

Group Project Assignment
Programming Ubiquitous Things
Spring 2025 - IFI/UiO

1. Introduction

This document describes the group project to be developed by each group (2 students[1]) for the course Programming Ubiquitous Things (PUT) at UiO in Spring 2025. This work focuses on the interaction between:

- a client application running on a smartphone (running Android, API 35), written in Kotlin (or in Java) and using Jetpack Compose (to make the interface) or providing the interface in XML, and
- a server based on Eclipse (IDE for Enterprise Java Developers with Spring Tools 4).

This work simulates the case of auto insurance and consists of developing an information system for a hypothetical insurance company that provides vehicle insurance products. This kind of insurance policies provides financial protection against physical damage resulting from traffic collisions and against liability that could also arise therefrom. Vehicle insurance may additionally offer financial protection against theft of the vehicle and possibly damage to the vehicle sustained from things other than traffic collisions, such as vandalism and damage sustained by colliding with stationary objects.

2. Goal

The goal of this project is to develop a mobile application that allows its users (the clients of the insurance company) to access a subset of services provided by the insurance company's information system through an Android mobile device. The application must support the following functionality:

- Login and logout;
- Modify the current password;
- Access detailed customer information (the client claims who has logged in previously);
- Submit a new insurance claim (a simple form; e.g., including the GPS coordinates of the accident);
- Update an existing claim;
- Show the map corresponding to the GPS coordinates of a claimed accident;
- Show the photo taken when an accident occurred.

Essentially, the mobile application will provide a client frontend to some basic services implemented by the information system of the insurance company. To read and write all relevant domain-specific data (whenever needed), the mobile app will communicate with a server responsible for maintaining the insurance company's database of customers and claims. The client running on the smartphone should use the "volley" library for this purpose. Please note that to invoke the server, you should use the following IP: `10.0.2.2:8080`

3. Functionality

This section presents some relevant information regarding each operation that a user can make when using the Android mobile app.

Run the server first and then the smartphone app. There are two files to consider on the server side: `person.json` and `claims.json`. See the file contents while or after reading this document. The contents of these two files are read into an in-memory data-structure when the server starts running. After that, only when needed, the above files are written and the in-memory data-structure updated accordingly.

After getting the data from the server (i.e., after the user has logged in and the corresponding claims have been obtained), it is assumed that the smartphone app supports disconnected work, i.e., the application can still run in the smartphone, even if the server is off-line and/or there is no network connection.

[1] The existing groups and its members are available in the IN5600 web page.

3.1 Login and Logout

First of all, after launching the mobile application, the user must **authenticate** himself to the system by providing a username (email) and a password (SHA-512 hash). Assume that the username and password have been introduced in the server when the customer has registered in the company. As a matter of fact, the server has two files called `person.json` and `claims.json`. They contain information regarding the users and their claims, respectively. The maximum number of persons is defined in the server as `PERSON_ARRAY_SIZE = 4.` This is currently defined as an array and the corresponding class in Java can be found in file `person.java`.

After reading the username and password from the mobile interface, the smartphone application must verify the correctness of the data inserted. This is done either:

- by invoking a remote method (provided by the server) that will verify this information and returns success or failure (you should not send the password in clear in the network), or
- is done locally by checking the authentication information stored locally in the smartphone (you should not store the password in clear in the smartphone); in fact, once the authentication is done successfully, there should be kept a file in the smartphone with the details of the logged in user; this should be created/written/read as a "DataStore" in Android.

If the authentication is correct, the user can proceed, performing several operations (already mentioned) through the mobile app's graphical interface (i.e. see his/her claims, insert new claim, etc.).

Finally, the logout consists simply of logging out from the app.

3.2 Modify the current password

Each user can change the password initially given (file `person.json`). For this purpose, a user just has to invoke the corresponding method in the server.

3.3 Access Claims

Each user may report several claims. The maximum number of claims (per user) is defined in the server as `CLAIMS_ITEMS_ARRAY_SIZE = 5.` This is currently defined as an array and the corresponding class in Java is in file `claims.java`. Once the claims are obtained for the first time from the server, the smartphone app must keep in a local "DataStore" file the claims corresponding to the user.

When claims are shown to the user, the data must be presented (in particular, the map and the photo) as clickable links so that the user can click on any one of them and go directly.

Note that the server offers a command line interface that allows a user to change the status of a claim. This should can (and should) be used by users when testing the app.

3.4 New Claim

When a new claim is inserted, please take into account the maximum number of claims available to be stored in the server. The mobile app should assume that a new claim belongs to the user who is logged in. Do not forget to invoke the server for this purpose and update the local claims "DataStore" (and the related in-memory data-structure, if that is the case).

3.5 Replace Claim

When a claim is replaced, only one or all of its data may be modified. The mobile app assumes that a claim being replaced belongs to the user who is logged in. Note that the user must provide the claim identifier of the claim that is being replaced. Do not forget to invoke the server for this purpose and update the local claims "DataStore" (and the related in-memory data-structure, if that is the case).

3.6 Map of the Accident

When showing a map corresponding to the GPS coordinates of a specific claim, the smartphone app should use Google Maps or OpenStreetMap. The GPS coordinates mentioned above are stored in the claims in the form of "latitude-longitude". Showing the map corresponding to the GPS coordinates of a specific claim

requires the use of Google Maps (or similar). For that purpose, you can have a look at the following website:
https://developers.google.com/maps/documentation/android-sdk/map
https://developers.google.com/maps/documentation/android-sdk/maps-compose
https://wiki.openstreetmap.org/wiki/Osmdroid

3.7 Photo of the Accident

When an accident happens, the user can take a photo and store it locally (i.e., his/her smartphone). While kept locally, a photo can be stored in a simple file; note that the corresponding claim keeps the name of the corresponding file. Finally, the photo is also to be sent to the server using the corresponding method in the server.

So, the first thing that a user must do is to choose a photo (e.g., you can use any existing file taken from the web) among those that exist locally. For this purpose, i.e., to choose a photo, an already existing application in the smartphone should be used. Then, to write the photo into a file and to show it on the smartphone's display, new activities should be created.

4. Baseline Functionality

In the baseline version, all the functionalities mentioned above must be provided except those that: 1) show a map corresponding to a claim, and 2) show a photo of an accident.

Note that each claim includes fields with both the GPS coordinates and the name of the file where a photo is stored. This data does not have to be inserted in the baseline functionality (i.e., it can be simply "na").

Note that allowing offline operation mode is part of the baseline version. Thus, the number of remote invocations to the server should be minimized to ensure its scalability. If the server is offline or the device is disconnected from the network, the method invocation does not work (this situation should be detected by the smartphone app). However, the application running in the smartphone must be capable to work after a reboot of the smartphone, a re-start of the app, when the server is not running, or when there is no network available. Thus, the goal of offline operation mode is to increase the scalability of the overall system and to have the app provide as much functionality as possible, even in the presence of network failures.

In the Baseline version, the app should handle gracefully all unexpected situations. In other words, the app should never crash.

In the claims shown to the user, present the data (e.g. the map or the photo) as clickable links so that the user can click on any one of them and go directly.

5. Advanced Functionality

In addition to the baseline functionality, students are encouraged to implement the following advanced features as follows:

1. The first advanced feature consists in showing a map with a mark where an incident has occurred. Note that in the baseline version, a claim can simply contain the "na" text for the "latitude-longitude" pair of the incident and it is not required to show the corresponding local in a map. However, in the advanced version both are needed.

2. The photo taken by the user is to be stored in the smartphone and also sent to the server (using the corresponding method in the server to upload the photo). As already mentioned, to allow the user to choose a photo among the local existing files, you should use an already existing application on the smartphone. Thus, the user must be able to store the file and see the corresponding image.

3. The app should handle gracefully all unexpected situations. In other words, the app should never crash.

6. Questions

Assume that a user, after modifying their password on a server (called Server-1), logs in to another server (called

Server-2). The detailed scenario is as follows:

a) While the client app is connected to Server-1, the user modifies the password;
b) The user logouts from the client so that a posterior login requires a connection to the server;
c) Server-1 is shut down, and the app connects to another server (Server-2) with the previous `person.json` file (still with the old password);
d) The user tries to log in to Server-2 using the new password.

Now, answer the following questions (in the report):

1. Is the login mentioned in step (c) above done correctly?
2. Why does the above situation happen?
3. What are the possible solutions for the above situation?
4. Explain one of such solutions (mentioned in the previous item).

7. Development Environment and Available Server Methods

The application must target the Android platform. The mobile application will be developed using the Android Studio IDE and tested using the Android Emulator virtual machine, which can run locally on each student's computer. The source code of the application should be implemented in Kotlin (or in Java) using Jetpack Compose (to make the interface) or providing the interface in XML. To test the application, use the Android Emulator, which is bundled with the Android Studio IDE. The Android Emulator is a virtual machine that emulates an Android device.

A server implementation will be provided by the faculty; it can be executed locally (i.e., on the student's computer) and must use Eclipse IDE for Enterprise Developers with Spring Tools 4. The server is written in Java, and its code should not be modified. This server has the `serverController.java` file where the remote invocations arrive, and the `server.java` file where you can find the code that effectively serves each one of the remote invocations.

To clarify any aspect related to the above mentioned methods, you can look at the corresponding files in the Eclipse IDE (e.g., files `person.json` and `claims.json`, `serverController.java`, etc.). For any issue, you may ask the faculty.

8. Suggested Development Stages

The faculty suggests the project to be developed in the following stages in the lab classes. Some extra work, done outside the labs, may be needed.

When starting this project, the initial step should be to specify the graphical user interface of your mobile application by producing a mock-up GUI of the application screens. You can use pen/pencil and paper, or some online tool such as https://ninjamock.com/ or https://balsamiq.com/wireframes/ (trial version).

As a suggestion, start by supporting the login from the user and storing the corresponding information locally in the smartphone using "DataStore"; note that from this point on, when the user logs-in, it should not be needed to remotely access the server.

Always remember that it is better to test the new code that has been inserted step by step. In other words, do not write all the code and then start testing it.

9. Deadlines and Deliverables

The project can be developed during the lab classes. Some classes (see "Detailed Plan" in the IN5600 web site) can be used to get help from the professor. The project delivery deadline is 2 May at 23h59, and the presentation dates are available in the file mentioned above ("Detailed Plan").

There are two deliverables; both are to be delivered in two separated zip files:

- A report describing the implementation details of the mobile application along with the answers provided to the questions mentioned above.

  The report must include a specification of the mobile interface, namely a wireframe diagram of the

app's screens (a template for such report is also provided in the PUT web site). Name this file as follows: "**report-groupXX-PUT-2025.zip**" where XX is the group number (see this number in the IN5600 web site).

- The source code of the Android Studio project packaged in a ZIP file. In Android Studio choose *File -> Export to Zip File.* Name this file as "**client_group_XX_project-PUT-2025.zip**" where XX is the group number (see this number in the IN5600 web site).

The details of the delivery process (e.g., the URL to be used) will be indicated in the IN5600 web site.

The total score of the project is broken down in the following parts (in a total of 20 points):

- 8 points: baseline functionality;
- 6 points: advanced features;
- 2 points: robustness;
- 2 points: report
- 2 point: code quality (e.g. readability, well structured, comments, etc.).