BOOKOPEDIA

MOBILE APPLICATIONS DEVELOPMENT 2



 | SOFTWARE SYSTEMS DEVELOPMENT |

Name: Tsvetoslav Dimov

ID: 20077038

Lecturer: David Drohan

Date: 12/04/2019

Table of Contents

[Introduction 2](#_Toc6094293)

[Overview 2](#_Toc6094294)

[UML Diagrams 3](#_Toc6094295)

[Activities 3](#_Toc6094296)

[Fragments 4](#_Toc6094297)

[Models 5](#_Toc6094298)

[Firebase Database schema 6](#_Toc6094299)

[Features 7](#_Toc6094300)

[User Registration & Login 7](#_Toc6094301)

[Searching for books 8](#_Toc6094302)

[Wish List 9](#_Toc6094303)

[Book info display 10](#_Toc6094304)

[Persistence approach adopted 11](#_Toc6094305)

[Authentication 11](#_Toc6094306)

[Real-time Database 11](#_Toc6094307)

[Git approach adopted 12](#_Toc6094308)

[UX / DX approach adopted 12](#_Toc6094309)

[References 13](#_Toc6094310)

# Introduction

The purpose of this document is to provide a detailed description of the Android application “Bookopedia”. It provides insight into the design and structure of each activity/fragment. After reading this document, the user will be aware of the functionality of this application and will be able to use it without any problem.

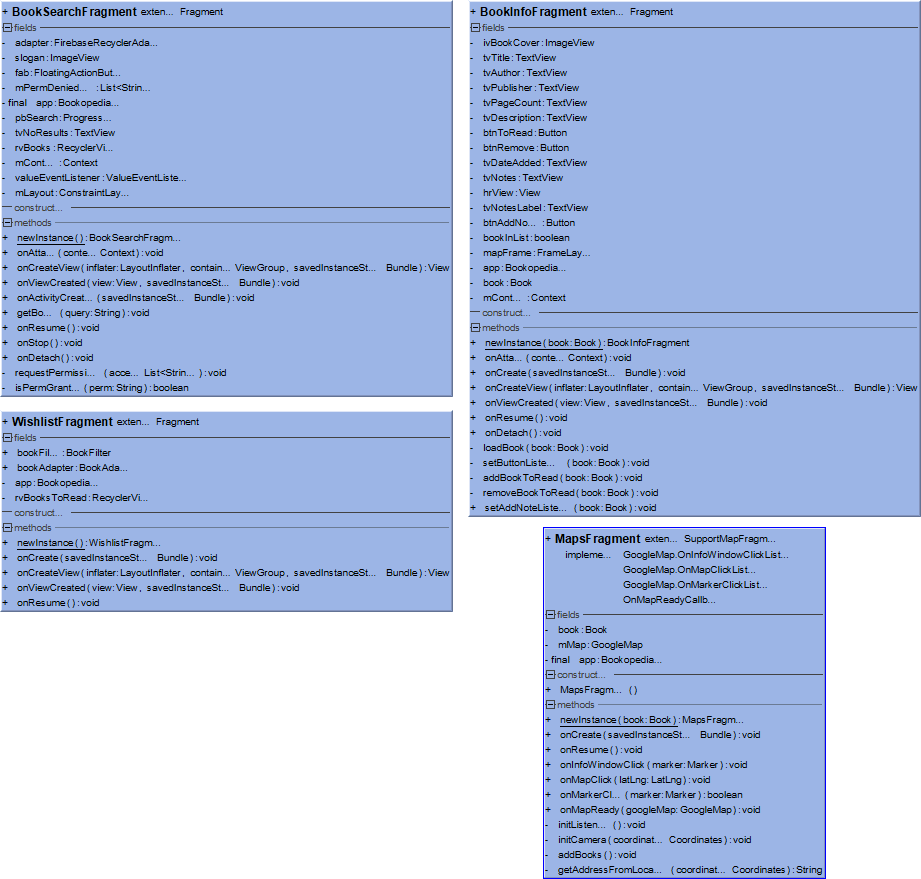
# Overview

Bookopedia is an Android application that provides the user with the ability to save books in custom lists, such as Wish list. Before being able to save books, login is required. The user can create a new account with their email address or use the Google sign-in. The application will automatically login the last user account, if they did not log out in their last session. After logging in, the user is presented with home screen, which will enable him/her to search for books using the Google Books API, as well as a barcode scanner button, which can be used to for quick search for a specific book. If the search returned any results, the user can select a book and view additional information about it, such as: description, number of pages, publisher. When a book is added to the Wish list, the following extra data is stored about the book: the date and time of submission, user location and personal notes. The app supports multiple users and stores books as collections with the unique ID of the user. If the user has added books to their Wish list, they can view the contents of their collection by navigating to the Wish list from the navigation drawer. There they can delete books by swiping left or by long clicking (this activates Action mode), then the user can select books and delete more than one book at a time. The user can navigate through the app using the navigation drawer. This app is constructed based on a single activity and many fragments, which are displayed in a placeholder fragment layout and interchanged based on user selection.

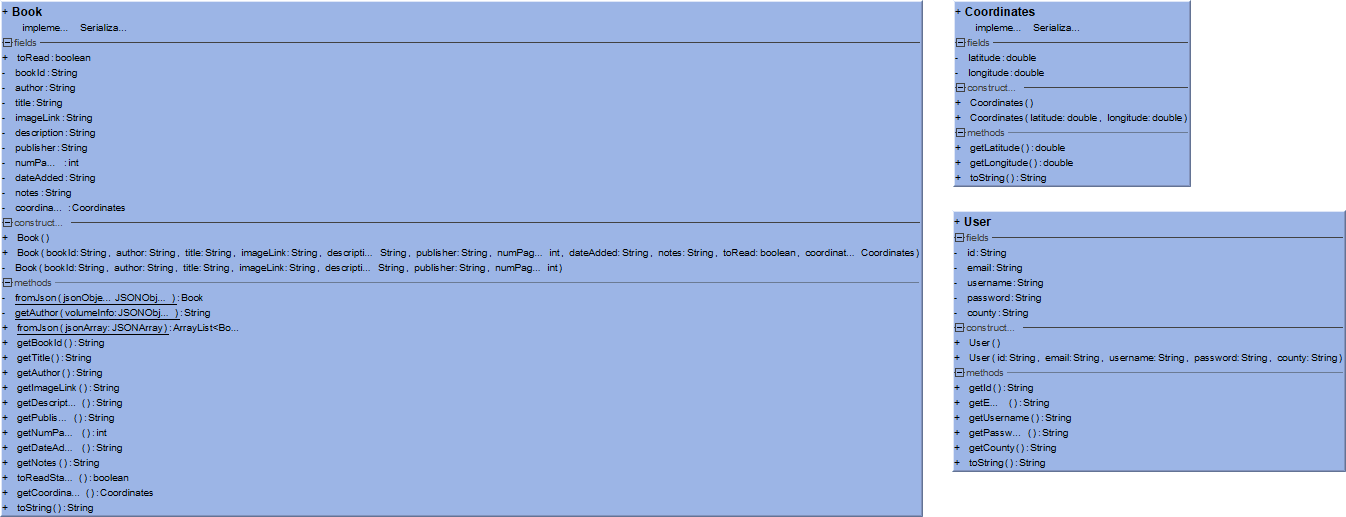
# UML Diagrams

## Activities

## Fragments



## Models



# Firebase Database schema



# Features

## User Registration & Login

Authentication is required for the usage of this application. The first time it is started, the user is presented with the Login screen. They can register by clicking the “SIGN ME UP” button, which will open the registration form, where they have to enter email, username, password and county. After filling out the registration form, the user is automatically logged in and redirected to Home activity.

If the user has an account, they can simply enter their email and password. By entering their credentials and pressing the “LOG IN” button, a search for match is being processed with ***Firebase authentication***. If the credentials match, user is taken to the Main activity, otherwise an error message pops up in a Toast.

Another way of authentication is using ***Google sign-in***. A much quicker, simplified and handy way to log in. In this way, linking the user’s Google account (if he has one) to the application. When they click on “LOG IN WITH GOOGLE”, a dialog where their Google email or phone number is needed, pops up. After entering these details, the application takes a few seconds to connect and then automatically redirects to Main activity. Multiple accounts can be stored, which makes for quick login in just one click, if multiple users are using the same device.

When the application is killed, there are two possible scenarios:

1. The user did not sign out, which means they will be automatically logged in the next time they start the application.
2. The user signed out, which means the next time they start the application, they will have to enter their credentials in order to use the application.

## Searching for books

The way searching works is the following: there is a search icon located in the action bar of the app. When the user clicks on it, a SearchView expands and there the user can enter their query. This app is using ***Google Books API*** to search for books. To ensure user experience is satisfying, when the user enters a search query and stops typing for 2 seconds, automatic search fires, in this way eliminating the need for the user to click on search button.

To send HTTP requests to the Google Books API, I am using a library called ***AsyncHTTP.*** The response we get from the API is JSON array. I then parse it into an array of Book objects, which are then stored on ***Firebase Database.*** If the query does not return any results, a message shows up to let the user know what happened.

Another way of searching for books is using the barcode scanner. This is possible by using the library called ***ZXing ("Zebra Crossing")***. To use that functionality, the user has to navigate to the Home screen of the app and click on the Floating action button with a graphic of magnifying glass looking at a barcode. Next thing, the app requests permissions to use the camera of the device and proceeds to open the camera app. After that, the user can scan a nearby book. The library extracts the ISBN number from the barcode and sends a request to the Google Books API using that ISBN number as search parameter. The results show in the RecyclerView, in the same way as they do with manual searching using the search icon from the Action bar.

To display the search results data I am using a RecyclerView with a ***FirebaseUI*** adapter. It functions in the same way as a normal adapter, but you can feed it with a Firebase query as a parameter and it will display the data pulled from the cloud. The results are persisted in Firebase to make sure that if the user exits the app and comes back to it the next day, their data will still be there. Book covers are loaded into ImageViews using the library ***Picasso***.

The user is able to select a book by clicking on the relevant row of the RecyclerView. By doing that, a new fragment loads, which displays detailed information about the book (title, author/s, publisher, page count, description).

## Wish List

The Wish list is a personal collection of books, which each user can create. The book data is persisted in a Firebase Database and pulled into a local ArrayList for handling and manipulation. Each user’s collection is stored in a collection, which primary key is their unique user ID. The user can navigate to their Wish list by opening the navigation drawer and selecting “Wish List”.

The data is displayed in a RecyclerView. The user is able to select a book by clicking on the relevant row of the RecyclerView. Similar to viewing a book that the user found through the search, the layout will show all the fields previously mentioned, with the addition of date/time of book addition to the list, notes that the user can customise to their liking and a Google map that shows the exact location where the book was added.

The user can ***filter*** through their books by using the search icon from the Action bar. When the user enters characters into the SearchView, the filtering is performed dynamically without the need of submitting the search query. All the filtering does is it takes the search string the user enters and checks if any of the book titles contain it. If there is match/es, the results are added to a new ArrayList and displayed to the user. If there isn’t a match, the RecyclerView will be empty.

Deletion of books is possible by long clicking on a RecyclerView item. This activates Action mode and the clicked item is marked as selected. The user can mark as many items as they want. Then they can click on the bin icon button and the books will get deleted from the database. Another way of deleting a book is by swiping left a RecyclerView item. This deletes the book from the database, shows a Snackbar that lets the user know the book is deleted and offers an action to undo the deletion. To make that possible I have created a custom callback for ***ItemTouchHelper.***

To make the user interface more attractive, each book item in the RecyclerView is wrapped in a ***CardView.***

## Book info display

When the user searches for books, the results are shown in a RecyclerView. Each RecyclerView item corresponds to a Book object. If the user clicks on any of the rows, they will be taken to a fragment, which displays detailed book information. All the fields are:

1. Book cover
2. Title
3. Author/s
4. Date added (if viewed through Wish list)
5. Publisher
6. Page count
7. Description
8. Notes (if viewed through Wish list)
9. Map showing the location where the book was added (if viewed through Wish list)

As stated, some details are hidden if the user is looking at a book that is not in their wish list, because they do not exist yet. If a book is added to the wish list of a user, new data is stored about the book (date added, notes and location). To get the current location of the user, I am using a library called ***AirLocation***. I request user permission to access their location and then proceed with fetching the latitude and longitude, after that I store them as a Coordinates object, add it as a field to the Book object and push it to Firebase Database.

The coordinates of the book are used as parameters for the ***Google Maps*** fragment that is displayed in each Book info page that is part of a Wish list. Using a ***custom marker*** for the map, it shows the location where the book was added to the list, as well as all other books that the user has in their wish list.

The user can also add notes for each book in their wish list. By default, the notes are empty. To add notes, the user has to navigate to a book and click on the button “ADD NOTES” at the very bottom of fragment. After saving the note, it will be displayed whenever the user opens that book.

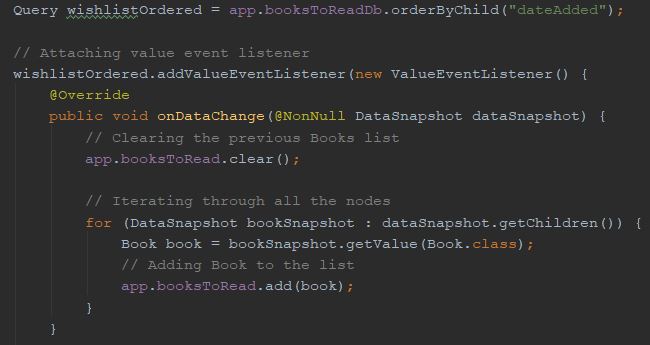
# Persistence approach adopted

Data is persisted in a cloud database – Firebase. It offers wide variety of useful services. In Bookopedia are utilised the following Firebase services:

## Authentication

This application offers two ways of authenticating the user. The first one is by email & password. A database with users is kept on Firebase server and every time someone tries to login, they have to enter valid credentials and be connected to the internet. The second way of authentication is by Google sign-in, where the user can link their Google account to the application for even easier and quicker login. Using Firebase Authentication makes building secure auth systems easier, while also improving the user experience.

## Real-time Database

When the user publishes an advert, data is uploaded to cloud database. It is stored in JSON format and can be viewed and managed in real-time from the Firebase console. When the application is started, it checks the database for changes and loads the most recent database version to show to the user:

# Git approach adopted

I committed to GitHub on a regular basis. Commits were done whenever I fixed bugs or introduced new features to the application. In my opinion, commits are very important. In a situation where problems occur, sometimes reverting to a previous commit is an easy fix and a way for the developer to find and fix their mistakes. For the duration of developing this application, on numerous occasions I had to revert to previous commit. When I was working on new features, I created new branches, e.g. implementing navigation drawer.

Besides that, GitHub repository that is committed regularly to is a secure way to backup project.

# UX / DX approach adopted

User experience was enhanced through the use of navigation drawer and app bar menu for easy navigation. On the login/registration screen, all the fields are validated (e.g. to check if the email address was in the correct format or if a user with that email exists). Whenever something is loading, e.g. searching for books or logging in, a ProgressBar shows up and the background blurs until the loading is finished.

On developer experience side, I made sure everything is neatly organised and easy to follow. I utilised MVC (Model/View/Controller) design pattern. They are all separated into their own packages. Code is thoroughly commented and formatted. Whenever the user makes an API query, I have limited the number of API calls to be sent only after 2 seconds have passed since last character was entered, instead of firing an API call after every character entered, which is an overkill. In the first part of the assignment, I used activities, but for the second part I redesigned my app structure and changed all activities to fragments, they are a lot more flexible and quicker than activities.

# References

Flaticon. (2019). *Flaticon, the largest database of free vector icons*. [online] Available at: https://www.flaticon.com/ [Accessed 13 Apr. 2019].

GitHub. (2019). *hdodenhof/CircleImageView*. [online] Available at: https://github.com/hdodenhof/CircleImageView [Accessed 13 Apr. 2019].

GitHub. (2019). *mumayank/AirLocation*. [online] Available at: https://github.com/mumayank/AirLocation [Accessed 13 Apr. 2019].

GitHub. (2019). *zxing/zxing*. [online] Available at: https://github.com/zxing/zxing [Accessed 13 Apr. 2019].

Guides.codepath.com. (2019). *Fragment Navigation Drawer | CodePath Android Cliffnotes*. [online] Available at: https://guides.codepath.com/android/Fragment-Navigation-Drawer [Accessed 13 Apr. 2019].

Medium. (2019). *Step by Step: RecyclerView Swipe to Delete and Undo*. [online] Available at: https://medium.com/@zackcosborn/step-by-step-recyclerview-swipe-to-delete-and-undo-7bbae1fce27e [Accessed 13 Apr. 2019].

N., Kadam, H. and Tyagi, N. (2019). *Doesn't Picasso support to download images which uses https protocol*. [online] Stack Overflow. Available at: https://stackoverflow.com/questions/23562794/doesnt-picasso-support-to-download-images-which-uses-https-protocol [Accessed 13 Apr. 2019].