UNIVERSITY OF ZAGREB
FACULTY OF ELECTRICAL ENGINEERING AND COMPUTING

MASTER THESIS Nr. 724

RECOMMENDER SYSTEM FOR MOBILE APPLICATIONS

Vanda Viljanac

Zagreb, June 2014.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

DIPLOMSKI RAD Br. 724

PREPORUČITELJSKI SUSTAV ZA APLIKACIJE NA POKRETNIM UREĐAJIMA

Vanda Viljanac

Zagreb, lipanj 2014.

Contents

List of Figures

List of Tables

Introduction

With a rapid increase of mobile applications available for download it has become a very difficult task for the user to find exactly what he wants. In order to choose one among all of the offered applications he has to put in a lot of effort and waste a lot of time. Most famous Apple's slogan: "There's an app for that!" has proven itself to be correct long time ago with the help of both *iStore* and *Google Play* which, today, offer over 1 million applications for iOS and Android platforms [1].

To make things easier for the users app markets are using recommender systems, which provide users with app suggestions. Unfortunately these kinds of recommendations neglect the very mobility of the users and their constantly changing context [2]. Although every application naturally serves a special purpose not every application can be addressed to a special context of use. To eliminate such downsides, in this work a new solution is proposed which monitors user's behavior and uses contextual information for creating recommendations.

The purpose of this thesis is to investigate how to aid users through the process of deciding which application to choose by providing them with recommendations. For that reason a recommender system for mobile applications is developed as a proof-of-concept.

In the first chapter the introduction into recommender systems is written which is followed with the introduction into mobile recommender systems. Then, related work is presented with examples of already developed app recommending engines. In the fourth chapter a proof-of-concept application called *AppDetective* is presented in a detailed way. Next, the results of *AppDetective* are commented and discussed. This work ends with a conclusion and suggestions for possible improvements and future work.

1. Recommender Systems

Recommender systems are tools and techniques that provide users with suggestions on items or information that may be of use or of interest to them [3]. They cope with the "information overload problem", i.e. the typical state of a web user, of having too much information to make a decision or remain informed about a topic. One of the major motivations for a recommender system is serendipity, to help the user make fortunate discoveries he was not explicitly looking for. Their core computational task is to predict the subjective evaluation a user will give to an item. This prediction is computed using a Number of predictive models that have a common characteristic. For example they exploit the evaluations or ratings provided by user's previous viewed or purchased items. Based on a particular prediction technique, recommendation systems have been classified into tree main categories [4]. In this chapter recommender system's techniques and approaches will be introduced.

1.1. Content-based

Content-based recommendations use methods which focus on item descriptions and characteristics. These methods create a profile for every user (*content-based profile*) that memorizes characteristics of previously viewed items. Comparing the information from the profile and item descriptions recommender system tries to find items suitable for the user. In other words content-based algorithms recommend those items that are similar to the items, which have been previously viewed by the user [5]. For example, a system recommending movies would analyze the movies a user likes to find out what they have in common in terms of content, i.e. actors, directors, genres, et cetera. This information will constitute the user’s preferences, which are used to find movies with a high degree of similarity to the liked ones [6].

Unfortunately content-based recommenders have a few drawbacks. One of them is the so-called *new user problem* in which the user has to view or rate certain amount of items before his preferences could be understood. Also, sometimes

they may recommend items that are too similar to each other and provide the user with unnecessary information.

Figure 1.1 illustrates an example of content-based recommending. Peter is the person receiving recommendations and he likes lemons. By comparing Anna's and Tom's likes it is obvious that the they both like lemons and grapes. The recommending engine therefore concludes that lemons and grapes are similar. If Peter likes lemons than he is recommended with grapes.

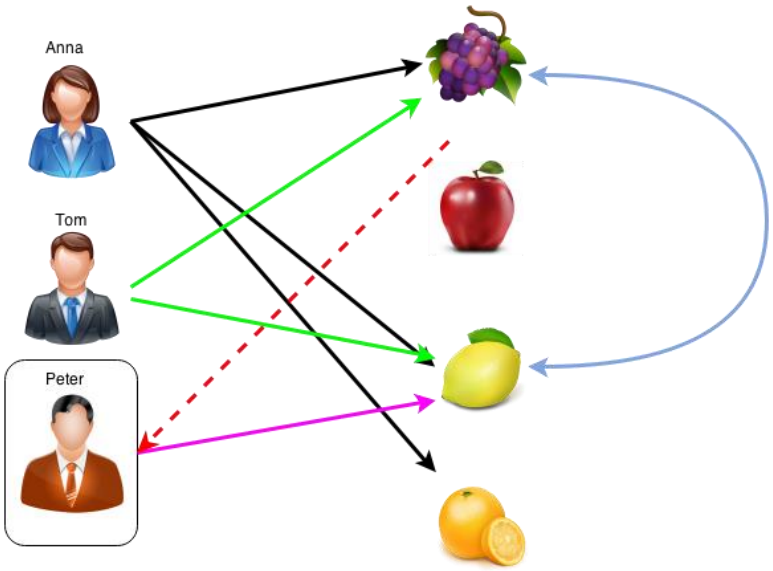

Figure 1.1. Content-based recommendations

1.2. Collaborative filtering

Collaborative recommender systems collect feedback information from the users that rate items. They connect the users, which manifest similarities in ratings and use that information for recommendations. In short, collaborative filtering algorithms recommend those items, which were approved by similar users. This kind of recommending is also called *personalized recommending* [5]. A movie recommender system would find peers, users who have similar rating patterns to the user receiving recommendations. The movies with the highest ratings according to the peers, and which the user has not yet seen, would then be recommended [6].

Collaborative recommenders take advantage of human judgments but also suffer from the *new user problem*. In addition, they suffer from so called the *new item problem* that causes new items to be ignored (i.e. not recommended) until a certain amount of users have rated the item [6].

Figure 1.2 illustrates an example of collaborative recommending. Peter is again the person receiving recommendations but this time he likes lemons and apples. If we search for the person with similar preferences like him then we can notice that Anna also likes lemons and apples. She is a user similar to Peter. Because she also likes oranges and grapes, both items are recommended to Peter.

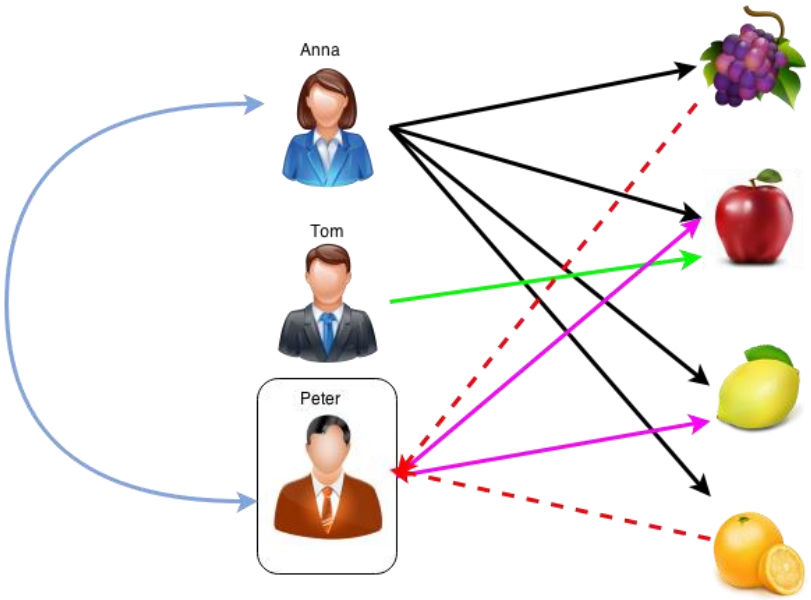

Figure 1.2. Collaborative recommendations

1.3. **Hybrid**

Both of the above-described techniques are efficient only in certain conditions and have different kinds of drawbacks. For this reason hybrid recommender systems have been developed in a way that they combine abilities of both collaborative and content-based recommendations. As an example, such a system could implement the two methods separately and then combine their results. A hybrid approach could also incorporate any other method, such as recommending items based on what has been consumed in locations close to the user.

2. Mobile recommendation systems

In the past recommender systems were developed only for personal computers. Nowadays smartphones replaced PCs in many aspects and we use them for making decisions and informing ourselves. Unfortunately this task is becoming harder every day as the amount of information on our mobile phones increases. The recommender systems that were developed for personal computers cannot be used for the same purpose because of the many differences in the domain of mobile phones that would make such recommendations less successful.

Some of the main differences and mobile service characteristic are:

- **User mobility** refers to the fact that the user can access a mobile information system in different locations. For example a traveler, landed at Vienna airport, can use a mobile phone to access *Tripadvisor* recommender system, for deciding where to book a room in Vienna. In this example, the user is mobile, he can access the same hotel recommender system everywhere, for instance, before or during the travel, and by means of several kinds of devices, e.g., through a kiosk, or with his laptop. A good recommender system should be designed in such a way that this user mobility is supported and exploited by using the knowledge about his current context i.e., that his current position is Vienna.

- **Device portability** refers to the fact that the device used to access the information system is mobile. For instance our user may access the hotel recommender system with a laptop or a smart phone, or a PDA (Personal Digital Assistant). Device portability is the dimension that has been studied in the past mainly because of the impact of the physical characteristics of the mobile devices, i.e., limited screen size, limited computation power and data storage, on the human/computer interaction.

- **Wireless connectivity** refers to the fact that the device used to access the recommender system is networked by means of a wireless technology such as Wi-Fi. The network is used to access some components of the recommender system that are not residing on the device, and without the need of any wire. Actually this is probably the most important technological development, which really caused the mobile revolution, but it has not been extensively exploited in the recommender system literature [4].

Mobile recommender systems development has been driven but also constrained by these three dimensions. The system can be aware of them and adapt its recommendations to them or try to avoid more complicated scenarios and ignore context data. For example the above-described *Tripadvisor* recommender system could exploit current time and location information of the user to decide on the number of recommended items and their ranking. Or even more, the user interface could be adapted to the device by taking the information of the device type. All of that in order to improve the usability and quality of the recommender system's service. On the other side it could provide static recommendations just taking explicitly stated user preferences into consideration.

One of the biggest challenges of recommending on a mobile phone is a smaller screen size compared to traditional personal computers. The screen size of future devices is unlikely to improve as it is a necessity for the device's mobility [7]. The common form to display recommendations is via list of ranked items in descending order, much like the results of a regular search. Several techniques have been proposed to solve the problem of the small screen when displaying search results on mobile devices, with the typical approach being to display a short description for each item in the result [7]. A short description of an item is not sufficient so it it necessary to enable a way of displaying more information about it.

Mobile devices also have some advantages so they are not all about limitations. For example context information can significantly improve the quality of recommendations. Physical location and weather data can be an important and valuable source of information. This type of data discovers the context in which the mobile phone is used at a certain time.

3. Related work

The great information overload in the context of mobile applications has led to the development of mobile recommender systems. Many of them focus on point-of-interest, travel, tourism, and media. However, there has not been a lot of research done in the area of recommending games and applications, which is the focus of this work.

Most popular mobile recommender systems today are [4]:

- **Tourist Guides** - this type of applications received the largest attention and popularity among all recommending applications. Their most popular functions are recommending relevant attractions and services based on the user's current location. The idea is to recommend those attractions and services that are near to the user or near to the location he enters himself.

- **Route recommendations** - offer directions to mobile users using the ability of network infrastructure to locate the exact position of the mobile user and give him detailed directions to reach a desired destination [4]. Route computation is usually the shortest path between 2 points but the user can also choose the quickest or the most economical path. Also it is possible to change routes by choosing different transportation means like car, public transport or walking.

- **Information recommendation** - is a very popular topic for recommendation systems, especially news and web pages recommendation. They are normally developed with content-based techniques as well as the hybrid ones. One of the most classical examples is daily news stories recommendation.

Based on some researches [8] explicit user ratings and input should be avoided and kept at a minimum because they require from the user to put in effort and spend time. Also it is emphasized that user's level of satisfaction in

recommendations is not only based on accuracy but it is desirable that recommendations are followed with explanations. Explanations could educate the user, as he will get a better understanding of why this item was recommended to him [9]. Diversity is also a very important factor in recommendations.

Below, some already developed recommender systems are described and their recommending techniques are studied. This is done to understand and gather knowledge about weaknesses and strengths of other solutions.

3.1. **Appazaar**

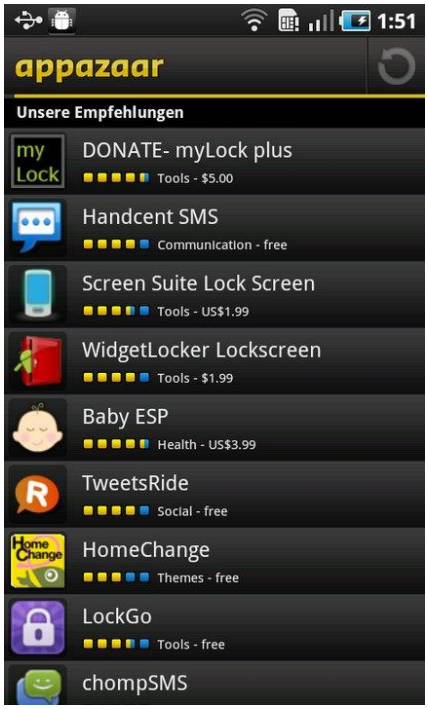

Figure 3.1. Application *Appazaar*

Appazaar (Figure 3.1) is a recommender system for mobile applications in a form of a widget. Based on users current and historical location, it recommends apps that might be of interest for the user. Therefore, it applies different algorithms from the research field of context awareness to analyze all the input data and create profiles of different situations. The *Appazaar* mobile application uploads observed user actions to a server, which in turn sends recommended applications to the mobile application, where they are presented [10].

3.2. **AppSpace**

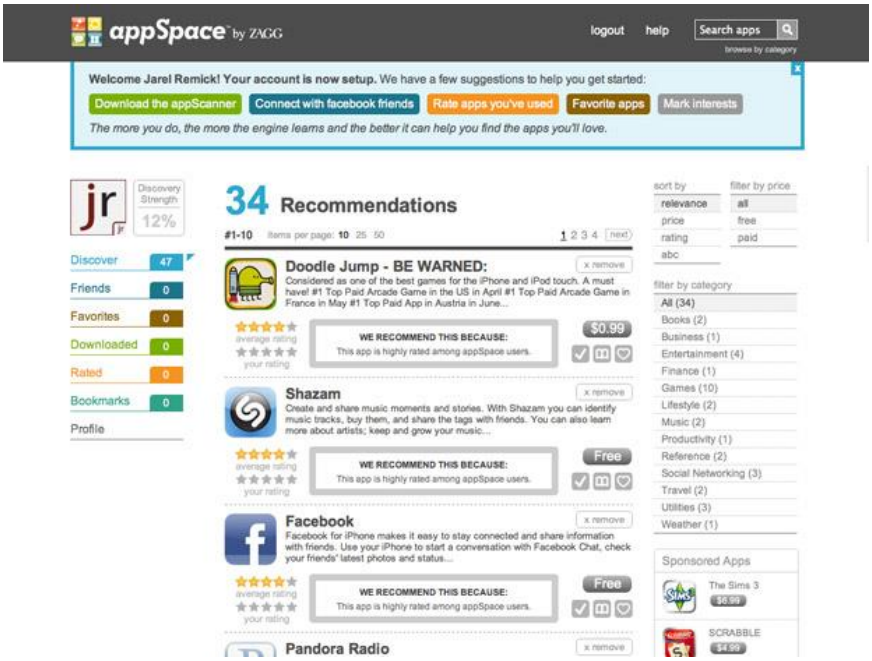

Figure 3.2. Application AppSpace

AppSpace (Figure 3.2) supports several platforms (iOS, Android, et cetera) and is a web-based service, which recommends applications to the user. The recommendation engine works based on information the user provides, meaning that the more complete the information and profile they use, the better the recommendations will be. The engine considers the users profile, current apps and the ratings when recommending new applications [11]. The biggest drawback of *AppSpace* is that the user has to explicitly inform the system of his preferences such as installed applications. However, a good feature is the list in which the user can place applications and name them. This could be very useful in a mobile context to enable users to save apps for later consideration in an archive. Lists could also be a way to obtain more metadata about an application, placing an application in a list could be seen as tagging the application with the list name [7]. *AppSpace* also gives feedback by letting the user know how good recommendations to expect, indicated by “Discovery Strength” [7].

3.3. **AppSmart**

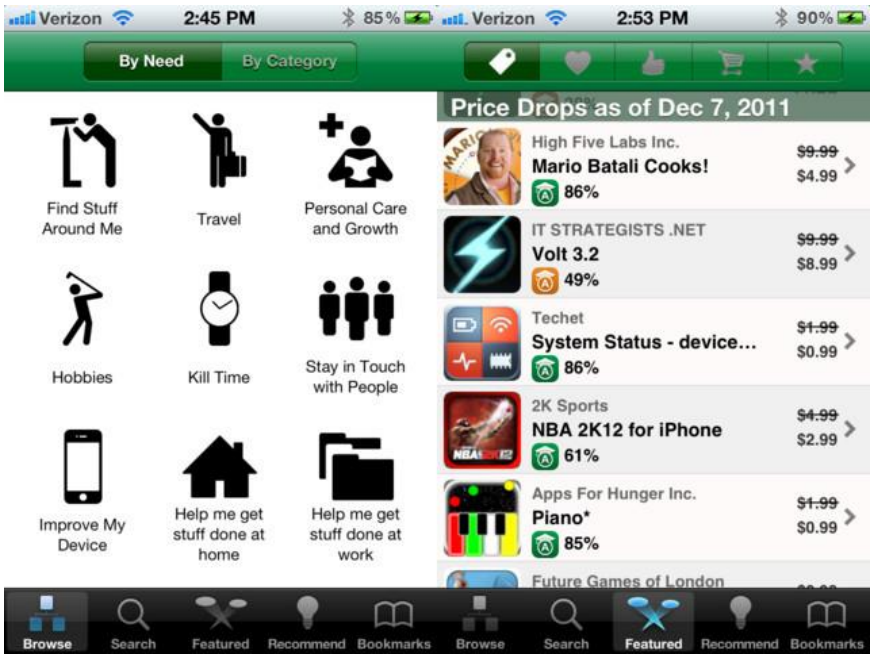

Figure 3.3. Application AppSmart

AppSmart (Figure 3.3) is an iOS recommender system that is a complete AppStore replacement. It has many other functionalities except recommending applications. It uses context, user ratings and also download history in its recommendations. There are two sections that list apps, either by Category (Entertainment, Utility, Music, etc.) or by Need (Find Stuff Around Me, Stay in Touch with People, Help Me Get Stuff Done at Home). User can view recommendations by others and see how they rank among other *AppSmart* users. *AppSmart* also provides daily app recommendations and push notifications for price drops [12].

4. AppDetective

Online app markets provide users with app suggestions. For example, *Google Play* recommends “users who viewed this app also viewed…” and “users who installed this app also installed…” applications, which are referred as “alsoview” and “alsoinstall” apps for simplicity. Obviously, such recommendations are generated by analyzing behavior of users on Google play, and they are assumed to be some kind of collaborative filtering recommendations. The recommendations help users discover new apps based on the experiences of other users. However, solely adopting the collaborative filtering approach also retains its intrinsic limitations. In particular, recommendations may be restricted to the experiences of users, most of whom only know a limited number of apps [2].

To eliminate such issues, in this work, new methods for recommending applications are proposed. First method follows user behavior and brings conclusions about his preferences. Based on those preferences, new apps, similar to those, which he often uses, are proposed. The second method uses users context such as current location to provide user with a personalized and useful recommendations.

These methods are integrated into a recommender system, which is implemented as an Android application called *AppDetective*. *AppDetective* is a proof-of-concept system and it is used for discussions, evaluations and conclusions written in this work.

In the Figure 4.1 main entities that participate in the AppDetective's recommending process are presented. The user starts the application with his mobile device on which the recommendation engine is installed. The recommendation engine communicates with Google Play Store via Google Play Engine and with third party services. After it gets the necessary data it provides the user with recommendations.

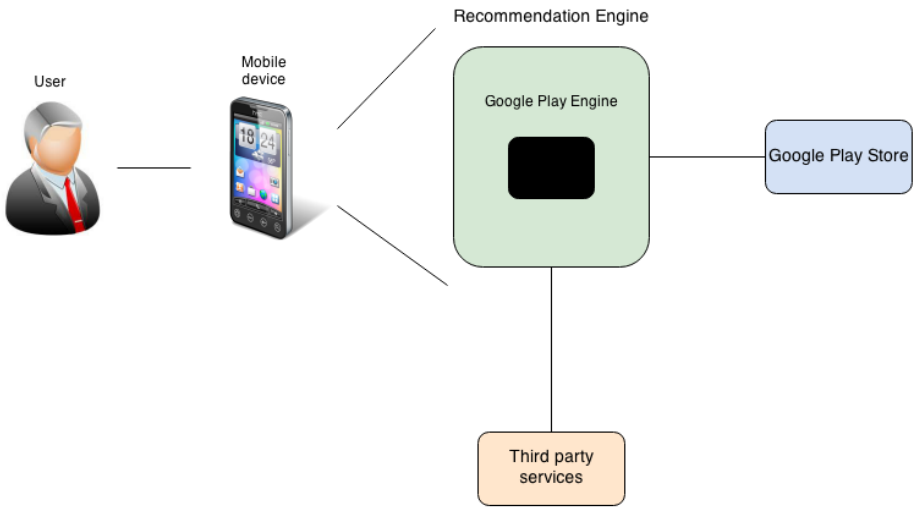

Figure 4.1. Entities that participate in AppDetective's recommendations

In the next chapter the recommender system *AppDetective* will be described and presented in a more detailed way.

5. AppDetective's Functionalities

On the figure below (Figure 5.1) main menu of the application is shown.

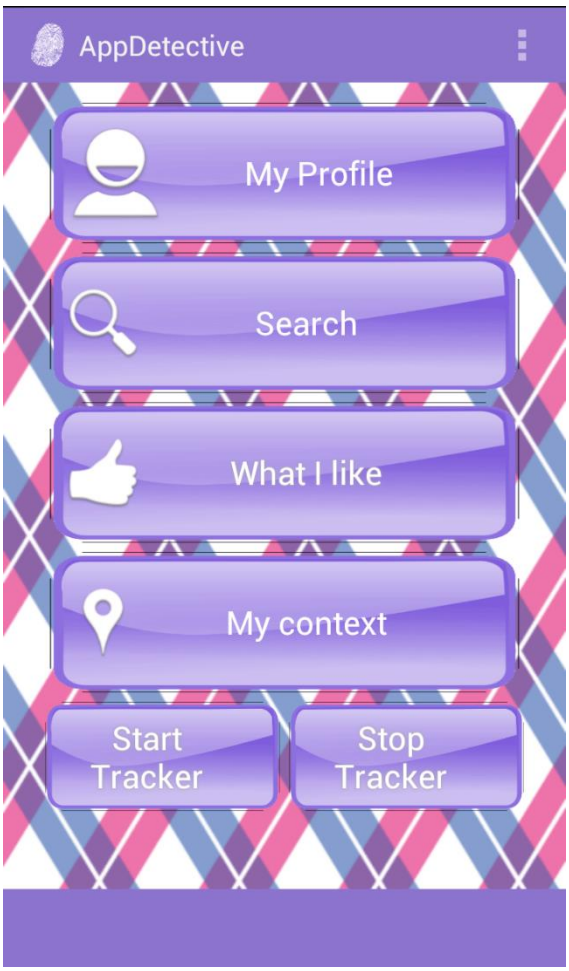

Figure 5.1 AppDetective's main menu

It consists of:

1. My Profile button
2. Search function button
3. What I like button
4. My Context button
5. Start Tracker and stop Tracker buttons

In the this chapter all main menu functionalities will be explained.

5.1. **User Profile**

By clicking the *My Profile* button user navigates to the *User Profile activity*. There, the user can check, among other basic information, collected context information that is used for app recommending. The purpose of this activity is for the user to see all the basic data gathered on one place. The Figure (Figure 5.2) shows which information User Profile presents and how does it look like.

Figure 5.2. User profile functionality

5.2. **Search function**

In the search function the user can search for applications by entering a key word. The result is generated using Android Market Engine which searches for the most popular apps that are connected to the key word. How Android Market Engine works is described in the chapters below. The search result is a list that contains first 10 apps that were retuned by the Android Market Engine. On the figure (Figure 5.3) an example is presented where the search key word is "maps". As it is shown all recommended applications are in some way connected to maps. The results are ordered in such a way that the most popular app on Google Play is the first app in the list. On the figure (Figure 5.3) it is obvious that Google Maps are more popular than Google Earth for the given query.

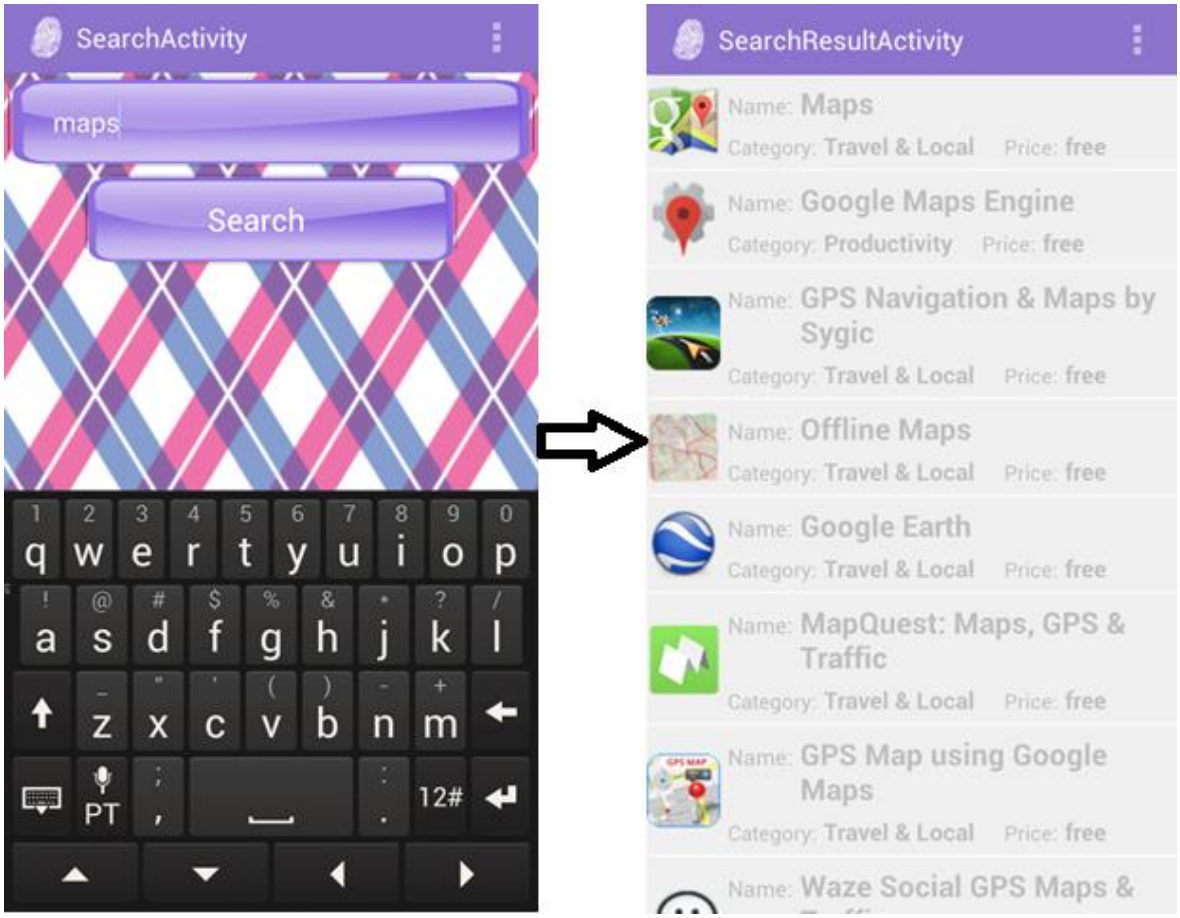

Figure 5.3. Search functionality

5.3. **What I like**

What I like activity shows the list of user's favorite apps. User's favorite app is the one that he uses the longest and the most often. Information about users favorite apps is given by the *Tracker* which's functionality will be explained in the chapters below. The list is consisted of maximum 10 apps scaled by the usage time. On the Figure (Figure 5.4) the example of users favorite apps is shown. Among all applications that are installed on users Android device, WhatsApp is the one he likes the most. It is also shown that he was using it for 5 minutes.

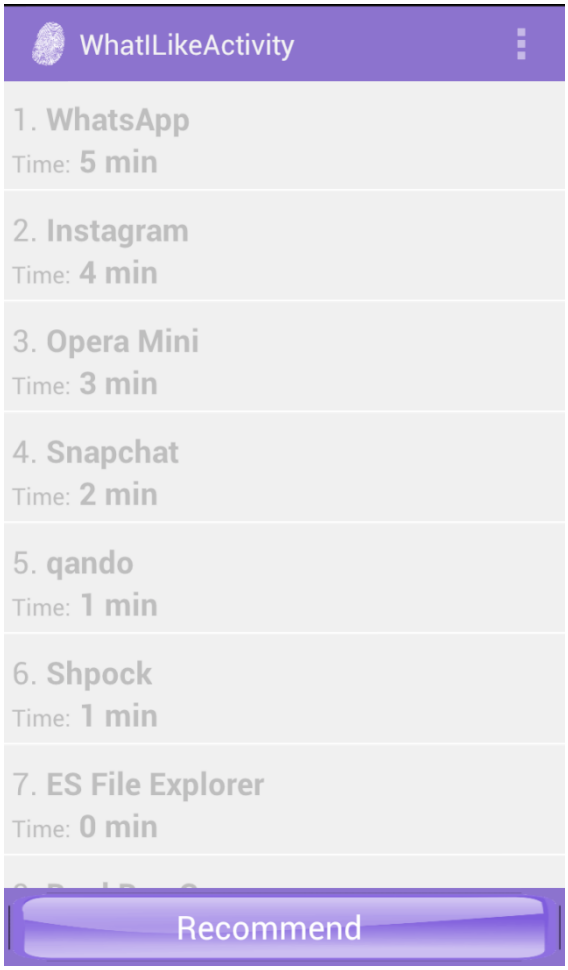

Figure 5.4. List of users favorite apps

At the bottom of the activity there is a button *Recommend*. This button calls the function, which provides the user with recommendations based on what he likes.

5.4. **Recommendations based on user preferences**

The Tracker monitors user's behavior. Every time the user uses an application the Tracker writes down the usage time in the database. If we look at the figure above (Figure 5.4) we can see that the Tracker has observed that the user has used WhatsApp for 5 minutes, Instagram for 4 and Opera browser for 2 minutes. For AppDetective's recommendation engine it means that he likes WhatsApp the most, then Instagram and then Opera and so on. It also means that he will try to find for the user apps that are connected to those applications. It does it by handing over data about users favorite applications to the Google Play Engine and receiving results about similar applications. The applications that are recommended to the user are not applications which have the same purpose as the application he likes, for example messenger applications WhatsApp and Viber, but those applications that are somehow connected to the liked applications. If the user likes WhatsApp then he will maybe want an application that provides extra smiles for WhatsApp.

In the figure (Figure 5.5) the result of recommendations based on likes is shown. We can notice that the first three applications are somehow connected to the users favorite application (WhatsApp), second three apps are connected to Instagram and so on.

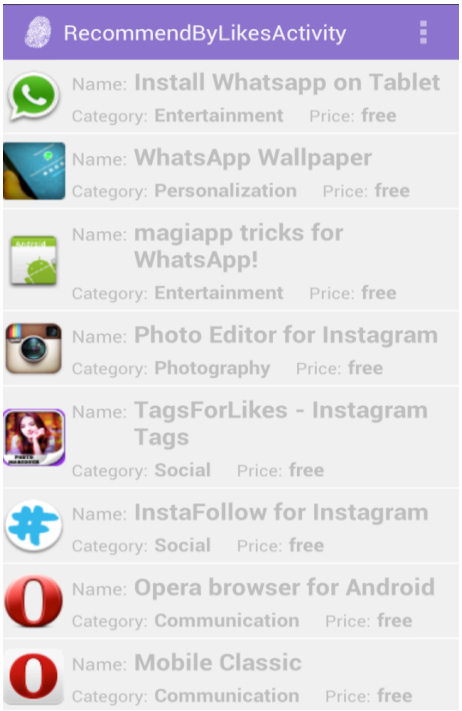

Figure 5.5. Recommendations based on user preferences

In the figure (Figure 5.6) the sequence diagram of the user preference recommendation procedure is shown. There, the workflow and the communication between actors is presented. The process starts with the user when he presses the button *Recommend* (Request for recommendations*)*. That action triggers the recommender engine which then queries the local SQLite database with classical SQL commands requesting for app usage times (Request user preference information). The database's response is an ordered set where the most used application with its usage time is positioned first in a result set (Return user preference information). Using the list of user's favorite apps the recommender engine now queries Google Play with the help of Google Play Engine (App query based on user preferences). That query is a HTTP request with app names as key words in a query. The response from Google play is also a HTTP response that contains a list of suggested apps that are connected to the key word (Return list of apps based on a query). This list is then presented to the user as Recommendations (Provide recommendations).

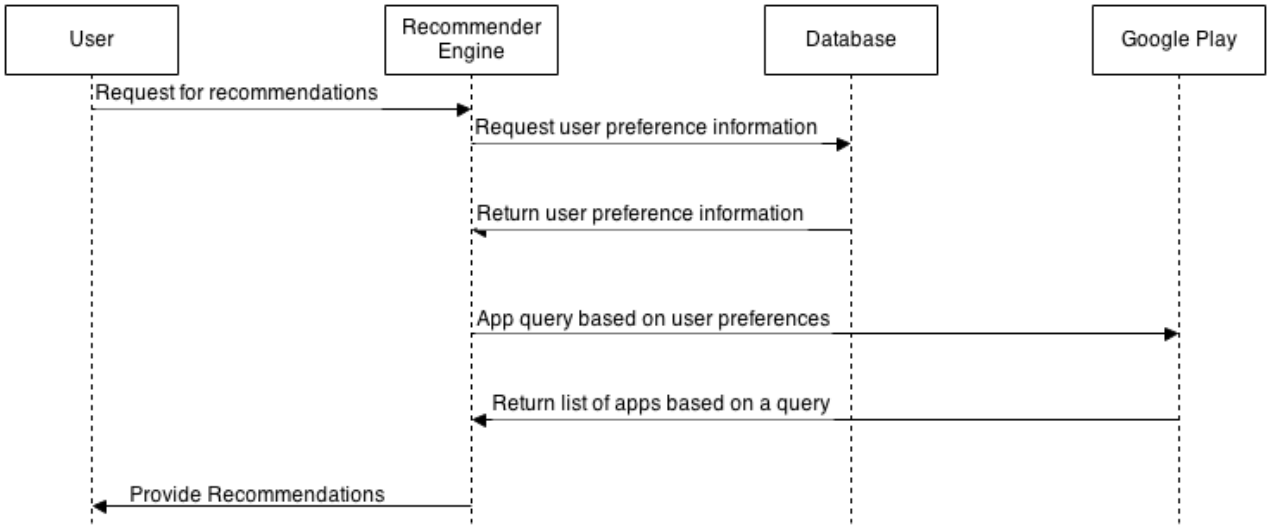

Figure 5.6. Sequence diagram for recommending based on user's preferences

5.5. My Context

My context option represents the users current context as seen on figure (Figure 5.7). It represents only that context information that can be understandable to the user (time of day, location, temperature).

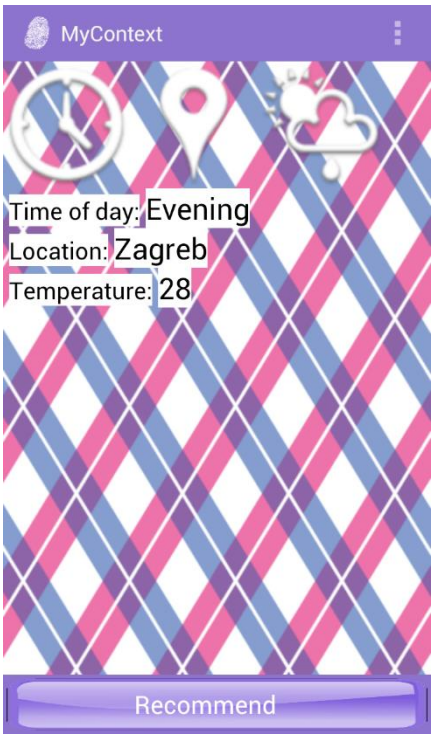

Figure 5.7. AppDetective's context information

The context is refreshed every time the user starts the application. It consists of elements that can be seen in the table (Table 5.1).

Table 5.1. Context collected by AppDetective

Physical context	Logical context
geo-coordinates	city, street
temperature	warm, cold
date	time of the year (spring, summer...)
time in milliseconds	part of the day (morning, evening...)

The context is divided into physical and logical. As it can be seen the physical context is captured by mobile phone sensors while the logical context is physical context converted into user-understandable format.

Again, at the bottom of the page there is a *Recommend* button. By clicking the button the user receives app recommendations based on his current context.

5.6. **Recommendations based on user context**

The idea of recommending applications based on users current context is to recommend those applications that might be useful to the user in the current situation. For example recommending an application that requests an outdoor activity like a *Running Tracker* may be a good recommendation on a sunny day, but a bad recommendation on a rainy day. The procedure when recommending based on user context is the same as recommending based on users preferences except the input data for Google Play Engine is different. The input data is users current context which can be seen in the table above (Table 5.1). Only the right column of the table is used i.e. logical context. For example if the user is currently located in Zagreb then the input information is Zagreb. As it can be seen on the figure (Figure 5.8) the results are connected to the current context. First three results are somehow connected to Zagreb, second three to the part of the day (evening), third tree to the year season (summer) and so on.

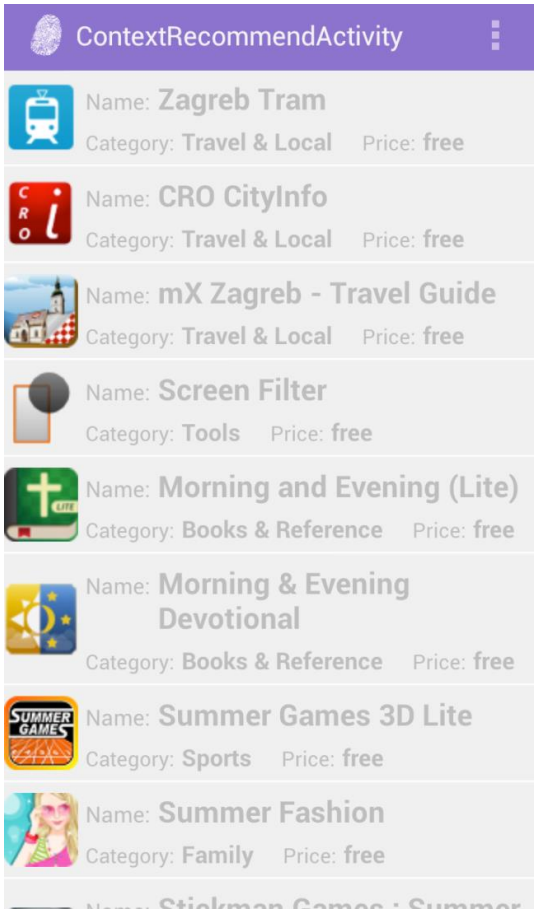

Figure 5.8. Recommendations based on user context

In the figure (Figure 5.9) the sequence diagram of the user context recommendation procedure is shown. There, the workflow and the communication between actors is presented. The process starts with the user when he presses the button *Recommend* (Request for recommendations*)*. That action triggers the recommender engine which then reads the current context from the device's sensors (Request user context, Return user context based on sensors). Then the recommender engine sends a HTTP request to the weather and location web services asking for the current weather report and the current address for the given geo-coordinates (Request current weather and location information). The web services respond using the JSON format (Return current weather and location information). Using the collected user context the recommender engine now queries Google Play with the help of Google Play Engine (App query based on user preferences). That query is a HTTP request with context information (i.e. Zagreb, evening) as key words in a query. The response from Google play is also a HTTP response that contains a list of suggested apps that are connected to the key word (Return list of apps based on a query). This list is then presented to the user as Recommendations (Provide recommendations).

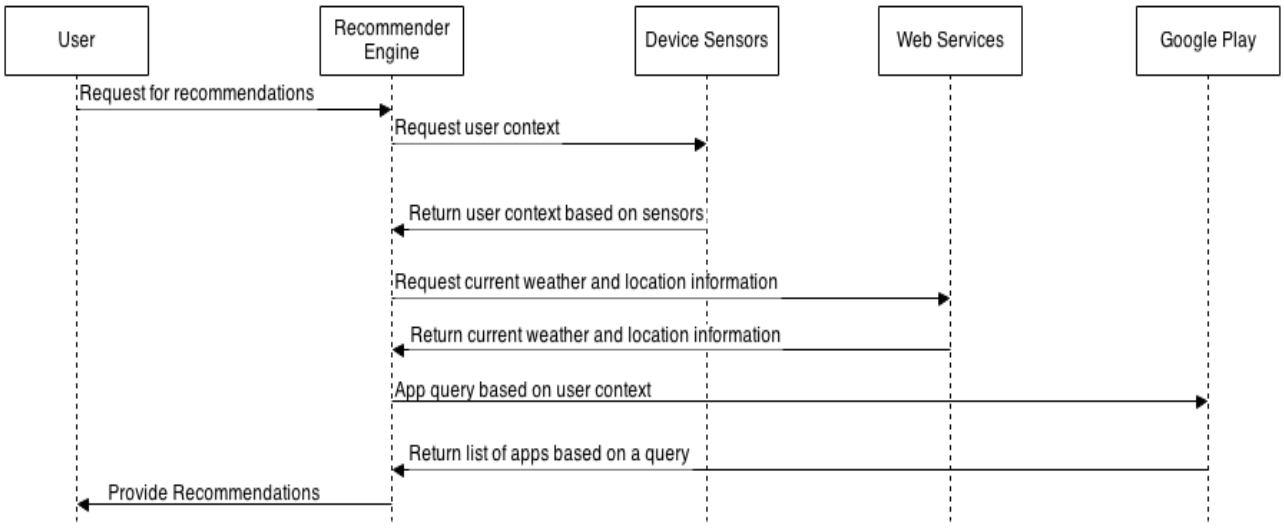

Figure 5.9. Sequence diagram for recommending based on user's context

5.7. **Start and Stop Tracker**

With the *StartTracker* button the Tracker service is started in the background. After this moment users behavior is monitored even after the application is turned off. In order for the Tracker to stop monitoring it is necessary to press *StopTracker* button. Trackers behavior can be draining for the battery so the user has the ability to stop the service whenever he wants.

6. Tracker

Tracker is an android service started by user via *Start Tracker* button in the main menu. The fact that Tracker is a service means that it works in the background completely separately from the main thread. It also means that it is not stopped when the AppDetective is paused but only when the user wants to do so by clicking the "Stop Tracker" button in the main menu.

Trackers main functionality is gathering application usage information and writing it to the database. It wants to know which installed applications are used by the user and how long. Tracker does it in a way that it calls the systems application manager and asks for the list of running processes. It does it in a loop repeatedly every second because the system cannot notify the Tracker when the process is stopped (application has ended) so the Tracker must find it out by itself. It finds it out by comparing running apps in the former second and in the current second. After it concludes that one of the applications is not used anymore it writes down the usage time and other relevant data into the database.

The biggest problem concerning Tracker is the battery consumption because it does not wait for events but runs in a loop. The best way to lower the battery consumption is to make checking intervals bigger (i.e. 3 seconds) which would make the results less precise.

7. Mobile Context information

Context information is information about user's environment. It offers new opportunities and exposes new challenges in terms of time-aware, location-aware, device-aware and personalized applications. Such applications constantly need to monitor the environment, called context, to allow the application to react accordingly to this context [13]. When it comes to context information it is not only important to supply them but also to interpret them and send them to further processing.

Context can be used in two ways. On the one hand applications can customize themselves according to the context for better usage e.g. when the location of the user determines how does the device behave. On the other hand context information can be used for creating new types of applications, like in our case with *AppDetective*.

Also there are two types of context - physical and logical [13]. Physical context represents the value of environment sensors like GPS-coordinates while logical context gives meaning to physical context and in this case could be the street name or the city name.

In *AppDetective* recommendations are calculated with the help of context information. Whenever app is started context information is gathered and stored.

Mobile context is an intersection between location, time, presence, handset capabilities and preferences as seen in Figure 7.1.

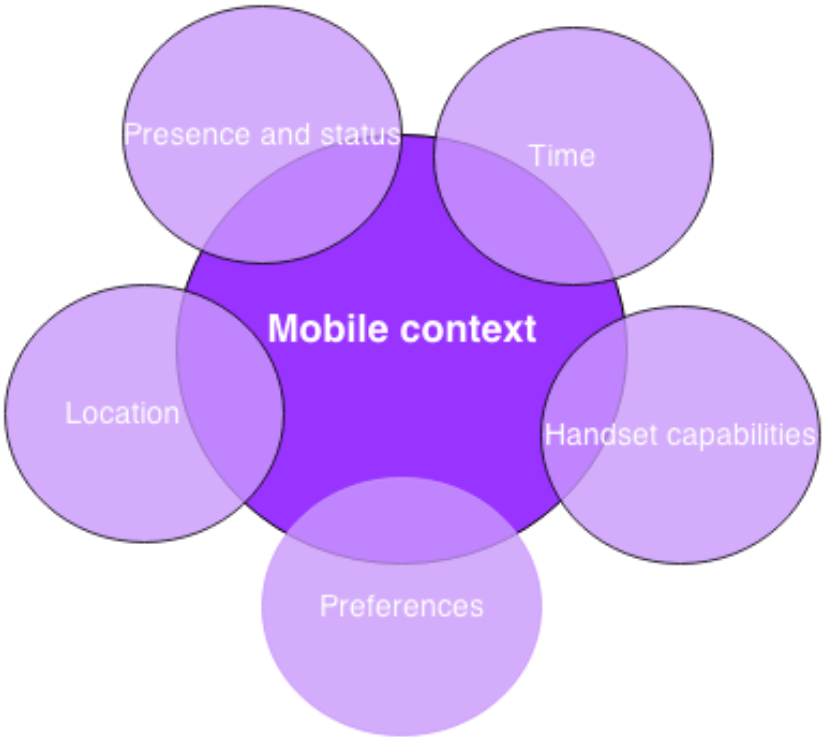

Figure 7.1. Mobile context information

8. Android Market Engine

Android Market Engine is an open-source java library that scraps Google Play Store since Google Play does not provide its own official API. It offers special functions that allow the programmer to query Google Play Store, download app reviews, app images, app comments and even apps themselves [16]. It does not return same results as Google Play search engine and that is the reason it is appropriate for AppDetective. It queries in a way that if a key word is given the apps returned will be apps that are connected to this word in someway.

It can be considered as a black box that is a part of AppDetective's recommender engine. The query keywords are black box's input and the list of apps are its output. It means that upon request, the Android Market Engine will return recommended items for a specified user. It can be designed to be used as a separate service, but in this project it is integrated into AppDetective recommender engine.

Android Market engine is a part of AppDetective's engine. It is used like some kind of interface for communicating with Google Play Store. On the figure (Figure 8.1) the structure and the position of Android Market Engine is shown. Its input is user preference information and context information and its output are list of apps connected to the input.

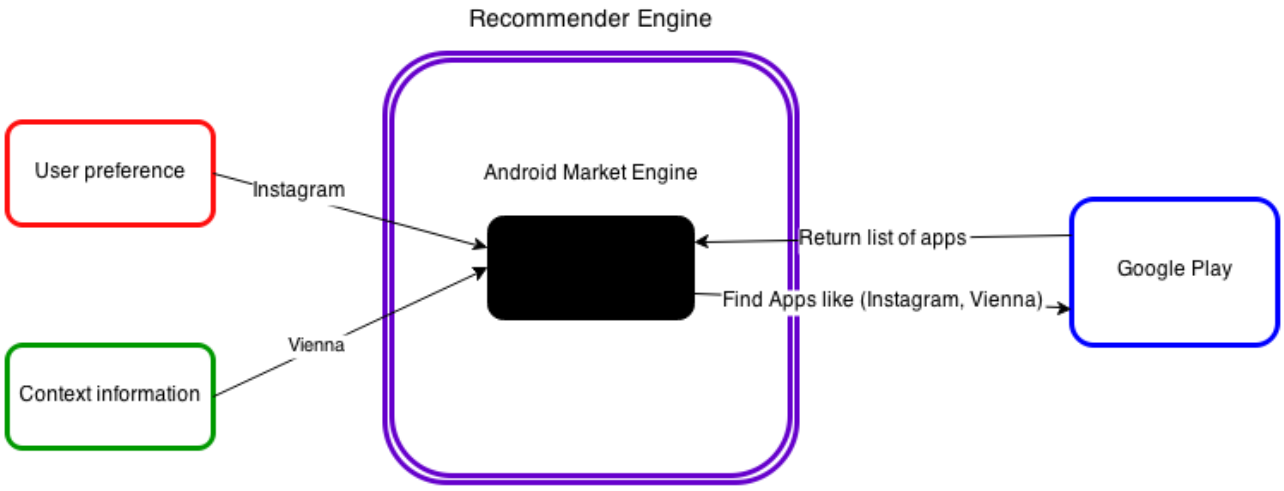

Figure 8.1. Android Market Engine

Android Market Engine is not a stand-alone entity but an integrated part of AppDetective's recommender engine. It recieves input collected by the application (i.e. Instagram as the user's favourite app and Vienna as a current location).

9. Third Party Services

The third party services are web services that are used by AppDetective in order to get logical context from the physical one. The first service is a geographical coder [17] which returns the exact position of the device (country, city, street name) when geo-coordinates are given (longitude and latitude). The second service is a weather service [18] which returns weather forecast for the current location, as well as current temperature and other weather details.

The communication with both services is carried out using the HTTP protocol in a separate asynchronous thread.

For demonstration purposes the result of a weather web service is shown in the code (Code 9.1). The result is in a JSON format and it provides different kind of weather information including wind strength, temperature, pressure and similar. For the query:
http://api.wunderground.com/auto/wui/geo/WXCurrentObXML/index.xml?query=Zagreb where it is obvious that the weather request is for Zagreb the response is:

```
<weather>Partly Cloudy</weather>
<temperature_string>70 F (21 C)</temperature_string>
<temp_f>70</temp_f>
<temp_c>21</temp_c>
<relative_humidity>78%</relative_humidity>
<wind_string>From the Variable at 2 MPH</wind_string>
<wind_dir>Variable</wind_dir>
<wind_degrees>0</wind_degrees>
<wind_mph>2</wind_mph>
<wind_gust_mph/>
<pressure_string>29.95 in (1014 mb)</pressure_string>
<pressure_mb>1014</pressure_mb>
<pressure_in>29.95</pressure_in>
```

Code 9.1 – Result of a weather query

10. Implementation

The developed proof-of-concept recommendation system is an Android application and since the development for Android platform is carried out in Java the Eclipse IDE was a tool that was chosen for the implementation. The Android platform provides APIs to access the required context information as well as to track application usage. AppDetective recommends based on two different datasets. One set is the context information and the other set are the information collected by Tracker which tracks application usage on the device. The Tracker is running in the background as a service and records the start time of the applications and their runtime. To preserve power it only keeps track of applications while the device is not in standby mode which is logical because the user can use apps only when the device is in the active mode. Also, the user can control by himself when does he want the tracker to collect information and when does he want the tracker to stop tracking by clicking the corresponding buttons in the main menu. All Tracker's data are stored into a local SQLite database. The context data is collected every time the app is started or when the location changes. The mobile application also communicates with web services via HTTP requests in order to get information about the weather prognosis for the current location. Also it uses HTTP requests to execute queries for the Android Market. As there is no official Google Play API, an open source Android API is used for searching for apps, for getting app screenshots and for downloading apps. Both responses from the weather service and Google Play are received in the JSON format for easier parsing.

11. Database

The database used for this project is an Android SQLite database. SQLite is an Open Source database. It supports standard relational database features like SQL syntax, transactions and prepared statements. The database requires limited memory at runtime which makes it a good candidate for being embedded into other runtimes [14].

It saves its data locally which means it stores the database in to a private disk space that's associated with that application. That disk space is the device´s internal storage which differs from the SD card which is also called the device´s external storage. Applications data is secure, because this area is not accessible to other applications by default [15]. When an application is uninstalled its correlated SQLite databases are deleted but when the app is reinstalled the data is preserved.

SQLite is one of the fastest-growing databases around, but that's growth is in terms of popularity and has nothing to do with its size.

The database created for AppDetective is called *AppDetectiveDatabase* and it consists of only one table called *AppUsage*. The fields of *AppUsage* can be seen in the table below (Table 11.1). The fields give a complete description of a certain application that is installed on user’s device as well as the amount of time the user has spent using that application. Two examples of database records (WhatsApp and Viber) are also given.

Table 11.1. Examples of AppDetectiveDatabase records

id	name	package	category	price	rating	description	image	time
82734...	WhatsApp	com.whatsapp	communication	free	4,2	...	/mnt/sdcard/image_whatsapp	36
22345...	Viber	com.viber	communication	free	3,9	...	/mnt/sdcard/image_viber	12

The field meanings are :

- *id* - application identification number
- *name* - applicaiton name
- *package* - application package
- *category* - application category
- *price* - price for downloading the application
- *rating* - application rating on Google Play
- *description* - short application description
- *image* - url to the folder where the application image is stored
- *time* - time in minutes which indicates how long was the application used by the user

12. System design

AppDetective generates recommendations based on two input sets. One is context data and the other one is app usage data. Context data is collected from the information from the mobile device sensors or by communicating with web services. App usage data is collected by the Tracker and then stored in the database. After the data is collected and the user requests for recommendations via recommendation user interface the app queries Android Market (Google Play) and presents the recommended apps to the user based on the response from the Android Market. The application model is shown on the photo (Figure 12.1) where we can see that it consists of two main parts - app engine with the user interface and the background service called Tracker. The Tracker works independently from the engine and records user behavior.

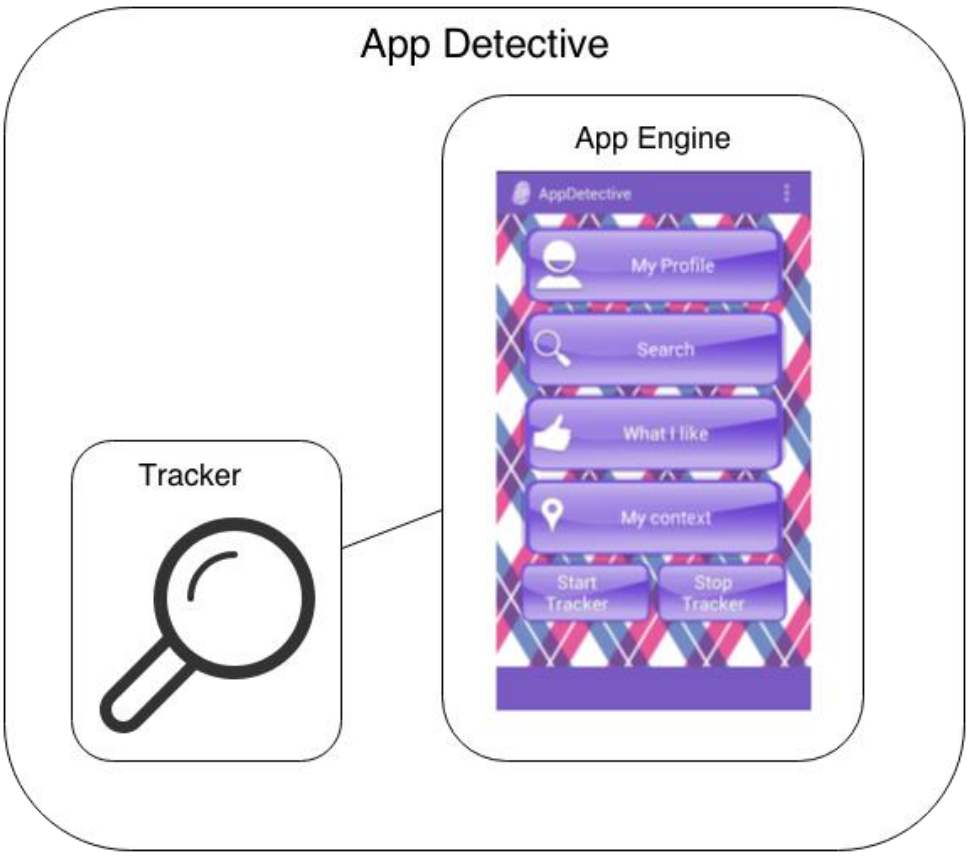

Figure 12.1. AppDetective's system design

On the next photo (Figure 12.2) all the elements of the recommender engine are shown. AppDetective communicates with many different entities in order to provide recommendations.

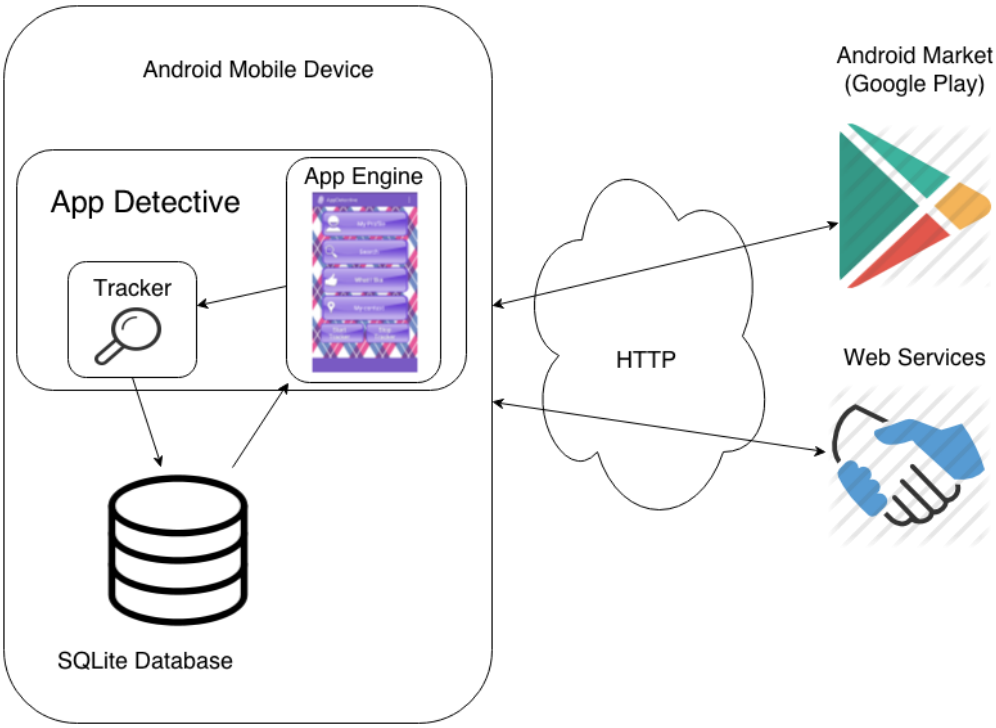

Figure 12.2. Entities that participate in AppDetective recommending

The entire process begins with the App engine which starts the Tracker. Then, after following user behavior the Tracker writes down data into the SQLlite database. Before providing the user with recommendations app Engine reads data from the database. For contextual information the engine communicates with web services via Internet and for app information it communicates with Google Play.

13. Downsides of the mobile recommender systems

There are some problems which need to be taken into perspective when designing mobile recommender systems. They are arising from certain characteristics of mobile devices which are mostly related to the limitations of the user interface. In this chapter the focus will be on general issues of mobile recommender systems and their solution in AppDetective.

13.1. Problems

First of all, recommendation sessions on small screen devices can be difficult and frustrating for end-users. It is known that users can actually read and understand information offered by small interfaces, but the size of the display can impact on user's performance. For example, on a small screen the user may be forced to carry out extensive scrolling while browsing a web page, and the more a user has to scroll down, the smaller are the chances for an item to be clicked. In addition, a user on a small screen is less effective in completing an assigned task when compared to users of large screens [4].

In addition, mobile devices offer limited input and interaction capabilities. Most existing mobile phones incorporate only a standard 12-key numeric keypad thus making quite complicated any text-input. More advanced, expensive, and bigger devices include a miniature QWERTY keyboard, but these keyboards are still much smaller than the classical ones [4]. Mobile devices also have a small number of control keys for assisting users with navigation and scrolling tasks.

Because of the above limitations, not only browsing, but especially information search, and in particular item recommendations, are problematic on mobile devices. Entering queries, e.g., based on keywords or on preferred product attribute values, is too time consuming and complex. Moreover it is quite difficult for users to process the returned result lists.

Another important issue is related to the cost of the using the recommender system. Nowadays we access the Web with ADSL connections with flat rates. These same connections could be used by advanced mobile devices equipped with Wi-Fi cards, but when we are really moving outside our home or office, we need a fast 3G data connection that is becoming reasonably priced (and flat rate) only recently [4]. For these reasons, AppDetective has tried to adjust user interface to support better browsing and searching.

13.2. **Solutions**

Recommendations are typically displayed as a ranked list of items which is also the case with AppDetective's recommendations. To address the limitations due to the small screen size it is important to convey as much information as possible on the presented item optimizing the display usage. The typical approach in mobile search consists of using snippet texts, i.e., short descriptions of the item description content. It is necessary to display only a subset of the item features that are considered as more important and not all of them [4]. That's why, in AppDetective, after search or recommendation request only important details of the result are presented to the user. In order to get more detailed information he has to click on the wanted item. See figure (Figure 13.1).

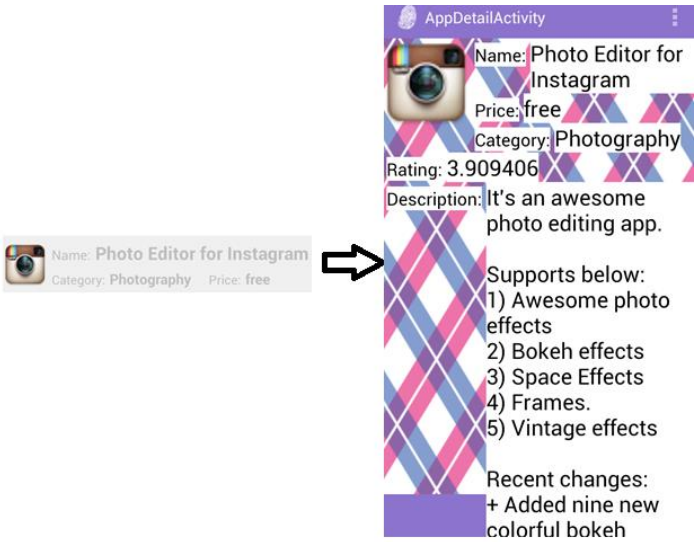

Figure 13.1. Detailed app information

Also the recommender systems should avoid requesting user input in order to prolong user sessions and keep the user interested. Mobile phone users have much shorter browsing sessions than the PC users do. AppDetective requests entering keywords from the users only in the search option.

14. Discussion and Analysis

The Android platform, which was the chosen test platform of this thesis project, has proven itself to be adequate. Possibilities to collect implicit application consumption data proved to be very good as well as the context information offered by the Android system.

Metadata for application was collected from the Google servers through a third party implementation of the Android Market API. Application portals or stores, which may run on a mobile device need to provide a web service for application metadata. It should therefore be possible to obtain this data for other platforms and portals as well [7].

A problem, when the user cannot receive recommendations without some kind of consumption history, with which are some recommenders faced, is in AppDetective solved. AppDetective can use only user's context information like location, which can be obtained immediately, to derive recommendations. The second problem when a new item, which no users have consumed, cannot be considered has not been solved because the third party API decides which apps are chosen.

Diversity - making sure recommended items are not too similar, was not implemented during this project, but the concept was identified as interesting for application recommendations. It is a desirable feature, which should be optional when requesting recommendations from the recommender engine. For instance, a request could include a parameter indicating the level of desired diversity [7].

By the same analogy with the *new item problem* and the *new user problem* in AppDetective *new context problem* appeared. It is the problem of basing recommendations on a context which has yet to be seen by the system. If there is no item consumption for this new context, the system is not able to produce recommendations, in analogy with the new user problem for collaborative filtering [7].

AppDetective delivers satisfying results for applications, considering the limited data for the available applications. If the system successfully recommends applications it is reasonable to consider that with minor changes it could recommend other kinds of items like movies or music.

Conclusion

In order to help the users to choose an application they want to download a recommender system called AppDetective was developed. Users choice was guided by recommendations based on user's current context and their preferences. Their behavior and their actions on the mobile phone were monitored by the application and with that information conclusions about their preferences were brought. AppDetective shows the potentials of mobile recommender systems over the classical recommender systems. It demonstrates how can the constantly changing context and the constantly changing preferences influence recommendations. It tries to exploit all of the advantages a mobile phone offers, like sensors and context and give to the user the most useful recommendations.

As it was said, very little of the literature has focused on context-based recommendations of mobile apps, not to mention global similarity measuring among apps. The current understanding of both user preferences and market features is far from sufficient. In this work these problems were therefore studied and an experimental solution was built. Overall, the goals set out for the project were met and the basic problems and challenges were overcome.

For the future work there are several problems that are waiting for the solution. One of the most important ones is the user's privacy. Although personalization of the application is found useful to the users, it may still cause a feeling of concern or discomfort when their location is gathered or their behavior is monitored, which could turn them away from the service.

Also the context information could be much thoroughly gathered and recommendations could take into consideration other context like users battery level or the users network signal.

For the future it would be also useful if the similarity among the apps would be measured by the app itself and not using the Google Play Engine.

Bibliography

[1] Kate T., Gauri S., Poonam S., Pooja S., Nikhilesh J, Emilee, Iziam, Bas O, *Android vs iOS,* Available: http://www.diffen.com/difference/Android_vs_iOS, 15.4.2014.

[2] Xia X., Wang X., Zhou X, *Evolving Recommender System for Mobile Apps: A Diversity Measurement Approach,* School of Computer Science, National University of Defense Technology, Changsha, China, *30.6.*2013

[3] Ricci F., Rokach L., Shapira B., Kantor P. B., *Recommender Systems Handbook*, Springer-Verlag New York, 1st edition, 2010.

[4] Ricci F., *Mobile Recommender Systems,* Faculty of Computer Science, Free University of Bolzano, Italy, 11.1.2010.

[5] Viljanac V., *Ranking of Facebook friends based on user profiles,* Faculty of Electrical Engineering and Computing, Zagreb, June 2012.

[6] Adomavicius G., Tuzhilin A, *Towards the Next Generation of Recommender Systems: A Survey of the State-of-the-Art and Possible Extensions*, IEEE Transactions on Knowledge and Data Engineering, Vol. 17, No. 6, 2005, pp. 734-749

[7] Davidsson C., *Mobile Application Recommender System, Uppsala University, Sweden, December 2010.*

[8] David M. N., *Implicit Rating and Filtering, Proceedings of the fifth DELOS Workshop on Filtering and Collaborative Filtering*, Budapest, November 1997.

[9] Herlocker L.J., Konstan A.J., Riedl J., *Explaining collaborative filtering recommendations,* ACM conference on Computer supported cooperative work, Budapest, November 1997.

[10] Appazaar information: *appazaar beta, Avaliable:*http://appazaarbeta.android.informer.com/

[11] AppSpace information: *New application recommendation engine AppSpace,* Avaliable: http://www.socialmediaportal.com/News/2009/08/ZAGG-launches-new-app-recommendation-engine-AppSpace.aspx

[12] Tahnk J.L., *10 Apps for finding Apps*, 12.2.2012, Avaliable: http://mashable.com/2012/02/12/apps-for-finding-apps/#gallery/10-apps-for-finding-apps/50bdde0eb589e40aaa00013d

[13] Hofer T., Schwinger W., Pichler M., Leonhartsberger G., Altmann J., *Context-Awareness on Mobile Devices - the Hydrogen Approach*, 36th Hawaii International Conference on System Sciences, IEEE, 2002, Avaliable: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.102.9433&rep=rep1&type=pdf

[14] Android official site, SQLitedatabases: http://developer.android.com/training/basics/data-storage/databases.html

[15] Vogel L, *Android SQLite database and content provider, Avaliable:* http://www.vogella.com/tutorials/AndroidSQLite/article.html, *19.8.2013,*

[16] Alexandre T., *An open-source API for the Android Market,* Avaliable:

https://code.google.com/p/android-market-api/

[17] Weather web service: http://openweathermap.org/API

[18] Location web service: http://www.wunderground.com/

Summary

Recommender system for mobile applications

This work investigates how to guide users through the process of deciding which application to choose by providing them with recommendations. For that reason a recommender system for mobile applications was developed as a proof-of-concept. This recommender system is an Android application that recommends other mobile applications for download based on mobile context and user's preferences.

At the beginning the introduction into recommender systems is written which is followed with the introduction into mobile recommender systems. Then, related work is presented with examples of already developed app recommending engines. In the following chapter a proof-of-concept application called AppDetective is presented in a detailed way. Next, the results of AppDetective are commented and discussed. This work ends with a conclusion and suggestions for possible improvements and future work.

Keywords: recommender system, mobile, context, preferences, application, deciding, suggestions, download.

Sažetak

Preporučiteljski sustav za aplikacije na pokretnim uređajima

Ovaj rad istražuje kako voditi korisnike kroz proces odabira novih mobilnih aplikacija na način da im se pruže preporuke. U tu svrhu razvijen je mobilni preporučiteljski sustav u obliku Android aplikacije koji korisniku preporučuje druge mobilne aplikacije za preuzimanje na temelju mobilnog konteksta i korisničkih preferencija.

Na početku rada nalazi se uvod u preporučiteljske sustave iza kojeg slijedi uvod u mobilne preporučiteljske sustave. Zatim je predstavljen prijašnji rad koji se bavi tom temom te su dani primjeri već razvijenih preporučiteljskih aplikacija. U slijedećim poglavljima aplikacija razvijena u sklopu ovog rada naziva AppDetecitve je objašnjena na detaljan način. Na kraju, donesen je zaključak te su dani prijedlozi o unaprijeđivanju razvijene aplikacije.

Ključne riječi: preporučiteljski sustav, mobilni, kontekst, preferencije, aplikacija, odlučivanje, sugestije, preuzimanje.

Dodatak A: Uputstva za pokretanje na hrvatskom jeziku

Android aplikacija *AppDetective* izrađena je u svrhu diplomskog rada na Fakultetu elektrotehnike i računarstva.

Kako bi se aplikacija instalirala na najlakši mogući način potrebno je preuzeti priloženu AppDetective.apk datoteku na osobno računalo. Zatim je potrebno uzeti mobilni uređaj sa Android operacijskim sustavom te ga USB kabelom spojiti sa osobnim računalom i omogućiti prijenos podataka sa računala na mobitel. Preuzetu AppDetective.apk datoteku nužno je spremiti na bilo koju lokaciju vanjske memorije mobitela (SD karticu).

Nakon prijenosa datoteke moguće je odspojti mobitel. Koristeći pretraživać datoteka na mobilnom uređaju potrebno je pronači lokaciju na kojoj je prethodno spremljen AppDetective.apk.

Dvostrukim klikom na AppDetective.apk pokreće se instalacija aplikacije. Kako bi instalacija bila uspješno dovršena potrebno je aplikaciji odobriti sve sigurnosne zahtjeve. Nakon što je i ovaj korak završen aplikacija je instalirana te spremna za korištenje.

Appendix B: Installation instructions

Android application AppDetective was developed for the purpose of master thesis at the Faculty of Electrical Engineering and Computing, University of Zagreb.

The first step for installation is to transfer the enclosed AppDetective.apk file to personal computer. Next, it is necessary to connect a mobile device using an USB cable with the computer and enable data transfer between the mobile device and the PC. After that, the downloaded AppDetective.apk should be transfered on the mobile devices external storage (SD card).

Now, it is possible to disconnect the mobile device from the PC. The next step includes using file explorer to navigate through the files on the mobile phone and to find where AppDetective.apk was saved.

After finding it by double clicking on it activate the installation. Next, it is necessary to allow all the permissions that are requested from the app. After this is done AppDetective is ready for use.