Project Report

Ubiquitous and Mobile Computing - 2018/19

Course: MEIC

Campus: Tagus

Group: 15

Name: José Canana Number: 82039 E-mail: jose.canana@tecnico.ulisboa.pt

Name: José Brás Number: 82069 E-mail: jose.bras@tecnico.ulisboa.pt

Name: Diogo Cardoso Number: 94024 E-mail: diogo.f.cardoso@tecnico.ulisboa.pt

1. Achievements

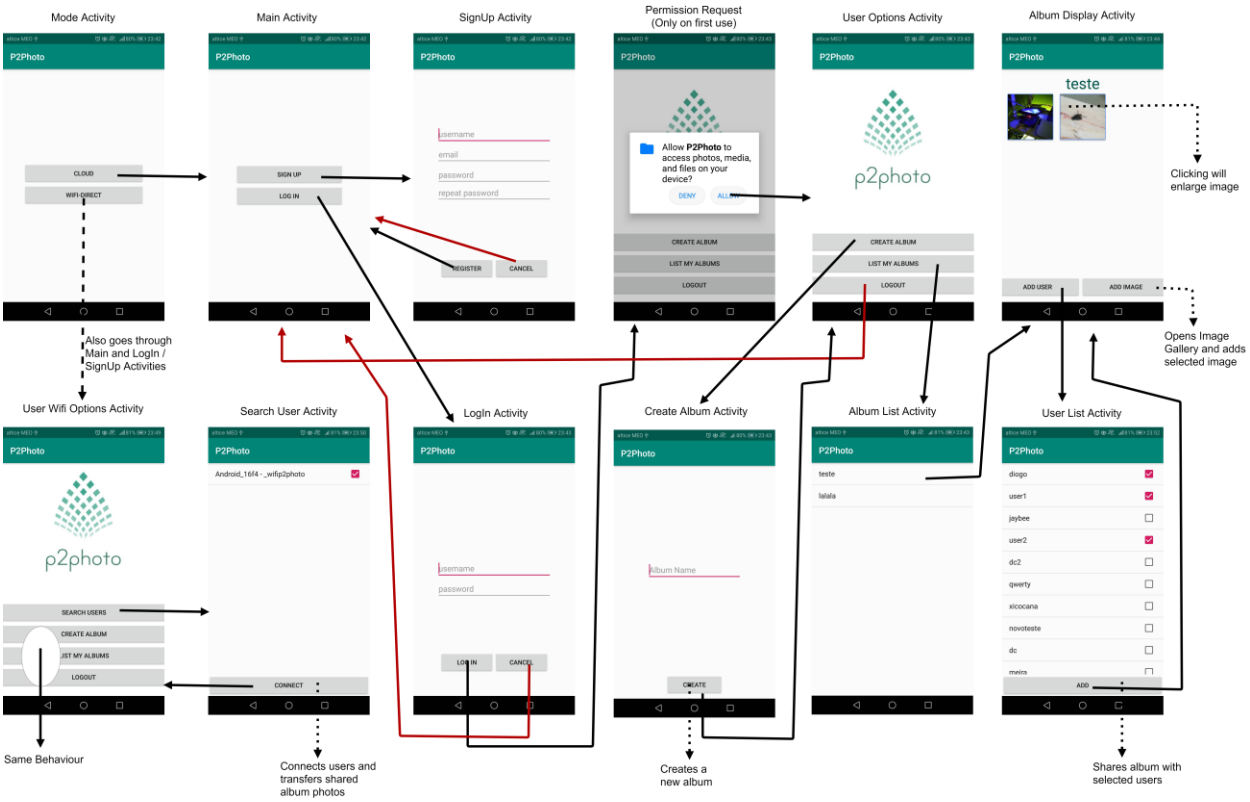
The cloud mode was completely implemented, successfully accomplishing all the project requirements. On the Wireless mode we ran into some problems. The *find users* feature only seems to work on some devices and the creating of the ad-hoc network is somewhat inconsistent, making the user having to restart the activity in order for the connection to be established. Due to the previous problem, the sharing of albums between users have the same inconsistency and as a result, in most of the occasions, only the users photos are displayed and not the entirety of the album.

Version	**Feature**	**Fully / Partially / Not implemented?**
Cloud Mode	Sign up	Fully
	Log in / out	Fully
	Create albums	Fully
	Find users	Fully
	Add photos to albums	Fully
	Add users to albums	Fully
	List user's albums	Fully
	View album	Fully
Wireless Mode	Sign up	Fully
	Log in / out	Fully
	Create albums	Fully
	Find users	Partially
	Add photos to albums	Fully
	Add users to albums	Fully
	List user's albums	Fully
	View album	Partially
Advanced	Security	Not Implemented
	Availability	Fully

Optimizations

To reduce the waiting time and improve the user experience while the album is loading we have implemented a loading icon and the images appear one at a time (as they load), also, this is done in a background process so the current activity shows no lag or abrupt interruptions.
All actions and their consequent conclusions are followed by toast messages to inform the user of the current state of the application and its actions.
On the advanced feature *Availability* we opted to replicate all the user's albums into the device's disk memory, this proved to be only a little slower than the previously implemented memory LruCache (almost unnoticeable) and gives the user the possibility to view the albums even if offline.

2. Mobile Interface Design



We have chosen to implement a simple information flow with the goal to provide to the user an easy to use interface. After an initial choice of mode (Cloud or WiFi-Direct) the user is presented with the options of creating a new account or logging in to an existing one (if a session is already in progress on the device this options will not appear until a log out request). The main activity is a simple menu with all the basic actions a user can perform (create and list albums). The options to add photos and share the album are only presented while viewing the corresponding album in order to facilitate the user actions.

Both modes have similar interfaces being the only difference an extra option on the WiFi-Direct main menu where the user can find nearby users and consequently create a wireless connection and share photos. The remaining flow of this app mode is equal to the previous one.

3. Cloud-backed Architecture

In this mode, we use the new google drive rest API to save and load photos from the user's account. The users Albums are maintained by our java server, although it has no permissions to access the album's photos. The server is also in charge to register new users and validate the access of current ones.

3.1 Data Structures Maintained by Server and Client

Starting with the server, this module utilizes JSON files as resources to save and load user and album information. The user's file saves usernames, passwords, emails and session ids (one token per session). The user album's file saves the album name, its google drive id and the corresponding album catalog drive id. The user album's file is also used to store the corresponding users that share the album.

On the client side, only the username and session id are maintained to perform server operations that will return all the necessary information. The server name and port are hardcoded in the client application.

3.2 Description of Client-Server Protocols

The client-server communications were developed using sockets due to its easy usability and low abstraction level giving us more control over the shared information between both ends. The information is shared using JSON files due to its non-standard form making the messages more appropriate to the different possible replies.

In the beginning of each client request, a first message is sent with the corresponding remote method command, this message prepares the server to respond to the desired client need. On every method conclusion, the server returns to the client a success message, with all the requested information, or an error message followed by the corresponding exception.

In the end of the communication, a final message is sent by the client to successfully close the socket and make both ends disconnect securely.

The server functionalities are as follows:

- Login: The client sends its credentials and receives or not connection to the system.
- Sign-up: The client sends its credentials and a new record is created on the server side. Usernames are unique and passwords have secure restrictions (size, special characters, …)
- Add Album: The client sends the album name and a new record is created on the server side. We assume that album names are unique. In case of cloud mode some other fields are also sent (album drive id, catalog drive id, …)
- Get Albums: The server sends all the client's albums, this is, the albums owned by the client and all others shared with the user.
- Get Token: The server sends the session id giving the client access to the application. This information is stored on the device making the client connected until a Logout request or session expiration.
- Get Users: Returns all active users in the application.
- Add User: The album owner can call this method to add more users to the respective album.
- Exit: Client can calls this method when it pretends to disconnect itself from the server. This method exists so that both ends can close the socket channel safely.

3.3 Other Relevant Design Features

In order to provide a better user experience, and as part of an advanced feature, an image cache is used (in disk) to store replicas of the albums. In this mode the cache is used to store the entire album and makes it available to the user in case on disconnections, which prevents the app to make google drive download requests.

4. Wireless P2P Architecture

In this mode, all albums and corresponding photos are stored locally on the user's device and the server remains with the previous functionalities and restrictions. In order for the users to share albums and transfer photos between the devices a wireless ad-hoc network is established.

4.1 Data Structures Maintained by the Mobile Nodes

The applications makes use of the device's local storage and in an initial setup a folder for the application resources is created, this is where the albums will be stored. For every album created a child directory is made.

4.2 Description of Messaging Protocols between Mobile Nodes

The mobile nodes connections are created with only two elements, a group owner (the server node) and a peer (the client node).

In the beginning of the communication, both nodes are trying to discover one another and the first one to succeed / connect becomes the group owner. After the initial setup, both usernames are shared and an internal server method is called to find what albums (if any) these clients share with one another. In case of albums shared, both clients start to send their owned photos, one at the time, and they proceed to be store them on the other users WiFi cache.

When the photo share process ends, the group owner closes the connection, and both users are free to proceed using the application.

4.3 Other Relevant Design Features

In this mode, an image cache was also implemented but this time it is only used to store shared albums, in particular, only other users photos, due to the fact that the owner photos are already locally stored.

5. Advanced Features

5.1 Security

Due to some group communication problems and extensive work in other courses this feature was not implemented in the final version. Although, we already had some thoughts on how to tackle this issue. Since only the catalog files needed to be encrypted to provide this security feature, our main idea was to create a new server method where a unique password would be made for each album in its creation process. This secret would be used to cipher and decipher the album catalog file and would be shared between all the members of the album in question. This would be easy to implement in our current project but as said before we did not manage our time correctly and were unable to finish the feature.

5.2 Availability

Initially we started by implementing an LruCache, using the application internal memory, to help the load of images and to make them more available in case of errors or disconnections. This implementation proved to be inefficient in loading time gain and regularly the image bitmaps would be recycled making them unavailable. To solve this problem we implemented a disk cache (one cache per app mode) where the albums are completely replicated. As said before, on the Cloud implementation the entirety of the albums are replicated but on the Wi-Fi mode only the shared photos are stored. In moments where the application is running with no connection problems, when photos are received or downloaded a copy is made to their corresponding album cache directories. When the application loses connection, an internal application flag is switched, and the normal behavior occurs but using the created cache directories.

6. Implementation

As we were already using Java to develop the application and all elements of the group had similar project experiences with the language, we decided to also use it to implement the server. Sockets were also used to provide all the communications between the clients and the server in order to maintain a similarity with the client-client communications as explained in the previous sections.

For the cloud mode we chose to use google drive as we thought it was widely used and therefore it would have good and available documentation. This was not the case but we were able to learn it and used the new rest API successfully. To perform download operations we started by implementing our own method but after some research we found a better solution using an external framework *Glide*.

In regards to the WiFi-Direct mode we have chosen to use service discovery in order to only communicate with app users. To implement this mode we also used individual threads (for the group owner and the peer) and their corresponding handlers.

Between sessions and activities some information is needed for the application to have its expected behavior. To accomplish this we used the built in android shared preferences to store usernames and session ids, and implemented a new singleton class *DataHolder* to send / store useful information.

In order to safely and securely login into the application we have implemented a Persistent Login util which is in charge of session tokens verifications, this is, it grants user permission to use the service or askes the server for a new token in case the current one has already expired.

7. Limitations

As said before, in the WiFi mode there are some known limitations where some users are unfindable making the album share feature inconsistent and sometimes inexistent. Due to this problem, some of the photos in shared albums are sometimes lost between shares and the process fails. In the current state of the project the users need to attempt the image transfer several times until they are able to succeed. This happens because the clients need to be synchronized in order to the transfer to be successful.

Due to the fact of the already specified album's share problems, some bugs may appear on the album display activity. In case of failed transfer the image may appear sliced or incomplete and in case of total transfer failure the photo will not appear.

In regards to the cloud mode, one future interesting addiction to the project would be the possibility for the user to add photos to albums even if offline. A pendent process would be created and executed when the client reconnected to the Internet.

8. Conclusions

Thinking back on all the time spent implementing the project we were able to discover new technologies and overcome several problems. We started to implement the cloud mode using a deprecated API (checkpoint submitted version) and after losing a lot of functionality as the API functions were being removed we had to completely migrate to the new rest API costing the team several hours of work rebuilding the application.

During the project we also understood the importance of improving the performance of android applications, most of the time being frustrated with our own work.

Overall, this was a very interesting project that made us have to research and learn by our own which, in our opinion, are valuable skills to gain.

The practical classes were helpful and project oriented which also facilitated the project development and our team work.