Mobile Application Development 2020-21

A car pooling application

Lab3 – Storing data in a database

Learning objectives

- Using the ViewModel
- Using Cloud Firestore database
- Using Google Sign-In
- Using Firebase Cloud Messaging

Description

You will improve the app created in the previous lab to save data on a database and retrieve them.
The main page of the application will become a section that contains the list of advertisements of other users. If no trip is present, a message is shown. Each item of this list, on click, allows to show the details of it. The user is able to search trips he is interested in. Furthermore, an user is able to notify the person who published the trip that he wants to book it.
The login to the car pooling application must be implemented using Google Sign-in. If the user is not authenticated, he must login before using the application.
The project will be kept on bitbucket as a private repository.

Steps

1. Open the project you have created in the previous lab.
2. Model the data of your application and store it in the Cloud Firestore database.
    a. Create a model of the data needed by the application, identifying the collections and the relationships among them (users, trips, bookings, …).
    b. Using the Firebase assistant of Android Studio, create a new Cloud

Firestore database.

3. Update your application using the Cloud Firestore database.
    a. Update the user profile management saving and retrieving data from Firestore
    b. Update the item fragments saving and retrieving data from Firestore
    c. Commit the project. Push it onto the remote repository.
4. Add a login page to your application.
    a. This login page is shown every time you launch the app and the user is not logged in. Use Google Sign-In to authenticate your users https://firebase.google.com/docs/auth/android/google-signin
    b. Commit the project. Push it onto the remote repository.
5. Create a new fragment named OthersTripListFragment.
    a. Its purpose is to browse all the trips belonging to other users. You will manage this list using a RecyclerView.
    b. If the list is empty, a message must be shown.
    c. The single items will be shown as *CardViews*, each one representing a compact representation of the item information. Update the card you made in the previous lab in order to be used in both the two lists, passing it a parameter that customizes the visualization. By clicking anywhere on the card (except on the button) a navigation to the TripDetailsFragment will take place.
    d. Add a search menu item that allows to filter the shown trips (by departure location, departure date and time, price, ...)
    e. Set this new fragment as the main page in the navigator
    f. Commit the project. Push it onto the remote repository.
6. Update TripDetailsFragment so that it can be used both from OthersTripListFragment and the old TripListFragment
    a. Customize the fragment with a parameter to hide or show the proper functions (eg. you don't need to edit a trip if it is not yours).
    b. Add a listener to intercept changes to the document on the database and keep the view updated accordingly.

    c. Add a FAB to notify the owner of the trip that you are interested in booking it. Use Firebase Cloud Messaging to send notifications https://firebase.google.com/docs/cloud-messaging

    d. The owner of the trip should be able to view the list of other users that are interested to book it.

    e. Commit the project. Push it onto the remote repository.

7. Update ShowProfileFragment so that it can be reused to see current user profile or the profile of another user that wants to book one of its trips

    a. Not all the fields should be shown for privacy reasons

    b. The navigation to edit another user profile should be removed

    c. Commit the project. Push it onto the remote repository

8. Update the TripEditFragment with trip state management

    a. When a driver accepts to offer a trip to another user, its status is properly updated, the buyer of the trip is notified and if there aren't other available seats all the interested users are notified.

    b. If a driver doesn't want to offer one of his trips anymore, it can block the trip advertisement and, if somebody was interested in that trip, a notification is sent.

    c. Commit the project. Push it onto the remote repository.

Summary

ViewModel

- The Android app architecture guidelines recommend separating classes that have different responsibilities.
- A *UI controller* is UI-based class like `Activity` or `Fragment`. UI controllers should only contain logic that handles UI and operating system interactions; they shouldn't contain data to be displayed in the UI. Put that data in a `ViewModel`.
- The `ViewModel` class stores and manages UI-related data. The `ViewModel` class allows data to survive configuration changes such as screen rotations.

- The `ViewModel` should never contain references to fragments, activities, or views, because activities, fragments, and views do not survive configuration changes.
- `ViewModel` is one of the recommended Android Architecture Components.
- `ViewModelProvider.Factory` is an interface you can use to create a `ViewModel` object that survives configuration changes and may be configured with relevant data before it is made available to the owning activity/fragment.

Submission rules

- Lab must be submitted by May, 12th at 23:59
- The functionalities implemented in your code as well as the design of the user interface will be evaluated
- Before submitting, clean the project using *Build -> Clean Project*
- Create a zip file with your project and name it groupXX_lab3.zip
- Upload it on the Polito web portal (only one student of the group must upload it); for multiple uploads, only the most recent file will be evaluated