A car pooling application

Lab2 – Trip Ads management

Learning objectives

- Managing lists, cards
- Using fragments
- Using the androidx Navigation component
- Improve the UX using animations and other form of feedback

Description

You will extend the app created in the previous lab to allow a user to manage its trip advertisements. Each advertisement consists of a basic set of data (that can be extended):

- Photo of the car
- Departure and arrival location
- Departure date and time
- Estimated trip duration
- Number of available seats
- Price
- A description for additional information (eg. available space for luggages, …)
- Optionally, intermediate stops and estimated date and time

The main page of this section contains a list of trip advertisements. If no trip is present, a message is shown. Each item of this list, on click, allows the advertisement to enter in "edit mode". This page is similar to the one you implemented for editing the user profile.
In edit mode, a "Save" button must be inserted to store the information; if the user taps the back button the changes are discarded.
The main page features a floating action button that allows to create a new advertisement, using the same view for editing an existing one.
The app should contain a navigation drawer for switching between the two different sections: user profile and advertisements.

The project will be kept on BitBucket as a private repository.

Steps

1. Open the project created in the previous lab.

2. Create the MainActivity, based on the Navigation Drawer template

    a. The purpose of this activity is to host all the screens related to the application (the list of trip advertisements by the current user, the details of a single trip, the editor screen for creating/modifying a single advertisement, the user profile and the corresponding editor screen). All screens will be implemented as fragments and the activity will act as a coordinator, relying on the Navigation library for switching among different screens.

    b. All data managed by the application will be persisted in the local file system, as plain files for the images and as SharedPreferences for the structured data, so that, when the app is started again, no content is lost. Each fragment will be responsible for reading or writing the content it manages.

3. Create the TripDetailsFragment. The purpose of this class is to display the details of a trip advertisement: these should comprise, at least, the following items:

    i. Photo of the car

    ii. Departure and arrival location

    iii. Departure date and time

    iv. Estimated trip duration

    v. Number of available seats

    vi. Price

    vii. A description for additional information (eg. available space for luggages, …)

    viii. Optionally, intermediate stops and estimated date and time

    ix. Add any other extra data items as relevant to the application that your group is conceiving

    b. The screen must be responsive to different screen sizes and different

orientations: check that everything works as expected choosing different device presets, both in landscape and in portrait mode.

   c. This view has a menu item that allows to edit the advertisement.

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

4. Create a second fragment named TripEditFragment that allows the user to edit the content of a trip advertisement

   a. The layout file will mimic the previous one with some exceptions: all *TextView*s will be replaced by corresponding *EditText*s or the corresponding material design component,*TextInputLayout*; the ImageView will show an *ImageButton* on top of it, labelled with a camera icon; the date must be chosen with a *DatePicker*.

   b. Consider using the fragment arguments as a way to communicate data from the invoker to the target (what must be communicated?)

   c. As in the previous lab, if the user clicks on the *ImageButton* a floating context menu appears, allowing to select an image from the phone gallery or use the camera to take a picture.

   d. If the user rotates the device, data entered so long should not be lost

   e. This activity has an option menu with a single item, named "Save". If it is pressed, data is persisted and a navigation to a relevant screen (which one?) takes place.

   f. Commit the project. Push it on the remote repository.

5. Create a third fragment named TripListFragment

   a. Its purpose is to show a list of advertisements. You will manage this list using a RecyclerView. A floating action button (FAB) will be part of this layout. Mark it as "home" in the navigation graph

   b. The single trips will be shown as *CardViews*, each one representing a compact representation of the trip information. The card will provide a button to edit the content. By clicking anywhere on the card (except on the button) a navigation to the TripDetailsFragment will take place.

   c. If the list is empty, a message must be shown.

   d. If the FAB is pressed, a navigation to the TripEditFragment will take place, allowing a new trip to be added to the list. If the new trip is saved, the list will have to be updated accordingly. How can you ensure

it? (take into consideration that fragment instances may be reused by the system, instead of being re-created)

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

6. Convert the ShowProfileActivity and EditProfileActivity of the previous lab into fragments respectively named ShowProfileFragment and EditProfileFragment.

    a. The layout can probably be reused as it is, the logic of the activity must be adapted to deal with the Navigation library

    b. The same considerations for passing data along the navigation previously mentioned apply.

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

7. Implement the drawer navigation, showing in the headerLayout an extract of the user profile and provide links to the TripListFragment and to the ShowProfileFragment. More links will be added here in future labs.

    a. Improve the navigation experience introducing animations when switching from different sections and/or with feedback to the user (e.g. Snackbars when an action is performed).

    b. Commit the project. Push it on the remote repository

Summary

Navigation components

To use the Android *navigation library*, you need to do some setup:

- Add dependencies for navigation-fragment-ktx and navigation-ui-ktx in the module-level build.gradle file.
- Add an ext variable for the navigationVersion in the project-level build.gradle file.

Navigation destinations are fragments, activities, or other app components that the user navigates to. A *navigation graph* defines the possible paths from one navigation destination to the next.

- To create a navigation graph, create a new Android resource file of type Navigation. This file defines the navigation flow through the app. The file is in the res/navigation folder, and it's typically called navigation.xml.
- To see the navigation graph in the Navigation Editor, open the navigation.xml file and click the Design tab.
- Use the Navigation Editor to add destinations such as fragments to the navigation graph.
- To define the path from one destination to another, use the Navigation Graph to create an action that connects the destinations. In the navigation.xml file, each of these connections is represented as an action that has an ID.

A *navigation host fragment*, usually named NavHostFragment, acts as a host for the fragments in the navigation graph:

- As the user moves between destinations defined in the navigation graph, the NavHostFragment swaps the fragments in and out and manages the fragment back stack.
- In the activity_main.xml layout file, the NavHostFragment is represented by a fragment element with the name android:name="androidx.navigation.fragment.NavHostFragment".

To define which fragment is displayed when the user taps a view (for example a button), set the onClick listener for the view:

- In the onClick listener, call findNavController().navigate() on the view.
- Specify the ID of the action that leads to the destination.

Conditional navigation navigates to one screen in one case, and to a different screen in another case. To create conditional navigation:

- Use the Navigation Editor to create a connection from the starting fragment to each of the possible destination fragments.
- Give each connection a unique ID.

- In the click-listener method for the View, add code to detect the conditions. Then call findNavController().navigate() on the view, passing in the ID for the appropriate action.

The options menu

The *options menu* is a menu that the user accesses from the app bar by tapping the icon with the three vertical dots ⋮ . To create an options menu with a menu item that displays a fragment, make sure the fragment has an ID. Then define the options menu and code the onOptionsItemSelected() handler for the menu items.

1. Make sure the fragment has an ID:

- Add the destination fragment to the navigation graph and note the ID of the fragment. (You can change the ID if you like.)

2. Define the options menu:

- Create an Android resource file of type Menu, typically named options_menu.xml. The file is stored in the Res > Menu folder.
- Open the options_menu.xml file in the design editor and drag a Menu Item widget from the Palette pane to the menu.
- For convenience, make the ID of the menu item the same as the ID of the fragment to display when the user clicks this menu item. This step is not required, but it makes it easier to code the onClick behavior for the menu item.

3. Code the onClick handler for the menu item:

- In the fragment or activity that displays the options menu, in onCreateView(), call setHasOptionsMenu(true) to enable the options menu.
- Implement onCreateOptionsMenu() to inflate the options menu:

```
override fun onCreateOptionsMenu(menu: Menu?, inflater: MenuInflater?)
{
```

```
    super.onCreateOptionsMenu(menu, inflater)

    inflater?.inflate(R.menu.options_menu, menu)

}
```

- Override the onOptionsItemSelected() method to take the appropriate action when the menu item is clicked. The following code displays the fragment that has the same ID as the menu item. (This code only works if the menu item and the fragment have identical ID values.)

```
override fun onOptionsItemSelected(item: MenuItem?): Boolean {

    return NavigationUI.onNavDestinationSelected(item!!,

        view!!.findNavController())

        || super.onOptionsItemSelected(item)

}
```

The navigation drawer

The *navigation drawer* is a panel that slides out from the edge of the screen. There are two ways for the user to open the navigation drawer:

- Swipe from the starting edge (usually the left) on any screen.
- Use the drawer button (three lines) ☰ on the app bar at the top of the app.

To add a navigation drawer to your app:

1. Add dependencies to build.gradle (app).
2. Make sure each destination fragment has an ID.
3. Create the menu for the drawer.
4. Add the drawer to the layout for the fragment.
5. Connect the drawer to the navigation controller.
6. Set up the drawer button in the app bar.

These steps are explained in more detail below.

1. Add dependencies to build.gradle:

- The navigation drawer is part of the Material Components for Android library.
  Add the Material library to the build.gradle (app) file:

```
dependencies {

    ...

    implementation
"com.google.android.material:material:$supportlibVersion"

    ...

}
```

2. Give each destination fragment an ID:

- If a fragment is reachable from the navigation drawer, open it in the navigation
  graph to make sure that it has an ID.

3. Create the menu for the drawer:

- Create an Android resource file of type Menu (typically called
  navdrawer_menu) for a navigation drawer menu. This creates a new
  navdrawer_menu.xml file in the Res > Menu folder.
- In the design editor, add Menu Item widgets to the Menu.

4. Add the drawer to the layout for the fragment:

- In the layout that contains the navigation host fragment (which is typically the
  main layout), use <androidx.drawerlayout.widget.DrawerLayout> as the root
  view.
- Add a <com.google.android.material.navigation.NavigationView> view to the
  layout.

5. Connect the drawer to the navigation controller:

- Open the activity that creates the navigation controller. (The main activity is typically the one you want here.) In onCreate(), use NavigationUI.setupWithNavController()to connect the navigation drawer with the navigation controller:

```
val binding = DataBindingUtil.setContentView<ActivityMainBinding>(

        this, R.layout.activity_main)

NavigationUI.setupWithNavController(binding.navView, navController)
```

6. Set up the drawer button in the app bar:

- In onCreate() in the activity that creates the navigation controller (which is typically the main activity), pass the drawer layout as the third parameter to NavigationUI.setupActionBarWithNavController:

```
val binding = DataBindingUtil.setContentView<ActivityMainBinding>(

    this, R.layout.activity_main)

NavigationUI.setupActionBarWithNavController(

    this, navController, binding.drawerLayout)
```

- To make the Up button work with the drawer button, edit onSupportNavigateUp() to return NavigationUI.navigateUp(). Pass the navigation controller and the drawer layout to navigateUp().

```
override fun onSupportNavigateUp(): Boolean {

    val navController = this.findNavController(R.id.myNavHostFragment)

    return NavigationUI.navigateUp(navController, drawerLayout)

}
```

Submission rules

- Lab must be submitted by April, 28th 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_lab2.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