на данни. Той играе ролята на косвен (implicit) интерфейс, с който да работят останалите компоненти от сървърната част.
  - Spider се грижи за събиране на информация за различните приложения от уеб страници. Данните се запазват за по-нататъшен 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 пре-

[10] `www.rssphone.com/google-play-store-800000-apps-and-overtake-apple-appstore/`
[11] `http://www.facebook.com`

дишния 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.



Фигура 1.1: UML диаграма на сървърния компонент

- *Клиент* предоставя, може би, най-важната част от системата - това, с което потребителят 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 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.

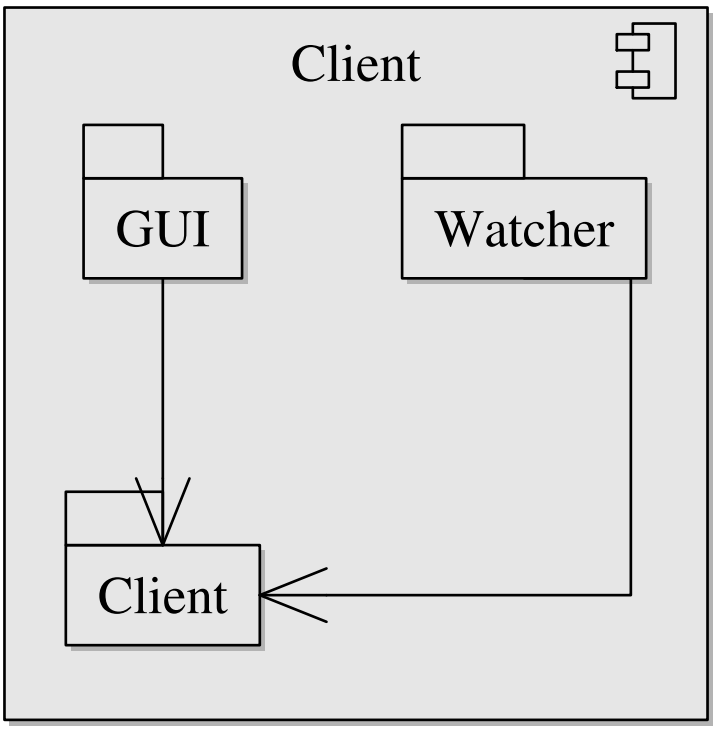
– Client комуникира със сървърната част, като предоставя информация за потребителя и извлича препоръки. Важно за модула е да извършва функциите си по напълно прозрачен начин за потребителя.

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 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 1.3 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.

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



Фигура 1.2: UML диаграма на клиента

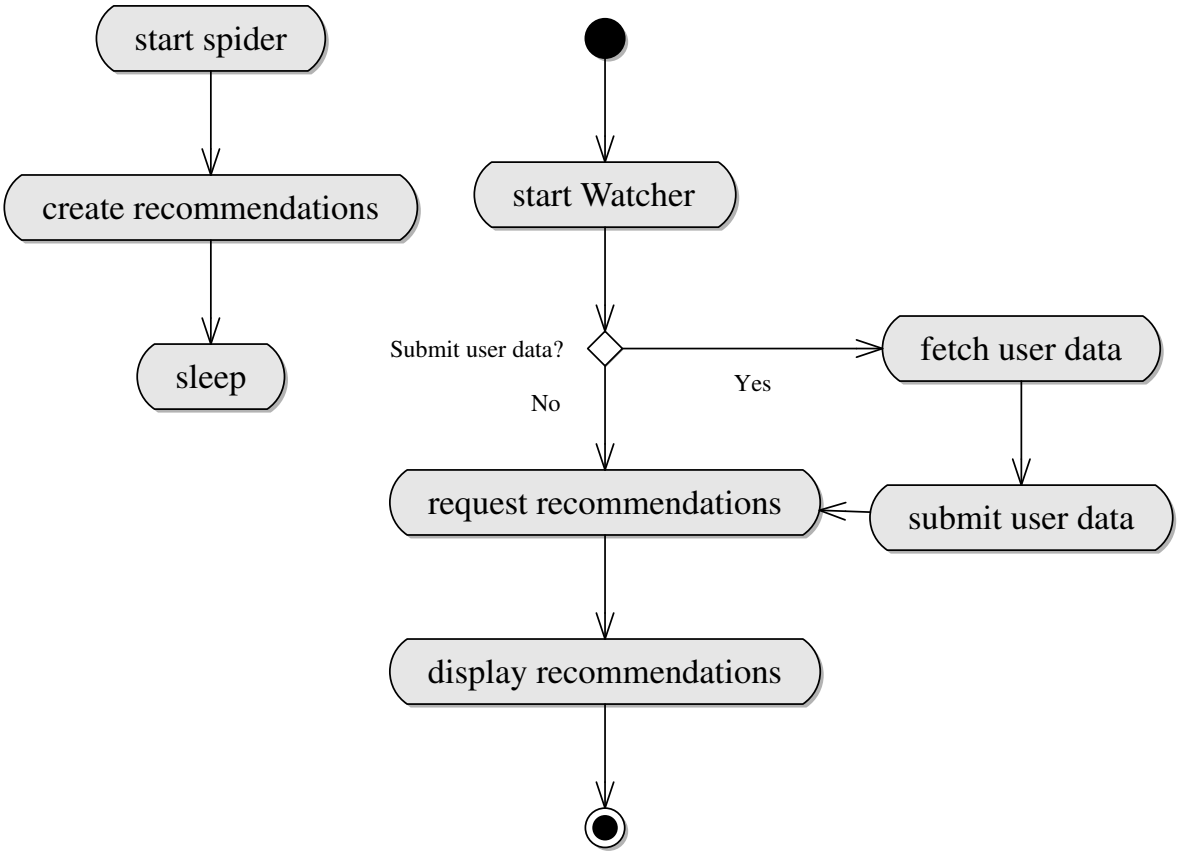
1.2.1 Основен модел на системата

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.

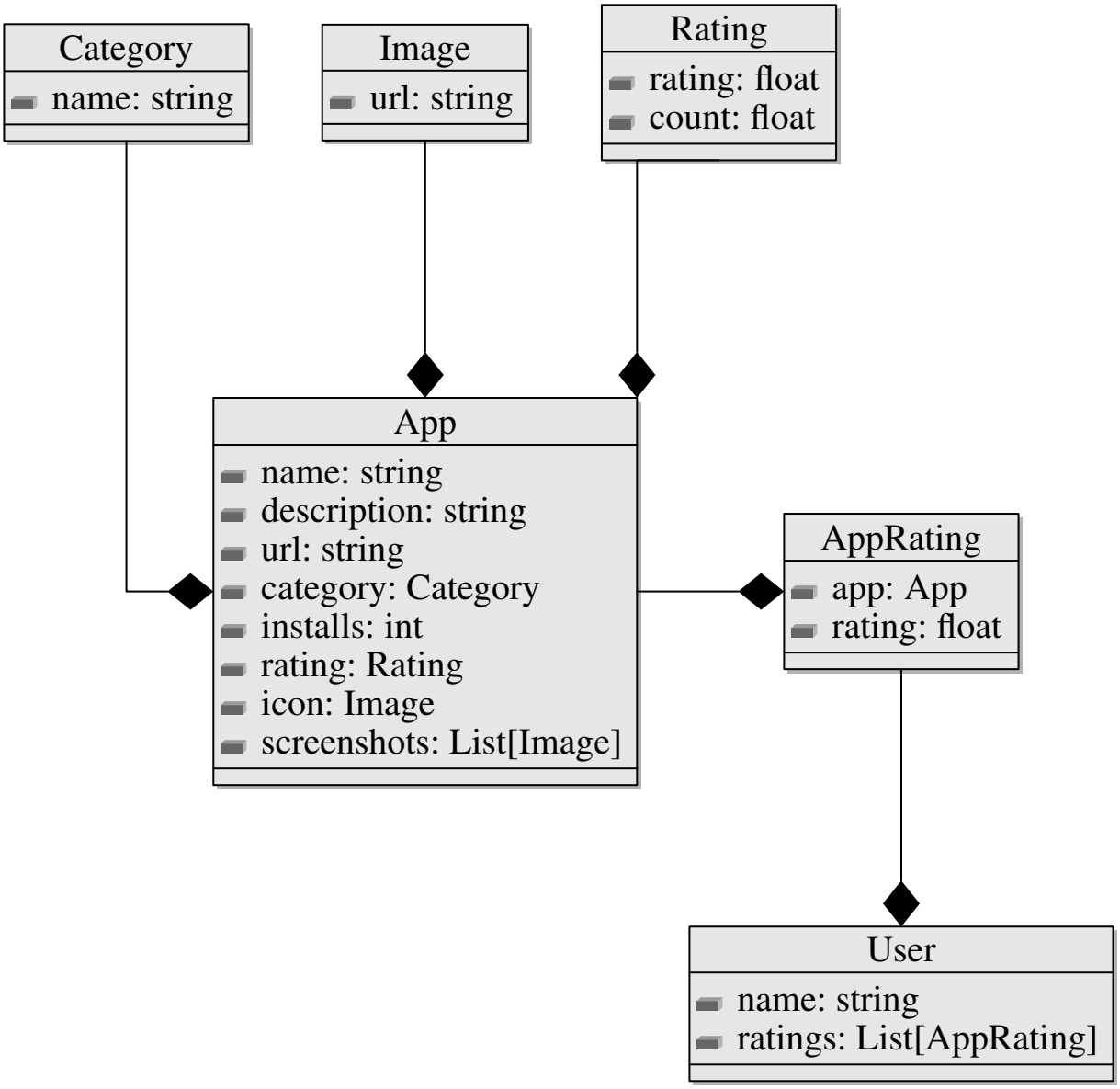
1.3 Използвани технологии

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



Фигура 1.3: Activity диаграма на системата



Фигура 1.4: Клас диаграма на модела на системата

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]

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

Python[18] е език за програмиране от високо ниво, който се стреми да предостави четим код. Предоставя начин за писане на по-малко редове, спрямо *C*. Той поддържа множество парадигми, включително обектно-ориентиран, императивен и функционален стил. Код написан на *Python* може да бъде изпълняван върху множество платформи.

Езикът предоставя разнообразни допълнителни модули за работа с графични среди, математически операции, мрежи, уеб сървъри и други. Това богатство го прави много подходящ за почти всякакъв вид приложения. *Python* е

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

широко използван в научни и бизнес среди[19].

Scrapy

Scrapy[20] е библиотека с отворен код от високо ниво за събиране на информация от уеб пространството. Използва се за извличане на структурирани данни и автоматично тестване.

Библиотеката е бърза и лесна за работа, като предоставя и добра документирана. Предоставя паралелна обработка на данните, както и самите заявки към страниците. Основният метод за извличане на информация е чрез *XPath* изрази.

mongoengine

Тази библиотека свързва *MongoDB* и сървърната част на приложението. *Mongoengine*[21] предоставя обектно-ориентиран подход за операриране с База от данни (Database) (БД). Добре документирана е и предлага лесен начин за използване. *Mongoengine* надгражда функционалност върху *PyMongo*, който е драйвър за *MongoDB*.

bottle

Bottle[22] е бърза, микро библиотека за изграждане на малки уеб приложения. Тя предоставя лесно извличане на параметри от заявка, интеграция с различни двигатели за изгледи, вграден сървър и възможност за използване на по-големи такива. *Bottle* не зависи от нищо друго, освен стандартната библиотека на *Python*.

Библиотеката е особено подходяща за създаване на *REST*-пълни ПИП, по лесен и бърз начин. Приложение, изградено с *bottle*, се тества лесно чрез вградени механизми.

[19] `http://python.org/about/quotes/`
[20] `http://scrapy.org/`
[21] `Mongoengine`
[22] `http://bottlepy.org/docs/dev/`

1.3.2 Клиент

От страна на клиентската част, библиотеките, които се ползват, са пряко зависими от платформите, на които приложението трябва да се използва. За разработката на добро *Android* приложение, езикът, който предоставя най-големи възможности и библиотеки е *Java*. Това, разбира се, налага някои ограничения.

Android SDK

Софтуерни инструменти за разработка (Software Development Kit) (СИР) на *Android* предоставят голяма част от стандартната версия на *Java* СИР. Съществуват и много допълнения, като някои от тях предоставят възможности за:

- работа с графичната среда на *Android*
- услуги
- бази от данни
- карти
- GPS
- жироскоп
- връзка с интернет

Предоставя се и лесна интеграция с *Eclipse*[23], *IntelliJ*[24] и други среди за разработка. За процесът на пакетиране на приложението се използват *Maven*[25], *Gradle*[26], *Ant*[27]и др.

Допълнителни функционалности се интегрират постоянно в по-новите версии. Извършва се тяхното частично добавяне в по-стари версии, чрез т.нар.

[23] `http://www.eclipse.org`
[24] `http://www.jetbrains.com/idea/`
[25] `http://maven.apache.org/`
[26] `http://www.gradle.org/`
[27] `http://ant.apache.org/`

Android support library, която се поддържа от официалните разработчици на *Android*.

Retrofit

Retrofit[28] предоставя стриктно типизиран подход към *REST* ПИП за *Android*. С използване на библиотеката, част от работата на програмиста се спестява като се използва метапрограмиране и по-точно анотации в *Java*.

Библиотеката позволява изпращането на всички видове възможни заявки и добавяне на всякаква информация към тях. Поддържат се различни типове документи, като обработката на *JSON* е лесна и ефективна, при получаване и изпращане. Възможно е прикачването на файлове и изпращане на форми.

RoboSpice

Retrofit е много добра библиотека за извършване на заявки, но не предоставя начин за кеширане на резултати, определяне време за изпълнение на заявки, използване на много ядра, изпълнение в многонишкова среда и др.

Всички свойства се предоставя от *RoboSpice*[29]. Библиотеката е специфично създадена за работа при мобилни устройства и предоставя защита от блокиране, показване на грешки и загуба на памет.

Благодарение на модуларната си структура, *RoboSpice* се интегрира лесно за работа с *Retrofit* и я използва за извършване на самите заявки.

Библиотеката предоставя механизъм за извличане на информация от наличната на устройството и спестява извършване на тежки мрежови операции. За това се използват различни начини за запазване на информация като файлове, БД и др.

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

[28] `http://square.github.io/retrofit/`
[29] `https://github.com/octo-online/robospice`

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 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*[30] 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.

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

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

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

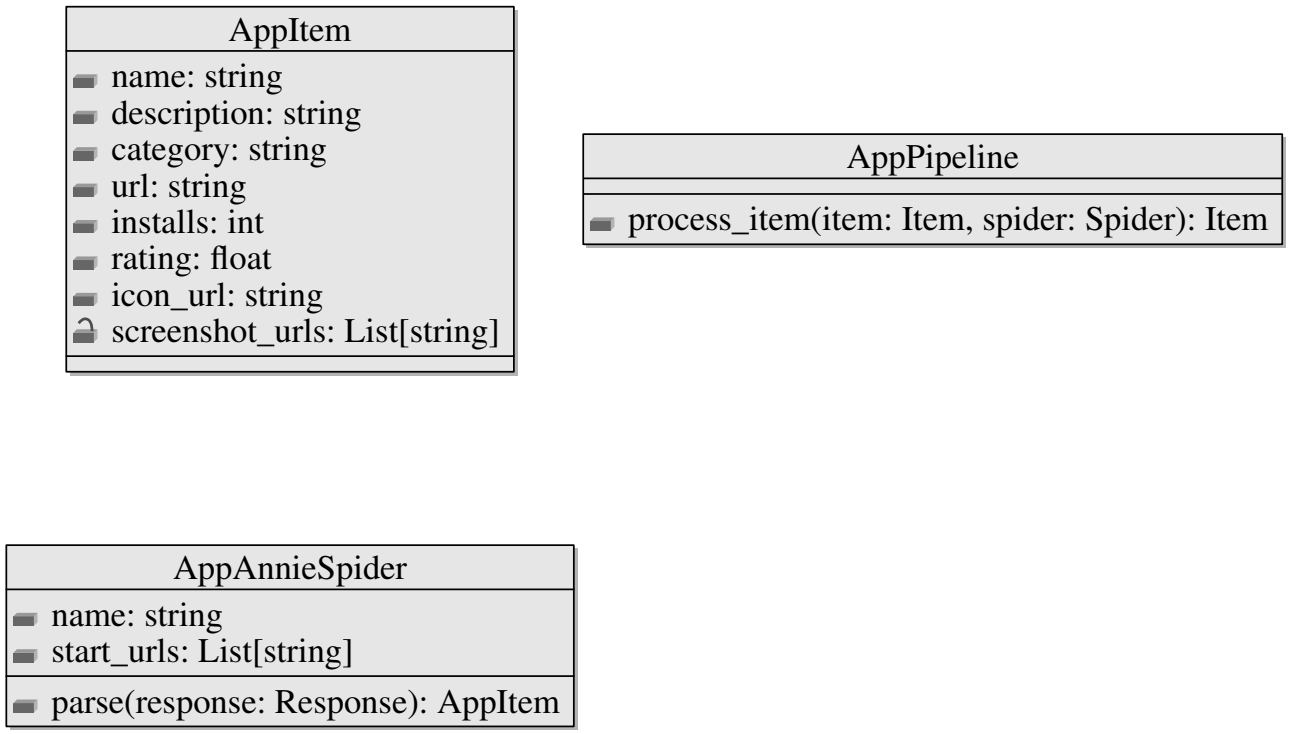
За извличане на информация за отделните приложения съществуват множество уеб страници, две от които са *Google Play*[31] и *AppAnnie*[32]. Вторият предоставя допълнителна информация, като например ранкът на приложението в различните държави, което може да се окаже предимство при по-задълбочен анализ за създаване на препоръки.

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*[33] изрази. В следващи версии на библиотеката ще е възможно и използването на *CSS*[34] 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.

[31] `http://play.google.com/`
[32] `http://www.appannie.com/`
[33] `http://www.w3.org/TR/xpath/`
[34] `http://www.w3.org/TR/CSS2/selector.html`



Фигура 1.5: Клас диаграма на Spider

Регистрацията на потребител се състой в събиране и запазване на информация, подадена от клиента. По-ясен поглед върху класа, представляващ 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.

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) (ЕР).

ЕР 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 с 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 1.1 е псевдокод за изчисляване на Евклидово сходство. Резултатът от извикване на тази функция е нормализирана стойност в интервала *[0, 1]*.

Алгоритъм 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 потребители, продължително даващи високи или ниски оценки за всички приложения (grade inflation[35]).

Описаният проблем се решава от СП. Тази метрика е малко по-сложна,

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

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 1.2 представя псевдокод за изчисление на СП.

Алгоритъм 1.2: Коефициент на Пиърсън за сходство между двама потребители

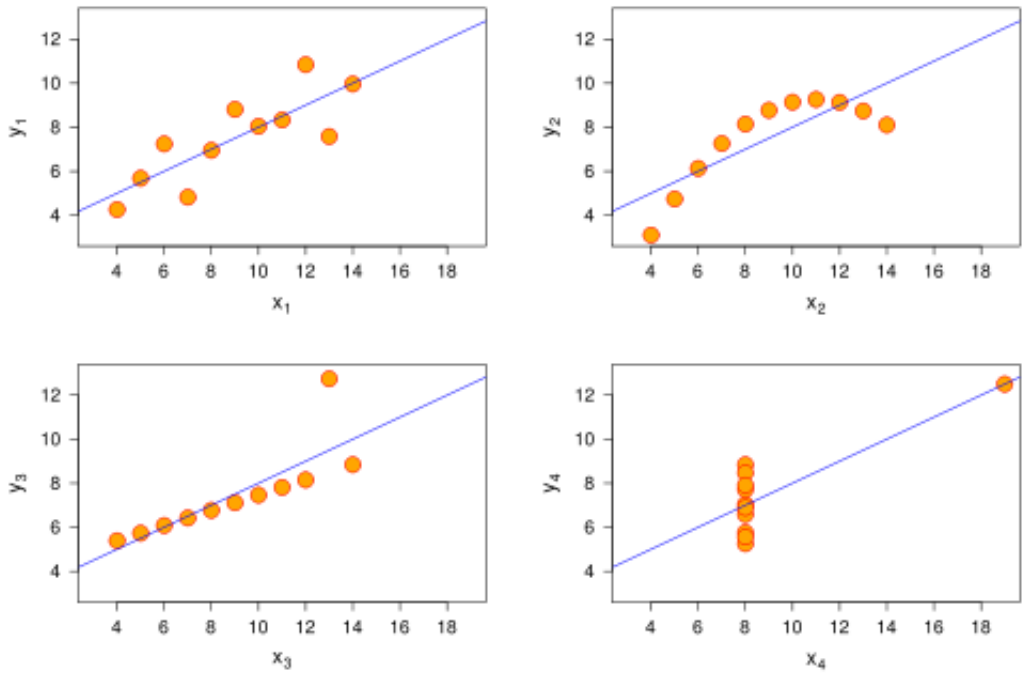
Data: Two users p and q

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.



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

Това прави решението ни лесно за промяна и оставя възможността за набиране на допълнителна информация за проблема.

Създаване на препоръки

Алгоритъм 1.3 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 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.

Алгоритъм 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;
```

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
- Приемане на информация за потребителя

Тези свойства се групират две по две и по този начин се разработват паралелно. Това не е задължително 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.

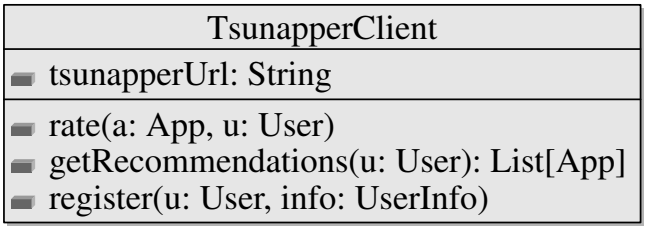
Някои чести проблеми при създаване на подобна архитектура са: [Vitvar]

- 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.

1.4.4 Повторение III

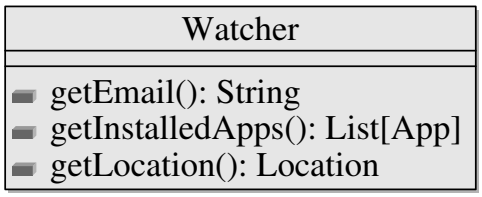
В последното повторение се реализират следните възможности на системата:

- Събиране на информация за потребителя
- Създаване на Графичен потребителски интерфейс (Graphical User Interface) (ГПИ)

Събирането на информация се разделя на няколко подзадачи:

- Извличане на електронната поща на потребителя
- Извличане на инсталираните приложения
- Извличане географското място на устройството

За всяка подзадача се създават тестове и имплементация, която ги удовлетворява. Реализацията се намира в модула *Watcher*. Събраните данни се изпращат на сървъра за запазване и използване. На диаграма 1.8 е показан *Watcher*.



Фигура 1.8: Клас диаграма на Watcher

Информацията получена от модула спомага за решаване на проблема "студен старт". Списъкът от инсталирани приложения се използва за имитиране на

рейтинг за всяко от тях, даден от потребителя. Използвайки тази информация се изчисляват препоръки, въпреки липсата на потребителска активност. При липса на достатъчен брой инсталирани приложения се предоставят препоръки за високо оценени апликации и игри в съответния мобилен пазар.

Графичен интерфейс

Чрез разпознаване и именуване на добри практики за създаване на ГПИ, ние ги виждаме по-добре. Виждаме повече детайли, защото мозъците ни са настроени да търсим за тях. [Tidwell] Използването на тези практики прави задачата за разработване по-лесна, но и по-ясна програма за потребителя.

С навлизането на умните мобилните устройства, взаймовръзката с технологиите се промени от тази на дигитален асистент до допълнителен човешки орган с невероятни сензори. [Nudelman] Това прави изграждането на ГПИ още по-важна, но и сложна задача.

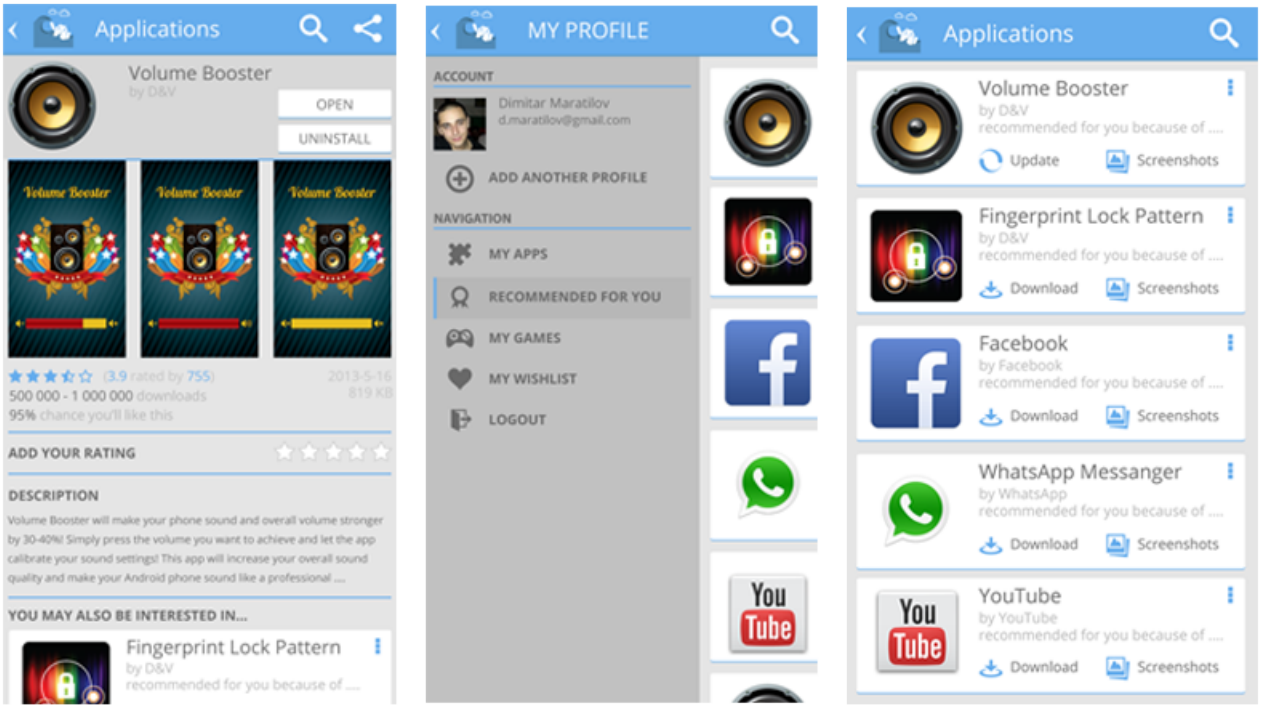
ГПИ на системата се изгражда на базата на добри практики за дизайн. Визуалната част се създава от професионален дизайнер, който ги спазва и комбинира за създаване на приятно изживяване на потребителя.

За реализация на ГПИ съществуват помощни библиотеки за създаване на автоматизирани тестове. Изпробване на вида и усещането от приложението се извършва от реални потребители. Реализацията на това повторение доставя цялостната функционалност, предвидена за системата.

За създаване на цялостно добро изживяване на таблети и телефони се създават различни размери и съотношения на графичните ресурси. Използването им е подпомогнато от библиотеките на *Android*.

Проверка за коректно създадено ГПИ се извършва върху емулатор с различни резолюции, телефон и таблет. Важно е и тестването върху различни версии на *Android*. Голяма част от компонентите са приложими в по-новите версии. Библиотека за поддръжка предоставя част от възможностите и за по-стари устройства.

Фигура 1.9 показва ГПИ на системата в действие.



Фигура 1.9: Графичен интерфейс на мобилното приложение

Глава 2

Използване на системата

Системата е разделена на два основни компонента и всеки от тях може да бъде стартиран на отделно устройство или на едно - общо такова.

Сървър

Сървърната част е проектирана за разполагане върху мощен компютър. За тестови цели, тя може да се преведе в изпълнение и на по-слаби машини.

Процес на разполагане

Системата е предвидена за мултиплатформена употреба, но за целите на дипломната работа е представен метод за разполагане върху *Ubuntu Server*[1]. Предполага се версия *12.04* или по-нова.

За инсталиране на MongoDB се изпълнява следната команда:

```
> sudo apt-get install mongodb-server
```

Ubuntu предоставя инсталация на *Python* по подразбиране. За добавяне на допълнителни модули се използва *pip*[2]. За инсталацията му се изпълнява следната команда в терминала:

```
> sudo apt-get install python-pip
```

[1] `http://www.ubuntu.com/server`
[2] `https://pypi.python.org/pypi/pip/`

Допълнителите модули се добавят с помощта на *pip* чрез следните команди:

```
> sudo pip install behave
> sudo pip install mongoengine
> sudo pip install bottle
```

Изпълнение на тестовете предоставя начин за проверка за това дали всичко необходимо е инсталирано правилно. В терминала, текущата директория се променя на *features*, която е поддиректория на главната за проекта. Изпълнява се:

```
> behave
```

Примерен изход от тази команда е предоставен на фигура 2.1.

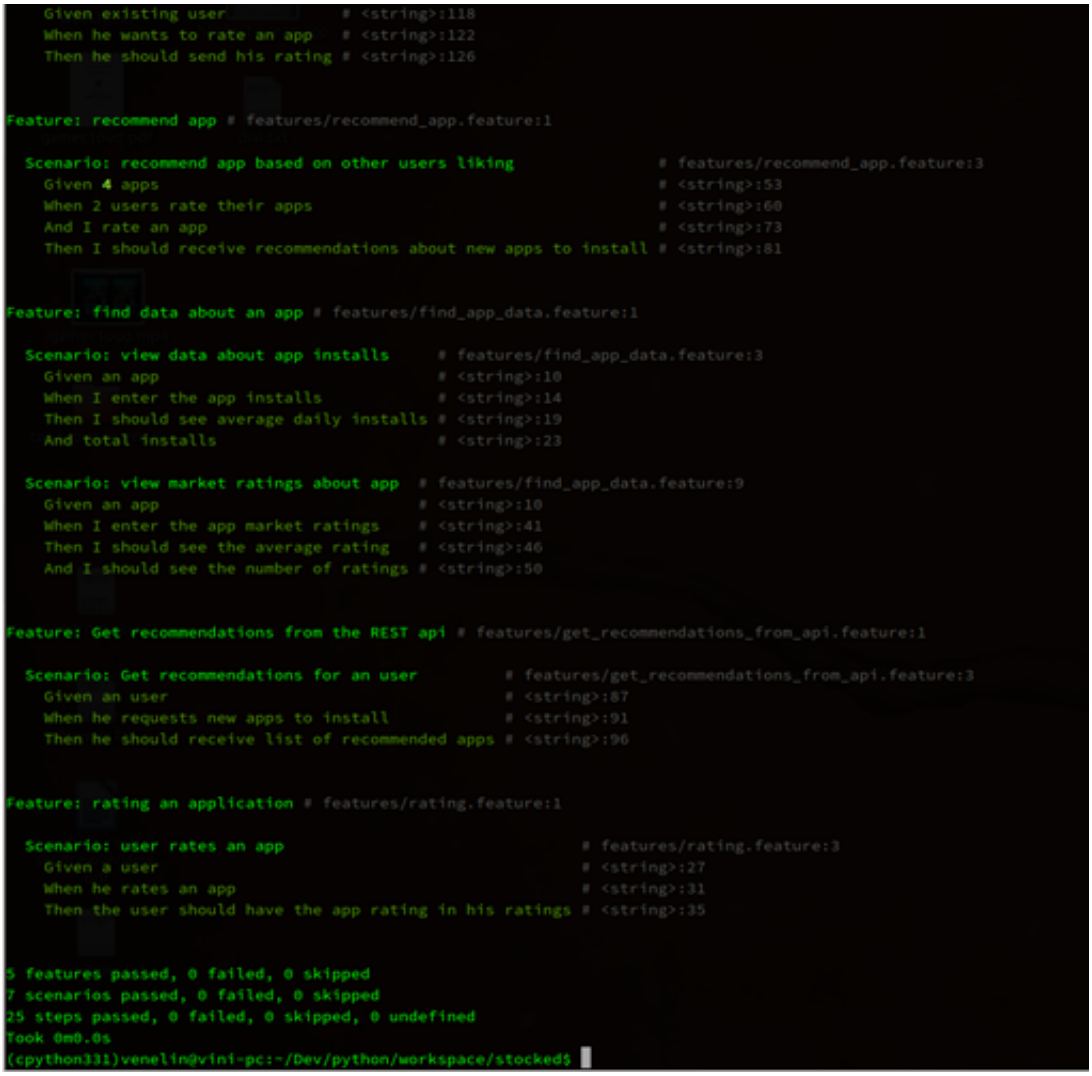
Стартиране на уеб приложението става чрез следната команда (изпълнена в главната директория на проекта):

```
> python webapp.py
```

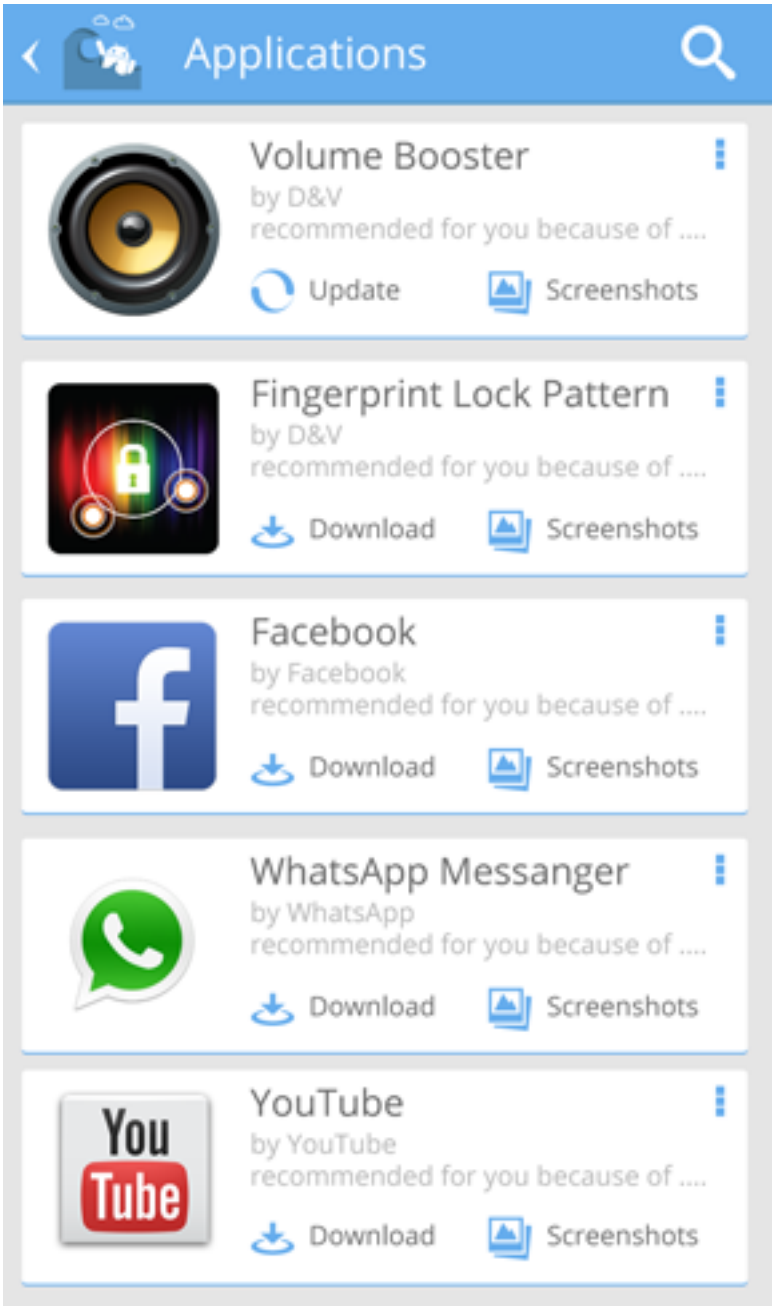
Клиент

Клиентската част използва *Android SDK*. Подробни инструкции за инсталацията му са дадени на официалния сайт[3]. То се използва за пакетиране на мобилното приложение. Апликацията се стартира по стандартния начин за всички подобни приложения и работи върху емулатор и реални устройства. За правилната му работа е необходима свързаност между сървъра и клиента. Стартирано, приложението е показано на фигура 2.2.

[3] `http://developer.android.com/sdk/`



Фигура 2.1: Примерен изход от провеждане на тест на сървърната част



Фигура 2.2: Примерен поглед върху клиентското приложение

Заключение

Резултати

За постигане целите на дипломната работа е създадена СП на мобилни приложения. Разработено е и клиентско приложение за *Android*, което комуникира с нея. Двата основни модула са ясно разграничени, като всеки от тях е създаден от допълнителни подмодули.

Методът за създаване на проекта е ГРС. Той намаля сложността при разработката и улеснява добавянето на нова функционалност. Системата е написана в стил - Обектно-ориентирано програмиране (Object-oriented programming) (ООП). Това допълнителни улеснява добавянето на нова функционалност и модули.

Създаденият проект предоставя единен и приятен ГПИ за таблети и телефони. Модулът за създаване на препоръките е тестван и предоставя добри предложения за различни категории мобилни приложения. Той става все подобър, с нарастването на потребителите, използващи системата.

Системата се държи стабилно по време на тестове, но това остава да бъде потвърдено или отхвърлено в работна среда.

Приноси

- Проучване на различни СП и техни имплементации.
- Проучване на различни технологии, необходими за създаване на система-

та.

- Създаване на система за препоръки на мобилни приложения.
- Създаване на клиентско приложение за системата, разработено за *Android*.
- Провеждане на тестове, върху имплементация на системата.

Проблеми и ограничения

Основните проблеми по време на разработка са обвързани с липсата на задълбочени познания в областта. Предишен опит за създаване на СП напълно липсва, както и добри познания за създаване и разгръщане на големи системи в реална обстановка. Имплементация на системата е на език, който е почти непознат. Това означава, че тя не е е написана по много добър начин, подходящ за използвания език. Авторът не е запознат с много от основните идеи и концепции за създаване на подобни системи, но тази работа може да се разглежда като подход за научаването им.

Системата е разработена под *Ubuntu OC*, която предоставя мощни инструменти за създаване на подобни проекти. Мениджърът на пакети (*aptitude*), понякога предоставя стари версии, които не осигуряват пълната функционалност на по-новите такива.

Python е един такъв пример. Стандартната версия, която се предоставя е 2.7.x. Тя е добре поддържана, но липсват много от възможностите на 3.3.x.

Проблемът с *Python* се състой в това, че много модули не поддържат последната версия на езика. Пример за такъв е *Scrapy*. Налага се използването на виртуални среди и използването на различни версии за езика, за различни части от системата. Още повече, синтаксисът на езика в някои от версиите се различава.

Официално поддържаните версии на повечето хостинг доставчици все още са в диапазона 2.5.x - 2.7.x. Разработчиците на библиотеки, искат да използват кода си в продукция и насочват усилията си към поддръжка на тези версии. Така, новите възможности на езика не се предоставят на разработчиците, ползващи тези библиотеки.

За създаване на сървърния модул се използват множество малки библиотеки, които предоставят малко документация и липсва стандартен подход за комбинирането им с други такива. *Behave* предоставя инфраструктурата за създаване на тестове, но не предоставя вградена библиотека на самото писане на тестове, което прави първоначалното настройване трудно.

При клиентската част се наблюдават няколко проблема, един от които е фрагментацията. *Android* предоставя възможност за инсталация върху множество различни устройства, вариращи от телефони до телевизори, хладилници и други. Много от тях нямат очакваната функционалност, поради различния хардуер, който използват. Версиите на ОС варират[4], като по-старата *Gingerbread*, все още доминира над по-новите такива.

Това поставя поне два проблема пред разработчиците на системата:

- Кои устройства ще се поддържат?
- Кои версии ще се поддържат?

Основен проблем за различните устройства са и различните гъстоти на пиксели, резолюции и големина на екраните. Изграждат се поне два различни изгледа за таблети и за телефони. Системата поддържа устройства с версия на *Android*, поне Gingerbread, което налага неизползването на някои по-нови функционалности или само частично такива.

Необходимо е избиране на библиотеки за клиентската част, които са ефективни и бързи, за да направят усещането за приложението приятно и удобно за потребителя. Въпреки големия брой вградени библиотеки и сравнително утвърдилата се платформа, все още липсват модули за лесна комуникация с *REST* услуги.

Бъдещо развитие

Системата е изградена по начин, който предразполага към лесното разширение и дори по-фундаментални промени във функционалността. Трябва да се вземе под предвид, че тя не е доказала своята ефективност и функционалност в работна среда, поради което редица модули може да бъдат променени.

[4] `http://www.androidbg.com/android-statistics-for-may-2013`

Ясно изразената граница за разделение на сървърна и клиентска част е предпоставка за развитие само на един от двата модула, без другия да бъде засегнат. Следните допълнения биха подобрили ефективността и удовлетворението на потребителите при използване на системата:

- Интеграция с *Facebook SDK*[5]
- Създаване на клиент за *iOS*[6]
- Извличане и обработка на допълнителна информация за потребителите
- Извличане на допълнителна информация за приложенията от други източници
- Паралелно изчисление на препоръките
- Автоматично стартиране изчисление на препоръките чрез МО

[5] `http://www.facebook.com`
[6] `http://www.apple.com/ios/`

Списък на фигурите

Списък на алгоритмите

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)

БД База от данни (Database)

СИР Софтуерни инструменти за разработка (Software Development Kit)

ГПИ Графичен потребителски интерфейс (Graphical User Interface)

ООП Обектно-ориентирано програмиране (Object-oriented programming)

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.

[Nudelman] Greg Nudelman. *Android Design Patterns: Interaction Design Solutions for Developers*. 2013.

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

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

[Tidwell] Theresa Neil. *Mobile Design Pattern Gallery*. 2012.

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