CAT3

Presentation

In this Continuous Assessment Test (CAT), we will store information from our mobile app in our mobile device.

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:

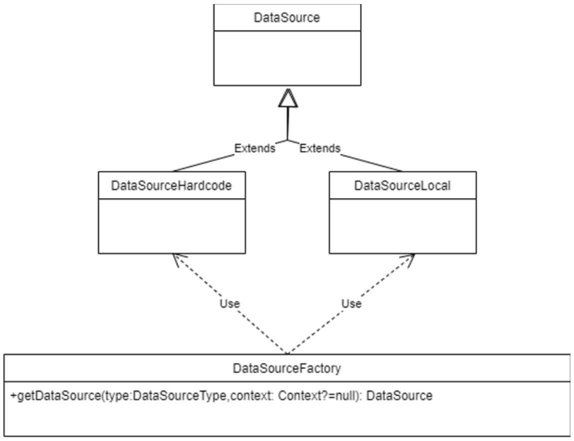
- Code using a programming language for developing native mobile apps (Kotlin).
- Learn how to use advanced features of the Kotlin language.
- Use database managers for mobile apps (Room, SQLite).
- Design how events are managed within the mobile app.
- Use different technologies for storing persistent information locally in the mobile device.
- Learn platform-independent technologies for storing and exchanging data (JSON, XML).

Problem statement

In this continuous assessment test we will expand the seminar application that we developed in the previous CAT. This time we will evolve the model from hard-coded to a local SQLite database. We will also work with the local file system, ViewModels, activity usage and tests.

To preserve consistency with the previous CAT, we will use a software design pattern called Factory. A design pattern is a standard way to design classes to perform common tasks within software development. In our case, the goal is to isolate the code of the activity and the fragments from the type of data model we are using.

To achieve this, we will create a `DataSource` class that defines standard methods for accessing data. In the previous CAT we implemented `DataSourceHardcode` that stored the data within the code. Now, we will be implementing a `DataSourceLocal` class which will allow us to access data stored in a local SQLite database. Our factory is the class `DataSourceFactory` that has a `getDataSource` method that allows us to get either a `DataSourceHardcode` or a `DataSourceLocal` class depending on the parameter we provide when invoking the method.



In this CAT we will start from a new code template that is provided together with this problem statement. The organization of the template is similar to the one in the previous CAT, but remember to use this new template as a starting point for your submission! The database for our app has two users: `user1@uoc.com` and `user2@uoc.com`, both with password: `123456`.

You will have to submit a .ZIP file with the project, including the code you have added to solve the different exercises in this CAT.

1. **Create the local database** (Weight: 10%)

   First we will create the local database if it does not exist. To do this we will modify some methods of the `DbHelper` class found within the `data\localstorage` folder.

   (a) Locate the comments `//BEGIN-CODE-UOC-1.1` and `//END-CODE-UOC-1.1` in the class `DbHelper`. Insert the code to create the local database using the following `CREATE` SQL commands:

   - Create the user table.
   - Create the seminar table.
   - Modify the `SQL_CREATE_item` string to add the field `item_correct_answer` of type `INTEGER` and `item_answer1`,`item_answer2`,`item_answer3`,`item_answer4` of type `TEXT` as the end fields of the `CREATE` statement. Then, create the Item table using the previously modified `SQL_CREATE_item string`.
   - Create the user-seminar mapping table.

   This class already offers the properties `SQL_CREATE_user`, `SQL_CREATE_seminar`, `SQL_CREATE_item`, `SQL_CREATE_user_seminar` that contain the necessary SQL strings.

   (b) We will then upload the initial data from the tables. To do so, insert your code between the comments `//BEGIN-CODE-UOC-1.2` and `//END-CODE-UOC-1.2`. Traverse the list `command_list` and perform the inserts.

2. **Local filesystem** (Weight: 10%)

   After the comment `//END-CODE-UOC-1.2` you will find a code snippet between the comments `//BEGIN-CODE-UOC-2` and `//END-CODE-UOC-2`.

   At this point, in the code, insert a comment to answer the following questions:

   (a) Explain in detail what is being done in that snippet in the specific case of our application. (5-lines)

   (b) What folder structure is created? (2-lines)

   (c) In what quality are the images stored in the file system? (2-lines)

3. **Implement the class DataSourceLocal** (Weight: 25%)

   Let us start expanding our Factory model that we already started in the previous CAT. We have created a new `DataSourceLocal` class that is going to inherit from `DataSource` but instead of returning the hard-coded data, it will read data from the database that we have created in the previous exercises. Class `DataSourceLocal` is already created in the template that we provided, but we will need to fill in some blanks. To this end, we will use cursors to get the answer to each query.

(a) Locate the comments //BEGIN-CODE-UOC-3.1 and //END-CODE-UOC-3.1 within the login() method of class DataSourceLocal. Use the following select instruction to verify the login:

Comment the following lines of code:

```
// var user_harcoded:User = User(1, "Jane Doe")
// result = Result.Success(user_harcoded)
```

Use instead the following SQL string:

```
val str_sql = "SELECT * FROM user WHERE user_username=$username
and user_pwd=$password"
```

This string is already defined within the method.

(b) Locate the comments //BEGIN-CODE-UOC-3.2 and //END-CODE-UOC-3.2 within the selectSeminarsUser() method. Use the following SELECT statement to upload a user's seminars.

```
SELECT * FROM seminar, user_seminar
WHERE  usersem_user_id= $user_id and usersem_seminar_id = sem_id
```

Load all the class attributes you'll need in later code. Remember what you used in CAT2.

The goal of this cursor is to populate the property userSeminarList.

Fill in the seminar image_path attribute with the path, considering that the path is built by concatenating the following strings:

context.filesDir.path + "/media/seminar/" + sem_id + ".jpg"

(c) Locate the comments //BEGIN-CODE-UOC-3.3 and //END-CODE-UOC-3.3 within the selectItems method. Use the following SELECT statement to load items from a seminar:

```
SELECT * FROM items WHERE item_sem_id= $id
```

Load all the class attributes you'll need in later code. Remember what you used in CAT2.

4. **Use the class DataSourceLocal** (Weight: 10%)

Change the code to use the new DataSourceLocal class instead of the DataSourceHardcode class we used in the previous CAT. In the DataSource file, locate the DataSourceFactory and change the Default to DataSourceType.Local. This code can be found between the comments //BEGIN-CODE-UOC-4 and //END-CODE-UOC-4.

At this point, in the code, insert a comment to answer the following questions:

(a) Why can the implementation of the method `getDataSource` return an element of type `DataSourceLocal` or `DataSourceHardcode` if its declaration returns an element of type `DataSource?`. (2-lines)

(b) Would adding more `DataSource` kinds affect the rest of the program, for example, the user interface? (2-lines)

(c) Which other local `DataSource` kind could we add? (2-lines)

5. **Read files from the filesystem** (Weight: 10%)

Now we are going to modify the adapters to read the images from the file system. Use the method `BitmapFactory.decodeFile` to load the bitmap from the file system using the `item.image_path` value as a parameter and assign the `itemImageView` bitmap.

(a) Modify the `bind` method of `ItemViewHolder` (between the comments `//BEGIN-CODE-UOC-5.1` and `//END-CODE-UOC-5.1`) read the image from the attribute `image_path`

(b) Modify the `onCreate` callback from `QuizFragment` to read the image from the attribute `image_path` (between `//BEGIN-CODE-UOC-5.2` and `//END-CODE-UOC-5.2`).

6. **Create an activity AddSeminarActivity** (Weight: 25%)

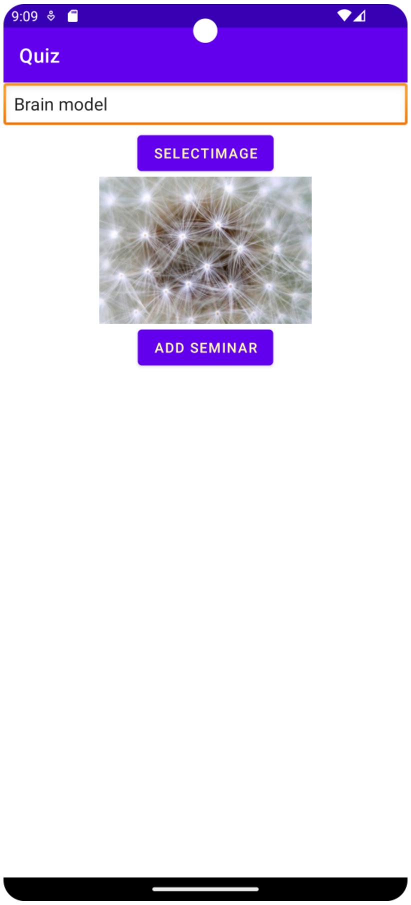
We are going to create a new activity where users can add a new seminar by pressing the `New` button.

Due to the architecture of our application, it would be more appropriate to use a fragment but for didactic reasons we will use an activity. The template does not include this activity.

(a) Create a new Activity named `AddSeminarActivity` by right-clicking the package `"com.uoc.pr1"` from the project and select: (New - Activity - Empty Views Activity). This will open a screen where you can add the Activity Name and the source Language you want to use (we will be using **Kotlin**).

Press "Finish". This will create two files: the `AddSeminarActivity` class and the `activity_add_seminar` layout.

(b) In the layout file `activity_add_seminar.xml`, create the following design (*tip*: Use a vertical LinearLayout to ease the creation of the screen).



Brain model

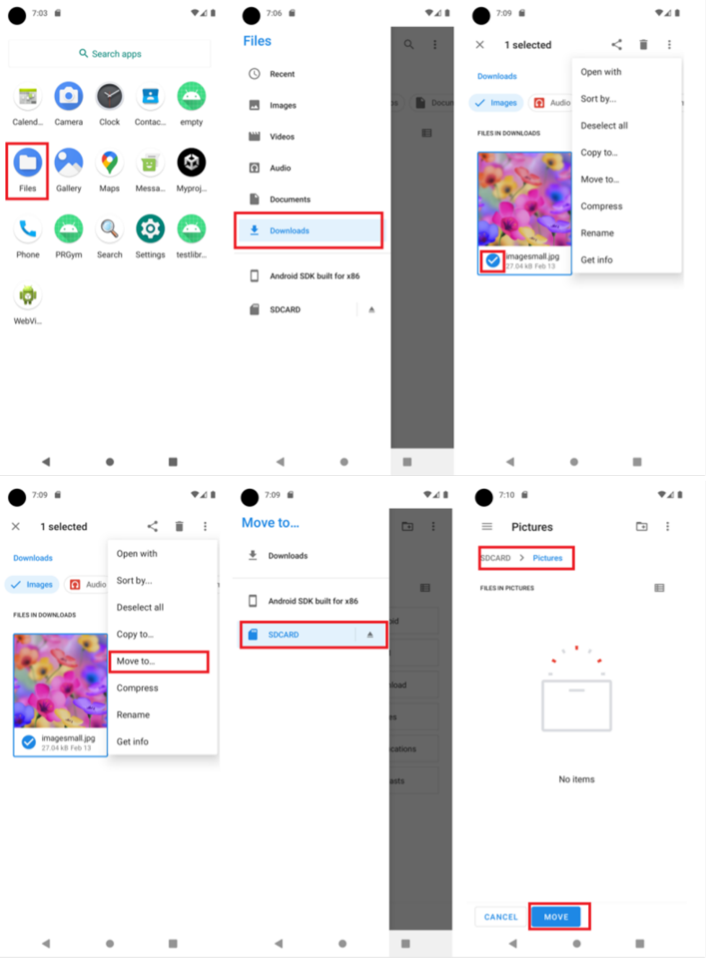
The title must be `@+id/editTitle`. Use the id: `btnNew` for the New Button and `btnSelectImage` for the select image button.

(c) In the listener of the button `btnNew`, which corresponds to the Add Request text, within the class `SeminarFragment`, open the activity `AddSeminarActivity`. Create the corresponding intent and use the method `getResult.launch(intent)` to launch a new activity and then collect the result within `//BEGIN-CODE-UOC-6.5`. Since you are responding to an event from a button when you create the Intent, you must use `this.context`. Write your code between the comments `//BEGIN-CODE-UOC-6.2` and `//END-CODE-UOC-6.2`.

(d) In the class `AddSeminarActivity`, implement the listener for button `btnSelectImage` and use the code from the course wiki to open the system activity that allows you to obtain an image from the gallery (section 4.1.3 from the wiki). Please, add the comments `//BEGIN-CODE-UOC-6.3` and `//END-CODE-UOC-6.3` wrapping your code.

To test it, you can easily upload images to the emulator by dragging the image from your computer to the emulator home screen. That image will go to the `Download` folder of the emulator file system.

Open the `Files` application on the emulator and follow these steps to move the image to the gallery:



Remember that you should only upload small images to avoid hindering the performance of your emulator.

(e) In the same class, implement the listener for the button `btnNew`. This button returns the result and ends the activity. Please, add the comments `//BEGIN-CODE-UOC-6.4` and `//END-CODE-UOC-6.4` wrapping your code. You can use the class `AddSeminarActivity` to create the extra parameters like this:

```
val l:AddSeminarResult =
AddSeminarResult(binding.editTitle.text.toString(),uri)
```

(f) In the code that receives the results in class `SeminarsFragment`, access the `DataSource` using the `DataSource.getDataSource` factory and add the seminar using the `addSeminary(title,uri)` method. To make the call you will need to make the appropriate casts with the operators `?` and `!!`. Place your code between the comments `//BEGIN-CODE-UOC-6.5` and `//END-CODE-UOC-6.5`.

(g) For the `DataSourceLocal` `addItem` method to work you must:

i. Make the insert in the database. Simply run the insert. `//BEGIN-CODE-UOC-6.6.1` and `//END-CODE-UOC-6.6.1`

ii. Copy the file from its location to the file system of your application. The path to the source file is the uri that we receive as a parameter.
The path to the target file is built as follows:

```
val directory_path = this.context.filesDir.path +
    "/media/seminar/"
image_path = directory_path + new_id + ".jpg"
```

You can see an example of related code in the method `MoveResources` from class `DbHelper` in `data\localstorage` (see exercise 2). `//BEGIN-CODE-UOC-6.6.2` and `//END-CODE-UOC-6.6.2`

iii. Below you will find the comments `//BEGIN-CODE-UOC-6.6.3` and `//END-CODE-UOC-6.6.3`. Explain between these two marks what is being done in this fragment. *Clue:* Check how the behavior of the app changes when we add an item if we comment this code.

7. **Testing** (Weight: 10%)

The performance of unit tests is very important in the verification of methods that do not involve the user interface.

Create a unit test for the method `seminarRecommendation` from class `DataSource`. This method will recommend a seminar based on parameters such as duration, our skills and whether we have previous experience or not. We want to test it using JUnit4.

First, we want to test it with the following parameters: `duration = 30, skills = 'IT', experience = false` and `result = "advanced"`

Add a comment inside the class created for this test. Inside this comment, you should record the result shown by Android Studio where you can check whether the result is correct or incorrect. In the same comment, also answer the following question: Do we need to run the emulator to perform this type of test? Why?

Resources

Basic Resources

- Course Wiki: Section 6 - Debugging and testing
- Course Wiki: Section 7 - Advanced Kotlin features
- Course Wiki: Section 8 - Events
- Course Wiki: Section 9 - Local data persistence

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. 10%
  2. 10%
  3. 25%
  4. 10%
  5. 10%
  6. 25%
  7. 10%
- The use of artificial intelligence tools is not allowed in this activity.

  The course plan and the UOC's site on academic integrity and plagiarism have information on what is considered misconduct in assessment activities and the consequences this may have.

Submission format and deadline

You must submit a ZIP file including the project (answers to textual exercises should be included in comments within the code, as described for the instructions of each exercise). The name of the file you submit must be **CAT3_SurnameName.zip**. This document must be submitted in the **Continuous Assessment Record** of your classroom **before 23:59 of 2025/04/13**. **Late submissions will not be accepted.**