Ghent University

Cloud Applications and Mobile (E736010)

Lab 6: Mobile Development with Kotlin

Professor:
Prof. dr. ir. Sofie Van Hoecke

Assistants:
Ing. Cédric Bruylandt
Ing. Tom Windels
Dr. Jennifer B. Sartor

2024 - 2025

Table of contents

1 Introduction

An overview of the big picture for all labs can be found in Figure 1. We will finally complete the IoT stack in this lab by implementing a consumer of our sensor data. We will start with getting a beginner mobile application written in Kotlin up and running in an emulator or on an Android mobile device. You will see how your app can use the Egress endpoints set up in Lab5 to obtain the sensor data and visualize that data in an application. We will then extend the application by adding additional functionality. You will also be able to be creative in developing your mobile application further to give a potential user a nice user experience while they browse the sensor data on their phone.

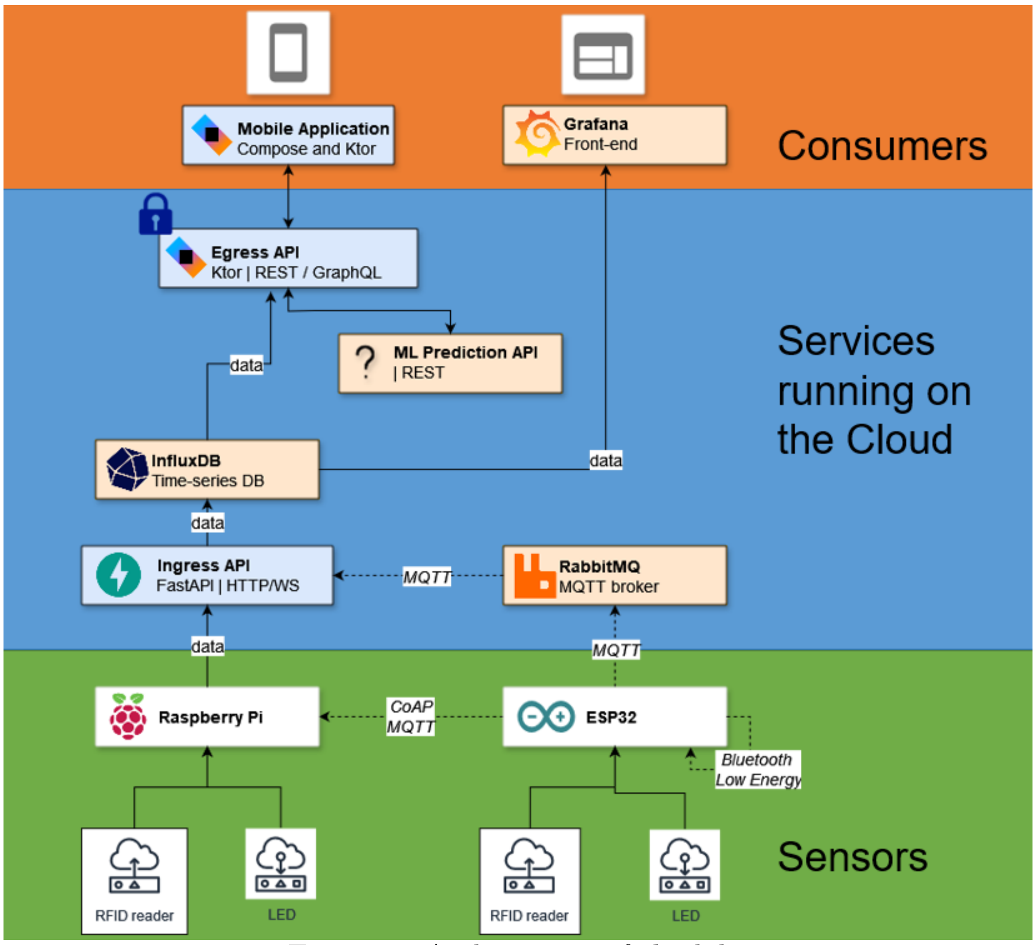


Figure 1: Architecture of the labs

2 Goals

1. Creating a mobile application.
2. Finishing our IoT stack development, from sensors through microservices to consumers.
3. Interpreting a given source code and expanding it with extra functionality.

3 Software

3.1 Android

You will need to download and install Android Studio. The instructions per operating system are here. We will also need to have a working Android emulator in case we want to test the app within Android Studio directly. See this Android webpage on how to run apps on an emulator.

You can create a virtual device. Choose the device on the first screen, a recent Pixel(a) mobile phone is fine, and target the latest stable Android version, version 14. You may first have to click *Download*. Keep in mind that downloading new SDK's will take time and storage space. Alternatively, you can set up your own Android device using the instructions found here, which works better when using Android Studio on lower end hardware.

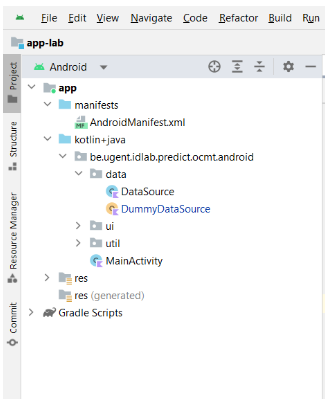
4 Tutorials

4.1 Tutorial 1: Understanding Jetpack Compose

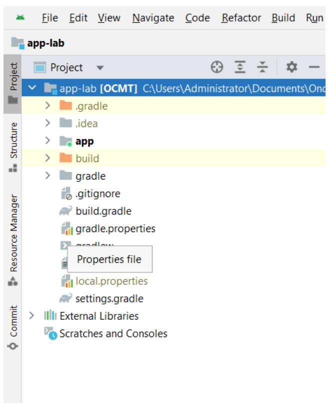
Jetpack Compose is a toolkit for native UI development on Android. A tutorial for how to get started can be found here. There are also examples of how to use Compose in the theory slides.

4.2 Tutorial 2: Project View in Android Studio

In the project tab of Android Studio, you can see the file structure of your project, see Figure 2. While "Android" is the default view of a project (see Figure 2a), we recommend selecting the "Project" view (see Figure 2b), which can be selected at the top of the Project window in a drop-down menu. Project view provides a 1-to-1 mapping of the files you see in the Project window with what is actually in your directory structure, whereas the Android view hides some unused files.



(a) Project files in the Android view



(b) Project files in the Project view

Figure 2: Different views of a project within Android Studio

4.3 Tutorial 3: Consuming the Egress API

You are now able to take on the challenge of creating a dashboard to visualize the data exposed by the Egress API. The source code containing all the required components to achieve this is provided for you.

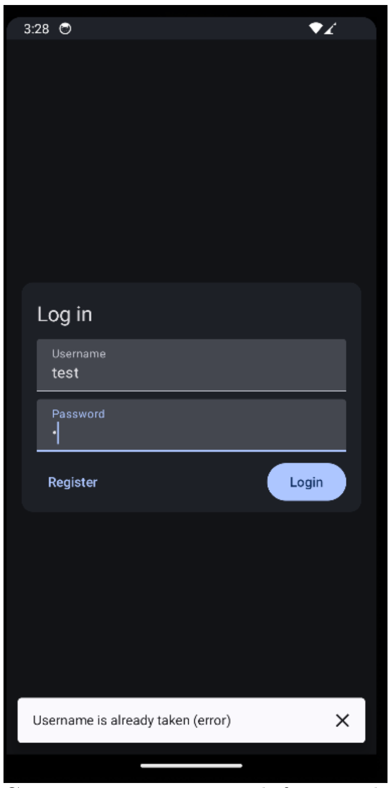
When looking at the folder structure of the code that we have given you, you find that most of the logic of the application is in `app/src/main/java/be/ugent/idlab/predict/ocmt/android`. In that folder, you will see a file `MainActivity.kt` and three folders: `data`, `ui`, and `util`. In `MainActivity.kt`, you can see the implementation of `onCreate`, which sets up what to do when the mobile application is created. In this case, we set the UI wrapped in the theme defined in `ui/theme/Theme.kt` using the screen management defined in `ui/screen/Root.kt`. Inside the `MainActivity` class, we also set up a user session, and services for our Egress endpoints. The user session and services are defined in the `data/service/` folder. If you look inside the files in this folder, you can see that the `UserSession.kt` file takes care of user credentials, registering and logging a user in to your Egress, and the corresponding error checking. The `SourcesService` is in charge of asking your Egress for the list of sources, or `sensor_names`, stored in our database, using the user session for authentication. The files `CountsService` and `ForecastService` are for communicating with the corresponding Egress endpoints and getting either counts or forecast data back.

There are a few files in the `util` folder that define helpful functions that could be useful in the app development. In the `data` folder, we have the logic to get the data that we want to visualize in our app. `Egress.kt` is an important file because it defines the endpoints in Egress with which we need to communicate. This file also translates the parameters that we need to send to the endpoints into a nicely formatted string to POST to the REST endpoints. The file `Models.kt` includes a class of `Credentials`, consisting of a username and password, and a class representing the `User`, where the token returned by Egress is stored. The `ui` folder contains all of the UI logic. There are separate folders for the code that takes care of screens, the theme, graphs, components, etc. The `composition` folder helps with navigation. The `components` folder defines the UI specifics of some components needed to visualize parts of the screens in our app, such as the `RouteWidget` and `ObservedCountsWidget`. In the `ui/screen` folder, you will find the various screens that make up our app, including the `AuthenticationScreen`, the `LandingScreen` (defined in `Main.kt`), and the `ObservationsScreen`. Examples of these screens can be seen in Figures 3, 4, and 5. The `LandingScreen` is where a user ends up after logging in, and lists the different `sensor_names` that you have pushed to your InfluxDB, or sources, from which the user can choose. After choosing a source, the user goes to the `ObservationsScreen`, where a graph shows the RFID counts over time for that source. Finally, the files in `ui/graph/` specify how the graph is actually rendered when given sensor data.

You can open up the project in Android Studio to easily inspect the code. You should be able to run the mobile application code that we provide for you.

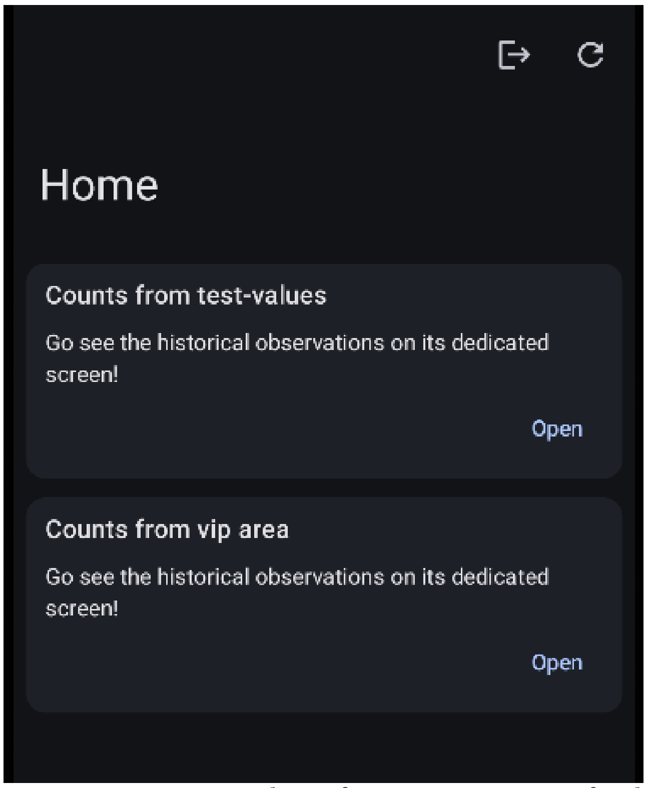
4.3.1 App Organization

An overview of the main files, components, and classes used in the mobile application and their relationships are shown in Figure 6. You can see the `MainActivity` in the middle,



Username is already taken (error)

Figure 3: Starting page, used for authentication



Counts from test-values

Go see the historical observations on its dedicated screen!

Open

Counts from vip area

Go see the historical observations on its dedicated screen!

Open

Figure 4: Home page containing a list of `sensor_names` fetched from Egress

which is the entry point and controller of the app. The UI parts of the app development can be seen to the right, and the elements used for the data layer implementation are on the left. As mentioned above, the `MainActivity` makes instances of the `UserSession`, `SourcesService`, `CountsService`, and `ForecastService`, which each use the `Egress` object to interact with your Egress microservice. On the UI side, we have a `Root` screen, which directs the app to the correct screen, first the `Authentication` screen, then the

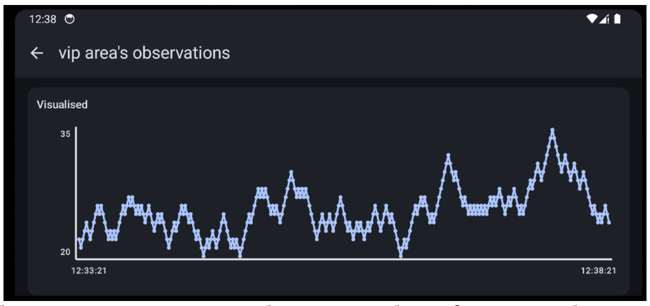


Figure 5: Observations page, where graphs of sensor data are displayed

LandingScreen, and finally to the ObservationsScreen. These screens use components to render their visualizations. The screens also go through the MainActivity to access the data layer elements, such as the services and UserSession.

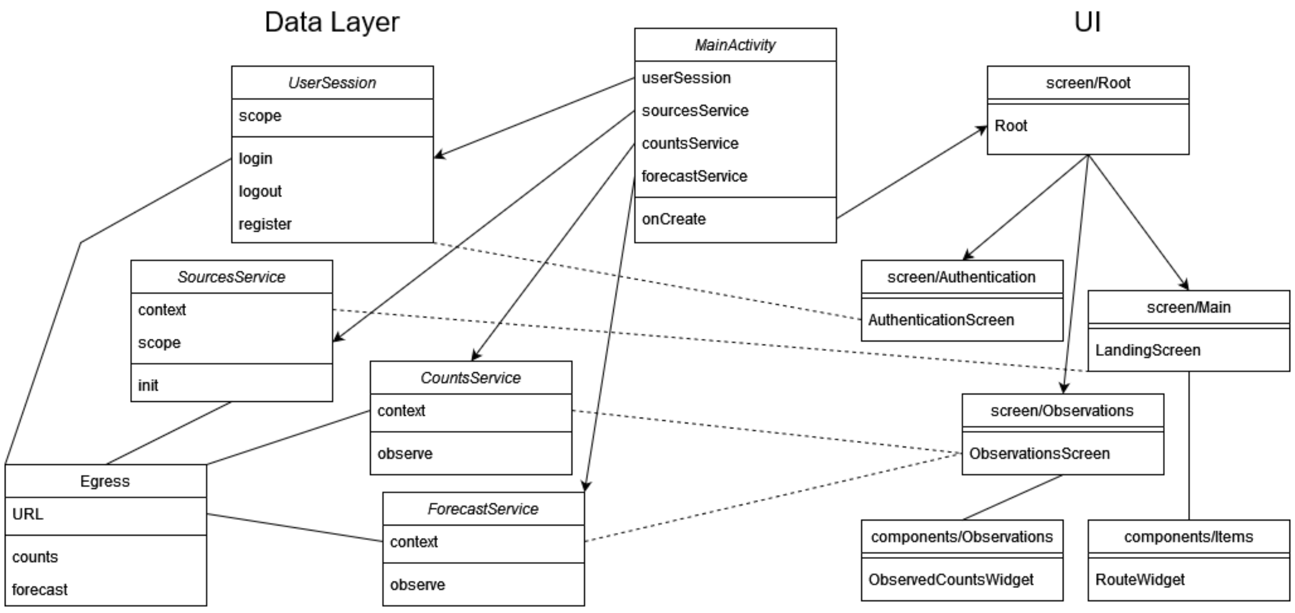


Figure 6: Files and classes used in the mobile application and their relationships

5 Tasks

5.1 Task 0: Running the app

The app has to be configured to use your Egress instance. In Egress.kt, modify the Egress object's URL value to match yours.

You can build and run the app on the emulator or physical device you configured in the preparation part (Section 3). You should see the log in screen. Notice how clicking any of the buttons shows a pop up notification "not yet implemented".

5.2 Task 1: Authentication

Complete the functionality of the application's authentication flow in order to navigate to the other pages using a valid token. In UserSession, the logic to update the application state and handle login/register requests are already in place, but the logic to execute these requests isn't implemented yet. You will have to implement `login(credentials)` and `register(credentials)` yourself. There is already a globally defined `httpClient` available (in `data/Util.kt`) which you can use to contact the relevant endpoints on your Egress instance.

5.3 Task 2: Sources overview

While we are now signed in, we cannot navigate to any observation screen yet: the list of sources, or `sensor_names`, is empty. Similar to `UserSession`, `SourcesService` has its own state implemented, but the implementation to get the sources from Egress is missing. Similar to Task 1, you now have to implement the request to get that data from the relevant Egress endpoint, with the key difference being that this request now requires authentication. There is already a utility function `authorizedRequest` defined in `service/Util.kt` for the `UserSession` class, helping you by already setting the available token. After this is implemented, you should be able to see the `LandingScreen` with a list of sources displayed.

5.4 Task 3: Counts & forecast

When navigating to an observation screen, we see that the widgets remain empty: the implementations to continuously monitor both the counts and forecast endpoints are missing. These have to be implemented in the `observe(...)` method found in `CountsService` and `ForecastService`. This method returns a `Flow`: this represents a stream of data asynchronously retrieving new values. This is exactly what we need for our use case, as we want to periodically fetch data samples as long as necessary. Similar to `SourcesService`, these requests require authorization.

This page details how such a flow can be constructed from asynchronous data requests. In the example, the function `simple()` asynchronously sends out numeric values into its created data stream and waits in between submissions. In our app, the UI logic is responsible for `collect`ing this stream of data. This in turn dictates how long the stream remains active: as soon as the UI stops the collection process, the flow is stopped, and no additional requests will be made. The graph logic already contains everything necessary to manage these collection processes.

Tip: The already present `Response` classes can be used to process the data retrieved from your Egress endpoints.

5.5 Task 4: Attendance

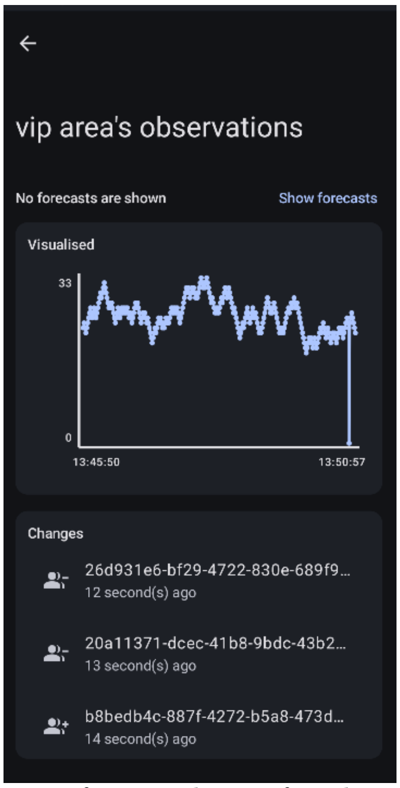
Similar to Task 3, the attendance integration is missing. This time, however, the UI widget is also not yet implemented. The `AttendanceService` has to expose a stream of attendance entries with similar arguments as `CountsService`. You will have to implement the bottom part of the page displayed in Figure 7, which shows the RFID IDs and whether they are leaving or arriving.

Tip: Now you are responsible for observing the values retrieved from the `Flow`. In the theory slides, we have covered how you can do this in a Compose-friendly manner.

Tip: The Material 3 design guide has several components listed, with most being directly available in Jetpack Compose. In Figure 7, `ListItems` are used to display the individual entries.

Tip: Given that the number of entries can be big, a `LazyColumn` should be used: it can analyze what portion of the entries are and are not visible depending on the scroll position, and cleverly only shows the necessary subset of entries as a result.

Tip: The `ObservationsScreen` currently has an empty `Spacer`, stylized to take the missing widget's place and style. The `Modifier` applied to that `Spacer` can be reused on your own UI widget implementation.



26d931e6-bf29-4722-830e-689f9...
12 second(s) ago

20a11371-dcec-41b8-9bdc-43b2...
13 second(s) ago

b8bedb4c-887f-4272-b5a8-473d...
14 second(s) ago

Figure 7: Reference design for observations

5.6 Extra tasks above 14 points

While we have completed the core functionality of the application, it can use some further refinement. For the last 6 points of this assignment, you can choose what tasks you implement. You are free to make improvements to any of the shortcomings, or introduce new features!

Possible ideas for extra tasks are:

1. disabling the login/register buttons while username/password is missing;
2. restricting username & password to not contain any white space/newlines;
3. allow the user to provide time bounds on the observation detail screen;

4. manage an ongoing notification containing the data retrieved from `CountsService`;
5. make user sessions persistent (as long as the token is valid) by saving it locally and checking local storage on app start first;
6. improved landscape support, where the observations screen uses a `Row` to put the graph and observations UI side-by-side instead;
7. introduce a new detail screen, e.g. displaying the attendance in greater detail.
8. distinguishing data from the past displayed in the graph from the forecasts for the future, visually.

6 Questions

1. If you had more time, what else would you like to add to your mobile application?
2. If you toggle between hiding and showing forecasts in the observations detail screen, you can see the graph "flicker". Why do you think that is? How would you fix this?

7 Summary

You have now completely implemented your IoT stack - from collecting and sending sensor data, to our Ingress and Egress services running in the cloud, to finally our mobile application that consumes and visualizes the data in this lab. This lab focused on working with native Android through Android Studio using Jetpack Compose, allowing you to run your consumer application on a mobile device.

8 Material you need to submit

Prepare a lab report that elaborates on the solved tasks along with an explanation of the code. Give the lab report a structure that mimics the structure of this document. Do not forget to answer the question(s) in Section 6. You should take screenshots or short movies that show that you got each task working. Screenshots can be added to your report, and videos can be bundled into your zip file. If you had to do anything noteworthy in the preparation section, also include this in your report, but it's not obligatory to say something about the preparation part. Only the **.pdf** extension is allowed for the report. Name your lab report **LAB6_FamilyName_FirstName.pdf**. Archive the report together with the source code for the exercises and videos, and name this archive **Lab6_FamilyName_FirstName.zip**. Hand everything in using Ufora. You can hand everything in multiple times, only the last file will be saved.

In addition, you will be expected to **DEMO your mobile application** for us after the end-of-the-year exam. We expect this to take 10 minutes, and will ask a few questions. All you need to do ahead of time is make sure there is data that can be displayed.