IN5600 – Course Project

Group 8

Henrik Torland Klev, 580965, henriktk@ifi.uio.no

Hans Christian Tranøy Jenvild, student number(?): 559133, hanscje@ifi.uio.no

As of 06.05.2019 neither of us have a candidate number available at studentweb.uio.no.

1. Achievements

Version	Feature	Status
Baseline	Login	Fully
	List Customer Information	Fully
	Submit New Claim	Fully
	List Claim History	Fully
	Read Claim	Fully
	Read Claim Messages	Fully
	New Claim Message	Fully
	Logout	Fully
Advanced	Off-line Mode	Fully
	New Message Notification	Fully

2. Mobile Interface Design

When starting out, we made a few key decisions regarding design. It was decided that we would like the visuality to be as simple as possible. The displayed information is stacked in key-value pairs, scrollable where needed, and navigations is done mostly via buttons or the simple side-menu. The only exception to this navigation, is in the History-activity. Here, you enter the information regarding a specific claim by clicking on the name of the given claim.

A sketch of the UI is attached.

3. Baseline Architecture

3.1 Data Structures Maintained by the Application

There are a few components that we'd like to mention here; the menu, the Async WS-caller, Connection-Checker, and the OnActivityResult-methods.

Menu

The menu/sidebar is displayed on all activities, except the Main (Login) activity. This component is not as modular as one would hope, as each activity has a slightly modified version. We would ideally like to make this into a class, as to not have such duplicated code throughout the project. All this aside, we find that the sidebar is ideal for its simplicity and familiarity. It does the job that needs to be done.

The Async WS-caller

This "component" is the engine behind all network traffic to and from the server. All calls are made thought an AsyncWebServiceCaller, which has an execute-method. This execute-method takes a string-array with the parameters needed for a specific server-function. When the execute-method is done, the processFinished-method is executed in the calling activity.

The processFinished-activity is called with the result from the AsyncWebServiceCaller, and the result is handled there. The result is handled as a pure string-object, and in some cases it is better to have it stored as a JSON-object or in a HashMap of sorts.

Connection-Checker

This is a thread that is created on application startup, which constantly checks if the server is available. It checks every 4 seconds, until a connection is established. After a connection is established, it checks every 12 seconds to ensure that there is still connectivity.

OnActivityResult

This method is also somewhat of a component in the project. All activities that start other activities have this component. This is what handles the logout, either in case of user-initiated logout, an invalid session ID, or some other user-related error.

3.2 Login and Logout

First time the user tries to log in, the user will provide username and password. If unsuccessful, it will be denied. If successful, it will add the username and password to an Account and adds it the phones connected accounts. If a user has an account connected to the phone, it will ask the user if it would like to login with that account. In this case, the username and password from the account is retrieved, meaning the user won't have to retype it. The session ID is cached upon successful login and is not removed unless the user logs out (or it becomes invalid due to server reset). This means that the user will stay logged in except for in the case of logout or server crash.

If the user logs out, the application will backtrack through all the activities navigated between and finish them all. It will also set the corresponding error-codes. When returned all the way back, remove the session ID from cache and null out pointers to it. Then, logout and display a message depending on successful/unsuccessful return from server.

3.3 List Customer Information

Happens when user is logged in and is displayed on the home screen. The customer info retrieved from the server is stored in a HashMap. Further, TextViews are created in Key/Value-pairs, constrained to the layout, and then displayed to the user.

3.4 Submit New Claim

Got some input-checking (e.g. plate number has to be between 6-10 characters), and a data-picker has been added such that there will not be problems with date-format (e.g. mm/dd/yyyy vs dd/mm/yyyy).

If successfully submitted, display that to the user. Otherwise, display that something went wrong.

3.5 List Claim History

Works very in the same way as List Customer Information, except the view is scrollable to support long list of claims. Can click the name of a claim to see its details. We had an idea to implement sorting of claims, but we fell a bit short in time.

3.6 Read Claim

The read-claim activity shows the information of a given claim. When the information of a claim is retrieved from the server, the messages are also retrieved in order to display how many messages are in a given claim. In order to not discard these messages, we decided that we would pass the messages in the intent-extras from the read-claim activity, to the chat-activity. This does however cause an issue where the chats message cannot be updated without going from the Chat-activity to the Read-Claim activity and back to the Chat-activity. This is a rather dirty fix that was implemented to update the messages without increasing the complexity of the Chat-activity.

3.7 Send Claim Message

See next subsection.

3.8 Read Claim Message

Read Claim Message and Send Claim Message are implemented in the same activity (Chat-activity). All messages for a claim are stored in an array list, and then displayed, with Server on the left and User on the right. On the bottom of the activity, the user can send a new message for a claim. When the new message is sent, the dirty fix is initiated. The activity finishes back to the read-claim activity, the new messages are fetched, and the chat-activity is started again with the new intent extras (new messages). This is because the server adds sender and date to the message, and we did not want to add them ourselves, as to keep consistent with the data from the server. Therefore, after sending the message, we had to retrieve it again from the server and not just append it to the array-list.

4. Advanced Functionality

Offline Functionality

All data retrieved from the server is cached to temp-files. If the connectivity the server goes away, then the project will read data from the cached file. The caches are overwritten when new data is finally retrieved.

Message Notification

There is a thread created at application start, that check through all claims and reads the number of messages in all claims. If the number of messages is greater than in the last iteration, then a new message has been added. In this case, notify the user.
A limitation of this implementation is that a notification is sent when the user adds a new message. This is possible to avoid, but would require either communication between the chat-activity and the notification-thread, or more checking and complexity in the notification-thread.
We wanted to ensure that no notification would be sent the first time the application ran, and therefore we said that if the previous number of messages is 0, then it’s the first iteration. This has the side-effect that if there are no message in any of the users claims, and a new message is added, it will not notify the user. This could be solved by creating a flag that is active only the first iteration and check for the flag instead.
As described in the code, there is currently no way for the notification to tell which claims has new messages. We had an idea to use counters and arrays to provide this ability, but we decided against this, as our solution still provides the required functionality.

5. Limitations

The current known limitations are already described. These are

- Chat-messages are clunky to update when a new message is sent
- Notification System
    - New notification when user sends a message.
    - If total messages goS from 0 to 1, no notification is sent.
    - No way to tell which specific claim has new messages.

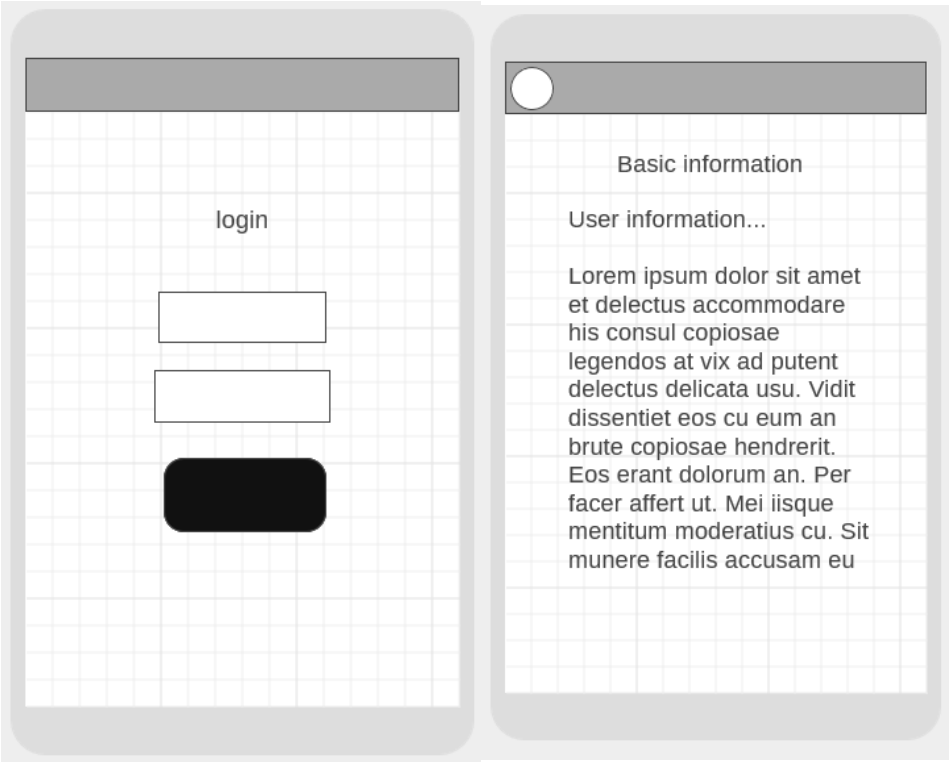
6. Conclusion

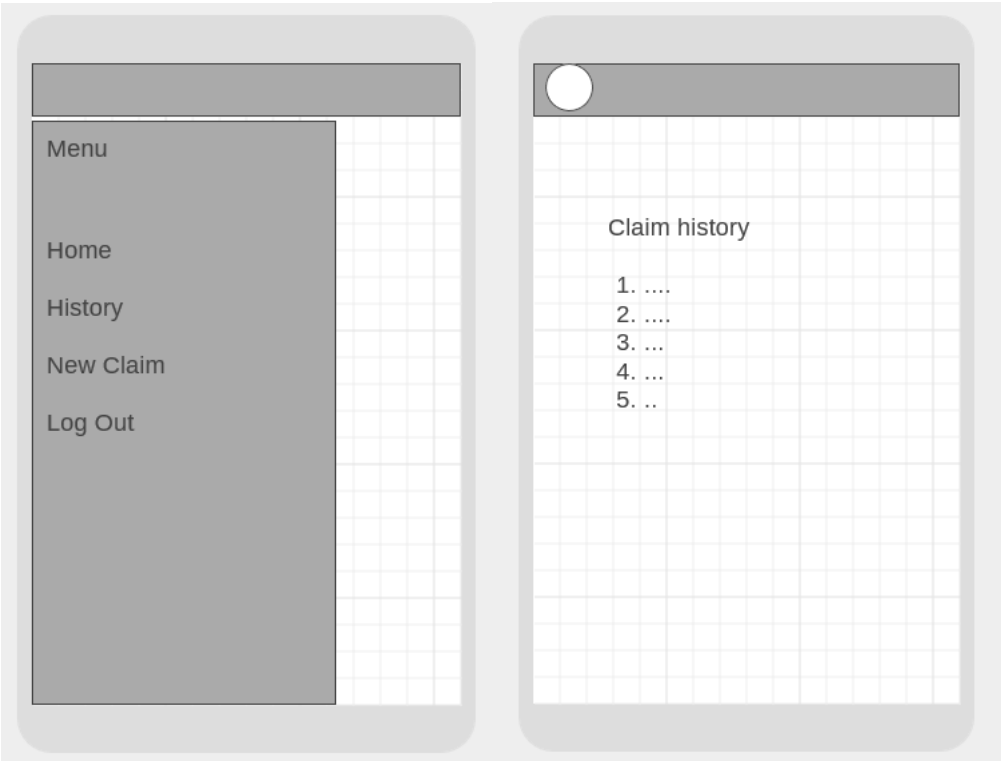
We feel that we have implemented the required functionality to a satisfactory degree. The implementation is rather simple, and when one understands the base-components described in 3.1, then the code is rather simple to read and follow. The UI is uniform and kept minimalistic, the offline-mode works like a charm, and we have added extra functionality (e.g. connectivity-checker, accounts) that was not required just to give the app a better and more complete feeling when using. This app has way beyond 100 hours or work added into it, and we feel proud with what we've accomplished.

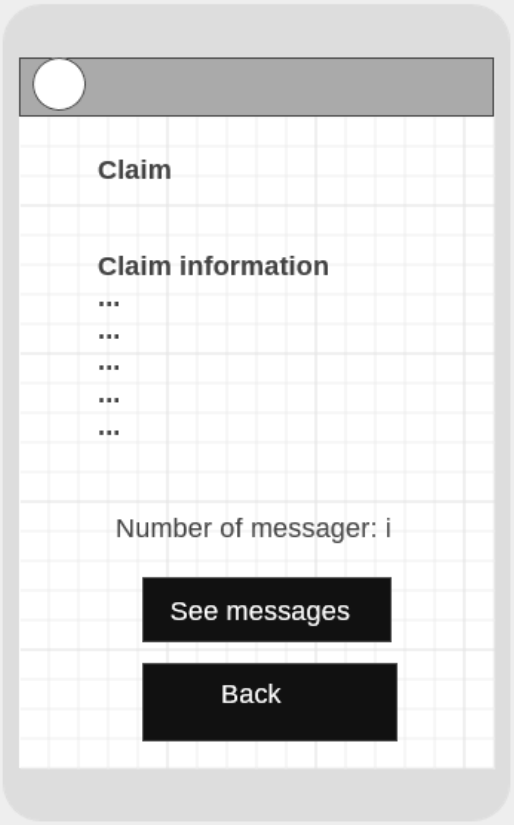
Would be great if the server API was RESTful, which would make it easier to get some of the information from the server. Also, resetting the server resets the session ID, which renders all previously created session IDs useless.

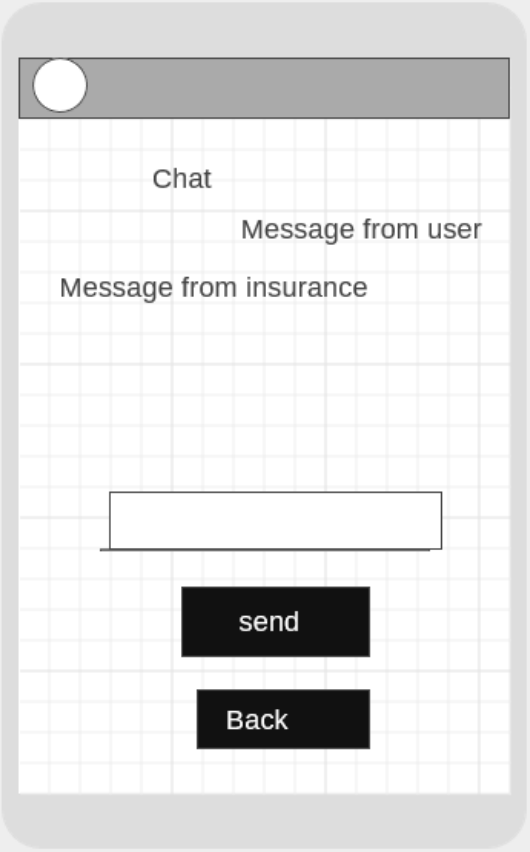
7. Annex

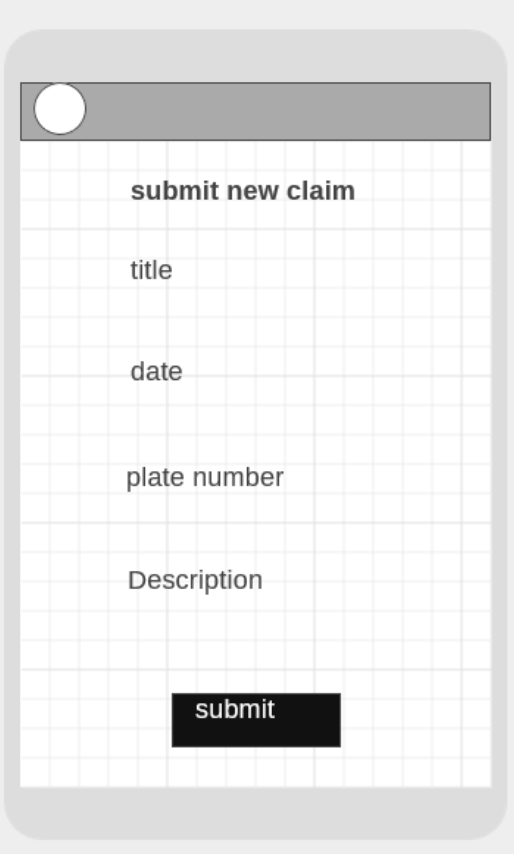
7.1 Application Wireframe


login
Basic information
User information...
Lorem ipsum dolor sit amet et delectus accommodare his consul copiosae legendos at vix ad putent delectus delicata usu. Vidit dissentiet eos cu eum an brute copiosae hendrerit. Eos erant dolorum an. Per facer affert ut. Mei iisque mentitum moderatius cu. Sit munere facilis accusam eu


Menu
Home
History
New Claim
Log Out
Claim history
1. ....
2. ....
3. ...
4. ...
5. ..


Claim
Claim information
...
...
...
...
...
Number of messager: i
See messages
Back


Chat
Message from user
Message from insurance
send
Back


submit new claim
title
date
plate number
Description
submit