MOAPD: BikeShareApp

This is the documentation for the Assignment #1 for Mobile App Development subject. The assignment is based on the development of an app which consists in a bike sharing service, the user must be able to register rides and handle bike related information. I developed the app using Kotlin in Android Studio following the slides and reading material provided by the teachers as well as documentation and code fragments found on the Internet.

DESIGN CHOICES

General design

Since I had no previous experience in mobile app development, the project's initial layout was based on the exercises sessions featuring the functionalities: start ride, end ride, accessible with buttons, and list all rides. When you press one of the start/end ride buttons, it leads to a new screen where you will find the same layout design since both share the fragment xml file but different implementation. For selecting the bike, I decided to add a spinner in order to facilitate the user experience and also the location is set automatically by Internet data or GPS. Additionally, when finishing a ride a dialog window is displayed informing of the duration and price of the ride.

Going back to the main screen, when pressing the List Rides button all rides are displayed and represented as a RecyclerView with a ViewHolder composed of the bike's name, start time and place, end time and place. Furthermore, I considered necessary to make accessible in the main screen a "Find to Find" functionality since it is one of the most common operations in this type of app. It is implemented with a Google Maps fragment showing the position of bikes available around your current

location. In order to display the bikes accurately I had to store the latitude and longitude in the database since it does not work with string addresses. That would be all on the ride related screens.

The bike related screens are accessible through the use of the toolbar, with a tool icon in the upper right. Once pressed, the application jumps to new screen with the title Bikes, a TextView used for displaying the selected bike, a RecyclerView showing information on the stored bikes, and four buttons: Add new bike, bike info, show data(which activates the RecyclerView) and delete bike.

The feature of adding a new bike is managed on a new screen where various fields are asked to the user for instance the name of the bike which have to be unique, the type of bike and the price per hour as well as a photo of the bike. The photo taken is displayed as a Bitmap in an ImageView and stored in the database as a ByteArray. To use the camera a special permission is needed which is asked on the BikeShare screen firstly, and asked again when trying to take a picture if it was declined previously.

Regarding bike information is also managed from a different screen but accessible from the main managing bikes fragment where we have to select the bike we wish to obtain the data. There, we can check the availability of the bike and the location once the first ride has started as well as the information the user entered when registering the bike. Moreover, we can consult the rides history of the bike itself on a RecyclerView, distinct to the one in the main screen, containing starting and ending place, the time of the ride in minutes and the cost of the ride.

Eventually, I decided to add an essential functionality in every application which is a login and registering screen where you can create a user and start the app exclusively for that user. Although it is not implemented entirely, there is no singleton for the user once it has logged in, I considered it necessary to establish the basics for a real app experience and prepare the project for future development. New features could be added, for example, handling payments, linking bikes and rides to users, storing information relative to the user... Furthermore, I was not comfortable with the password being stored as a normal string and implemented a function to encrypt it to SHA-256 to make it more real. Finally, I would like to add that not only on the initial screens but all screens I make use of the toast function, which displays a pop-up message, in order to handle common mistakes by the user.

Database design

Storing the data obtained from the user is one of the most important processes of an application. In my case, I decided to use SQLite using Room Persistence Library since it provides a robust structure composed of various layers to respect simultaneous access to the database. The lower layer is the dao which is the interface responsible for interacting with the database through queries and prepares the data for being accessible from the repository. The last layer is the view model(VM) that access the repository and allows the data to be handled in the user interface.

The project consists of 3 data classes, each with its own database structure:

- **Ride,** stores all information related to rides and provides the functions necessary to get and modify its attributes. With the VM, we can interact with the database to get, for example, the instances of ride given a bike name.
- **Bike,** stores all information related to bikes and provides the functions necessary to get and modify its attributes as for the location of the bike or a photo of the bike which the DB allows as to store as a ByteArray.
- **User,** stores all information related to users. It is essential to the process of logging and registering an exclusive username for using the application.

From my point of view, using SQLite with Room Library is a solid method to store and access the data used in your application although it might be a slow feature to programme in the beginning. Finally, I desire to admit that the storage in my application could be improved by using an online database, for example Realm, since handling accesses to the same tables from multiple devices is more accurate to reality.

Primary functions implementation

- **Start Ride,** the function is implemented in the OnClickListener method of an Add Button. We have automated the process of obtaining the location and I have programmed a spinner to show the available bikes, we can suppose we have all the information requested without errors when the user click the Add Button. Nevertheless, I still make sure the bike is still available when creating the ride instance since in a real life situation another user could have got it seconds before. Once checked, the ride instance is created and registered in the database, same with the bike selected which information is updated and uploaded in the database.

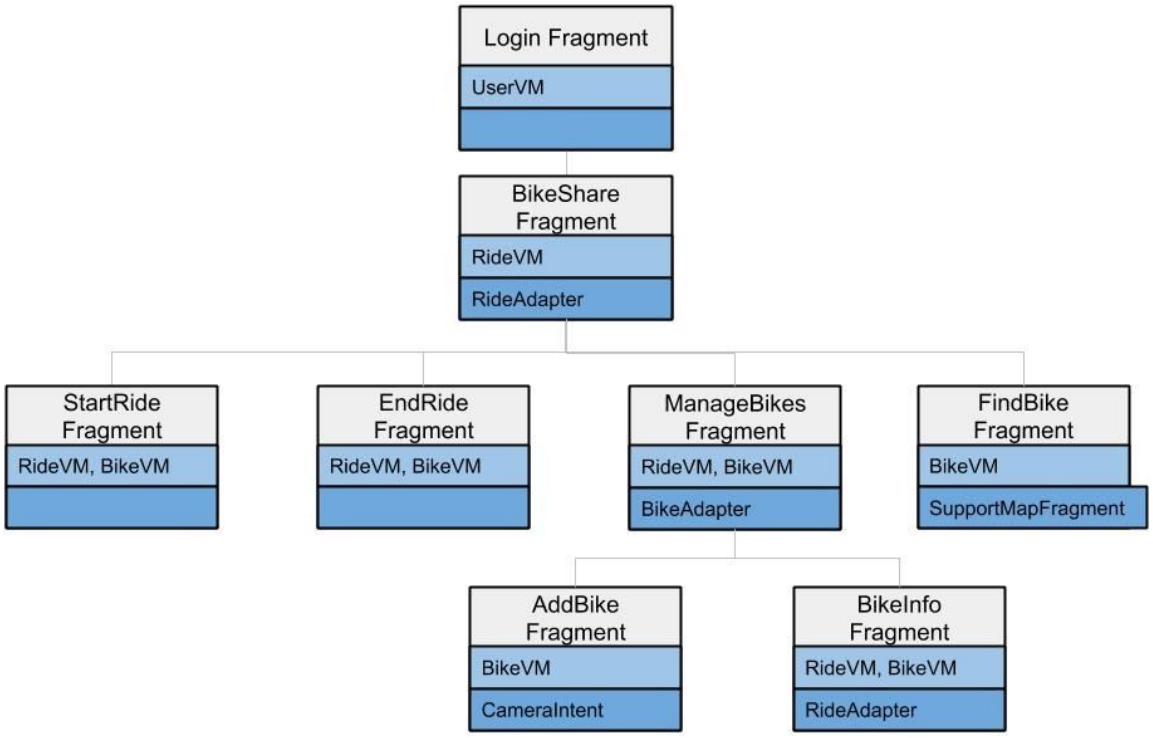
- **Add Bike,** the process of adding a bike is quite similar to starting a ride, a few fields are requested before being able to add the new bike, for instance, we need a bike name which must be unique, the type of bike, the price per hour

of the bike and a picture of it taken with the camera. Only when all this information is provided the bike instance is created, the possible errors are handled with a toast function.

```
mAddButton.setOnClickListener { it: View!
    val name = mBikeName.getText().toString().trim()
    if (name.isNotEmpty()) {
        var alreadyExists = bikeVM.getBikeByName(name)
        if (alreadyExists == null) {
            if (mPriceHour.getText().toString().trim().isNotEmpty()) {
                if (photoSafe != null) {
                    var latLng = LatLng(0.0,0.0)
                    mBike = Bike(
                        name,
                        mTypeBike.selectedItem.toString(),
                         mLocation: "Not available",
                         mLatitude: 0.0,
                         mLongitude: 0.0,
                        mPriceHour.getText().toString().trim().toFloat(),
                        photoSafe!!,
                         mAvailable: true
                    )
                    bikeVM.insert(mBike)
                    updateUI()
                } else toast( message: "Take a picture of the bike")
            } else toast( message: "Price per hour is empty")
        } else toast( message: "Bike name already exists")
    } else toast( message: "Name is empty")
}
return view
```

- **Find a bike,** this feature is not implemented exactly as a function but as a whole fragment since I wanted to have it on full screen and also it is how Google API provides it. Firstly, access to the current location of the user is obtained on the BikeShare fragment using Fused Location Provider and passed as an argument to the FindBike activity and subsequently to the fragment, I will use it to focus the camera around it. In order to incorporate Google Maps to the application, the API is needed which requires a key obtained on Google Cloud Platform, otherwise our application would not work. Then, we are able to get a Google Maps fragment instance which I use to loop through the available bikes and add markers, with their name tagged, on their current locations. This means all available bikes are uploaded to the map; one possible optimization would be display only the bikes which can be seen from the current zoom configuration and update it when the camera changes. This could be done with a precomputed list of near bikes, for example, 10 kilometres square from the initial location, considering that making the computations every time the camera changes would severely slow down the execution.

User Interface diagram and features



Problems and bugs

I have noticed there is a bug depending on the device you are running the app. In my case, I debug the whole project in my Xiaomi Redmi Note 7 which happens to have a very large screen. Once the app was finished, I asked my friends to install the app and they informed me that some of the could not see part of the bottom part. I tried to fix it making the layout more general, but my final conclusion was that I would have to design from scratch some of the screens.