

CO3015 Computer Science Project

Dissertation

An Android Application of a Canteen Ordering System

Despina Dorotheou (dd186)

Contents

1	Abstract.....	4
2	Introduction	5
2.1	Motivation:	5
2.2	Definition:	5
2.3	Aims:	6
2.4	Objectives:.....	6
3	Survey of Literature/Information Sources	7
4	Requirements	8
4.1	User/Android Application	8
4.1.1	Login/Signup.....	8
4.1.2	Main	8
4.1.3	Main/Home	8
4.1.4	Main/Search	9
4.1.5	Main/Basket	9
4.1.6	Main/Account	10
4.2	Admin/Web Application	11
4.2.1	Login	11
4.2.2	Main/Orders	11
4.2.3	Products.....	11
4.2.4	Offers	11
4.2.5	Deals	12
4.2.6	RESTful Service	12
5	Software Architecture.....	13
5.1	MySQL.....	13
5.2	Web Application	15
5.3	Android Application.....	16
6	Implementation.....	16
6.1	Web Application	16

6.2	Android Application.....	20
7	Testing	23
7.1	Web Application	23
7.1.1	Repository Layer.....	23
7.1.2	Service Layer	24
7.1.3	Controller Layer.....	26
7.2	Android Application	29
7.2.1	Signup Activity	29
7.2.2	Login Activity	30
7.2.3	Main Activity	31
8	Critical Appraisal.....	34
8.1	Critical Analysis	34
8.2	Personal Development.....	35
9	Conclusion	35
10	References.....	36
11	Appendix.....	37
11.1	POST request to Firebase Platform.....	37
11.2	JavaScript Search function	37
11.3	Java objects to JSON objects	38
11.4	HTTP GET request.....	38
11.5	Deserialization.....	39
11.6	Vegan/Vegetarian Filters.....	39
11.7	Check order status	40
11.8	Edit/Delete Order	40

DECLARATION

All sentences or passages quoted in this report, or computer code of any form whatsoever used and/or submitted at any stages, which are taken from other people's work have been specifically acknowledged by clear citation of the source, specifying author, work, date and page(s).

Any part of my own written work, or software coding, which is substantially based upon other people's work, is duly accompanied by clear citation of the source, specifying author, work, date and page(s).

I understand that failure to do this amounts to plagiarism and will be considered grounds for failure in this module and the degree examination as a whole.

Name: **Despina Dorotheou**

Signed: **Despina Dorotheou**

Date: **02/05/2018**

1 Abstract

The canteen of the university needs an application which will be used by students and staff to order food fast and easy while they are in the campus. By using the application, the users should be able to search for a meal and purchase it. Therefore, that application would help the users save some time of the time they usually spent in large queues during their lunch break.

I have developed an Android application that serve this purpose. The application is designed in such a way to provide users with the ability to search for product through various ways such as searching for daily offers and deals, as well as an extremely fast checkout.

Furthermore, I have developed a web application that administrators can use to modify the database's data without any MySQL, or Computer Science knowledge, in general, and send said data to the Android application using the RESTful API. Moreover, the web application provides the administrators with the ability to handle all the upcoming orders made by the Android application's users and notify them about their order's status.

Due to the fact that the project is large and consists of two very different applications, the implementation challenges were many. Those challenges needed be tackled and solved fast using a wide variety of different techniques.

2 Introduction

2.1 Motivation:

Like the rest of my generation, I have grown in a world where we are surrounded by technology. Technology, and more specifically phones, has become essential for most of our daily activities such as ordering food.

The motivation behind the project was its target audience, mostly university students. As a student, I knew the pressure that was caused by a very busy schedule and it was easy for me to understand why a university needs an application for its canteen. The application purpose was to make the life of every student or staff, who spends a large number of hours in the university campus, easier, therefore, to make my own life easier.

2.2 Definition:

The main purpose of the project is to develop an Android application which will be used by students and staff, so they can purchase food from the university's canteen fast and easy manner.

By using the application, users will be able to see a list of menu options, deals and offers of the day. Furthermore, they will be able to search for a meal using options such as ingredients or food type, as well as set their dietary and cost constraints.

Once the users choose their meal, they will be able to pay for it in advance through the application (debit /credit card) or at the collection point (cash).

After placing their order, the users will be informed about their order's status through a series of notifications and within a limited time frame (5 minutes), they will be able to update or delete their order.

Finally, there will be a web application for the administrator, which will be used mainly to connect the Android application with the database.

By using the web application, administrators will be able to modify the data in the database, handle upcoming orders and notify the user in real time.

2.3 Aims:

My main goal for this project is to make a user-friendly and helpful Android application of a canteen ordering system.

I aim to learn as many techniques as possible about designing, implementing and testing an Android application, as well as making sure the application will make the life of every user easier.

I intend to make a very simple and easy to use web application where administrators without any MySQL, or Computer Science knowledge, in general, can add, delete and change information in the database, which will be connected to the Android application.

Moreover, I intend to improve my problem-solving skills by tackling any new possible challenges that might come up during the project. I also would like to improve my time management skills since it is a project that is going to take limited time, and there will be many challenges that will need to be accomplished during that period.

2.4 Objectives:

Firstly, I want to make my application as helpful as possible with the most accommodating and functional features, so that every student and staff member would want to have it on their phone. In order to achieve this, I decided that while developing the application I would also be in contact with students and have them give feedback on some of the features.

Secondly, to achieve my goals, on the one hand, I need to acquire familiarity with Android SDK which will entail understanding concepts of Android and using said concepts programmatically with Java. Furthermore, I need to improve my knowledge regarding XML for the application's design. On the other hand, I need to do a more thorough research on Spring boot Web applications as well as on RESTful API which is based on representational state transfer technology that will help me to force these two applications (Android and Web) to communicate with each other and share information, more specifically, database data, through HTTP requests.

Thirdly, I must do a market research, collect valuable information that will help me find out whether there is a market for the product/service (application) that I intend to develop during this project; and help me to make the correct business decisions. I, also, need to do a research on similar applications, find some new ideas, figure out how they work and what things I should do or avoid.

Finally, I have to make the best decisions possible regarding all the expected or even unexpected challenges that will come up while working on the project.

3 Survey of Literature/Information Sources

The first thing that I did was, a background research on the Android applications. Due to the fact that I have never been involved with Android applications before, I wanted to familiarise myself with the Android SDK. In order to do that, I followed some tutorials and I created a simple, hello-world style Android application. [1] I also researched similar applications to the one I have developed, to get an idea about their functionalities and their designs. The research also helped me to figure out things that I would like to avoid or do better in my application. Some of these applications were the Upay [2] and the Uber Eats [3]. I researched the Upay application because the university is currently using it to share their canteen's promotions and also because it helps the users to pay faster for their meals on the counter. The reason I researched Uber Eats application is because of its design as I believe it is very simple and user-friendly, therefore it inspired me to do something similar.

Furthermore, I researched Spring boot Web applications despite developing a simple Spring boot Web application before. I watched a course-type tutorial in order to get some help regarding which frameworks to choose, i.e. code libraries, tool sets, application programming interfaces etc, and dependencies I was going to use. [4]

In addition, I did a background research on how to connect my database to the system. At the beginning, I started researching how to connect the database directly to the Android application but then I decided to connect the database to the web application and the web application to the Android one, using the RESTful API. I chose the RESTful API mainly because of its client-server principle – “By separating the user interface concerns from the data storage concerns, we improve the portability of the user interface across multiple platforms and improve scalability by simplifying the server components.” [5].

I did a research on the RESTful API concerning how to use it in order to send HTTP requests from an Android application to a web application but also how to handle the responses. I watched a course-type tutorial explaining how the HTTP requests and responses work and how to connect the two applications. [6] Later, I researched how to serialize and deserialize objects since I wanted to send the list of products (objects) from the web application to the Android application. [7]

In addition, I researched how to upload images in a database using Binary large object [8] (blobs) and how to include the images in the json (JavaScript Object Notation) response to the Android application through the RESTful API. I found out that one way of achieving this is Base64 [9] encoder, which helped me to convert an image from an array of bytes to an encoded string.

Moreover, I did a research on the Firebase Cloud Messaging [10] (FCM) and then I decided to include it in my project for the notification service. Further to this, I had to research how I can connect my applications with the Firebase Cloud Messaging in a way that the web application will make a POST request to Firebase and the Firebase will send a notification to the Android client [11].

Finally, during testing for the web application I did a research on how to test a Spring Boot Application [12] and I did further reading on Mockito Framework [13] but at the end I

decided to use both in order to test all the main parts of MVC. Despite this, for the Android application, while researching on how to test an Android application I found out the Robolectric Framework [14] which allows a test style similar to black box testing [15] and tests both user interface (UI) and functions with the suitable unit testing scenarios.

4 Requirements

The following are the project's requirements separated into each application.

4.1 User/Android Application

4.1.1 Login/Signup

Requirement ID	Requirement
#R001 [COMPLETED]	The user will be able to login into his/her account.
#R002 [COMPLETED]	The user will be able to create an account through the sign-up page.
#R003 [COMPLETED]	The user will be able to click on "Remember me" and save his/her email.
#R004 [COMPLETED]	The application will send HTTP request to the web application with the user's email and password for authentication.
#R005 [COMPLETED]	The application will send HTTP request with the information that the user entered in the sign-up page to check the information entered and create the account.

4.1.2 Main

Requirement ID	Requirement
#R006 [COMPLETED]	The application will send HTTP request to take all the products from the database.
#R007 [COMPLETED]	The application will send HTTP request to take all the deals from the database.
#R008 [COMPLETED]	The application will send HTTP request to take all the offers from the database.
#R009 [COMPLETED]	The user will be able to swipe down a refresh the data.

4.1.3 Main/Home

Requirement ID	Requirement
#R010 [COMPLETED]	The user will be able to click on a category and see all the relevant products.
#R011 [COMPLETED]	The user will be able to see the active categorical deals.

#R012 [COMPLETED]	The user will be able to see all the active offer and add the whole offer in the basket with one click.
#R013 [COMPLETED]	The application will prevent the user from adding an offer that is out of stock in the basket.

4.1.4 Main/Search

Requirement ID	Requirement
#R014 [COMPLETED]	The user will be able to see all the products in a scroll down list.
#R015 [COMPLETED]	The user will be able to see if the products are out of stock.
#R016 [COMPLETED]	The user will be able to click on a product and see more information about it, such as description, ingredients etc.
#R017 [COMPLETED]	The user will be able to search through the products by their name.
#R018 [COMPLETED]	The user will be able to search through the products by ingredients.
#R019 [COMPLETED]	The user will be able to set dietary constraints such as vegan, vegetarian etc.
#R020 [COMPLETED]	The user will be able to set cost constraints.
#R021 [COMPLETED]	The user will be able to search for a product using more than one filter.
#R022 [COMPLETED]	The user will be able to put a product into his/her favourite list.
#R023 [COMPLETED]	The user will be able to remove a product from his/her favourite list.

4.1.5 Main/Basket

Requirement ID	Requirement
#R024 [COMPLETED]	The user will be able to see all the products in basket, their quantity, a total of their prices, the applied discount and the final total by clicking on the basket icon.
#R025 [COMPLETED]	The application will save any products that are left in basket before the user signs out.
#R026 [COMPLETED]	After login, the application will put in the basket products that the user left from a previous session if they are still in stock.
#R027 [COMPLETED]	The user will be able to add a product in the basket.
#R028 [COMPLETED]	The user will be able to add a quantity of a product in the basket by increasing and decreasing the quantity of the product.
#R029 [COMPLETED]	The user will be able to remove products from the basket.

#R030 [COMPLETED]	The application will prevent the user from adding a product that is out of stock in the basket.
#R031 [COMPLETED]	Every time the user adds or removes a product, the application sends a HTTP request to update the product's quantity in the database.
#R032 [COMPLETED]	The application will calculate and apply the discount on the order.
#R033 [COMPLETED]	The user will be able to see the discount.
#R034 [COMPLETED]	The user will be able to pay for his/her order with a debit/credit card.
#R035 [COMPLETED]	The user will be able to select the pay by cash option and pay at the collection point.

4.1.6 Main/Account

Requirement ID	Requirement
#R036 [COMPLETED]	The user will be able to see all his/her favourite products in a section in the account page.
#R037 [COMPLETED]	The user will be able to see his/her previous orders in the order history section
#R038 [COMPLETED]	After checkout, the user will be able to see his order status.
#R039 [COMPLETED]	After checkout, the user will be able to add/remove items during the first 5 minutes.
#R040 [COMPLETED]	After checkout, the user will be able to delete the order during the first 5 minutes.
#R041 [COMPLETED]	If the user is editing his/her order after checkout, the admin will be notified by a HTTP request.
#R042 [COMPLETED]	The application will send a HTTP request every 10 seconds to check the status of the order until the order is collected.
#R043 [COMPLETED]	After checkout, if the order is ready for collection, the user will be notified by a notification with his/her order number.
#R044 [COMPLETED]	After checkout, if the order is collected, the user will be notified by a notification.
#R045 [COMPLETED]	The user will not be able to modify his/her order if the order is ready, even if the order is ready before the first 5 minutes.
#R046 [COMPLETED]	The user will be able to logout.

4.2 Admin/Web Application

4.2.1 Login

Requirement ID	Requirement
#R047 [COMPLETED]	The admin won't have to sign up, his/her account will be given.
#R048 [COMPLETED]	Only admins will be able to login into the web application.

4.2.2 Main/Orders

Requirement ID	Requirement
#R049 [COMPLETED]	The admin will be able to see a table of all the orders in the main page that are not collected.
#R050 [COMPLETED]	If an order's status is pending, the admin will be able to click on a "ready" button and notified the user that his/her order is ready for collection.
#R051 [COMPLETED]	If an order's status is ready for collection, the admin will be able to click on a "collected" button and notify the user.
#R052 [COMPLETED]	From the main page the admin can go and see the products, the offers and the deals.
#R053 [COMPLETED]	The main page will reload every 5 seconds, so the table with the orders is up to date.

4.2.3 Products

Requirement ID	Requirement
#R054 [COMPLETED]	The admin will be able to see all the products in a table.
#R055 [COMPLETED]	The admin will be able to add new products.
#R056 [COMPLETED]	The admin will be able to delete a product.
#R057 [COMPLETED]	The admin will be able to edit the product's information.

4.2.4 Offers

Requirement ID	Requirement
#R058 [COMPLETED]	The admin will be able to see all the offers in a table.
#R059 [COMPLETED]	The admin will be able to add new offer and add the products that are in the offer.
#R060 [COMPLETED]	The admin will be able to delete an offer.
#R061 [COMPLETED]	The admin will be able to edit the offer's information.

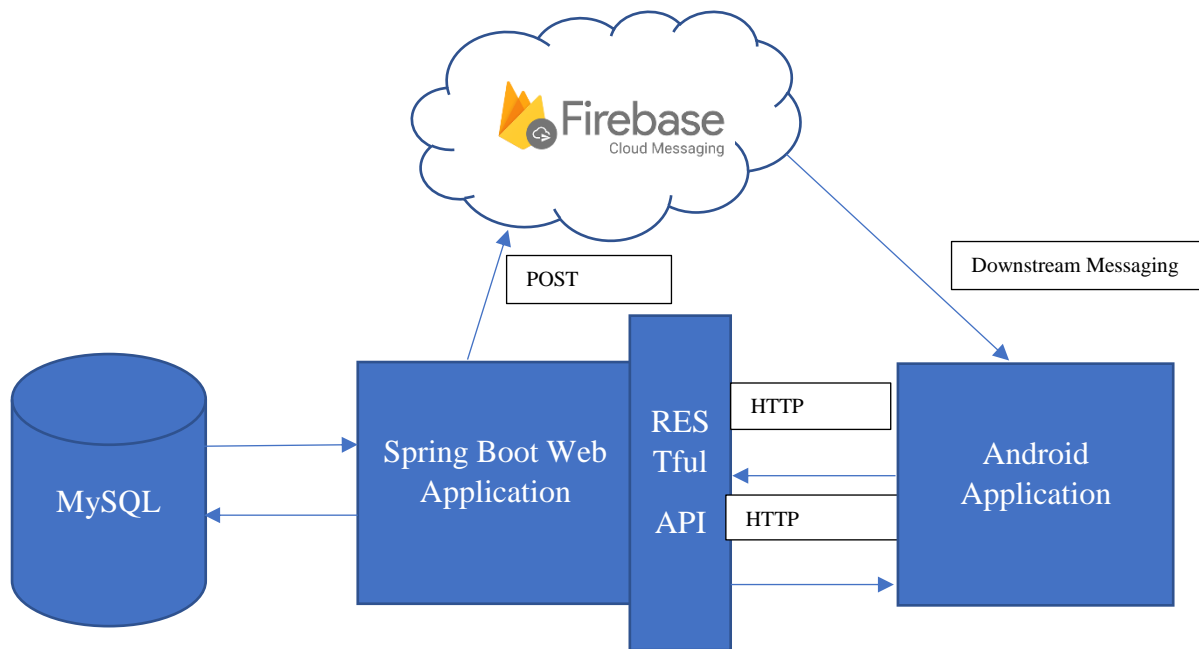
4.2.5 Deals

Requirement ID	Requirement
#R062 [COMPLETED]	The admin will be able to see all the deals in a table.
#R063 [COMPLETED]	The admin will be able to add new deal and add the categories that are in the deal.
#R064 [COMPLETED]	The admin will be able to delete a deal.
#R065 [COMPLETED]	The admin will be able to edit the deal's information.

4.2.6 RESTful Service

Requirement ID	Requirement
#R066 [COMPLETED]	The application will respond to HTTP request for login with a json object of the user if the user exists and it will respond with "Invalid" if the user does not exist in the database.
#R067 [COMPLETED]	The application will respond to HTTP request for sign up with "ok" if the user account was created successfully and it will respond with "Already existing user" if a user with that email exists in the database.
#R068 [COMPLETED]	The application will take the necessary information from the HTTP request for adding or deleting a product of the favourites list of a user and make the necessary changes in the database.
#R069 [COMPLETED]	The application will respond to the HTTP requests for the products/deals/offers with json arrays.
#R070 [COMPLETED]	The application will take the necessary information from the HTTP request for the checkout, create a new order and send it back to the android application as a json object. (same procedure for pay by card or cash)
#R071 [COMPLETED]	The application will take the necessary information from the HTTP request for deleting an order and make the necessary changes in the database.

5 Software Architecture



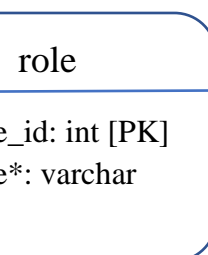
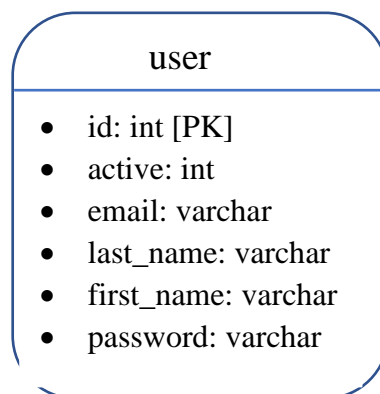
Software Architecture figure

My software system consists of three components, MySQL, the Spring Boot Web application and the Android application.

5.1 MySQL

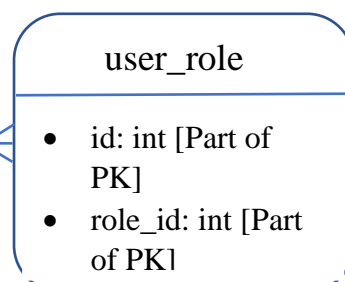
The web application is directly connected with the MySQL database, which is used by the whole software system. The database consists of 14 tables. All tables are created in such a way that every table's relationships are meaningful, efficient and easy to understand.

Firstly, there is the "user" table where the users are stored.



Then, there is the "role" table where are stored the different roles with their ids.

**role is the role's name, ex. admin*



And, there is a join table named "user_role", which joins the "user" and "role" tables for "many to many" relationship, meaning that every user may have more than one role and a role may belong to more than one users.

The “product_category” is the join table that joins the “product” and “category” tables for “one to many” relationship. Indicating that one product belongs to one category, but a category may have one or many products.

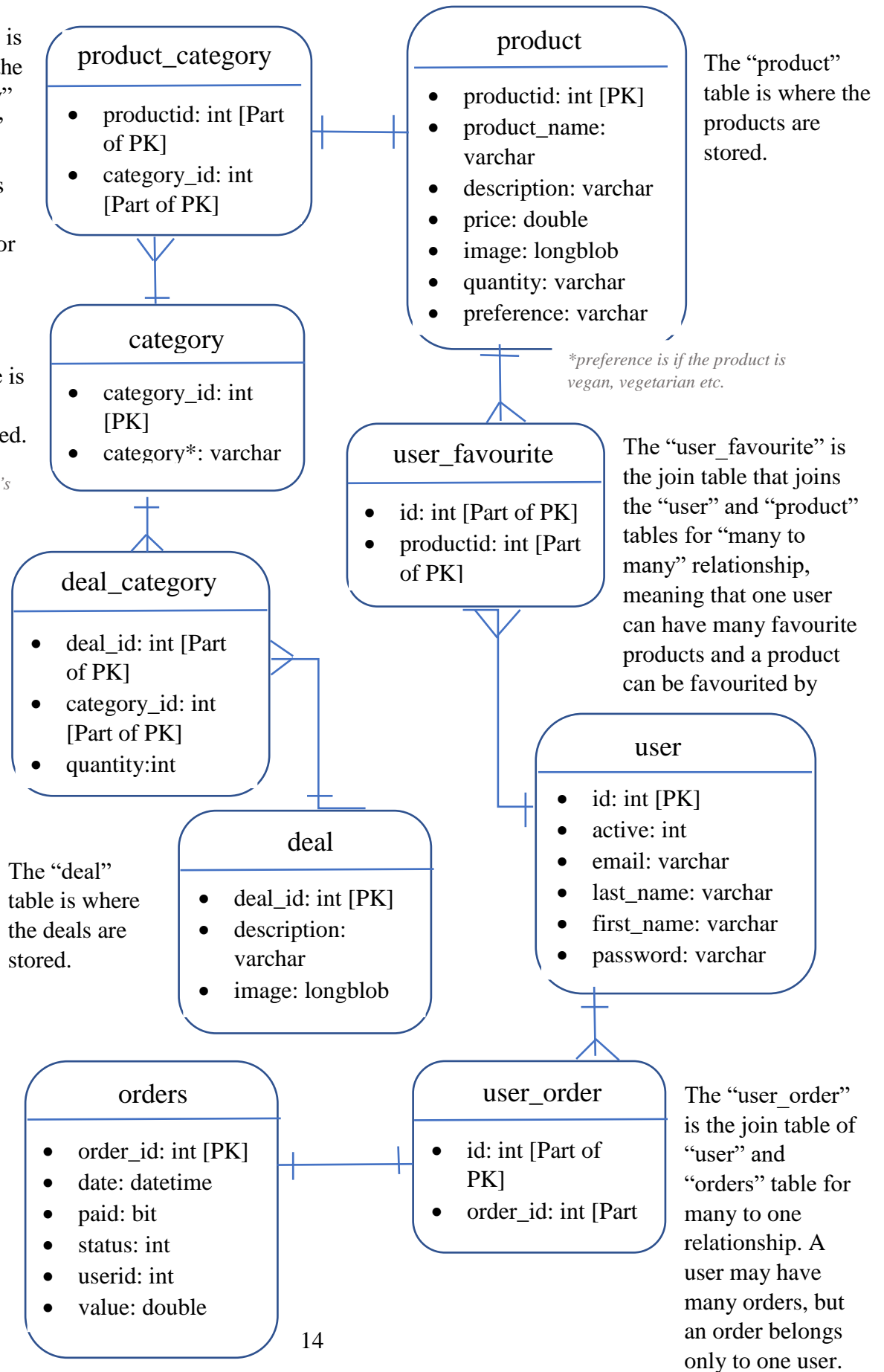
The “category” table is where the categories and their ids are stored.

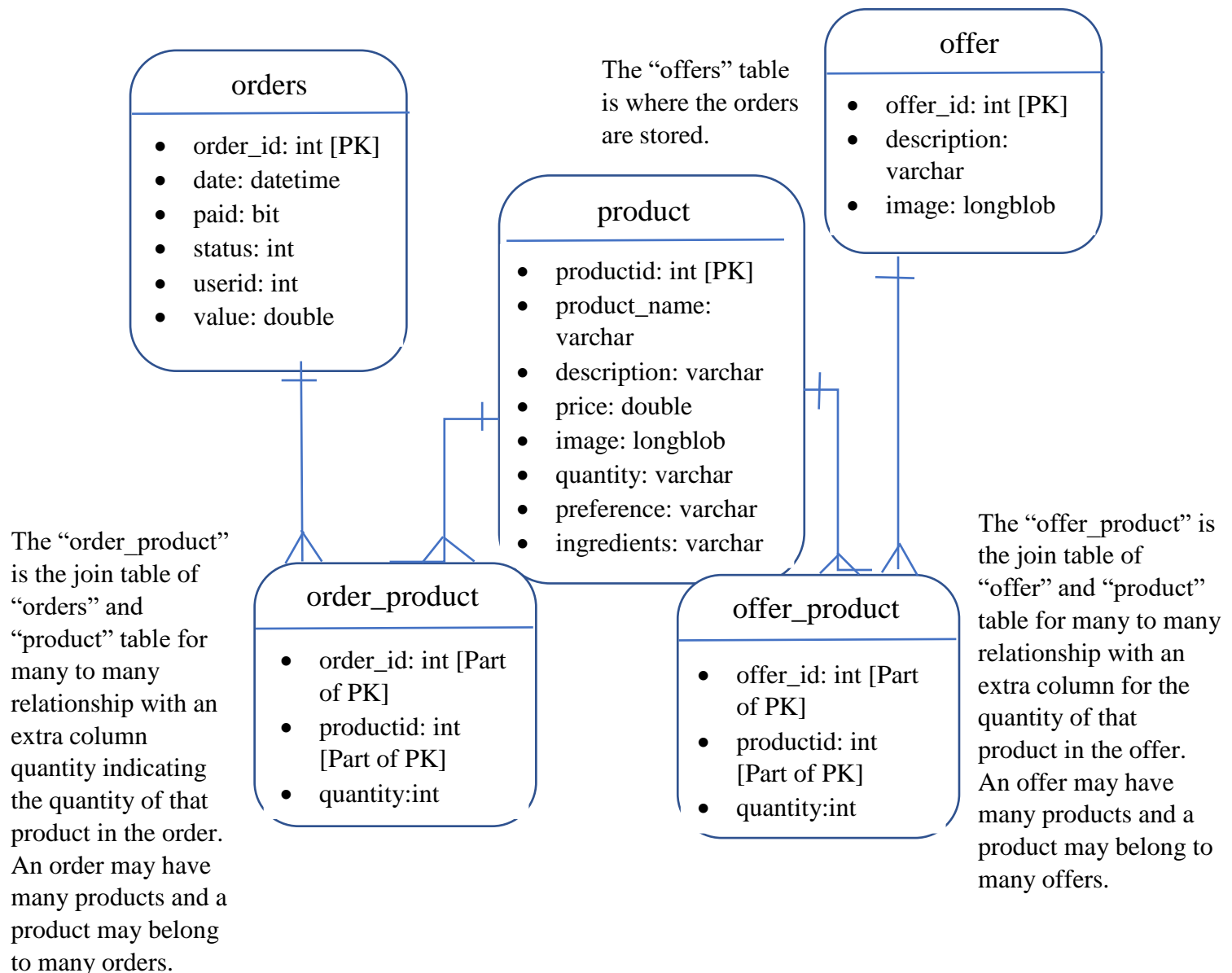
**category is the category's name, ex. Drinks*

The “deal_category” is the join table that joins the “deal” and “category” tables for “many to many” relationship with an extra column for the quantity of that category in the deal. Indicating that one category may be in many deals and a deal

The “deal” table is where the deals are stored.

The “orders” table is where the orders are stored.





5.2 Web Application

The web application is built as a simple Spring Boot application in MVC design and it was developed for three purposes. The first is to serve the administrators so they can modify the data and the tables in the database automatically. The second is to receive HTTP requests¹ from the Android application and respond back with the information that is been requested. To do so, the web application must serialize² the information. This is achieved using the RESTful API whose Uniform interface principle benefits the whole system architecture – “By applying the software engineering principle of generality to the component interface, the overall system architecture is simplified, and the visibility of interactions is improved. In order to obtain a uniform interface,

¹ The Hyper Transfer Protocol (HTTP) works as a request - response protocol between a client and server.

² Serialization is a mechanism of converting the state of an object into a byte stream.

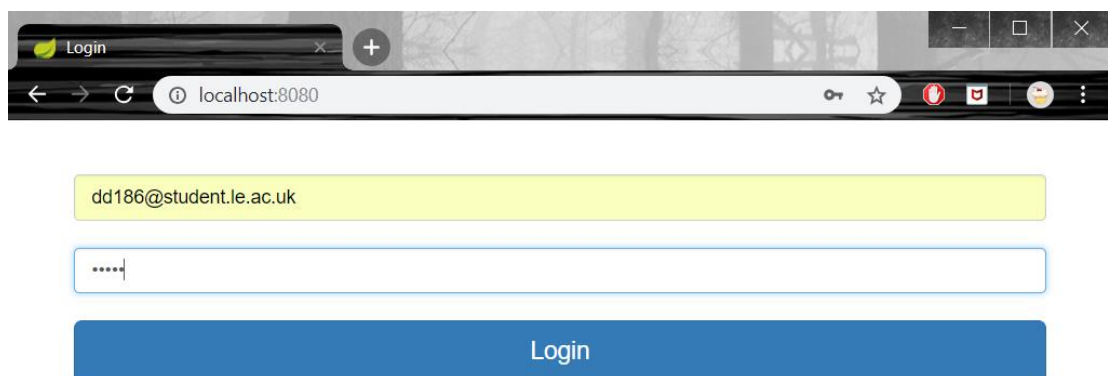
multiple architectural constraints are needed to guide the behaviour of components. “³. Finally, the third purpose of the web application is to give the ability to the admin to handle upcoming orders and keep the users informed. This is achieved by utilizing the Firebase platform. One of the Firebase’s products is Firebase Cloud Messaging which can be used for downstream messaging⁴. Basically, the web application sends a POST HTTP request including the message and the topic that the Firebase will target, every user has a unique topic, and the Firebase does the rest.

5.3 Android Application

The Android application is the main component of the software system and is the only part that interacts directly with the user. It takes its data from the web application by sending the appropriate HTTP requests. When the web application sends back the response to the request, the Android application takes the response, deserializes⁵ and stores the information that is needed. Besides that, the Android application is designed in such a way to provide many abilities to the user such as easy access to the canteen’s products, product offers, categorical deals and fast checkout. Finally, the Android application utilized Firebase to receive notifications.

6 Implementation

6.1 Web Application



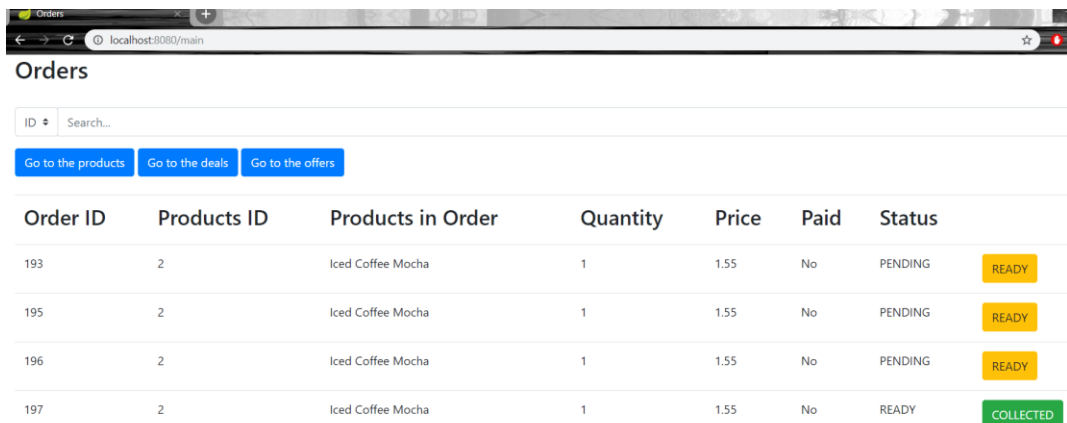
Login page screenshot

The web application has the login page where only the administrators can login in to, using their already provided email and password. (#R047 & #R048)

³ *WHAT IS REST API*. (n.d.). Retrieved from RESTful API Tutorial: <https://restfulapi.net/>

⁴ Downstream Messaging is when you are sending a push notification from your App Server towards the Client Application.

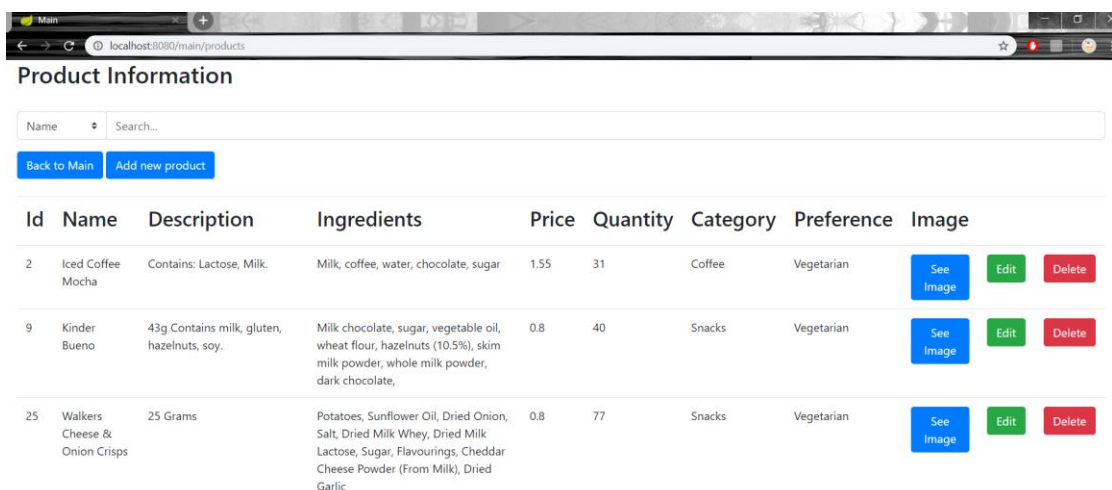
⁵ Deserialization is the reverse process where the byte stream is used to recreate the actual Java object in memory.



Order ID	Products ID	Products in Order	Quantity	Price	Paid	Status	
193	2	Iced Coffee Mocha	1	1.55	No	PENDING	READY
195	2	Iced Coffee Mocha	1	1.55	No	PENDING	READY
196	2	Iced Coffee Mocha	1	1.55	No	PENDING	READY
197	2	Iced Coffee Mocha	1	1.55	No	READY	COLLECTED

Main page screenshot

Then there is the main page where the administrators can see a list of all the orders that are not collected, and which reloads every 5 seconds to keep the table up-to-date. (#R049 & #R053) For every order there is a button that will be used by the admin to change the order status. If an order is currently pending there will be a yellow “READY” button which sends a POST request to the Firebase platform to notify the user that his/her order is ready for collection (See section 11.1) and it must be clicked only if the staff prepared the order for the customer. (#R050) If an order is ready for collection, there will be a green “COLLECTED” button that informs the user that his/her order is collected. The button must be clicked only if a customer has come, shown the right order ID and collected it. (#R051) From the main page the admins are able to navigate through the web application and see the products, the offer and the deals that are in the Android application. (#R052). In the page there is a search input bar where the admins can search for an order by its ID number.

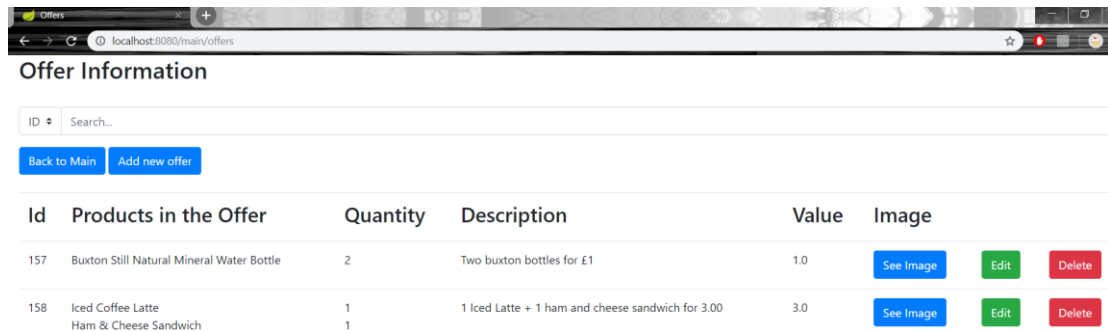


Id	Name	Description	Ingredients	Price	Quantity	Category	Preference	Image
2	Iced Coffee Mocha	Contains: Lactose, Milk.	Milk, coffee, water, chocolate, sugar	1.55	31	Coffee	Vegetarian	See Image Edit Delete
9	Kinder Bueno	43g Contains milk, gluten, hazelnuts, soy.	Milk chocolate, sugar, vegetable oil, wheat flour, hazelnuts (10.5%), skim milk powder, whole milk powder, dark chocolate,	0.8	40	Snacks	Vegetarian	See Image Edit Delete
25	Walkers Cheese & Onion Crisps	25 Grams	Potatoes, Sunflower Oil, Dried Onion, Salt, Dried Milk Whey, Dried Milk Lactose, Sugar, Flavours, Cheddar Cheese Powder (From Milk), Dried Garlic	0.8	77	Snacks	Vegetarian	See Image Edit Delete

Product page screenshot

There is the products’ page where the administrator can see a table of all the products. (#R054) Every product has an “Edit” button which navigates to a page where the admin can see and easily modify the product’s info (#R057), a “Delete” button which deletes the whole product from the database (#R056) and a “See Image” button

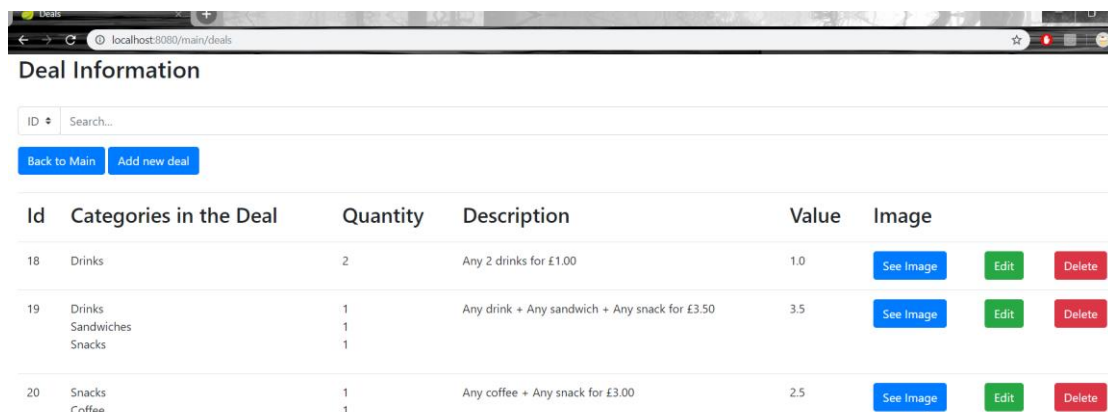
which displays the product's image. There is also an "Add new product" button where the admin can add a new product in the database. (#R055). There is also a search input bar and a search filter so the administrators can find a product through various ways. (See section 11.2)



Id	Products in the Offer	Quantity	Description	Value	Image
157	Buxton Still Natural Mineral Water Bottle	2	Two buxton bottles for £1	1.0	See Image Edit Delete
158	Iced Coffee Latte Ham & Cheese Sandwich	1 1	1 Iced Latte + 1 ham and cheese sandwich for 3.00	3.0	See Image Edit Delete

Offers page screenshot

There is also the offers' page where is a table with all the active offers. (#R058) Like the products' page there are three buttons for every offer a "See image" button that displays the offer's image, an "Edit" button for editing the offer (#R061) and a "Delete" button for deleting the offer from the database. (#R060) There is also a button "Add a new offer" which navigates to the add offer page where the admin can select which products are in the offer (at least 2) from a drop-down list and add the offer's information. (#R059)



Id	Categories in the Deal	Quantity	Description	Value	Image
18	Drinks	2	Any 2 drinks for £1.00	1.0	See Image Edit Delete
19	Drinks Sandwiches Snacks	1 1 1	Any drink + Any sandwich + Any snack for £3.50	3.5	See Image Edit Delete
20	Snacks Coffee	1 1	Any coffee + Any snack for £3.00	2.5	See Image Edit Delete

Deals page screenshot

In addition, there is the deals' page where is a table with all the active deals in the Android application. (#R062) Like products' page and offers' page, for every deal there is a "See image" button displaying the deal's image, a "Delete" button for deleting the deal from the database (#R064) and an "Edit" button for editing it (#R065). Then there is an "Add new deal" button where admins can create a new deal by selecting the categories that are in the deal (at least two) from a drop-down list and add the deal's information. (#R063)



Some screenshots of the JSON responses

The web application is also used to respond to the Android application's HTTP requests. In other words, the Android application will send requests to get the data from the database, for example login and sign up with the user's data and displaying products, deals, offers etc. For instance, the web application will check if the user exists in the database, in case the user wants to login to the Android application and respond back with the user's data if the login process was successful and with "Invalid" otherwise. (#R066) In another case, where the user wants to sign up, the web application will create a new user account and respond with "ok" if the whole sign up process was successful and with "Already existing user" otherwise. (#R067) The Android application needs the products, offers, and deals to load so it sends HTTP requests to get them and the web application converts the requested data to json objects and sends them to the Android application. (See section 11.3) (#R069) If the user add/remove a product in/from his/her favourite list, the Android application sends a request with the necessary information and the web application makes the appropriate changes in the database. (#R068) When the user checkouts from the Android application, it sends a request with the order's information and the web application creates the order, include it in the user's order list and responds back with the order as a json object. (#R070) After checkout the user is able to change the order or delete it, if he/she decides to do so the Android application informs the web one to do the necessary changes. (#R071)

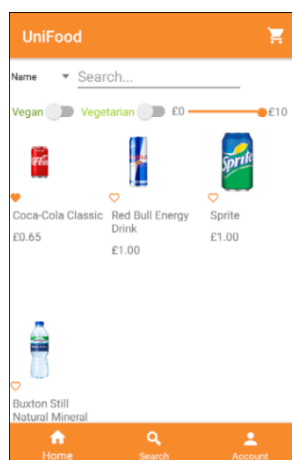
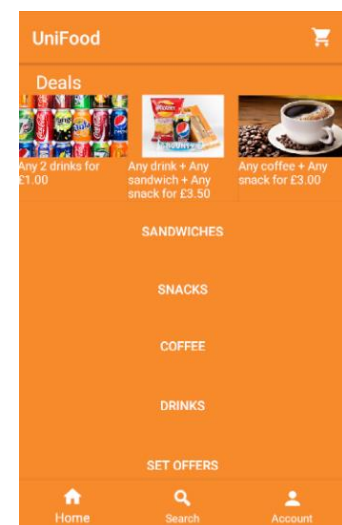
The web application is designed as a responsive application that auto-adjusts according to the device screen size. Furthermore, I decided to use the bootstrap framework to give the application a sleek look and feel.

6.2 Android Application

The Android application has the login page somewhere that both users and administrators can login (#R001) into and the sign-up page where non-registered users can create an account. (#R002) There is “Remember me” check button in the login page and once the user clicks on that button the application will remember his/her email for the next sessions. (#R003) When the user is trying to login or sign up, the application sends a HTTP request and waits for a response. (See section 11.4) When this situation occurs, the application waits for a user’s json object in response for the login and an “ok” response for successful sign up. As soon as Android application gets that response, the main page is displayed for login or the login page is displayed for sign up. If either login or sign up process was unsuccessful the application displays the appropriate error messages. (#R004 & #R005) Non-registered users cannot login until they register.

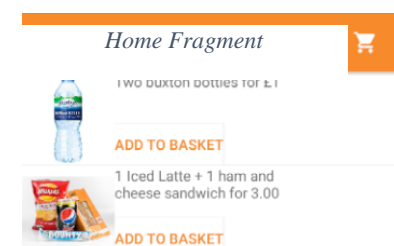
the
Login and Sign up page

After the user is logged in, she/he can see the main page which consists of a bottom-navigating menu for home, search and account. While the main activity is loading the Android application sends requests to the web one to get the products, the deals and the offers from the database. (#R006 & #R007 & #R008) Once the application receives the responses, it deserialized them (See section 11.5) and the home fragment is displayed on the screen. The user is able to refresh the application’s data at any moment in the main activity by swiping down the whole page. (#R009) At the top of the home fragment, there is slide-left list of the current active categorical



“Drinks” category

deals (#R011) and under the list there are five buttons, four for the different product categories, and one for the offers. When the user clicks on one of the categories a list of all the products of that category is displayed on the screen (#R010) and when the user clicks on the “Set offers” button a page containing all the current active offers is displayed. In the offers’ page the user has the option to add the whole offer in the basket. (#R012) If an offer goes out of stock while the user looks at



Offers Fragment

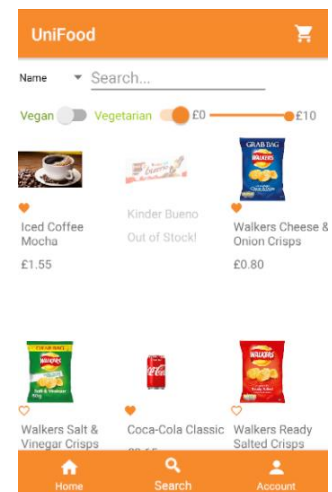
the list, he/she will not be able to add that offer in the basket. (#R013)

Furthermore, there is the search fragment where the user can see all the products in the database in a scroll down list.

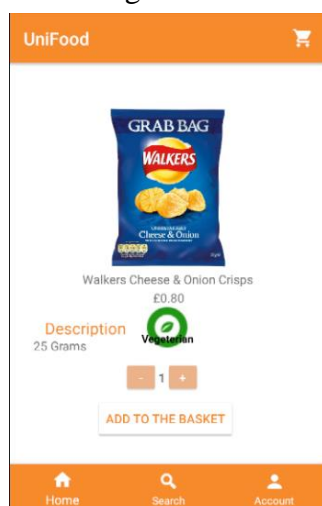
(#R014) If a product goes out of stock while the user looks at the list, he/she will be able to see it but won't be able to add it in the basket. If the user refreshes the search page, he/she will be able to see the products that out of stock. (#R015) At the top of page there is a search input bar and a search filter. The user is able to search through the current displayed products either by their name or ingredients. (#R017 & #R018) He/she can also set dietary constraints such as "Vegan" or

"Vegetarian" by clicking one of them at the top of the page.

(#R019) There is also a seek bar for the cost constrains, which the user can set to the maximum amount of money he/she is willing to pay for a product. (#R020) Moreover, the user is able to combine the search filters to find the right product. (See section 11.6) (#R021) In the search fragment, the user can add or remove products to his/her favourite list by clicking on the small heart under the products. (#R022 & #R023)



Search Fragment

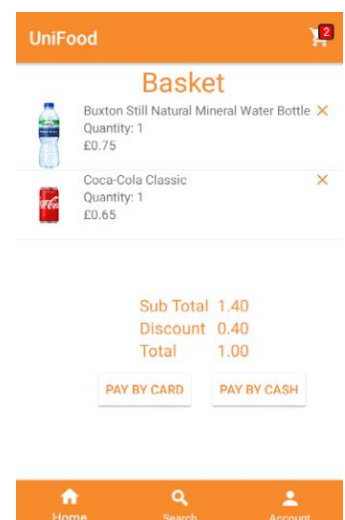


Product Info Fragment

that time, if are any, and their quantity. For every product in the basket page there is a "X" button which can be used to remove that product from the basket. (#R029) Every time the user adds/removes a product to/from the basket, the Android application sends a request to the web application to update the quantity of that product in the database. (#R031) At the bottom of the page, the user can see the sub total of the products, the discount that was applied, if any applies, and the total after the discount. (#R032 & #R033) If the user logs out, the application saves the products that are left in the basket and when the user logs in again, it checks if the products are still in stock and if so, it puts them back in the basket. (#R025 & #R026) Finally, the user can checkout and pay using a card or select the cash option and pay at the collection point. (#R034 & #R035) When an order is

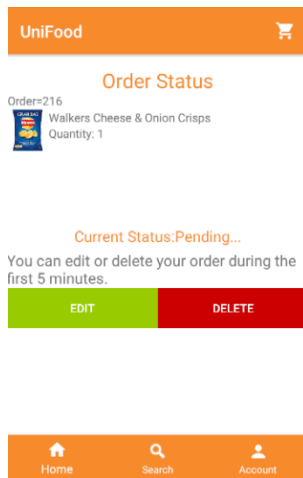
Finally, when the user clicks on one of the products, the product's information page is displayed on the screen. The product fragment contains all the details about the product such as name, price, description etc. (#R024) At the bottom of the page there is there are three buttons, two for increasing or decreasing the quantity of the product that the user wants to put in the basket and the "Add to the basket" button which adds the product to the basket. (#R027 & #R028). If the user is seeing a product and that product becomes out of stock, then he/she won't be able to add it in the basket. (#R030)

In addition, there is the basket page where the user can see the products that are in his/her basket at that time, if are any, and their quantity. For every product in the basket page there is a "X" button which can be used to remove that product



Basket Fragment

made, the application subscribes to his/her unique topic, the one that the Firebase will target. After that the Android application sends a request every 10 seconds to check the order status until the order is collected. (#R042) (See section 11.7)

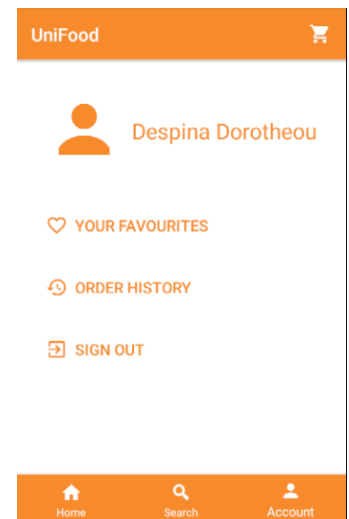


Order Status Fragment

Therefore, there is the order status page where the user can see his/her order's status, the unique order ID number and a message saying that he/she is able to modify or delete his/her order in the first 5 minutes, after 5 minutes that message disappears. (See section 11.8) (#R039 & #R040 & #R045) If the user decides to click on the "Edit" button then all the products that was in the order are back in the basket and he/she able to add more, or even remove some product. While the user is editing his/her order, the application sends a request to the web application with the order number to inform the admin that the order is being edited by the user.

(#R041) The user must confirm the changes by clicking the "Confirm" button in the basket page. If the user decides to delete the order, he/she must click on the "Delete" button and the Android application sends a request to the web application to delete the order. After checkout, if the order is ready for collection, the user will be notified by a notification with his/her order number and then he/she be notified again when the order is collected. (#R043 & #R044)

Finally, there is the account page where the user can see his/her favourite products, his/her order history and log out. (#R036 & #R037 & #R046) There is, in addition, a hidden option, order status, which is visible only when the user made an order and he/she is waiting for updates. (#R038)



Account Fragment

7 Testing

7.1 Web Application

7.1.1 Repository Layer

Test Name	Description	Status
testFindByCategory	<ul style="list-style-type: none"> ○ Given a string category name. ○ When categoryRepository.findByCategory(categoryName). ○ Return the category with that name. ○ Assert category object is not null. 	[PASSED]
testSaveCategory	<ul style="list-style-type: none"> ○ Given a category object. ○ When categoryRepository.save(category). ○ Then save the category in the database. 	[PASSED]
testDeleteCategory	<ul style="list-style-type: none"> ○ Given a category object. ○ When categoryRepository.delete(category). ○ Then delete the category from the database 	[PASSED]
testFindDealById	<ul style="list-style-type: none"> ○ Given an int ID. ○ When dealRepository.findById(ID). ○ Return the deal with that id. ○ Assert deal object is not null. 	[PASSED]
testSaveDeal	<ul style="list-style-type: none"> ○ Given a deal object. ○ When dealRepository.save(deal). ○ Then save the deal in the database. 	[PASSED]
testDeleteDeal	<ul style="list-style-type: none"> ○ Given a deal object. ○ When dealRepository.delete(deal). ○ Then delete the deal from the database 	[PASSED]
testFindOfferById	<ul style="list-style-type: none"> ○ Given an int ID. ○ When offerRepository.findById(ID). ○ Return the offer with that id. ○ Assert offer object is not null. 	[PASSED]
testSaveOffer	<ul style="list-style-type: none"> ○ Given an offer object. ○ When offerRepository.save(offer). ○ Then save the offer in the database. 	[PASSED]
testDeleteOffer	<ul style="list-style-type: none"> ○ Given an offer object. ○ When offerRepository.delete(offer). ○ Then delete the offer from the database 	[PASSED]
testFindOrderById	<ul style="list-style-type: none"> ○ Given an int ID. ○ When orderRepository.findById(ID). ○ Return the order with that id. ○ Assert order object is not null. 	[PASSED]
testSaveOrder	<ul style="list-style-type: none"> ○ Given an order object. ○ When orderRepository.save(order). ○ Then save the order in the database. 	[PASSED]
testDeleteOrder	<ul style="list-style-type: none"> ○ Given an order object. ○ When orderRepository.delete(order). ○ Then delete the order from the database 	[PASSED]

testFindProductById	<ul style="list-style-type: none"> ○ Given an int ID. ○ When productRepository.findById(ID). ○ Return the product with that id. ○ Assert product object is not null. 	[PASSED]
testSaveProduct	<ul style="list-style-type: none"> ○ Given a product object. ○ When productRepository.save(product). ○ Then save the product in the database. 	[PASSED]
testDeleteProduct	<ul style="list-style-type: none"> ○ Given a product object. ○ When productRepository.delete(product). ○ Then delete the product from the database 	[PASSED]
testFindRoleByRoleName	<ul style="list-style-type: none"> ○ Given a string role name. ○ When roleRepository.findByRole(roleName). ○ Return the role with that name. ○ Assert role object is not null. 	[PASSED]
testSaveRole	<ul style="list-style-type: none"> ○ Given a role object. ○ When roleRepository.save(role). ○ Then save the role in the database. 	[PASSED]
testDeleteRole	<ul style="list-style-type: none"> ○ Given a role object. ○ When roleRepository.delete(role). ○ Then delete the role from the database 	[PASSED]
testFindUserById	<ul style="list-style-type: none"> ○ Given an int ID. ○ When userRepository.findById(ID). ○ Return the user with that id. ○ Assert user object is not null. 	[PASSED]
testFindUserByEmail	<ul style="list-style-type: none"> ○ Given a string email. ○ When userRepository.findByEmail(email). ○ Return the email with that name. ○ Assert user object is not null. 	[PASSED]
testSaveUser	<ul style="list-style-type: none"> ○ Given a user object. ○ When userRepository.save(user). ○ Then save the user in the database. 	[PASSED]
testDeleteUser	<ul style="list-style-type: none"> ○ Given a user object. ○ When userRepository.delete(user). ○ Then delete the user from the database 	[PASSED]

7.1.2 Service Layer

Test Name	Description	Status
testFindDealById	<ul style="list-style-type: none"> ○ Given an int ID. ○ When dealService.findById(ID). ○ Return the deal with that name. ○ Assert deal object is not null. 	[PASSED]
testSaveDeal	<ul style="list-style-type: none"> ○ Given a deal object. ○ When dealService.save(deal). ○ Then save the deal in the database. 	[PASSED]
testDeleteDeal	<ul style="list-style-type: none"> ○ Given a deal object. ○ When dealService.delete(deal). ○ Then delete the deal from the database. 	[PASSED]
testFindAllDeals	<ul style="list-style-type: none"> ○ When dealService.findAll(). 	[PASSED]

	<ul style="list-style-type: none"> ○ Return a list of all the deals in the database. 	
testFindOfferById	<ul style="list-style-type: none"> ○ Given an int ID. ○ When offerService.findById(ID). ○ Return the offer with that name. ○ Assert offer object is not null. 	[PASSED]
testSaveOffer	<ul style="list-style-type: none"> ○ Given an offer object. ○ When offerService.save(deal). ○ Then save the offer in the database. 	[PASSED]
testDeleteOffer	<ul style="list-style-type: none"> ○ Given an offer object. ○ When offerService.delete(offer). ○ Then delete the offer from the database. 	[PASSED]
testFindAllOffers	<ul style="list-style-type: none"> ○ When offerService.findAll(). ○ Return a list of all the offers in the database. 	[PASSED]
testFindOrderById	<ul style="list-style-type: none"> ○ Given an int ID. ○ When orderService.findById(ID). ○ Return the order with that id. ○ Assert order object is not null. 	[PASSED]
testSaveOrder	<ul style="list-style-type: none"> ○ Given an order object. ○ When orderService.save(order). ○ Then save the order in the database. 	[PASSED]
testDeleteOrder	<ul style="list-style-type: none"> ○ Given an order object. ○ When orderService.delete(order). ○ Then delete the order from the database 	[PASSED]
testFindAllOrders	<ul style="list-style-type: none"> ○ When orderService.getAll(). ○ Return a list of all the orders in the database. 	[PASSED]
testFindProductById	<ul style="list-style-type: none"> ○ Given an int ID. ○ When productService.findById(ID). ○ Return the product with that id. ○ Assert product object is not null. 	[PASSED]
testSaveProduct	<ul style="list-style-type: none"> ○ Given a product object. ○ When productService.save(product). ○ Then save the product in the database. 	[PASSED]
testDeleteProduct	<ul style="list-style-type: none"> ○ Given a product object. ○ When productService.delete(product). ○ Then delete the product from the database 	[PASSED]
testFindAllProducts	<ul style="list-style-type: none"> ○ When productService.findAll(). ○ Return a list of all the products in the database. 	[PASSED]
testFindCategoryByName	<ul style="list-style-type: none"> ○ Given a string category name. ○ When productService.findByCategory(categoryName). ○ Return the category with that name. ○ Assert category object is not null. 	[PASSED]
testFindAllCategories	<ul style="list-style-type: none"> ○ When productService.findAllCategories(). ○ Return a list of all the categories in the database. 	[PASSED]
testFindUserById	<ul style="list-style-type: none"> ○ Given an int ID. ○ When userService.findById(ID). ○ Return the user with that id. ○ Assert user object is not null. 	[PASSED]
testFindUserByEmail	<ul style="list-style-type: none"> ○ Given a string email. 	[PASSED]

	<ul style="list-style-type: none"> ○ When userService.findUserByEmail(email). ○ Return the email with that name. ○ Assert user object is not null. 	
testSaveUser	<ul style="list-style-type: none"> ○ Given a user object. ○ When userService.saveUser(user). ○ Then save the user in the database. 	[PASSED]
testSaveUserCustom	<ul style="list-style-type: none"> ○ Given a user object. ○ When userRepository.saveUserCustom(user). ○ Then save the user in the database. 	[PASSED]
testMatchPass	<ul style="list-style-type: none"> ○ Given an encrypted string and one that is not. ○ When userService.passMatch(raw, encrypted) ○ Return true if are the same string and false otherwise. 	[PASSED]

7.1.3 Controller Layer

Test Name	Description	Status
testEditDeal	<ul style="list-style-type: none"> ○ When perform GET request ○ With url “/main/deals/edit” ○ With param name “dealId” and value any int ○ Expect status 200 ○ Expect view with name “editDeal” 	[PASSED]
testAddNewDeal	<ul style="list-style-type: none"> ○ When perform MULTIPART request ○ With url “/main/deals/add” ○ With param name “id” and value -1 ○ With param name “category1” and value any int ○ With param name “category2” and value any int ○ With param name “description” and value any string ○ With param name “value” and value any double ○ Expect status 302 ○ Expect redirected url “/main/deals” 	[PASSED]
testAddDeal	<ul style="list-style-type: none"> ○ When perform MULTIPART request ○ With url “/main/deals/add” ○ With param name “id” and value any int ○ With param name “category1” and value any string ○ With param name “category2” and value any string ○ With param name “description” and value any string ○ With param name “value” and value any double ○ Expect status 302 ○ Expect redirected url “/main/deals” 	[PASSED]
testDealImage	<ul style="list-style-type: none"> ○ When perform GET request ○ With url “/main/deals/image” ○ With param name “dealId” and value any int ○ With content type “IMAGE_JPEG_VALUE ○ Expect status 200 	[PASSED]
testDeleteDeal	<ul style="list-style-type: none"> ○ When perform GET request ○ With url “/main/deals/delete” 	[PASSED]

	<ul style="list-style-type: none"> ○ With param name “dealId” and value any int ○ Expect status 302 ○ Expect redirected url “/main/deals” 	
testDirectToLoginPage	<ul style="list-style-type: none"> ○ When perform GET request ○ With url “/login” ○ Expect status 200 ○ Expect view with name “login” 	[PASSED]
testDirectToLoginPage2	<ul style="list-style-type: none"> ○ When perform GET request ○ With url “/” ○ Expect status 200 ○ Expect view with name “login” 	[PASSED]
testDirectToMainPage	<ul style="list-style-type: none"> ○ When perform GET request ○ With url “/main” ○ Expect status 200 ○ Expect model with attribute “orders” and value any list of orders. ○ Expect view with name “ordersPage” 	[PASSED]
testEditOffer	<ul style="list-style-type: none"> ○ When perform GET request ○ With url “/main/offers/edit” ○ With param name “offerId” and value any int ○ Expect status 200 ○ Expect view with name “editOffer” 	[PASSED]
testAddNewOffer	<ul style="list-style-type: none"> ○ When perform MULTIPART request ○ With url “/main/offers/add” ○ With param name “id” and value -1 ○ With param name “product1” and value any int ○ With param name “product2” and value any int ○ With param name “description” and value any string ○ With param name “value” and value any double ○ Expect status 302 ○ Expect redirected url “/main/offers” 	[PASSED]
testAddOffer	<ul style="list-style-type: none"> ○ When perform MULTIPART request ○ With url “/main/offers/add” ○ With param name “id” and value any int ○ With param name “product1” and value any int ○ With param name “product2” and value any int ○ With param name “description” and value any string ○ With param name “value” and value any double ○ Expect status 302 ○ Expect redirected url “/main/offers” 	[PASSED]
testOfferImage	<ul style="list-style-type: none"> ○ When perform GET request ○ With url “/main/offers/image” ○ With param name “offerId” and value any int ○ With content type “IMAGE_JPEG_VALUE” ○ Expect status 200 	[PASSED]
testDeleteOffer	<ul style="list-style-type: none"> ○ When perform GET request ○ With url “/main/offers/delete” ○ With param name “offerId” and value any int 	[PASSED]

	<ul style="list-style-type: none"> ○ Expect status 302 ○ Expect redirected url “/main/offers” 	
testOrderReady	<ul style="list-style-type: none"> ○ When perform GET request ○ With url “/main/order/ready” ○ With param name “orderId” and value any int ○ Expect status 302 ○ Expect redirected url “/main” 	[PASSED]
testOrderCollected	<ul style="list-style-type: none"> ○ When perform GET request ○ With url “/main/order/collected” ○ With param name “orderId” and value any int ○ Expect status 302 ○ Expect redirected url “/main” 	[PASSED]
testDirectDealsPage	<ul style="list-style-type: none"> ○ When perform GET request ○ With url “/main/deals” ○ Expect status 200 ○ Expect model with attribute “deals” and value any list of deals. ○ Expect view with name “dealsPage” 	[PASSED]
testDirectOffersPage	<ul style="list-style-type: none"> ○ When perform GET request ○ With url “/main/offers” ○ Expect status 200 ○ Expect model with attribute “offers” and value any list of deals. ○ Expect view with name “offersPage” 	[PASSED]
testDirectProductPage	<ul style="list-style-type: none"> ○ When perform GET request ○ With url “/main/products” ○ Expect status 200 ○ Expect model with attribute “products” and value any list of deals. ○ Expect view with name “productsPage” 	[PASSED]
testEditProduct	<ul style="list-style-type: none"> ○ When perform GET request ○ With url “/main/products/edit” ○ With param name “offerId” and value any int ○ Expect status 200 ○ Expect view with name “editProduct” 	[PASSED]
testAddNewProduct	<ul style="list-style-type: none"> ○ When perform MULTIPART request ○ With url “/main/products/add” ○ With param name “id” and value -1 ○ With param name “name” and value any string ○ With param name “description” and value any string ○ With param name “ingredients” and value any string ○ With param name “price” and value any double ○ With param name “quantity” and value any int ○ With param name “category” and value any string ○ With param name “preference” and value any string ○ Expect status 302 ○ Expect redirected url “/main/products” 	[PASSED]
testAddProduct	<ul style="list-style-type: none"> ○ When perform MULTIPART request 	[PASSED]

	<ul style="list-style-type: none"> ○ With url “/main/products/add” ○ With param name “id” and value any int ○ With param name “name” and value any string ○ With param name “description” and value any string ○ With param name “ingredients” and value any string ○ With param name “price” and value any double ○ With param name “quantity” and value any int ○ With param name “category” and value any string ○ With param name “preference” and value any string ○ Expect status 302 ○ Expect redirected url “/main/products” 	
testProductImage	<ul style="list-style-type: none"> ○ When perform GET request ○ With url “/main/deals/image” ○ With param name “productId” and value any int ○ With content type “IMAGE_JPEG_VALUE” ○ Expect status 200 	[PASSED]
testDeleteProduct	<ul style="list-style-type: none"> ○ When perform GET request ○ With url “/main/products/delete” ○ With param name “productId” and value any int ○ Expect status 302 ○ Expect redirected url “/main/products” 	[PASSED]

7.2 Android Application

7.2.1 Signup Activity

Test Name	Description	Status
backToMain	<ul style="list-style-type: none"> ○ When the “back” button is clicked ○ Verify a new activity type “Login.class” has started 	[PASSED]
createAccount	<ul style="list-style-type: none"> ○ Given “Email” is an email string ○ and “Name” is a name string ○ and “Last Name” is a last name string ○ and “Password” is a password string ○ and “Password 2” is the same string as “Password” ○ When “Sign up” button is clicked ○ Verify a new activity type “Login.class” has started 	[PASSED]
failedCreateAccount1	<ul style="list-style-type: none"> ○ Given “Email” is an invalid email string ○ and “Name” is a name string ○ and “Last Name” is a last name string ○ and “Password” is a password string ○ and “Password 2” is the same string as “Password” ○ When “Sign up” button is clicked 	[PASSED]

	<ul style="list-style-type: none"> ○ Verify the error message of “Email” is “Invalid email address” 	
failedCreateAccount2	<ul style="list-style-type: none"> ○ Given “Email” is an email string ○ and “Name” is an invalid name string ○ and “Last Name” is a last name string ○ and “Password” is a password string ○ and “Password 2” is the same string as “Password” ○ When “Sign up” button is clicked ○ Verify the error message of “Name” is “Invalid name!” 	[PASSED]
failedCreateAccount3	<ul style="list-style-type: none"> ○ Given “Email” is an email string ○ and “Name” is a name string ○ and “Last Name” is an invalid last name string ○ and “Password” is a password string ○ and “Password 2” is the same string as “Password” ○ When “Sign up” button is clicked ○ Verify the error message of “Last Name” is “Invalid last name!” 	[PASSED]
failedCreateAccount4	<ul style="list-style-type: none"> ○ Given “Email” is an email string ○ and “Name” is a name string ○ and “Last Name” is a last name string ○ and “Password” is a password string ○ and “Password 2” is not the same string as “Password” ○ When “Sign up” button is clicked ○ Verify the error message of “Error” textView is “Passwords do not match!” 	[PASSED]
failedCreateAccount5	<ul style="list-style-type: none"> ○ Given all fields are empty ○ When “Sign up” button is clicked ○ Verify that all fields have error message “is required!” 	[PASSED]

7.2.2 Login Activity

Test Name	Description	Status
invalidUser	<ul style="list-style-type: none"> ○ Given the user entered invalid credentials ○ When “Login” button is clicked ○ Verify error’s text is “Invalid Email or Password!” 	[PASSED]
invalidUser1	<ul style="list-style-type: none"> ○ Given all fields are empty ○ When “Login” button is clicked ○ Verify all fields have an error message “is required!” 	[PASSED]
validUser	<ul style="list-style-type: none"> ○ Given the user entered correct credentials ○ When “Login” button is clicked ○ Verify a new activity type “Main.class” has started 	[PASSED]
signUpPage	<ul style="list-style-type: none"> ○ When “Sign up” button is clicked ○ Verify a new activity type “Signup.class” has started 	[PASSED]

7.2.3 Main Activity

Test Name	Description	Status
searchFragment	<ul style="list-style-type: none"> ○ When the user clicks on the search button on the bottom navigation ○ Verify search fragment is displayed on the screen 	[PASSED]
homeFragment	<ul style="list-style-type: none"> ○ When the user clicks on the home button on the bottom navigation ○ Verify home fragment is displayed on the screen 	[PASSED]
accountFragment	<ul style="list-style-type: none"> ○ When the user clicks on the account button on the bottom navigation ○ Verify account fragment is displayed on the screen 	[PASSED]
testSetBasketBadgeNum	<ul style="list-style-type: none"> ○ When main.setBasketBadgeNum(int) ○ Verify the badge textView's text is that int 	[PASSED]
basketFragment	<ul style="list-style-type: none"> ○ When the user clicks on the basket icon ○ Verify basket fragment is displayed on the screen 	[PASSED]
basketFragment1	<ul style="list-style-type: none"> ○ Given empty basket list ○ When the user clicks on the basket icon ○ Verify basket fragment is displayed on the screen 	[PASSED]
testExtractProductsFromJson	<ul style="list-style-type: none"> ○ When main.extractProductsFromJson(correctJson) ○ Return the list of products 	[PASSED]
testFailedExtractProductsFromJson	<ul style="list-style-type: none"> ○ When main.extractProductsFromJson(wrongJson) ○ Return null 	[PASSED]
testOrderStatusPage	<ul style="list-style-type: none"> ○ When the user clicks on the "Order Status" button in the account fragment ○ Verify order status fragment is displayed on the screen 	[PASSED]
testOrderHistoryPage	<ul style="list-style-type: none"> ○ When the user clicks on the "Order History" button in the account fragment ○ Verify order history fragment is displayed on the screen 	[PASSED]
testFavouritePage	<ul style="list-style-type: none"> ○ When the user clicks on the "Favourites" button in the account fragment ○ Verify favourites fragment is displayed on the screen 	[PASSED]
testSignoutBtn	<ul style="list-style-type: none"> ○ When the user clicks on the "Sign out" button in the account fragment ○ Verify a new activity type "Login.class" has started 	[PASSED]
testSandwichCategory	<ul style="list-style-type: none"> ○ When the user clicks on the "Sandwiches" button in the home fragment ○ Verify search fragment is displayed on the screen 	[PASSED]

testSnacksCategory	<ul style="list-style-type: none"> ○ When the user clicks on the “Snacks” button in the home fragment ○ Verify search fragment is displayed on the screen 	[PASSED]
testCoffeeCategory	<ul style="list-style-type: none"> ○ When the user clicks on the “Coffee” button in the home fragment ○ Verify search fragment is displayed on the screen 	[PASSED]
testDrinksCategory	<ul style="list-style-type: none"> ○ When the user clicks on the “Drinks” button in the home fragment ○ Verify search fragment is displayed on the screen 	[PASSED]
testSetOfferBtn	<ul style="list-style-type: none"> ○ When the user clicks on the “Coffee” button in the home fragment ○ Verify offers fragment is displayed on the screen 	[PASSED]
testAddOffersToTheBasket	<ul style="list-style-type: none"> ○ Given an offer ○ When main.addOfferToBasket(offer) ○ Verify the offer is in the basket list 	[PASSED]
testRemoveFromTheBasket	<ul style="list-style-type: none"> ○ Given a product and the basketNumTextView ○ When main.removeFromBasket (product, text view) ○ Verify product is not in the basket list 	[PASSED]
testEditOrder	<ul style="list-style-type: none"> ○ When the user clicks on the “Edit” button in the order status fragment ○ Verify basket fragment is displayed on the screen and the order’s products are back in the basket list. 	[PASSED]
testDeleteOrder	<ul style="list-style-type: none"> ○ When the user clicks on the “Delete” button in the order status fragment ○ Verify home fragment is displayed on the screen and the pending order is null 	[PASSED]
testFromMapToList	<ul style="list-style-type: none"> ○ Given a map ○ When main.fromMapToList(map) ○ Return the product list 	[PASSED]
testAfterPlaceOrder	<ul style="list-style-type: none"> ○ When main.afterPlaceOrder() ○ Verify the basket list is null, the editing Boolean is false, and the order status fragment is displayed on the screen 	[PASSED]
testCountBasketBadge	<ul style="list-style-type: none"> ○ When main.countBasketBadge() ○ Verify the basket badge textView’s is correct 	[PASSED]
testFindTotal	<ul style="list-style-type: none"> ○ Given a basket list ○ When basketFragment.findTotal() ○ Verify the total based on the products in the basket is correct 	[PASSED]
testFindDiscount	<ul style="list-style-type: none"> ○ Given a basket ○ When basketFragment.findDiscount() 	[PASSED]

	<ul style="list-style-type: none"> ○ Verify the discount based on the products in the basket is correct 	
testFindDiscount1	<ul style="list-style-type: none"> ○ Given another basket ○ When basketFragment.findDiscount() ○ Verify the discount based on the products in the basket is correct 	[PASSED]
testPayByCardBtn	<ul style="list-style-type: none"> ○ Given the basket list is not empty ○ When the user clicks on the “Pay by card” button in the basket fragment ○ Verify the card info fragment is displayed on the screen 	[PASSED]
testPayByCashBtn	<ul style="list-style-type: none"> ○ Given the basket list is not empty ○ When the user clicks on the “Pay by cash” button in the basket fragment ○ Verify the order status fragment is displayed on the screen 	[PASSED]
testConfirmChanges	<ul style="list-style-type: none"> ○ Given the user is editing his/her order ○ When the user clicks on the “confirm” button in the basket fragment ○ Verify the order status fragment is displayed on the screen 	[PASSED]
testProductInfo	<ul style="list-style-type: none"> ○ When the user clicks on an “item” of the list of products in the basket fragment ○ Verify the product info fragment is displayed on the screen. 	[PASSED]
testConfirmDetailsBtn	<ul style="list-style-type: none"> ○ When the user clicks on the “Confirm” button in the card info fragment ○ Verify the order status fragment is displayed on the screen. 	[PASSED]
testVeganSearch	<ul style="list-style-type: none"> ○ When the user clicks on the “Vegan” button in the search fragment ○ Verify that the gridView has only vegan products 	[PASSED]
testVegetarianSearch	<ul style="list-style-type: none"> ○ When the user clicks on the “Vegan” button in the search fragment ○ Verify that the gridView has only vegetarian products 	[PASSED]
testSearchByName	<ul style="list-style-type: none"> ○ When the user types “milk” in the search input ○ Verify that the gridView has only products with the substring “milk” in their name 	[PASSED]
testSearchByName2	<ul style="list-style-type: none"> ○ When the user types “water” in the search input ○ Verify that the gridView has only products with the substring “water” in their name 	[PASSED]
testSearchByIngredients	<ul style="list-style-type: none"> ○ When the user types “lactose” in the search input and the search filter is ingredients ○ Verify that the gridView has only products with the substring “lactose” in their ingredients. 	[PASSED]

testCostFilter	<ul style="list-style-type: none">○ When the user set the cost seek bar to 1○ Verify that the gridView contains only products that their price is equal or less than £1	[PASSED]
testAddToTheBasket	<ul style="list-style-type: none">○ When the user clicks on the “Add to the basket” button in the product info fragment○ Verify the home fragment is displayed on the screen and the product is in the basket list	[PASSED]
testIncreaseQuantity	<ul style="list-style-type: none">○ When the user clicks on the “+” button in the product info fragment○ Verify that the quantity displayed on the screen is increased by one	[PASSED]
testDecreaseQuantity	<ul style="list-style-type: none">○ When the user clicks on the “-” button in the product info fragment○ Verify that the quantity displayed on the screen is decreased by one	[PASSED]

8 Critical Appraisal

8.1 Critical Analysis

The main aim of the project was to develop a user-friendly and helpful Android application of a canteen ordering system and an easy to use web application.

At the time, there were many project’s parts that I wish I could do different. For instance, I wish I could predict, some of the unexpected challenges that I faced, from the start and make a more relative project plan. There were many times, that I was struggling to keep up with the initial plan. Most of the time, I was behind the plan because of the challenges.

Due to the fact that this was the first time that I developed an Android application and furthermore it was the first time I developed something this big by myself, it was very difficult for me to estimate the time that every part of the project was going to take. Yet I believe, from the experience that I gained from this dissertation, that I will be able to plan my next projects better.

Furthermore, I decided to focus mainly on the Android application’s functionalities, and I believe that if I had more time, I could focus more on some other project’s aspects such as design and testing, which were very important too.

Moreover, due to the lack of time at the end of the project, I couldn’t use an external API for the “pay by card” functionality and that’s why I decided to simulate the whole process than using an API that was not working properly.

Overall, I have implemented as many essential, recommended and optional requirements of the project as possible and I believe the result was better than I expected. Besides my wrong time estimations, I believe the project’s results served the original aims.

8.2 Personal Development

The research I done during this project has refined my knowledge regarding Android applications and their development. During this time, I also realised that Android development is something that I am very interested at.

While I was working on the project, I have considerably improved my problem-solving skills as well as my knowledge regarding the Java programming language. The project was quite challenging, mainly, because I had to develop two applications, hence, I had to work and learn many different technologies at the same time.

Moreover, during the project, I had the opportunity to be involved with the whole testing process for the first time and due to the fact that I had to test two applications this has helped me understand the testing's importance in both web and Android development.

Furthermore, I improved my time management skills since it was a project which had to be completed on framework. The numerous expected or even unexpected challenges, that I faced during that time, made me force myself to solve them as fast and as efficient as possible. I believe all these skills that I improved and all these new technologies that I learned are necessary for my own future career in the Computer Science industry.

9 Conclusion

Ordering System applications, in general, are very useful and necessary nowadays and this is the main reason why I chose this project.

To conclude, the dissertation's purpose was to develop an Android application of a canteen ordering system. My goal was not only to develop the application but also to make it easy-to-use, user-friendly and with many useful functionalities. Due to the fact that the application targets mostly university students, I believe that I developed it in such a way to be as useful as possible. The result was pretty good, but I was disappointed that I did not organised my time better since the application had prospects to become even better. The lack of time at the end of the project, also, prevented me from fully test some functionalities.

Overall, regarding the project's requirements, all were satisfactorily completed, and at the end, I even added some more to make the application almost ready for the market.

10 References

- [1] *Create an Android project*. Available at: <https://developer.android.com/training/basics/firstapp/creating-project> (Accessed: Dec 5, 2018).
- [2] *Upay*. Available at: <https://www.upay.co.uk/>
- [3] *Uber Eats*. Available at: <https://www.ubereats.com/en-GB/>
- [4] *First Spring Boot App tutorial*. Available at: <https://app.pluralsight.com/player?course=spring-boot-first-application&author=dan-bunker&name=spring-boot-first-application-m1&clip=0&mode=live>
- [5] *WHAT IS REST API*. (n.d.). Retrieved from RESTful API Tutorial: <https://restfulapi.net/>
- [6] *The course-type tutorial for HTTP requests*. Available at: <https://classroom.udacity.com/courses/ud843/lessons/ca00b576-ec4d-4d0f-939e-6d28f55faf7b/concepts/2914a2ea-7f15-4915-8113-d0be6e023526>
- [7] <http://appsdeveloperblog.com/java-into-json-json-into-java-all-possible-examples/>
- [8] *Binary Large Object*. Available at: <https://dev.mysql.com/doc/refman/8.0/en/blob.html>
- [9] *Base64 tutorial and information*. Available at: <https://www.baeldung.com/java-base64-encode-and-decode>
- [10] *Firebase Cloud Messaging*. Available at: <https://firebase.google.com/docs/cloud-messaging>
- [11] *Set up a Firebase Cloud Messaging client app on Android*. Available at: <https://firebase.google.com/docs/cloud-messaging/android/client?authuser=0>
- [12] *Spring Boot Testing*. Available at: <https://www.baeldung.com/spring-boot-testing>
- [13] *Mockito Framework*. Available at: <https://www.baeldung.com/mockito-annotations>
- [14] *Robolectric Framework*. Available at: <http://robolectric.org/>
- [15] *Black Box Testing*. Available at: <http://softwaretestingfundamentals.com/black-box-testing/>

11 Appendix

11.1 POST request to Firebase Platform

```
private void sendNotification(String msg, int topic) throws JSONException {
    JSONObject body = new JSONObject();
    body.put( "to", " /topics/" + topic);
    body.put( "priority", "high");

    JSONObject notification = new JSONObject();
    notification.put( "title", "UniFood");
    notification.put( "body", msg );

    body.put( "notification", notification);

    /*
    {
      "notification": {
        "title": "UniFood",
        "body": "Happy Message!"
      },
      "to": "/topics/userID",
      "priority": "high"
    }
    */

    HttpEntity<String> request = new HttpEntity<>(body.toString());
    CompletableFuture<String> pushNotification = androidPushNotificationsService.send(request);
    CompletableFuture.allOf(pushNotification).join();

    try {
        String firebaseResponse = pushNotification.get();
        System.out.println(new ResponseEntity<>(firebaseResponse, HttpStatus.OK));
    } catch (InterruptedException | ExecutionException e) {
        System.out.println(new ResponseEntity<>( body: "Push Notification ERROR!", HttpStatus.BAD_REQUEST));
        e.printStackTrace();
    }
}
```

Screenshot from OrderController.java

11.2 JavaScript Search function

```
<script type="text/javascript">
    function searchFunction() {
        var searchKeyWord = document.getElementById("searchValue");
        var filter= searchKeyWord.value.toUpperCase();
        var table = document.getElementById("table");
        var tr = table.getElementsByTagName("tr");

        for ( var i=0; i<tr.length; i++){
            var td = tr[i].getElementsByTagName("td")[ $("#select").val()];
            if (td){
                var columnVal = td.textContent || td.innerHTML;
                if(columnVal.toUpperCase().indexOf(filter)>-1){
                    tr[i].style.display= "";
                } else {
                    tr[i].style.display= "none";
                }
            }
        }
    }
}
</script>
```

Screenshot from productsPage.jsp

11.3 Java objects to JSON objects

```
//method for creating Json Array of the products for the android app
private JSONArray createProductList(List<Product> products) throws SQLException {
    JSONArray result = new JSONArray();
    if (!products.isEmpty()) {
        for (Product p : products) {
            JSONObject product = new JSONObject();
            product.addProperty( property: "id", p.getId());
            product.addProperty( property: "name", p.getName());
            product.addProperty( property: "description", p.getDescription());
            product.addProperty( property: "price", p.getPrice());
            product.addProperty( property: "quantity", p.getQuantity());
            if (p.getImage() != null) {
                product.addProperty( property: "image", Base64.encodeBase64String(p.getImage().getBytes( POS: 1, (int) p.getImage().length())));
            }
            product.addProperty( property: "preference", p.getPreference());
            product.addProperty( property: "ingredients", p.getIngredients());
            JSONObject category = new JSONObject();
            category.addProperty( property: "id", p.getCategory().getId());
            category.addProperty( property: "category", p.getCategory().getCategory());
            product.add( property: "category", category);
            result.add(product);
        }
    }
    return result;
}
```

Screenshot from RestController.java

11.4 HTTP GET request

```
public String makeHttpRequest(URL url) throws IOException {
    String jsonResponse = "";
    HttpURLConnection httpURLConnection = null;
    InputStream inputStream = null;
    try{
        httpURLConnection = (HttpURLConnection) url.openConnection();
        httpURLConnection.setRequestMethod("GET");
        httpURLConnection.setReadTimeout(1000);
        httpURLConnection.setConnectTimeout(15000);
        httpURLConnection.connect();
        //if the request was successful(response code 200),
        //then read the input stream and parse the response.
        if(httpURLConnection.getResponseCode() == 200){
            inputStream = httpURLConnection.getInputStream();
            jsonResponse = readFromStream(inputStream);
        }
    }catch (IOException e){
        e.printStackTrace();
    } finally {
        if (httpURLConnection!=null){
            httpURLConnection.disconnect();
        }
        if (inputStream != null){
            inputStream.close();
        }
    }
    return jsonResponse;
}
```

Screenshot from HttpGetRequest.java (AsyncTask)

11.5 Deserialization

```
//method used to extract the products for the given json string
public List<Product> extractProductsFromJson(String productString) {
    //if the json string is empty or null, the return early.
    ObjectMapper mapper = new ObjectMapper();
    if (TextUtils.isEmpty(productString)) {
        return null;
    }
    try {
        List<Product> products = new ArrayList<>();
        products = mapper.readValue(productString, new TypeReference<List<Product>>() {
        });
        return products;
    } catch (Exception e) {
        System.out.println("Something wrong with the deserialisation of products ");
        e.printStackTrace();
        return null;
    }
}
```

Screenshot from Main.java

11.6 Vegan/Vegetarian Filters

```
Switch vegan = rootView.findViewById(R.id.vegan_switch);
Switch vegetarian = rootView.findViewById(R.id.vegetarian_switch);
vegan.setOnCheckedChangeListener((buttonView, isChecked) -> {
    //list used to store the products that was displayed on the screen before seekbar changes
    List<Product> result = new ArrayList<>();
    if (isChecked) {
        veganFilterApplied = true;
        for (Product p : searchProducts) {
            if (p.getPreference().equals("Vegan") && p.getPrice() <= maxFilter) {
                result.add(p);
            }
        }
        vegetarian.setClickable(false);
    } else {
        veganFilterApplied = false;
        for (Product p : searchProducts) {
            if (p.getPrice() <= maxFilter) {
                result.add(p);
            }
        }
        vegetarian.setClickable(true);
    }
    currentDisplayedProducts = result;
    gridView.setAdapter(new ProductAdapter(main, result));
});

vegetarian.setOnCheckedChangeListener((buttonView, isChecked) -> {
    //list used to store the products that was displayed on the screen before seekbar changes
    List<Product> result = new ArrayList<>();
    if (isChecked) {
        vegetarianFilterApplied = true;
        for (Product p : searchProducts) {
            if ((p.getPreference().equals("Vegetarian") || p.getPreference().equals("Vegan")) && p.getPrice() <= maxFilter) {
                result.add(p);
            }
        }
        vegan.setClickable(false);
    } else {
        vegetarianFilterApplied = false;
        vegan.setClickable(true);
        for (Product p : searchProducts) {
            if (p.getPrice() <= maxFilter) {
                result.add(p);
            }
        }
    }
    currentDisplayedProducts = result;
    gridView.setAdapter(new ProductAdapter(main, result));
});
```

Screenshot from SearchFragment.java method onCreateView(...)

11.7 Check order status

```
public void checkStatus(){
    t = new Timer( false);
    t.schedule(() -> {
        main.runOnUiThread() -> {
            OrderStatusFragment.currentStatus = Main.makeARequest.get("http://10.0.2.2:8080/rest/checkStatus/" +Main.pendingOrder.getId());
            main.setTextStatus(currentStatus);
            if (currentStatus.equals("Ready for collection!")){
                OrderStatusFragment.isVisible = View.GONE;
                main.setGoneEditDelete(); // (or GONE)
            }
            if (currentStatus.equals("Collected!")){
                t.cancel();
                //5 mins
                CountdownTimer timer = new CountdownTimer( millisInFuture: 300000, countDownInterval: 1000) {
                    @Override
                    public void onTick(long millisUntilFinished) {
                    }
                    @Override
                    public void onFinish() {
                        Main.pendingOrder = null;
                        Fragment fragment = new HomeFragment();
                        main.loadFragment(fragment, tag: "home");
                    }
                }.start();
            }
        }
    }, delay: 0, period: 10000);
}
```

Screenshot from OrderStatusFragment.java

11.8 Edit/Delete Order

```
public void setTimer(LinearLayout edit_delete){
    //5 mins
    CountdownTimer timer = new CountdownTimer( millisInFuture: 300000, countDownInterval: 1000) {
        @Override
        public void onTick(long millisUntilFinished) {
        }
        @Override
        public void onFinish() {
            OrderStatusFragment.isVisible = View.GONE;
            edit_delete.setVisibility(View.GONE); // (or GONE)
        }
    }.start();
}
```

Screenshot from OrderStatusFragment.java