Project Report

Ubiquitous and Mobile Computing - 2018/19

Course: MEIC

Campus: Alameda

Group: 17

Name: Dorin Gujuman	Number: 83448	E-mail: dorin.gujuman@tecnico.ulisboa.pt
Name: Filipe Miguel Fernandes Martins	Number: 83458	E-mail: filipe.f.martins@tecnico.ulisboa.pt
Name: Hugo Rafael Silva Guerreiro	Number: 83475	E-mail: hugo.guerreiro@tecnico.ulisboa.pt

1. Achievements

Describe which features stated in the project specification were implemented. Fill out the following table. For each feature, indicate its implementation state. If partially implemented, describe what was achieved.

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	Fully
	Add users to albums	Fully
	List user's albums	Fully
	View album	Fully
Advanced	Security	Fully
	Availability	Partially

Optimizations

Describe additional optimizations that you have implemented, e.g., to reduce latency and improve usability

We made a local cache for Cloud Mode, this way we do not need to fetch the photos every time we want to view them.
Every photo is downloaded in parallel.. (Instead of sequentially) .
Every IO operation is made using either threads or async tasks.

2. Mobile Interface Design

(note: In the description we will be referring to the wireframe in the appendix)

When the application starts, the user (1.) must pick the desired mode of operation (Cloud-backed or WiFi-Direct) after that he must either Log-in (2.1.), which may include registration (2.1.1.) and Logging in the cloud provider (2.1.2.) or confirm his Login (2.2.). If the user selected WiFi-direct he must manage his connections (3.).

Once a user is in the Album activity (4.) he can create albums, add users to albums and view an album (5.). In the album view he can add more photos to that specific album.

3. Cloud-backed Architecture

3.1 Data Structures Maintained by Server and Client

- **Server:** Maintains a database with tables containing the following data: Users (id, username, public key, and salted hash of the password), Sessions (cookies of users), Albums (id and name) and finally a table containing the relationships between the users and the albums, including their respective slices.
- **Client:** Each client maintains its username, id, cookie and their own private keys. The client also stores an internal representation of the metadata related to each album and their members. Internally, in the mobile app data, we also store the collected photos.

3.2 Description of Client-Server Protocols

1. ***Registration:*** The client queries the server with the username, password and public key. The server then verifies and saves the data.
2. ***Login:*** The client queries the server with the username and password and the server verifies and responds with a cookie that must be used in all further communication.
3. ***Search users:*** The client queries the server with a string that closely matches the requested username and the server responds with a list of users that match this substring.
4. ***New album:*** The client queries the server with the new album name and it's secret and the server register's it.
5. ***Add user to album:*** The client queries the server with one of it's albums, the user to add and his secret and the server registers it.
6. ***List a user's album:*** The client can always fetch the information about his albums and their slices.
7. ***Set a slice url:*** When a user detects it belongs to a new album it must set his slice url, so he sends the url and the server registers it.

4. Wireless P2P Architecture

4.1 Data Structures Maintained by the Mobile Nodes

Each mobile node contains a list of the other available peers and their respective location. Apart from that, each node also has a representation of the metadata related to each album and its members. Internally, in the mobile app data, we also store the collected photos.

4.2 Description of Messaging Protocols between Mobile Nodes

Overall, all messages follow the Json format. Each message that is sent over the socket contains two parts:

- **The header:** 24 bytes and is made of
  - [content size: 4 bytes] : Integer that represents the size in bytes of the content of the message.
  - [message type: 4 bytes] : Integer that represents the type of the message (json or photo).
  - [arg1: 8 bytes] and [arg2 : 8 bytes] : General usage arguments to pass additional arguments for the protocol implementation.
- **The Content:** The content depends on the type of the message. If the content is a json file then we only need to deserialize the received bytes into a readable json object that itself then implements an application layer protocol. If the content is a photo, the first N bytes represent the photo name and the rest of the bytes are the photo. The header arg1 is also set with the size in bytes of the photo name.

5. Advanced Features

5.1 Security

Secrets: Cloud-backed security works as follows:

1. Every user generates a Key-Pair of RSA keys. And stores the public key in the server.
2. When a user adds an Album it encrypts its slice with an AES key that it randomly generates which will then be encrypted with his own RSA public key and stored in the server (we call the encrypted key: secret).
3. When a user wants to add another user to an album he must provide that new user's secret, by obtaining the other user's Public Key and encrypting the original AES key of the Album with it.

Therefore, every user in the album has the secret and they can access it to obtain the key to the album slices, by decrypting their instance of the secret with their Private Key. By doing this we ensure that even though the **p2server** has access to the slices it won't be able access the photos because it never has direct access to the album key which is used to decrypt it.

5.2 Availability

(Note: The availability protocol works under the assumption that each node has infinite local storage.)

Whenever an user adds a photo to an album, he broadcasts this photo to every peer that is available and belongs to that album. From time to time, each node will synchronize with its peers. On this phase, every node will ask the others for any missing photos from their slice. Additionally they will also ask for missing photos from other users' slices that currently are not connected.

As an example let's consider two peers, A and B, that both belong to a certain album that contains the photos [slice A: [4], slice B: [1,2,3], slice C: [5]]. A and B are online and C is down.

A will tell B that for his slice he has photos 1 and 2. B will receive this message and will see that he has overall uploaded photos 1 ,2 and 3, therefore he will send to A the missing photo 3. B will also ask A for any missing photo that B might have from his slice. A and B will see that C also belongs to the album and will ask each other if any has photos from C from a previous synchronization when he was online.

6. Implementation

The P2Photo system has two components : **p2server** and the **p2photo** application (for Android devices).

6.1 p2server

The server was implemented using a framework named Flask programmed in Python. It allowed us to build a simple and easy to manage RESTful web service. The data format used in the communication between the **p2server** and the **p2photo** application is JSON.

6.2 p2photo

The mobile application was developed for Android, programmed in Java and tested with APIs over level 25. Is also split into two different application modes (**Cloud-based** and **Wifi-Direct**) that mostly differ on where the photos are stored and also on how they are shared. In practice both application modes extend a common abstract class so as to have modularity in the system architecture.

6.2.1 Cloud-based

The choice of cloud provider for this mode was **Dropbox** and the reasoning is made based on the easiness of the initial configuration for developers.

External Libraries: A java library for the **Dropbox Core** API (dropbox-sdk-java).

Android Components:

When different activities need to access globally stored information, most of the times, the same instance of a SharedPreferences object is used. There are other times where it is necessary for different activities to share information in a more direct and isolated way, and that is through the startActivityForResult() method call.
In order to retrieve/send information from/to the **p2server** HTTP requests are created and executed as background tasks. To communicate with **Dropbox** the dropbox-sdk-java library is used.

The most relevant Activity (apart from the Mode/Register/Login Activities) is the DropboxGalleryActivity which combined with a DropboxProvider (the class responsible for requesting and processing all the necessary data such as the photos and slices) is able to display all the albums and their photos to the user.

Persistent state maintained:
Previously downloaded photos are cached (stored locally in the device's persistent memory) so as to avoid wasting bandwidth resources.
All the keys generated in a device are also stored in that device. What this means is : the secret keys generated when an album is created and the key pair that is generated when a user registers on the system are **only** stored on the device in which these actions were performed.

6.2.2 Wifi-Direct

Peer discovery
Initially, every user has to connect to the current group owner (if there isn't a group owner, it is generated automatically upon the first connection). After this initial connection is established, each node starts a local server to accept incoming messages from other peers. When this server is set up, the peer will register itself on the group

owner. From time to time, each node will ask the group coordinator for available peers in the network and their respective address.

Monitors and timed jobs
The nodes have different monitors and timed jobs depending on the role. Each of these run on a separate thread:

- Group coordinator node:
    - Liveness monitor: Checks for dead and alive nodes in the network.
- Other nodes:
    - Heartbeat: From time to time tells the group coordinator that he is still available.
- For both:
    - Synchronization: This job is responsible for synchronizing the photos with other users automatically.
    - Photos queue processing: This thread is responsible for processing a queue that has photos to send to other users.

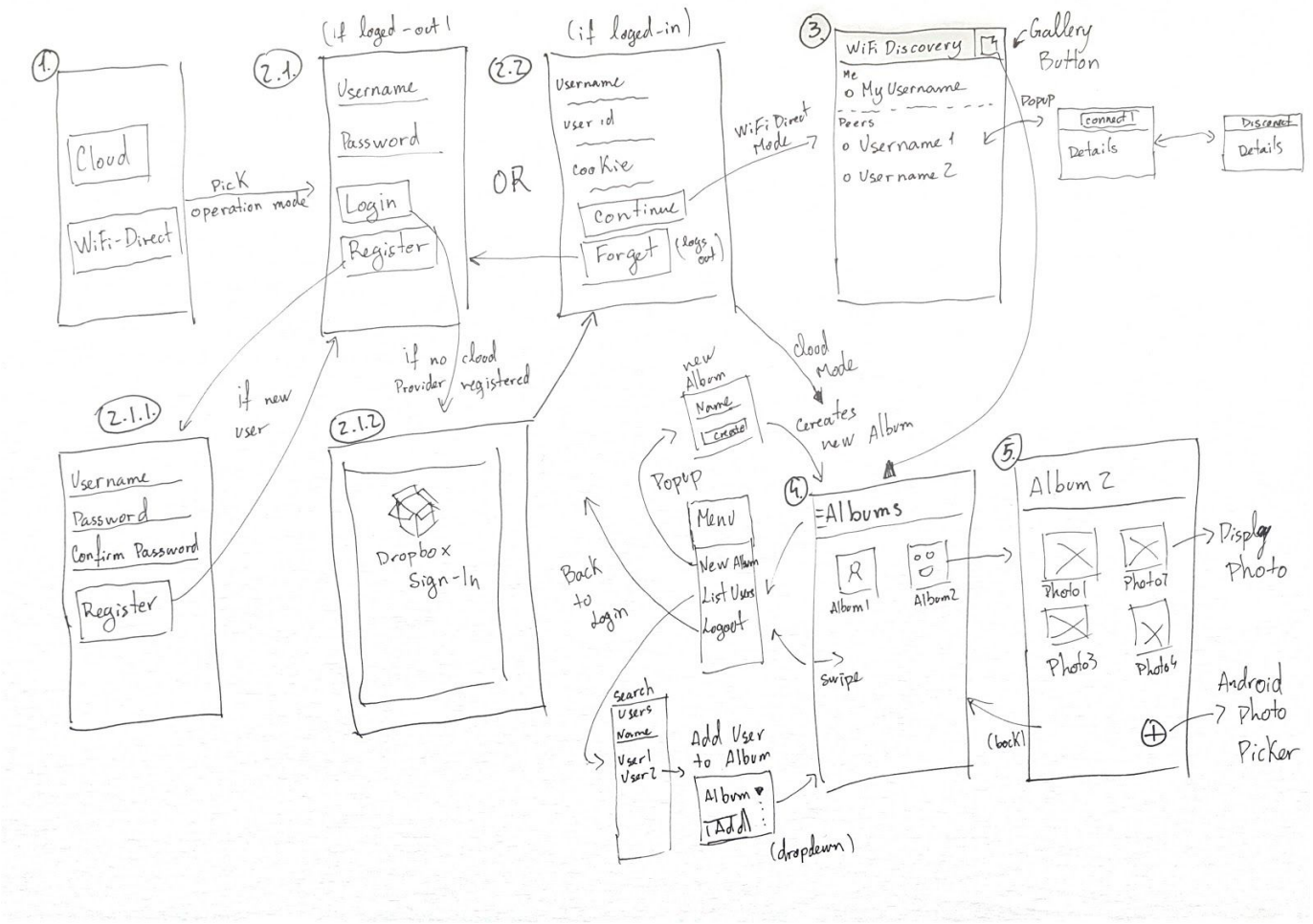
Android components:
Like the other module, the wifi direct module also needs to access shared information across the application like the user id and his cookie. All the HTTP requests are created and executed as background tasks.
The most important Activity in this module is the WifiDirectActivity. The module responsible for all the logic of how the peers behave is the module WifiDirectConnectionManager. Additionally we also have a socket manager responsible for implementing the communication protocol, a client socket manager and a server socket manager that create threads to handle the connection details.

7. Limitations

- Since private keys are kept locally and not transmitted to a secure place online, an account becomes locked to the phone that creates it, otherwise it will be unable to decrypt the secret and won't be able to see the album slices.
- WiFi-Direct discovery is unstable and sometimes requires a WiFi restart/App restart.
- In WiFi-direct, due to a time shortage we discarded the photo passing protocol and instead of sending it as aforementioned, we convert the bytes of the photo to a base 64 string and send it over a json object to the other node, effectively making every photo 3.5 times larger than it actually is. This has some impact on the performance of the app.

8. Conclusions

Overall, this version of the P2Photo system has all the core functionalities although there are many optimizations that could be implemented in order to reduce the impact of some of the limitations mentioned in the previous section. For instance: to solve the "account locking" issue, one possible solution is to have a trusted third-party in charge of performing the key sharing. Also, the interface of the application could be improved both aesthetically and functionally, but since these were not the relevant topics to be evaluated in this project, not much time was allocated to improving those areas.
Regarding the project in itself, the WiFi-Direct module was troublesome to implement as compatibility issues between devices would arise occasionally. To mitigate this issue, the groups were advised to use Termite, but that is also a problem in itself as running several VM's of Android devices on the same machine combined with the resource requirements of Android Studio, would severely impact the performance of the machine, sometimes even crashing the whole system. Therefore we decided not to use Termite to develop the WiFi-Direct module and we believe it should be something worth a discussion for future editions of the course.


1.
Cloud
Wi-Fi-Direct
Pick operation mode
2.1.
(if loged-out)
Username
Password
Login
Register
OR
2.2
(if loged-in)
Username
user id
cookie
continue
Forget
(logs out)
Wi-Fi Direct Mode
3.
Wi-Fi Discovery
Gallery Button
Me
My Username
Peers
Username 1
Username 2
Popup
connect!
Details
Disconnect
Details
2.1.1.
if new user
Username
Password
Confirm Password
Register
2.1.2
if no cloud Provider registered
Dropbox Sign-In
Cloud Mode
new Album
Name
create!
creates new Album
Popup
Menu
New Album
List Users
Logout
Back to login
4.
Albums
Album1
Album2
swipe
5.
Album 2
Photo1
Photo2
Photo3
Photo4
Display Photo
back!
Android Photo Picker
search
Users
Name
User1
User2
Add User to Album
Album
Add!
(dropdown)