INFO2051-G04
Object-oriented programming on mobile devices

The Beer App

by
Dimitra Paranou
Maxime Thomassin
Christophe Duchesne

Liège University
May 8th, 2018

The Beer App **2**
Application Design 3
Application overview 3
How activities are linked and related 5
Object-Oriented Programming 7
Encountered Problems 11
Fixed problems 11
Known bugs 12
Future Features 13
External APIs 14
Google Map Api 14
Google GeoCode Api 14
Google Mobile Vision Api 15
Conclusion 16

The Beer App

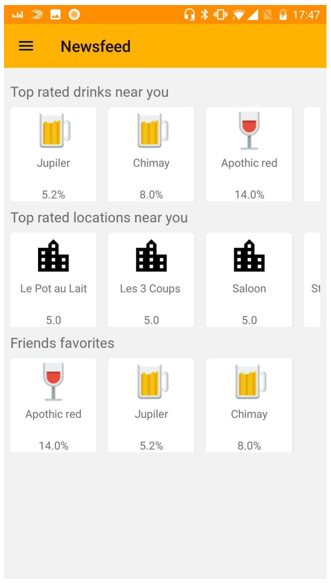
The application discussed in the present document, The Beer App, is an android application destined to give users the possibility to easily find and discover alcoholic beverages and their selling points. The application is community-based, so ultimately every person using it would contribute to it by adding new or existing drinks and locations, exponentially populating the shared database to contain precise and complete entities all around the world. The Beer App will recommend drinks and establishments based on the user's position, and the social feature enables users to connect together by being friends and then contribute to their recommendations by showing the friends' favorites drinks. A map feature allows the user to have a quick visual overview of locations around him, and a simple click on one of them shows him some of the location's top rated drinks. The purpose of the app and the discovery and research mechanism are therefore very intuitive, simple and fast to use.

BeerApp on the Google Play Store : https://play.google.com/store/apps/details?id=view.beerapp
BeerApp Privacy Policies and Terms and Conditions : https://beerapp-6eeab.firebaseapp.com/

Application Design

Application overview

When logging in, the user is sent to the DashboardActivity, this activity shows him the best drinks and locations around him. A click to any of the square will display more information of that location or drink. The MapsActivity is part of the core feature of the application since it provides the user the things to do around him. Each markers can be selected to have a quick overview of that location.



17:47

Newsfeed

Top rated drinks near you

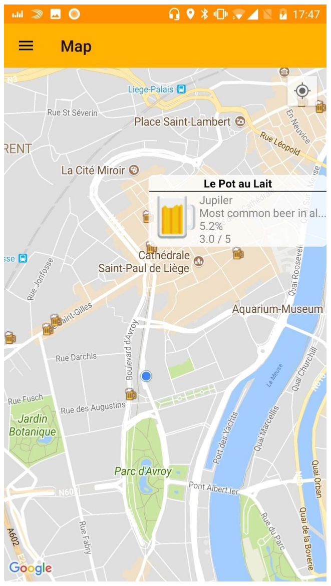
Jupiler	Chimay	Apothic red
5.2%	8.0%	14.0%

Top rated locations near you

Le Pot au Lait	Les 3 Coups	Saloon
5.0	5.0	5.0

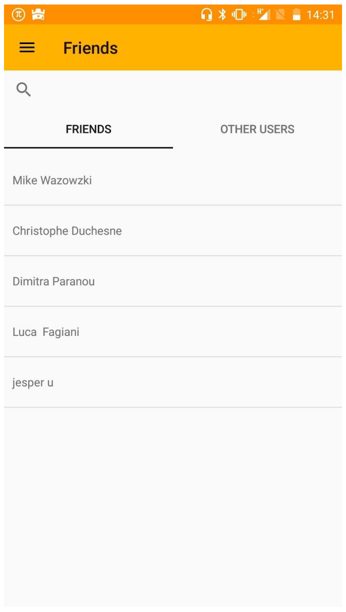
Friends favorites

Apothic red	Jupiler	Chimay
14.0%	5.2%	8.0%



Le Pot au Lait
Jupiler
Most common beer in al...
5.2%
3.0 / 5

Moreover, the users have the possibilities to add each other as friends. This allow them to share their favorite drinks and develop a network of friends that share the same passion.



14:31

Friends

FRIENDS OTHER USERS

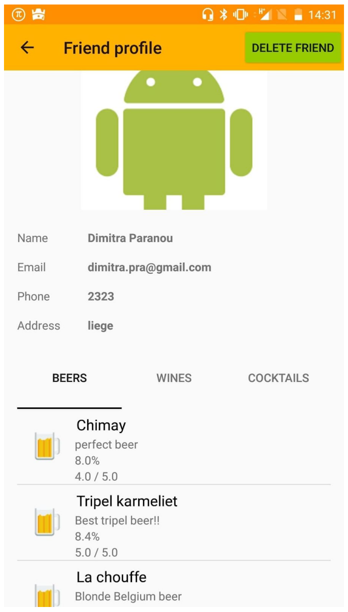
Mike Wazowzki

Christophe Duchesne

Dimitra Paranou

Luca Fagiani

jesper u



14:31

Friend profile DELETE FRIEND

Name	Dimitra Paranou
Email	dimitra.pra@gmail.com
Phone	2323
Address	liege

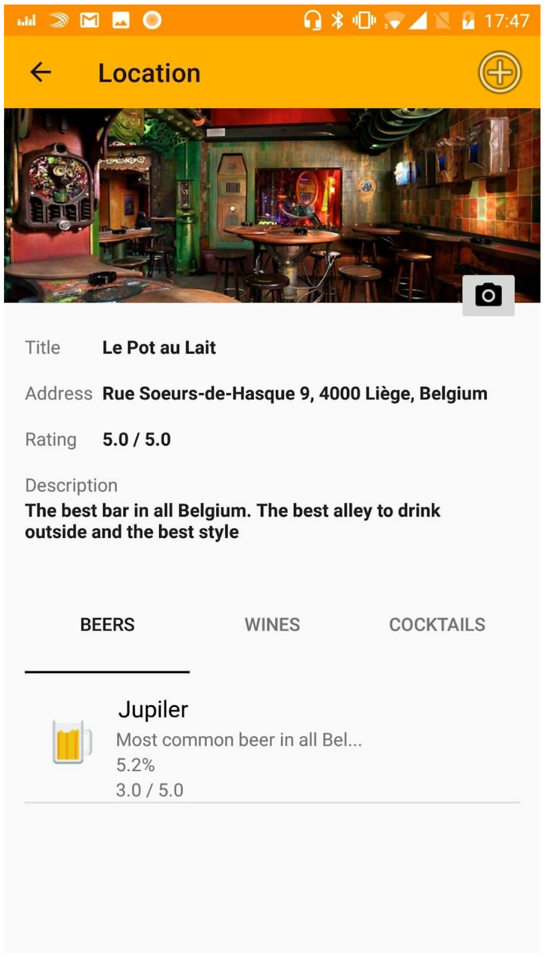
BEERS WINES COCKTAILS

Chimay
perfect beer
8.0%
4.0 / 5.0

Tripel karmeliet
Best tripel beer!!
8.4%
5.0 / 5.0

La chouffe
Blonde Belgium beer

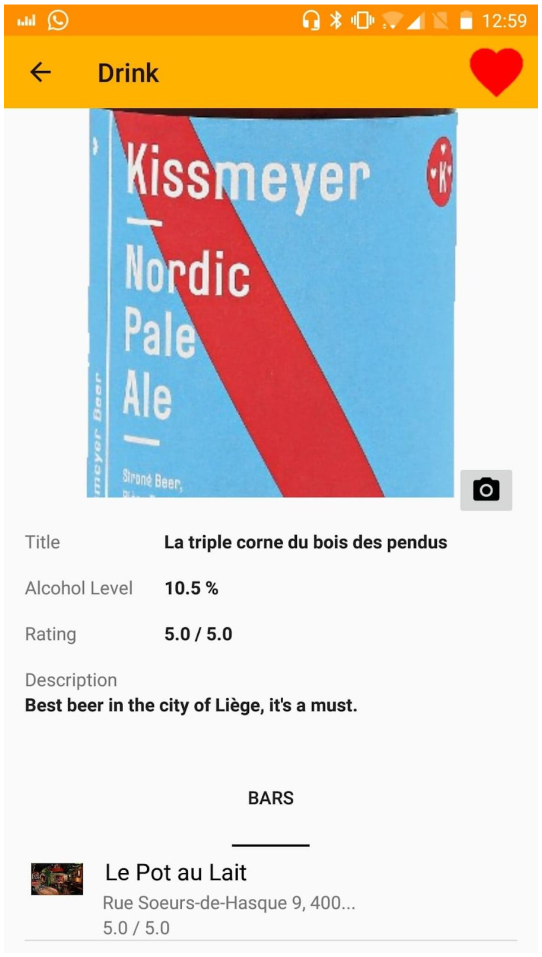
Most of the activities lead to the LocationDisplayActivity and DrinkDisplayActivity. With the MapsActivity, those two are part of the core features of the application. Since they have a many to many relationship, they can access in each other as shown on those screenshots :



Title	Le Pot au Lait
Address	Rue Soeurs-de-Hasque 9, 4000 Liège, Belgium
Rating	5.0 / 5.0

Description
The best bar in all Belgium. The best alley to drink outside and the best style

Jupiler
Most common beer in all Bel...
5.2%
3.0 / 5.0



Title	La triple corne du bois des pendus
Alcohol Level	10.5 %
Rating	5.0 / 5.0

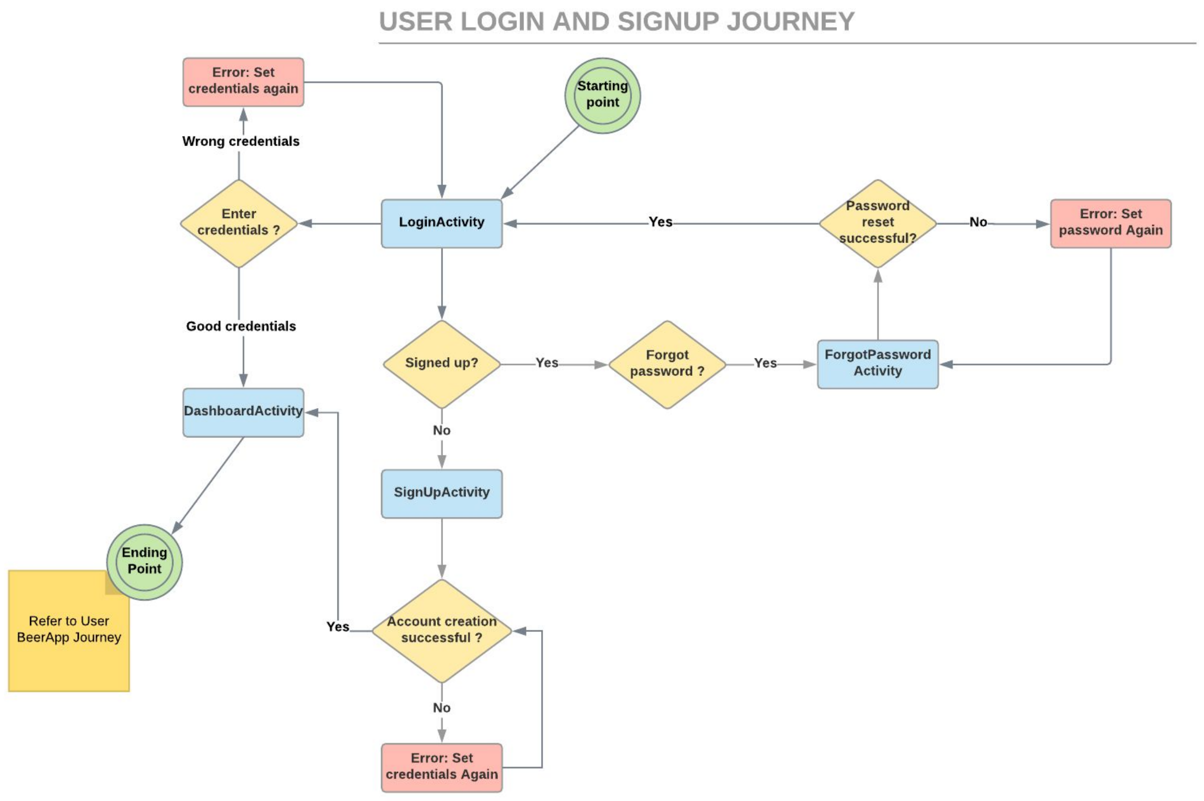
Description
Best beer in the city of Liège, it's a must.

Le Pot au Lait
Rue Soeurs-de-Hasque 9, 400...
5.0 / 5.0

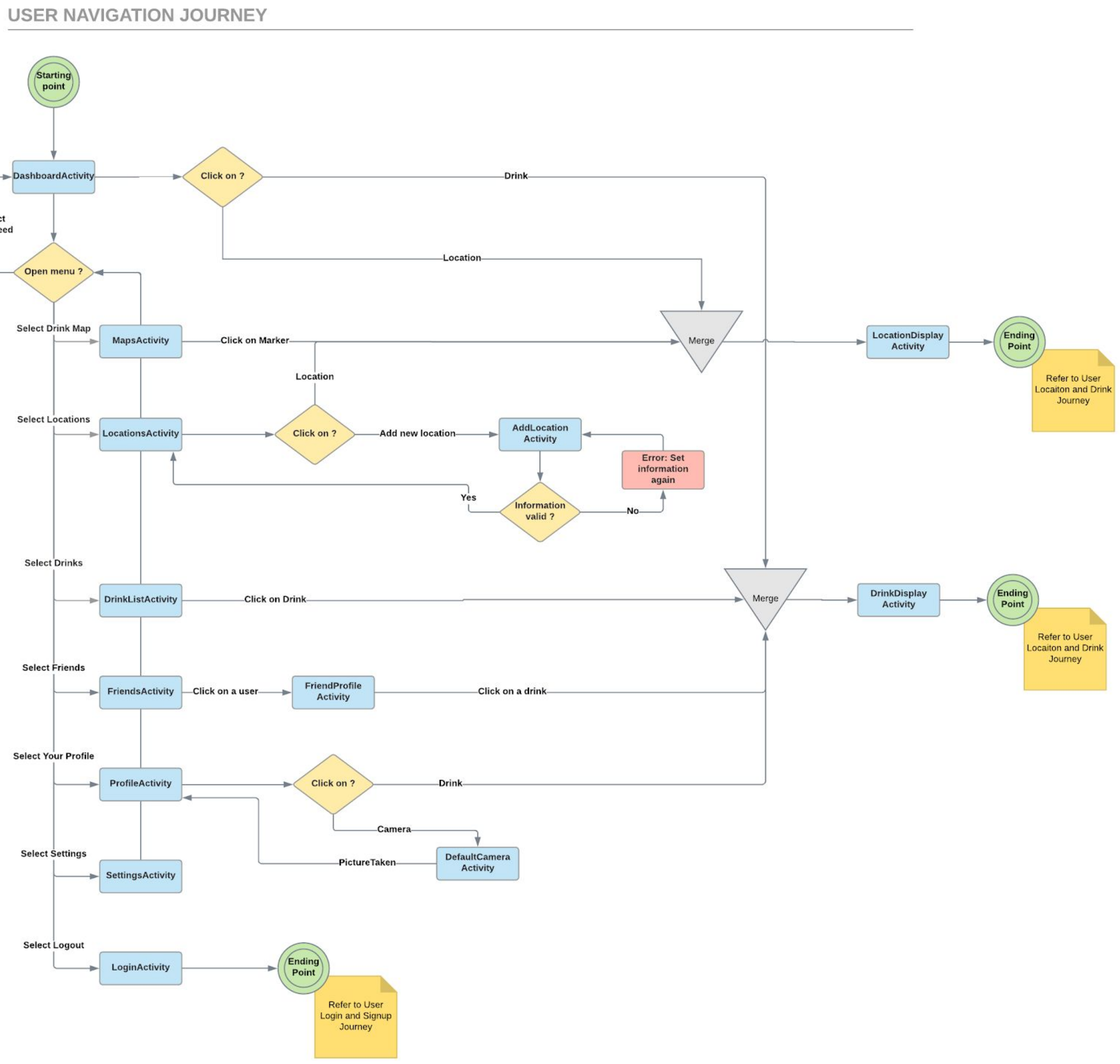
How activities are linked and related

The three following diagrams represent how a user can launch all the different activities. The goal is to see how each activities are related and linked and not to show every possible action.

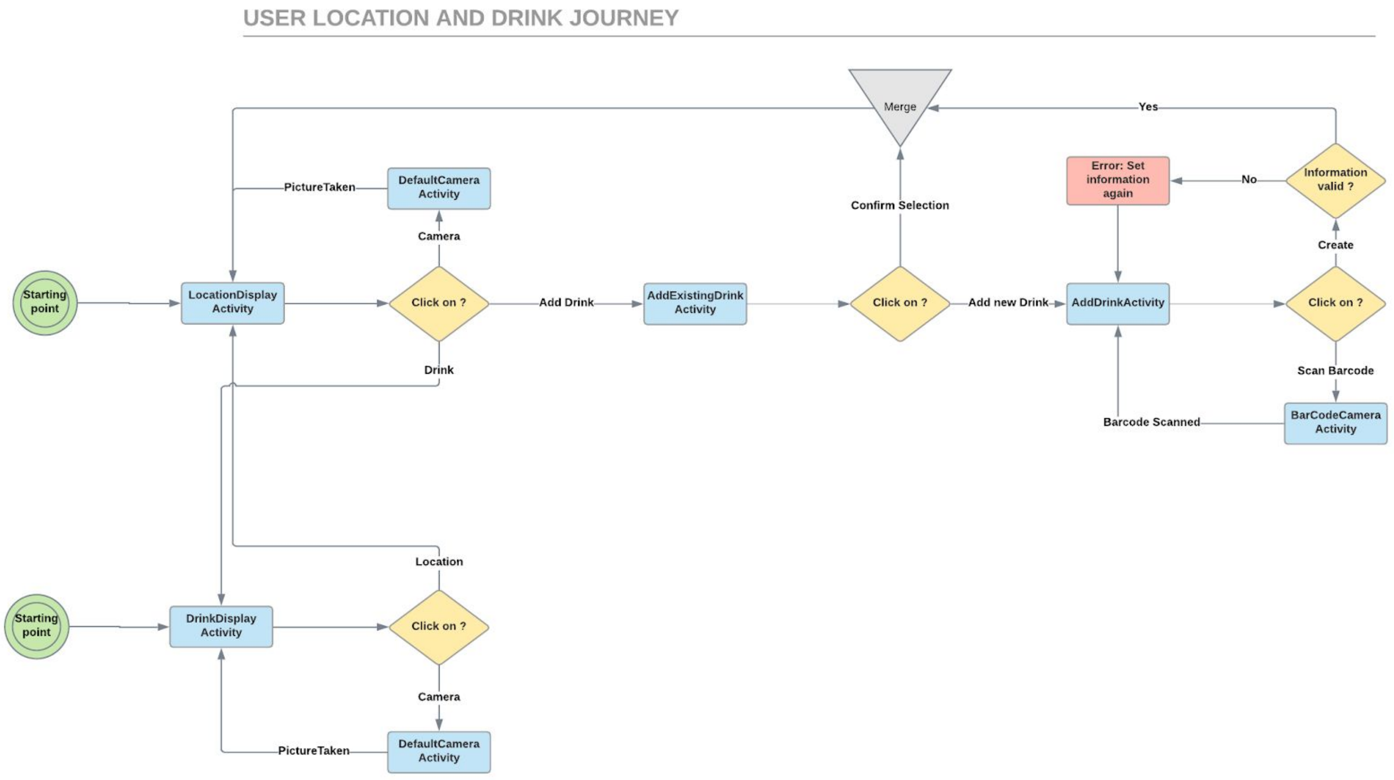
The user login and signup journey represent the authentication and signup process. Once a user connects to the application, each time he opens the app again, he will go directly to the navigation journey until he logs out.


USER LOGIN AND SIGNUP JOURNEY
Error: Set credentials again
Starting point
Wrong credentials
Enter credentials ?
LoginActivity
Yes
Password reset successful?
No
Error: Set password Again
Good credentials
Signed up?
Yes
Forgot password ?
Yes
ForgotPassword Activity
DashboardActivity
No
SignUpActivity
Ending Point
Refer to User BeerApp Journey
Yes
Account creation successful ?
No
Error: Set credentials Again

The navigation journey represent the activities that can be launched from the menu drawer of the application.

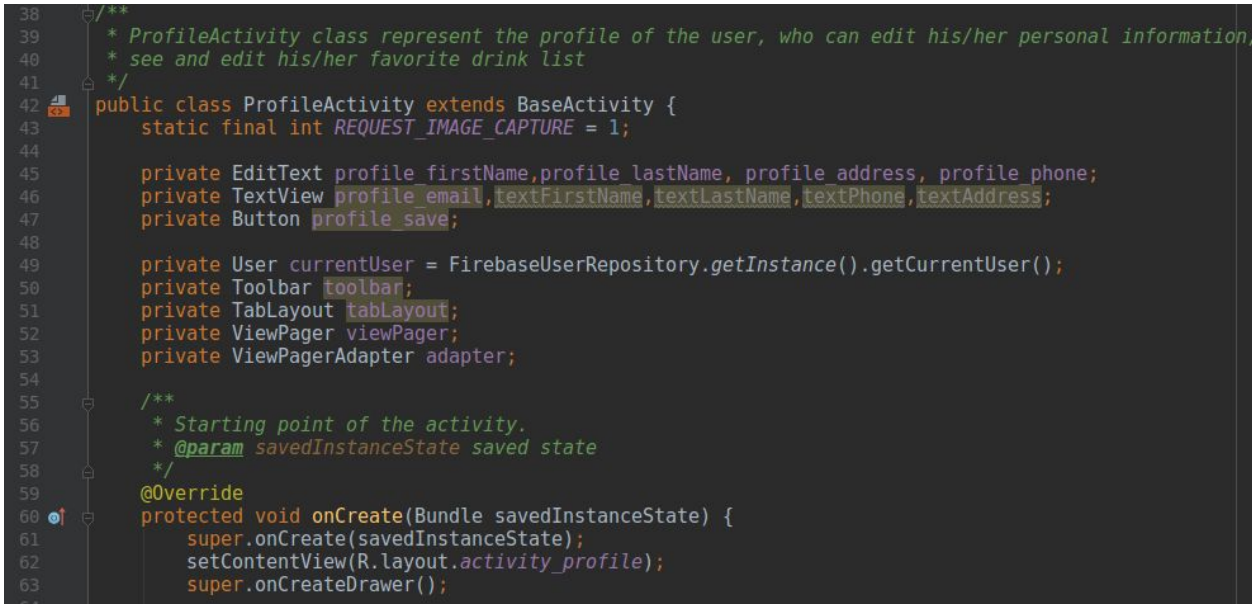


Following the navigation journey, the user location and drink journey show the activities that can be launched from the LocationDisplayActivity and the DrinkDisplayActivity.



Object-Oriented Programming

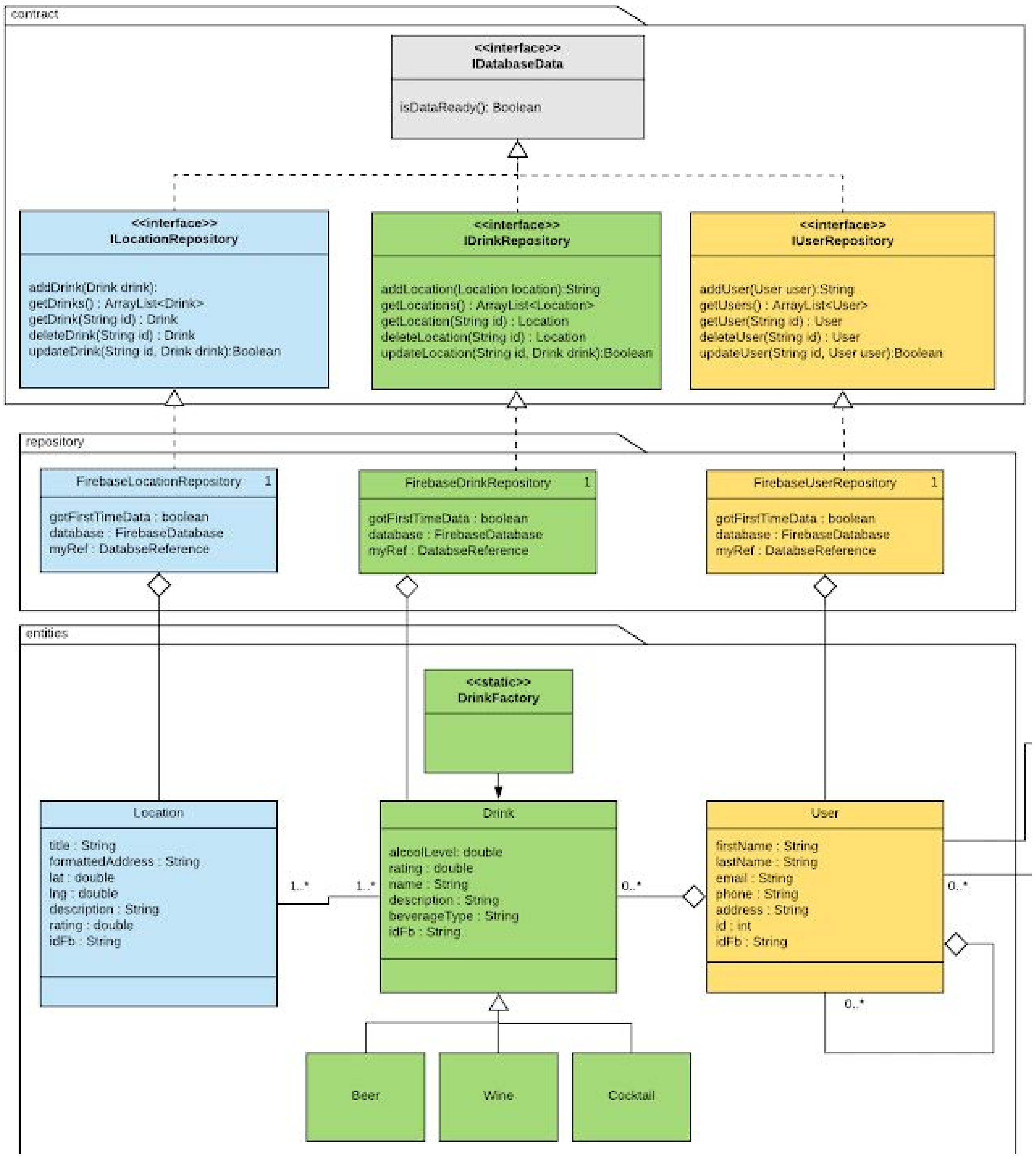
Being in an object-oriented programming course, it was obvious that the BeerApp would be designed and conceived with OO in mind. In the application, patterns and OO design can be found majoritarily in the model structure implementation, for example all the entities used, their creation and persistence within the application and the ways used to access and use them properly. As with navigation and activities, the main feature regarding OO is that all our main navigation pages (the ones found in the main menu) extend our BaseActivity class so that they share the implementation of the navigation drawer only by calling the *OnCreateDrawer()* in their own *OnCreate* method. The only exception is the MapsActivity, but the reason behind this will be discussed in the encountered problems section. Else, it was more put to use in the code layer, for example by simply encapsulating the smallest actions and features in simple methods and by respecting coherence within classes.



```java
/**
 * ProfileActivity class represent the profile of the user, who can edit his/her personal information
 * see and edit his/her favorite drink list
 */
public class ProfileActivity extends BaseActivity {
    static final int REQUEST_IMAGE_CAPTURE = 1;

    private EditText profile_firstName,profile_lastName, profile_address, profile_phone;
    private TextView profile_email,textFirstName,textLastName,textPhone,textAddress;
    private Button profile_save;

    private User currentUser = FirebaseUserRepository.getInstance().getCurrentUser();
    private Toolbar toolbar;
    private TabLayout tabLayout;
    private ViewPager viewPager;
    private ViewPagerAdapter adapter;

    /**
     * Starting point of the activity.
     * @param savedInstanceState saved state
     */
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_profile);
        super.onCreateDrawer();
```

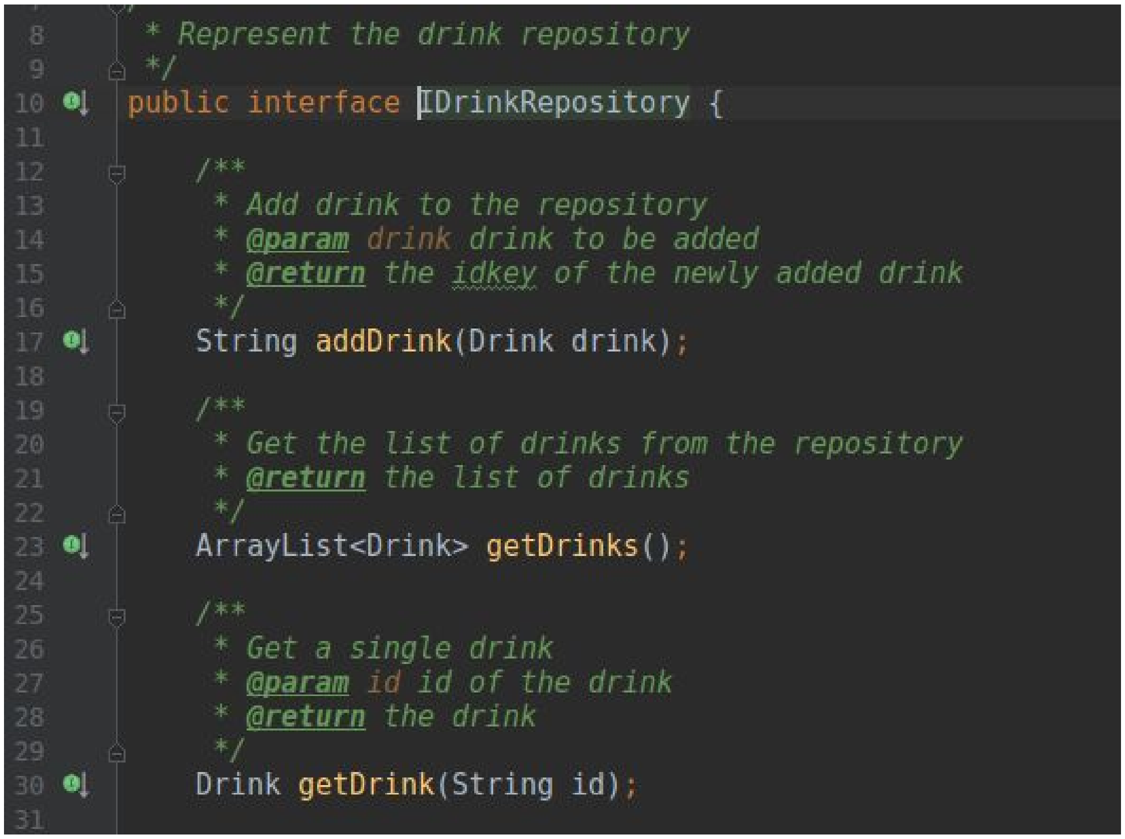
Example of a class extending BaseActivity and calling the OnCreateDrawer method

As for the application model structure, all the entities were properly implemented in a way that they are generic and can be easily created and accessed. Because the Firebase backend solution was used, there was also a concern regarding the repository usage to keep saved items and entities in the database. In the first place, a Drink class was created so it could be extended by any kind of more specific classes like beer, wine or cocktails. In this case it would be very easy to add other types of drinks in future development, soft drinks or coffees for example, or to change individual drink properties. Furthermore, the Drink creation is accessible via the *DrinkFactory*, using polymorphism to create any kind of drink by a single and generic *createDrink* method. The location management is not using this structure because there is only one type of location for now, but the same mechanism could easily be implemented if needed ultimately.


contract
<<interface>>
IDatabaseData
isDataReady(): Boolean
<<interface>>
ILocationRepository
addDrink(Drink drink):
getDrinks() : ArrayList<Drink>
getDrink(String id) : Drink
deleteDrink(String id) : Drink
updateDrink(String id, Drink drink):Boolean
<<interface>>
IDrinkRepository
addLocation(Location location):String
getLocations() : ArrayList<Location>
getLocation(String id) : Location
deleteLocation(String id) : Location
updateLocation(String id, Drink drink):Boolean
<<interface>>
IUserRepository
addUser(User user):String
getUsers() : ArrayList<User>
getUser(String id) : User
deleteUser(String id) : User
updateUser(String id, User user):Boolean
repository
FirebaseLocationRepository 1
gotFirstTimeData : boolean
database : FirebaseDatabase
myRef : DatabseReference
FirebaseDrinkRepository 1
gotFirstTimeData : boolean
database : FirebaseDatabase
myRef : DatabseReference
FirebaseUserRepository 1
gotFirstTimeData : boolean
database : FirebaseDatabase
myRef : DatabseReference
entities
<<static>>
DrinkFactory
Location
title : String
formattedAddress : String
lat : double
lng : double
description : String
rating : double
idFb : String
1..*
1..*
Drink
alcoolLevel: double
rating : double
name : String
description : String
beverageType : String
idFb : String
0..*
User
firstName : String
lastName : String
email : String
phone : String
address : String
id : int
idFb : String
0..*
0..*
Beer
Wine
Cocktail

UML of the application's model entities and backend structure

For the Firebase database usage, three singleton repository classes, one for drinks, locations and users, are used to access the existing database items and to add some to them. They all implement their specific interface, making it extremely easy to access the live firebase ressources from anywhere in the application, for example by simply calling the *getDrinks()* method on the *FirebaseDrinkRepository* singleton instance to access all the existing drinks of the database.



```java
 * Represent the drink repository
 */
public interface IDrinkRepository {

    /**
     * Add drink to the repository
     * @param drink drink to be added
     * @return the idkey of the newly added drink
     */
    String addDrink(Drink drink);

    /**
     * Get the list of drinks from the repository
     * @return the list of drinks
     */
    ArrayList<Drink> getDrinks();

    /**
     * Get a single drink
     * @param id id of the drink
     * @return the drink
     */
    Drink getDrink(String id);

```

IDrinkRepository interface example

Still regarding OO, one flaw that the App actually has is a lack of Exception management. This is still a problem right now, and it causes debugging and troubleshooting in particular cases to be more difficult. For instance, because the database will keep growing bigger and bigger, going through lists iterations of entities while debugging can juste become exhaustive and it could be take a really long time to spot a particular item error if it was the case. Adding some custom and more frequent errors exception catching would be a very good practice to upgrade the ease of maintenance for the App.

Encountered Problems

Like any programmer, beginner or not, it's impossible not to face problems, so it was expected that the team was going to face different issues. Some of those were finally solved after some research, but the app still has things that can be improved. All the major problems that have been addressed are listed below.

Fixed problems

- The implementation of the navigationDrawer along the overall application navigation faced some problems during the first stage of development. Fortunately, it was only a matter of time and research, and the learning curve about activities and basic android concepts was passed, making it not too hard to fix afterwards.

- Passing objects between activities was solved by creating more classes and transferring data through Firebase Repository.

- The implementation of a Facebook login feature was started in the first part of the project, but after troubles to synchronized, the decision was to remove it and only use Firebase email authentication for now to store user’s data.

- The usage of activities instead of fragments in the first milestone caused some problems in the long term, because the app should have been refactored almost entirely only for one thing : the mapsActivity was not compatible with the navigationDrawer just by extending BaseActivity. The fastest and easiest solution was then to duplicate the code of BaseActivity in the MapActivity. In a future major refactoring or enhancing phase of the app, switching to more fragments usage would definitely be the biggest and most important flaw to deal with. Still, the app is working great for the actual needs and it was decided by the developers that, in the context of the current iteration, it is not affecting the overall user experience enough to attempt those major changes.

- The Drink and Location entities have a many to many relationship. This cause trouble on how to reference and link them together without recreating the same instance twice. Move over, it created lot of complexity for CRUD operations on the document database. For example, when deleting a drink, we have to delete all is reference in the locations and vice-versa.

Known bugs

- Fragments and ListView can not be adapted correctly on some pages and as a result the UI faces some problems (LocationDisplayActivity). Mostly activities are used even knowing that the best practice is to use fragments.
- Unfortunately, taking pictures and replacing the location, drink or profile images does not work on the tablet that was lent to us for the project. This is an issue we didn’t have time to fix and actually didn’t really understand.

Future Features

Obviously, the BeerApp is not perfect and is not as complete as it could ultimately be, and some existing features are not as developed as we would like them to be. Therefore, some features present in the app are simply not completely finished or totally upgraded, and some are not implemented because they weren't core features that we thought would be really necessary to have in a first release for the due date.

To begin with, the feature that has the biggest flaw actually in the app is the rating system. As of right now, the simple rating implemented only gives the possibility to the user creating the drink or location to rate it, which doesn't really makes sense considering the community contribution in our app. For some weird reason, the team only thought about that issue when making the report and explaining the overall utility of the app for the Play Store description, giving us no time to implement it before the due date. Hence, our simple solution would be to make ratings on every drink editable by the user after its creation, so that whenever a user would put his own rating on a drink it would add it to the drink's ratings to show a rating average and the number of users who rated the drink.

The next features in the list for bigger releases would be a complete usage of the barcode scanner, a more developed friends feature, the possibility to connect to the app via third party connection systems like Facebook or Google and the sharing of image resources. At the moment, the use of the Google Mobile Vision API makes it possible and easy to detect almost any kind of barcodes and getting its numerical or string value, but in a goal of automatically identifying a drink product when adding it to the app, it needs an access to a huge product barcode database so we can find the corresponding one. A quick search on internet makes it is easy to understand that this kind of service would cost some money, but because the feature is already fully functional as it is, it would probably not be difficult to complete it in the event of the service being available. The friends feature, on the other hand, would not cost money and could be a very interesting feature to add in the near future. The goal would be to implement a friend request system to be notified if another user adds you as his friend. The user could then accept the request or not, and some public user informations would be accessible or not depending on your friend status with others. In the third place, connecting via external parties connection systems is such a common feature in so many applications today that it would easily make it more accessible and appealing to a lot of users. As discussed in the last section, a Facebook login feature was actually almost completed in the first stage of the app, but due to some problems it was let go for the final release. There is already a Firebase related functionality that could be used, but some work would definitely have to be done to complete this feature. Finally, a small but nice feature that our app would need is the sharing of image taken by the users for drinks, locations and profile pictures. In the actual release, the pictures taken with the camera are only saved on the user's device. Saving the pictures on Firebase would not only make the images visible to everyone, but would also limit local storage use on the users' devices.

External APIs

Google Map Api

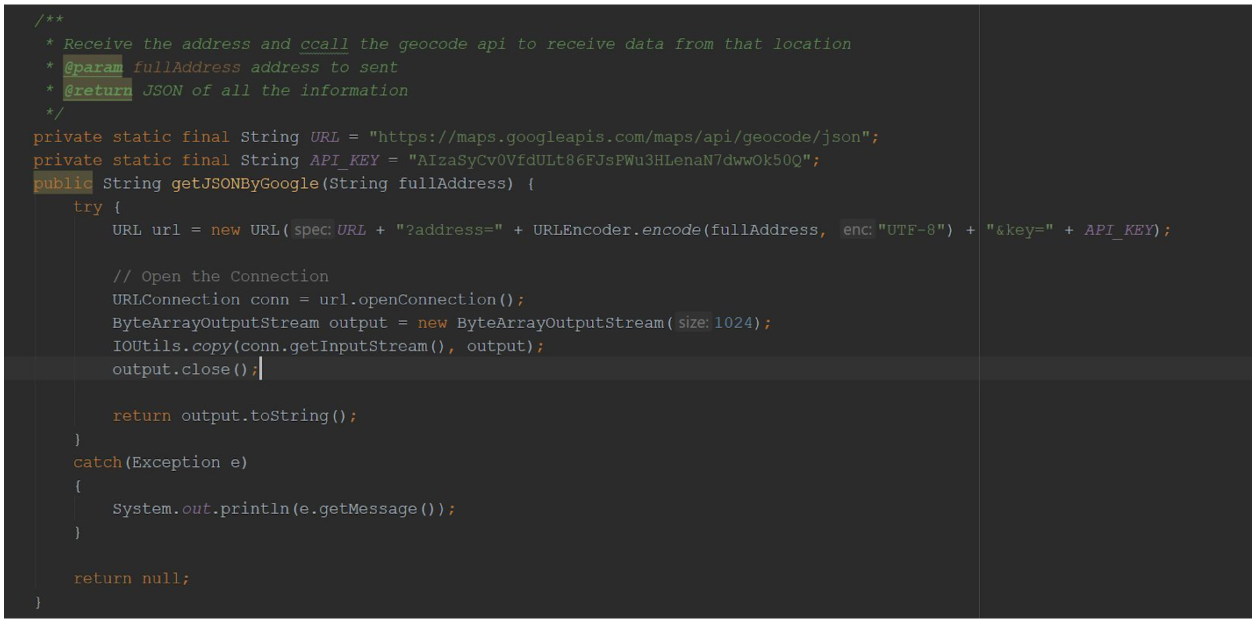
The main feature of the BeerApp is the map where all the bars are displayed. To make this work, the google map api was used. By overwritten two main functions of the api : the setMarker to be able to add our custom markers with data and the InfoWindow to be able to create a window with the list of drinks.

Since this Api was added, the use of the emulator stop working because of a compatibility problem with Google Play Services. According to many thread of StackOverflow, this is an issues that come and go every year. It was impossible to fix it, thus to develop the application, a physical device has to be used.

https://developers.google.com/maps/

Google GeoCode Api

This Api was used to transform an approximated address into an exact position with latitude and longitude. The coordinate are then used to place markers on the map. It is possible to use the API with XML or JSON, but we decided to use JSON format because it is better to transmit and receive data.



```
/**
 * Receive the address and ccall the geocode api to receive data from that location
 * @param fullAddress address to sent
 * @return JSON of all the information
 */
private static final String URL = "https://maps.googleapis.com/maps/api/geocode/json";
private static final String API_KEY = "AIzaSyCv0VfdULt86FJsPWu3HLenaN7dwwOk50Q";
public String getJSONByGoogle(String fullAddress) {
    try {
        URL url = new URL( spec: URL + "?address=" + URLEncoder.encode(fullAddress,  enc: "UTF-8") + "&key=" + API_KEY);

        // Open the Connection
        URLConnection conn = url.openConnection();
        ByteArrayOutputStream output = new ByteArrayOutputStream( size: 1024);
        IOUtils.copy(conn.getInputStream(), output);
        output.close();

        return output.toString();
    }
    catch(Exception e)
    {
        System.out.println(e.getMessage());
    }

    return null;
}
```

(The URL and API_KEY are position over the method only for this screen shot)

https://developers.google.com/maps/documentation/geocoding/intro

Google Mobile Vision Api

The Mobile Vision API is used for detecting and retrieving the information (alphanumeric, URL or else depending on the barcode type) linked to a barcode for the feature in the addDrinkActivity. It is the only usage in the App for now and is all located in the *BarcodeCameraActivity*.

https://developers.google.com/vision/

Conclusion

After months of work, the team managed to have a good result with the application. The encountered problems, whether they have been totally solved or not, have not discouraged us from stopping work and the final edition was ready on time. We tried to have a result as much as possible easier for a user and enjoy his time using it. Of course there are things to improve and added for a more complete app, but with this timeframe we are quite satisfied. We hope you enjoy our app, Cheers!