# Project Assignment for the Course on Programming Ubiquitous Things Spring 2019 IFI/UiO

# AutoSure - an app for auto insurance

## 1 Introduction

This document describes the project to be developed by the students for the course named Programming Ubiquitous Things (PUT) at UiO in Spring 2019. This work focuses on the case of auto insurance and consists on developing an information system for AutoSure, a hypothetical insurance company. The project is to be developed in groups of 2 students.

The global scenario is that of an insurance company, AutoSure, 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 there from. 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 AutoSure customers – 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 basic steps and functionality:

- Login, logout;
- Access detailed customer information;
- Submit a new insurance claim (i.e., a simple form that describes the vehicle entity, date, and claim details);
- List the history and the status of the customer's insurance claims;
- Access detailed claim information;
- Exchange messages between the user and AutoSure. This can be used by the user to ask for more detailed status of the claims or to leave some feedback. AutoSure employees can also send messages or reply to the user.

In the baseline version (see Section 3), these functionalities must be provided by the application while connected to the network. To read and write all relevant domain-specific data, the mobile app will communicate with a server responsible for maintaining the insurance company's database of customers and claims. The server offers a Web Service (WS) interface that can be remotely invoked by the mobile app.

The target mobile platform of this project is Android (5.0 or higher). 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. A server implementation will be provided and can also be executed locally (i.e., on the student’s laptop).

## 3 Baseline Functionality

Essentially, the mobile application will provide a client frontend to some basic services implemented by the information system of the AutoSure insurance company. This information system is implemented by a server that manages all data about the company's customers and the corresponding insurance claims. The server provides a WS interface that can be remotely invoked by the mobile application. The application must support the following use cases.

First of all, after launching the mobile application, the user must **authenticate** himself to the system by providing a username and a password. Assume that the username and password have been introduced in the server when the customer has registered in the company. After reading the username and password from the mobile interface, the application must invoke a WS call that will verify this information and return a success or failure message. If the authentication is correct, the user can proceed, performing several operations through the mobile app's graphical interface: **list** customer information, **file** a new insurance claim, **list** the insurance claim history, **display** detailed information of specific claims, **view** and **send** messages regarding a particular claim, and **log out** of the session.

By **listing customer information**, the application must display five pieces of information on the screen about the user who is currently logged in: customer name, fiscal number, address, date of birth, and insurance policy number**.** This information must be retrieved from the server by invoking a specific WS call.

The user can also **file a new insurance claim**. In this case, he/she must provide a **title**, a number **plate** from the customer list of number plates available, an **occurrence date** and a short **description** for the claim. The application will then submit this data to the server using a specific WS call. Internally, the server will assign both an identifying number to the claim and the submission date of the claim.

In addition to submitting a new claim, the user can **list the history of all his past insurance claims** and read their specific details. To fetch the claims' history, the mobile app issues a request to the server, which returns a list of claim items. Each claim item contains the title and the identifier of the claim. After receiving the list of claim items, the mobile app must show this list to the user, sorted by identifier in ascending order.

The user can then **browse the claim history and view detailed information about a particular insurance claim**. All the user has to do is to select a given claim item from the list and the mobile app will fetch the following tuple from the server and display it on the screen: claim id, issuing date, title, plate, description, and claim status. The claim status can take one of three possible values: **pending**, **accepted**, or **denied**. Pending means that the claim has been received by the server, but was not yet handled by an internal claims handler. Accepted or denied means that the claim has been handled by a claims handler, who has accepted or denied the claim, respectively (see Section 5 for more details regarding the change of a claim status).

Still regarding a particular claim, the user can **send** messages to the server and **receive** messages from it. These messages contain only text and can be used to send feedback or to ask questions regarding the claim. AutoSure can also reply or send messages regarding a claim (for example, to give the user more detailed information regarding the status of the claim). To get the messages from the server, the application must use a specific WS call which returns an ordered set of exchanged messages. To send new messages, the application uses another specific WS call that saves the new message in the server.

**While the user is logged in, he/she can repeatedly perform any of the operations**: list customer information, file new insurance claim, list insurance claim history, read claim details, and send/receive messages regarding a claim. To perform these operations, the mobile application must invoke a WS call with a valid session identifier. The session identifier is a number returned by the server when the user authenticates successfully and must be preserved in memory by the mobile app while the user is logged in. For simplicity reasons, the communication does not need to be ciphered.

Finally, **the user can log out from the current session**. When the user logs out, the mobile app wipes the session identifier from memory. Thus, the user must authenticate again before he can access the insurance company's services.

## 4 Advanced Features

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

1. Allowing a simplified offline operation mode (while being capable to work even after a reboot of the smartphone or a re-start of the app). In the basic architecture (functionality described in the previous section), the mobile application must perform a WS method invocation in order to read some relevant data from the server. However, if the server is offline, or the device is disconnected from the network, the method invocation will fail, resulting in an error message being displayed to the user. The goal of offline operation mode is to provide some functionality in the presence of network failures. This can be achieved by maintaining a local in-file cache of server data. With this approach, even if the network is not reachable, the user can still see data previously retrieved from the server;
2. The second advanced feature consists in providing notifications to the user when new messages from the AutoInSure server are received by the client. This can be achieved by having a service that periodically checks if there are any new messages regarding any claim. If so, a notification should be displayed to the user.

Note that both in the Baseline and in the Advanced cases, the app should handle gracefully the inexistence of network connection. In other words, the app should never crash.

You have the freedom to specify how this offline operation mode and notifications will work.

## 5 Development Environment and Tools

The project is to be developed for Android smartphones, using the Android Studio IDE. The source code of the application must be implemented in Java. To test the application, use the Android Emulator, which is distributed with the Android Studio IDE. The Android Emulator is a virtual machine that emulates an Android device.

Faculty will provide a Web Service implementation of the insurance company server. Through a WSDL interface, this server exposes the following remote call interface (written below in pseudo-code):

- *login*(username, password) → sessionId | 0
- *getCustomerInfo*(sessionId) → JSONCustomer | ErrorMessage
  (Customer contains: customerName, fiscalNumber address, dateOfBirth, policyNumber)
- *getClaimInfo*(sessionId, claimId) → JSONClaimRecord | Null -> ErrorMessage

(ClaimRecord contains: claimId, submissionDate, occurenceDate, claimTitle, plate,description, status)

- *listPlates*(sessionId) → JSONPlateList | ErrorMessage
  (List of String (number plates) associated with the customer)

- *listClaims*(sessionId) → JSONClaimItemList | ErrorMessage
  (list of ClaimItem, each containing: claimid and claimTitle)

- *listClaimMessages*(sessionId, claimId) → JSONClaimMessageList | ErrorMessage
  (list of ClaimMessage, each containing: sender, message, and date)

- *submitNewClaim(sessionId, claimTitle, occurrenceDate, plate, claimDescription)* → *True* | *False*

- *submitNewMessage*(sessionId, claimId, message) → True | False

- *logout*(sessionId) → True | False

The Web Server code will be publicly available on the website with PUT related information. It can be compiled and executed on your local computer using the Eclipse Java EE IDE (you can get the file to install, "Get Eclipse IDE 2018-12", from the following URL from the Eclipse web site: https://www.eclipse.org/downloads/packages/release/kepler/sr2/eclipse-ide-java-ee-developers).

The server runs as a standalone process listening on port 8080. To check that the server is listening, open a browser, and follow the URL (this will show the WSDL interface of the server):

http://localhost:8080/AutoInSureWS?WSDL

Once the server is listening, it is possible to test your mobile application on the Android emulator. As you interact with the mobile app through the emulator's virtual interface, the app will generate WS requests to the local server. The server provides a command line interface that allows you to approve or deny pending claims and to send messages for a specific claim. This feature allows for emulating the claim handling workflow, and it will help you to test and debug your application.

## 6 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.

**Lab Class 5 - Specification of the mobile interface.** The goal of this stage is 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).

**Lab Class 6 and 7 - Prototype implementation of stand-alone interface screens.** Based on the GUI mock-up produced in the first stage, implement a prototype of each screen as a stand-alone Android Activity. The layout of each Activity should reflect the GUI specification. It is not necessary to implement the Activity's internal logic.

**Lab Class 8 and 9 - Integration of the stand-alone interface screens**. After implementing all Activities of the mobile application, integrate them together as a single application. This task

requires you to implement the logic that allows the user to navigate across the screens (i.e., Activities) without making WS calls.

**Lab Class 10** - **Implementation of WS remote calls to the server**. This stage consists of adding remote call invocations to the mobile application prototype. These calls will be triggered by the Activity code. The server code will be required in this stage only in order to respond to the WS requests.

**Lab Classes 11, 12 - Implementation of advanced features** and testing. Conclude the implementation and testing of your application, and implement the advanced features.

## 7 Deadlines and Deliverables

The project is to be developed mainly during the lab classes. The project delivery deadline is **May 7 at 17h CET** and the presentation dates are available on the PUT site (see file with the detailed plan).

There are two deliverables:

- A report describing the implementation details of the mobile application. 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).

- The source code of the Android Studio project packaged in a ZIP file
  In Android Studio choose *File -> Export to Zip File*

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 (3 each advanced feature);
- 2 points: robustness;
- 2 points: report
- 2 point: code quality (e.g. readability, well structured, factorization).