Project Report

Ubiquitous and Mobile Computing - 2018/19

Course: MEIC

Campus: Tagus

Group: 22

Name: André Fonseca Number: 84698 E-mail: andre.filipe.alves.fonseca@tecnico.ulisboa.pt

Name: Leonor Loureiro Number: 84736 E-mail: sebastiao.amaro@tecnico.ulisboa.pt

Name: Sebastião Amaro Number: 84767 E-mail: leonor.filipa@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	Fully
	Add users to albums	Fully
	List user's albums	Fully
	View album	Fully
Advanced	Security	Fully
	Availability	Fully

Optimizations

Store photos from cloud albums in cache, and load them when opening the album. The *list album photos* functionality, in the cloud mode, is the application's functionality that presents the highest latency, as it performs multiple server requests, cryptographic operations and downloads before displaying the photos. To reduce this latency, we store the album photos in the application's cache when they are downloaded.When the album is opened again, the photos in cache are loaded and displayed without having to be downloaded again.

2. Mobile Interface Design

The application has the following screens:

- **Choose Mode -** allows a user to choose the application mode: Google (Cloud-Backed) or P2P. Once the user selects a mode, it is redirected to the *Login* screen.
- **Login -** allows a user to login in the application and contains a link that redirects the user to the *Register* screen. If the login is successful the user is redirected to *List Albums* screen.
- **Register -** allows a user to register and contains a link that redirects the user to the *Login* screen. If the register is successful the user is asked to login his google account and give permissions to the application, after which he is redirected to the *List Albums* screen.
- **List Albums -** shows the list of the logged in user albums. If the user selects an album in the list, is redirected to the *List Photos* screen.
    - Toolbar: has a menu option that allows a user to perform logout, another that syncs the album's list with the server and another that redirects the user to the *Settings* screen.
- **List Photos -** displays the album photos and has a button that allows the user to share the album (redirects to the *Add Users* screen) nand another that allows the user to add a photo to the album (redirects to the *android's gallery*)
    - Toolbar: has a menu option that allows a user to perform logout, another that syncs the album's metadata with the server and downloads new photos, and another that redirects the user to the *List Albums* screen.
- **Add Users -** list all application user, and allows the user to select another user from the list with which to share the album.
- **Settings -** contains three fragments: one that displays the server log, another that displays the application log and another that allows the user to define the maximum cache size.

3. Cloud-backed Architecture

3.1 Data Structures Maintained by Server and Client

Server Data Objects:

- **Album** - represents a photo album, containing the album's members, photos download urls, file id of the catalog file in each user's google drive, and the album's secret key encrypted with each user's public key.
- **User** - represents a user, contains of his albums, username, password and public key.
- **Operation** - information about operations performed, such as timestamp, operation type, user IDs and album's name.

Server Persistent Data Structures:

- A **List** of **Operations** - a log that contains all relevant operations performed by the server.
- A **List** of **Users**, containing all application user's and their albums, representing the system's current state.

Client Persistent Data Structure Objects:

- User's asymmetric **key pair**, stored in the application's android keystore, that allows the user to encrypt/decrypt the album's secret keys.
- **Cached** photos, with are stored in the application's cache directory, and limit cache size, which is stored in the device's preferences.

3.2 Description of Client-Server Protocols

The Client-Server communication is made through http requests, from client to restful API on the server.
For the following operations, take into account:

- **mode** represents if the application is running in P2P mode or Cloud-Backed mode.
- **token** represents a login token generated and signed by the server, which allows a user to maintain his session active for a period of time, in a secure way, without needing to reintroduce his password.
- **url** the user's catalog encrypted url.
- **fileID** represents the file id of the user's catalog file in the google drive.

- The server may return appropriate error messages that are not represented here.

The server provides the following operations:

Register & Login
- Client sends: username, password (and his public key, in the register operation)
- Server Responds with login token or error message

Get Group Membership:
- Client sends: login token, username, album name, mode (= cloud)
- Server responds with all members and their encrypted links

Get Secret Key
- Client sends: token, username and the album name
- Server return the album's secret key encrypted with the user's public key

Get Users
- Client sends: token and username
- Server returns usernames and public keys of the registered users

Get User Albums
- Client sends: token, username and mode
- Server replies with the name of all albuns belonging to the user

Share Album
- Client sends: token, user1's name, user2's name, album's name, key , mode (= cloud)
  - the key is the album's secret key encrypted with the user2's public key
- Server responds: Success, if it was successful

Create Album
- Client sends: token, username, album name, url, file ID, encrypted secret key, mode
- Server responds: Success, if it was successful

Update Album
- Client sends: token, username, album name, url, fileID, mode
- Objective: updates the information of an album, used after an album being shared with a new user, this new user must add his catalog and other information to the album
- Server responds: Success if it was successful

Get Operations Log
- Client sends request
- Server responds with a list of all operations executed by the server.

3.3 Other Relevant Design Features

- The way persistent data is stored and updated in the server, guarantees operation atomicity, preventing the system's from being corrupted if the server crashes during an update.
- The user's passwords are hashed with salt, using Bcrypt algorithm, before being stored. The salt ensures that equals passwords generate different hashes, mitigating password attacks, such as rainbow tables.
- The server generates a session token that is valid for a period of time, allowing authentication of request without sending the password, therefore limiting password exposure in the network.
- The application caches downloaded album photos, and displays cached photos when opening an album, therefore reducing the latency of this functionality (this feature is more detailed in the achievements - optimizations section).

4. Wireless P2P Architecture

4.1 Data Structures Maintained by the Mobile Nodes

Application Data Structures:

- **List** of **Users** in Group - containing the list of users currently in the same Wifi Direct Group as the logged in user.

Application Persistent Data Structures:

- **Catalog files**, which are stored in the device's internal memory, and that contain the filenames of the album photos
- **Photo's files**, which are also stored in the device's internal memory, and that contain the compressed bitmaps of the album photos added by the user.

- **Cache files**, which are stored in the application's cache memory and that contain the compressed bitmaps of the album photos added by other album members.

4.2 Description of Messaging Protocols between Mobile Nodes

- When broadcast receiver registered in the application detects a membership change in the Wifi Direct Group, the application contacts the new devices, asking them their username, and adding them to the list of user in his group.
- Upon opening an album, the application checks if any of album members is in the Wifi Direct Group. If they are, the application requests their photos, to which the they will respond with the compressed photos from their album catalog. Once received, the photos are decompressed and displayed.

4.3 Other Relevant Design Features

The server has the same features as those described in the Cloud-Backed architecture.

5. Advanced Features

5.1 Security

- Upon registration, the user's generate an asymmetrical key pair, storing the private key in the application's keystore, and upload their public key to the server.
- The catalog's urls are encrypted with a secret key, generated when the album is created. This secret key is encrypted with the public key of each album member before being stored in the server.
- To download the album photos, the user has to request the album's secret key, decrypt it using his private key, and then use it to decrypt the catalog's urls before downloading them.
- To share the album, the user has to obtain the other user's public key, use it to encrypt the album's secret key and upload the encrypted key to the server.
- Since the album catalog's urls are encrypted, only the album member's, who can decrypt the secret key and therefore the url's, are able to download the catalogs and, consequently, the album photos.
- The most correct approach, relies on a trusted third party to distribute the user's public keys. However, for simplification purposes, we trust the server to provide the correct public keys.

5.2 Availability

- When a user downloads the album photos from another album member, these photos are store in the application's cache memory, identified by album they belong to and the user that added them.
- When opening an album, if an album member is not reachable, the album's cached photos are loaded.
- The cache's used space is checked when a photo is cached, to ensure it does not surpass the limit size. If it reaches the limit, then cached photos are deleted, oldest first, from cache until the used cached space reached a third of the limit size.

6. Implementation

Sever:

Server was implement using Java 8, with Spring Boot framework to develop a REST API.

External libraries:

- Spring Boot framework (REST)
- Bcrypt (Password hashing)
- Jsonwebtoken (login tokens)

Application:

The application was developed using Android SDK. The application was tested using the AVD Manager integrated with the Android Studio.

External libraries:

- Google Play Services API - perform google sign in and give drive permissions to the application

- Google Drive V3 REST API - to read/write files on the google drive
- Termite API - P2P Wifi Direct operations
- Loopj Async HTTP Client Library - used to perform HTTP requests

Android Components:

- Broadcast Receiver that listens for changes to the System's Wifi P2P state
- Each screen mentioned in the Mobile Interface Design section is implemented in a different activity.
    - Each setting option mentioned is implemented in a different fragment.

Application Handlers/Managers:

- **Wifi Manager** that handles the changes detected by the broadcast receiver
- **Google Sign In Manager** that handles the google sign in operations and creates the credential that allows the application to access the user's associated google drive files
- **Google Drive Handler** that handles the read / write operations in the google drive and that handles the catalogs and photos downloads
- **Security Manager** that implements all cryptographic operations
- **File Manager** that handles the read / write operations in the device's internal memory
- **Cache Manager** that handles the read / write operations in the application's cache memory
- **Server API** that handles all server requests

The operations executed by these managers are performed in asynchronous tasks.

Global State:

All activities share a global state through a *GlobalVariables* class that extends the android *Application* class. It contains:

- User - logged in user info
- Token - current server session token
- Operations log - contains a log of all relevant operations performed by the application
- Mode - the current application mode (Cloud-Backed or P2P)
- Handlers/Managers relevant for the current mode
- Map between album names and corresponding file IDs

Communication:

- Communication between peers is performed through sockets.
    - When a user logs in in the P2P mode, a server socket is opened to receive request from other users. When it receives a request, opens a client socket to send the response.
    - The server socket is closed on logout.
- Communication between the application and the server is made through HTTP request to the server's REST API
- Communication between the application and Google Drive are made using the Google Drive REST API V3.

7. Limitations

- The security feature is only available for devices with API 23 or above. Since the Cloud Mode requires this feature, therefore it has the same API restrictions.
- The mode P2P is only available for devices with API 24 or below because of Termite compatibility

8. Conclusions

With this project, we developed a full stack application that works both in cloud mode and P2P mode. We worked with multiple new technologies and gained insight on mobile application development. However, we found the project to be too extensive, and working with the Termite API was very complex due to the limited documentation available. Considering the project's load, we believe it should represent a higher percentage of the final grade.

Anex

Wireframe Diagram