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

Group: 4

Navn: Emilie Mæhlum E-mail: emilima@ifi.uio.no

Navn: Rune Hovde E-mail: runehovd@ifi.uio.no

1 Achievements

Baseline	Login	Fully implemented
	List Customer Information	Fully implemented
	Submit New Claim	Fully implemented
	List Claim History	Fully implemented
	Read Claim	Fully implemented
	Read Claim Messages	Fully implemented
	Logout	Fully implemented
Advanced	Off-line mode	Not implemented (started)
	New Message Notification	Partly implemented

2 Mobile Interface Design

There are small differences between the initial wireframe sketch and the final Android layouts, mainly in the styling of the app. Not everything looks “as good” as in the wireframes, but the layouts are pretty much the same.

3 Baseline Architecture

3.1 Data Structures Maintained by the Application *Describe which data structures you maintain during the execution of your Android app and how you maintain them.*

For the data; all data is stored locally in the class, except for values added to extras and passed to new activities via intents.

We are using the data models and the “WSHelper” from the “lab4autosureapp”.

All activities have a private class which is a subtask of AsyncTask. In this class, we do the necessary WS calls for the specific activity.

Describe the conceptual solution and implementation of each of the baseline functionalities listed in the following subsections.

3.2 Login and Logout

The login activity has two fields, one for the username and one for the password. When the user clicks the log in button, the activity sends a request to the WS with the input in the two fields. If the request returns a valid session id, the user will be able to log in. If not, a message is shown on the screen with a message “Wrong username or password”.

This functionality is implemented in the login activity.

3.3 List Customer Information

List customer information presents 5 fields to the user - name, fiscal number, policy number, birth date, and address. Whenever this activity gets presented to the user, it is made a call to the WS to get that users details based on the sessionID.

This functionality is implemented in Customer_details activity.

3.4 Submit New Claim

Submit new claim contains 4 text fields - title, plate number, occurrence date, and description. Note that the plate number field is not a dropdown, but just a plain text field with available plate numbers as a hint. The user won't be able to submit a new claim without filling in all of the fields with a valid plate number. The user will get an error message if any of the fields are blank or the plate number is invalid. If all of the fields are filled in with valid data they will be able to make a new claim and are navigated back to the main (navigation) page.

This functionality is implemented in the newClaim activity.

3.5 List Claim History

List claim history presents a field for each of the user's claims. This is made so each field appears dynamically based on the number of claims the users have made. If the user files a new claim, there will appear a new field for the new claim in the claim history.

This functionality is implemented in History. It also uses RecyclerViewAdapter to make all the fields for the claims.

3.6 Read Claim

Read claim shows information about a specific claim. It contains 5 fields - title, claimID, plate number, occurrence date, and description. Whenever this activity gets presented to the user, it is made a call to the WS to get the details on the claim based on the sessionID and claimID. It also has a button which you can press to open an activity which shows all the messages associated with that claim.
This functionality is implemented in HistorySpecificClaim.

3.7 Send Claim Message and 3.8 Read Claim Messages

The functionality for reading and sending messages is all done in the same activity, messageListMain. The layout_list_messages.xml layout file has three TextViews that has the message, date it was sent and the sender. This layout will appear dynamically based on the number of claim-messages the claim has. If the user sends a new message, the new message will appear.

The activity_message_list_main.xml is where the actual layout is. It has a button and a textfield. When the user presses the "send"-button, the message in the textfield is sent to the web-service and the message will appear.

This functionality is implemented in RecyclerViewAdapterMessages. This is where the layouts are adapted.

4 Advanced Features *Describe any implemented advanced features.*

Off-line functionality: Barely started on it. We have just made a class LocalStorage with some simple functionality to store and read data. This is not used any other places in the project yet.

Notifications: This was started on, but not completed. We had some problems with trying to get the threads or services to sleep/wait so that they wouldn't continuously make web-service calls and crash the app. We tried using a broadcastReciever, but to no avail. We ended up having to take everything away, even though we almost got it to work if you pressed a button to check for new messages. What is included is the last try we did, a service FindNewMessages. This is not connected to the app, but is still included in the project.

5 Limitations *List the currently know limitations of your project (e.g., bugs, restrictions in the test environment, etc.)*

There are no off-line functionality or notifications in the project. There are no known bugs in the project. The project is not thoroughly, so there may be some bugs we don't know about, but the suggested tests from the testing document work fine. We assume that the user is using the app in the way it is supposed to, but it may be errors when using the app in other ways.

6 Conclusions *State the conclusions of this work.*

This project has been very interesting, a lot of fun and we have learned a lot, while also giving us a lot of hours of frustration. All in all, we are happy with the result, but we are aware that it is not perfect and there are several things we would have done differently. Looking back we wish we would have planned more, like the architecture of the app, how it is supposed to work and the structure of the code. Also, we would like to have documented the code better and been more consistent. If we were more forward thinking with respect to how we managed information, we think we would have an easier time getting the advanced features working.

Also, none of us are able to go to the lab classes due to conflicting commitments, so we both feel like it would have been better if we were able to go in these classes to get some advice.

Please provide some input on how the practical component of the course could be improved in future editions.

The practical component of the course has been good. It is nice to relate the theoretical part of the course to the practical part. It would have been nice if there were some different project to choose from.

The hardest part about starting a project like this is “best practises” and usual implementations of the different functionalities.

A suggestion to improve the practical component could be to have a small obligatory assignment before starting the project to get to know the tools we are working with and getting a little feedback on the work we have done. Another alternative could be to deliver the project in more than one delivery, so we would get feedback along the way.

7 Annex

7.1 Application wireframe