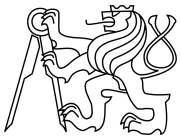
CZECH TECHNICAL UNIVERSITY IN PRAGUE

FACULTY OF INFORMATION TECHNOLOGY



ASSIGNMENT OF BACHELOR'S THESIS

Title:	Android Mobile Application for Analysis of Cosmetic Products Composition
Student:	Oleksandra Liutova
Supervisor:	Ing. Dana Vynikarová, Ph.D.
Study Programme:	Informatics
Study Branch:	Software Engineering
Department:	Department of Software Engineering
Validity:	Until the end of summer semester 2016/17

Instructions

The aim of the thesis is to analyse, design, and implement a mobile application for analysis of chemical composition of cosmetic products. The main function of the application is to warn the user about possible dangerous or unhealthy substances. A cosmetic product composition can be specified in several ways - by scanning the product´s bar code, by typing the product´s name, or by typing the product´s chemical composition. The application then finds the product in a private database and provides information about chemical substances contained in the product and their evaluation.

- Make an analysis of existing applications for analysis of cosmetic products composition.
- Make an analysis of user requirements for such an application.
- Select technologies for its implementation.
- Based on the analysis, design an Android application and implement it using the selected technologies.
- Test and document your application.
- Assess the benefits of your application.

References

Will be provided by the supervisor.

L.S.

Ing. Michal Valenta, Ph.D.
Head of Department

prof. Ing. Pavel Tvrdík, CSc.
Dean

Prague December 19, 2015

Czech Technical University in Prague
Faculty of Information Technology
Department of Software Engineering

Bachelor's thesis

Android Mobile Application for Analysis of Cosmetic Products Composition

Oleksandra Liutova

Supervisor: Ing. Dana Vynikarová, Ph.D.

15th May 2016

Acknowledgements

I want to thank everyone who helped me work on this thesis, especially my supervisor Ing. Dana Vynikarová, Ph.D. for her guidance and helpful advice, and my parents for supporting me.

Declaration

I hereby declare that the presented thesis is my own work and that I have cited all sources of information in accordance with the Guideline for adhering to ethical principles when elaborating an academic final thesis.

I acknowledge that my thesis is subject to the rights and obligations stipulated by the Act No. 121/2000 Coll., the Copyright Act, as amended, in particular that the Czech Technical University in Prague has the right to conclude a license agreement on the utilization of this thesis as school work under the provisions of Article 60(1) of the Act.

In Prague on 15th May 2016 .....

Czech Technical University in Prague
Faculty of Information Technology

Citation of this thesis

Liutova, Oleksandra. *Android Mobile Application for Analysis of Cosmetic Products Composition.* Bachelor's thesis. Czech Technical University in Prague, Faculty of Information Technology, 2016.

Abstract

This bachelor thesis describes a process of design and implementation of an Android application for analysis of cosmetic products composition. The main goal of this application is to help user with choosing high quality cosmetic products, suitable for him, by providing information about healthiness of individual products and substances in their composition. The work contains analysis of present state of this problem. Within several chapters the process of application development is gradually described: analysis, design and implementation. I have succeeded in creating a prototype of this application with basic functions like searching for product by scanning its barcode or by name, searching for particular substances in composition, displaying complete information about products and compositions and warning user about allergens. Based on the testing phase, there are possibilities of further development described in the last chapter.

Keywords Android application, cosmetic products composition, barcode scanning, user interface, database, multiple languages

Abstrakt

Tato bakalářská práce popisuje proces návrhu a implementace Android aplikace pro analýzu složení kosmetických produktů. Hlavní cíl této aplikace je usnadnit uživateli výběr kvalitní kosmetiky, vhodné přímo pro něj, na základě poskytnutých informací o zdravotních rizicích produktu a jeho složení. Práce obsahuje analýzu současných řešení tohoto problému. V několika dalších kapitolách je postupně popsán proces vývoje aplikace: analýza, návrh, a implementace. Úspěšně byla dokončena tvorba prototypu aplikace se základní funkcionalitou. V prototypu je umožněno hledání produktu oskenováním jeho čárového kódu nebo podle jména, dále vyhledávání konkrétních látek ve složení, zobrazení kompletní informace o produktu a kompletní informace o jeho složení. V neposlední řadě aplikace dokáže zobrazit uživateli varování o alergenech ve složení výrobku. V poslední části je určeno, jakým směrem, v návaznosti na testování, by se mohl řídit budoucí vývoj.

Klíčová slova Android aplikace, složení kosmetických přípravků, čtečka čárových kódů, uživatelské prostředí, databáze, podpora více jazyků

Contents

List of Figures

List of Tables

Introduction

The number of smartphone users has been growing extremely fast in the last few years. Most of us are used to smartphones helping us in our daily life: we use them for communication, planning, searching for information, etc. I believe the result application of this bachelor thesis has the potential to make its contribution to improving our lives as well.

Nowadays consumer awareness of cosmetic ingredients is growing and people are more and more interested in products' impact on their organism. Being one of those people, I often find myself in a situation when I need to buy some cosmetic or hygiene products, and I have to go through composition of different offered items, searching for unhealthy substances or things I am allergic to. While shopping for personal care products, it's very hard to choose those products, that are of the highest quality and are actually beneficial for our organism and effective. It's not easy to understand the meaning and impact behind a product's composition for a person who is not a chemist, pharmacist or another professional in this area. The situation is even worse for those people, who are allergic to certain substances. Often one substance can have around 20 different names, so the allergic person has to look through composition, looking for any name of a huge number of his allergens' names. I realized, that all of these inconveniences can be resolved by creating a special mobile application, and that is why I have chosen it as the topic of my Bachelor thesis.

The first chapter of this document - "Analysis" - contains analysis of existing solutions of this problem, requirements analysis and use-case modeling. In chapter "Design", where I present application design, such as choosing technologies, database model and user interface design. After design phase, a prototype with basic application functionality was implemented, which is described in chapter "Implementation". This prototype has been properly tested, and results of testing phase is presented in next chapter - "Testing". The last chapter - "Future improvements" - is based on testing phase and presents possible future improvements of application.

Aim of Bachelor thesis

The aim of this Bachelor thesis is to create a prototype of an application for analysis of cosmetic products composition. User will be able to find a product he is interested in by scanning it's barcode or inserting its name and see if the product contains any harmful substances or allergens. If the product is not present in the database yet, it will be also possible to provide it's composition and see it's evaluation. A special function of this application will enable user to add a list of substances he is allergic to. This way, when some product contains any of those substances, the application will warn him.

Chapter 1

Analysis

1.1 Target users group

Since the application provides user with evaluation of products' composition and gives detailed information on it, the main target users group contains basically everybody, who is interested in quality of cosmetics he/she uses. Functionality of warning about allergens defines another target group: people, who are allergic or sensitive to some substances. These two groups are depicted in a Venn diagram in picture 1.1.

1.2 Analysis of existing solutions

In this section I will present the results of performed analysis of existing solutions. Below there are descriptions of three applications, that have similar functionality to the one I design in this Bachelor thesis.

1.2.1 Healthy Living

Application *Healthy Living* offers the following functionality: scanning product's barcode, search product by name, favorites, history and others.[1] It has its own of evaluating product's healthiness. Healthy Living is based on two databases: Skin Deep, which provides information about personal care products, and Food Scores, that contains information about food.

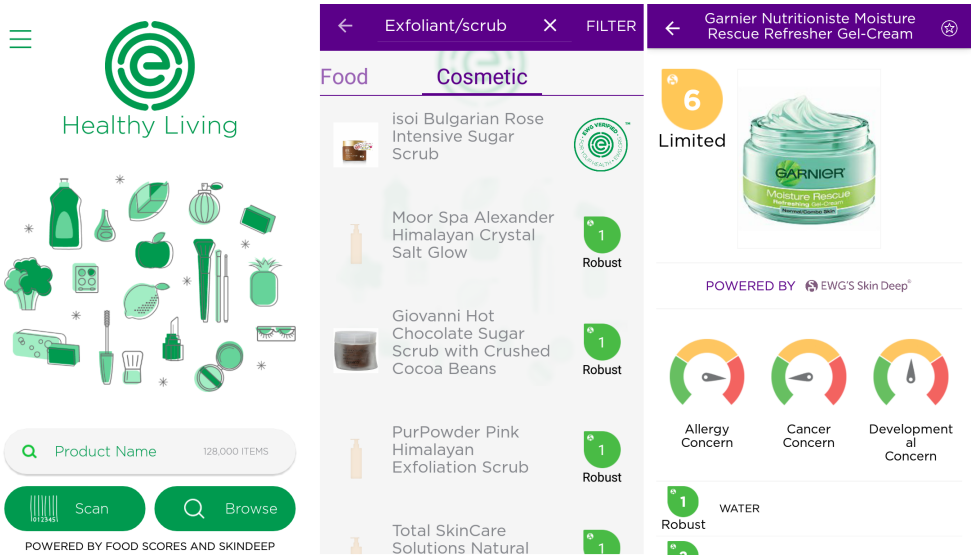


Figure 1.2: Healthy Living screenshots[1]

Advantages:

- As you can see in picture 1.2, *Healthy Living* has nice-looking and intuitive user interface.
- User can browse through different products according to their categories.

Disadvantages:

- According to users' reviews on Google Play, the application's database does not contain enough data to be actually useful.
- Does not offer an option of entering substances manually in case the product is not present in database.
- Items with substances' names in composition are not clickable - there is no description of how substances effect human organism.
- Does not have a functionality of allergens' detection.
- Offers a shortened history of search of only last five products.

1.2.2 GoodGuide

Application *GoodGuide* claims to offer similar functionality.[2] According to description of application on Google Play, their database contains information about over 170 000 products. However, it's last update took place on March

5, 2012, and I have not succeeded in getting it to work on any of 5 different Android devices. As you can see on last screenshot of picture 1.3, application kept displaying message "Network Error. Tap to Reload" and did not succeed in connecting to the internet. The first two screenshots were taken from application's page on Google Play and display what application should look like.

Figure 1.3: GoodGuide screenshots[2]

1.2.3 Think Dirty

Think Dirty is a project worth mentioning in this chapter. As described on their official webpage, *"Think Dirty® app is the easiest way to learn about the potentially toxic ingredients in your cosmetics and personal care products. Just scan the product barcode and Think Dirty will give you easy-to-understand info on the product, its ingredients, and shop cleaner options!"*[3] Think Dirty claims to have information about over 350 000 products of over 3 200 brands. Unfortunately, their application is not yet available for public use, so I didn't have the opportunity to test it. At picture 1.4 you can see what the application is going to look like.

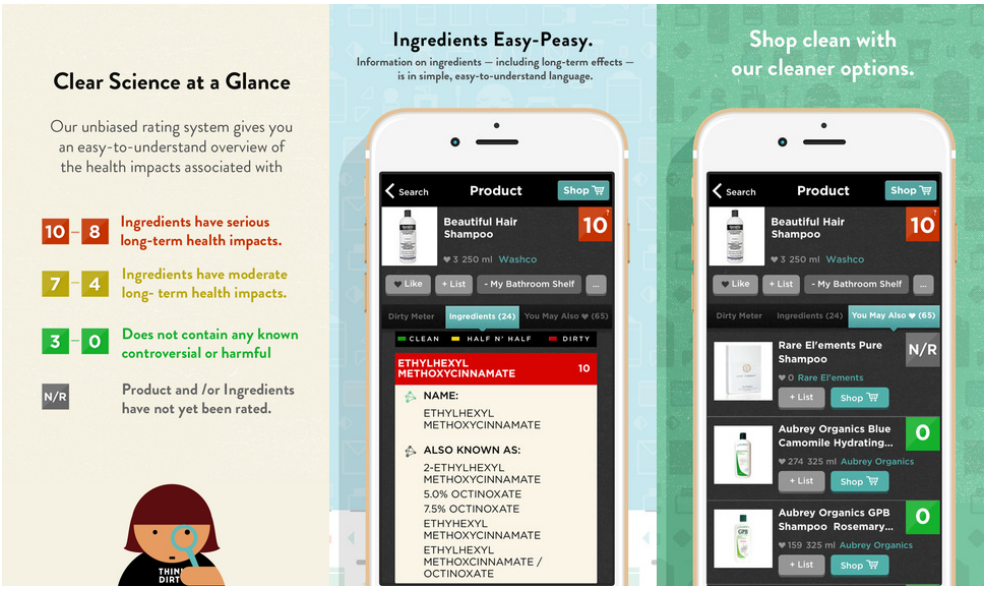


Figure 1.4: Think Dirty screenshots[3]

1.2.4 Summary

In conclusion, out of three applications with similar functionality, described in this chapter, only *Healthy Living* is actually available for public use. And this application leaves a lot of space for improvements. This shows how yet unexplored the market of personal care awareness is. It is also worth mentioning, that none of presented solutions is oriented on Czech market.

1.3 Requirements analysis

1.3.1 Functional requirements

F1 Search in products The application will implement search of product in the database: it will request certain information from server and receive it back.

F2 Search in substances The application will be able to search in substances and retrieve information about it.

F3 Scan barcode The application will enable scanning product's barcode in order to find it in database.

F4 Display product information The application will display recieved product information to the user.

F5 Display composition information The application will display received information for each substance in composition.

F6 Display history of search The application will be able to display the history of user's search in products and enable moving to any product by clicking on it.

F7 Add/remove product from favourites The application will enable user to add any product to favourites or remove it, and later access all of his favourite products.

F8 Edit list of user's allergens The application will enable user to add and edit a list of substances he is allergic to.

F9 Allergens detection While displaying composition, the application will highlight all the substances, that user is allergic to.

F10 Reporting error The user will be able to report any kind of error he detected in application. These reports will be stored on server and resolved later.

F11 Change app language The user will be able to change language, in which information in application is displayed.

1.3.2 Nonfunctional requirements

N1 Synchronization with server The application must be able to retrieve actual data from server.

N2 Supporting different languages Application's architecture will support different languages on both client and server side. That means, it will display information in language chosen by user, and it will also request information in this language from server.

N3 Simple and intuitive design The application's design must be simple and intuitive.

N4 High level of supportability on mobile devices Minimum 70% of mobile devices should be able to run this application.

1.4 Use case modeling

1.4.1 Use cases and their scenarios

UC1 Search product by scanning its barcode

1. In main menu user clicks *Scan Barcode*.
2. Application opens photo camera with barcode scanner.
3. User focuses camera on barcode of his product.
4. Application detects barcode, processes it and directs user to layout with information about the product.

UC2 Search product by entering its name

1. In main menu user clicks *Search by Product Name*.
2. Application opens layout with EditText field.
3. User starts entering his product's name.
4. Application shows tips for products' names.
5. User clicks on the tip with his product's name.
6. Application directs user to layout with information about the product.

UC3 Search composition information by entering each substance's name

1. In main menu user clicks *Type Composition*.
2. Application opens layout with EditText field, an *Add* button for adding new fields and a *Continue* button.
3. User enters first substance's name using tips from application.
4. User clicks on *Add* button.
5. Application adds a new EditText field.
6. In this way user enters all needed substances and clicks on *Continue* button.
7. Application directs user to layout with information about entered composition.

UC4 Add product to favourites

1. In product layout user clicks on an empty star next to product's picture.
2. The star gets golden and application displays a notification "Added to favourites".

UC5 Find a product in history of search

1. User opens the navigation drawer and selects *History*.
2. Application directs user to layout with a list of products' names with corresponding timestamps.
3. User clicks on the product he was looking for.
4. Application directs user to layout with information about the product.

UC6 Edit list of allergens

1. User opens the navigation drawer and selects *My allergens*.
2. Application directs user to layout with a list of user's allergens he entered before, an *Add* button for adding new fields and a *Save* button.
3. User clicks on a cross next to one of allergens.
4. Application removes that allergen.
5. User clicks on *Add* button.
6. Application adds a new EditText field.
7. User enters allergen's name using tips from application and clicks *Save*.
8. Application displays a notification "Saved".

UC7 Report error

1. In product layout user clicks on button *Report error*.
2. Application displays a Dialog Fragment with a Spinner for choosing error type, EditText for error description and buttons *Cancel* and *Send*.
3. User selects error type, enters description and clicks on button *Send*.
4. Application displays a notification "Error report sent".

UC8 Change application language

1. User opens the navigation drawer and selects *Settings*.
2. Application directs user to layout with a Spinner for choosing language.
3. User chooses different language.
4. Application changes the language - all visible labels get changed.

1.4.2 Functionality coverage by use cases

As you can see in table 1.1, all of functional requirements are covered by use cases.

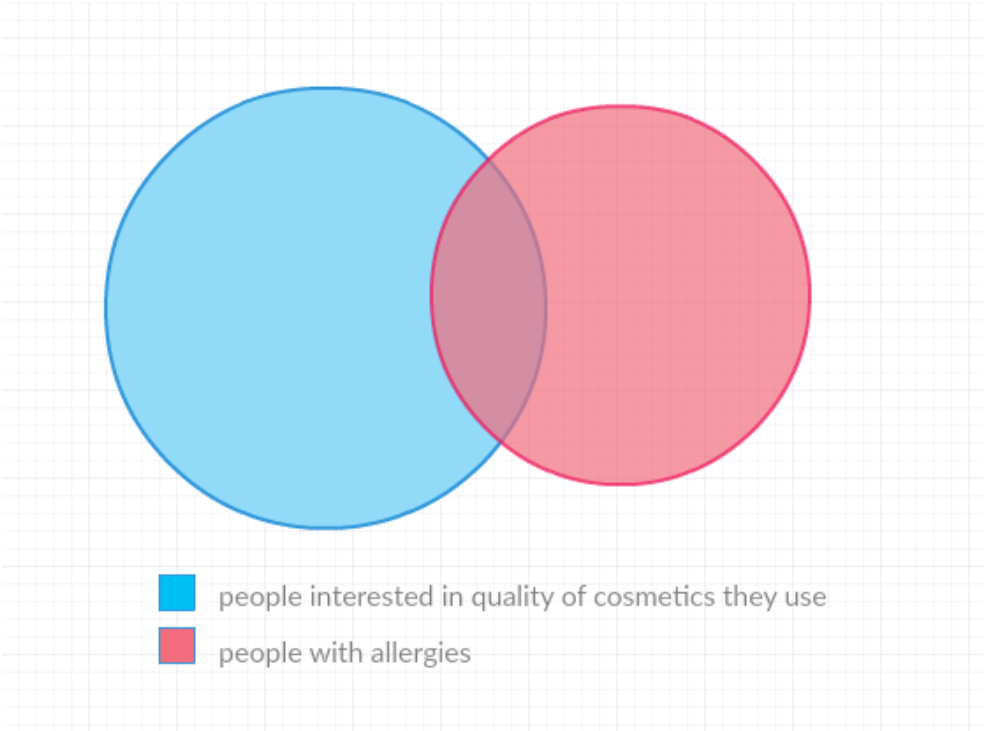


Figure 1.1: Target users groups Venn diagram

Table 1.1: Functionality coverage by use cases

	F1	F2	F3	F4	F5	F6	F7	F8	F9	F10	F11
UC1	X	X	X	X	X				X		
UC2	X	X		X	X				X		
UC3		X			X				X		
UC4							X				
UC5						X					
UC6								X			
UC7										X	
UC8											X

Chapter 2

Design

2.1 Choosing technologies

2.1.1 Android

Nowadays there is a relatively big number of different mobile platforms. I decided on creating an application for Android platform, because, as you can see in picture 2.1 it is the most popular mobile operating system today, and it is used by over 80% of smartphone users. I set Minimum SDK to API 14: Android 4.0 (IceCreamSandwich). This means, that 97.4% of android devices that are active on Google Play Store will be able to run it. Therefore, in general, over 77% of smartphone users would have access to this application, and that would fulfill the supportability requirement.

2.1.2 Android Studio

Android studio is a modern standard for developing Android applications. It is maintained by Google and is based on popular IDE(Integrated development environment) IntelliJ IDEA. Android studio provides a lot of features, for example code templates, Lint tools for analyzing code, WYSIWYG(acronym for "What you see is what you get") editor for layout design and so on. Gradle is provided in Android studio by Android Plugin for Gradle. Using simple commands in Gradle file allows to customize and configure build process. This is an advantage for creating multiple versions of one app within the same project.

Here are the main debug tools of Android Studio:

- Inline debugging,
- CPU monitor,
- Memory monitor,

- Code inspections.

Android Studio also enables using annotations. This powerful feature helps to optimize debugging and develop more advanced code. For example a user can define parameter of some function with annotation @NonNull and if he calls this function, Android Studio will warn him in case there is a possibility of null parameter.

2.1.3 Parse

The Parse platform provides a complete backend solution for mobile applications.[5] I have decided to use it for creating database for a few reasons:

1. Parse provides very intuitive API for different platforms, including Android.

2. It is an objective database, which means that no mapping between SQL tables and Java objects is necessary. This makes working with such a database much easier.

3. Parse provides free and reliable access to their server under condition of retrieving at maximum 1000 objects at a time. For current project, this is a neglectable limitation.

2.1.4 Realm

"Realm is a mobile database: a replacement for SQLite & Core Data."[6] I have decided to use it for implementing mobile database for the following reasons:

1. Realm provides very intuitive API.

2. It is an objective database, which means that no mapping between SQL tables and Java objects is necessary.

3. Realm is a developing technology, currently having no limits or charge for using it.

2.1.5 Butterknife

Butterknife[7] is a useful tool for programming in Android. It provides a functionality of binding objects: ButterKnife connects all of the views with IDs in layout with Java code. First I had to call ButterKnife.bind(...) function and then use ButterKnife annotation @BindView(ID), where ID has to be unique in the whole layout.

In an example below you can see a comparison of code, that has the same functionality, but the first is written in Android, and the second uses Butterknife. As you can see, Butterknife helps to keep code more clear and understandable.

Android without ButterKnife:

```
TextView mTextView;

@Override
public void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);

setContentView(R.layout.main_activity);

mTextView = (TextView) findViewById(R.id.text);
mTextView.setOnClickListener(new OnClickListener() {

@Override
public void onClick(View v) {
Toast.makeText(this, "Clicked!", Toast.LENGTH_SHORT).show();
}
});
}
```

Android with ButterKnife:

```
@Override
public void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);

setContentView(R.layout.main_activity);

ButterKnife.bind(this)

}

@OnClick(R.id.text)
public void onClickButton(){
Toast.makeText(this, "Clicked!", Toast.LENGTH_SHORT).show();
}
```

2.1.6 Barcode scanner

Nowadays there is a wide variety of barcode scanners available for free usage. I have chosen to use ZXing ("Zebra Crossing") library for barcode scanning, as it is easy to integrate into project, it doesn't take too much space, it has very intuitive interface and I was completely satisfied with its quality of scanning barcodes.

2.1.7 Picasso

Picasso[8] is a simple tool, that helps loading images to layout asynchronously. Android has specified lifecycles of activities and fragments and if, for instance, user changes orientation of screen, all activities and fragments are destroyed and created again. This process can stop downloading image and cause errors. This pitfall is handled automatically by Picasso. It also takes care of memory and disk caching so it is efficient when you want to display some images more than once. Picasso also provides some functions for images transformation like, for example, resize or centerCrop.

Below you can see an example of code using Picasso. It is simple for understanding and for writing as well. Without Picasso the same functionality could be achieved by creating a fragment, setting it as retained (which means it will not be destroyed after configuration changes) and connecting it to current activity/fragment where image has to be displayed.

```
Picasso.with(context) // Context like activity or application context
.load(url) // URL from we want to download image
.placeholder(R.drawable.placeholder) // image which will be shown
    before image is downloaded
.error(R.drawable.placeholder_error) // image which will be shown if
    there is error
.into(imageView); // view where we want to place our image
```

2.1.8 Support Library

Support Library[9] provides the latest Android features like RecyclerView or DrawerLayout even on older devices. Some functions and elements were added to Android after release version. For maintaining backward compatibility Google created library which implemented these new features with code understandable for older devices.

According to [10] there are two most used versions of Support Library:

- v4 version - designed for devices with Android 1.6 and higher,
- v7 version - designed for devices with Android 2.1 and higher.

Table 2.1: Server and mobile storage comparison

	Server	Mobile
Storage size	Theoretically unlimited	Very limited - storing data on a mobile device is highly costly
Requires internet access	YES	NO
Access speed	Depends on internet access - in most cases pretty slow	Fast

In this project I have used v7 version of Support Library, because in contains the elements I needed to use (like, for example, RecyclerView).

2.2 Database model

Even though the server side is not part of this thesis, it was necessary to design a database structure for storing data on device and on server. In this section I will explain which way of storing data I have chosen and why.

First of all, I split the database into 2 parts - server and mobile. Each of them has its specificities. You can see a comparison table in figure 2.1.

I have decided to use server for storing general data - information about products and substances, and also storing error requests from users. This data in constant for all users. As for mobile storage - I use it for storing user-specific information: user's allergens, favourites and history of search. This solution additionally strengthens users' privacy - application does not send any user's private information to server. It is stored locally only. As I have mentioned in chapter "Choosing technologies", I have decided to use objective database model. The database model on picture 2.2 demonstrates the structure of data storage on both server and client side. There are 14 classes in the model: 3 of them are used for storing data on mobile device, and the rest - on server. Arrows in this model represent foreign key relationships between classes. These relationships are implemented by simply storing id of referenced object in String format. For example, each object Product, as any other object on server, has its own object id, generated by Parse. Each instance of class ProductBarcode holds a reference to instance of Product by storing its id in variable productID.

2.3 UI design

In this section I will present user interface I have designed for this application.

2.3.1 Home screen

Home screen is one of the most often displayed screens, it is shown when user starts application. It has to provide fast and easy access to the most often used functions - that are scanning product's barcode, searching product and typing composition. That is why the home screen will have 3 corresponding buttons, spread all over the screen, as you can see in picture 2.3 The first button opens a photo camera with barcode scanner, the second opens layout for search, and the third one opens layout for entering composition. There will also be a drawer available in every screen, including home, through hamburger menu icon in upper left corner of the screen. The drawer will provide access to history, favorites, allergens, settings, help, about and home.

2.3.2 Product screen

A product screen has to present information about product in a clear and understandable way, corresponding to modern standards of Android applications design. In picture 2.4 you can see the wireframe I have designed for this screen. At the top of the screen there is a picture of product, in its upper left corner there is a star for adding to favorites, in upper right corner - icon showing product's safety level. Under the picture there is product name, and under it - composition table. There is also a button for reporting any kinds of errors in order to receive user's feedback.

2.3.3 Enter product name screen

As you can see in the second wireframe in picture 2.4, Enter Product Name Screen is very straightforward: it has an EditText field on top, where user can enter input. As he types, application gives him tips with names of available products in a list under. Clicking on one of these tips will direct user to a screen with chosen product's information.

2.3.4 My allergens screen

As shown in picture 2.4, My Allergens Screen contains a list of EditView fields with names of user's allergens and cross icons on the right for deletion. There is a plus button for adding new fields and a save button for confirming changes.

2.3.5 History screen

History Screen is depicted by the last wireframe in picture 2.4. It consists of a simple list of history records, containing name of product and a timestamp. Each element in the list is clickable and holds a reference to its product.

2.3.6 Summary

In this chapter I have described the structure of a few basic screens of application. Of course, there are a lot more of them, but their design is very straightforward or similar to the presented wireframes.

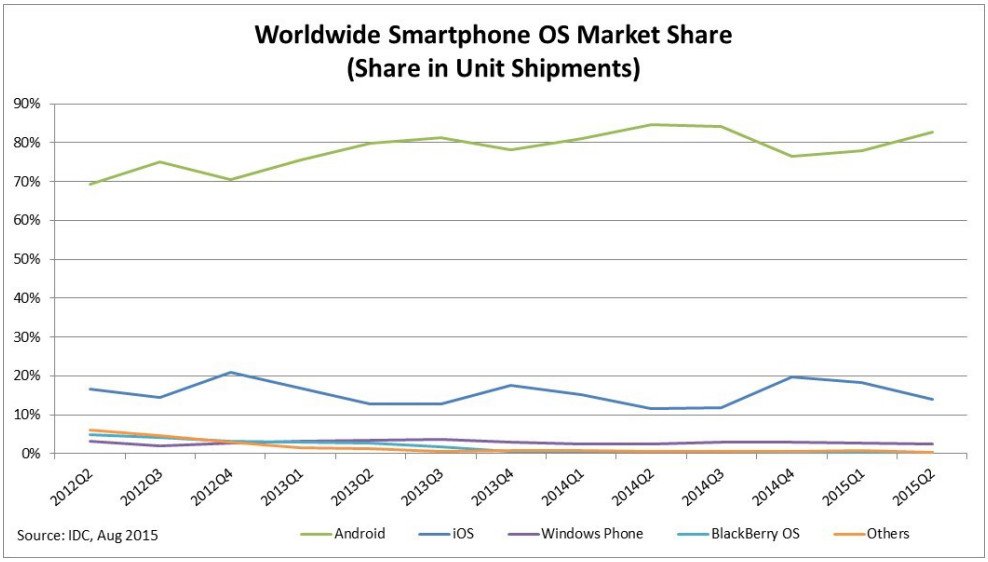


Figure 2.1: Android market share[4]

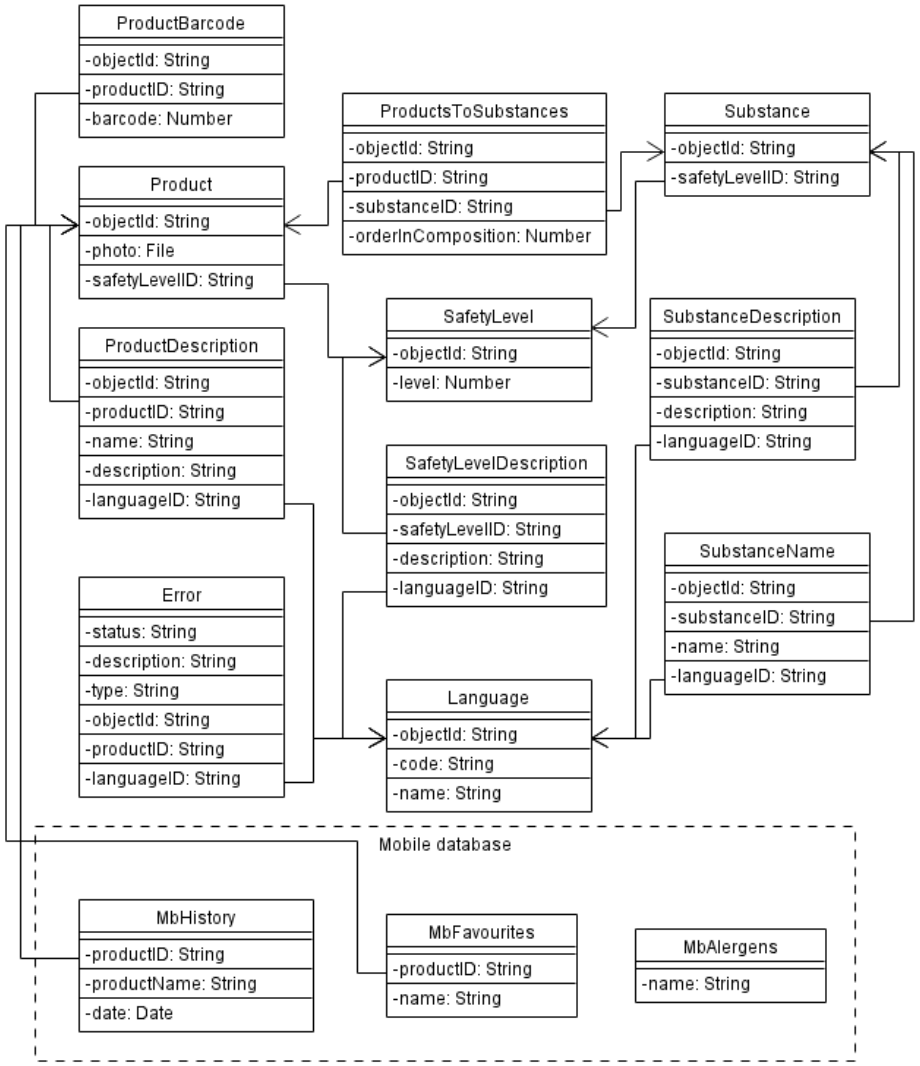


Figure 2.2: Database model

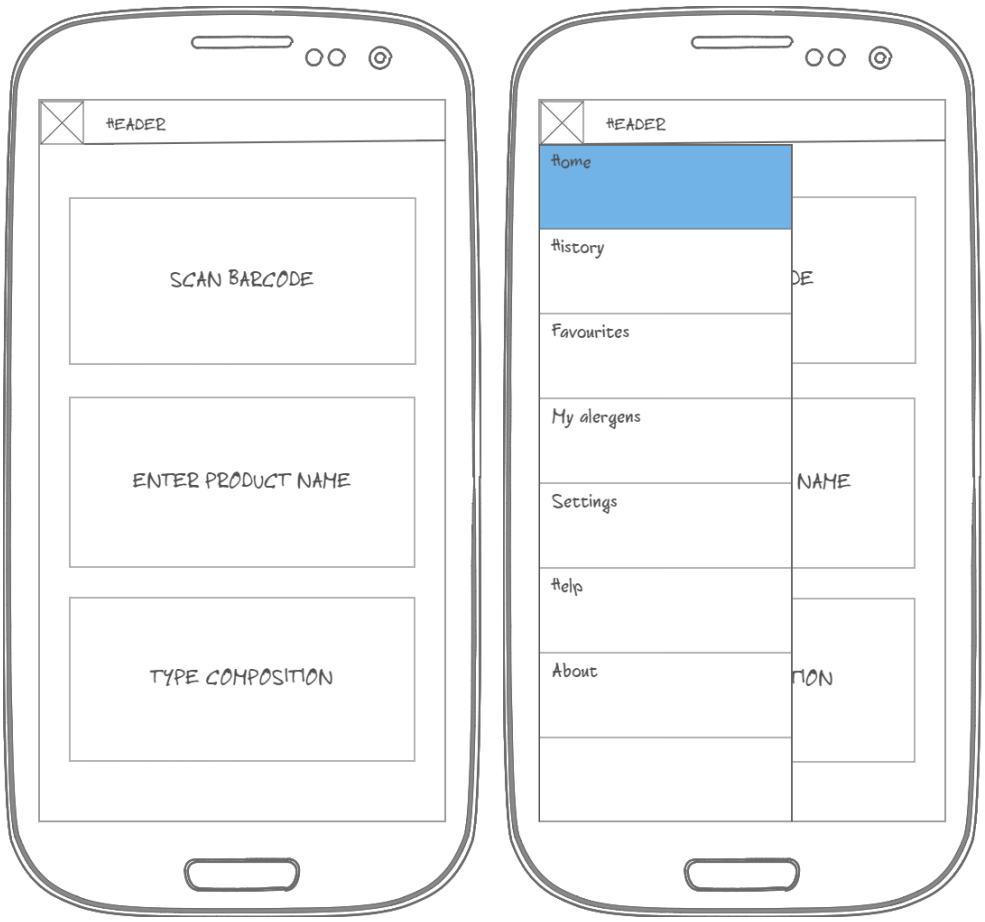


Figure 2.3: Home wireframes

Figure 2.4: Other wireframes

Chapter 3

Implementation

3.1 Packages structure

In picture 3.1 you can see the structure of packages I created for this project for better code maintenance. This picture is a screenshot directly from Android Studio and shows the hierarchy of packages in this project very clearly.

The helper package contains classes with general functionality like hiding keyboard. This function is called every time user clicks away from EditText element. Its code looks like this:

```
public static void hideKeyboard(BaseActivity activity) {
    View view = activity.getCurrentFocus();
    if (view != null) {
        InputMethodManager imm = (InputMethodManager)
            activity.getSystemService(Context.INPUT_METHOD_SERVICE);
        imm.hideSoftInputFromWindow(view.getWindowToken(), 0);
    }
}
```

Package listener contains 4 interfaces, that fragments executing asynchronous tasks implement.

Package model contains 15 classes, that are used for mapping objects from database. Classes with prefix "Db" are from server database and extend ParseObject. Each of these classes implements a few getters and setters and method getQuery, which is later used for building queries. Classes with prefix "Mb" are from mobile database and extend RealmObject. Each of them has some variables and corresponding getters and setters.

Package network contains 4 classes used for executing asynchronous tasks. I will describe them a bit more in section Asynchronous operations.

The last package view contains 3 further packages: activity, adapter and fragment. Activity package contains 4 activities: BaseActivity, MainActivity, BarcodeScannerActivity and SplashScreenActivity. BaseActivity is each

other's activity parent. It is used to implement some functions, that are common for all the activities. SplashScreenActivity is started on application launch. It displays a splash screen with application's logo and after a determained period of time starts MainActivity. There is also BarcodeScannerActivity, which is responsible for scanning barcode and is started when user presses "Scan Barcode" button.

Package adapter contains 5 adapters. I will describe them in section RecyclerView.

The fragment package contains 14 fragments. All of them extend BaseFragment. It has a method getBaseActivity, which returns BaseActivity. This BaseActivity maintains all fragments' changes and transfer. This is done by calling replaceFragment method. Each fragment can also change title in Toolbar through BaseActivity. All of this you can see in the code below.

```
// in BaseFragment
    public BaseActivity getBaseActivity() {
        Activity activity = getActivity();
        if (activity != null) {
            return (BaseActivity) activity;
        }
        return null;
    }

// in BaseActivity
    public void replaceFragment(BaseFragment baseFragment) {
        getSupportFragmentManager()
            .beginTransaction()
            .replace(R.id.container, baseFragment,
                baseFragment.getTag())
            .commit();
    }
// and if we need to replace fragment in another fragment
getBaseActivity().replaceFragment(ProductFragment.newInstance(null,
    currentProductsIDsList.get(position)));
```

3.2 RecyclerView

RecyclerView[11] is a very common element in Android development. The main reason for creating this class was simplifying displaying a large data set. ListView was used before RecyclerView, but nowadays tends to be replaced by RecyclerView. According to [12] the main reason for this is that RecyclerView has an advantage of using ViewHolder pattern, which is a very efficient way to implement a list of items. Next important reason is that RecyclerView has Layout Manager. It is used for easy implementation of different ways of displaying items.

RecyclerView, unlike ListView, supports better item animation and more intuitive item decoration. For filling RecyclerView with data user needs to implement class which is inherited from BaseAdapter. The important function that needs to be overridden is getView. It returns layout for one item in the list.

In this project I have implemented several RecyclerViews and here are their corresponding Adapters:

- AlergensAdapter,
- CompositionTableAdapter,
- FavouritesAdapter,
- HistoryAdapter,
- TypeCompositionAdapter.

Below you can see a short example of Adapter used for displaying history of search. It has a constructor for passing some parameters, onCreateViewHolder method, which is called when ViewHolder is created, onBindViewHolder method, which is called every time a new element of the list appears on screen, and getItemCount, which returns the size of the list. In the same class you can see a ViewHolder class implemented. It has only constructor and some class variables. Its main function is to keep information about every single element.

```
public class HistoryAdapter extends
    RecyclerView.Adapter<HistoryAdapter.ViewHolder> {

    ArrayList<HistoryTableRow> itemsData;
    BaseActivity baseActivity;

    public HistoryAdapter(ArrayList<HistoryTableRow> itemsData,
        BaseActivity baseActivity) {
        this.itemsData = itemsData;
        this.baseActivity = baseActivity;
    }

    @Override
    public HistoryAdapter.ViewHolder onCreateViewHolder(ViewGroup
        parent, int viewType) {
        View itemLayoutView = LayoutInflater.from(parent.getContext())
                .inflate(R.layout.item_history, null);
        ViewHolder viewHolder = new ViewHolder(itemLayoutView,
            baseActivity, this);
        return viewHolder;
    }
```

```
@Override
public void onBindViewHolder(HistoryAdapter.ViewHolder holder,
    int position) {
    holder.txtViewName.setText(itemsData.get(position).getName());
    holder.txtViewDate.setText(itemsData.get(position).getDate());
}

@Override
public int getItemCount() {
    return itemsData.size();
}

public static class ViewHolder extends RecyclerView.ViewHolder {
    TextView txtViewName;
    TextView txtViewDate;

    public ViewHolder(View itemView, final BaseActivity
        baseActivity, final HistoryAdapter adapter) {
        super(itemView);
        txtViewName = (TextView)
            itemView.findViewById(R.id.history_item_name);
        txtViewDate = (TextView)
            itemView.findViewById(R.id.history_item_date);
        itemView.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                baseActivity.replaceFragment(ProductFragment
                 .newInstance(null,adapter.itemsData
                    .get(getAdapterPosition()).getId()));
            }
        });
    }
}
```

3.3 Asynchronous operations

Android has a certain feature: if the main thread is frozen for too long (for example with some loop in code) - Android shows up dialog asking user if he wants to terminate it. To avoid this it is necessary to perform more complex operations in background thread (asynchronously). Android provides a class for this, it is called AsyncTask[13]. *"An asynchronous task is defined by a computation that runs on a background thread and whose result is published on the UI thread. An asynchronous task is defined by 3 generic types, called Params, Progress and Result, and 4 steps, called onPreExecute, doInBackground, onProgressUpdate and onPostExecute."*[14]

I have implemented following asynchronous tasks:

- AsyncTaskComposition,
- AsyncTaskLanguageID,
- AsyncTaskProductFragment,
- AsyncTaskTypeComposition.

All of them inherit from AsyncTask. Below there is a basic example of AsyncTask, which gets preferred language's code from server. It has a constructor for passing Context and method doInBackground, which performs all needed operations.

```
public class AsyncTaskLanguageID extends AsyncTask {

WeakReference<Context> mC;
String newLanguageId;

public AsyncTaskLanguageID(Context mC) {
this.mC = new WeakReference<Context>(mC);
}

@Override
protected Object doInBackground(Object[] params) {

if (mC.get() != null) {
String languageCode = mC.get().getSharedPreferences("preferences",
    Context.MODE_PRIVATE).getString("languageCode", "");

ParseQuery<DbLanguage> query = DbLanguage.getQuery();
query.whereEqualTo("code", languageCode);
List<DbLanguage> objects = null;
try {
objects = query.find();
} catch (ParseException e) {
e.printStackTrace(); }
newLanguageId = objects.get(0).getObjectId();
SharedPreferences.Editor editor =
    mC.get().getSharedPreferences("preferences",
    Context.MODE_PRIVATE).edit();
editor.putString("LanguageId", newLanguageId);
editor.commit();
}

return null;
}
}
```

3.4 Multiple languages

As this application's target user group is not country-specific, application should support at least a few basic languages. For changing language in running application it is necessary to inform resources and update Configuration file (with information about language, orientation and so on). Updating configuration causes restart of application, but this is the process which guarantees that all new resources (strings and layouts) are loaded properly. The code for this looks like this:

```
public static Configuration
    getConfigurationFromPreferences(String language) {
    Locale locale = new Locale(language);
    Locale.setDefault(locale);
    Configuration config = new Configuration();
    config.locale = locale;
    return config;
}
```

So in order to change application's language, it is necessary to call function updateConfiguration on Resources from SettingsFragment, where there is a spinner with available languages:

```
getBaseActivity()
  .getResources()
  .updateConfiguration(Helper
     .getConfigurationFromPreferences(newLanguageCode),
       getBaseActivity().getResources().getDisplayMetrics());
getBaseActivity().recreate();
```

It is also necessary to take care of the language, in which data will be downloaded from server. Current chosen language is stored in SharedPreferences and can be accesses in the following way:

```
getSharedPreferences("preferences",
    MODE_PRIVATE).getString("languageCode");
```

3.5 Database

3.5.1 Local storing

As I used Realm for storing objects locally, they need to inherit from RealmObject. Below there is an example of one of these objects. MbHistory is used for storing user's history of search, so each instance contains ID and name of a product, that user found, and a timestamp.

```
public class MbHistory extends RealmObject {

    private String productName;
    private Date date;
    private String productID;

    public String getProductName() {
        return productName;
    }

    public void setProductName(String productName) {
        this.productName = productName;
    }

    public Date getDate() {
        return date;
    }

    public void setDate(Date date) {
        this.date = date;
    }

    public String getProductID() {
        return productID;
    }

    public void setProductID(String productID) {
        this.productID = productID;
    }
}
```

It is very easy to save a new object in local storage. Function createObject takes care of it and has to be called in a transaction block. In the example below realm is a Realm object and represents the whole local database.

```
realm.beginTransaction();
 MbFavourites favourite = realm.createObject(MbFavourites.class);
 favourite.setProductID(productID);
 favourite.setName(productName);
 realm.commitTransaction();
```

Realm uses very intuitive syntax for retrieving objects from database (for example, functions like **where**, **contains**, **equalTo** and so on). Below there is a simple example of getting results from Realm. This code checks if a product with given ID is saved in favorites or not.

```
        RealmResults<MbFavourites> favourites =
            realm.where(MbFavourites.class).equalTo("productID",
```

```
            productID).findAll();
        if (favourites.size() > 0) {
            isFavourite = true;
        } else {
            isFavourite = false;
        }
```

3.5.2 Storing on server

For storing objects on server I have used Parse. These objects have to inherit from ParseObject. Below you can see that an object has to be mapped to a table in Parse by annotation @ParseClassName(name). It implements getters and setters by calling ParseObject methods getString and put, providing the name of corresponding column in parameters.

```
@ParseClassName("ProductBarcode")
public class DbProductBarcode extends ParseObject {

    public static ParseQuery<DbProductBarcode> getQuery() {
        return ParseQuery.getQuery(DbProductBarcode.class);
    }

    public String getProductID() {
        return getString("productID");
    }

    public void setProductID(String value) {
        this.put("productID", value);
    }

    public String getBarcode() {
        return getString("barcode");
    }

    public void setBarcode(String value) {
        this.put("barcode", value);
    }
}
```

The following code is from AsyncTaskTypeComposition and it retrieves from Parse a list of substance names in the given language.

```
ParseQuery<DbSubstanceName> query = DbSubstanceName.getQuery();
        query.whereEqualTo("languageID", languageID);
        List<DbSubstanceName> objects = null;
        try {
            objects = query.find();
        } catch (ParseException e) {
```

```
        e.printStackTrace();
    }
    if (objects != null && objects.size() > 0) {
        list = new ArrayList<String>();
        for (DbSubstanceName item : objects) {
            list.add(item.getName());
        }
    }
```

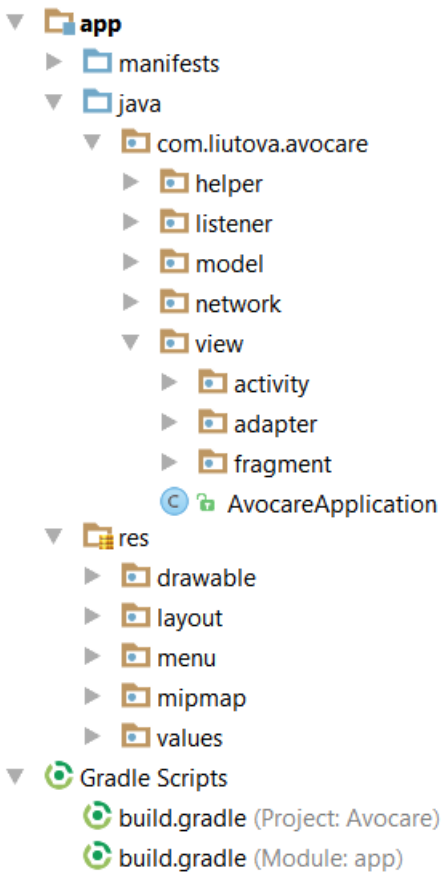


Figure 3.1: Packages structure

Chapter 4

Testing

Quality assurance is a very important part of software development process. Throughout the whole development process some simple tests were performed to evaluate implemented part's performance. And at the end of the project I have performed user tests split into 2 parts: performing operations and questionnaire. I have asked 4 people with different levels of knowledge of Android to perform certain operations with application, describe what issues they faced and then asked them to fill a questionnaire. For testing purposes I have filled database with testing data. It includes 5 products with complete information about them and 64 substances from their compositions.

After performing code review I have asked 4 people with different levels of knowledge of Android to perform certain operations with application, describe what issues they faced and asked them to fill a questionnaire. In following parts I will present the results I have got.

4.1 Performing operations with application

Each of 4 users was given a task to perform following operations:

1. open application,
2. find product by scanning its barcode,
3. find product by name,
4. add product to favorite,
5. access this product from favorites,
6. access this product from history,
7. find information about this composition: glycerin and water,
8. fill a list of allergens with some values.

4.1.1 First user

The first user is an Android developer, so he has a good knowledge of Android operating system. Here is how he coped with given operations:

- Did not have any problems with starting application.
- Scanned barcode without problems.
- Found product by name without problems.
- For adding product to favorites first tried to go to favorites, but then realized that he had to do it on screen with product and handled it in the end.
- Accessed product from favorites without problems.
- Accessed product from history without problems.
- Was a little bit confused as where to find information about composition.
- Filled a list of allergens without problems.

4.1.2 Second user

The second user has been using Android mobile device for less than a year. Here are the results of her performing given operations:

- Did not have any problems with starting application.
- Scanned barcode without problems.
- Had problems with finding product by its name: did not click on tips displayed by application.
- Added product to favorites without problems.
- Accessed product from favorites without problems.
- Accessed product from history without problems.
- While looking for information about given composition entered the whole composition into one EditText item, did not see the plus sign for adding new substances.
- Filled a list of allergens without problems.

4.1.3 Third user

The third user has been using Android mobile devices for about 3 years. Here is how he performed:

- Did not have any problems with starting application.
- Scanned barcode without problems.
- Found product by name without problems.
- Took a while to find how to add product to favorites, but managed it.
- Accessed product from favorites without problems.
- Accessed product from history without problems.
- Took some time to find where to enter composition, but then found the right button on home screen.
- Filled a list of allergens without problems.

4.1.4 Fourth user

The fourth user has 2-years experience with using Android mobile devices. Here are the results of his performing given operations:

- Did not have any problems with starting application.
- Scanned barcode without problems.
- Found product by name without problems.
- Added product to favorites without problems.
- Had problems accessing product from favorites: did not realize there was a drawer in application.
- Accessed product from history without problems after being shown the drawer.
- Took a while to find where to enter composition.
- Filled a list of allergens without problems.

4.2 Questionnaire

1. How would you rate intuitiveness of user interface design on scale from 1 to 10, 10 being the most intuitive?
   - *User 1* I would say 8. There were some misconceptions, but in general it was ok.
   - *User 2* 10, I liked it.
   - *User 3* 7, a few things were not very intuitive, but in general it was fine.
   - *User 4* 9, it was pretty good.
2. Would you consider using this application?
   - *User 1* If the database gets larger, then yes.
   - *User 2* Definitely yes.
   - *User 3* Yes.
   - *User 4* If the database gets filled, definitely yes.
3. What would you improve in this application?
   - *User 1* Speed.
   - *User 2* Searching product by name is confusing, because it's often hard to say what the name actually is.
   - *User 3* The speed of search can be improved, and number of products in database.
   - *User 4* Nothing comes to mind.

4.3 Summary

Even though the testing group was pretty small, it is possible to make certain conclusions about developed application prototype based on their results. All of users handled most operations well, though each of them had certain issues and suggestions.

Confusion as where to find certain button or how to perform input This issue can be solved with a quick guide on the first launch of application.

Low speed of searching This can be resolved by moving some operations on server side.

Incompleteness of database For now the database is filled with few data for testing purposes. In the future it will be necessary to upload sufficient amount of data in order for application to be usable.

Chapter 5

Future improvements

5.1 Data

For this Bachelor thesis I have implemented a prototype of Android application. That means, that application supports the main required functionality. There is a big issue, however, that stands on the way of using this application - and that is lack of data. It is necessary to fill created database with sufficient amount of real data in order for it to be ready for usage. This requires certain investments, and if it is done in the future, the application can be uploaded to Google Play, becoming publicly available.

5.2 Improving speed

Testing phase showed that the speed of search is not sufficient. In the future this issue can be fixed by implementing part of code on server side. For now the server is a simple objective database, and it's necessary to perform a few server calls in order to gain needed information. Each server call is time-consuming, and this significantly slows down the whole search process. Exporting these operations to server will improve the speed considerably.

5.3 First launch guide

As testing phase showed, some of the application functions can be confusing, that's why it's a good idea to implement a simple user guide for first application launch. This can be done by creating a Showcase View.

5.4 Products comparison

One possible useful improvement could be comparing a few products among themselves. When a user is in a situation, when he wants to choose one

product, that would be most suitable for him, it would be useful to have comparison of chosen candidates displayed on his mobile device.

5.5 More available languages

For now application supports only 2 languages: English and Czech. However, its architecture is designed in such a way that it can support any number of languages. That is why adding more languages requires minimal change in the application. It is important to note, that adding new languages does not guarantee covering new markets - the key issue here is data. It is country-specific: products, barcodes, substances' names, descriptions and so on. And that means, that if we want to make application available in a certain region, it is necessary not only to provide application interface in this region's language, but also to upload necessary data to server.

5.6 Users' evaluation of products

Another nice feature could be user's evaluation of product's characteristics like effectiveness, corresponding price, smell or comfortableness of usage. For each product user will see his own evaluation and average evaluation. Average evaluation can significantly help users to choose good products matching their taste.

Conclusion

The aim of this Bachelor thesis was to create a prototype of an application for analysis of cosmetic products composition, that would fulfill desired functionality. This aim has been accomplished. The analysis of similar applications and requirements analysis have been conducted. Subsequently, application design has been created based on established requirements. After design phase, a prototype with basic application functionality was implemented. This prototype has been properly tested. All of functional and nonfunctional requirements have been fulfilled. However, the application is not yet ready for public usage, mainly because the database is not yet filled with valid data.

Bibliography

[1] *EWG's Healthy Living* [online]. EWG's Healthy Living, 2016 [viewed. 2016-05-06]. Available at: https://play.google.com/store/apps/details?id=com.skindeep.mobile

[2] *GoodGuide* [online]. GoodGuide, 2012 [viewed. 2016-05-06]. Available at: https://play.google.com/store/apps/details?id=com.goodguide.app

[3] *Think Dirty* [online]. Think Dirty, 2016 [viewed. 2016-05-06]. Available at: http://www.thinkdirtyapp.com/

[4] *Smartphone OS Market Share, 2015 Q2* [online]. Framingham: IDC Research Inc., 2015 [viewed. 2016-01-31]. Available at: http://www.idc.com/prodserv/smartphone-os-market-share.jsp.

[5] *Parse* [online]. Parse, 2016 [viewed. 2016-04-29]. Available at: https://parse.com/docs/android/guide

[6] *Realm* [online]. Realm, 2016 [viewed. 2016-04-29]. Available at: https://realm.io/docs/java/latest/

[7] WHARTON, Jake. *Butter Knife - Field and method binding for Android views* [online]. Jake Wharton, 2013 [viewed. 2016-2-2]. Available at: http://jakewharton.github.io/butterknife/.

[8] *Picasso - A powerful image downloading and caching library for Android* [online]. Square, Inc. 2013 [viewed. 2016-2-2]. Available at: http://square.github.io/picasso/.

[9] *Support Library* [online]. Google Inc. 2016 [viewed. 2016-04-29]. Available at: http://http://http://developer.android.com/tools/support-library/index.html

[10] *Support Library Features* [online]. Google Inc. 2016 [viewed. 2016-04-29]. Available at: http://developer.android.com/tools/support-library/features.html

[11] *Creating Lists and Cards* [online]. Google Inc. 2016 [viewed. 2016-04-29]. Available at: http://developer.android.com/training/material/lists-cards.html

[12] GUPT, Mohit*Android RecyclerView vs ListView — Comparison* [online]. Truiton, 2015 [viewed. 2016-04-29]. Available at: http://www.truiton.com/2015/03/android-recyclerview-vs-listview-comparison/

[13] *Processes and Threads* [online]. University of Southern California, 2009 [viewed. 2016-05-06]. Available at: http://developer.android.com/guide/components/processes-and-threads.html

[14] *AsyncTask* [online]. Google Inc. 2016 [viewed. 2016-05-06]. Available at: http://developer.android.com/android/os/AsyncTask.html

[15] *Model Description* [online]. Google Inc. 2016 [viewed. 2016-05-06]. Available at: http://csse.usc.edu/csse/research/CORADMO/

Appendix A

Application's screenshots

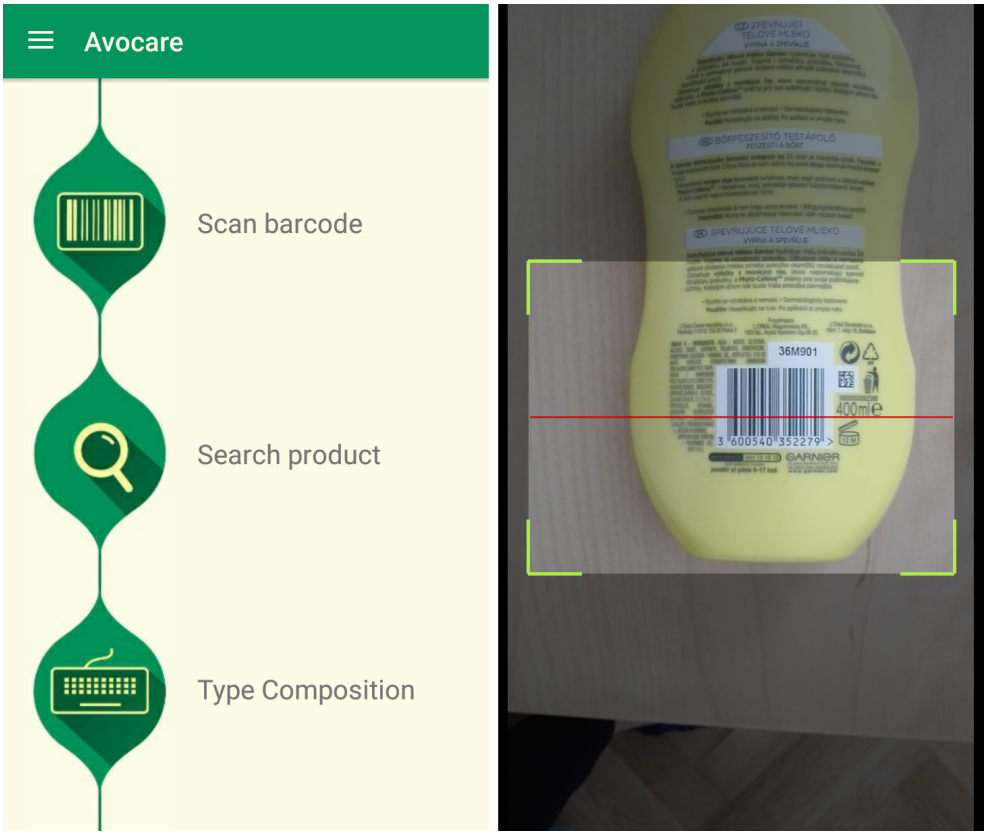


Figure A.1: Screenshots of home screen and barcode scan screen

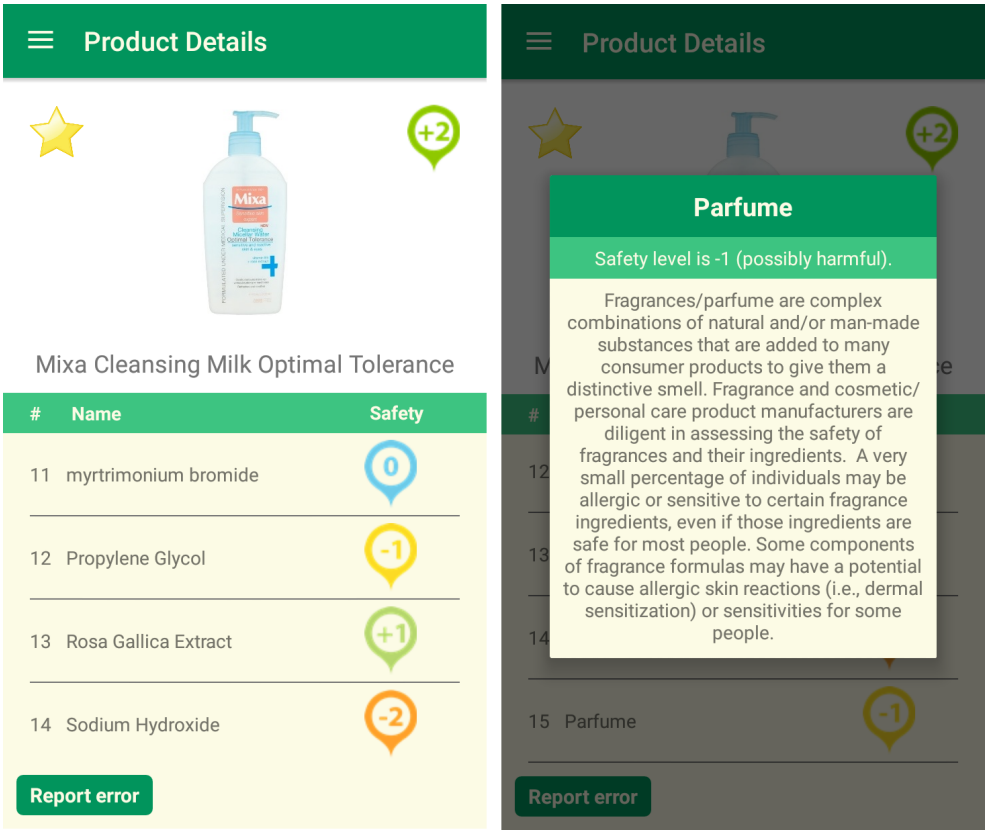


Figure A.2: Screenshots of product screen and substance description dialogue

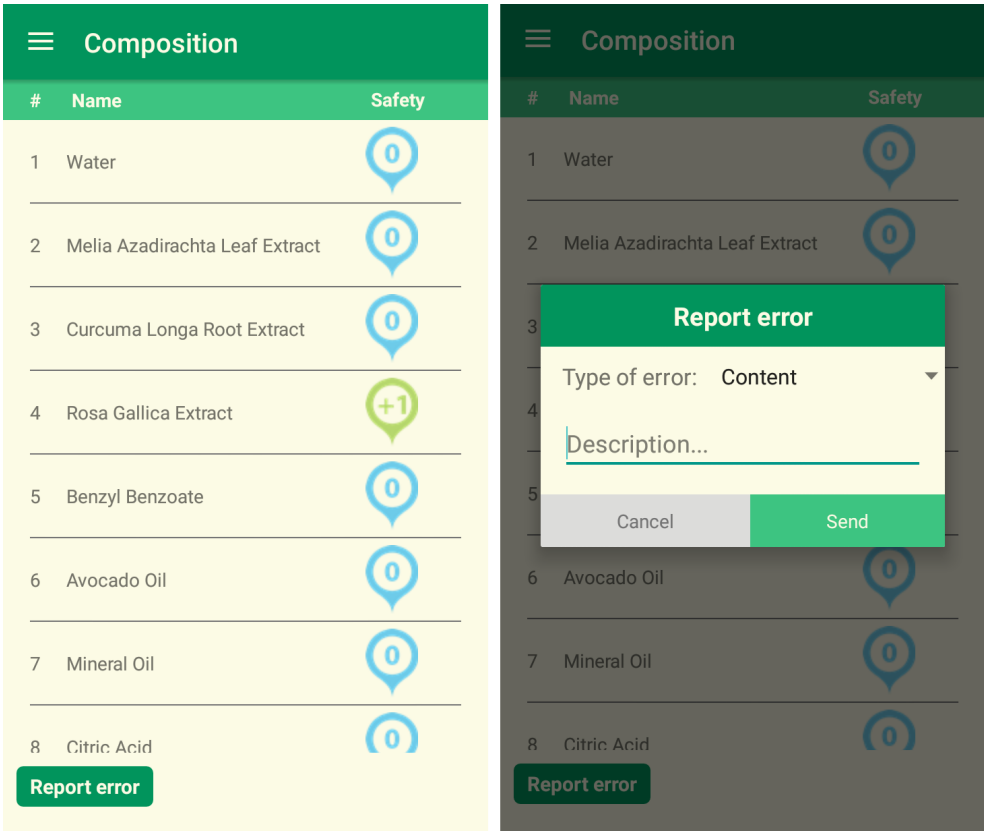


Figure A.3: Screenshots of composition screen and report error dialogue

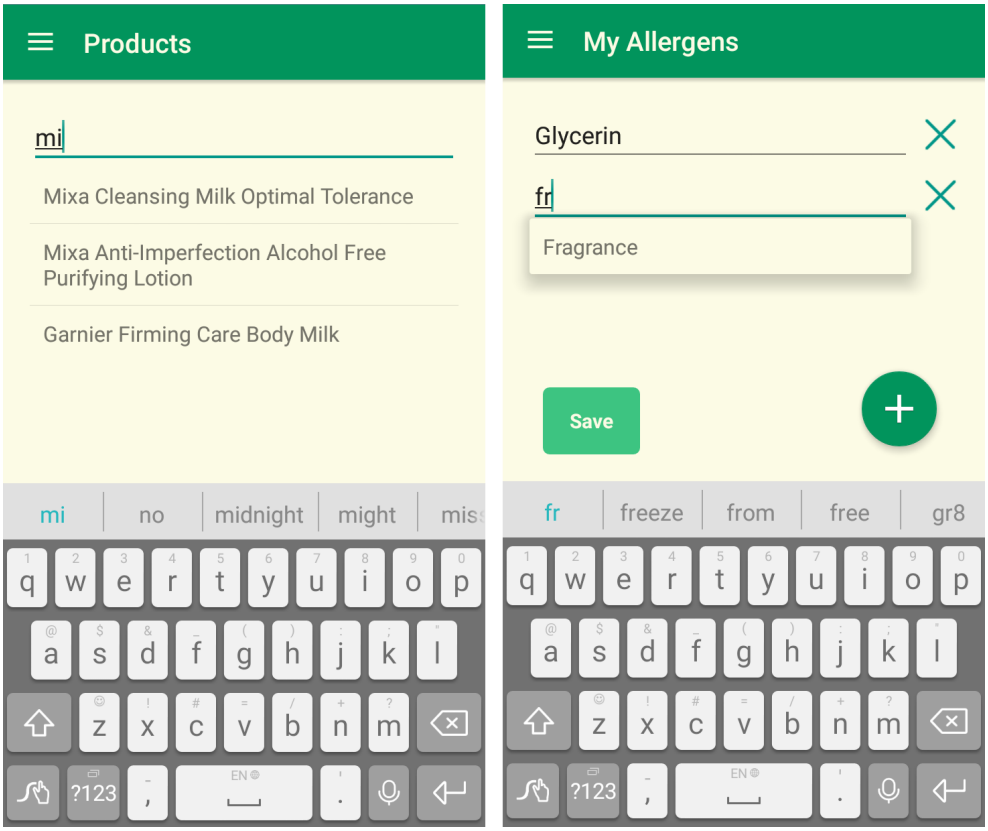


Figure A.4: Screenshots of product search screen and allergens screen

Appendix B

Material for testing

Figure B.1: Barcodes of products present in database

Appendix C

Contents of enclosed CD

- `readme.txt` ..... the file with CD contents description
- `apk` ..... the directory with executable apk file
- `src` ..... the directory of source codes
  - `implem` ..... implementation sources
  - `thesis` ..... the directory of LaTeX source codes of the thesis
- `text` ..... the thesis text directory
  - `thesis.pdf` ..... the thesis text in PDF format