CAT4

Presentation

In this Continuous Assessment Test (CAT), you will learn how to connect a mobile app to a back-end that provides data and services.

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

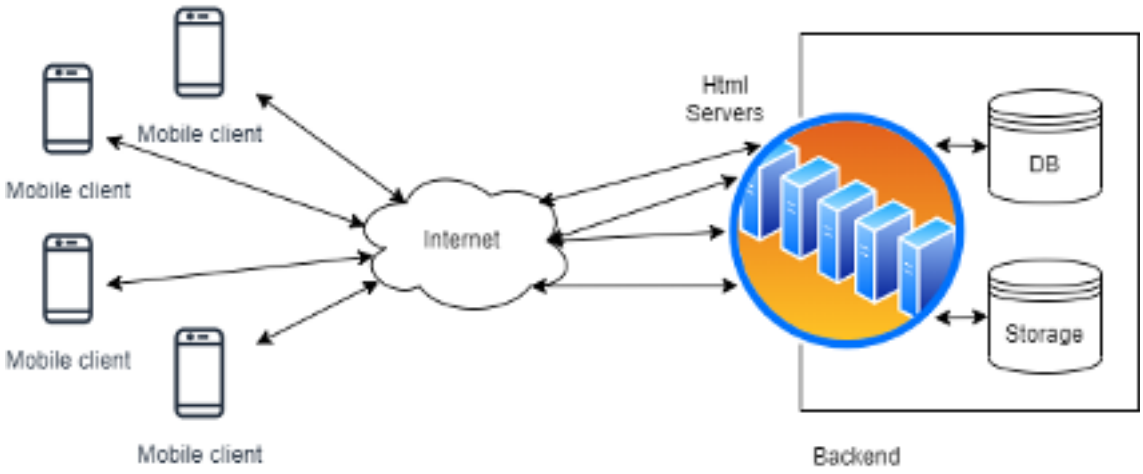
The learning outcomes of this CAT are the following:

- Code using a programming language for developing native mobile apps (Kotlin).
- Use an external back-end provider (Firebase).
- Connect the front-end of your mobile app to a back-end.
- Develop a custom proprietary back-end.
- Learn different strategies to implement a back-end for your app.

Problem statement

In this continuous assessment test we will continue working on the gym application. However, this time we will move from a hard-coded data model (CAT2) or local database (CAT3) to a backend-based model. Thus, a new code template is provided to be used as the starting point of this activity, which works in the same way as in previous CATs.

The back-end includes all the services or data offered by the online part of an application. Most applications require back-end functionality. These services or data may be core features or simply auxiliary ones, such as gathering analytics about app usage.



We will continue to maintain consistency with previous assessment tests by using the Factory design pattern. In this CAT, we are going to implement the methods of the `DataSourceFirebase` class. The `DataSourceFactory` class will now default to this `DataSourceFirebase` class.

Firebase is the back-end provided by Google for easy mobile app development. It provides a wide variety of back-end features. Firebase aims to cover all the features required by a mobile app so that the application developer does not need to develop a custom proprietary back-end. In this particular CAT, we are only going to use two features, **Firestore Database** and **Storage**. Firestore Database is a flexible and scalable NoSQL database. Meanwhile, Storage offers a disk space for storing files.

We ask you to submit the same **compressed project in .zip format** implementing the requested

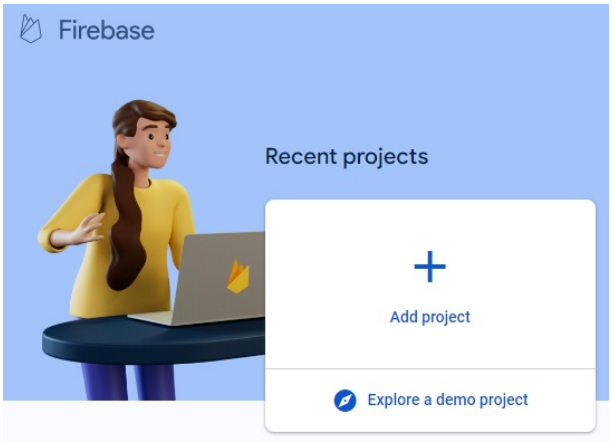
features and **a .PDF file** with the answer to theoretical exercises. The theoretical exercises have been highlighted with the text "(THEORY)" at the beginning of the question.

1. **Firebase environment preparation** (Weight: 10%)

   From a browser, we access the URL `https://firebase.google.com/`, click on "`Get started`" and create a new project. **Important:** We must use a personal Google account. A UOC account could not be considered valid by Google.

   Explain and add screenshot in the PDF file to let us evaluate the process you have followed.

   (a) Add Project

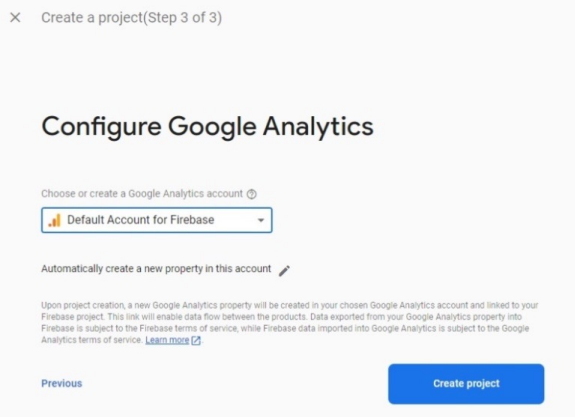


   We are going to add two things to this project:

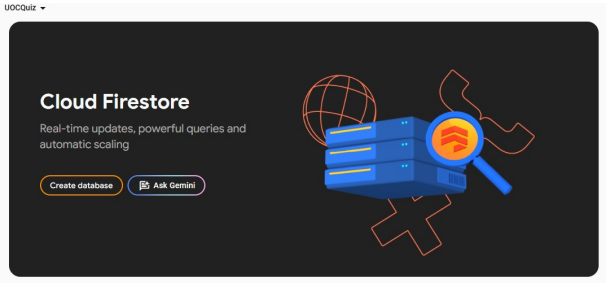
   i. The back-end functionalities we will need.
   ii. The applications that will use it.

   Once we click on "`Add`", we will open an assistant. In the first step, the assistant will request two pieces of information:

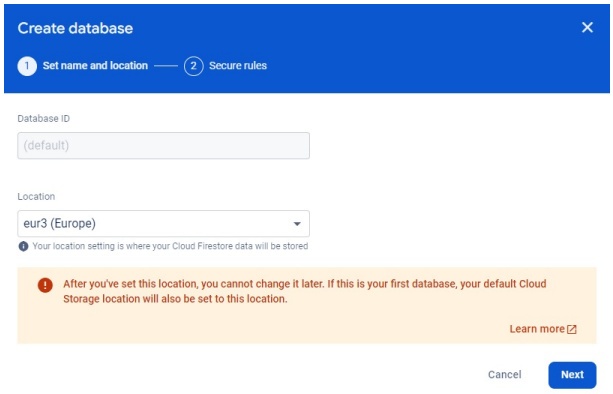
   i. The project name, "`UOCQuiz`".
   ii. Leave Gemini active on Firebase.
   iii. Then, it will ask if we are going to use Google Analytics in this project. This step is very important because Google Analytics allows us to gather metrics that can help us improve the functionality of our app. We can also run small snippets of code when Analytics detects that a condition is met. As an Analytics account, we can use the default one for Firebase.



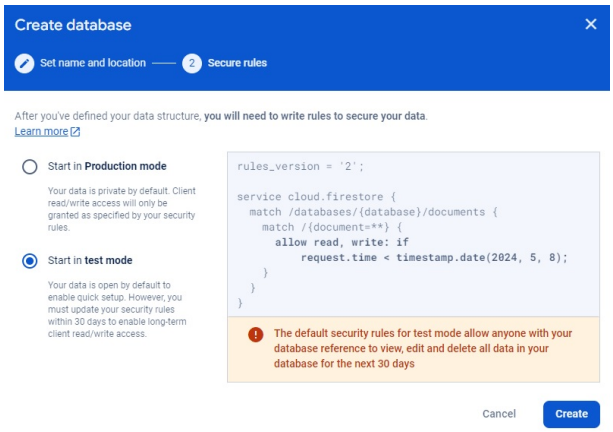
After a few moments, a page opens where we can add the functionalities. We open the menu on the left ("`Build`") and select "`Firestore Database`". This opens a new page, where we click on the "`Create database`" button.



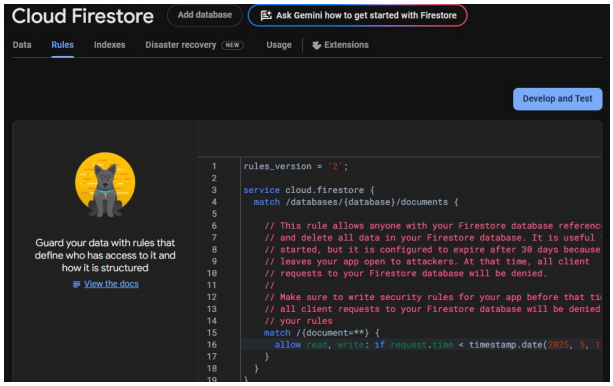
In the first step, we will select the location of the data center where our back-end will store our data.

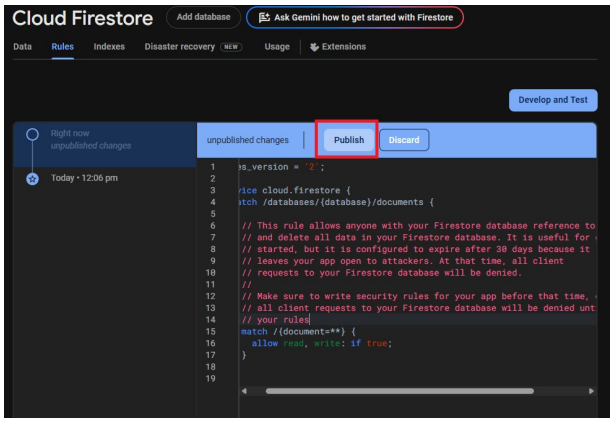


In the next step, Firebase asks whether we want to put the database into production or if it will be a test database. This basically affects safety rules. We will start by creating the database in test mode and we can change this decision later. This will create a single security rule that will only control the amount of time that the database has been in test mode. If this counter exceeds one month, we will no longer be able to connect to the database unless we modify the security rule again.

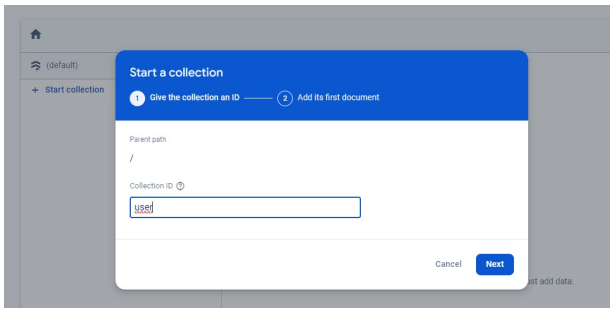


(b) Let's go to the rules tab to change the rule so we can always access the DB.





(c) Let's go back to the Data tab. Data is organized into collections, and within each collection we have documents. We start by creating our first collection, which we will call "`user`". To create the collection, click on the "`+ Start collection`" button and put "`user`" in the `Collection ID` field.

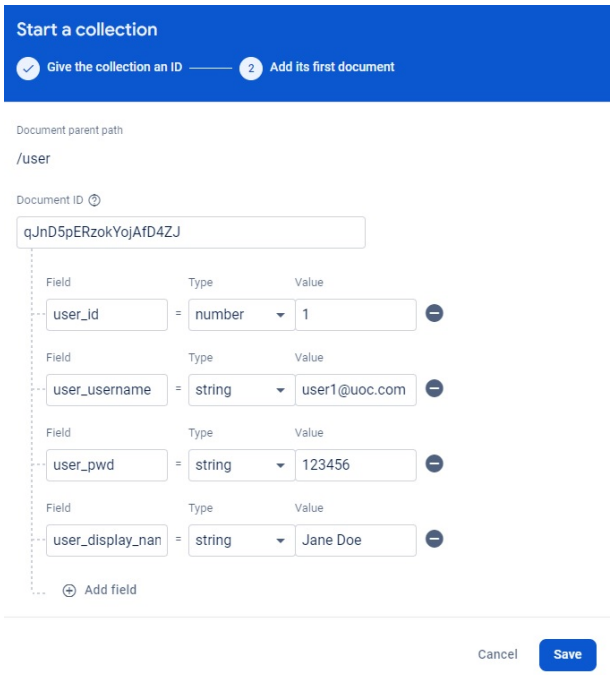


Then, we have to click the Auto-ID button to provide an automatic identifier to the document. After that, we will manually fill in the data about the first user using the same values that we provided in the following `INSERT` statement from the previous CAT:

```
INSERT INTO 'main'.'user'
  ('user_id', 'user_username', 'user_pwd', 'user_display_name')
  VALUES ('1', 'user1@uoc.com', '123456', 'Jane Doe')"
```

To determine the type, look at the string `SQL_CREATE_xxxx` in the same file. In the first case, look at the `SQL_CREATE_user`

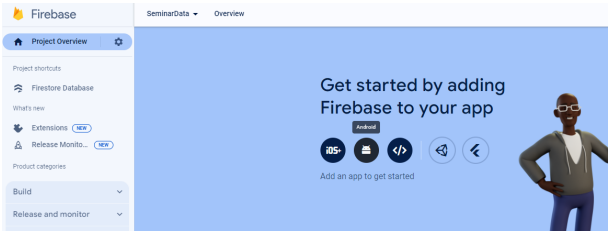
```
    private val SQL_CREATE_user = """
CREATE TABLE "user" (
"user_id" INTEGER,
"user_username" TEXT,
"user_pwd" TEXT,
"user_display_name" TEXT,
PRIMARY KEY("user_id")
);
```



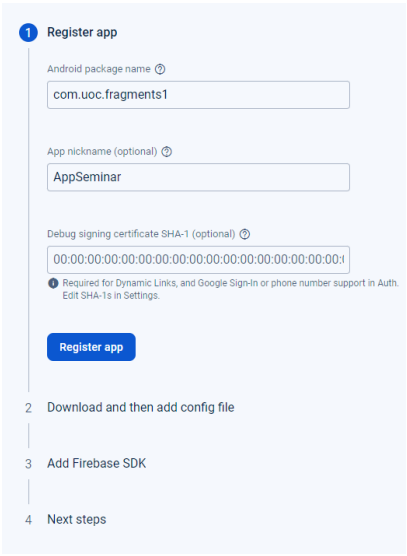
(d) Similarly, we also need to create the `seminar` collection. You can copy the values from the seminar INSERT statements from the `DbHelper` project class. You must also add the field `sem_image_url(string)` with values: `sem_id = 1 -> sem_image_url = "https://www.revolumedia.com/uoc/img/introduction_mobile.jpg" sem_id = 2 -> sem_image_url = "https://www.revolumedia.com/uoc/img/mobile_environment.jpg"`

(e) Finally, we need to create the `user_seminar` collection. The values to be used to populate this collection are in the `user_seminar` INSERT statements.

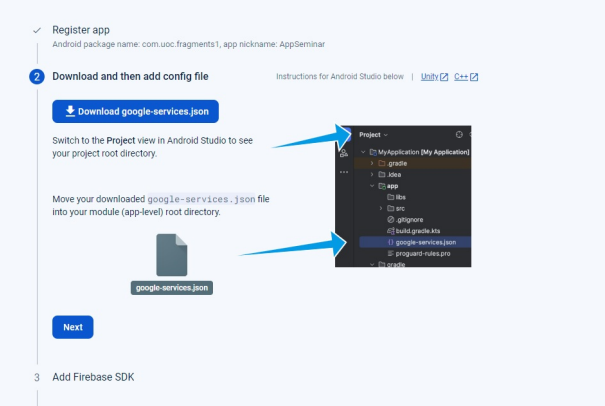
(f) The next step is adding our Android app to the Firebase project. From the main menu of our `SeminarData` project on the Firebase web console, we click the "`+ Add app`" button and we click on the "Android" button:



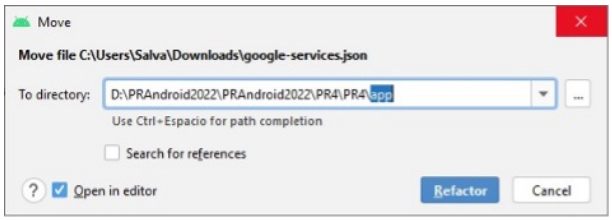
Since we are not going to use functionalities that require a SHA-1 signature, we can leave the value empty. Then, we fill in the remaining fields as indicated in the image and click on the "`Register app`" button.



On the next screen we must download the file "google-services.json" by clicking on the "`Download`" button.



Dragging it as directed will bring up the following dialog, where we should select the "`Refactor`" button:



Moving on inside the web assistant, in the new step we are told to modify the `build.gradle` files. These modified files have already been provided to you in the template of this CAT but some action is needed from your side. **Important:** Simply edit the `build.gradle` (Module: fragments1.app) and remove the comment from the plugin with id the following id: '`com.google.gms.google-services`'. After this change, the plugins should be now:

```
plugins {
    id 'com.android.application'
    id 'org.jetbrains.kotlin.android'
    id 'kotlin-parcelize'
    id 'com.google.gms.google-services'
}
```

2. **Back-end programming (theoretical questions)** (Weight: 15%)

To answer these questions, we will use the code provided in the template regarding Login. Remember that, to login in our app, the username is `user1@uoc.com` and the password is: `123456`.

(a) (THEORY) This CAT has a fundamental difference from previous activities. Where is the data located, compared to previous CATs? What impact does this have on data access speed?

(b) (THEORY) When method `loginAsync` ends, does it know at that time whether the login attempt is correct or not?

(c) (THEORY) What is the role of listener that is passed as parameter to the `loginAsync` method?

To answer questions 2.b) and 2.c), place two breakpoints within the `login` method from class `LoginViewmodel`:

```
22
23      fun login(username: String, password: String) {
24
25          listener.onLogin = { result ->
26              if (result is Result.Success) {
27                  _loginResult.value =
28                      LoginResult(success = LoggedInUserView(displayName = result.data.displayName,result.data.userId))
29
30              }
31              else {
32                  _loginResult.value = LoginResult(error = R.string.login_failed)
33              }
34
35          }
36
37          val result = loginRepository.loginAsync(username, password,listener)
38      }
```

3. **Loading a user's seminars** (Weight: 15%)

(a) In the method `readSeminarsUserIdsAsync` of the `DataSourceFirebase` class, the comments `//BEGIN-CODE-UOC-3.1` and `//END-CODE-UOC-3.1` populate the local result list with the value of the `usersem_seminar_id` that are related to the user who has logged in. We set the related user ID in the property `_user_id` in the previous call loginAsync. To query Firestore use the table "`user_seminar`" and the `whereEqualTo` operator. When accessing the elements, remember to cast the type:

```
val sem_id = document.data?.get("sem_id") as Long
```

You have to navigate all the response documents and add them to the result (in this case there will only be one, but there could be more). Remember to call the listener:

`listener.onSeminarsUserIds(result)`

(b) Now, within the method `selectSeminarsUserAsync` of the class `DataSourceFirebase`, when the listener is run we will already have the ids of the seminars related to the user. Between the comments `//BEGIN-CODE-UOC-3.2` and `//END-CODE-UOC-3.2` write the code to request the full details of the seminars in order to be able to access the name. We will create Seminar class instances and add them to the `userSeminarList` property.

i. Get seminars information from Firebase.

```
 .whereIn("sem_id", list_ids)
.orderBy("sem_id")
```

ii. And finally, each seminar must be added to the list of internal seminars (userSeminarList).
Remember to convert the `sem_id` to Int when creating the Seminar because Firestore returns it to us as Long.

iii. call the listener to indicate that we have already uploaded a user's seminars.

```
listener.onSeminarsUser()
```

iv. We go to the method called bind of ItemViewHolder, and within the comments //BEGIN-CODE-UOC-3.3 - //END-CODE-UOC-3.3, we load the seminar image using the Glide library.

```
 Glide.with(context)
.asBitmap()
.load(item.image_path)
.into(object : CustomTarget<Bitmap>() {
    override fun onResourceReady(resource: Bitmap,
    transition: Transition<in Bitmap>?) {
        // Handle the bitmap (e.g., set it to an ImageView)
        itemImageView!!.setImageBitmap(resource)
    }

    override fun onLoadCleared(placeholder: Drawable?) {
        // Remove references to the Bitmap, if necessary
    }
})
```

v. Within the onCreateView method of QuizFragment, between the comments //BEGIN-CODE-UOC-3.4 and //END-CODE-UOC-3.4, we also use the previous code to load the image.

At this point when you log in, you should already see the list of the user seminars with `user_id = 1`.

4. **Creating item table** (Weight: 15%)

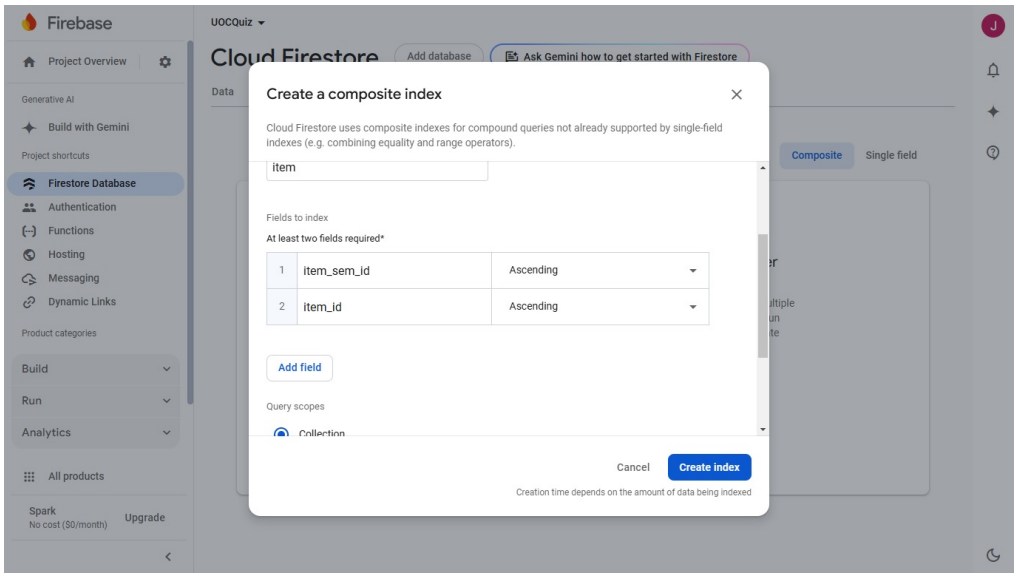
   We will go back to Firestore to create the `item` collection.

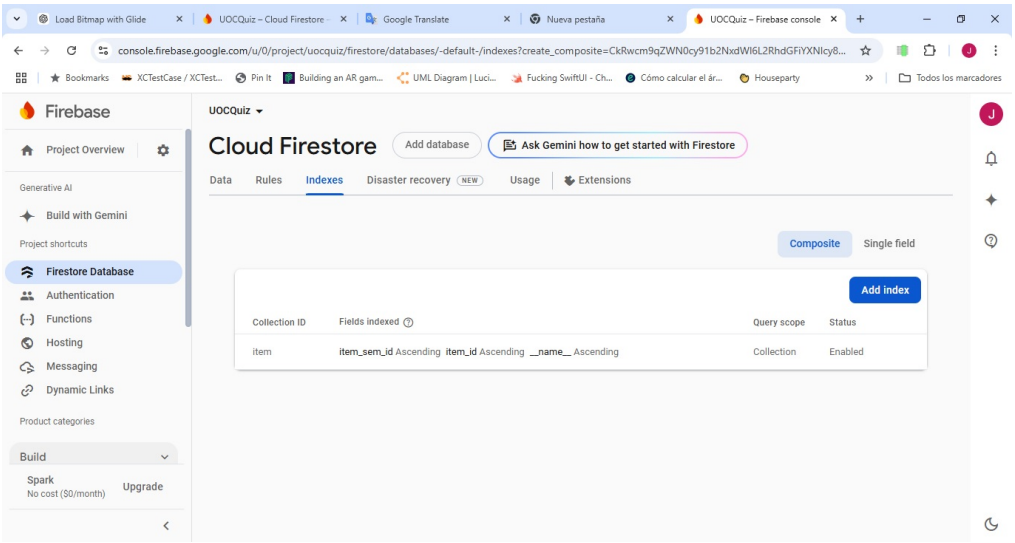
   (a) First, we are going to create the `item` collection with Auto-Id with the following properties:

   ```
   item_id:number,
   item_type:number,
   item_question:string,
   item_sem_id:number,
   item_link:string,
   item_correct_answer:number,
   item_answer1:string,
   item_answer2:string,
   item_answer3:string,
   item_answer4:string
   ```

   You can fill in the contents using the values from the project DBHelper file following the `INSERT` statements.

   (b) Next, you must create an index for the query. Creating an index must take a little time.





(c) Now, let us read the items. To do this, we are going to populate the method `selectItemsSeminary` of the class `DataSourceFirebase`. We are going to place our code within the following comments: `//BEGIN-CODE-UOC-4.3` and `//END-CODE-UOC-4.3`.

Hint: We must load the items of the seminars whose ID we receive as a parameter:

```
db.collection("item")
.whereEqualTo("item_sem_id",id )
.orderBy("item_id")
```

Loading items consists in creating an instance of the Item class with the document data from the item collection. We will add each item to the `seminarItemList` property. Finally, we will call the listener method `listener.onItemsSeminar()`.

5. **Inserting new Quiz** (Weight: 25%)

We provided the activity called `AddSeminarActivity`. We prompted the user to edit the Title, Duration, and Level. The image is always set to `https://www.revolumedia.com/uoc/img/questions1.jpg`. When the `Add` button is pressed, the insertion process is triggered.

(a) Within the comments `//BEGIN-CODE-UOC-5.1` and `//END-CODE-UOC-5.1` implement the method `getNewSeminarId`.

i. First query the seminar collection to get the highest value of `sem_id` (Tip: you can order the result DESCENDING and limit the result to 1.

ii. Once you have the last `sem_id`, add 1 to it.

iii. Use `listener.onNewSeminarId(sem_id.toInt())` to set the new seminar id. In case of Failure with the Firebase query, set it as -1.

(b) Complete the addSeminarAsync method. Within the section marked by `//BEGIN-CODE-UOC-5.2` and `//END-CODE-UOC-5.2`, insert the hashMap into the seminar collection.

(c) In the `addOnSuccessListener` listener of the add method, create a `hashMap2 = hashMapOf<String, Any>` to insert into the user_seminar table.

(d) (THEORY) How many listener calls are we using to add a new seminar to a user?

(e) When the insertion is complete, remember to create an item and use it to call the method `ReloadViewModelSeminar` so that the interface can refresh.

The insertion might have been successful, and you may need to scroll through the seminar list to see the new seminar.
After exiting and re-entering the application, you must check that the new seminar appears and confirm that it has been successfully saved in Firestore.

6. **Animation** (Weight: 20%) If the user answers all the questions correctly, we want a "Wow" message to appear on the screen. It should enter from the bottom of the screen and exit from the top. For pedagogical purposes, we want its width and height to be adjusted to match the screen width before the animation begins.

   (a) Add an `ImageView` with this configuration in `activity_main.xml` right after the `LinearLayout` close tag:

   ```
   <ImageView
           android:id="@+id/animatedImageView"
           android:layout_width="300dp"
           android:layout_height="300dp"
           android:src="@drawable/anim_wow"
           android:layout_marginBottom="-100dp"
           app:layout_constraintBottom_toBottomOf="parent"
           app:layout_constraintStart_toStartOf="parent"
           app:layout_constraintLeft_toLeftOf="parent"
           app:layout_constraintRight_toRightOf="parent"
           android:visibility="invisible"
           android:elevation="10dp" />
   ```

   (b) Create a method in the `MainActivity` activity with the signature `public void Animate()`. Inside this method

      i. Create a reference to `R.id.animatedImageView` called `animatedImageView`.

      ii. Access the screen width in pixels:

      ```
      val screenWidth = resources.displayMetrics.widthPixels
      ```

      iii. To access the current width and height of the image, use the code:

      ```
      val layoutParams = animatedImageView.layoutParams
      ```

      iv. We adjust the width of the image to match the screen's width so it occupies the entire width, and we also set the height of the image to match its width to prevent distortion. Modify the width and height of `layoutParams`

      ```
      layoutParams.width = screenWidth // Width in pixels
      layoutParams.height = screenWidth
      ```

      v. Reassign the `LayoutParams` with:

      ```
      animatedImageView.layoutParams = layoutParams
      ```

      vi. Access the screen height in a similar way to how the width was accessed:

      ```
      val screenHeight = resources.displayMetrics.heightPixels
      ```

      vii. Create an instance of the `ObjectAnimator` class to animate the Y-coordinate:

      ```
      val translationY = ObjectAnimator.ofFloat(
      ```

```
            animatedImageView,
            "translationY",
            screenHeight.toFloat(),
            -(screenHeight + 300).toFloat()
        )
        translationY.setDuration(3000)
```

viii. Add an `Animator.AnimatorListener` to translationY, and in the `onAnimationEnd` method, make the image invisible.

(c) Now, at the end of the `ShowAlertResult` method in `MainActivity`, if correct is equal to count, execute the new Animate method.

Resources

Basic Resources

- Course Wiki: Section 10 - Concurrency
- Course Wiki: Section 11 - Back-end
- Course Wiki: Section 12 - Services

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. 15%
  3. 15%
  4. 15%
  5. 25%
  6. 20%
- 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 with all changes and a PDF file with the answer to textual/theory exercises.

The name of the file you submit must be **CAT4_SurnameName.zip**. This document must be submitted in the **Continuous Assessment Record** of your classroom **before 23:59 of 2025/05/11**. **Late submissions will not be accepted.**