Project Report

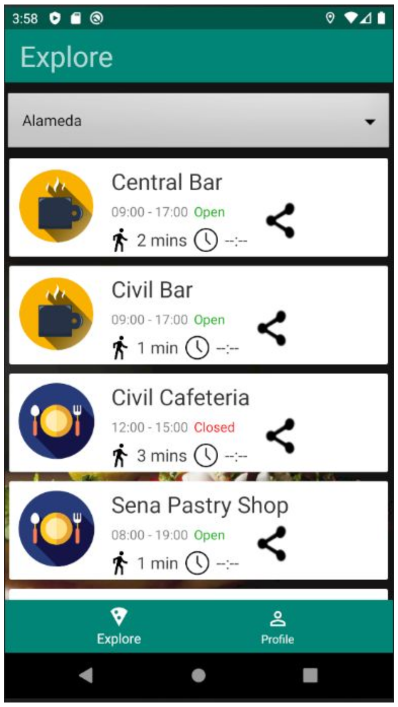
Mobile and Ubiquitous Computing - 2019/20

Course: MEIC

Campus: Alameda

Group: 9

Name: Diogo Pereira	Number: 86409	E-mail: diogomarante@tecnico.ulisboa.pt
Name: Francisco Matos	Number: 86415	E-mail: francisco.a.c.matos@tecnico.ulisboa.pt
Name: Pedro Custodio	Number: 86496	E-mail: pedrombcustodio@tecnico.ulisboa.pt



1. Features

Component	Feature	Fully / Partially / Not implemented?
Mandatory Features	List Food Services by Campus	Fully
	Show Food Service Information and Menu	Fully
	Show Dish Information and Photos	Fully
	User Profiles	Fully
	Dish Information Submission	Fully
	Dish Photo Submission	Fully
	Automatic Campus Selection	Fully
	Food Service Queue Estimation	Fully
	Photo Caching	Fully
	Cache Preloading	Fully
Securing Communications	Encrypt Data in Transit	Fully
	Check Trust in Server	Partially
Meta Moderation	Data Flagging	Not implemented
	Data Sorting and Filtering	Not implemented
	User Trustworthiness	Not implemented
User Ratings	Dish Rating Submission	Fully
	Average Ratings for Dishes / Food Services	Fully
	Rating Breakdown (Histogram)	Fully
User Accounts	Account Creation	Fully
	Login / Logout	Fully
	Account Data Synchronization	Fully
	Guest Access	Fully
Social Sharing	Sharing Food Services	Fully
	Sharing Dishes	Fully
Friend Tracking	Location Sharing Option and Selfie	Not implemented
	Show Photos of People at Food Services	Not implemented
	Sort Photos by Friend Likelihood	Not implemented
User Prompts	Prompt Upon Queue Entry	Not implemented
	Prompt Upon Queue Exit	Not implemented
Localization	Translate Static Content	Not implemented
	Translate User Submitted Content	Not implemented
Dietary Constraints	Dish Categories and User Diet Selection	Fully
	Dish Filtering	Fully
	Food Service Filtering	Fully

Optimizations

Openrouteservice API is only called one time after the app is launched when the user changes the campus. This saves power and time but makes the information less accurate if the user moves without relaunching the app.
Whenever an user uploads an image, two different images are created, one smaller (thumbnail) and the original. This way, if the user does not need to see the image in its fullest resolution (tapping the image), only the thumbnail will be used, which converts to faster downloads and less cellular data usage.

Food services mapping: by using an HashMap to store all food services, it allows for a faster and less power consuming access of a single food service. The extra space required is not significant to justify not taking this approach. The same thing goes for the food service - beacon name mapping.
UI list food services: since some information in the layout requires an API/server request, the app sets the view before the requests and then updates with the proper information.
Offline mode: all the static information regarding food services is stored in the Global state without using any server requests. This allows the user to have access to the list of food services in each campus.

2. Mobile Interface Design - see wireframe.png

3. Server Architecture

3.1 Data Structures Maintained by Server and Client

Two maps, “places’ and “users”, are used by the server in order to keep the state of the system in memory. The first one, “places” has all the information corresponding to the Food Services, such as the statically defined data (time-tables), and crowd-sourced data (dishes, menus, ratings, etc.). While “users”, contains all the information necessary to identify a user, as well as whether he is logged in or not, and whether he is in a queue or not. Meanwhile, the client does not need to have information about every user, so it only has a class with the current user information (null if it is a guest). For the Food Services, a similar map to the “places” was implemented.

3.2 Description of Client-Server Protocols

For the communication, we are using HTTPS protocol, in order to encrypt all communication Client-Server. On top of it, a REST-like API was implemented, which kept the communication simple, easy to use and extendable if needed.

3.3 Other Relevant Design Features

For optimizations, the server’s certificate should not be bundle with the rest of the application, since this way it is not possible to update it. Which means, whenever the certificate is expired, or compromised the system would stop working. Moreover, only the server is authenticated, not the clients. Finally the password and other sensible information should be salted and hashed, which is not implemented in our application.

4. Implementation

4.1 State

Mobile: the state is maintained using a global state that stores all the data of the application except the duration to the food services.

4.2 Threads and AsyncTasks

Whenever the server receives a request, it automatically creates a new thread to handle such request. This is done through “http.NewServeMux()”, which is handled by the library “net/http” from the language itself.

Whenever the application wants to make an https request, it creates an AsyncTask to fulfill that request.

4.3 Android components

4.3.1 Broadcast Receivers

WIFI Broadcast Receiver - This broadcast receiver handles beacon connections updates from the termite application

4.3.2 Activities

Activity Name	**Connected activities (intent objects)**	**Global State Variables**	**API and server (data)**
Add Picture Activity	Photo Gallery Dish Activity (Dish Name, Category, Price) Share (share body) List Food Service Activity Profile Activity	Image Memory Cache	Server - uploadImage
Dish Activity	Add Picture Activity (Dish Name, Category, Price, Food Service Name) Share (share body) List Food Service Activity Profile Activity	Food Service	Server – rateMenu Server – fetchCacheImages
Food Service Activity	Share (share body) Menu Activity (Food Service Name) List Food Service Activity Profile Activity Google Maps	Localization Food Service	Google Maps
List Food Services Activity	Food Service Activity (Food Service Name, Duration, Queue Time) Profile Activity	Localization Campus Food Services Duration	Openrouteservice (Duration)

Login Activity	Register Activity List Food Service Activity	Localization Status Connectivity Food Services	Server – prefetch Server – registerUser Server - toggleQueue
Menu Activity	Dish Activity (name, category, price, food service, dish index) List Food Service Activity Profile Activity	Food Services	Server - uploadDish
Profile Activity	List Food Service Activity	Food Services User	
Register Activity	List Food Service Activity	User	

4.4 Communications handling

Activity - Activity: using intents with the content described above

Activity - Server: Activity establishes an Https URL Connection with the server to either retrieve or upload data. This is done using JSON.

Activity - Openrouteservice: Activity establishes an Https URL Connection with the server sending the user and the food services' coordinates. The API then responds with a matrix with the distances between the coordinates sent. This is done using JSON.

4.5 Server programming language and platform used

The server was implemented in golang in the Visual Studio Code editor, where we used a REST-like API.

4.6 External libraries

github.com/safari/regression : This was the library used to train and predict waiting queue times in our go server.
com.synnaps:carouselview:0..1.5 : Used to display the dish pictures.
de.hdodenhof:circleimageview:3.1.0 : Used to display the user profile picture
com.github.AnyChart:AnyChart-Android:1.1.2 : Used to create the rankings histograms.

5. Running the Prototype

Run the prototype following these steps:

0 – Setup: Before running the prototype there are a few code changes needed in order to work on device. First the user needs to change the variable URL in the states/GlobalClass to his/her own IPv4 address. Then, if the user wants to run on an emulator with durations, he needs to change the function getLocation2 , also in states/GlobalClass, by replacing NETWORK_PROVIDER with GPS_PROVIDER. Then the user needs to create an emulator or connect your device to his/her computer.

1 – Launch server: open your terminal in the server folder and type the command: “go run server.go”. If you are using windows, we recommend you install and use Git Bash.

2 – Launch Termite Client: Install Termite and launch it on your terminal.

3 – Launch FoodIST: After selecting a device, just press the ‘Run’ button and the app should be running within a few seconds.

6. Simplifications

Campus localization: in order to detect if the user is inside a campus, the app checks if the user is inside a bounding box where the min/max vertex are the min/max coordinates. This avoids complex computations and possible API requests. Moreover, since the campus are too far apart, the worst case scenario is the app detect a campus and the user is near it (not inside). This also gives a slack for the localization accuracy.

7. Bugs

LocationListener - mobile location cannot use the gps as provider despite the permissions enabled and network working fine. It uses network instead. This at least avoids forcing the user to turn on gps.
Server - Access to server’s structures are not synchronized, which can lead to race conditions

8. Conclusions

This project allowed the group to get a better understanding of Android programming. It allowed us to explore many functionalities from simple layout construction to more complex areas like localization.

One of the main problems we had was testing because of how long it takes to test both on emulator and device. In addition, sometimes the app would just explode without any logs. Regarding Https requests, we had to use Postman to test because many of the requests messages would not show in the Android Studio profiler. Also, testing Termite and queue regression together is very hard given that the regression needs samples and termite works in real time. Our approach was to test things separately first.

Another time consuming problem was getting the user localization. We ended up using a location listener but could not fix a bug where the mobile is not able to get the gps provider.

One thing that would probably improve the flow of the practical component of this course would be a list of possible API to use. Some parts of the project could not be done without using an external API. Therefore, providing an open source alternative early on would make the flow much clearer and more dependent on students.