Project Report

Ubiquitous and Mobile Computing - 2018/19

Course: MEIC / METI

Campus: Tagus

Group: 8

Name: David Gonçalves	Number: 73891	E-mail: david.s.goncalves@tecnico.ulisboa.pt
Name: João Pires	Number: 76412	E-mail: joao.silva.pires@tecnico.ulisboa.pt
Name: Leonardo Troeira	Number: 94104	E-mail: leonardo.troeira@tecnico.ulisboa.pt

(PAGE LIMIT: 5 pages – excluding the cover)

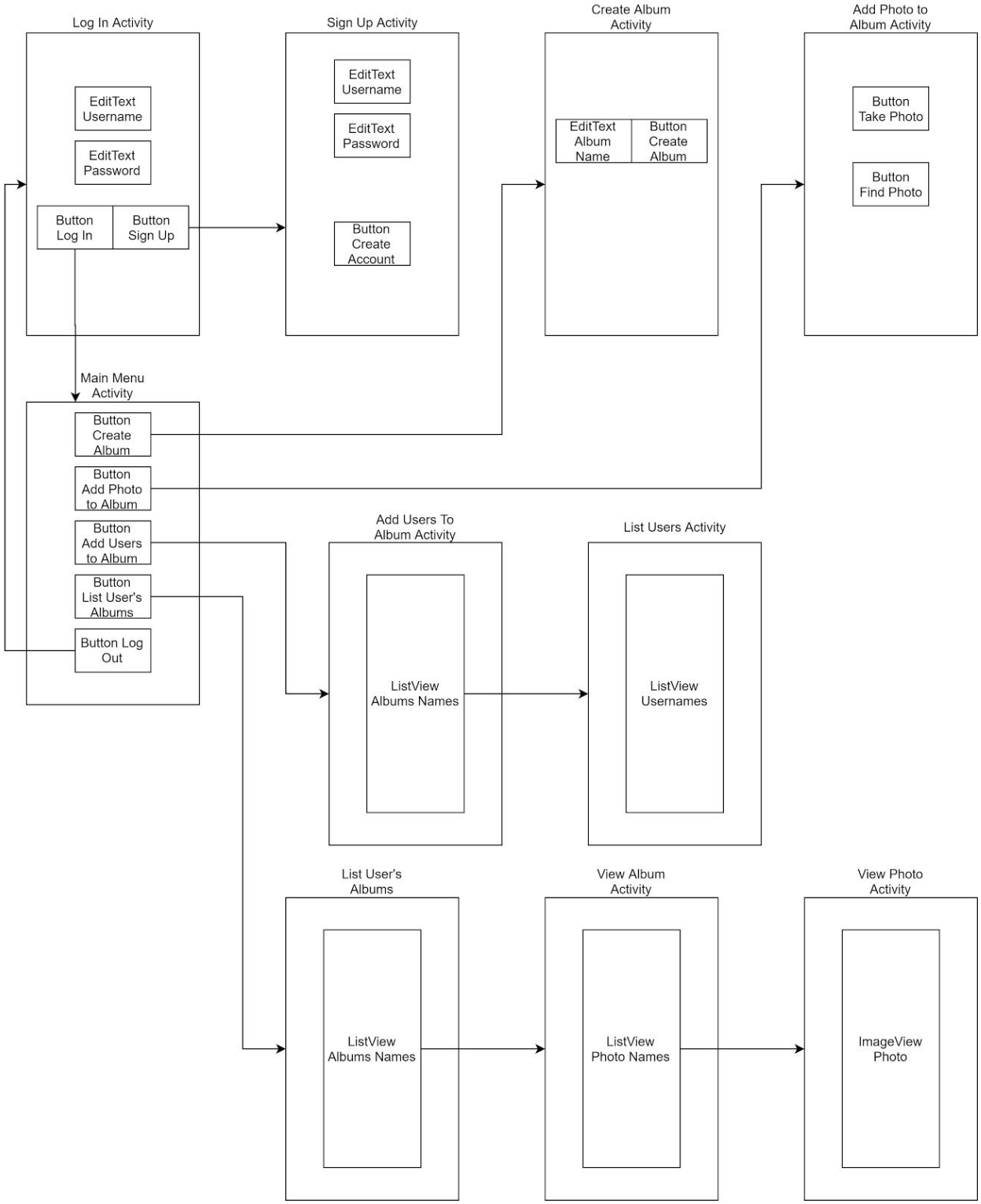
1. Achievements

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

Partially Implemented

In the find users and add users to albums features on the Wireless Mode we are only able to list the devices around the user, but
we are not able to add those users to the album.

2. Mobile Interface Design



On startup our app starts on Login Activity were the user is presented with two options, either enter their username and password and press the login button, taking him to the Main Menu Activity, or the user might push the sign up button, which direct the user to the Sign Up Activity were the user can create a new account.

The Main Menu Activity lists the options available to the user of the application, in our main menu we have five buttons, which will lead the user to the following operations. The users can create albums in Create Album Activity, where they input the name of the album that they intend to create. We have the Add Photo to Album Activity that allows the user to add photos to the albums. There is the List User's Albums that shows the albums that the users are a member from and when the user selects an album, it will progress to the View Album Activity that will list the photos that are inside the album, and when a user press the name of a photo it will transition to the View Photo Activity were the photo will be displayed. In the Add Users to Album Activity the user can select an album that wants to share with another user, now the List Users Activity will open and the user can now select the user to share the album.

3. Cloud-backed Architecture

In our project we use the Google Drive API. The Drive API makes use of 2 Classes, DriveResourceClient and DriveCLient. DriveClient handles high-level functions like sync, DriveResourceClient handles access to Drive resources/files (e.g. create folders (albums), create files (indexFiles and images)). We also had the need of using the REST API, because we couldn't perform some operations with the old one (e.g. set folders to public). So we get the Authenticated Client Account and set the Drive service of REST API to that authenticated account. And at this point in our project we can make use of both APIs, the deprecated one and the REST API.

3.1 Data Structures Maintained by Server and Client

In the DriveConnection class we have 3 structures, PhotoAlbum, PhotoImage and IndexAlbum. Each one of this struct keeps track of DriveId, Name of file and in the PhotoAlbum an ArrayList of PhotoImages, tracking the images of that album. Every time the user creates an Album, it will be added to an ArrayList<PhotoAlbum> and his consequent indexFile will also be added to ArrayList<IndexAlbum>. Also when the user adds a Photo to an album, the photo will be added to the ArrayList<PhotoImages> of that specific album.

In the Server side we store the album metadata (name, indexes URLs and AES key) in an SQLite database.

3.2 Description of Client-Server Protocols

4. Wireless P2P Architecture

4.1 Data Structures Maintained by the Mobile Nodes

In the mobile nodes to store the albums and photos, there is an index file that stores the names of albums that belong to the user, and for each album there is an album index file that stores the names of the photos that are saved in this album.

4.2 Description of Messaging Protocols between Mobile Nodes

Our plan was to implement the message between mobile nodes using TCP sockets, were users would use the sockets to allow users to join the membership group of the albums and receive updates to these albums.

We were able to implement the initial part of the communication mechanism, where the users find each other using the WIFI Direct technology to establish the TCP connection.

5. Advanced Features

5.1 Security

In order to prevent a malicious application provider from retrieving user photos from the catalog files in private cloud stores we used AES in CBC mode with PKCS5Padding and a random IV. We chose CBC over ECB due to the problem with patterns in ECB over large amounts of data, and we have no need for parallelization that CTR would provide us. Moreover, we chose block cipher over stream cipher because we know the site of the data beforehand. Each album has a different key which is randomly generated when creating it, because the confidentiality of an album depends on its key being unique and unknown to others. This key is stored in the server, encrypted with each user's — the ones who have access to the album — public keys.

Therefore, each user has a pair of asymmetric RSA keys that are created at sign up, a public key used to encrypt the album's AES key, and a private key used to decrypt it. Because storing this private key in the user's mobile is not practical (the user may uninstall the app, clear app data, switch mobile…), we store both keys in the server; however, the private key is sent and stored encrypted (for obvious reasons) using AES with an SHA-256 hash of the user's password as key. However, since the user's password itself is sent and stored in the server, we've decided to send it has an SHA-512 hash (to be different than the SHA-256 hash), this way the server won't figure out the key to decrypt the user's private key.

Moreover, if an attacker would get access to the server's database, he wouldn't be able to fetch the user's passwords (in this case the SHA-512 of it) as they are stored encrypted using PBKDF2 with a random salt, which is stronger than just using a hash and a salt.

Because these keys and encrypted url's are stored as garbled bytes that are not convenient to send through TCP sockets (between the client and server) as the null byte would be interpreted as the end of the stream, we perform a base64 encoding of them before sending.

6. Implementation

Describe any relevant implementation choices, for example:

- *Additional external libraries used in the project*
- *Android components (e.g., services, activities); how they communicate with each other; how they share global state*
- *Threads that exist in the mobile application and their function*
- *How sockets and communications are handled*
- *Persistent state maintained on the mobile device*

We could have chosen any programming language and platform to host our server, but since we were already working with Java and Gradle in Android Studio, we decided to go with these. In addition we used SQLite for storing data persistently, thread safe and with fault tolerance.

In the client we have a class called GlobalClass to maintain the application state, it contains a ServerConnection object to interact with the server, a DriveConnection object to interact with the cloud, the user's asymmetric keys and the storage mode.

The communication is done through TCP sockets. The client has a class called ServerConnection which contains a set of methods to interact with the server.

7. Limitations

List the currently know limitations of your project (e.g., bugs, restrictions in the test environment, etc.)

Our application was not intensively tested, therefore unexpected bugs may occur, some bugs which we've noticed are:

1. Sometimes the "Find photo" button in the Add Photo To Album does not launch the screen to pick a photo, after canceling or failing to add a photo.
2. When the connection to the server is lost, we should destroy all activities and launch the login activity, we did not do this.
3. In the Wireless P2P architecture the system can only use mobile phones as a test environment, the Android Studio emulator is not supported.

8. Conclusions

This project was very interesting and educational, using a wide range of technologies that are useful in realistic projects. It tackled:

- Android programming, which is highly popular at the moment of this writing, and somewhat distinct from the conventional PC programming.
- Distributed systems, using cloud services, peer-to-peer, and a centralized server.
- Security, using asymmetric cryptography, symmetric cryptography, and cryptographic hash functions.
- Replication, in order to assure availability.

However, the downside we experienced was that the workload was somewhat excessive — considering that we had to work on more projects from other courses at the same time — with perhaps too many features to implement, more than required for the scope of this course. For instance, the security module was not taught to us and could be dispensed. The distributed system modules were also more than necessary (either use cloud or peer-to-peer).