|  |  |
| --- | --- |
|  | **Project Assignment for the Course on**  **Programming Ubiquitous Things**  **Spring 2021**  **IFI/UiO** |
|  |  |

**Template for Group Project Report**

**(an insurance company)**

Group: G5

Name: Vemund Sjøvold Sundal E-mail: [vemundsu@uio.no](mailto:vemundsu@uio.no)

Name: Hina Shahzad E-mail: hinas@uio.no

# **1** **Achievements**

|  |  |  |
| --- | --- | --- |
| **Version** | **Feature** | **Fully/Partially/Not implemented?** |
| Baseline | Login | Fully |
| Logout | Fully |
| Modify the current password | Fully |
| Access claims | Fully |
| Insert new claim | Fully |
| Update claim | Fully |
|  | Off-line operation | Fully |
| Advanced | Map of the accident | Fully |
| Photo of the accident | Fully |
|  | Answers to the questions | Fully |

# **2** **Mobile Interface Design**

The difference between the initial wireframe and the final product is minimal. Some are based on the fact that we were not fully familiar with the server code and task description that we had parts and functionality in the wireframe that was not correct. Example a person information page and delete claim button, they were not needed so we did not make them. The other change is that we shuffled the fields in read and edit claim.

# **3** **Baseline Architecture**

## **3.1** **Data Structures Maintained by the Application**

In the vehicle insurance application data structures that are mainly used during the implementation of the baseline architecture are Array, objects and files. Array data structure has been used to store the claims information. For instance, in the vehicle insurance application ClaimResponse is a class that contains a number of claims, claimsId, claimsDes, claimPhoto, claimLocation, claimStatus and each claims information is saved in an array data structure against each userId.Person.json and claim.json file is also a data structure that is mainly used to implement the baseline architecture of the application. Server is responsible to save the user data and claims data in the form of json data structure. Reading and writing through these json files makes possible to implement the vehicle insurance application baseline functionality. In addition, claimPhoto taken by the client is saved into the claimPhoto array in claim.json file and is also a data structure to make the implementation possible.Two objects person and claims has used to access the person information for instance, email id, password, user id etc and claim object contains all the information about the claims for instance claim description, status, location, claims id.

## **3.2** **Login**

Launching the mobile application it opens the LoginActivity. The user authenticated himself by providing the email address and password, that is encrypted to MD5 hash. User enters the same email and password that is stated in person.json file and after the verification they are successfully logged on the mobile application and then open the ClaimOverviewActivity. First time the client logs in it will be through the remote method PostRemoteLogin. Later it will first try to login based on the locally stored email and password.

## **3.3** **Logout**

Logout will open the LoginActivity while clearing all tasks and creating a new history stack. On logout all claims and user data related to the user that logout is deleted from the device. The logout is a menu option on the claim overview page.

## 

## **3.4** **Modify Password**

In the options menu you have a change password item. When the user selects that menu item a dialog box will appear. To modify the password, the user must enter the correct current password, new password and repeat the new password. If input is correct it will update the SharedPreferences and invoke remote method /methodPostChangePasswd on the server.

## **3.5** **Access Claims**

The claims are downloaded from the server by the help of Volley, all the communication connected to claims is in a communication layer, ServerCommunicationInterface. The communication layer uses the userId passed from LoginActivity. When the claims are downloaded from the server they are parsed to a ClaimResponse object by the use of GSON. Then the ClaimResponse object is mapped to an array of Claim objects. This array is inserted into a custom Adapter named ClaimsListAdapter. ClaimsListAdapter creates a clickable list of claims that takes the user to the specific claim. The specific claim is displayed by passing the information as String to the ClaimDetailsActivity that displays the information in the ReadClaimFragment. When the claims overview page is resumed it will reload the claims.

## **3.6** **Insert new claim**

The top button on the claim overview is an add claim button. The button takes the user to ClaimDetailsActivity and shows an empty AddEditClaimFragment. The fields can be filled in and when pressing the add button the information is passed to ClaimDetailsActivity by a listener. The claim is sent to the server by the use of the communication layer, then a ReadClaimFragment is opened to display the added claim. The client can not add more than five claims.

## **3.7** **Update claim**

Update claim is opened by pressing the edit button on the read page of a claim. The button opens an AddEditClaimFragment that contains the data from the claim. Here the user can change or update the information of the claim, then press the save save button to pass the information to ClaimDetailsActivity by a listener. The claim is sent to the server by the use of the communication layer, then a ReadClaimFragment is opened to display the updated claim.

## **3.8** **Off-line operation**

The claims are saved to SharedPreferences(SP) when they are downloaded from the server. Aswell whenever a claim is added or updated the SP is updated. When a request to add or update a claim fails. They will be saved in SP “needUpdate” by the id of the claim missing an update or add. When the application later updates the server it will find the id of claims that have been updated or added, then get the claim information from SP. This way you avoid doing multiple calls if a claim has been updated multiple times. When a missing update is added to SP it will first check if the same id is in the add missing list, this is also for it to only run once if it was added then updated offline. All the update server from offline storage is done in the UpdateServerFromLocallyStorage, if a claim fails to be added or updated it will be added back to the missing lists.

Password hash and email of users are saved to SP on login or password change. These will be used for offline login. The offline login works by getting the userId of the email and password hash that has been written in, if they exist in the SP they will return a userId if the userId matches the login is successful.The password hash is a StringSet to handle multiple users with the same password. All handling of SP is done in the SharedPreferencesHandler class.

# **4** **Advanced Features**

## **4.1** **Data Structures Maintained by the Application**

For saving and sending image files a byte array is used. To maintain them they are stored in a file.

## **4.2** **Map of the accident**

The map is displayed in a MapFragment. The first way to use the map is as a read map. Here you can’t change the marker, it will just be the map with a set marker. This can be used by pressing the location button in the read page for a claim. The button is gray and dissabled if a location is not set.

In the edit page the user can also press the location button but here it will open a map with a draggable marker. If the user presses the select location button the latitude and longitude is passed back to the AddEditFragment through the ClaimDetailsActivity, where it will set the location on the claim. The map is implemented by the use of GoogleMap. The MapFragment is laid on top of the AddEditClaimFragment so the claim’s unsaved changes are not removed. It has two different button layouts that it will show or hide based on if it is an edit or read map.

## **4.3** **Photo of the accident**

The photo is shown in the claim read page, if the photo is not in the described file location it will download the image file from the server to display it and save it locally. When selecting a picture the user will press either the add or change image button this will open the inbuilt action\_pick intent and then get the picture from onActivityResult where it will first start to create the file and upload the picture to the server after the save or add button for the claim is pressed. If the upload of the picture failed or the client is not connected it will still store the file and mark it for need upload in SP.

## **4.4** **Two Servers**

*e.* I understand “done correctly” as “correct way to simulate the real case”. In that case yes it is a solution to simulate the client connecting to a new server. The way it’s not correct is that in reality the servers would most likely both be running at the beginning.

*f.* This could happen in multiple ways, one the first server disconnects or simply that the client is closer to server-2 and his request gets routed to the other server.

*g.* I have described two real life scenarios and on based on how this assignment works

1. Is to have server-1 update server-2. I prefer this solution since in a realistic setting you have to update all information between server-1 and server-2. Not only login but also all claim information. This solution also makes the functionality not relaient on the user to update the server-2, but the service provider can control it themself. This also makes it possible to use multiple devices. Depending on the size of the network (number of servers and geographically) I would solve this in different ways.

If it is a small network the result from the update password would not respond to the user before all servers are updated. On a larger network I would start a worker that will update all the servers. I would also have new servers to look for updates from other servers under boot before going online.

1. Is to store previous passwords on the client device then when login try to login with old passwords, then update the server by sending a request. This has the pro of that there is no delay between when you can login to server-2 after disconnecting from server-1. The downsides is if you are to login to the server from another device it will not work and minor security risk since you should not store old passwords.
2. The reality is that there is no form of token or any other validation to do calls on the server. Because of this we don’t need to login on the server to update a password. This means we only need to perform a local login on the device then try to update the server. Since we don’t know if we are connected to a new server we will always try to update the server, if the password has ever been changed locally. Even if it’s solved like this it will be the same number of requests, since we don’t try to login remotely.

*h.* Used number 3. quite simply when the password is changed it changes the local login information in SharedPreferences and makes a SharedPreference for updated on the email. Then on login if local login is successful check if the email is marked for update. If it is then set the password the user just typed in as the new password. This way we avoid storing the plain text password also.

# **5** **Limitations**

* If status can be overridden by the client if the client updates the claim between when the claim was downloaded and updated from client side. This window could be big if the client is not connected to the server.
* Does not delete image file when logout
* If server connects between update and getClaim the update can be overwritten
* If you change password on the server you can still login with the old password if it is stored on the device, it will even overwrite the password if the password has been changed on the device.

# **6** **Conclusions**

Vehicle insurance application is the application for the clients of insurance companies. The client is able to use the different services provided by the insurance companies information system through an android mobile services. This application has been designed with login, logout, modified password, add claims, edit claims, set the location of accident, take the photo of the place where accident occured and update the claims on the server side as well as handled locally by the SharedPreferences. Server is responsible to maintain a data structure containing files person.json and claim.json that is used to read and write the data and on the other hand client uses the application and accesses all the data through the volley library.

# **7** **Annex**

## **7.1** **Application wireframe**





