Project Report

Mobile and Ubiquitous Computing - 2020/2021

Course: MEIC

Campus: Tagus

Group: 12

Name: Miguel Levezinho **Number**: 90756 **E-mail**: miguel.levezinho@tecnico.ulisboa.pt

Name: Rafael Figueiredo **Number**: 90770 **E-mail**: rafael.alexandre.roberto.figueiredo@tecnico.ulisboa.pt

Name: Ricardo Grade **Number**: 90774 **E-mail**: ricardo.grade@tecnico.ulisboa.pt

1. Features

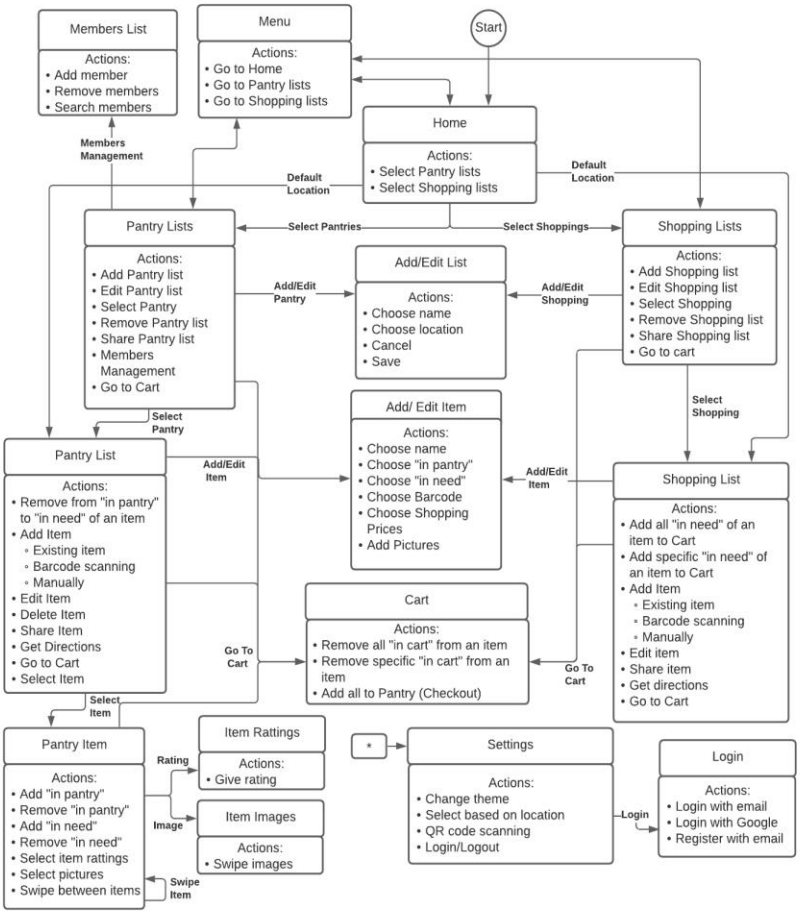
Component	Feature	Fully / Not implemented?
Mandatory Features	Support Multiple Pantry Lists and Shopping Lists	Fully
	Associate Lists with a Location	Fully
	Automatic List Opening When Near Location	Fully
	Show / Edit List Information	Fully
	Show / Edit List Items	Fully
	Associate / Filter Items with Shopping Lists	Fully
	Select Items with Barcode	Fully
	Add Pictures to Items	Fully
	List Synchronization	Fully
	Store Checkout Queue Estimation	Fully
	Crowd-source Pictures	Fully
	Crowd-source Prices	Fully
	Caching	Fully
	Cache Preloading	Fully
Securing Communications	Encrypt Data in Transit	Fully
	Check Trust in Server	Fully
Meta Moderation	Data Flagging	Not implemented
	Data Sorting and Filtering	Not implemented
	User Trustworthiness	Not implemented
User Ratings	Item Rating Submission	Fully
	Show Average Ratings	Fully
	Rating Breakdown (Histogram)	Fully
User Accounts	Account Creation	Fully
	Login / Logout	Fully
	Account Data Synchronization	Fully
	Guest Access	Fully
Social Sharing	Sharing Items	Fully
	Sharing Item Data in Context	Fully
Optimize the Shopping Process	Optimize for Time	Not implemented
	Optimize for Cost	Not implemented
User Prompts	Prompt Photo Submission	Fully
	Prompt Price Submission	Fully
Localization	Translate Static Content	Not implemented
	Translate User Submitted Content	Not implemented
Suggestions	Compute Most Likely Item Pairings	Not implemented
	Prompt the User with Suggestions	Not implemented
Smart Sorting	Compute Item Ordering in Store	Fully
	Sort Shopping List for Store	Fully

Optimizations

The selection of Items by Barcode is carried out by scanning them. In this way, the User saves time by not having to search for or enter its Barcode. If an Item with this Barcode already exists, all the crowded information is fetched and used to create the Item.
Item Images can be resized and rotated to fit a predefined size, making them homogeneous and better displayed on the screen with sufficient resolution.
Items can be removed from the Shopping Cart after being added to it, making the User able to remove them if he makes a mistake.

A User can delete a Pantry, a Shopping or an Item if he no longer wants it or created it by mistake.
A User can change the theme in the Application between light and dark mode, conveniently the Application maintains the mode that the User has chosen.
A User can create a new Item in a Pantry while he is on a Shopping List, in case he finds the Item on a shelf and wants to bring it in, by scanning the Item's Barcode, for example. In addition, the User can also edit an Item, in a Shopping List, allowing an easy change of the Price of the Item.
A User can easily add an Item to a Pantry that he already has in another Pantry, choosing from the List of Items that already exist in his other Pantries.
A User can login with Google other than entering his email and password, being able to login with the press of a single button and without the need to memorize a password for the Application.
The Sharing of an Item is carried out by generating a link that, when clicked, redirects to a fragment of our Application where the Item can be created in a Pantry of choice with the Item Name and all crowded information filled in. The Sharing Preview is composed of a friendly message and an Image that appears on WhatsApp and Twitter.

2. Mobile Interface Design



- Each box represents a fragmented view or a dialogue.
- The start circle points to where the application starts.
- Arrows represent possible navigations between views.
- The box with the asterisk means that all views have an action that leads to the Settings view.

3. Server Architecture

For the server-side of the project, Firebase was the chosen architecture to support ShopIST. The android application uses five main services from this provider:

- **Cloud Firestore**: A NoSQL document driven database where all the app information is being stored and, when applicable, shared. This service also provides an implementation for the data cache, and listeners to notify changes to the data and update it.
- **Firebase Cloud Storage**: A robust Cloud Storage service meant for user-generated data. In the context of the project, used to store images.
- **Firebase Cloud Functions**: An extension for serverless computing, which falls under FaaS (Function as a Service) where the focus can be put solely on the application end (i.e., writing the application logic that will be run on the cloud as functions). This paradigm is also much more cost-efficient from a production perspective since billing is based on used resources.
- **Firebase Authentication**: Used for creating different accounts. Supporting the following authentication mechanisms: Guests, Email / Password, or Google.
- **Firebase Dynamic Links**: Used to share links for items in social networks, related to the Social Sharing feature.

3.1 Data Structures Maintained by Server and Client

At the server, a document-driven database is maintained, and each collection of documents is briefly described below:

- **User**: Contains user-related data that can be used by the Application.
- **Pantry**: Represents a pantry in the application, and contains:
    - General information (name and location).
    - A map of pantry items (associated item, *In Need* and *In Pantry*).
    - Relevant cart state (number of items *In Cart* assigned to this pantry, and from which user).
    - A list of users who share this Pantry.
- **Shopping**: Contains shopping general information (name and users sharing).
- **CrowdShopping**: Contains additional shopping information, which is applicable for sharing (location and queue time).
- **Item**: Contains general information of an item (name).
- **CrowdItem**: Contains additional item information, applicable for sharing (pictures, prices, ratings, and barcode).
- **Uniqueness**: Used to efficiently enforce uniqueness between Barcodes.
- **Token**: Stores tokens that are used to share pantries and / or Shoppings.
- **LinearRegression**: Relates to CrowdShoppings and stores information needed to calculate respective queue times.

At the client-side, two types of cache are maintained. A data cache, which stores database information in case the user goes offline; And an image cache, that is used to cache images on disk to prevent the overhead of having to request the images to Firebase each time. Both caches ensure some measure of availability to the application.

3.2 Description of Client-Server Protocols

To communicate with the server, the application uses a system of listeners, which actively listen to changes in the database and notify all relevant clients of the modifications. For example, when a user makes changes to a shared pantry, the database is updated, and listeners notify the clients (the pantry is shared with) of the change.

Another way to communicate with Firebase is using the Cloud Functions. In the event that some server-side logic is needed (for example, calculating the queue time of a shopping), a function call is made.

All communications with Firebase are done in a secure fashion. In addition, authentication is required to access / modify Firestore resources by means of security rules, written to ensure integrity and confidentiality. It is also worth adding that Firebase uses a derivation of Scrypt for password hashing.

4. Implementation

For this project, the Java programming language is the most used to develop the Android application. Regarding the Server, as stated above, it is implemented using Firebase. The Functions that are responsible for the Logic of the Backend are written in JavaScript. These functions can be called directly on the Application or be triggered by modifications to the Database. The Firebase's Database our server uses is Firestore which allows for the data to be synced across multiple Clients by creating real-time listeners of the data of interest, supporting the Application while offline by maintaining a Cache that is kept in the Mobile Device. In order to prevent malicious access to the Application data, our Database is secured with Security Rules that check whether the User is authenticated and authorized to access such data. The Images of the Items are kept in the Firebase Cloud Storage which can be downloaded through a Link that only authenticated Users have permissions to access.

External Libraries

- Firebase Libraries: Cloud Firestore, Cloud Storage, Cloud Functions, Firebase Authentication, Firebase Dynamic Links.
- Lists Pull to Refresh Layout: PullRefreshLayout.
- To Download and Cache Images: Glide.
- To Swipe List Item: SwipeLayout.
- To Resize and Rotate Images: uCrop.
- To Scan Barcodes and QRCodes: zxingAndroidEmbedded.
- Google / Firebase API Calls: Retrofit Gson OkHttp.
- Number Picker: NumberSlidingPicker.
- To Pick Pantries and Shoppings Location: PingPlacePicker.
- For the Rating Histogram: RatingReviews.

Android Components

Our Application is implemented using fragments, which can be navigated through using the actions that are defined in the navigation graph. When a fragment needs to pass arguments to another, it sends a Bundle that is received by the other fragment. Furthermore, all of them have access to the same ViewModel, which is bound to the *MainActivity*, where the general information about Pantries, Shoppings and Items (except its images) are mapped in memory.

Existing Threads in the Application

The UI Thread is responsible for dispatching events to the appropriate UI Widgets.

Worker Threads are generated whether a long-time operation needs to be performed. For example, calling the Google / Firebase API or when reaching out to the Server either to access / modify data or call Cloud Functions. This way the UI Thread is not blocked, always displaying feedback progress to the User when necessary.

How Communications are Handled

Already explained above in section **3.2** (implemented by Firebase).

How the State is Maintained

Already explained above in section **3.1** (Client-side caches).

5. Simplifications

- The Application assumes that whenever a User enters the checkout, it detects its leaving accurately, not hours later. To minimize the error that such operation could induce to our Application, there could be agreed a certain threshold of waiting in Queue, and if the threshold were superseded, the Queue Time would not be updated.
- In a context where multiple Users of a shared Pantry are shopping at the same time, the Cart is private to each User. The Cart could be made public by summing each *In Cart* quantity of all User's Carts that possess the shared Pantry and guaranteeing synchronization between Cart's operations.

6. Conclusions

This project helped us learn how to develop a non-trivial Android Application, allowing us to understand several key aspects of mobile development. It also allowed us to learn how Firebase works as it was used as a serverless alternative to implementing a lot of features that were needed for the project. We also learn more about SQLite, even though it ended up not being needed in the final project due to caching made by Firestore.

We would like to thank professor Luís Pedrosa, for all the time spent responding to our questions, even off office hours on discord.

Feedback on the Project

- In addition to the additional features proposed in the project statement, we thought it might be interesting to allow students to request personalized features, which students would like to implement in the project, which would need to be accepted by the teacher and receive a percentage based on its difficulty.
- The project statement can be misleading in some ways, it could be made clearer by adding well-divided objective points based on the features that were needed to implement.
- The presentation of the project in the laboratory class could have been earlier, as it would help the students to have a clear idea of the project. On the one hand, there may be some students who are waiting for the presentation of the project to start the development of the same. On the other hand, students who have already started would already have a design mindset and would not pay as much attention to the subject of the presentation.