CAT 1 SOLUTION- 2022-223

Mobile App Development (Universitat Oberta de Catalunya)

CAT2

Presentation

In this Continuous Assessment Test (CAT), we will create the front-end of a simple Android app using Android Studio and the Kotlin programming language.

Competencies

This CAT will develop the following competencies of the Bachelor's degree in Techniques for Software Development:

- Adapt to new software development technologies and to future environments, updating professional skills.
- Design and build computer applications using development, integration and reuse techniques.
- Develop cross-platform applications.

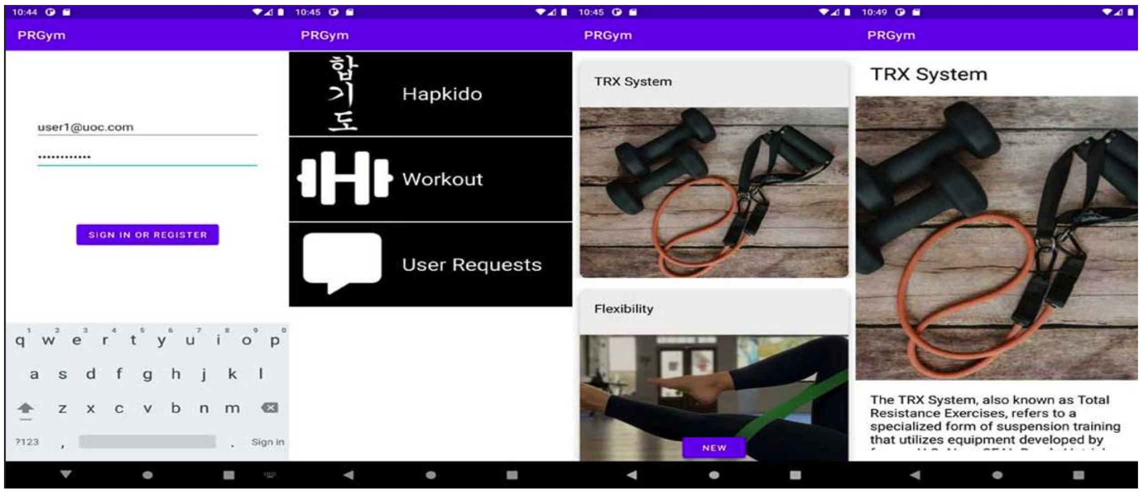
Objectives

The learning outcomes of this CAT are the following:

- Install and use an IDE (integrated development environment) for mobile app development (Android Studio).
- Code using a programming language for developing native mobile apps (Kotlin).
- Design the architecture of a mobile app (activities, fragments, intents).
- Design the user interface of a mobile app (layouts, views).
- Debug and test mobile apps.
- Develop the front-end of a simple mobile app.
- Learn different strategies and frameworks for cross-platform mobile app development.

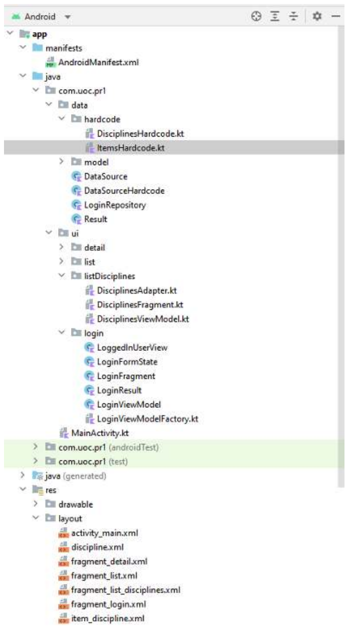
Problem statement

The set of continuous assessment activities in this course will lead us to build a mobile app for a gym. In this app, users can log in and then access contents related to the sport disciplines they practice. Users will also be able to send queries and requests to the gym staff. Finally, in the last part of the course we will implement a specific feature: face-to-face outdoor meetups using geolocation to perform outdoor sport activities.



The TRX System, also known as Total Resistance Exercises, refers to a specialized form of suspension training that utilizes equipment developed by

In this CAT2, we will focus on building an app using fragments and designing its user interface. Your submissions will consist in a .ZIP file with the compressed project, answering the questions and implementing the features requested in the different exercises of this activity.



```
Android
app
  manifests
    AndroidManifest.xml
  java
    com.uoc.pr1
      data
        hardcode
          DisciplinesHardcode.kt
          ItemsHardcode.kt
        model
        DataSource
        DataSourceHardcode
        LoginRepository
        Result
      ui
        detail
        list
        listDisciplines
          DisciplinesAdapter.kt
          DisciplinesFragment.kt
          DisciplinesViewModel.kt
        login
          LoggedInUserView
          LoginFormState
          LoginFragment
          LoginResult
          LoginViewModel
          LoginViewModelFactory.kt
      MainActivity.kt
    com.uoc.pr1 (androidTest)
    com.uoc.pr1 (test)
  java (generated)
  res
    drawable
    layout
      activity_main.xml
      discipline.xml
      fragment_detail.xml
      fragment_list.xml
      fragment_list_disciplines.xml
      fragment_login.xml
      item_discipline.xml
```

In this CAT, you will become familiar with the architecture of the app and the user interface. To this end, we provide a *template project* which provides the basic architecture. The model will be *hard-coded* in the template. This means that its data will be defined within the source code. This approach is used frequently in the initial steps of a mobile app development project, when you are evaluating a prototype.

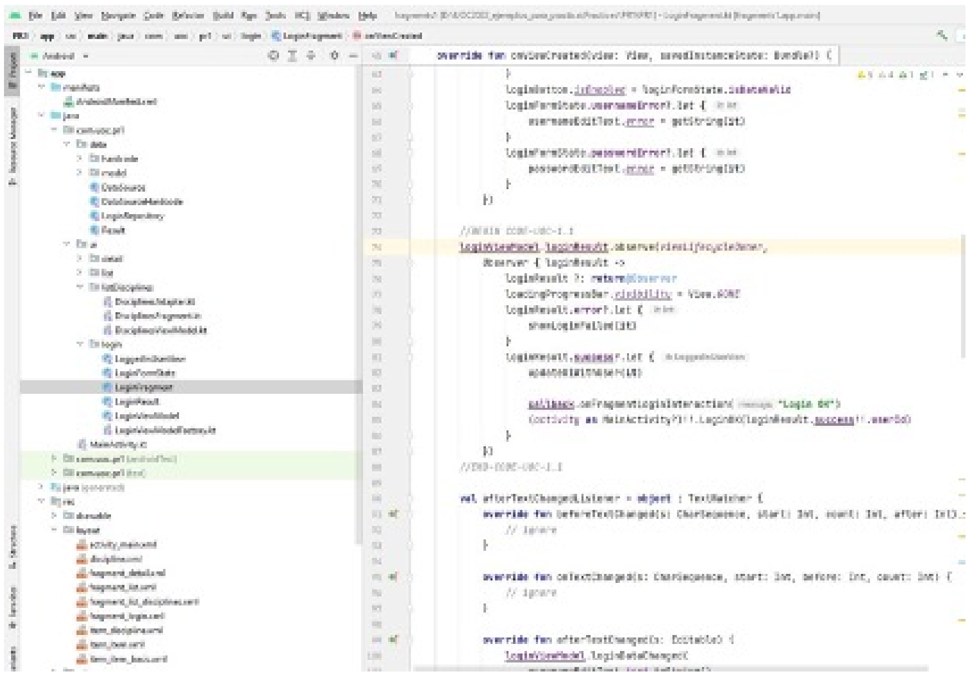
You should also become familiar with the project browser. This tool will be used frequently throughout all the course. Thus, we encourage you to browse different folders in the project to understand what type of elements can be found within each one.

When running the app, you will see a login prompt. You use any email with a valid syntax (`nnnnn@dddd.ccc`) as username. The password needs to have more than 5 characters (any combination of a suitable length will work). After the login screen, you will see a list of items. The interface and behavior of this app will evolve as you complete the different exercises.

Important: When you open the project for the first time, a prompt may ask if you want to replace the SDK in a given path with the SDK in your computer. You should answer "Yes".

1. **Login screen** (Weight: 20%)

   The login screen is the first element of our app that will require accessing data. Remember that, in this activity, all the relevant data is hard-coded.



```
    override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
            }
            loginButton.isEnabled = loginFormState.isDataValid
            loginFormState.usernameError?.let {
                usernameEditText.error = getString(it)
            }
            loginFormState.passwordError?.let {
                passwordEditText.error = getString(it)
            }
        })

    //BEGIN-CODE-UOC-1.1
    loginViewModel.loginResult.observe(viewLifecycleOwner,
        Observer { loginResult ->
            loginResult ?: return@Observer
            loadingProgressBar.visibility = View.GONE
            loginResult.error?.let {
                showLoginFailed(it)
            }
            loginResult.success?.let {
                updateUiWithUser(it)

                callback.onFragmentLoginInteraction( "Login OK")
                (activity as MainActivity?)!!.LoginOK(loginResult.success!!.userId)
            }
        })
    //END-CODE-UOC-1.1

    val afterTextChangedListener = object : TextWatcher {
        override fun beforeTextChanged(s: CharSequence, start: Int, count: Int, after: Int) {
            // ignore
        }

        override fun onTextChanged(s: CharSequence, start: Int, before: Int, count: Int) {
            // ignore
        }

        override fun afterTextChanged(s: Editable) {
            loginViewModel.loginDataChanged(
```

   (a) In the class `LoginFragment`, locate the callback `onViewCreated`. Add a comment that explains in 5 lines or less what the block of code between `//BEGIN-CODE-UOC-1.1` and `//END-CODE-UOC-1.1` is doing.
   Clue: Notice that the code is observing the property `loginResult` of our model. What is it doing?

   (b) In the same callback, the code block between the comments `//BEGIN-CODE-UOC-1.2` and `//END-CODE-UOC-1.2` implements the response to clicking the login button. Add a comment that explains in 5 lines or less what actions are performed when the button is pressed. Which method and class is being used? Why?
   Clue: When you move your mouse over a method call while pressing the `Ctrl`, pressing the left button of the mouse takes you to the method declaration.

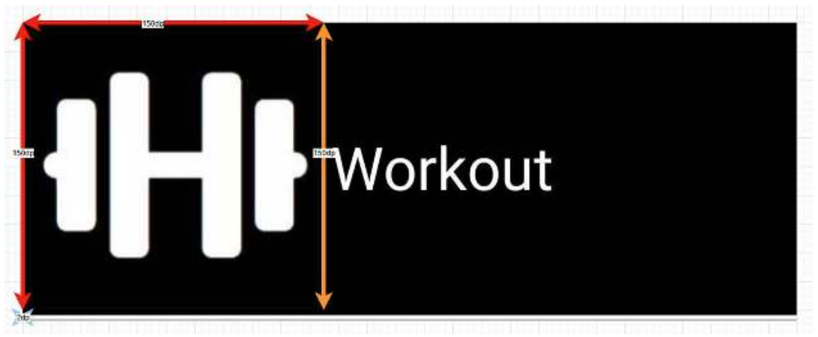
   (c) Implement the following behavior: whenever we execute the `login` method from class `DataSourceHardcode`, if the username is `"user1@uoc.com"` you should return the user

User(1, "Jane Doe"). Otherwise, you should return the user User(2, "John Doe"). Your code should be placed between the placeholder comments //BEGIN-CODE-UOC-1.3 and //END-CODE-UOC-1.3.

2. **List of disciplines** (Weight: 20%)

   In this exercise, we will modify how the list of disciplines is presented to users.

   (a) We ask you to modify the layout file item_discipline.xml to achieve the following design:

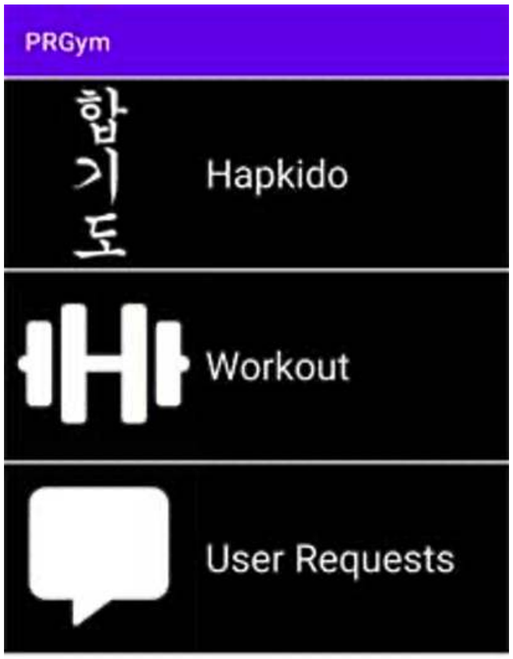
   

   The picture will be set to the left and will measure 150dp x 150dp. The image inside this part should be visible in full and centered. To the right of this image, we find a text with a height of 150 dp, white color and font size 30dp. Moreover, this text should be vertically centered. On the bottom, there will be a small horizontal line that is 2dp wide.

   The id of the ImageView must be "@+id/item_image". The id of the TextView must be "@+id/item_text".

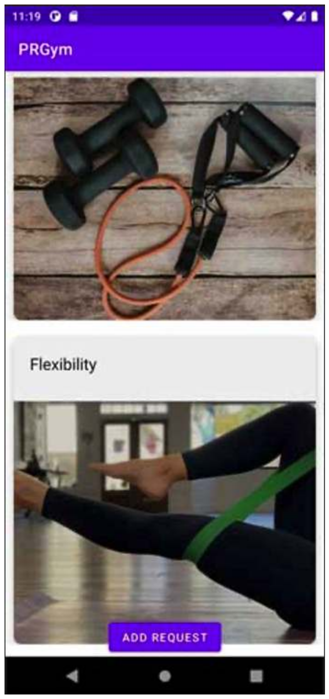
   Clue: An horizontal line can be seen as a View with a fixed height and a given background color.

   (b) After performing the previous changes, now we can uncomment the code between the comments //BEGIN-CODE-UOC-2.2 and //END-CODE-UOC-2.2 in the bind method of class DisciplinesAdapter. Test that the final result looks like the following:



3. **Add request button** (Weight: 10%)

   Modify the layout of the list of items of a discipline to add a button to the bottom area of the screen. This button should be visible over the list of items. The text of this button should be `"Add request"`. Moreover, create the callback that manages button clicks for this button.



Important: If you press an element in the list of items, there is a transition that changes from the list of items to a detailed view of the selected item. You can take a look at this code to learn how this transition is implemented.

4. **Items in a discipline** (Weight: 50%)

   In this exercise, we will focus on the items of a discipline. In the previous exercise, all items were of type IMAGE. Now, we want to add items of a different type (BASIC) that will only include textual information. This means that it will not be necessary to fill in the attributes that refer to the image (even if they have information, it will be ignored when processing items of type BASIC).

   We will break this complex task into several smaller subtasks:

   (a) Modify the second item in class `ItemsHardcodeDiscipline1` and change its type to `ItemType.BASIC`.

(b) Create a new layout file called `item_item_basic.xml` that is a copy of `item_item.xml`. Remove the ImageView from this layout. Then, change the `id` of the CardView to `"@+id/card_view_basic"`.

(c) Modify the callback `onCreateViewHolder` in `ItemsAdapter`. It should create a layout of type `R.layout.item_item` or `R.layout.item_item_basic` depending on the `viewType` it receives as a parameter. All the code should be placed between the comments `//BEGIN-CODE-UOC-4.3` and `//END-CODE-UOC-4.3`.

(d) Modify the method `bind` so that it only fills the `itemTextView` if the item is of type BASIC. All the code should be placed between the comments `//BEGIN-CODE-UOC-4.4` and `//END-CODE-UOC-4.4`.

(e) At the end of exercise 3, we highlighted how pressing an item in the list of disciplines starts a transition to switch from the list to the details about the item. Obviously, if an item is of type BASIC, this transition is not possible because there is no image!

Modify the method `openDetail` in class `MainActivity`. If it receives a BASIC item, it should only add it to the commit of the supportFragmentManager. All the code should be placed between the comments `//BEGIN-CODE-UOC-4.5` and `//END-CODE-UOC-4.5`. This code should:

i. Hide the fragment `f2` (the list of items).
ii. Add in the `R.id.placeholder1` the fragment `f3` (the details about the item).
iii. Use `addToBackStack` to add the transaction of fragments to the stack.

(f) In the class `DetailFragment`, modify the callback `onCreateView` to delete element `v2:ImageView` from its parent if the item is not of type IMAGE. All the code should be placed between the comments `//BEGIN-CODE-UOC-4.6` and `//END-CODE-UOC-4.6`. Considering `var v2: ImageView = v.findViewById(R.id.item_image_detail)`, this code should:

i. From `v2`, access its parent.
ii. Cast it to `ViewManager`.
iii. Use method `removeView` from class `ViewManager`.

Resources

Basic Resources

- Course Wiki: Section 4 - Android app architecture
- Course Wiki: Section 5 - User interfaces

Additional Resources

- Official Android developer documentation
- Official Android Studio documentation

Assessment criteria

- All exercises in this CAT must be solved **individually**.
- Each exercise has the following weight:
  1. 20%
  2. 20%
  3. 10%
  4. 50%

Submission format and deadline

You must submit a PDF file including the answers to written exercises. The name of the file you submit must be **CAT1_SurnameName.zip** and you should include your full name within the PDF. This document must be submitted in the **Continuous Assessment Record** of your classroom **before 23:59 of 2022/11/09**. **Late submissions will not be accepted.**