**Mobile Application Implementation**

For the implementation of the mobile app the android studio [1] is used. The reason we choose to use android studio it’s because is easier for the mobile app to access the mobile local files in an android phone. It is for same reason that also hybrid development framework could not be used as it would be difficult to handle local file on both operating systems using the same codebase. Also, some external libraries where used to help do things like sending requests to the server which we will discuss later on.

**Implementation logic**

Generally, the mobile app will use the internal storage of the device to store information needed for the version control of the local files. The actual local files of the user will be stored in an external storage directory called WorkingDirectory which is created the first time the user lunches the app. Also, during the first lunch the app will ask the user for the necessary permissions to access the local file system and use the internet to send requests to the server.

If the user is not logged in the usual logic screen will appear to him with also the button that will direct him to another form to register if he does not already have an account. From the point that the user logs in he will remain logged in until he chooses to log out and for every time he opens the app he will get the main screen of the app. The main screen of the app will contain a list of the following:

* All the file names
* Show which files are only stored locally on the phone
* Show which files are only available online on the server
* Show which files are both online and locally and are updated
* Show which files are both online and locally places and are not updated
* Next to each filename a button with the appropriate action if there is
* Next to each filename a delete button

To obtain the list above the following implementation logic is used:

* Save all the file names from the local storage in a list
* Save all the file names you get from the server in a list
* Compare lists to find out which files are only local which only online and which on both
* Put them in a single list with a code field with value depending on their status
* Initiate the adapter that will create the list of files
* For each file do the following
  + Check the status of the file
  + If file is local
    - Show local file symbol next to file name
    - Initiate an upload button that on click does the following
      * Uploads the file
      * Saves the file version as 1
      * Refreshes screen
    - Initiate a delete button that will trigger the delete file dialog
  + If file is only online
    - Show online file symbol next to the file name
    - Initiate a download button that on click does the following
      * Downloads the file from server
      * Saves the file version as the version that is stored on the server
      * Refreshes screen
    - Initiate a delete button that will trigger the delete file dialog
  + If file is on both places
    - Initiate a delete button that will trigger the delete file dialog
    - If stored file version is different from the version stored on server
      * Show the not updated symbol next to the file name
      * Initiate a conflict button that will trigger the conflict dialog
    - If stored file version is the same as the version stored on server
      * If local file hash is different than the file hash stored on server
        + Show the not updated symbol next to the file name
        + Initiate a synchronise button that will

Upload the file

Increment the saved file version by 1

* + - * If local file has the same hash as the hash stored on the server
        + Show the updated symbol next to the file name

As you can see from the logic, a local file is kept on private storage that will contain all the file versions of the local files which are compared with the file versions stored in the server to determine if a file is up to date or not and what action is required. Then the versions will be updated on the local file based on the action performed by the user. That way we can detect potential conflicts that will occur if a user tries to update a file that has a more recent version stored on the server. When a conflict is detected as soon as the user initiates the action a conflict dialog will appear that has three options. Either for the user to force upload the file, just download the file version or get a diff file that will be generated and saved on the local storage with the same name as the file the user tried to upload. The logic for those actions is similar to the logic discussed above. Also, for the delete dialog the user will always have the option to delete the file either from the server or from local storage or from both places with the version of the file also deleted from the local information file.

Now let’s discuss the different external libraries that were used

**OkHttp**

OkHttp [2] is a library available on github that allows you to perform get and post requests on a server. The reason we used this library is for its simplicity into the building of the requests and that it allows you to easily carve multipart post requests that accept a binary file in the post request’s body that we will use to upload the files to the server.

**PersistentCookieJar**

PersistentCookieJar [3] is an extension to the OkHttp library that allows you to easily capture session cookies and reuse them in your OkHttp requests. The main reason we use it for its simplicity that will prevent us from having to manually process the session cookies.

**Differences between mobile application and desktop application:**

* Mobile Application saves the generated diff file with the same name as the file the user tried to upload but desktop application just shows the diff file to the user on screen
* Desktop Application shows information about each file’s size while the mobile application does not
* In the desktop app the user has the option to choose a local directory for his while in the mobile app a default directory is created that is always the same

[1]

|  |
| --- |
| @misc{androidstudio, |
|  | title = {Android Studio}, |
|  | howpublished = {\url{https://developer.android.com/studio}}, |
|  | note = {[Online; accessed 20-January-2019]}, |
|  | keywords = {cite} |
|  | } |

[2]

|  |
| --- |
| @misc{cookiejar, |
|  | title = {OkHttp library}, |
|  | howpublished = {\url{https://github.com/square/okhttp}}, |
|  | note = {[Online; accessed 8-February-2019]}, |
|  | keywords = {cite} |
|  | } |

[3]

|  |
| --- |
| @misc{okhttp, |
|  | title = {PersistentCookieJar library }, |
|  | howpublished = {\url{https://github.com/franmontiel/PersistentCookieJar}}, |
|  | note = {[Online; accessed 25-February-2019]}, |
|  | keywords = {cite} |
|  | } |