Project Report

Ubiquitous and Mobile Computing - 2016/17

Course: MEIC

Campus: Tagus

Group: 7

Name: Frederico Manuel de Oliveira Teixeira Number: 77928 E-mail: frederico.teixeira@tecnico.ulisboa.pt

Name: Constantin Zavgorodnii Number: 78030 E-mail: constantin.zavgorodnii@tecnico.ulisboa.pt

Name: Henrique Fernandes Alves Number: 87891 E-mail: henriquefalves@tecnico.ulisboa.pt

1. Achievements

Version	**Feature**	**Fully / Partially / Not implemented?**
Baseline	Sign up	Fully
	Log in/out	Fully
	List / create / remove locations:	Fully
	Post messages	Fully
	Unpost message	Partially
	Read message	Fully
	Edit user profile	Fully
	Support for different policies	Fully
	Centralized	Fully
	Decentralized (direct) message delivery	Fully
Advanced	Security	Fully
	Relay routing	Partillay implemented

2. Mobile Interface Design

The wireframe was carefully projected using *NinjaMock,* the implemented GUI can be observed in the Appendix Fig.1, the following paragraph describe succinctly the main pages.

Initially the user is prompted with the LogIn page where he needs to insert username and password. If the user desires to create a new account he can follow a hyperlink that leads to the SignUp page. Both successful log in or sign up leads to the main page where user can browse through nearby, personal and cached messages.with the help of a drawer the user has direct access to main functionalities identified:

- Messages - three fragments present a list of nearby posts given the user location, submitted messages and cached messages. This view present an option to add a new post and view details of a specific message;
- Locations - list of locations, see their details and add new locations.
- Profile - displays information about the user(username, email), key pair preferences and the ability the manage them.
- Settings - Enable and disable wifi-direct, define number of relay messages supported, frequently visited locations.

3. Baseline Architecture

3.1 Data Structures Maintained by Server and Client

Server

The Server has a Relational Database(MySQL) to store User Profiles, Posts, Interests and Locations in the Appendix Fig.2 database relationships diagram. The random authorization key is stored in memory.

Client3

- SharedPreferences: For information related with the login state this lowers the burden of the user to insert log in credentials every time the application is closed. Variables like: token, username, email, user interests(Key Pair Values),
- SQLITE: Storing Locations instances(GPS and WIFI), all interest keys the user can use.
- File: Posts are stored in 6 different files: (Nearby Posts, Posted Posts, Nearby Posts) for both Centralized and Decentralized.

All structures are initialized at the log in and erased on the log out.

3.2 Description of Client-Server Protocols

All communication are done via HTTPS and are only supported if the user has Internet connection.

Login - ***UserLoginTask(AsyncTask)***

The user sends username and password and receives his email and a token to be used on the following requests to the server.

Fetch UserData - ***FetchUserInterestsTask(AsyncTask)***

Done only once to get key pair preferences.

Fetch Interests Global List - ***FetchGlobalInterestsTask(AsyncTask)***

Done only when the background service is created.This retrieves all Key values introduced by all users. User sends a timestamp of last fetch time when getting the interests to minimize the amount of data downloaded.

Refreshing Token - ***RefreshTokenTask(AsyncTask)***

Whenever the user passes through the Login Activity it fetches a new Token which has a validity of 7 days, this enables the user to have access to the server database.

Every 5 seconds a Thread is launched by the *NetworkThread*:

Fetch Locations - ***LocationsRequester(Thread)***

User sends timestamp, the server sends list of locations inserted after timestamp.

Fetch Centralized Posts - ***CentralizedPostRequester (Thread)***

Sends client's GPS coordinates and list of SSID he sees, but only if he intersects new Location instances, sender does not send posts from the requester.

Post *CreatePostTask(AsyncTask)*

Sends all information according to the specified Post immediately after user creates it on the mobile phone.

Unpost *DeletePostTask(AsyncTask)*

Sends id of the Post to be deleted.

3.3 Description of P2P Protocol for Decentralized Message Delivery

We support two features, done by a background Thread *DecentralizedPostSender.* In order reduce connections in the WiFi-Direct we assume the nodes representing other users start with *DEV_,* access points with *WIFI_* and beacons with *BLE_*.

Post *SendPostTask*(*AsyncTask*):

- Sender verifies if new post in Decentralized mode was inserted by the user.
- Sender verifies if the user changed Location.
- Sender keeps list of <Devices, Delivered Messages>.
- Sender sends properties of the message(interests keypairs, location, policies).
- a. Receiver respects post properties - receives messages.

  b. Receiver do not respects post properties - message not sent.

6. a. Sender removes the history of sent posts for devices that have abandoned the network

   b.Receiver remove the nearby posts of devices that have abandoned the network

Unpost *UnpostTask(AsyncTask)*:

1. Sender verifies if user is on the post location.
2. Sender verifies if reachable devices are on the list of who received the message.
3. Sender sends entire message to the device.
4. Receiver only deletes if did not save the message.

4. Advanced Features

4.1 Security

To protect the against threats like information disclosure, tampering with data, Spoofing identity and Repudiation we implemented the following solution:

The server was deployed on the Google Cloud Platform which enables us to have valid digital certificate to prove the identity of the server and establish a TLS connection between the clients and servers, hence **privacy**, **integrity** and **server authentication** certified. On top of it the we defined a lookalike OAuth 2.0 authorization framework. The user authenticates sending username and password, the server verifies the credentials and if correct an authorization code(aka access token) is produced and used on every requests by the user.

4.2 Relay Routing

This feature was not carefully tested but the backbone of it was built and discussed. Each user specifies number of messages it can transport and frequently visited location(*FVL*) to maximize the probability of a message arriving at the destination. Sender indicates the message location and if it is contained on the *mule's FVL* it will be sent. Once the mule is on the location and it manages to pass the message to one node that fulfills the post policies, it will remove the post from mule list. This process is done by Thread *DecentralizedPostSender*.

5. Implementation

Programming language and platform used to implement the server

On the Server we used Django, python framework to build rest API with Response formats JSON. Base URL:https://cmu-locmess.appspot.com/api/

Endpoint	Allowed Methods	Authorization
login/	POST	All
registration/	POST	All
logout/	POST	Authenticated Users
interests/	GET	Authenticated Users
users/{username}/	GET, PUT	Authenticated Users
posts/	GET, PUT	Authenticated Users
posts/{post_id}	GET, DELETE	Authenticated Users
locations/	GET, POST	Authenticated Users
locations/{location_id}	GET, DELETE	Authenticated Users

Android components (e.g., services, activities); how they communicate with each other; how they share global state

Relevant **activities** related to the user interaction: *LoginActivity, RegisterActivity, LocationActivity, LocationCreationActivity, LocationDetailActivity, PostActivity, PostCreationActivity, PostDetailActivity, Profile Activity.*

Two **services** called *LocMessBackground* which is responsible for managing GPS location, Interests list, Location list, Posts list. The *WiFiP2pService* is responsible for handling WiFi Direct events.
To share global state we use:

- Application context to share manager's information.
- Shared Preferences to keep service state. Although it is implemented with the flag

START_STICKY which enables to keep running in background, the Operating System can at anytime end it.

How sockets and communications are handled

IncommingCommTask opens the socket in a specific port and waits for connections from other devices. After establishing a connections the *AttendClientTask*(AsyncTask) is launched to deal with incoming messages(accepting or rejecting). The tasks related with the sender were previously mentioned on the P2P Protocol.

How GPS events are handled

There are two option when getting the user's location: Android's Location API and Google's Location Services API. We chose the second one since Google declared it provides better battery performance, as well as more appropriate accuracy.

The main method is OnLocationChanged which is triggered when the minimum time or the minimum distance between the current and last location has changed according to the thresholds defined. When triggered, we verify if the Location instances the user reaches changed in order to synchronize nearby posts list and if he needs to send messages to nearby devices.

6. Limitations

Removed locations on the server are only known by the user when he tries to use a specific Location when creating a Message. The synchronization of the list could have been done when fetching the new Locations.

Fetching Posts are only made if the user change his Location instances which poses a problem in the given example: user X remains in Location *A* and a new post his posted by user Y, this new post will not be fetched by user X in Centralized Mode.

Posts deleted are not propagated to the receiver(Centralized MODE)

Posts that reached expired window are not removed.
User only can delete new locations if there are no posts associated

7. Conclusions

It was a excellent opportunity to work on Android development and keeping in account Context-Awareness, Location, Security and Design of the mobile and ubiquitous device.

8. Appendix

Fig. 1 - Wireframe

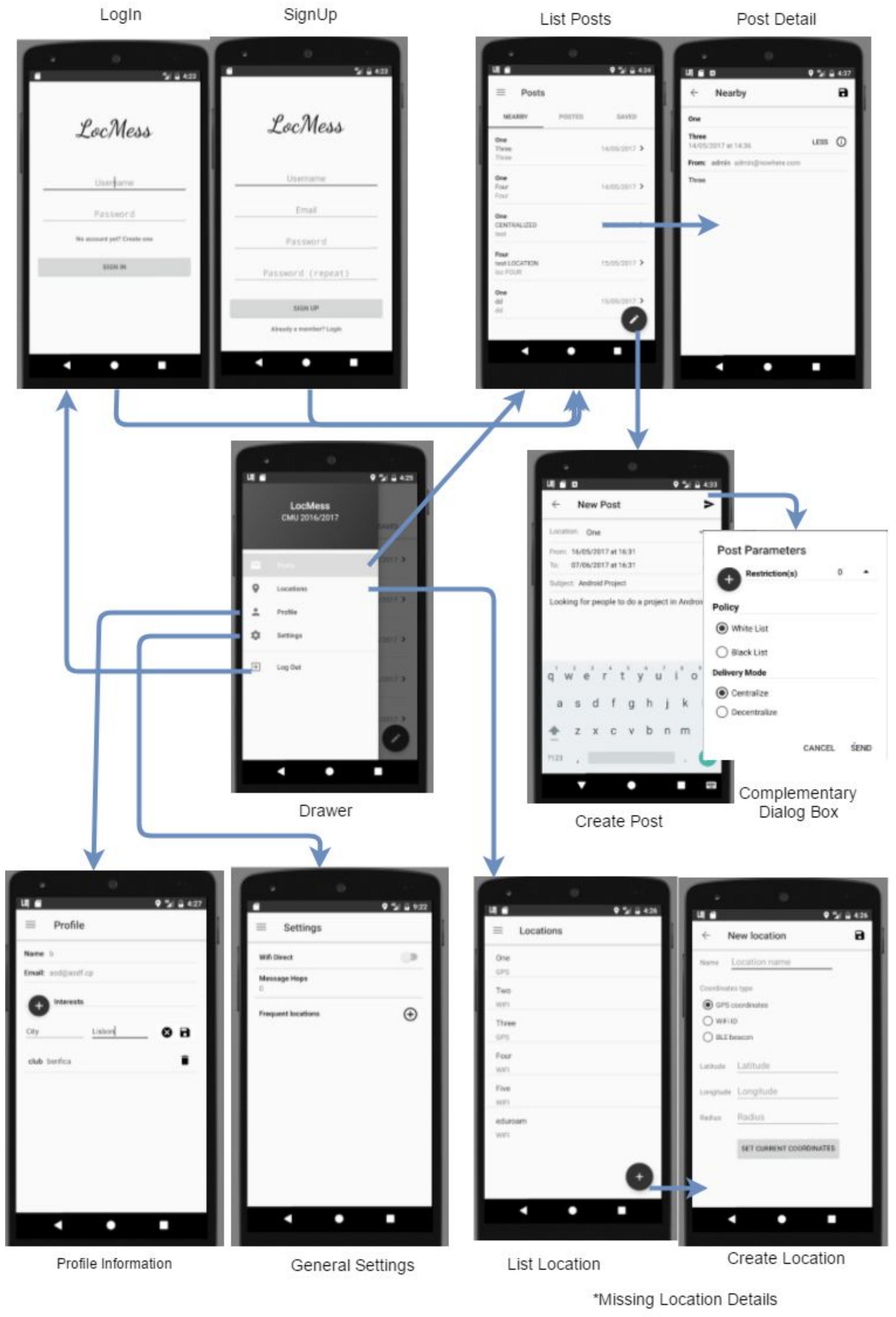
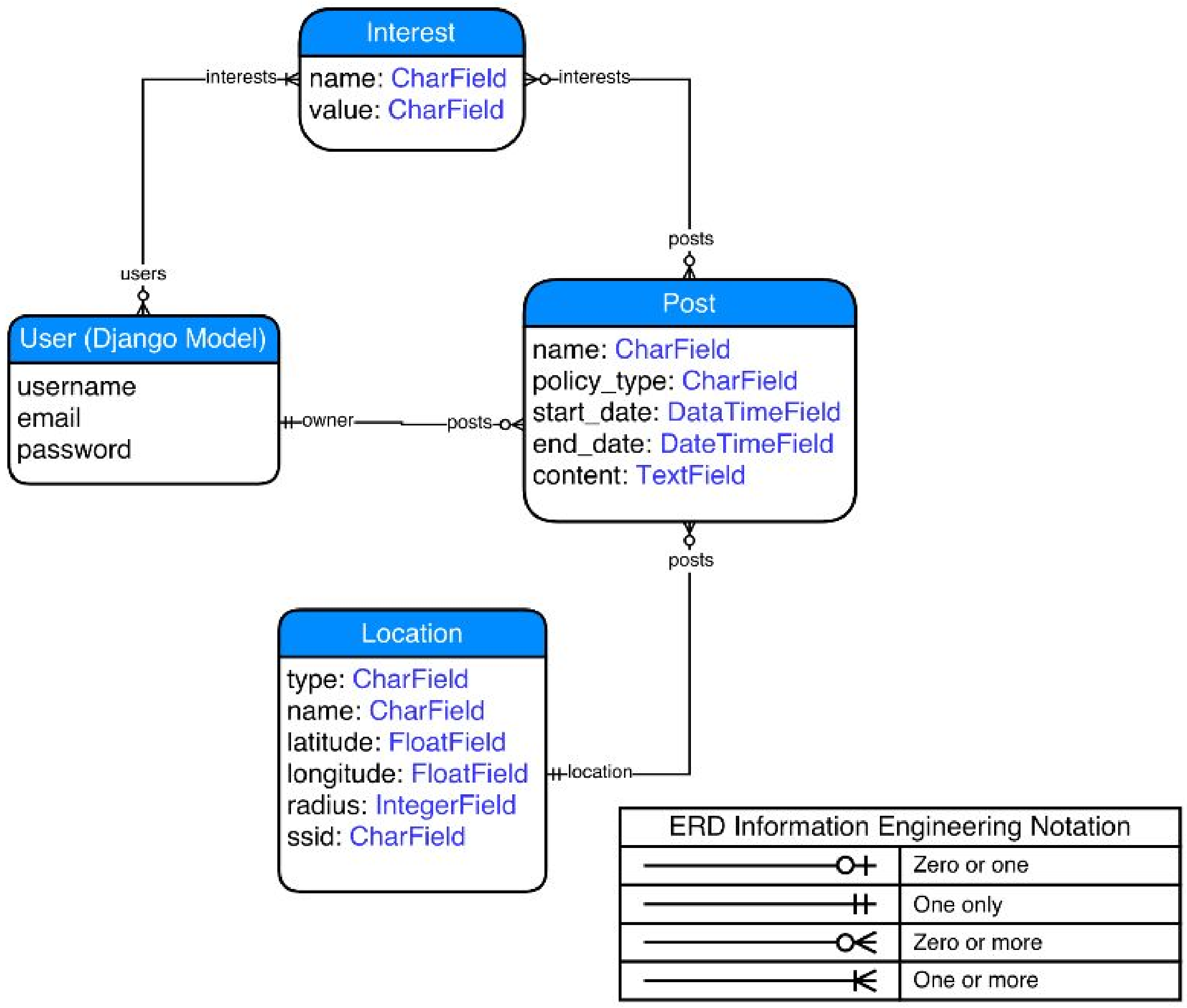


Fig. 2 - Server Database Relational


Interest
name: CharField
value: CharField
interests
interests
users
posts
User (Django Model)
username
email
password
owner
posts
Post
name: CharField
policy_type: CharField
start_date: DataTimeField
end_date: DateTimeField
content: TextField
posts
Location
type: CharField
name: CharField
latitude: FloatField
longitude: FloatField
radius: IntegerField
ssid: CharField
location
ERD Information Engineering Notation
Zero or one
One only
Zero or more
One or more