Project Report

Ubiquitous and Mobile Computing - 2018/19

Course: MEIC
Campus: Alameda

Group: 14

Name: José Peixoto	**Number:** 83492	**E-mail:** joao.m.mexia@tecnico.ulisboa.pt
Name: Miguel Marques	**Number:** 83532	**E-mail:** miguel.soares.marques@tecnico.ulisboa.pt
Name: Pedro Forjaz Figueiredo	**Number:** 83545	**E-mail:** pedronfigueiredo@tecnico.ulisboa.pt

1. Achievements

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	Fully
	Add photos to albums	Partially
	Add users to albums	Fully
	List user's albums	Fully
	View album	Not implemented
Advanced	Security	Fully
	Availability	Not implemented

2. Mobile Interface Design

(The app wireframe is in the last page)

We start by asking the user to select an operation mode. After selecting the mode, the user can sign up and sign in. After sign in, if the chosen option is cloud backed, the user should be redirected to the browser where a Dropbox login dialog will request login credentials and ask if the user accepts P2Photo's access requirements.

Subsequently, the user is presented with the main activity. In this view, the user can log out, returning to the sign up/ sign in view, go to the create album activity to create a new album or view his current albums. If he chooses the latter option, the list of albums will be downloaded from the server and listed. From the list, the user can click on an album and go to the manage album view. This will show in a grid-style list all the photos that belong to that album. Also in this activity, there's a possibility to add new photos to the album, to check out a particular photo by clicking on it and to see all the album members as well as to add new ones.

3. Cloud-backed Architecture

3.1 Data Structures Maintained by Server and Client

The server is implemented using tables/models. The default model is *Users* and it stores usernames and passwords (encrypted) of every user. Also, already implemented by the web framework, there is an *Authentication Token* table that is responsible for keeping all the tokens and associating them to the correct logged in user.

To develop the P2Photo app, we had to find a way to save albums and memberships so, for each one of these, we created a model. The first one simply contains a *name* and an auto-incrementing *ID*. The membership model is more complex since it stores a reference to the *album* (foreign key), a reference to a *user* (also, foreign key) and a *catalog URL* which points to the file on the cloud service that contains the links of all the photos that the user uploaded and that are part of the album. To address the security feature of the project, we added a *key* field to the model which acts as an encryption key to the photo catalogs stored on the cloud service.

Still on the server side, a file is created and updated inside the catalogs folder which contains all the addresses of the photo catalogs that make up an album and the corresponding users.

On the client side, there are models. These models are located in the package *pt.ist.cmu.models* and are an adaptation of the server models. The user model is the only one persistent throughout the app so that user information can be accessed anytime in any activity. This information is only deleted on logout. Still on the app side, the Dropbox API handles all the needed data and requests to log in.

3.2 Description of Client-Server Protocols

The server works as a REST API meaning all the calls are made to a fixed endpoint set in the app's *Constants* class. Also, all the communication is exchanged in plaintext and because Android doesn't allow this, there's a network security configuration file that allows the device to communicate with the specified endpoint in plaintext. This is not safe, for example, to exchange auth tokens, however, in a production scenario, the solution would be to add an SSL Certificate and upgrade the protocol to HTTPS and all the data exchange would be secure.

The app uses the *Retrofit* and *Gson* libraries, which are responsible for handling all the requests and turning the JSON responses into models. The *ApiService* class defines all the call to the server. The following table contains the mentioned requests and the corresponding description. All calls (except register and login) require user authentication with a valid auth token received upon login.

API Call	Description
POST /users/register	Registers a new user
POST /users/login	Logs in an existing user and returns an auth token
GET /users/logout	Logs out an user
GET /users	Returns all the existing users
POST /album/create	Creates an album with the name received
GET /album/{name}/username/{username}	Adds user {username} to the membership of album {name}
GET /album/{name}	Returns all the catalog data for album {name}
POST /album/{name}	Updates the catalog information of the album {name}
GET /album/user/{username}	Returns all the album of which user {username} is member

3.3 Other Relevant Design Features

Because the URL saved on the P2Photo server is a direct link, we can only download the photo catalogs and not edit nor delete them. Also, when trying overwrite a file on a user's dropbox account, the API would throw an exception stating that the upload was not possible. Because of these two reasons, we decided to keep the direct links to the catalogs and, when adding a photo, the catalog is downloaded, a new catalog is generated with previous links and the new one and then upload back to dropbox with a new name, leaving the old catalog there.

4. Wireless P2P Architecture

4.1 Data Structures Maintained by the Mobile Nodes

Each node maintains a catalog file for each album the user belongs to. This file contains the URI of all the photos (encrypted) the user contributed to that album (which are saved on the device).

4.2 Description of Messaging Protocols between Mobile Nodes

After the user has selected the Wi-Fi Direct mode, the node turns the feature on, making itself discoverable and contactable by other devices. When a user decides to manage a given album, it gets the usernames and corresponding IP address of peers in it's Wi-Fi Direct group, it then filters it to obtain the ones it shares the current album with (information obtained by making a request to the server). The node then requests these devices for their photos.

5. Advanced Features

5.1 Security

To ensure security, we created a field on the *Membership* model on the server called *key*. This works as a password for that relationship. For example, in the membership model instance where user ***a*** is a member of album *abc*, there's also a *key* that encrypts the photo catalog that contains the URLs that point to all the photos that user ***a*** has added to the album. Furthermore, this key encrypts all the photos that are pointed by this catalog so that the cloud provider can't see them. All the data is encrypted using symmetric cipher *Data Encryption Standard* working on cipher mode *Electronic Code Book*.

This approach would be totally safe if the key could not be intercepted. Although, like explained before, the communication between P2Photo server and app is in plaintext, allowing the key to be sniffed. We believe that this is not related to the project as this problem has an easy solution which would be upgrading to HTTPS and, in that situation, the explained security mechanism would work.

5.2 Availability

Unfortunately, availability was not implemented.

6. Implementation

The server was implemented in *Python*, using the *Django* web framework. To help develop the REST API, we used another framework called *Django Rest Framework*. To see the server in action, please follow the instructions on the *README.md* file. Moreover, and like we said before, we employ *Retrofit* as an API Client for the app, which makes all the server calls and the JSON decoding much easier. All the network communication is done on background threads because Android does not allow networking to go on the main thread. This is the only situation where there are threads running in the background.

Another design decision was that instead of using Shared Preferences to save user and album data, we used *Hawk* throughout the entire project, a library that uses Shared Preferences however it encrypts all the information before saving it. It is also much simpler to use and much more intuitive, allowing activities to access global data on the app.

To handle catalog and picture downloads, we use a library called *Fetch*. With a very simple configuration, *Fetch* has callback functions that allow the control of the downloads to go much more smoothly than it would be if it had been done using simple requests.

Activities exchange information with each other using *Intent Extras* meaning when we want to change the view, the intent object that is created to change the activity carries this extra information to the new view.

7. Limitations

On the cloud-backed architecture, there are no limitations.

On the wireless approach, because we had difficulty getting devices to find each other (either as peers or in a group), the communication between users was affected. Also, we could not manage to send the photos using the designated socket.

8. Conclusions

In conclusion, this project was interesting and allowed us to understand how the android lifecycle and development kit work. Although, we have to point out the fact that without the use of external APIs (used in real Android apps) to manage server calls and picture downloads, the development would be much more difficult and time consuming. Also, poor documentation and compatibility of the Dropbox Java API (or any cloud provider API) increased the complexity of the deployment the cloud-backed approach. Moreover, the Termite API was not easy to integrate with the project and the compatibility with the Android versions turned out to be a problem for us.


P2Photo
Select Application Mode
CLOUD STORAGE
WIFI-DIRECT
P2Photo
P2Photo
SIGN UP
LOG IN
Sign Up
Sign Up
Username
Password
SIGN UP
CANCEL
Sign In
Login
Username
Password
LOGIN
CANCEL
P2Photo
P2Photo
LOG OUT
CREATE ALBUM
VIEW YOUR ALBUMS
Create Album
Create Album
Album Name
CREATE
CANCEL
Your Albums
Your Albums
Album abc
Album 123
BACK
Album
Managing '123' album
ADD PHOTOS
ADD/LIST USERS
BACK
Photo
BACK
Add User
Managing Users of 'abc' album
Users You Can Add To Album:
b
c
Users That You Share Ownership With:
a
ADD
CANCEL