Project Report

## Ubiquitous and Mobile Computing - 2016/17

Course: MEIC

Campus: Alameda

Group: 12

Name: Luís Filipe Abreu dos Santos Number: 77900 E-mail: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Name: Catarina Alexandra Pereira Cepeda Number: 77966 E-mail: catarinacepeda@tecnico.ulisboa.pt

Name: Paulo Jorge Almeida dos Anjos Number: 87822 E-mail: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

*(PAGE LIMIT: 5 pages – excluding the cover)*

## 1. Achievements

*The following table describes which features stated in the project specification were implemented.*

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

\*change this, if we have time to implement HTTPS security

## 2. Mobile Interface Design

*Draw the activity wireframe of your program. Explain it succinctly. Describe any specific features of the program behavior that have not been explicitly stated in the project specification. The wireframe diagram can be added to the report in a separate appendix not counting for the 5-page limit.*

## 3. Baseline Architecture

*The following subsections describe the baseline architecture of our solution.*

**3.1 Data Structures Maintained by Server and Client**

The server stores and manages user accounts, locations, keys, *key-pairs* associated to each user, and messages posted in locations by the users. Each client maintains data related to each user that has logged in on the device – namely, the users’ *key-pairs* and their posted, received and read messages –, as well as a complete list of locations and keys pertaining to *key-pairs* of all registered users. In both the server and the clients, this data is persistent and stored in databases. Additionally, the client makes use of a key-value storage provided by the framework, to keep simple information such as the device’s current geographical coordinates and the list of SSIDs of detected WiFi signals – henceforth referred to as *coordinates* – which can be shared by several components.

**3.2 Description of Client-Server Protocols**

Initially the client must sign up or log in with the credentials provided by the user; the server will generate a session key in response, and send it back to the client, which stores it. The client may now issue requests for listing, creating or deleting *key-pairs*, locations and messages, which must be accompanied by the session key. These requests contain the data required to fulfill them, such as the information entered by the user, in the case of creation; or IDs defined by the server after creation, in the case of deletion.

With regards to the centralized message delivery, the protocol is as follows:

Periodically, the client will send a request to the server, containing the device’s current coordinates. The server will then retrieve all the *active* messages (whose visibility time window has not expired) posted in locations that match the coordinates sent by the client. For each of these messages, the server will compare its posting policy (whitelist and blacklist), as specified by the message’s author, with the *key-pairs* defined in the user profile associated with the request, and the messages that are selected are then sent back to the client.

The criteria used for filtering the messages are: if the whitelist is not empty, the client will not receive the message if the user profile does not contain any of the *key-pairs* in the whitelist. In the case where the user profile contains at least one *key-pair* in the message’s whitelist, or the whitelist is empty, the client will receive the message if the user profile does not contain any *key-pair* that is present in the message’s blacklist.

Upon receiving the messages, the client will merge them with the already stored messages. Messages that are stored locally but are not in the list of received messages are deleted from the database; messages that are in both the list of received messages and the database are kept; the remaining messages are new messages, and thus they are stored in the local database. (This approach is also used when merging a list of keys, *key-pairs*, or locations retrieved from the server.)

**3.3 Description of P2P Protocol for Decentralized Message Delivery**

The protocol designed for decentralized message delivery is described below:

Periodically, the client will scan the database for all the user’s posted, active P2P messages and, for each one, compare its location with the device’s current coordinates, retrieving those that match the device’s current location. For each message, the client will then create a *match request* associating each message’s unique P2P ID, generated in that moment, with the respective whitelist and blacklist. The unique P2P ID is a hashed value generated based on the user’s username and the local ID of the message. At this point, the *match request* is sent to every peer detected by the device.

When a peer, who is also a client, receives a *match request*, it will match each message’s posting policy against the *key-pairs* defined in the user profile, based on the same criteria described in the previous subsection. Additionally, to avoid storing the same message multiple times, the peer will also scan the database for received messages with the same P2P ID of a message present in the *match request*, and in case of match, will skip the filtering of the posting policy for that message. After this process, the peer will create a *match response* associating each message’s unique P2P ID with a boolean, representing whether the peer expects to receive that message.

Upon receiving the *match response*, the client will create a *message request* containing the information of every message which the peer declared to expect to receive in the response. When the peer receives a *match request*, it will store all the messages in its local database, and finally send an *acknowledgment* to the client, terminating the communication.

**3.4 Other Relevant Design Features**

*Describe any other relevant design features specific to your project. If there’s nothing to say, feel free to remove this subsection.*

The application and database have been designed with the intent of fully supporting multiple user profiles in a single device. Although we did not implement the components of the User Interface that would allow for a *fast-switch* between user profiles – currently, if the user wants to switch to another account, it will have to log out from the current account; however, all data will still be available when they log back in – we thought that it was important to note.

*Talk about design decisions re: doubling the alarm interval because of battery concerns, shutting down all services when the network is off. -> maybe not here, maybe in “optimizations” of implementation section*

## 4. Advanced Features

**4.1 Security**

*Describe the protocols that were designed to achieve the security goals of your project.*

**4.2 Relay Routing**

*Describe the protocols for relay routing.*

## 5. Implementation

*Describe any relevant implementation choices, in particular:*

* *Programming language and platform used to implement the server*
* *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*
* *Other issues you may find relevant, e.g., optimizations*

## 6. Limitations

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

## 7. Conclusions

*State the conclusions of this work.*

*Please provide some input on how the practical component of the course could be improved in future editions.*

## 8. Appendix