Technical Project Report - Android Module

Project X - Community collaboration platform for bicycle users

Subject:	Mobile Computing
Date:	Aveiro, 7th of November 2023
Students:	103154: João Fonseca 103183: Diogo Paiva
Project abstract:	This Android based application provides a community collaboration platform for bicycle users. It provides a map using CyclOSM (a custom OpenStreetMap layer), allows the creation of interest using the camera, recording of routes using the location and getting directions from the current location to a destination by calling an external navigation application. As a community, users can rate points of interest, share their routes and level up in the ranking system. The backend is based on services provided by Firebase. It provides authentication mechanisms, the database to store information about points of interest, recorded routes and users, and a storage for saving images of users and points of interest.

Report contents:

1 Application concept

This Android app was made for people that like to ride a bike, for fun or as a sport. It acts as a community collaboration platform for bicycle users. It provides a map using CyclOSM (a custom OpenStreetMap layer), and allows the creation of points of interest using the camera, recording of routes using the location and getting directions from the current location to a destination by calling an external navigation application. As a community, users can rate points of interest, share their routes and level up in the ranking system. This encourages the users to create more POIs and rate them, gaining experience points and climbing levels in return.

2 Implemented solution

Architecture overview (technical design)

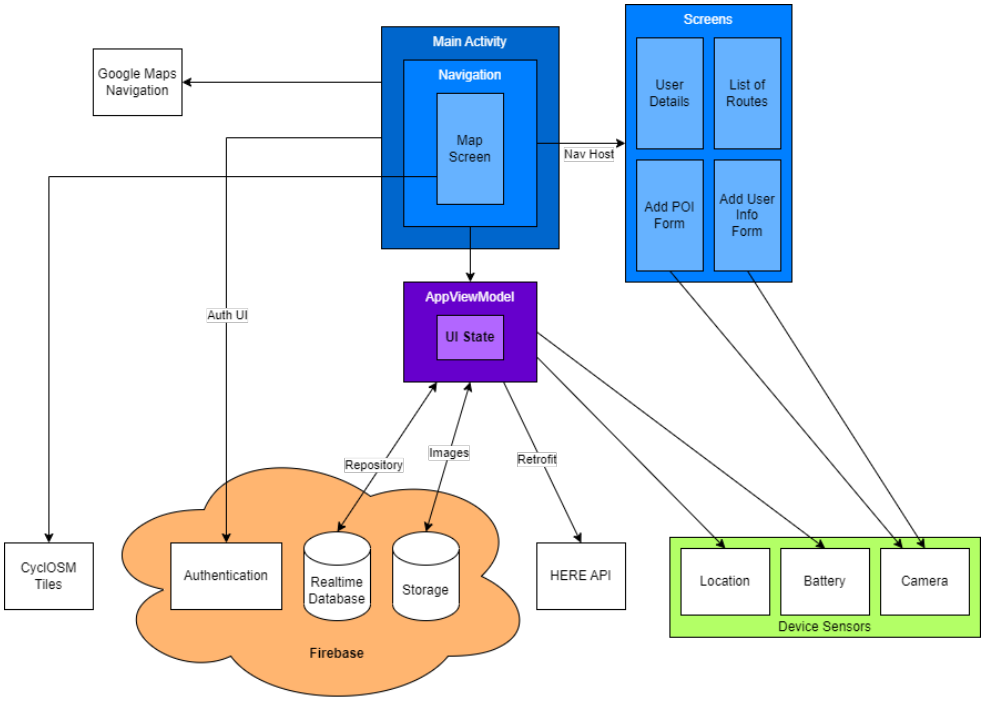

Fig. 1 – Architecture diagram.

In this application, we followed the recommended model-view-viewmodel architecture for Android apps. Our UI is constructed using Jetpack Compose[1], and is organized in different screens. These screens include the map view (which is the main screen), the user details page, the list of routes, and the forms to add user information after registration and to add a new point of interest. It is possible to navigate between pages, thanks to Navigation for Compose.

The data displayed in the UI depends on the state of the view model, which contains the data and business logic necessary for the UI to work. This allows us to define a boundary between the data and it's visual representation.

The backend is based on services provided by Firebase[2]. It provides authentication mechanisms, the database to store information about points of interest, recorded routes and users, and a storage for saving images of users and points of interest. This access is

abstracted by the means of repository instances we created, to allow for an easier development of the business logic. Geocoding and reverse geocoding services, used to search for locations on the map and to get the names of the start and destination points of a recorded route, are provided by HERE and its access is made using Retrofit.

We followed the recommended Android app architecture for building robust, high-quality apps[3]. We separated our app into the UI layer (screens), the domain layer (viewmodel) and the data layer (repositories + Firebase). Data is not stored in the UI components, instead we access the viewmodel to read and modify it through functions provided by it. The text strings inside the app use string resources, which allow for an easier translation of the app into multiple languages.

We have 5 data models which are stored in our Firebase backend: POI, Point, Rating, Route and User. The POI contains a list of Ratings, the Route contains a list of Points. So we have 3 repository interfaces – POI, Route and User – which provide methods of modifying these entities. Repositories connect to the Realtime Database and the Storage services provided by Firebase. When there is a need to save a picture (from an user or a point of interest), we first store it in the Storage service, get its URI, and update the object before storing it in the Realtime Database.

In the Realtime Database, the data is structured in the following way.

- Poi
  - poi_id
    - createdBy: string (user_ username)
    - description: string
    - latitude: double
    - longitude: double
    - name: string
    - pictureUrl: string
    - ratings: list<user: string (user_id), value: bool>
- route
  - user_id
    - route_id
      - createdBy: string (user_username)
      - destination: string
      - origin: string
      - points: list<latitude: double, longitude: double, timestamp: long>
      - totalDistance: double
      - totalDuration: int
- user
  - user_id
    - addedPOIs: int
    - displayName: string
    - givenRatings: int
    - pictureUrl: string
    - receivedRatings: int
    - totalXP: int
    - username: string

In the Storage, the data is structured in the following way.

- poi/
  - <poi_id>.jpg
- user/
  - <user_id>.jpg

To provide a route from the user's current location and a destination, we use the

device's Google Maps app. We create an intent for it, passing in the latitude, longitude and the mode ("b" for a bicycle route). When this intent is processed, Google Maps takes over our app and starts giving the indications.

To have access to the user's current location, either to display on the map, record a route or add a new point of interest, we had to include the right access permissions the Android Manifest. We used fused location provider, which allows for a more precise location by the means of combining multiple sources (such as GPS and Wi-Fi). We receive location updates through a callback, which allows the user location to be updated on the go, and is essential for us to record paths.

To add a profile picture and a new point of interest, the user needs to provide a picture from either the device's storage or the camera. In the case of a locally stored image, the user is able to access the device's file system and select the preferred image. In the case of the camera, the app opens the camera and the user can take a photo. Until the user submits the form, the picture is stored locally. Only after submission, it is upload to Firebase's Storage service.

To simulate the battery capacity of the bicycle, we accessed the device's current battery level, and fetch its current level every 10 seconds using a coroutine.

Implemented interactions

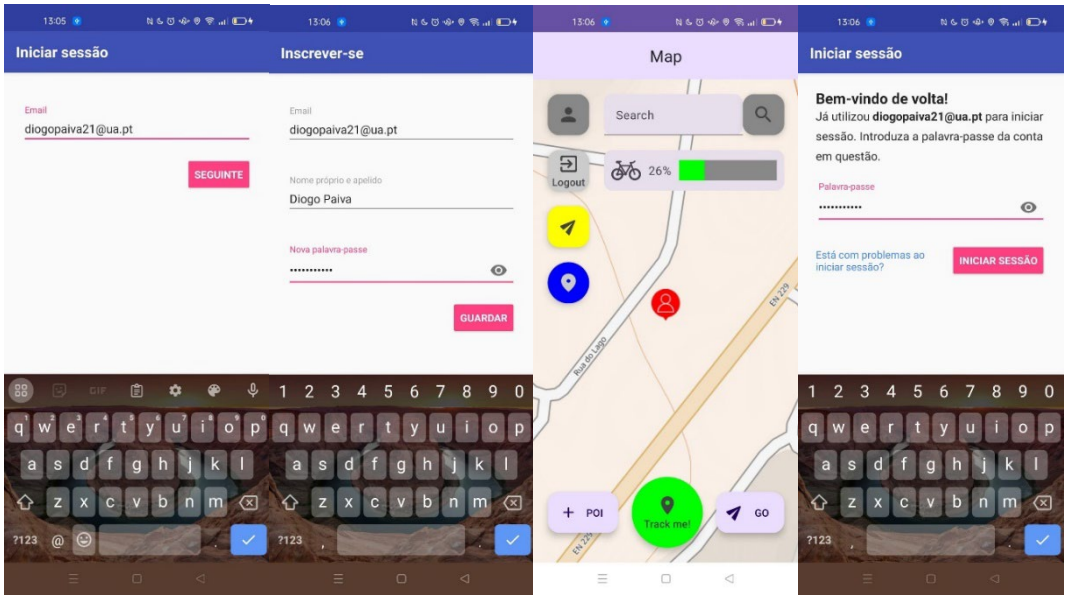

Fig. 2 – Login and Signup.

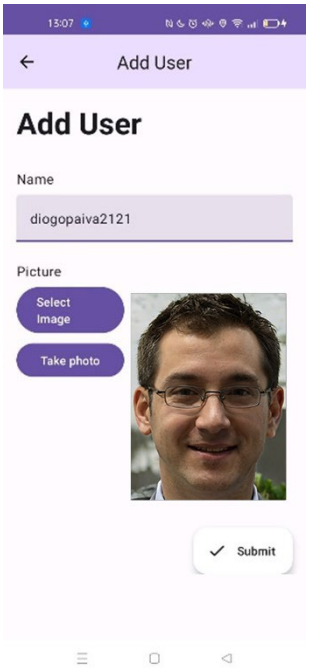

Fig. 3 – Add user details.

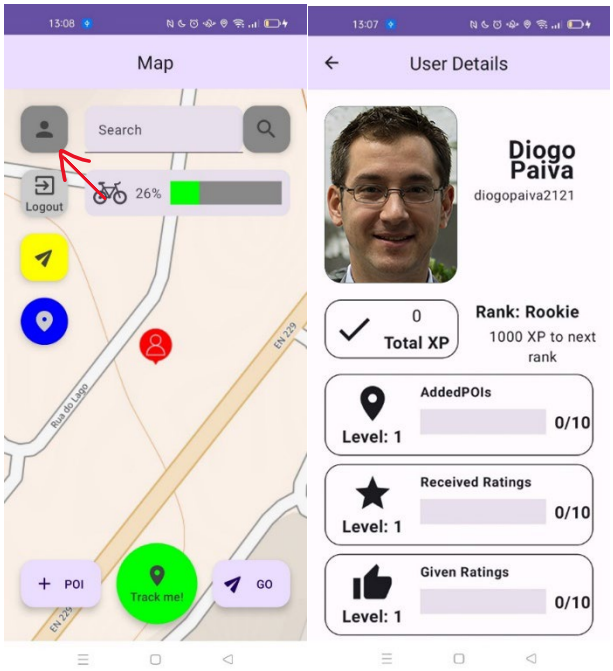

Fig. 4 – Check user details.

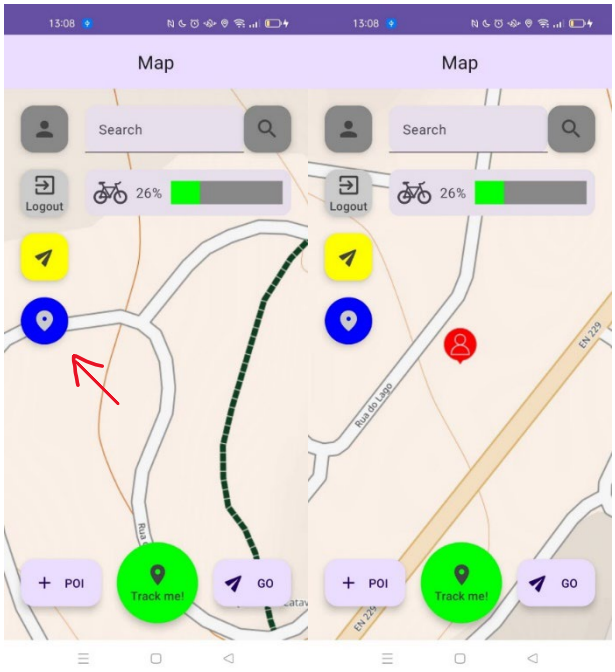

Fig. 5 – Go to user's location.

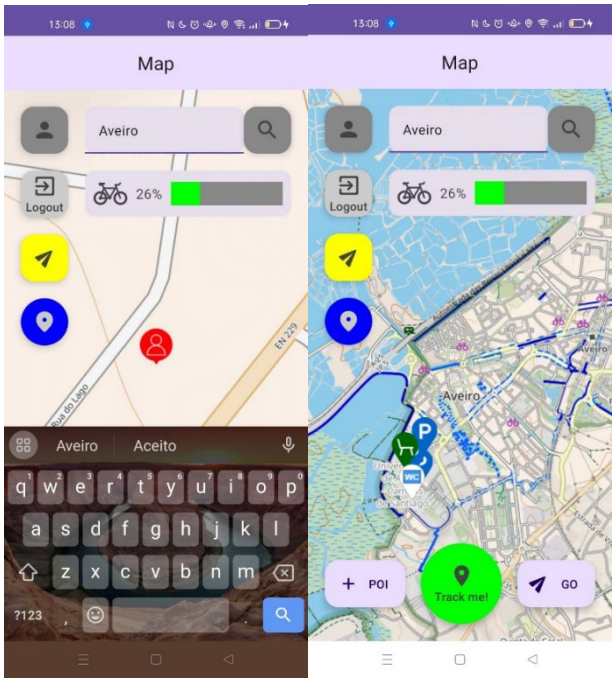

Fig. 6 – Search for a location.

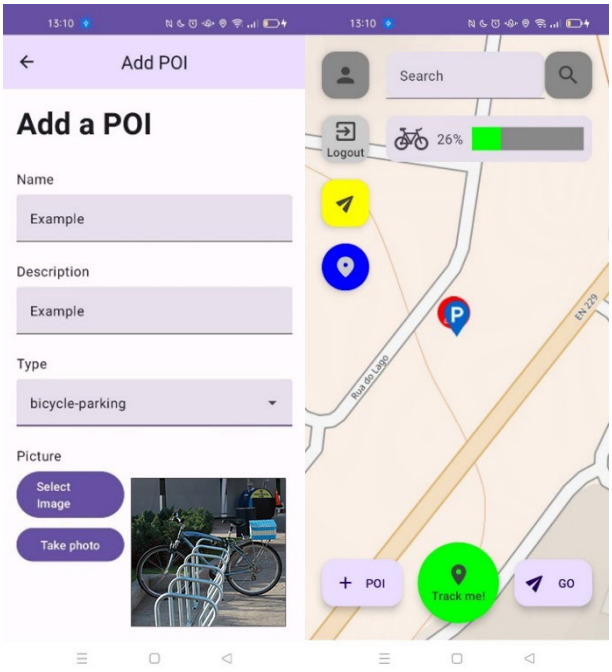

Fig. 7 – Add a point of interest.

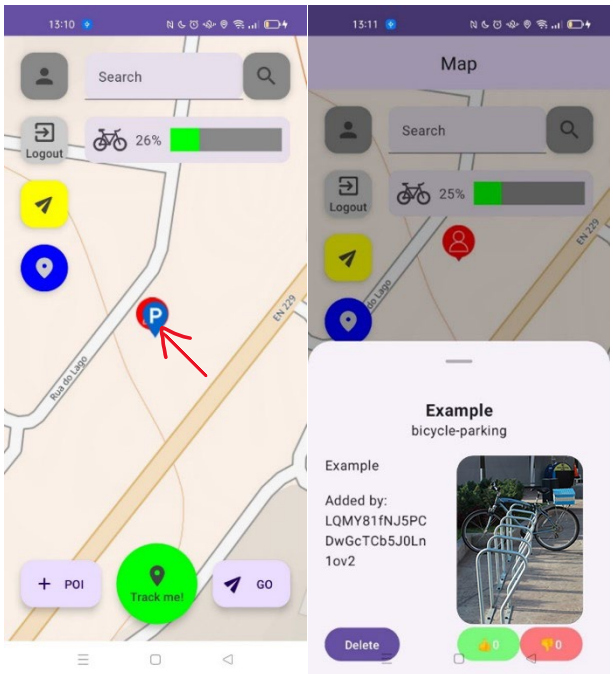

Fig. 8 – Check point of interest details.

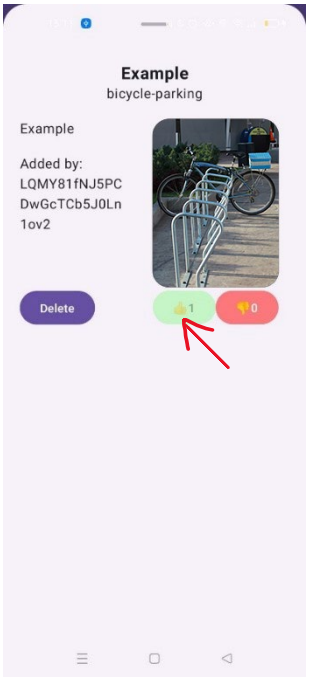

Fig. 9 – Rate a point of interest.

Fig. 10 – Delete a point of interest.

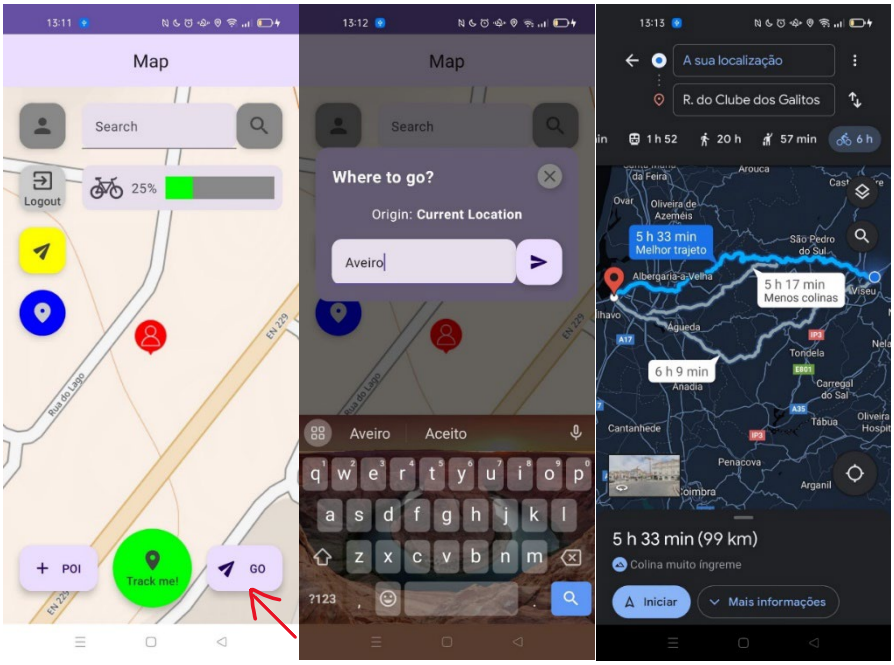

Fig. 11 – Go to a destination.

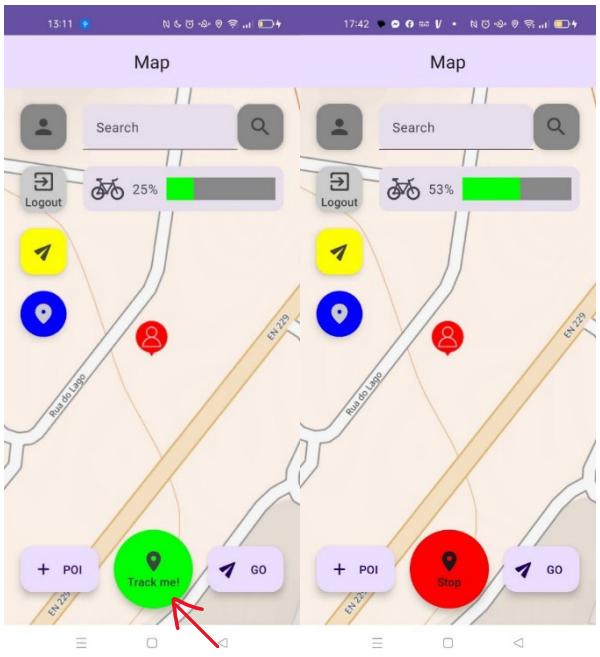

Fig. 12 – Track a route.

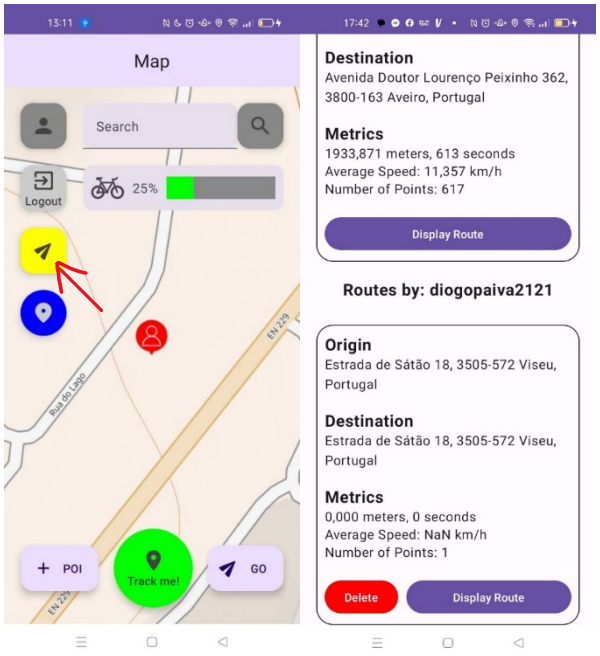

Fig. 13 – Check all routes.

- Display a route on the map.

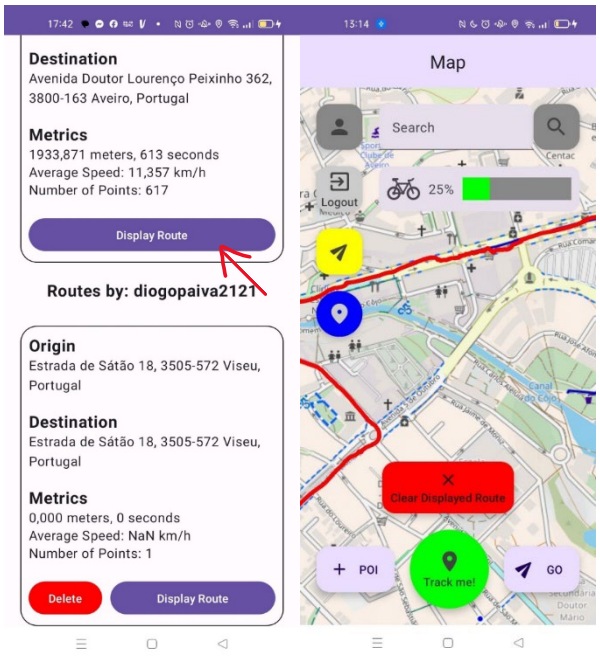

Fig. 14 – Display a route on the map.

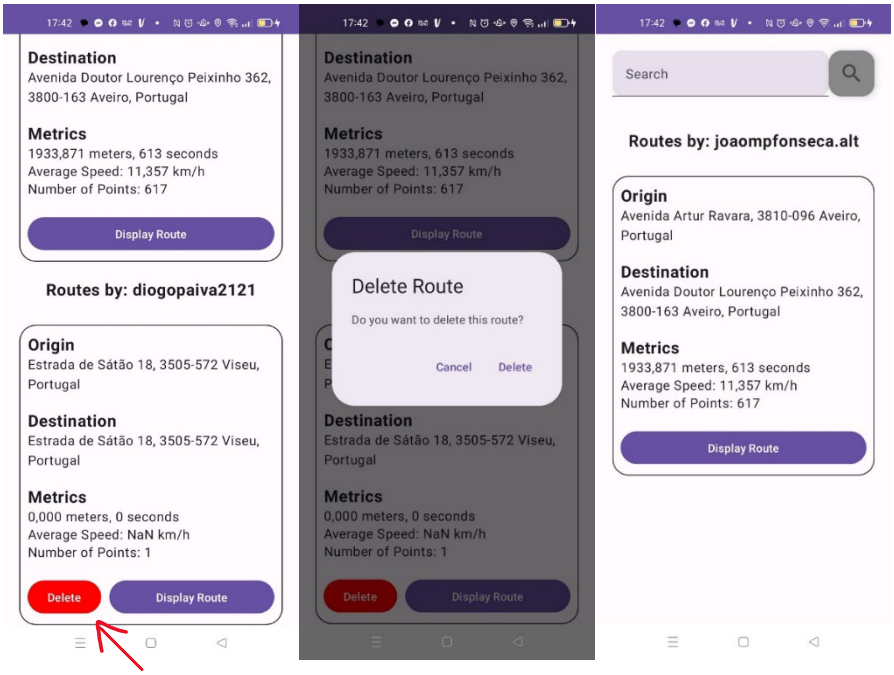

Fig. 15 – Delete a route.

Project Limitations

From the professor's suggestions, we didn't implement a rating and comment system in the page with routes. The battery level of the bicycle is mocked using the device's current battery level, since we didn't have access to a bicycle in which the battery status could be fetched. It is currently not possible to modify information from the user or points of interest on the map. We also planned to give directions from an origin to a destination point using the GraphHopper routing engine, but given time constraints we had to resort to using Google Maps's routing capabilities.

3 Conclusions and supporting resources

Lessons learned

The major drawback that could compromise the development of the entire app would be the ability to properly show a map. We had to do some research, but eventually found a working library for Jetpack Compose, which supported OpenStreetMap layers and common Leaflet features that were essential, like drawing markers and lines on the map[4]. It took a lot of time to understand how it worked, since it was a new library and didn't have proper documentation (we had to analyze an example project provided by the developers.

The implementation of Firebase authentication took more time than expected to setup, since there was a recent change to how it worked (in September of this year), which prevented us from using the email authentication. There was an obscure workaround, which involved disabling the security feature to make it work again.

The use of Jetpack Compose made the development of the app much easier than using views. Despite its novelty and lack of online examples, we were able to follow its intuitive documentation to learn about the composables. We would like to highlight that he tutorials given to us during practical classes contained the basic concepts we applied on this app, and learning these concepts before allowed for a smoother development experience on the project.

Work distribution within the team

Taking into consideration the overall development of the project, the contribution of each team member was distributed as follow: João Fonseca did 60% of the work, and Diogo Paiva contributed with 40%.

Project resources

Resource:	**Available at:**
Code repository:	https://github.com/joaompfonseca/cm-android-project
Ready-to-deploy APK:	https://github.com/joaompfonseca/cm-android-project/tree/master/apk

Reference materials

[1] Jetpack Compose Documentation
https://developer.android.com/jetpack/compose/documentation

[2] Firebase
https://firebase.google.com/

[3] Guide to app architecture
https://developer.android.com/topic/architecture (accessed on 03/11/2023)

[4] OSM Android Compose
https://github.com/utsmannn/osm-android-compose (accessed on 17/10/2023)