Mobile Application Development 2021-22

A TimeBanking application

Lab2 – Show and edit user profile

Learning objectives

- Using GitHub to share the project code within the group
- Designing early UX layers
- Designing basic UIs: layouts, colors, icons, menus, responsiveness
- Managing the activity life cycle
- Using intents to exchange data between activities
- Loading, scaling, and rotating pictures
- Persisting data in the file system

Description

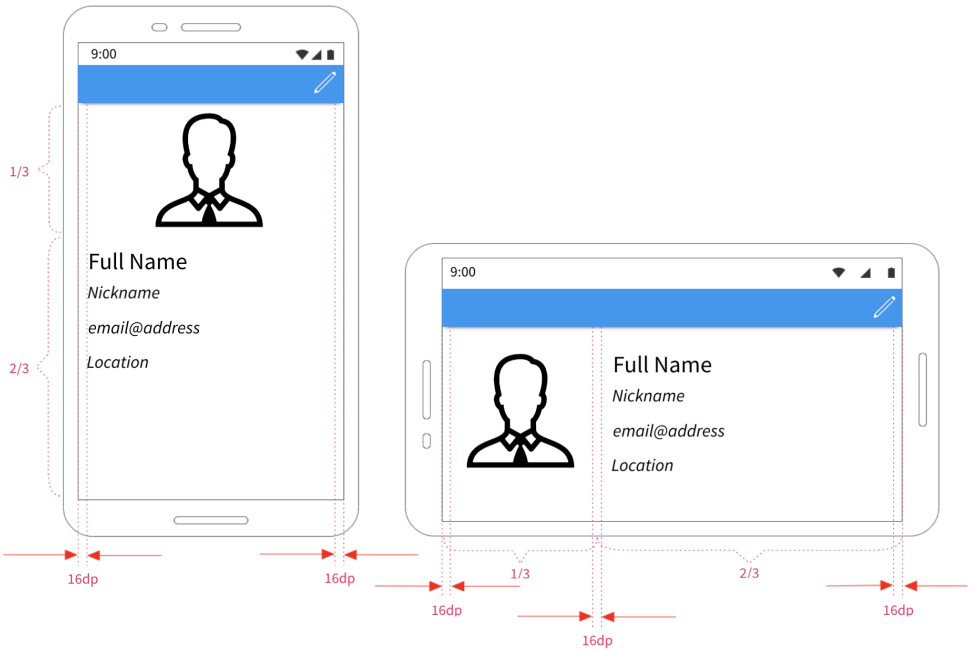
First of all you should define who your users are: their characteristics, objectives, needs, capabilities and limits. After that you can make a list of system requirements in order to meet the users objectives. At this point you will put the requirements in a relationship graph in order to start to design your information architecture. When the UX basics are defined, you can start with the implementation.
In order to participate in time banking, users must advertise their own skills, thus allowing others to take advantage of the offered time. Thus, the user profile represents a major section of the app.
You will create an activity that allows a user to display its own profile, and another by which the profile can be edited. The profile contains a picture and a basic set of data. When the app starts, it is in “view mode”, and it shows the current profile data. If no data is present, it shows some default content. A menu in the action bar allows the user to enter the “edit mode”. This has a similar layout as the previous one, but all fields are now editable. Moreover, touching the picture will trigger the camera app that allows the user to snap a new picture and return it to the app. Alternatively, the user can select the picture from the phone gallery. By pressing the back button when in “edit mode” all changes are persisted, the profile returns in “view mode” and all fields reflect the updated values.
The project will be kept on GitHub as a private repository.

Steps

1. Create an account on GitHub for each group member (if you do not have one, yet).
    a. Ask a single team member to create a new Android Studio project using the Empty activity template, choosing Kotlin as programming language
    b. Name the activity ShowProfileActivity. Make sure the “Generate Layout file” option is checked
    c. Select menu VCS → Share Project on GitHub and select the private option: this will create a new repository in the GitHub account
    d. Add all other group mates to the repository, granting them write privileges.
    e. Now other members can clone the project and cooperate: have care to manage push operations, not to overwrite existing code. Consider using a well known methodology like Git Flow (https://infinum.com/handbook/android/building-quality-apps/using-git)
2. Customize the ShowProfileActivity
    a. The purpose of this activity is to show a basic set of data about a user. This should comprise, at least, the following items:
        i. Photo (possibly a default image stored as a drawable resource)
        ii. Full Name (your real name)
        iii. Nickname (your chosen public identity)
        iv. E-mail address
        v. Location of the user (a string for now)
        vi. Skills (keyword) & Description (longer text)
        vii. Add any other extra data items as relevant to the application that your group is conceiving

b. Each piece of information should be maintained as a property of the activity and initialized with some static value (to be replaced later on)

c. Modify the basic layout as sketched in the following picture (remember that the colored app bar is generated programmatically, and it is not part of the layout file)



d. The app 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

e. Commit project. Push it on the remote repository

3. Following the guidelines in the menu Android guide (https://developer.android.com/guide/topics/ui/menus), create a menu resource file containing a single item, having a pencil icon, setting the “showAsAction” attribute to “always”

   a. In the activity kotlin file, override method onCreateOptionsMenu(...) and inflate the created menu. Try launching the app and verify that the icon properly shows in the right part of the app bar

   b. Override method onOptionsItemSelected(...) and add a simple reaction to the selection of the pencil item

   c. Launch your app and verify that everything works as expected

    d. Commit project. Push it on the remote repository

4. Create a second activity named EditProfileActivity that allows the user to edit his personal data

    a. The layout file will mimic the previous one with some exceptions: all *TextView*s will be replaced by corresponding *EditText*s, having their *inputType* attribute duly set to a suitable type of content, in order to support the data entry process; the ImageView will show an *ImageButton* on top of it, labeled with a camera icon

    b. If the user clicks on the *ImageButton* a floating context menu appears, showing the following options:

        i. Select an image from the phone gallery

        ii. Use the camera to take a picture

    c. The input fields must implement a smart keyboard

        i. Customizing the displayed keys to the data item to be entered (generic text, e-mail address, numbers, …)

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

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

5. Make the two activities communicating textual data via Intents

    a. In *ShowProfileActivity* add a private method named *editProfile()*. This will have to be invoked when the pencil button is pressed. Inside this method, create an explicit Intent targeting the *EditProfileActivity* class, and populate it with extra content (*intent.putExtra(...)*). Each extra item must have a name and the corresponding value. In order to reduce the risk of name clashes with existing keys, name your item out of your project package name, e.g. “groupXX.lab1.FULL_NAME”. Once the intent is fully decorated with the extra items, launch the other activity using method *startActivityForResult(...)*

    b. In method *onCreate(...)* of *EditProfileActivity* use the “intent” property to access the extra data which has been sent on invocation. Use these values to populate the various EditText boxes

    c. In order to make the *EditProfileActivity* return the updated values, an intent should be constructed and populated with the relevant data, when the back button is pressed. Override method onBackPressed(), making sure that it invokes the superclass one. Use the same key

names used for retrieving the initial values, for storing inside the newly created intent, the updated data. Once the intent is ready, invoke the method *setResult(...)* to define the outcome (Activity.RESULT_OK) and the intent carrying the extra data.

d. In *ShowProfileActivity,* override *onActivityResult(...).* Here you should check that the request code matches the one you have used when startActivityForResult(...) was invoked and that the result code is RESULT_OK: if this is the case, access the extra data in the returned intent and update all the *TextView*s

e. Run the application and check that everything works as expected

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

6. Update *EditProfileActivity* to allow the user to choose a different picture

   a. Carefully read the document https://developer.android.com/training/camera/photobasics, and use the contained guidelines to allow the user to launch the default camera application in order to snap a new picture and to return it to the application

   b. Use the returned bitmap to update the picture shown: if it is rotated, check the following document: https://stackoverflow.com/questions/14066038/why-does-an-image-captured-using-camera-intent-gets-rotated-on-some-devices-on-a

   c. Commit project. Push it on the remote repository

7. Persist all the information in the local file system, so that, when the app is started again, all edited content is retrieved

   a. Using the background information contained in the document https://developer.android.com/training/data-storage/shared-preference update the *ShowProfileActivity* in order to load, on startup, the profile data from *SharedPreferences* and to persist updated information whenever the *EditProfileActivity* returns updated data. The data must be serialized and deserialized using a *JSONObject* and the resulting String must be saved with the key *profile*.

   b. The user profile image must be saved to the local filesystem, following the guidelines contained in the document https://developer.android.com/training/data-storage/app-specific.

   c. Commit project. Push it on the remote repository

Summary

Activities

- An `Activity` is an app component that provides a single screen focused on a single user task.
- Each `Activity` has its own user interface layout file.
- You can assign your `Activity` implementations a parent/child relationship to enable Up navigation within your app.
- Activities may have an option menu which is located in the left side of the toolbar

Intents

- An `Intent` lets you request an action from another component in your app, for example, to start one `Activity` from another. An `Intent` can be explicit or implicit.
- With an explicit `Intent` you indicate the specific target component to receive the data.
- With an implicit `Intent` you specify the functionality you want but not the target component.
- An `Intent` can include data on which to perform an action (as a URI) or additional information as `Intent` *extras*.
- `Intent` *extras* are key/value pairs in a `Bundle` that are sent along with the `Intent`.

SharedPreferences

- The `SharedPreferences` class allows an app to store small amounts of primitive data as key-value pairs.
- Shared preferences persist across different user sessions of the same app.
- To write to the shared preferences, get a `SharedPreferences.Editor` object.

- Use the various "put" methods in a `SharedPreferences.Editor` object, such as `putInt()` or `putString()`, to put data into the shared preferences with a key and a value.
- Use the various "get" methods in a `SharedPreferences` object, such as `getInt()` or `getString()`, to get data out of the shared preferences with a key.
- Use the `clear()` method in a `SharedPreferences.Editor` object to remove all the data stored in the preferences.
- Use the `apply()` method in a `SharedPreferences.Editor` object to save the changes to the preferences file.

Paper Prototyping

- Even if Android Studio gives you very good tools for designing your app's UI, it's always important to start from a paper sketch
- Try to make a simple sketch of all the activities you're planning to design for your application, taking note of all the expected elements of the UI, for each requested feature
- Having a design/brainstorming phase with all the application's screens on a global paper scheme, will help you A LOT in the following phases, to avoid user experience problems - which always translate in unwanted redesigning/re-coding for your activities

Submission rules

- You have to submit your work by next lab
- 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