TECHNICAL REPORT

BSHC4IOT

Leon Mulvaney
14445618
BSc (Hons) in Computing – IoT Stream
X14445618@student.ncirl.ie
May 2018

Declaration Cover Sheet for Project Submission

SECTION 1 *Student to complete*

Name: Leon Mulvaney
Student ID: 14445618
Supervisor: Glen Ward

SECTION 2 Confirmation of Authorship

The acceptance of your work is subject to your signature on the following declaration:

I confirm that I have read the College statement on plagiarism (summarised overleaf and printed in full in the Student Handbook) and that the work I have submitted for assessment is entirely my own work.

Signature:  Date: **13/05/2018**

Table of Contents

Table of figures

Submission Links

Android Application Github Repo

https://github.com/LeonMulvaney/TheIntelligentFoodNetwork

Python Code Github Repo

https://github.com/LeonMulvaney/TheIntelligentFoodNetworkPython

*(The Python Github Upload also includes the "**SimpleMFRC522**" library file)*

N.B Application Code is also uploaded into the "Project Code" Moodle submission.

Glossary of Terms

Term	Description
API	Application Programming Interface
GUI	Graphical User Interface
GHz	Gigahertz
IoT	Internet of Things
NFC	Near Field Communication
Pi	Raspberry Pi Microcontroller
RFID	Radio Frequency Identification
UI	User Interface
XML	extensible Markup Language

Executive Summary

This report will outline the fundamental aspects of the final year Software Project. The project idea is to design and develop a “Smart Food Network”. The system is called “The Intelligent Food Network”. This Network embeds itself into the background of people’s everyday lives, building upon a major current trend – The Internet of Things (IoT).

This idea utilizes current RFID/NFC standards and makes use of an RFID Controller (Reader & Writer) – tracking food items being placed into and removed from a refrigerator or cabinet in a user’s home. A proprietary mobile application has been developed in accordance with the system. All that’s required for this solution to function is a Raspberry Pi, RFID Controller and a Mobile Device - with the latter being something that almost all of us already possess. The Android Application has been developed to provide the entry point for the end-user, allowing them to manage the whole system remotely - at the touch of a button.

The system incorporates a wide range of technologies and features and has been developed with a strong emphasis on usability & simplicity. An array of software components have been utilized to their maximum potential. The Intelligent Food Network ultimately provides a one for all solution to a common household problem. All fundamental elements of the system seamlessly interact and work together to provide the best possible user experience.

This report will outline the fundamental concepts of the idea, the primary technologies that have been implemented and how the entire system has been crafted from initial requirements to the final product.

1 Introduction

1.1 Background

At present, there is quite a lack of commercially available solutions to the idea. Yes, there have been some recent developments in terms of “Smart or Internet Fridges”, but these come alongside a hefty price-premium – some products exceeding €5,000. For the vast majority of people, this price premium is simply not financially viable. This idea will prove to be a much more cost-effective and financially viable solution to the commercially available products currently on the market.

The inspiration to undertake such a project came from my interest in both computing and electronics. I have also always been fascinated with the recently developing trends - especially the Internet of Things. When I saw the release of Samsung’s Smart Fridge, I was very interested. The fact that a major player in the electronics industry was focusing on IOT got me thinking. Then, I seen the price of these appliances – averaging €2500-€6000, I knew they would be expensive – but not this expensive. I believed there had to be a way to implement the same concepts, but at a much lower cost. There are also currently a plethora of Food Applications and Websites scattered across the net - although these are primarily based on user input. Some of these can recommend recipes – albeit being dependent on a user to log each food item – this is in no way as integrated and seamless as the proposed solution.

The project will utilize current technologies and trends throughout the development process. It will consist of A Mobile application coupled with a Server-Side back-end developed using scripting languages. Many forms of Hardware will also be used i.e. a Raspberry Pi, RFID Controller and RFID tags. I will also ideally make use of other Raspberry Pi sensors to increase usability of the system.

The project will interest a variety of customers, although the primary user-base will include Men and Women between the ages of 20 -50. There are many reasons for this - This age group is relatively young and is familiar with current technology, as a result they will be more open to using this new system. Many individuals which fall within this category may be homeowners, parents or students living on their own – as such they may live hectic lives. My system aims to alleviate some of the stresses of daily life i.e. What meals to prepare, Time Constraints, Lack of ingredients and reminding busy individuals not to forget to restock invaluable food items such as Milk or Bread etc.

1.2 Scope/Aims

The scope of this project is to create an Intelligent System based on food products within the home. The system will work seamlessly and not interfere with daily life. It will embed itself into the background, whilst allowing users to take full advantage of its features. The project will be based on the currently trending area of IoT and make use of various programming languages and technologies to achieve its aims. The system will utilize many hardware and software platforms. One of the fundamental aims is to create a competent and user-friendly system. The project will make efficient use of the project plan as described in the previous document and will be completed within the required time frame, to the highest possible standard.

Initially, RFID tags will be attached to food items – acting as the primary transmission medium between the Tag and Controller. The RFID reader will be attached to the Raspberry Pi, and Python Script will constantly loop – scanning for food items. Once the food items are passed into the Fridge or Cabinet, the reader will grab the corresponding data. The Raspberry Pi will then establish a connection with Firebase to push the data. Furthermore, the system will track the removal of food items using the same RFID methodologies as listed above.

An Android Application will be developed through Android Studio and will act as the intermediary between the user and the back-end system. Throughout development the system will follow current trends and development guidelines to produce a professional, industry-standard package.

The system will make use of Food and Recipe API's such as Spoonacular and Nutritionix. Using this information, the system will can provide recipe recommendations and a plethora of other useful features. Some of the features that will be included within the Application;

- The ability to check what food items are currently present within the home (Live Data)
- What food items need purchasing during the next shopping trip
- A Recipe recommendation for meals – based on a user's Favourite Recipes. (Implementation Spoonacular Recipe API)
- Access to thousands of Recipes, Ingredients and other food data via the Spoonacular API
- Data Analysis on what foods a user has in their home and shopping list to evaluate what needs to be purchased next trip
- Customisation of User Account (Profile Image, Weight etc.)
- Favourite Recipes Section
- Nutritional Data for any type of food imaginable (Implementation of Nutritionix API)

The system will make use of testing platforms to ensure all code and functionality is error-free.

Once all primary aspects of the system are fully functioning and operational, the aim is to move onto additional features to increase overall usability and user interaction with the system.

Finally - once error free and debugged, the system will be ideally deployed (Android Application) onto the Google Play Store where it may be downloaded by the targeted user base.

1.3 Technologies

A variety of technologies will be utilized to achieve the aims of the project. Throughout development, I will make use of my current knowledge whilst also aiming to implement new technologies to develop my knowledge even further. Below is a listing of the primary technologies that will be used throughout the development lifecycle.

1.3.1 Development

Python/Raspbian – Raspberry Pi

The primary device used for communication between the food items and my system is an RFID controller. The RFID controller will be connected to the Raspberry Pi. Raspbian, a Linux distribution or "distro" will be installed onto the Pi. This Operating System will enable development on the Pi headless and negate the need for a monitor, mouse and keyboard. The Raspbian Operating System will also enable control the RFID controller. A remote access viewer titled "VNC Viewer", will be used to gain remote access into the Pi and run commands through either a terminal or Command Line Interface.

Python is the primary programming language used on the Raspberry Pi. Python is a very powerful language and integrates well with Firebase and Android. Python is a scripting language, as such it will be used to develop all scripts on the Raspberry Pi i.e.

- Reading/Writing via the RFID Controller
- Establishing a connection with my Back-End(Firebase)
- Receiving data from Firebase
- Parsing Data to JSON
- Pushing Data up to Firebase Database

JSON

JSON or JavaScript Object Notation is a data exchange language. It is easily readable by both humans and computers and can be decoded by almost any programming language. JSON is based on Key-Value pairs. To send data from a Raspberry Pi to the back-end, data will first have to be parsed into JSON. Only then can it be transmitted. The Firebase database is also based around JSON - utilizing Key-Value pairs and Parent-Child relationships for data storage and retrieval. Some of the fundamental concepts within the system will require use of the JSON language.

Firebase

Firebase is a development platform which can be used for both mobile and web applications. It is developed by Google and incorporates a multitude of useful features including Authentication, Data Storage, Real-time Database & Crash Reporting. The system will make efficient use of the Firebase platform – primarily for its Authentication, Storage & Real-time Database functionalities. All data collected via RFID will be updated and posted to the Firebase platform. Firebase will also be used to analyse interaction – providing logs of any potential or catastrophic errors. The Firebase Database is built around JSON notation – this will allow for easy data transmission between the systems Python Scripts and the real-time database.

Android

One of the primary elements of the project will be an Android application. Android Studio will be the platform of choice for development - primarily because it is an IDE specifically targeted at Android Development. Android Studio is a very competent package which also includes an embedded Firebase assistant – a very useful feature. Java and XML will be used within the IDE for development of functionality and GUI. The Android Application will act as the intermediary between end users and the back-end system. The mobile platform will be capable of pulling data from Firebase and manipulating it to meet users demands. It will also make use of the Spoonacular and Nutritionix API's. The application is the only element of the system which the end user will see – as such it must feature an elegant and user-friendly GUI. It will also build upon current mobile design trends – specifically incorporating Googles Material design ques.

API's

The system will make efficient use of many API's – primarily API's found within the Rapid API platform. One of the primary food-based API'S that will be used is Spoonacular. Spoonacular is an online food-oriented platform which enables users to view recipes, ingredients, food products and much more. Developers are provided access to this huge data-set via the Spoonacular API. This API is very competent, and a large amount of quality documentation can be found online. This API will be heavily utilized within the project as it provides access to a massive database of information and will fulfil the needs of The Intelligent Food Network. This API will work well with my application – for example recipes will be pulled from the Spoonacular API based on the food contents within a user's home. User's will then be able to view, share and favourite recipes.

GitHub

GitHub will provide the primary means of version control for the system. All development resources including, scripts, files, layouts, designs and classes will be pushed to GitHub once created or altered. GitHub will assist development and hold the necessary system version back-ups if a failure occurs throughout development.

1.3.2 Testing

Internal & External Methodologies

Both Internal and External testing methodologies will be utilized to ensure the system is reliable and meets the requirements of the target user base. Testing examples include Testing on Multiple Devices, Testing the Security of the Platform and various forms of User Testing. Any findings or recommendations derived from the testing process will be implemented before the final product is deployed.

1.3.3 Deployment

Google Play Store

Once the system has been tested and is fully complete, the Android Application will be deployed onto the Google Play Store. This will allow any users whom adopt the system to download the corresponding application from a reliable source. The application will be available for not only Android devices, but also Google Chromebooks. (Google Chromebooks now support all Google Play Store Android Applications).

2 User Requirements Definition

2.1 Survey Overview

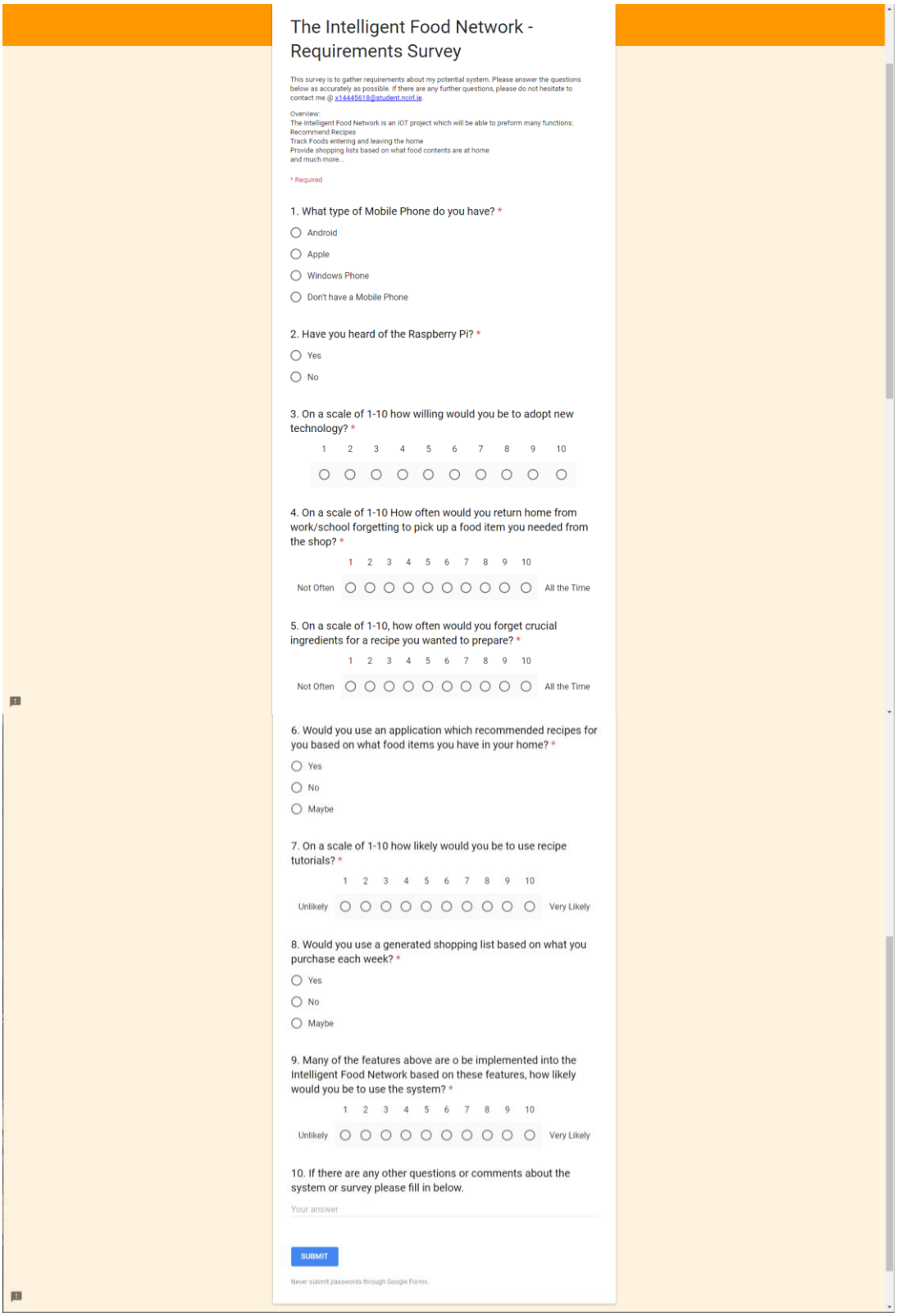
Customers play a crucial role in the success of any system. Without customers and a strong potential user base, any system can effectively be rendered useless. The Intelligent Food network is a relatively niche product. As such, attaining an idea of whether the potential solution would be able to hold its targeted consumer base was vital. To get the best possible understanding of what the customers wanted from the system, a survey was sent out.

Google Forms was used to create this survey. Google Forms is a program which is free to use and comes alongside the Google Docs suite of applications. Google Forms was chosen because it features an intuitive and user-friendly interface for creating and designing surveys & forms. It is also capable of generating tables, charts and graphs based on user responses - which would be crucial when preforming data analysis. Forms are saved instantly each time an alteration or addition is made, and the sharing capabilities are extremely straightforward. User results are returned instantly and the whole package negates the need for a thread of emails going back and forth.

The survey created was sent to a variety of potential users - these included work colleagues, neighbours and other students within the College. It was decided not to put too many questions on the survey – some potential customers may have lost attention and not finished – therefore 10 questions were chosen. When designing the survey, Single Choice or “Tick the Box” questions were primarily selected. From personal experience, these styles of answers provide much more focused results and keep the user interested as opposed to general statements which can sometimes return convoluted answers. One exception to this was question 10 where the interviewee could fill in any additional comments or questions.

Below is the survey that was created. The first question was based around what mobile platform users were using. This was a vital question as the initial application will only target Android devices. Other specific questions associated with the potential functional requirements i.e. different features and sections of the Application were also asked. These questions were varied ultimately provided a strong insight on what user requirements must be prioritized during development.

2.2 Survey Questions



The Intelligent Food Network - Requirements Survey

This survey is to gather requirements about my potential system. Please answer the questions below as accurately as possible. If there are any further questions, please do not hesitate to contact me @ x14445618@student.ncirl.ie.

Overview:
The Intelligent Food Network is an IOT project which will be able to preform many functions:
Recommend Recipes
Track Foods entering and leaving the home
Provide shopping lists based on what food contents are at home
and much more...

* Required

1. What type of Mobile Phone do you have? *
- Android
- Apple
- Windows Phone
- Don't have a Mobile Phone

2. Have you heard of the Raspberry Pi? *
- Yes
- No

3. On a scale of 1-10 how willing would you be to adopt new technology? *

1 2 3 4 5 6 7 8 9 10

4. On a scale of 1-10 How often would you return home from work/school forgetting to pick up a food item you needed from the shop? *

1 2 3 4 5 6 7 8 9 10
Not Often ... All the Time

5. On a scale of 1-10, how often would you forget crucial ingredients for a recipe you wanted to prepare? *

1 2 3 4 5 6 7 8 9 10
Not Often ... All the Time

6. Would you use an application which recommended recipes for you based on what food items you have in your home? *
- Yes
- No
- Maybe

7. On a scale of 1-10 how likely would you be to use recipe tutorials? *

1 2 3 4 5 6 7 8 9 10
Unlikely ... Very Likely

8. Would you use a generated shopping list based on what you purchase each week? *
- Yes
- No
- Maybe

9. Many of the features above are o be implemented into the Intelligent Food Network based on these features, how likely would you be to use the system? *

1 2 3 4 5 6 7 8 9 10
Unlikely ... Very Likely

10. If there are any other questions or comments about the system or survey please fill in below.

Your answer

SUBMIT

Never submit passwords through Google Forms.

Figure 1 - Requirement Survey

2.3 Survey Results

After sending the survey out to the potential user base, 15 responses in total were returned. Although this was not a huge number - it will provide sufficient grounding for further development. The responses were mostly positive - with most users having Android mobile phones – 73.3% to be exact. This brought the survey off to a great start as one of the major initial system requirements was an Android Mobile device.

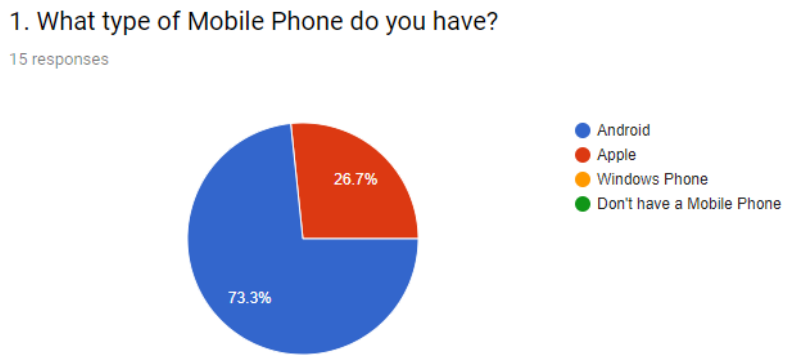


Figure 2 - Survey Result 1

The next question was rather varied, some people had heard of the Raspberry Pi, whilst others had not. The percentage people that had heard of the Pi was 66.7%. I believe the other individuals whom had not heard of the Raspberry Pi were work colleagues and neighbours - with the almost 70% being students and fellow IT oriented individuals. Although not required, it was a bonus if the surveyed knew what the Raspberry Pi was.

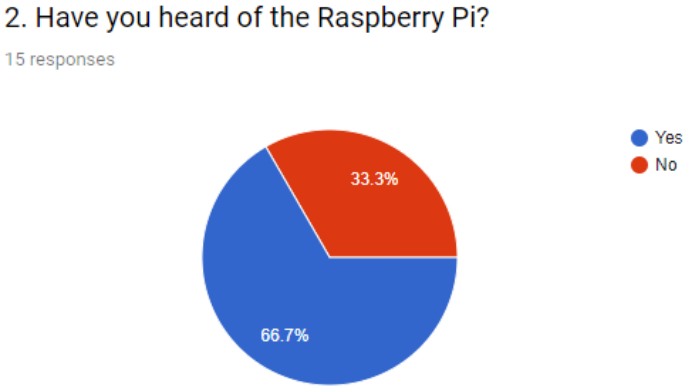


Figure 3 - Survey Result 2

The internet of things is a taking off at an exponential rate - albeit still being a new technology. The Intelligent Food Network is quite a niche system. It is different from many systems because it not only incorporates software (App) but features various hardware components. To fully utilize the system, users must be willing to adopt new technologies such as the Raspberry Pi and associated sensors. The results returned were exactly what was desired– most of the potential candidates fell within the category of "Likely – Highly Likely" to to adopt new technology. (1 = Not Likely, 10 = Highly lIkely). This was a major concern when conducting initial analysis as most people are reluctant to change – especially regarding the IT industry. The overall results were satisfactory.

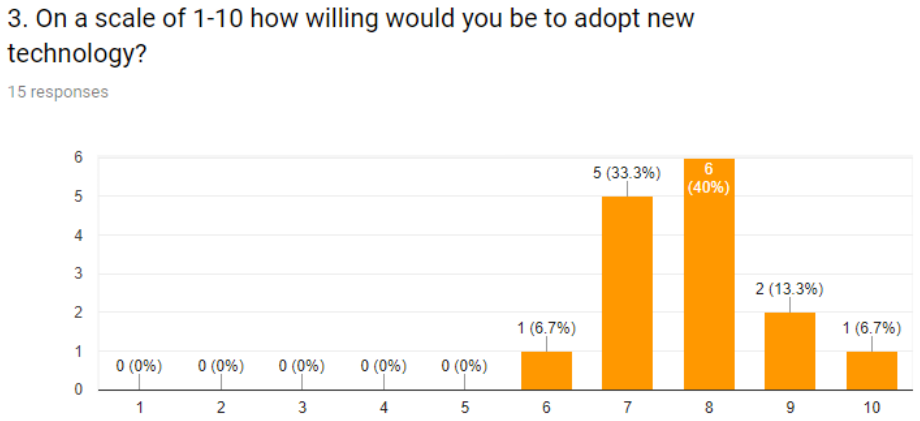


Figure 4 - Survey Result 3

There were many key features which were planned during the pre-development stage, these included recipe recommendations. Some questions were based around these primary features. Based on the results, the system would suit the users by solving an everyday problem. Question 4 results were varied. Users were asked how often they would return home forgetting to pick items up from the shop. Most people - 11/15 to be precise, said they would be likely to highly likely to forget items on their way home.

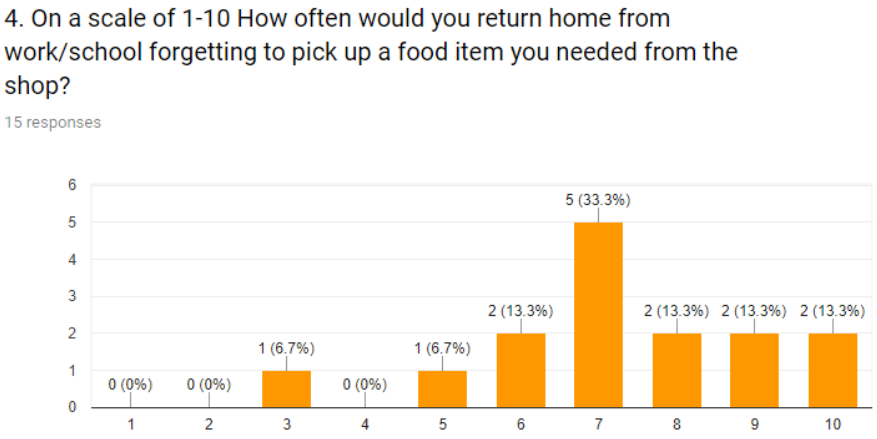


Figure 5 - Survey Result 4

When asked on a scale of 1-10, a large majority of users would forget to pick up crucial ingredients – as such my system would aim to alleviate some of these issues by providing a shopping list recommendation based on the contents within the home. Users would be able to then look at the application to see what they require rather than forgetting whilst in the supermarket.

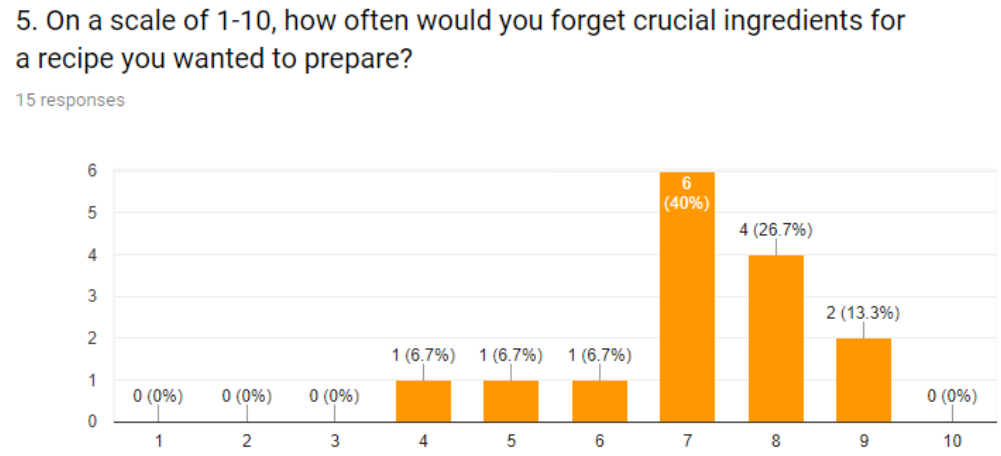


Figure 6 - Survey Result 5

Question 6 and Question 8 saw extremely positive feedback. 100 % of people said that they would user the recipe recommendation section and 100% of people also said they would use a generated shopping list. This is a huge plus for the project - people would be willing to adopt this system based on two of its primary features. 100% on two of the primary use cases was a significant feat and will definitely impact the Functional Requirements Section.

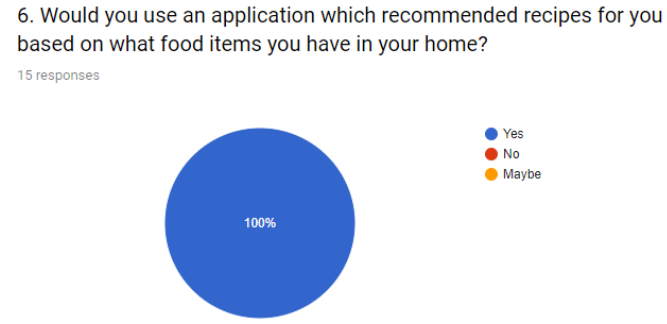


Figure 7 - Survey Result 6

Figure 8 - Survey Result 7

Question 9 of the survey summarises by asking people if they would like to use an application which encompassed all the features described in the earlier sections of the survey. All surveyed individuals clicked 7/10 or above for this question which meant that they were likely-highly likely to use the system with all the initial features. This was once again another positive result for the system – ultimately, the system would appeal to my target user base.

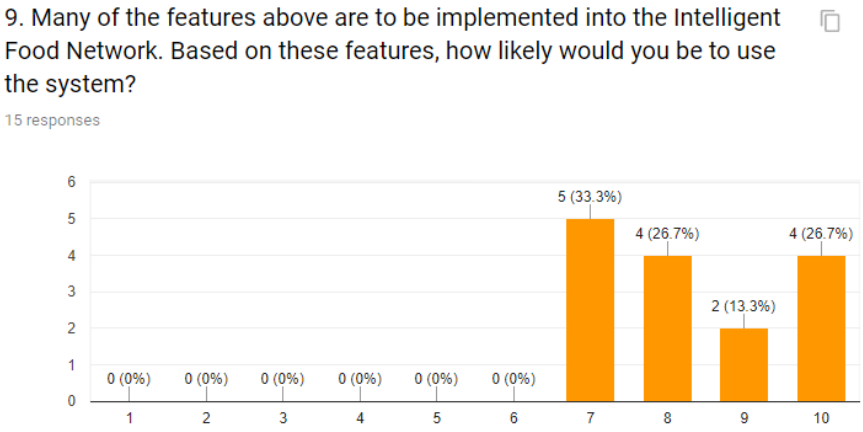


Figure 9 - Survey Result 8

The survey results will prove to be extremely beneficial during the further requirements and development stages. Many of the questions associated with the primary features and use cases returned positive results. The overall results were satisfactory. One of the most important aspects of creating a successful system is to work with users, therefore - many game developers for example provide Alpha and Beta iterations of games. The main goal of any system is to appease the user, what do they want and how can it be achieved. I was happy with the overall results and knew that the system now had good potential. The next phase was to work on the functional requirements. The final project will ultimately be derived from these crucial survey results.

3 Requirements Specification

3.1 Functional Requirements

The functional requirements below have been derived from the User Requirements survey.

3.1.1 System Requirements (Use Case Diagram)

Below is the System Use Case Diagram. The System Use case diagram provides an overview of the entire system. Each use case is described in detail in the following sections.

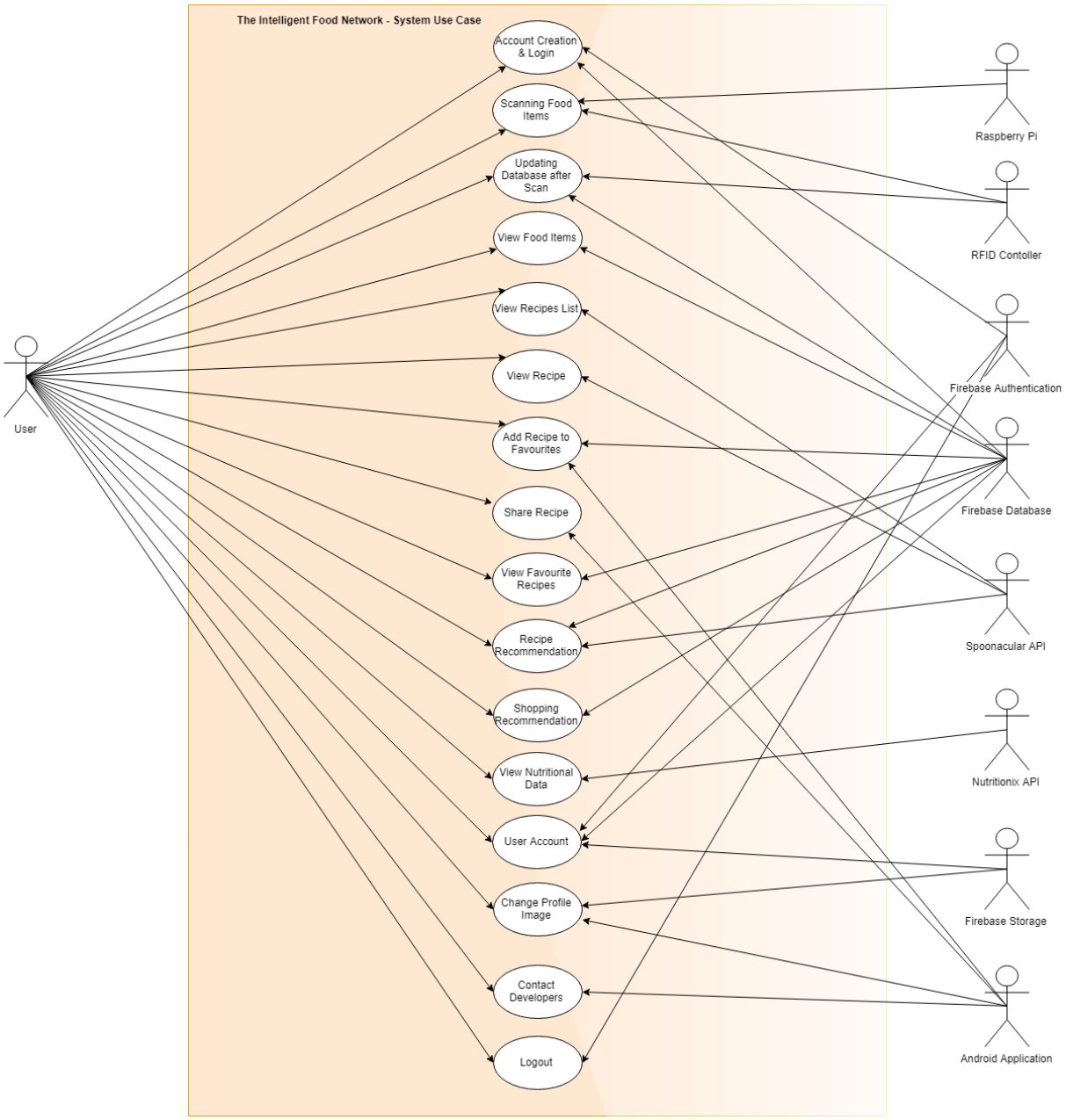


Figure 10 - System Use Case Diagram

3.1.2 Requirement Number 1 <Account Creation & Login>

Overview & Priority

Users can create an account within the Android Application. Doing so will enable them to take advantage of all the aspects of the mobile platform. None of the server-side functionality can be accessed if the user does not hold an account.

On start-up, the application will prompt the user to either login or create a new account. Specific credentials such as email and password etc. will be required for both login and sign up. Once the user submits their credentials - the Application will establish a connection with Firebase. Firebase Authentication will be used to manage the user accounts. If an account is not already in the system, a new one will be created. If an account is already present within the system, and user attempts to log in with the incorrect credentials - they will be presented with an error message.

Priority: High

Use Case

Scope

The Scope of this use case is to retrieve user credentials from the Application and allow them to login or create an account.

Description

This Use Case describes the Account Creation & Login process. The User will be prompted to enter their credentials when the Application starts. Based on the user input parameters, an account will be created and then logged in or logged in immediately (Providing correct credentials are input). Firebase Authentication will provide the back-end functionality for this use case.

Use Case Diagram:

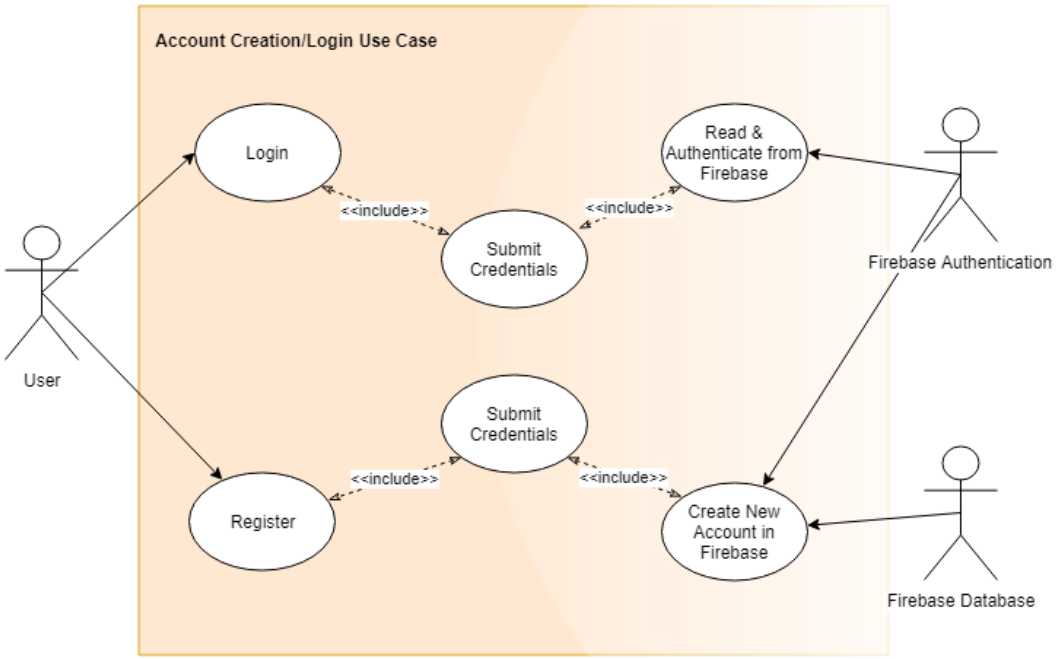


Figure 11 - Requirement 1 Use Case Diagram

Precondition
The User must have an active internet connection and have the application installed.

Activation
The system starts when the user initially opens the application.

Main Flow
1. The User Submits their credentials [A1]
2. The System establishes a connection with Firebase
3. The System Reads & Authenticates the credentials
4. The User successfully logs in to the application

Alternate Flow 1
1. The User navigates to the Registration page
2. The User submits their new credentials & account details
3. The System establishes a connecting with Firebase
4. The System takes in the new credentials
5. The credentials are added to the Firebase Authentication branch
6. A new table is created in Firebase database for the User
7. Return to Main Flow [3]

Exceptional Flow
N/A

Termination
The system terminates when the user logs in successfully or closes the application.

Post Condition
The User is directed to the Home Page and system now provides the user with access to all its features.

//----------End of Use Case ----------//

3.1.3 Requirement Number 2 <Scanning for Food Items>

Overview & Priority

The RFID controller acts as the intermediary between the Food Items and the Raspberry Pi. This controller must be connected to allow the system to update which items are entering and leaving a user's Refrigerator or Cabinet. It will receive the information written to the RFID tags and then send this information to the Raspberry Pi. The Server-Side and Mobile Application will not be able to function properly if this use case is not fulfilled.

This use case will be carried out seamlessly within the background. The user will simply have to place a food product into their Refrigerator or Cabinet for the RFID Controller to receive the data. Once received, the data will then be passed onto the Raspberry Pi before it is updated and pushed to the Database.

Priority: High

Use Case

Scope

The scope of this use case is for the RFID Controller to receive the information from the RFID tags, then pass this information onto the Raspberry Pi.

Description

This use case describes the reading process by which the RFID controller receives data when a food item is either placed or removed from a user's Refrigerator or Cabinet. The received data is then passed onto the Raspberry Pi.

Use Case Diagram:

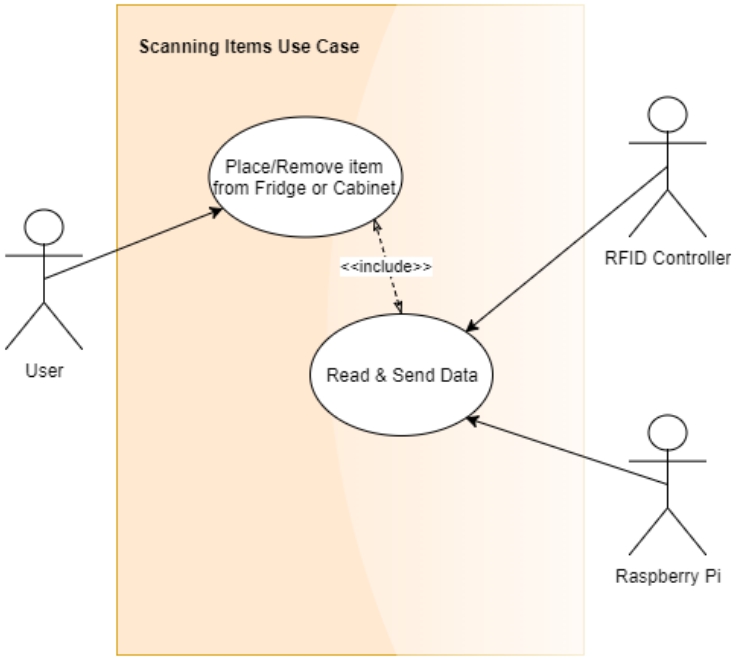


Figure 12 - Requirement 2 Use Case Diagram

Precondition
The Raspberry Pi must be powered on and the RFID reader ready to scan for items.

Activation
The system starts when a user places or removes an item from their Refrigerator or Cabinet.

Main Flow

1. The User places an item into the Fridge or Cabinet
2. The RFID Controller reads the Data
3. The RFID Controller sends the data to the Raspberry Pi

Alternate Flow
N/A

Exceptional Flow
N/A

Termination
The system terminates when data is read and sent onto the Raspberry Pi.

Post Condition
Once the data has been read and sent onto the Raspberry Pi, the system waits and scans for new items to read.

//----------End of Use Case ----------//

3.1.4 Requirement Number 3 <Updating Database after Scan>

Overview & Priority

The Raspberry Pi is an invaluable part of the system. It is a major requirement – providing the connection between the Firebase Database whilst also receiving the scanned information from RFID Controller via Python.

This requirement is an extension of Requirement 2 and once again the User is only required to pass a food item by the RFID Controller. Once the Controller receives the information from the RFID tags, it passes this data onto the Raspberry Pi. A connection is then established with the Firebase. Once the connection is secure, the Raspberry Pi parses this information into JSON format before pushing it up to the Database.

Priority: High

Use Case

Scope

The Scope of this use case is for the Raspberry Pi to receive data from the RFID Controller, then post this data to the Database (Firebase).

Description

This use case describes the process by which the Raspberry Pi receives data from the RFID Controller, connects to the Database, then updates the Database with the information it previously received.

Use Case Diagram:

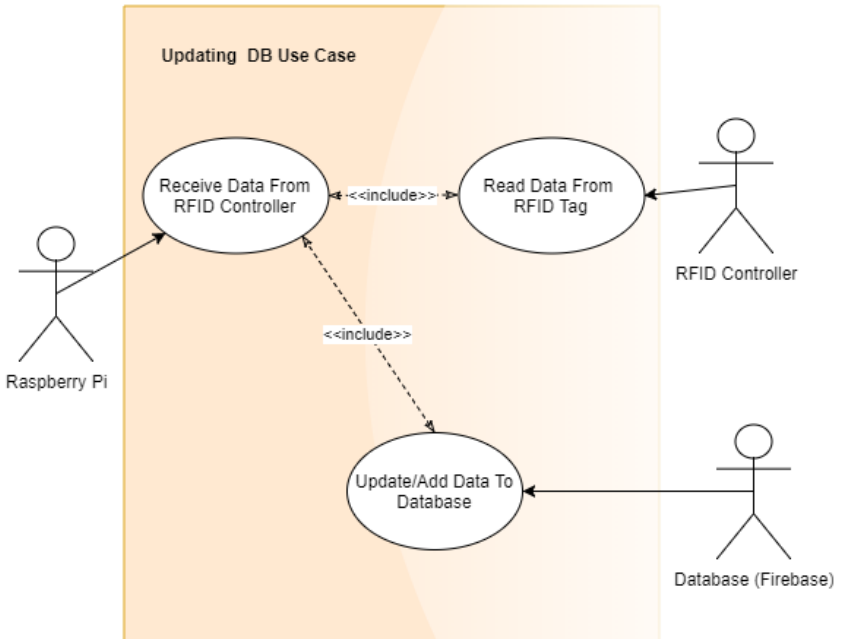


Figure 13 - Requirement 3 Use Case Diagram

Precondition
The RFID Controller must be connected and have just received data from the RFID tags.

Activation
The system starts when the RFID Controller sends data to the Raspberry Pi.

Main Flow

1. The RFID Controller reads data from the RFID Tag
2. The RFID Controller sends data to the Raspberry Pi
3. The Raspberry Pi receives data from the RDIF Controller
4. The Raspberry Pi establishes a connection with Firebase Database [A1]
5. The Raspberry Pi Updates Data in the Database

Alternate Flow 1

1. The Raspberry Pi Adds New Data in the Database
2. Return to Main Flow [5]

Exceptional Flow
N/A

Termination
The system terminates when Raspberry Pi consumes the data from the RFID Controller, then updates or adds a new entry to the Database.

Post Condition
Once the data has been read and sent onto the Database, the Raspberry Pi waits for more data to be send from the RFID Controller.

//----------End of Use Case ----------//

3.1.5 Requirement Number 4 <View Food Items>

Overview & Priority

One of the primary features of the Application is to enable users to remotely check what food items are present within their home. The "Checking Food Items" use case involves the Android application and the Firebase Database. This use case outlines how the user can browse through the food items in their home.

Once the user is signed into the application and clicks into the "My Food Network" Android Activity - they will be presented with a list of all the food items within their home (Refrigerator & Cabinets). The user may also be able to filter this information based on food categories i.e. Poultry, Dairy, Vegetables etc.

When the Android Activity is loaded, the App will establish a connection with Firebase, thus pulling the required information into the application. Although not vital in terms of back-end functionality, this use case encapsulates one of the Primary features of the system - as such it has been allocated a "High Priority".

Priority: High

Use Case

Scope

The Scope of this use case is to establish a connection with the Database, Read the required Data, then pull this data into the Android Application and parse to the UI.

Description

This use case describes the how the "My Food Network" section functions. The user will be presented with a home page and once they navigate to the "View My Food Items" section, they will be presented with an overview of all the food items in their home. The system will establish a connection with the Database, pull the associated data and then present this to the user. The user will be then be able to view and filter through the items via food Category.

Use Case Diagram:

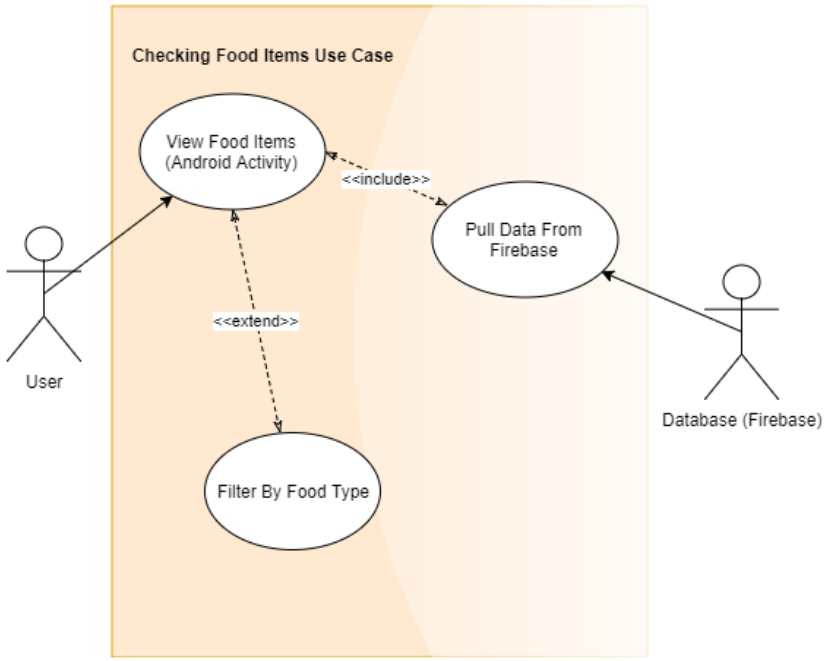


Figure 14 - Requirement 4 Use Case Diagram

Precondition
The User must have the application installed and have established a connection with the Internet.

Activation
The system starts when the User clicks into the" My Food Network" Android Activity.

Main Flow

1. The User Opens the "My Food Network" Activity
2. The Android Application establishes a connection with Firebase
3. The Android Application pulls the required data from Firebase [A1,A2]
4. The Android Application presents the data to the User within the App GUI
5. The User is notified of the status through the UI

Alternate Flow 1

1. The User Filters the contents by Food Type
2. Return to Main Flow [4]

Alternate Flow 2

1. The User has no food items
2. No items can be displayed in the UI
3. Return to Main Flow [5]

Exceptional Flow
N/A

Termination
The system terminates when the User returns to the Homepage, thus closing the associated section.

Post Condition
Once the user has either closed the Section, the system waits for the User to re-open the “My Food Network” section.

//----------End of Use Case ----------//

3.1.6 Requirement Number 5 <View Recipes List>

Overview & Priority

Users can view a listing of recipes from retrieved from the Spoonacular API in the “RecipesList”, “Recipes” and “RecipeDetails” activities. This use case makes a GET request to the API. The API returns a list of recipes that the Android Application can display within the UI.

Priority: High

Use Case

Scope

The scope of this use case is to allow the user to view a list of recipes in the “RecipesList” activity. This use case is relevant to two API endpoints

1) Recipes via Ingredient
2) Similar Recipes via ID

Description

This use case describes the process how the Application makes a GET request to the Spoonacular API for recipes and then parses the returned data to the UI.

Use Case Diagram:

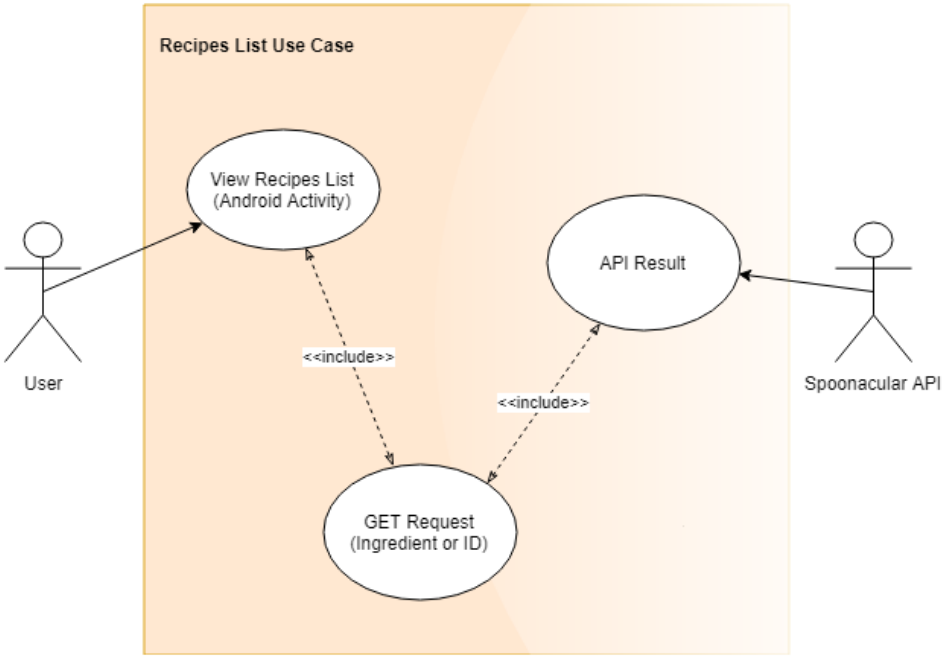


Figure 15 - Requirement 5 Use Case Diagram

Precondition
The User must be logged into the Mobile Application and have an active internet connection.

Activation
The system may start from different sections in the Application:

- When a User clicks the “View Recipes” option via the “My Food Network” activity
- When the User selects the “Similar Recipes” option via the “RecipeDetails” activity
- When the User Searches for a recipe via the “Recipes” activity
  In any of the scenarios above, the user is routed to the “RecipesList” activity.

Main Flow

1. The User clicks the “View Recipes” button [A1, A2]
2. The Android Application sends a GET request to the Spoonacular API using the endpoint “Recipes via Ingredient”
3. The API returns a list of recipes
4. The Android Application parses the list of recipes to the UI

Alternate Flow 1

1. The User clicks the “View Similar Recipes” button
2. The Android Application sends a GET request to the Spoonacular API using the endpoint “Similar Recipes via ID”
3. Return to Main Flow [3]

Alternate Flow 2

1. The User searches for a recipe via the “Search” option in the “Recipes” activity
2. The Android Application sends a GET request to the Spoonacular API using the endpoint “Recipes via Ingredient”
3. Return to Main Flow [3]

Exceptional Flow
N/A

Termination
The system terminates when the API result is parsed into the UI.

Post Condition
The system waits in an idle state for the user to select a recipe from the list.

//----------End of Use Case ----------//

3.1.7 Requirement Number 6 <View Recipe>

Overview & Priority

Users can navigate to the "ViewRecipe" activity to view recipe details. This section of the application will display a full recipe from the Spoonacular API including all details such as ingredients, instructions and other key information.

Priority: High

Use Case

Scope

The scope of this use case is to allow the user to view the Recipe they have requested. This section may be accessed in a variety of ways:

- When the User clicks a recipe from the results in the "Recipes List" activity
- When the User clicks on the "View Recipe" from the "Favourite Recipes" activity

Description

This use case describes the process how the Application makes a GET request to the Spoonacular API for a specific recipe, then displays the recipe details in the UI.

Use Case Diagram:

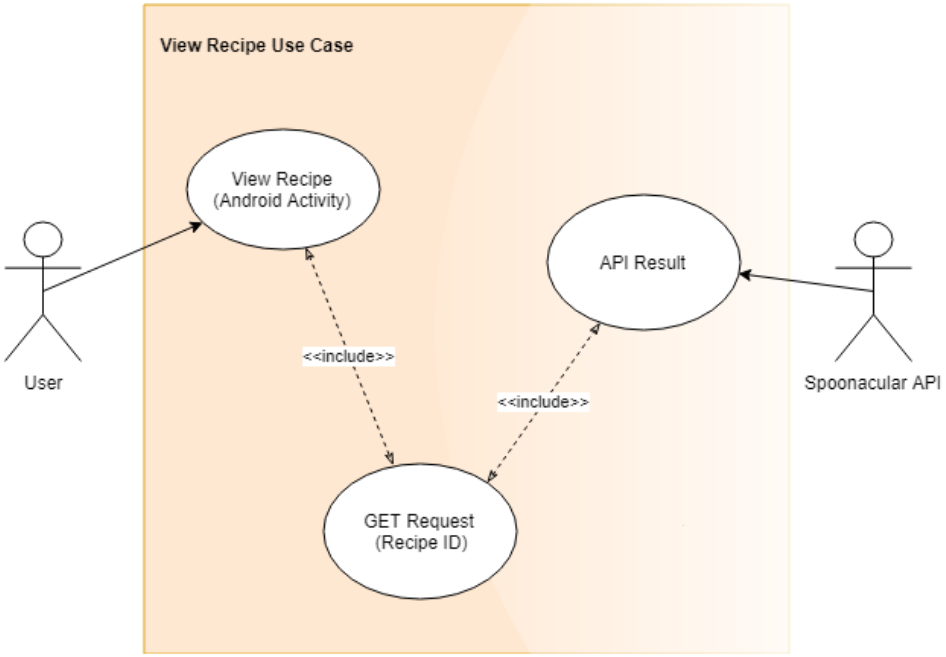


Figure 16 - Requirement 6 Use Case Diagram

Precondition
The User must be logged into the Mobile Application and have an active internet connection.

Activation
The system may start from different sections in the Application:

- When the User clicks a recipe from the results in the “Recipes List” activity
- When the User clicks on the “View Recipe” from the “Favourite Recipes” activity
  The Endpoint “Recipe via ID” is utilized in both circumstances.

Main Flow

1. The User clicks on the recipe from the “Recipes List” or “View Recipe” from their “Favourites Recipes” [A1]
2. The Application makes a GET Request, passing the recipe ID to the API
3. The API returns the Recipe Data
4. The Application parses the data to the UI

Alternate Flow 1

1. The User has no favourite recipes
2. The User will not be provided with the option “View Recipe” if they do not have any favourites
3. The user is prompted to add a recipe to their favourites to avail of the service
4. Return to Main Flow [4]

Exceptional Flow
N/A

Termination
The system terminates when the API result is parsed into the UI.

Post Condition
The system waits in an idle state for user interaction.

//----------End of Use Case ----------//

3.1.8 Requirement Number 7 <Add Recipe to Favourites>

Overview & Priority

If a user likes a recipe, the application provides the option to add it to their Favourites. All the Users Favourites are stored in their "FavouriteRecipes" table in the Firebase database. Users are not permitted to add duplicate entries.

Priority: Medium

Use Case

Scope

The scope of this use case is to allow the user to add a Recipe to their Favourites by saving the Recipe to the Favourite Recipes table in the Firebase Database.

Description

This use case describes the process how the Application adds a recipe to the user's favourite recipes table in the Firebase Database.

Use Case Diagram:

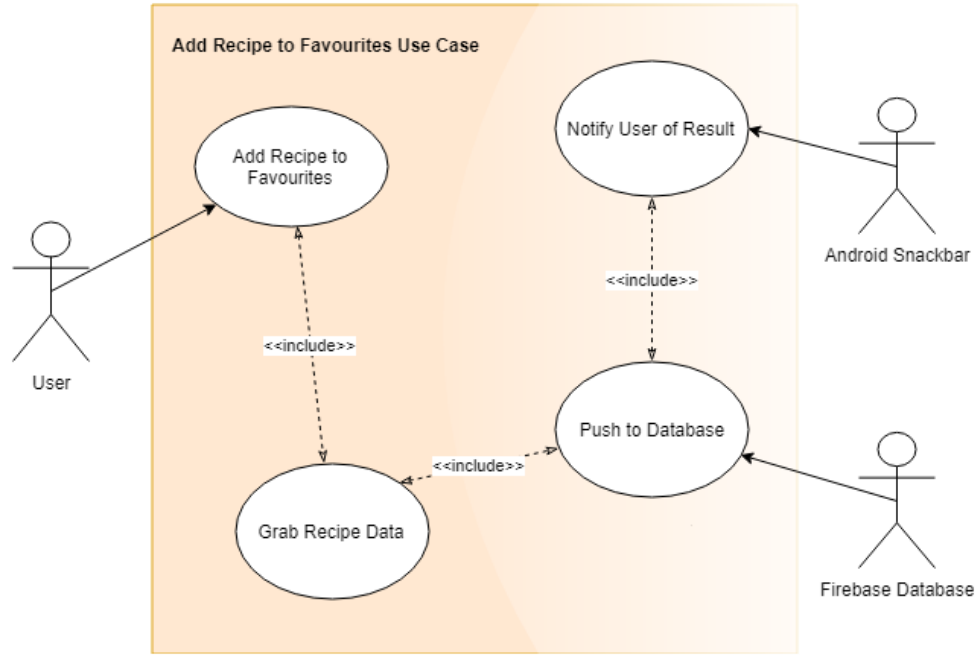


Figure 17 - Requirement 7 Use Case Diagram

Precondition
The User must be logged into the Mobile Application and have an active internet connection.

Activation
The system starts when the user clicks the Heart icon on the Action Bar in the “Recipe Details” activity.

Main Flow

1. The User clicks the Heart Icon to add the Recipe to their Favourites [A1]
2. The Application establishes a connection with the Firebase Database
3. The Application grabs the Recipe details
4. The Application pushes the Recipe Details to currently logged in users Favourite Recipes table in the Firebase Database
5. The user is Notified of the Result via the Android Snackbar

Alternate Flow 1

1. The Recipe is already in the Users favourites
2. The Recipe is not added to the User Favourites
3. Return to Main Flow [5]

Exceptional Flow
N/A

Termination
The system terminates when the User is notified of the result.

Post Condition
The system waits in an idle state for user interaction.

//------------------------------End of Use Case ------------------------------//

3.1.9 Requirement Number 8 <Share Recipe>

Overview & Priority

If a user likes a recipe, the application provides the option to Share it.

Priority: Low

Use Case

Scope

The scope of this use case is to allow the user to share a Recipe. The Application allows users to share a recipe on any of the applications they have installed on their Android device.

Description

This use case describes the process sharing a Recipe using the Android Application.

Use Case Diagram:

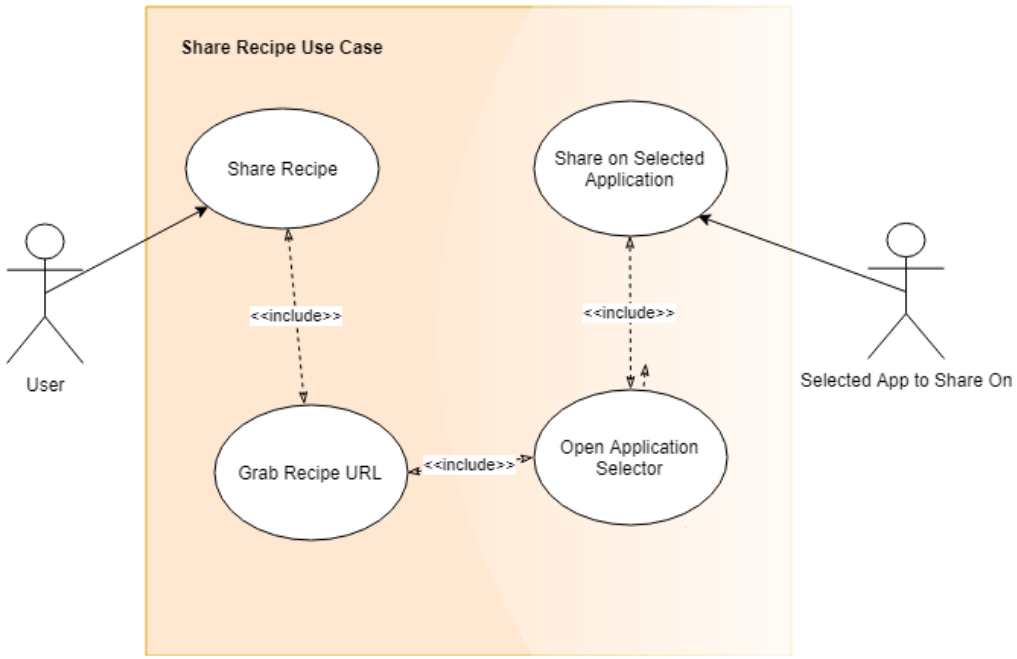


Figure 18 - Requirement 8 Use Case Diagram

Precondition
The User must be logged into the Mobile Application and have an active internet connection.

Activation
The system starts when the user clicks the Share icon on the Action Bar in the “Recipe Details” activity.

Main Flow

1. The User clicks the Share Icon to Share the Recipe
2. The Application grabs the current Recipe URL from the Recipe Object
3. The Application creates a new intent which grabs the users installed applications
4. The User can now choose which Application they would like to Share the Recipe on [A1]
5. The Recipe is Shared on the selected Application (i.e. WhatsApp, Email etc.)
6. The User is redirected back to the Intelligent Food Network Application

Alternate Flow

1. The User cancels the Share Intent
2. The Recipe is not Shared
3. Return to Main Flow [6]

Exceptional Flow
N/A

Termination
The system terminates when the User is redirected back to the Application.

Post Condition
The system waits in an idle state for user interaction in the “Recipe Details” activity.

//----------End of Use Case ----------//

3.1.10 Requirement Number 9 <View Favourite Recipes>

Overview & Priority

The Application provides the Users with the functionality to add a Recipe to their Favourites - therefore they must be able to view a listing of their Favourites.

Priority: High

Use Case

Scope

The scope of this use case is to allow the user to view a listing of their Favourite Recipes. All Favourite Recipes are stored in the Firebase Database.

Description

This use case describes the process of pulling all the User's Favourite Recipes from the Firebase Database and loading them into the UI.

Use Case Diagram:

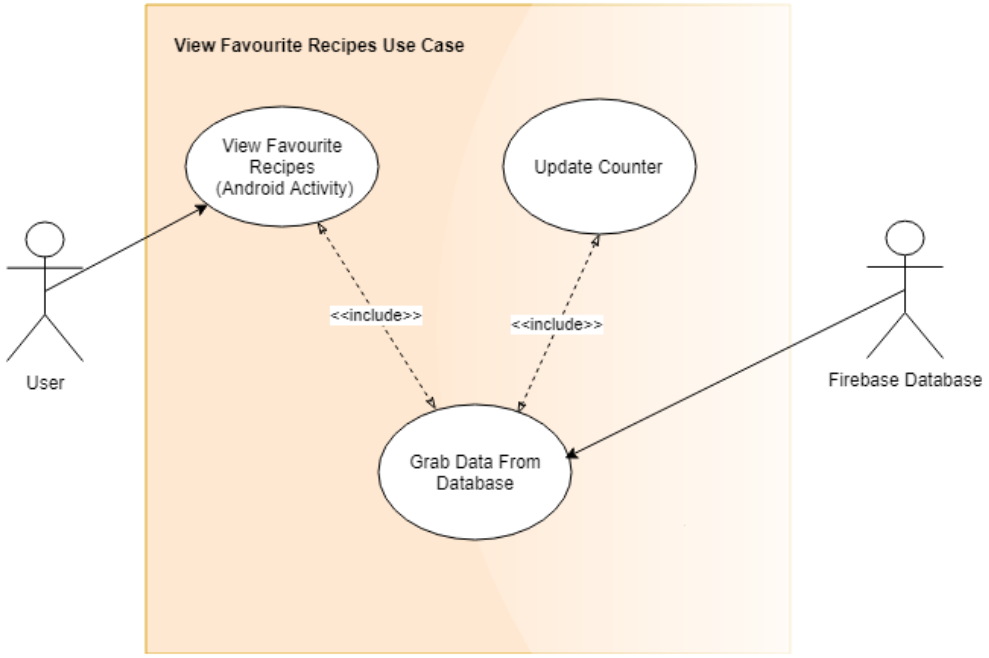


Figure 19 - Requirement 9 Use Case Diagram

Precondition
The User must be logged into the Mobile Application and have an active internet connection.

Activation
The system starts when the user enters the “Favourite Recipes” activity.

Main Flow

1. The User enters the “Favourite Recipes” activity
2. The Application establishes a connection with Firebase
3. The Application pulls a listing of all the Users Favourite Recipes [A1]
4. The Application parses the Favourites into the UI
5. The total number of Recipes are counted, and the counter is updated at the top of the UI
6. The User is notified of the result

Alternate Flow 1

1. The User has no Favourite Recipes
2. The counter defaults to Zero
3. Return to Main Flow [6]

Exceptional Flow
N/A

Termination
The system terminates when the User is notified of the result

Post Condition
The system waits in an idle state for user interaction in the “Favourite Recipes” activity.

//------------------------------End of Use Case ------------------------------//

3.1.11 Requirement Number 10 <Recipe Recommendation>

Overview & Priority

One of the fundamental concepts of The Intelligent Food Network is to provide users with recipe recommendations. Users are provided with the functionality to add and remove recipes to their favourites.

Each time a user visits the “Recipes” section, they will be presented with a recipe recommendation based upon their favourites. A list of all favourites is grabbed from the database, an API call is then made to pull similar recipes, thus generating the recommendation.

Priority: High

Use Case

Scope

The scope of this use case is to generate possible recipes the user may like based on their favourite recipes list. This use case uses the Spoonacular API for gathering the recommendations.

Description

This use case describes how the system will provide the user with possible recipes they may like prepare. Users favourite recipes are stored in the Firebase Database, as such this use case will require data retrieval from both Firebase and the Spoonacular API.

Use Case Diagram:

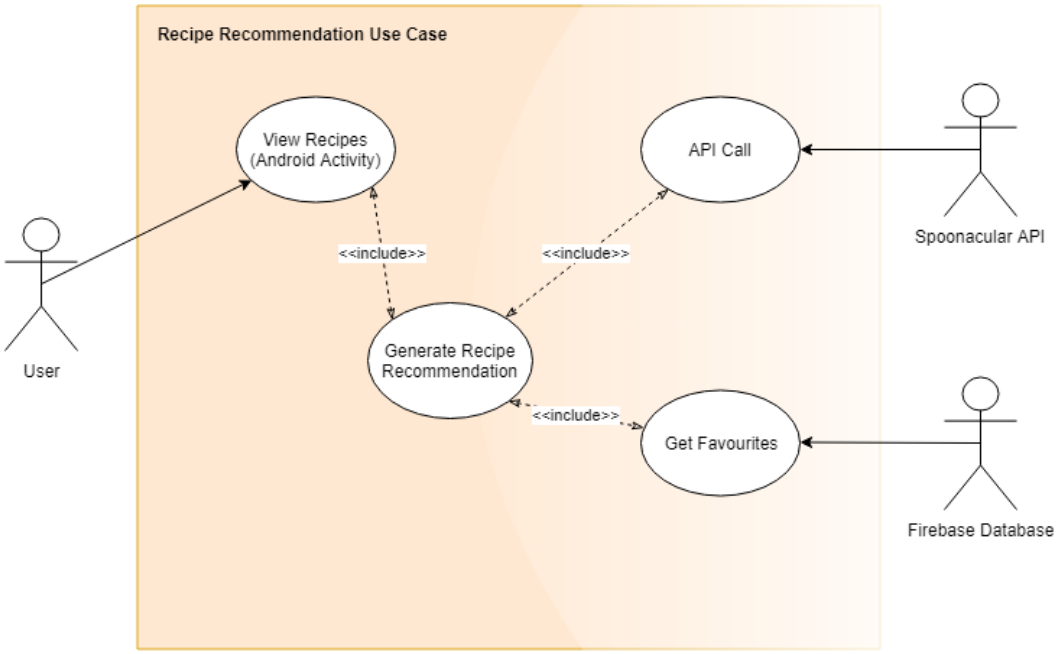


Figure 20 - Requirement 10 Use Case Diagram

Precondition
The User must be logged into the application, have an active internet connection and have at least one Favourite Recipe to avail of this service.

Activation
The system starts when the user clicks into the “Recipes” Activity.

Main Flow
1. The User Opens the “Recipes” Activity [A1]
2. The Android Application establishes a connection with Firebase
3. The Android Application establishes a connection with the Spoonacular API
4. The Android Application pulls the required data from Spoonacular and Firebase
5. The Android Application compares the data which is pulled from the API and Firebase
6. The Android Application presents the results to the user within the App GUI

Alternate Flow 1
1. The User has no favourite recipes
2. The Application does not generate a recommendation
3. The user is prompted to add a recipe to their favourites to avail of the service
4. Return to Main Flow [6]

Exceptional Flow
N/A

Termination
The system terminates when the user returns to the homepage or closes the application.

Post Condition
The system goes into a “wait” state for user interaction.

//------------------------------End of Use Case ------------------------------//

3.1.12 Requirement Number 11 <Shopping List Recommendation>

Overview & Priority

Another integral part of the system is to enable users to use the Mobile Platform to check which items they require on their next shopping trip – thus negating the need for opening even a single press or cabinet. Once a user clicks into the "Shopping List Recommendation" section on the app, they will be able to view what they require on their next shopping trip. This will be based on a comparison of the items within the users weekly shopping list and what food items they currently have in their home. This use case has been categorised as "Medium" priority - primarily because if it was excluded from the overall application it would not affect other functionalities or features.

Priority: Medium

Use Case

Scope

The scope of this use case is to create a list of items required on the next shopping trip for the user. This list will be created by pulling and comparing data from Firebase (Shopping List & Food Contents tables). The resulting data will then be pushed into the app UI.

Description

This use case describes how the system grabs the contents from the Firebase Database and compares them to evaluate what the user requires on their next shopping trip. The user will then be presented with a shopping list recommendation within the UI. This list is dynamic – i.e. it will change instantly if a food item is added or removed from the home via the RFID Controller.

Use Case Diagram:

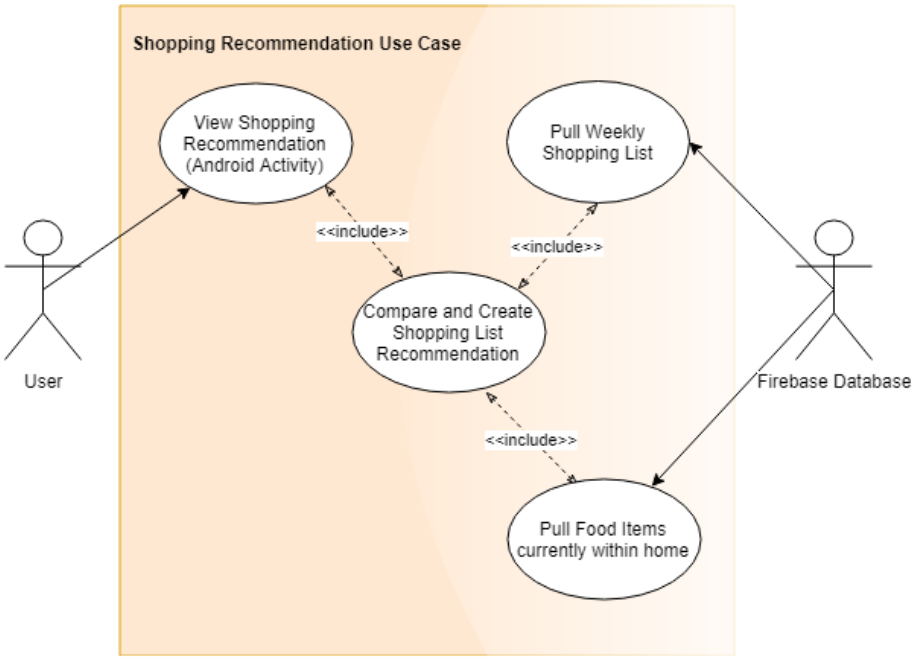


Figure 21 - Requirement 11 Use Case Diagram

Precondition
The User must be logged into the application, have an active internet connection and must have at least one item in their shopping list.

Activation
The system starts when the user clicks into the “Shopping List Recommendation” activity.

Main Flow
1. The User Opens the “Shopping List Recommendation” Activity [A1]
2. The Android Application establishes a connection with Firebase
3. The Android Application pulls the required data from Firebase
4. The Android Application compares the data which is pulled from Firebase (Shopping List and Food Contents in home)
5. The Android Application formulates a listing of potential items to purchase
6. The Android Application updates the UI

Alternate Flow 1
1. The User has no Shopping List items
2. The Application does not generate a recommendation
3. The User is prompted to add an item to their shopping list to avail of the service
4. Return to Main Flow [6]

Exceptional Flow
N/A
Termination
The system terminates when the UI is updated.

Post Condition
The system progresses into a “wait state” and awaits the next user command.

//----------End of Use Case ----------//

3.1.13 Requirement Number 12 <View Nutritional Data>

Overview & Priority

The Android Application provides User with the functionality of viewing nutritional data for any of the food items they please. The Nutritionix API is utilized to grab associated food data. Users can manually search for nutritional data, view nutritional data from the items in their home or view nutritional data of ingredients associated with a recipe.

Priority: Medium

Use Case

Scope

The scope of this use case is to display Nutritional Values of a particular food item to a user. A GET request is made to the Nutritionix API, the result is then parsed to the UI.

Description

This use case describes how the system will make a GET request to the Nutritionix API to provide the user with Nutritional Data based on their search criteria.
This use case may be utilized in various sections of the application – users can:

- Manually search for Nutritional Values via the "NutrientsSearch" activity
- View Nutritional Data of any food items in their home via the "My Food Network" activity
- View Nutritional Data of Ingredients associated with a recipe via the "Recipe Details" activity

Use Case Diagram:

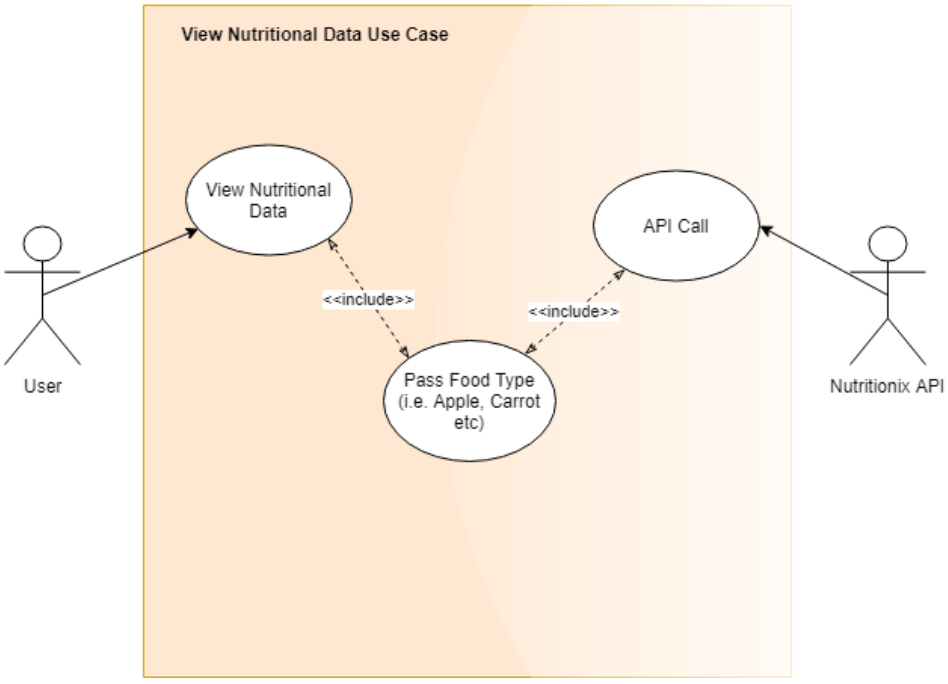


Figure 22 - Requirement 12 Use Case Diagram

Precondition
The User must be logged into the application and possess an active internet connection.

Activation
The system starts when the User requests Nutritional values via the activities outlined above.

Main Flow
1. The User Opens the “Nutrients Search” Activity [A1, A2]
2. The User manually searches for Nutritional Values
3. The Android Application establishes a connection with the Nutritionix API
4. The Android Application makes a GET request to the Nutritionix API
5. The API returns a result
6. The Android Application parses the result to the UI

Alternate Flow 1
1. The User clicks the “View Nutritional Info” dialog via the “My Food Network” activity
2. Return to Main Flow [3]

Alternate Flow 2
1. The User clicks the “View Nutritional Info” dialog via the “Recipe Details” activity
2. Return to Main Flow [3]

Exceptional Flow
N/A

Termination
The system terminates when the result is parsed to the Application UI.

Post Condition
The system goes into a “wait” state for user interaction.

//----------End of Use Case ----------//

3.1.14 Requirement Number 13 <User Account>

Overview & Priority

When a user first opens the application, they will be prompted to either log in or create an account. Once logged in - the user will be presented with the application homepage. From here, users will have the option to navigate to the “My Account Section”.

This section is based around a user’s account and anything associated with their profile. They will be able to view or manage their details such as Name, Phone Number, Weight and Upload a Profile Image.

Priority: Medium

Use Case

Scope

The scope of this use case is to provide users with basic account functionality within the Mobile Application. Users will be able to view and alter their account details using this service. This use case involves the Mobile Application, Firebase Authentication, Database and Storage.

Description

This use case describes the functionality that the “My Account” section provides within the mobile application. When users enter this section, they will be presented with a listing of their details. They may change or add to their details here if required. User credentials i.e. Email and Password will be stored into Firebase Authentication whilst other user detail such as contact information & phone etc. will be stored within the Firebase Database. When this information is requested by the user, it is pulled into the mobile platform.

Use Case Diagram:

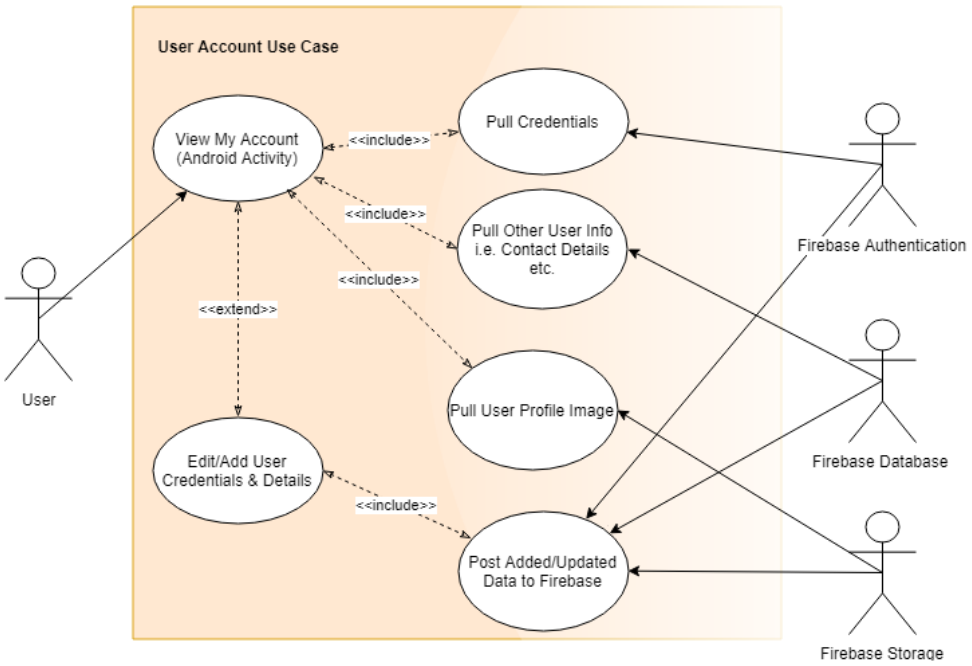


Figure 23 - Requirement 13 Use Case Diagram

Precondition
The user must be logged into their user account and possess an active internet connection.

Activation
The system starts when the user enters the “My Account” section/activity within the application.

Main Flow

1. The User opens the “My Account” Activity [A1]
2. The Android Application establishes a connection with Firebase Authentication, Database and Storage.
3. The Android Application pulls the data corresponding from the User Account
4. The Android Application presents the profile data to the user in the App GUI

Alternate Flow

1. The User opens the “Edit Account” Activity
2. The Android Application establishes a connection with Firebase Authentication & Firebase Database
3. The Android Application pulls the data corresponding from the User Account
4. The User makes alterations to their account details or uploads a new profile image
5. The Android Application posts the updated data to Firebase
6. The User is notified via confirmation message – i.e. Android Toast Widget
7. Return to Main Flow [4]

Exceptional Flow
N/A

Termination
The system terminates when the user exits the section or the mobile application.

Po**st Condition**
The system updates any changed data associated with the user profile and awaits another user request.

//----------End of Use Case ----------//

3.1.15 Requirement Number 14 <Change Profile Image>

Overview & Priority

Users which hold an account are given the option to edit their profile. This includes changing details such as Name, Phone and Weight etc. Users are also given the option to upload a profile image via the “Edit Account” activity.

Priority: Medium

Use Case

Scope

The scope of this use case is to allow the user to upload a Profile Image.

Description

This use case describes how the User updates their profile image via the “Edit Account” activity.

Use Case Diagram:

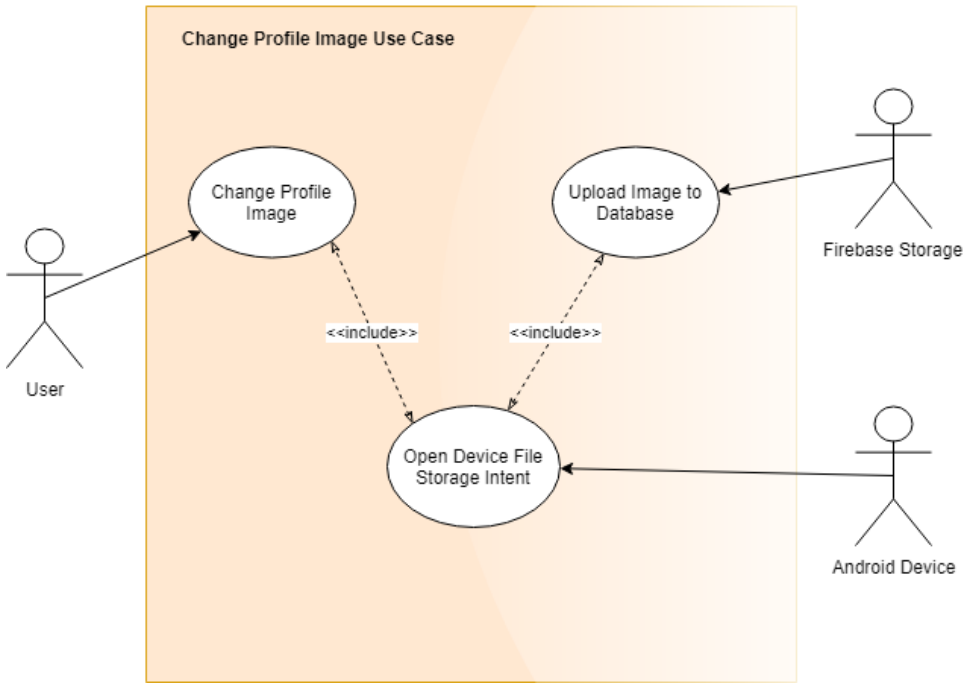


Figure 24 - Requirement 14 Use Case Diagram

Precondition
The user must be logged into the Mobile Application and possess an active internet connection.

Activation
The system starts when a user clicks on their profile image in the “Edit Account” activity.

Main Flow

1. The User clicks the Profile Image in the “Edit Profile” activity
2. The Android Application creates a new Intent which allows the user to access their device file storage (i.e. File Manager, Gallery etc)
3. The Android Application starts the Intent
4. The User selects an image from their Device Storage [A1]
5. The User is redirected back to the Intelligent Food Network Application
6. The Image is parsed into the Profile Image Thumnail so the User can see a preview
7. The User clicks the “Update Profile” button
8. The Image is posted to Firebase Storage
9. The User is notified of the result in the UI

Alternate Flow 1

1. The User Cancels the upload process
2. The User is redirected back the Application
3. Return to Main Flow [9]

Exceptional Flow
N/A

Termination
The system terminates when the User is notified of the result.

Post Condition
The system waits in an idle state for further User interaction.

//------------------------------End of Use Case ------------------------------//

3.1.16 Requirement Number 14 <Contact Developers>

Overview & Priority

Within the Mobile Platform, the "My Account" Section will also provide users with the facility to contact the developers – reporting issues or recommendations. From here users will be prompted to fill in some required fields within the UI.

When complete, the user can submit this information - where it will then be pulled into the default email client on the Android Device. This use case has been categorised as "Low" as it does not affect any other functionality within the application, nor is it a vital component within the system.

Priority: Low

Use Case

Scope

The scope of this use case is to provide users with the functionality to contact the developers regarding any queries or issues they have encountered. This use case will make use of the Android Systems default email client i.e. Gmail, Outlook etc.

Description

This use case describes the "Contact Developer" function where users will be able to contact the developers via the "Contact Developer" sub-section located within the "My Account" activity in the Mobile Application. This feature will make use of the default email client set on the user's device. Users will simply have to fill in some mandatory fields, this information will then be grabbed and placed into a draft email ready for sending.

Use Case Diagram:

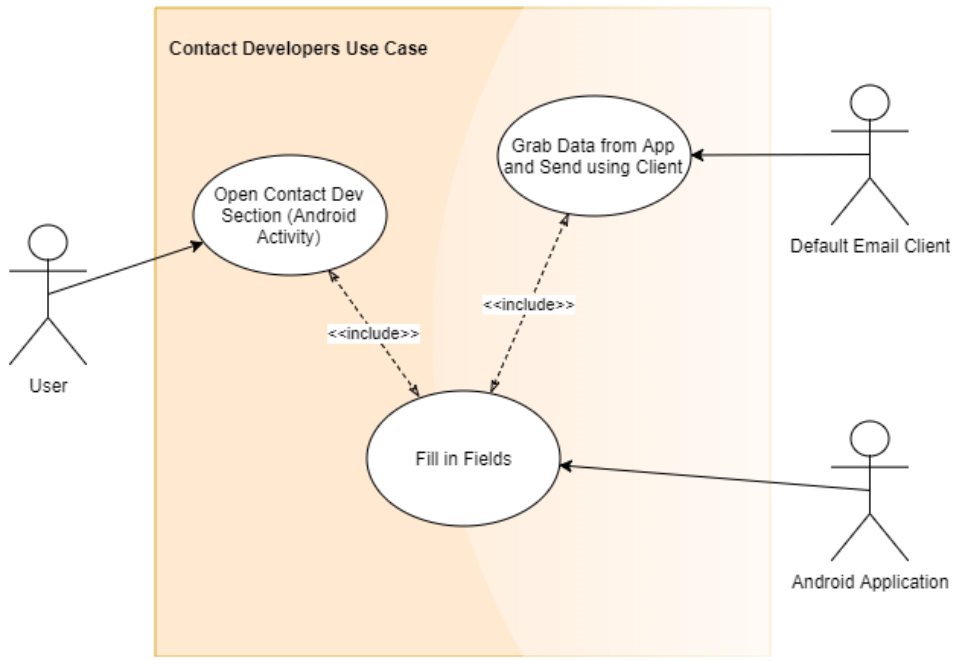


Figure 25 - Requirement 15 Use Case Diagram

Precondition
The user must be logged into their account and possess an active internet connection. This feature also requires that the user has at least one form of email client application installed onto their Android Device.

Activation
The system starts when the user clicks the Email icon at the top of the “My Account” activity.

Main Flow
1. The User Opens the “Contact Developer” Activity
2. The User fills in the required fields
3. The User Clicks Submit [A1]
4. The Android Application grabs the data and drags it into the default email client – thus, creating a draft email
5. The User sends the email
6. The User is Redirected back the Intelligent Food Network Application
7. The User is notified of the Result in the UI

Alternate Flow 1
1. The User does not have any email clients installed on their device
2. The Android Application cannot open any email clients
3. Return to Main Flow [6]

Alternate Flow 2
1. The User cancels the Email
2. The Email is saved as a draft in the Email Application
3. Return to Main Flow [6]

Exceptional Flow
N/A

Termination
The system terminates when the user is redirected back to the UI

Post Condition
The system sends and email through the user’s default email client, then idles in a “wait state” for additional user requests.

//----------End of Use Case ----------//

3.1.17 Requirement Number 16 <Log Out>

Overview & Priority

Users which hold an account may either log in or out of the Mobile Platform. This requirement will be based around the user log out process. This requirement has been categorised as “Low” as it will not affect any other elements within the system.

Priority: Low

Use Case

Scope

The scope of this use case is to allow the user to log out of the Mobile Application.

Description

This use case describes the log out functionality by which the user may log out of the Mobile Application.

Use Case Diagram:

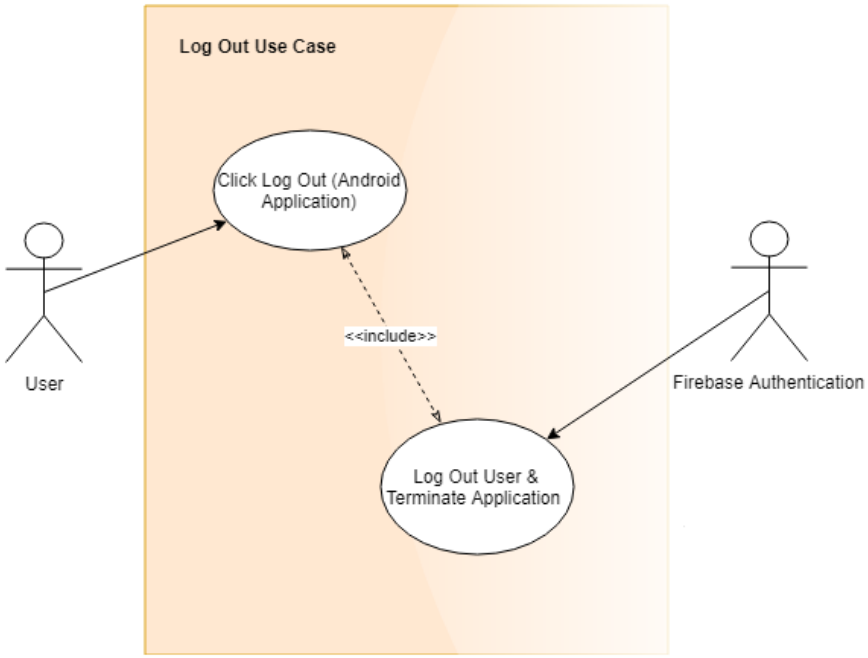


Figure 26 - Requirement 16 Use Case Diagram

Precondition
The user must be logged into the Mobile Application.

Activation
The system starts when a user clicks the “Log Out” icon on the homepage of the Mobile Application.

Main Flow

10. The User clicks the “Log Out” button
11. Firebase Authentication logs the User out of the Application
12. The Android Application terminates using the Finish() function
13. The User is returned to the Operating System Application Drawer

Alternate Flow
N/A
Exceptional Flow
N/A

Termination
The system terminates when the user has successfully logged out.

Post Condition
The user is logged out and the application closes.

//----------End of Use Case ----------//

3.2 Non-Functional Requirements

The system requires a variety of Non-Functional Requirements which are detailed below.

3.2.1 Performance/Response time Requirement

This requirement is associated with the overall system speed and fluidity. Both hardware and software components associated the system must work together in a seamless manner. The system must perform adequately on all targeted devices and platforms. There should be low-latency between user interaction and back-end response. The system must be capable of handling multiple user requests and returning any associated data in a swift and timely manner.

To uphold this requirement, the Android application was designed to target all devices as far back as Android 4.4 (Kit-Kat). (Min SDK 19 = Android 4.4, Target SDK 26 = Android 8.0)
This option was selected at the start of development when initially setting up the Android Studio project. The system design was fundamentally based around a "Thin-Client" architecture. All the heavy data and storage is located in the Firebase cloud platform (i.e. the "Thick-Client"). The mobile application can call to the database to retrieve the required data wherein the Android device acts as the "Thin-Client".
Priority: **High**

```
android {
    compileSdkVersion 26
    defaultConfig {
        applicationId "ie.ncirl.a14445618.theintelligentfoodnetwork"
        minSdkVersion 19
        targetSdkVersion 26
        versionCode 1
        versionName "1.0"
```

Figure 27 - Application Min & Max SDK

3.2.2 Availability Requirement

This requirement is associated with the amount of uptime & downtime that will incur within the system over a given period – i.e. a single year. An ideal scenario would see the system constantly operational whilst upholding an uptime of 100% - although in the real world this situation rarely occurs. Most systems such as Websites and Applications are usually within the uptime range of 98-99%. This is to allow for maintenance, patches, updates and bug fixes.

The primary access point between users and The Intelligent Food Network system is through the Mobile Application. All application data is stored within the Google Firebase platform where it can be downloaded at any time once the Google Servers are operational. (These have a very low downtime). Utilizing this service, the system experiences a very low downtime and thus, conforms to the current industry standard availability requirements.
Priority: **High**

Figure 28 - Firebase & The Intelligent Food Network

3.2.3 Backup & Recovery Requirement

This requirement defines how the system will implement a fall-back approach in the case of a catastrophic failure. The Intelligent Food Network is currently utilizing Googles Firebase platform - All content ranging from user accounts to encrypted passwords and food data are stored within this back-end system.

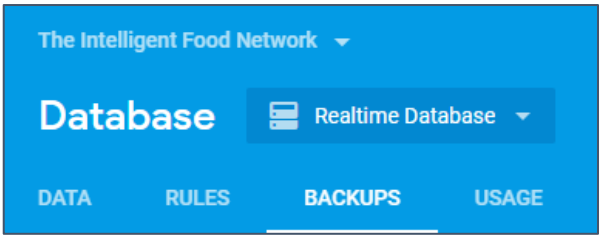


Figure 29 - Firebase Database

By default, Firebase automatically runs a daily back-up stores all content within Google Cloud Services. These backups enable the system to restore itself to a previous iteration in the case of a major failure - thus, providing adequate backup and recovery options if required. Firebase also provides developers with the tools to customise system back-ups to their own storage location.
All developed code associated with the system including Android Application files, Raspberry Pi Scripts etc have been pushed to a local repository on GitHub. The inclusion of GitHub not only provides a means of version-control, but also an alternative backup platform.
Priority: **High**

3.2.4 Reliability Requirement

This requirement is associated with how well the system will cope with general usage. The system must be able to withstand its intended usage and carry out user requests. This requirement can tie in well with Robustness - although it is primarily concerned with the targeted usage as opposed to strenuous usage described in the Robustness Requirements section below.

The system contains many elements which must work together to provide a seamless experience. These elements include the Raspberry Pi, RFID Controller, RFID Tags and The Android Application. The system must not crash and be competent enough to run on a multitude of different devices. The system must adhere to the role it is designed for.

The RFID Controller and Raspberry Pi must be capable of reading and processing all user food items, then posting them to Firebase. The Android Application must process all user requests whilst returning the appropriate response from the server in a timely manner. Such examples of the Android Application Reliability Requirements include-

- Listing Food Items
- Recommending Recipes
- Log in/Registration/Logout
- Editing User Account
- Viewing Nutritional Data

The system must not crash or function intermittently. The system has been tested with real life candidates and on a variety of different hardware to meet the outlined reliability requirements.
Priority: **High**

3.2.5 Robustness Requirement

This requirement defines how the system must be capable of handling errors coupled with a high amount of user requests. The main contact point between users and the system is the Android Application. This application must be robust and be able to withstand a high influx of user requests. The application has been designed to ensure it is as error-free possible

Both Internal and External testing methodologies have been utilized o extensively test the system and highlight any potential sections that may cause failures. One of the main points of failure during the development cycle was incorrect results received from the API's. To combat this, many Try-Catch style clauses have been implemented to resolve these issues and further increase overall robustness of the entire platform. An overview of system testing can be seen in "***Section 8 - Testing***".
Priority: **High**

3.2.6 Security Requirement

Figure 30 - Firebase Authentication

Security is an essential component within any software system. The Intelligent Food Network is no exception and must implement adequate security measures to keep both system and user data confidential and secure. The system security measures will be implemented in a variety of ways – The Raspberry Pi and RFID Controller will be connected to the user's secure local network. Most of today's Wireless Home networks feature some form of encryption or built in security, so this aspect is covered.

The Next aspect of the system is the Android Application – Similar to the above, the application has been designed to be primarily utilized on a secure home or 4G network. User accounts and passwords have been securely encrypted and stored within Firebase Authentication, whilst all other associated data linked to the User account is stored within the Firebase Database (Profile images in Firebase Storage). The application requires users to sign up or log into their account prior to access. Any data entered within the application is securely transmitted from the device to the Firebase servers. Users will only be provided access to information corresponding to their own account. Users are notified if they have provided the incorrect login credentials.
Priority: **High**

3.2.7 Maintainability Requirement

This requirement is associated with the ease in which the system can be maintained - i.e. provide platform updates, alterations & bug fixes. This section will primarily focus on the Android Application. The Application has been developed within the Android Studio IDE. Any alterations or changed to the source code has been recorded and pushed to the Cloud using GitHub. The Android Studio Platform provides all tools for system development, testing and deployment – as such any updates, alterations and bug fixes have been carried out within this single IDE. This negated the need for alternative programs/platforms and increased overall maintainability during the development cycle.
Priority: **Medium**

3.2.8 Portability Requirement

Portability or "machine-independency" refers to the way a system can run on a multitude of platforms and devices. The system has been developed within Android Studio – and is available across a variety of Android OS versions – 4.4 Kit Kat being the lowest target OS Version. The Android Studio IDE caters for development across multiple devices and platforms - ranging from Small-Screen Smartphones to large Android Tablets.

From the research and survey process outlined in "***Section 2 – User Requirements***", it is clear that the user base of the platform is varied. As such, portability plays a key factor in increasing satisfaction and overall usability. One such example is where a user has installed the application onto their mobile phone - yet also would like to access its features on their Android tablet. The Intelligent Food Network will be developed with scenarios like this in mind. The system has been developed to scale equally well regardless of device size or screen resolution.

There is also such a varied amount of hardware in today's electronics market that it wouldn't be viable to simply develop for a single platform or OS. My system will cater for all users, albeit without sacrificing overall functionality – therefore the system will only support Android Devices as low as 4.4 - KitKat. To conform to this requirement, the system was continuously tested on a variety of Virtual devices within the Android Studio IDE during the development cycle. Once the development cycle had elapsed, the Android APK was created for further testing on physical Android devices. The testing results can be seen in "***Section 8 - Testing***".
Priority: **High**

3.2.9 Reusability Requirement

Reusability refers to how assets may be re-used for future implementation or development. The system has been designed to make efficient use of any required assets –code snippets, classes, layouts and design elements such as icons and colour schemes have been reused within the Android Studio IDE. Android Studio provides specific sections for styles (i.e. Colors.xml and Strings.xml).

```
<resources>
    <color name="colorPrimary">#159B4A</color>
    <color name="colorPrimaryDark">#10783a</color>
    <color name="colorAccent">#535353</color>
    <color name="white">#FFFFFF</color>

    <color name="darkRed">#8B0000</color>
    <color name="darkBlue">#00008B</color>
    <color name="darkGreen">#006400</color>
</resources>
```

Figure 31 - Colours.xml

Colours.xml was specifically used during development to inherit the Application colour scheme. This negated the need for manually assigning a colour to each UI element. Strings.xml was also utilized in a similar manner – with all String being stored in the one file and referenced within the application. This design choice was specifically useful during development as it allowed all the Action Bar strings to be stored in one location where they could be programmatically referenced within each class.

These design choices have ultimately increased overall fluidity and work to reduce the latency between sections.
Priority: **Medium**

3.2.10 Resource Utilization Requirement

This requirement describes how the system makes use of its existing assets and resources. To scale efficiently across all platforms, the system must utilize the provided hardware. For example, the system must work just as smoothly on two devices – one with 4gb of RAM and the other with 1 Gb of RAM. Throughout development - this scenario and other similar scenarios have been accounted for. The Android Application was installed and tested on a variety of different devices. Therefore, the system will scale efficiently on any device regardless of screen size or resolution.

Priority: **High**

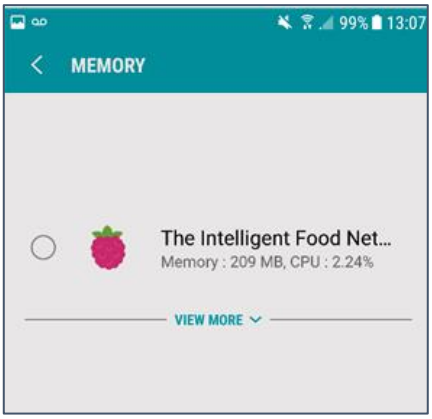


Figure 32 - Application Device Usage

4 Interface Requirements

4.1 Overview

The interface is the primary contact point between the user and the system - as such it plays an integral role in user interaction and overall satisfaction.

The system has been designed to achieve an industry-standard look and feel. The Android Application GUI is derived from Googles Material Design guidelines. An elegant, yet user friendly GUI was formulated to make the application look and feel as professional as possible.

The Android Application utilizes the same Custom theme throughout to provide a consistent and recognisable UI. Each section inhibits similar menus, buttons and options. This was once again to conform to the Material Design Framework. Effectively, each section in the application (Food Contents, Shopping or User Account has been allocated its own individual activity). This aided development, but also made code much more legible and easier to formulate. Any sections which were very similar utilized the same layout resource files – this process was undertaken for all similar GUI elements including buttons, navigation and menus etc. Developing the system in this manner not only reduced duplication of code, but also enabled the use of common Android design patterns - one such example being the “Structural Pattern” which was used when implementing the Custom Adapter functionalities.

4.2 Design Choices

As listed in the previous sections, the Application was developed in Android Studio. Android Studio provides a very useful GUI designer which supports both a “Drag and Drop” and “Text Creation” section. The Text Creation section was chosen to develop the GUI as it is much more accurate in designing the UI. It also enables pin-point accuracy placement of views such as buttons, images and other GUI elements. Linear Layouts were primarily utilized to hold the GUI elements. These were chosen as they correctly scale the application based on screen size - therefore enabling interoperability across a multitude of different Android devices.

Android Studio also provides the option to use Templates when creating an Activity. The “Blank” template was chosen for every Activity in the application. This aided the design of an original GUI which was built from the ground up – in contrast to just using an array of different templates.

A significant amount of time was spent researching different Food, Recipe and Nutritional applications. This research was used to formulate the best possible design queues and GUI elements from similar applications on the Play Store. Below are some of the designs that were taken into consideration during the GUI development phase:

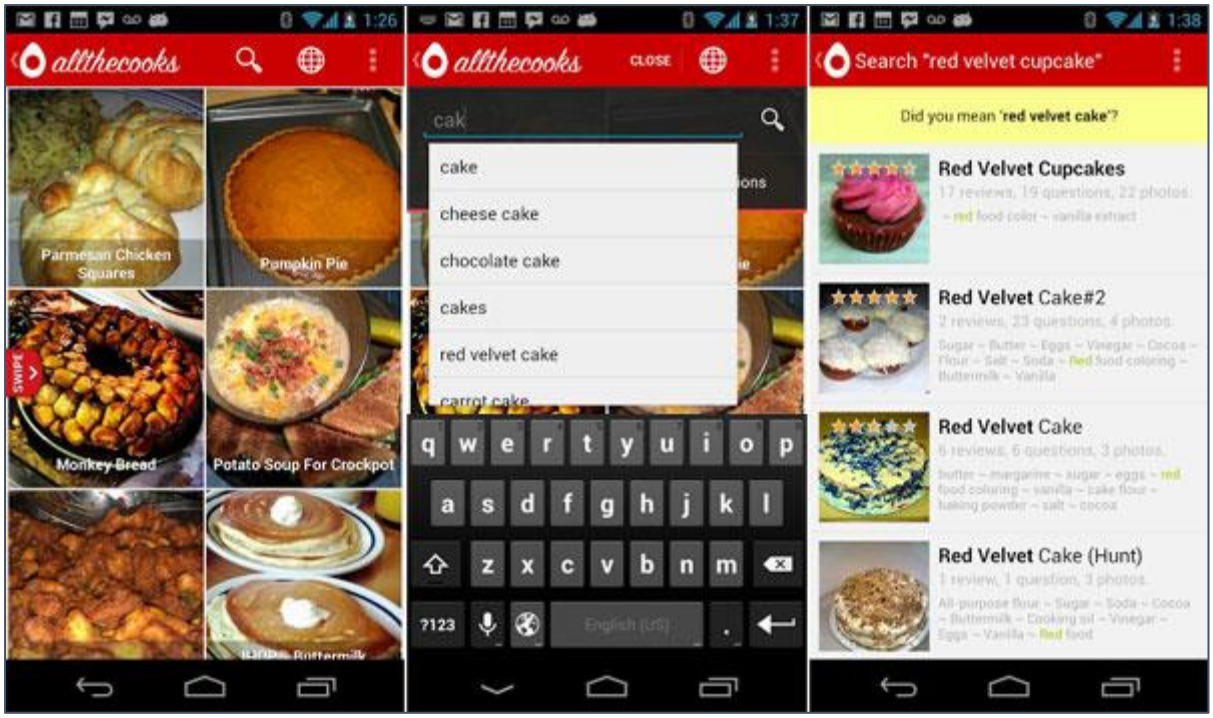


Figure 33 - GUI Idea 1

Figure 34 - GUI Idea 2

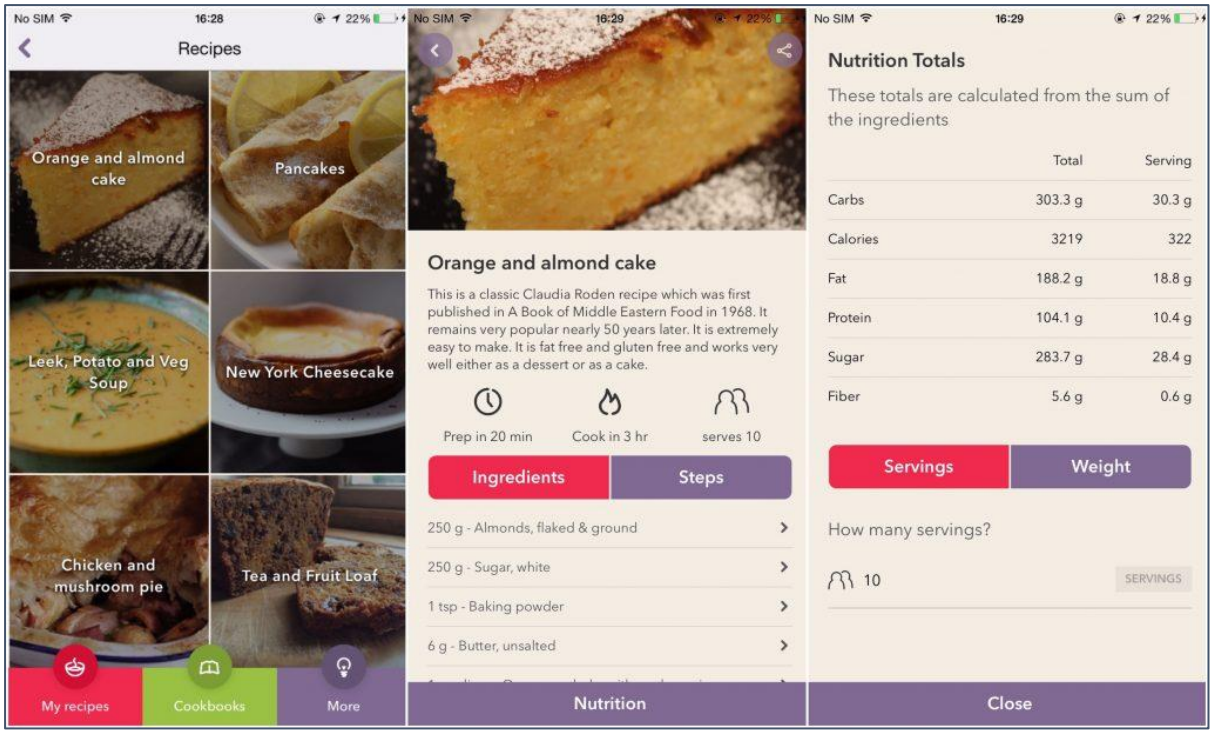


Figure 35 - GUI Idea 3

These are just a small selection of the designs that were researched but they capture some of the most important design ideas the application was derived from. An overview is as follows:

- The recipes section incorporates a GridView with the title of the recipe hovering over the image in the background. This design was utilized for almost every Recipe or Food application researched.
- The Action-bar was heavily utilized to hold useful components i.e. search, share and hold the current recipe/section title.
- Crucial recipe details (i.e. Preparation/Cook time and Serving size) are located at the top of the GUI in large and easily legible font.
- Most Food applications utilized either a Red or Green colour theme. After conducting more research, it was uncovered that these colours usually represent Appetite and Nature/Healthiness.
- Ingredients were clearly laid out in the recipes section and could be interacted with in the same section.
- Recipe images and titles were presented in large font with the supporting information (ingredients, instructions etc. following below - inhibiting a decreased font size).

These images significantly influenced the overall design of the application. One of the most crucial components being colour choice. Initially, the application was a blue-navy colour, but after conducting this research it was decided that it would need to be changed as blue was not deemed an ideal suit for the type of application. (Blue is more suited to Security applications).

After much chopping and changing of design the colour **#159B4A** was chosen. This colour can be described as a “Dark Cyan – Lime Green”. It perfectly suited the application and was deemed not too bright nor too dull.

4.3 GUI

4.3.1 Splash Screen

One of the most popular design cues within todays Mobile Applications are Splash Screens. A Splash Screen effectively acts as a "filler" between the user opening the application and the content loading. It can also be categorised as a "loading screen". The Splash Screen is a lightweight component and usually displays a logo or some form of trademark information when the application is initially launched. This is to allow the app time to pull any dependencies or data before it is displayed to the user. Splash Screen usually only last for about 1-4 seconds, but they provide the essential loading period the system requires on start-up – whilst also portraying a very professional feel within the app.

The application requires a variety of external dependencies such as Firebase Authentication, Storage, Database, as such a Splash Screen was implemented to ensure the system is allocated the vital loading time it requires. The splash screen is relatively simple - simply displaying the App Logo. The Logo was created in GIMP Image editing software. Once the loading phase is complete, the application progresses onto the user Login and Registration Activity.

Figure 36 – Splash Screen

4.3.2 Login Page

These GUI elements are associated with **Functional Requirement No 1**.

To access all the system features, users are required to either Log in or Register for a new account. The Log in Activity is relatively basic and only requires users to enter their credentials in the form of an email address and password. They may then click the login button below. This page will also provide a link to the "Register" activity so users whom do not have an account may create one. Firebase Authentication has been designated as the primary form of Authentication within the system.

4.3.3 Registration Page

The registration activity can be accessed from a button at the bottom of the Login Page. This page allows users to sign up to the application. An option to return to the login activity is also present below the registration button. Users are required to enter mandatory fields such as name, email address, contact number and a password. Both Login and registration pages feature the application Logo at the top section and prevent the user from logging in with incorrect credentials.

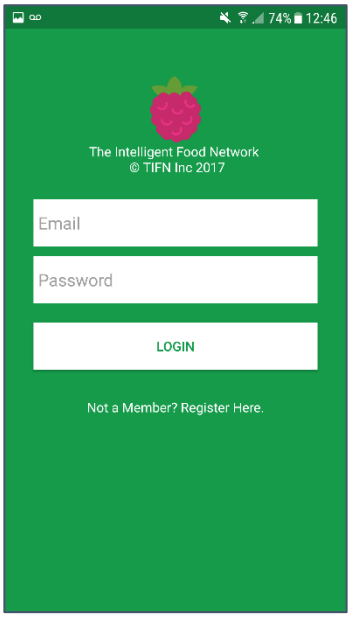


Figure 37 - Login

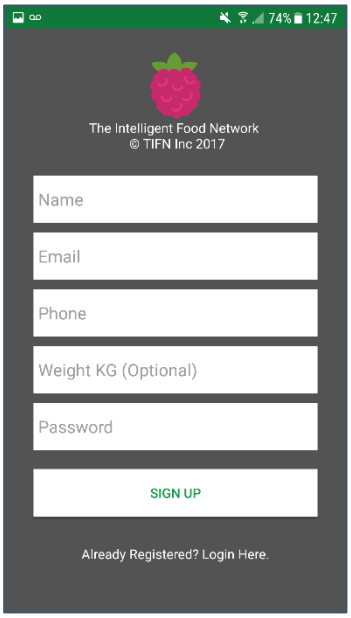


Figure 38 - Registration

4.3.4 Home Screen

One of the fundamental GUI elements that should be present in any mobile application is a Home Screen. After successful login or registration, users will be sent to the Home screen. From here, they can navigate to the desired sections. The Home screen features primary icons (ImageButtons) which represent each section of the application. A minimalistic design approach was taken when designing this section. All icons are large and clearly labelled to increase usability. Similar to the login activity, the application logo is located at the top.

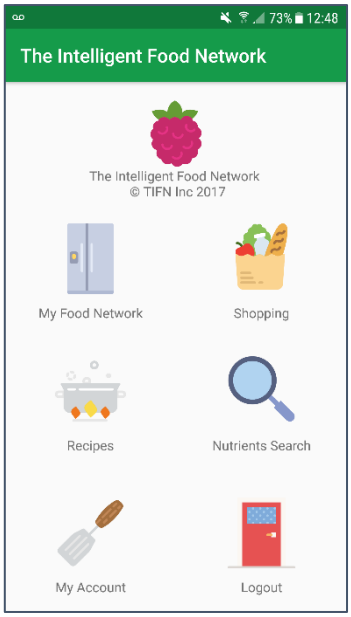


Figure 39 - Home

4.3.5 My Food Network Page

This GUI element is associated with **Functional Requirement No 4**.

The My Food Network activity allows users to check what food items they currently have in their home. This section is a key component to both the overall system and the Android Application. When a user enters this section, they will able to view items in their home based upon the data from the RFID tags. They can also filter these items based on category i.e. Dairy, Poultry, Fruit & Veg etc. This page displays crucial information i.e. food Description, Food Category, and Expiration Date. From here users can view the food contents in their home, filter by category, view recipes based on each item and also view nutritional data. If an item is added or removed from the Fridge or Cabinet, this section will dynamically update in real time. (Either adding or removing the item). This section was originally designed using a listview, although a custom gridview was added later throughout development as it was a better suited this UI section.

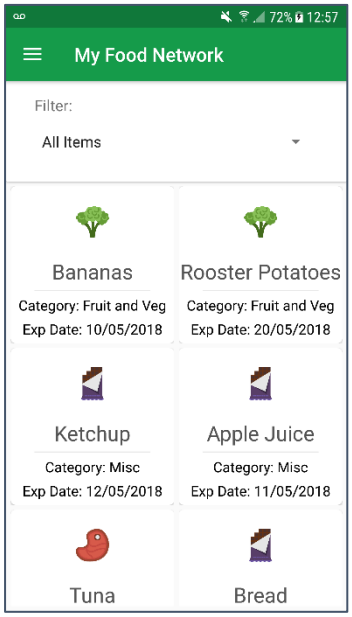


Figure 40 - Food Contents

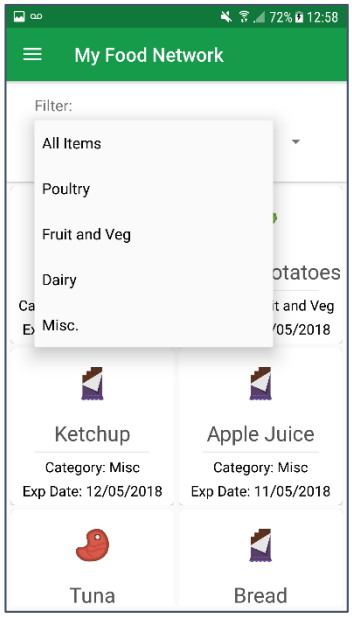


Figure 41 - Filter by Category

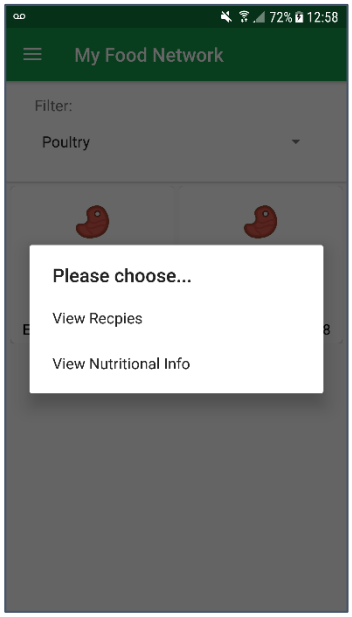


Figure 42 - Options Menu

4.3.6 Recipes List Page

This GUI element is associated with **Functional Requirement No 5**.

This UI is called upon from various sections in the application. It is used to display a listing of all recipes that meet the required criteria. All the information displayed in this activity is received directly from the Spoonacular API (Endpoint: Recipe via Ingredient). The example in the image lists all recipes that require the ingredient Tuna. When a user clicks the "***View Recipes***" option from the "***My Food Network***" activity they will be presented with this view. This GUI pulls data from the API and parses it into a custom listview via the custom listview adapter. The custom listview adapter and custom gridview adapters are used extensively throughout the application. The custom listview was designed so that the recipe title could be seen above the image. This was done using a relative layout which placed the black bar above the image. The opacity of the black bar was then changed to blend in better with the image. Finally, the text was placed above both the image and the black bar. This UI is also utilized for another API endpoint (Similar Recipe Results).

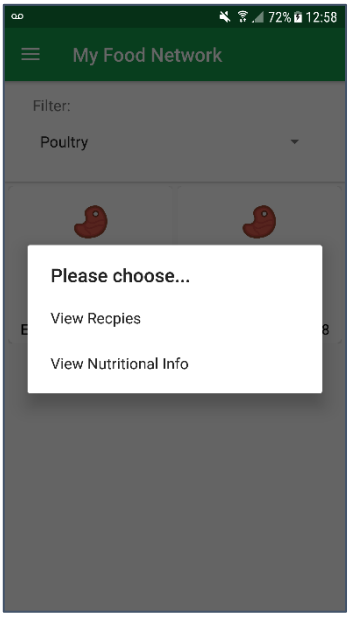


Figure 43 - View Recipes Option

Figure 44 - Recipes List

4.3.7 Recipe Details Page

This GUI element is associated with **Functional Requirement No 6, 7 & 8**.

The Recipe Details activity is used to display a selected recipe. This activity can be accessed from a various sections within the application. Users will be presented with all the details of the recipe they were recommended or clicked on. This section utilizes multiple custom listviews and a plethora of different buttons, menus and options. This is perhaps one of the most extensive sections within the entire application. All data presented in this view is pulled directly from the Spoonacular API (Endpoint: Recipe Details via ID). At the top of the UI is the cover image for the recipe. A comprehensive overview can be found below including, Recipe title, serving among, preparation time, cook time, ingredients and step by step instructions. The action bar is also effectively utilized, allowing users to share and favourite the recipe.

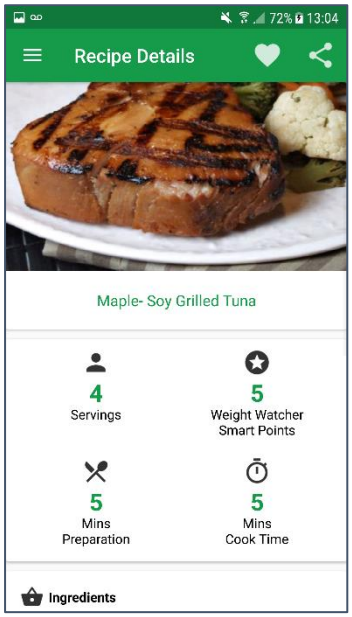


Figure 45 - Recipe Details

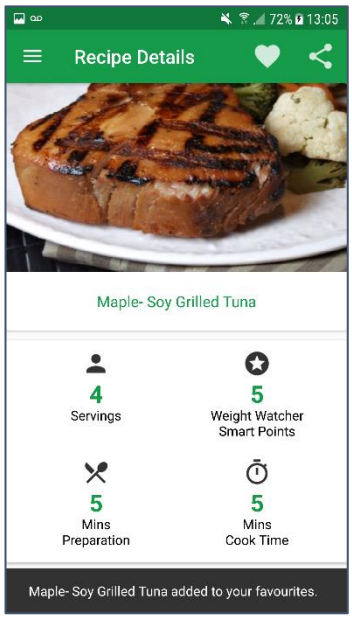


Figure 46 - Added to Favourites

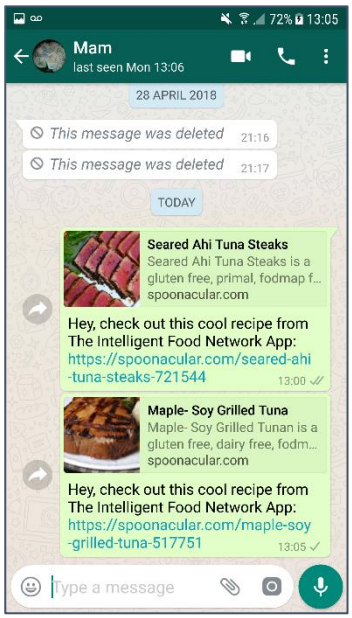


Figure 47 - Share via WhatsApp

Scrolling down to the ingredients, users will have the option to add to shopping list or view nutritional data (Nutritional data is pulled from the Nutritionix API). If a user already has an item in their shopping list, an icon will display at the side (A miniature shopping trolley). Users will not be able to add duplicate items to their favourites or shopping list and will be presented with a Snackbar error if they attempt to do so. If a user has an ingredient in their home, the ingredient will be highlighted green.

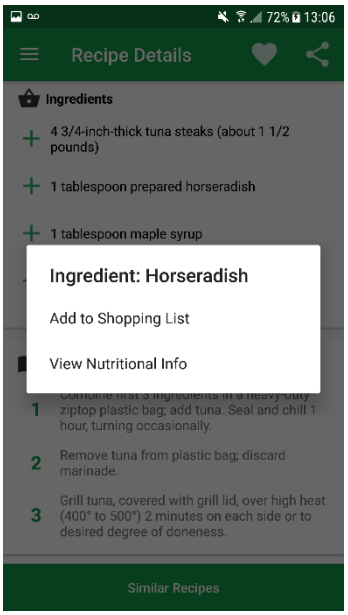


Figure 48 - Ingredient Options

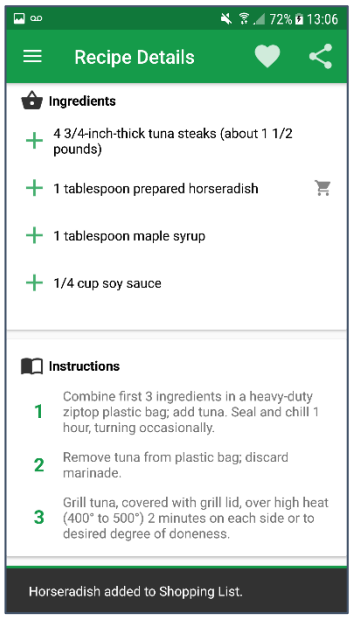


Figure 49 - Ingredient Added to Shopping

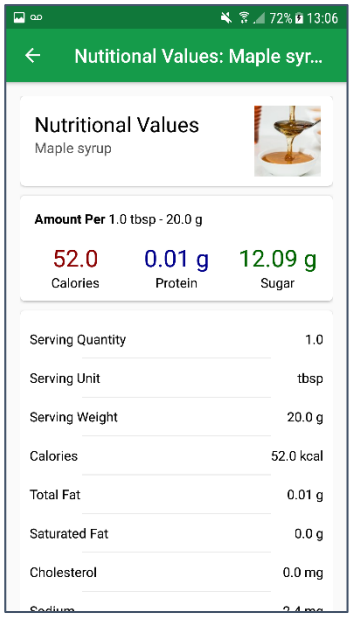


Figure 50 - Ingredient Nutritional Values

At the end of the view is also a button which enables the user to navigate to similar recipes. This data is once again pulled from the Spoonacular API (Endpoint: Similar Recipes by ID).

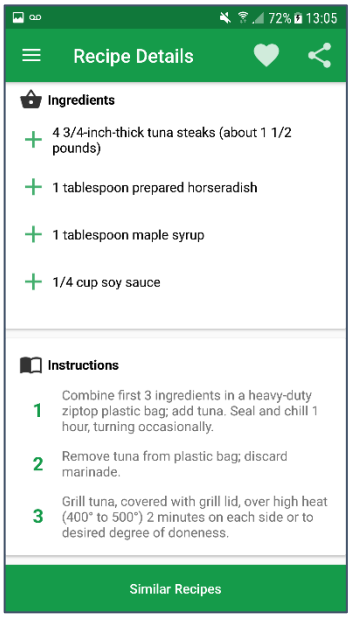


Figure 51 - Similar Recipes Option

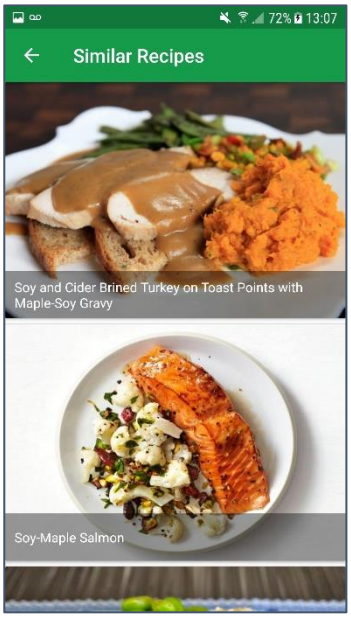


Figure 52 - Similar Recipes List

4.3.8 Shopping List Page

This GUI element is associated with **Functional Requirement No 6**.

The Shopping List activity allows users to store a weekly shopping list. This list can be populated manually using the add button at the top of the Action-bar or by adding ingredients through the "***Recipe Details***" activity. This UI is similar to the "***My Food Network***" activity as it utilizes a similar custom gridview. From here, users can add and remove items and also view nutritional data via the Nutritionix API. (Endpoint: Nutritional values via food name). The Android AlertDialog is utilized to add manually add the items. (This UI element can be seen in various section within the application).

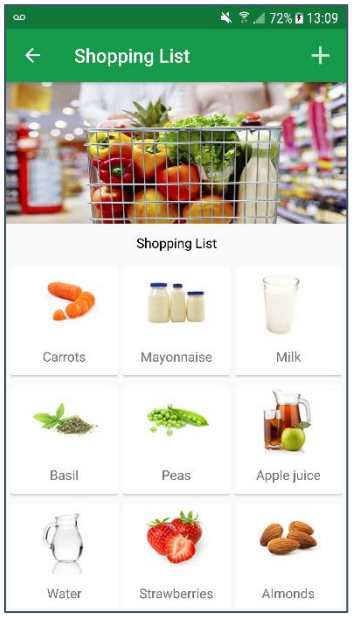


Figure 53 - Shopping List

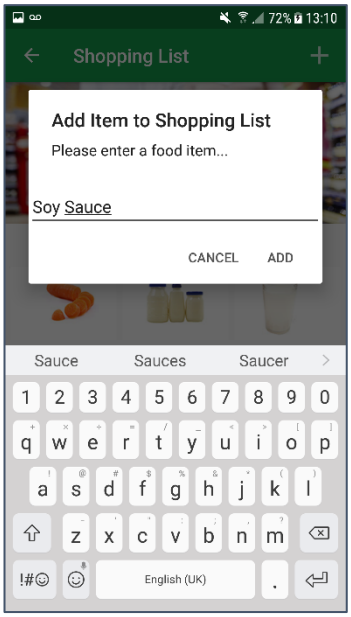


Figure 54 - Add Item to Shopping List

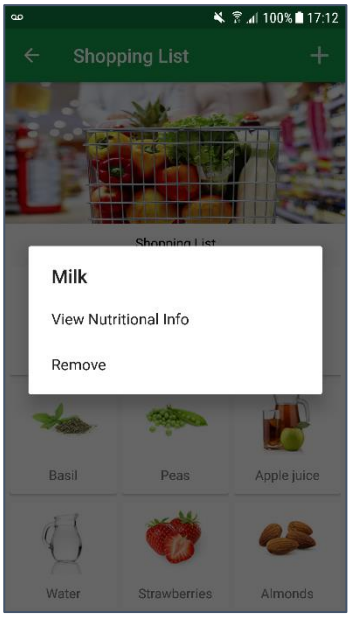


Figure 55 - Item Options

4.3.9 Shopping Recommendation Page

This GUI element is associated with **Functional Requirement No 11**.

The Shopping Recommendation activity is used to show users what they require on their next shopping trip. This section incorporates a simple listview which shows the results of the recommendation algorithm. The algorithm compares the users shopping list and the food contents within their home and then formulates an output. For example, if a user has Carrots in their fridge and Carrots on their weekly shopping list, the algorithm will not include carrots on this list. To contrast, if a user had carrots on their shopping list but did not have carrots in their fridge, carrots would be present on this list. This list will also dynamically update without refreshing or exiting the activity.

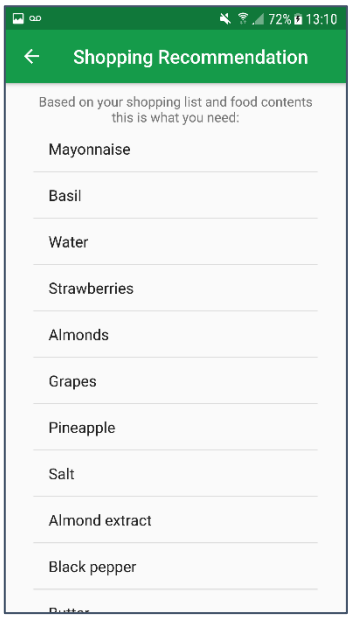


Figure 56 - Shopping Recommendation

4.3.10 Recipes/ Recipes Recommendation Page

This GUI element is associated with **Functional Requirement No 10**.

The Recipes activity allows users to interact with all the recipe functions the system provides. From here users can navigate to their favourite saved recipes or search for recipes via ingredient using the API (Endpoint: Recipe via ingredient). One of the most important components of this activity is the recipe recommendation. Each time this activity is loaded, users will be recommended a recipe. This recommendation is based upon a user's favourites. (Every user account will receive different recommendations as they will all have different saved recipes). The recommendation algorithm gets all the signed in user's favourites from the Firebase database. It then selects a random favourite from the list, grabs the selected favourite ID and then makes a call to the API. The endpoint: Similar recipes via ID is used to grab a list of similar recipes. A random recipe from this list is then selected and parsed into the cardview once the activity loads. The user may click on the recommendation and will then be presented with the "***Recipe Details***" UI (outlined previously above).

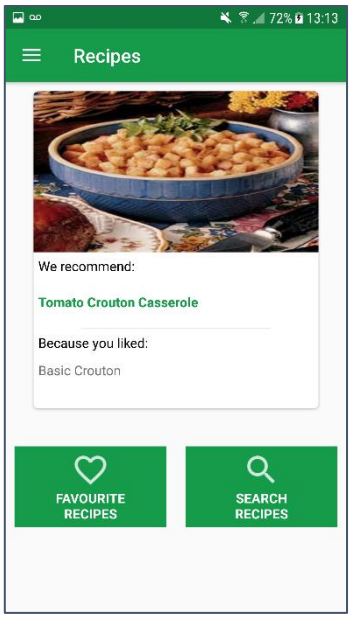


Figure 57 - Recipe Recommendation

4.3.11 Favourite Recipes Page

This GUI element is associated with **Functional Requirement No 9**.

The favourite recipes activity lists all the signed in user's favourite recipes. This activity makes use of a custom Gridview to display all the recipes. Each user created in the system will have different favourite recipes. Almost all recipe and food applications seem to use a grid of images with the corresponding text resting on top for this type of section. As such, this design choice was also chosen. Initially, this section was created using cardviews, although it was decided that a more "Flat" looking UI would be appropriate – one again to conform to Google Material Design Guidelines. A count of the total number of recipes can be seen at the top of this UI. A user may View or Remove any of the recipes in the grid. Once a recipe is added or removed, the counter will dynamically update. This counter may also be seen in the "User Profile" section of the application.

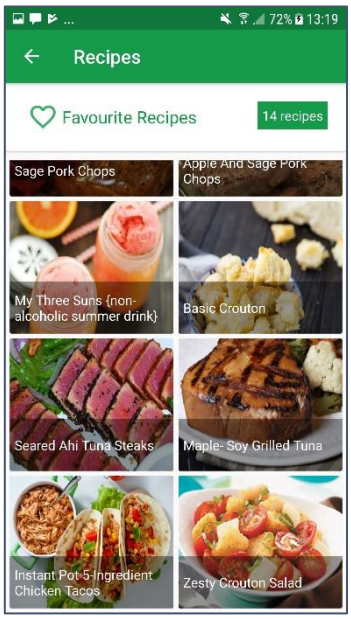


Figure 58 - Favourite Recipes

4.3.12 Recipes Search

From the Recipes section, the user can either view the recipe recommendation, view their favourites or search for a recipe. When a user searches for a recipe, an AlertDialog UI element will pop up. This will prompt the user to enter and ingredient to find a recipe based upon. This section of the Recipes activity once more utilizes the Spoonacular API (Endpoint: Recipe via Ingredient). Once the search is complete, the user will be presented with the same UI outlined above titled "***Recipes List***"

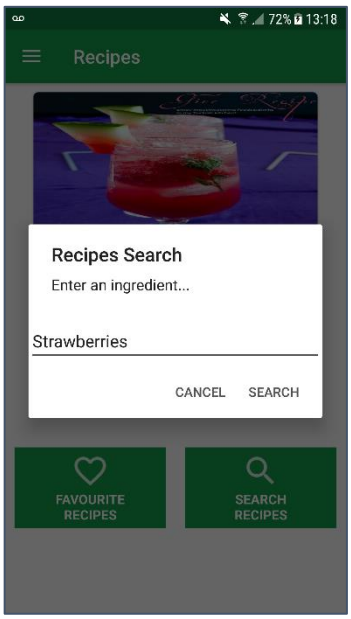


Figure 59 - Recipe Search

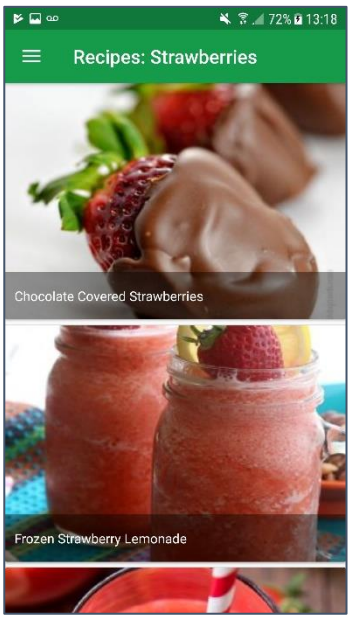


Figure 60 - Recipe Search Result

4.3.13 Nutrients Search Page

This GUI element is associated with **Functional Requirement No 12**.

The Nutrients Search page allows users to search for nutritional values based upon food name. This section utilizes the Nutritionix API (Endpoint: Nutritional values via Foodname). Once the request is sent to the API, the data received is passed to a custom listview. The Custom listview is then used to design the layout of this UI section. The API returns a significant amount of data back to the application; therefore, it was decided that this UI section be separated using Cardviews. This increased overall legibility. The listview was chosen for this section so it could be reused in other parts of the application. For example, the same Nutritional Values UI can be accessed through the "***My Food Network***", "***Shopping List***", and "***Recipe Details***" activities. Effectively, any food type's nutritional values can be accessed from almost every applicable section in the application.

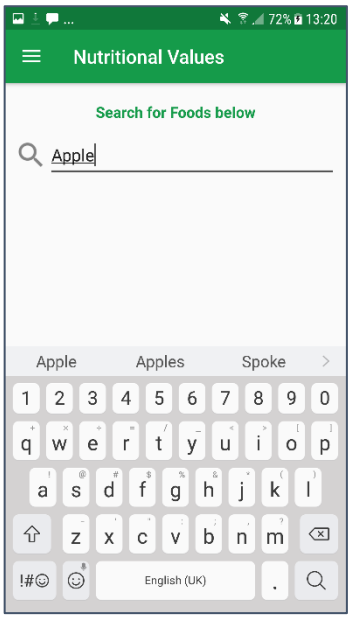


Figure 61 - Nutritional Search

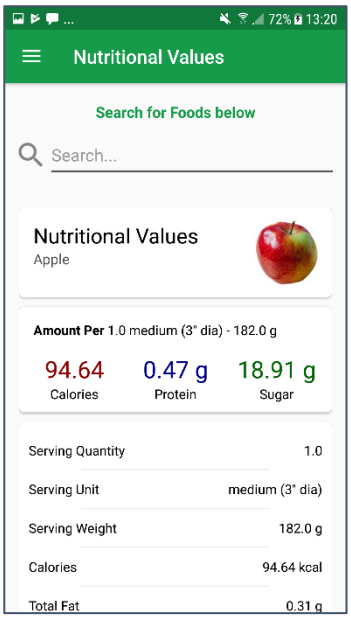


Figure 62 - Nutritional Search Result

4.3.14 User Profile Page

This GUI element is associated with **Functional Requirement No 13**.

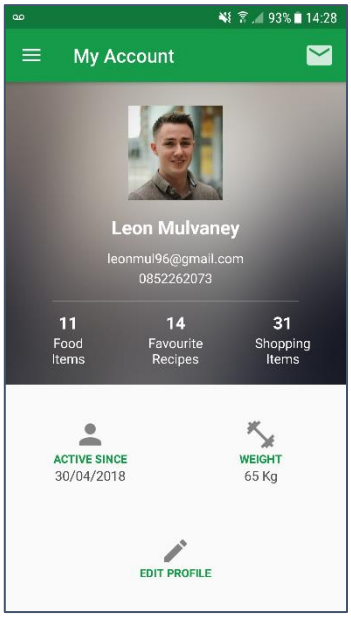


Figure 63 - User Profile

The "***My Account***" activity can be accessed via the homepage. This section allows the signed in user to view their profile. A grey gradient-blur image is used as the background for this section to provide the affect as seen in the screenshots. Built in Android icons were also used to give this section more character. The "View" Android UI element is also used here as a break (Designed to be very similar to a HTML HR) and can be seen just below the user phone number and just above the edit profile option.

The profile UI consists of a linear layout which displays:

- Profile Image
- Name
- Email
- Phone Number
- Number of Food Items in Home
- Number of Favourite Recipes
- Number of and Shopping List Items
- The date (Active Since) the user signed up to The Intelligent Food Network
- The User Weight (N.B this is an optional extra)

All data here is pulled from Firebase Authentication, Database and Storage. Users can also click the "Edit Profile" button which will bring them to the "***Edit Profile***" activity.

4.3.15 Edit Profile Page

This GUI element is associated with **Functional Requirement No 13 & 14**.

This activity inhibits similar design characteristics as the "***My Account***" activity. The same background is used, although it now fills the entire UI. From this section, users can make changes to:

- Their profile image via Android Device storage – by clicking on their current profile image
- Name
- Phone Number
- Weight (N.B this is still optional)

Once a user updates their profile, the changes will be applied instantly.

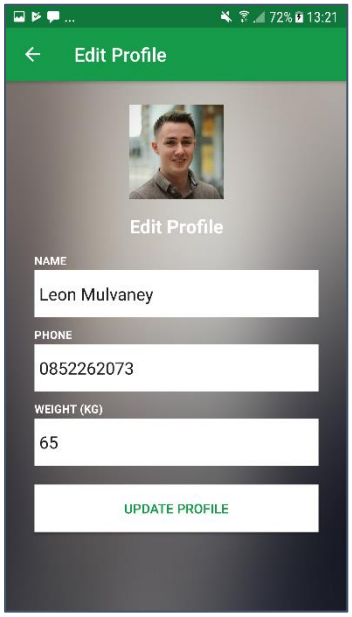


Figure 64 - Edit Profile

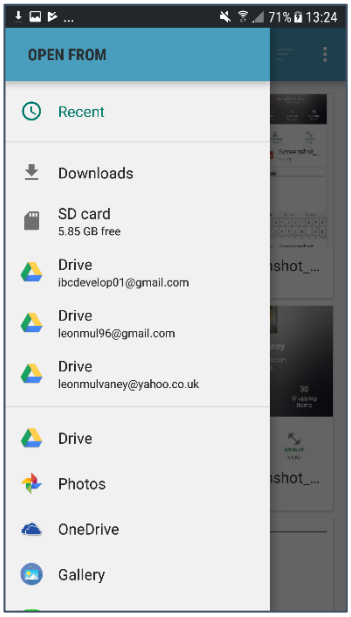


Figure 65 - Upload Image Dialog

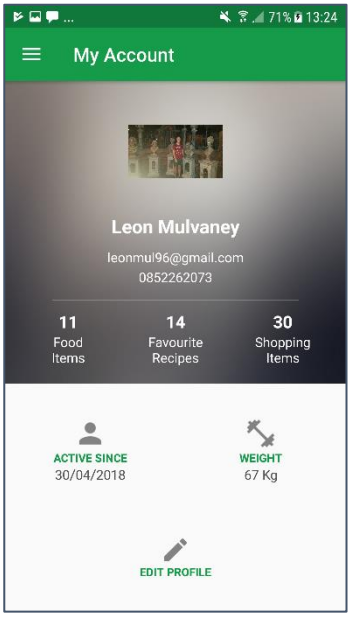


Figure 66 - Updated Profile

4.3.16 Contact Developers Page

Users are given the option of contacting the developers through the User Profile activity. This is an additional feature that is present on many commercial applications on the Play Store. Usually the option is located in the Hamburger Menu or App settings Activity. The Email icon at the top right corner of the Profile page is used to access the “Contact Developers” section. This section utilizes EditText Views to allow the user to input their query. Once complete, the user will be routed to select an email client they wish to submit the query through. The details are then grabbed and placed into a draft email.

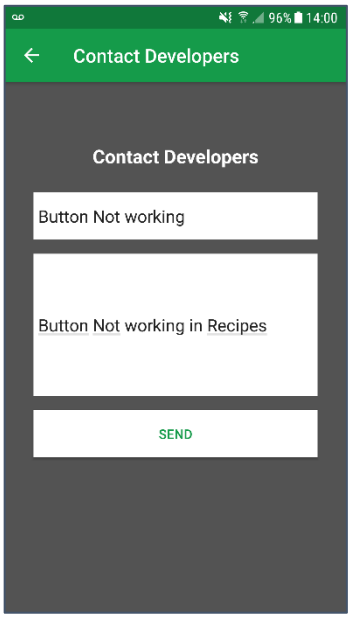


Figure 67 - Contact Developers

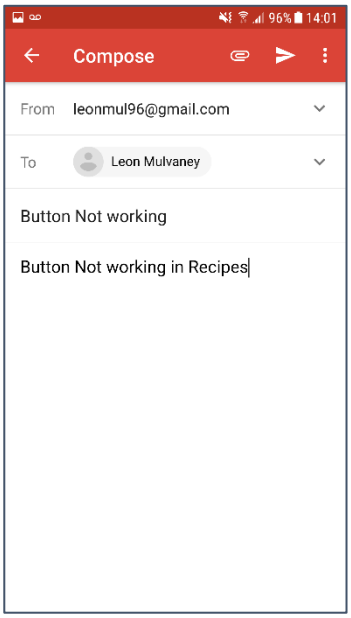


Figure 68 - Draft Email

4.3.17 New User Account – Empty Sections

The above GUI sections are all associated with an active user account. If a user has just signed up for an account, they will have no food items, favourite recipes or shopping list items. If this is the case, the application will apply specific UI defaults and notify the user via either the Android Snackbar or Toast. These defaults can be seen below:

Figure 69 - New Account Creation

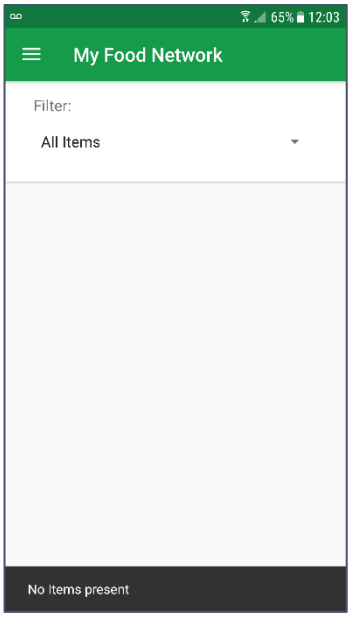


Figure 70 - No Food Items

Figure 71 - No Shopping List Items

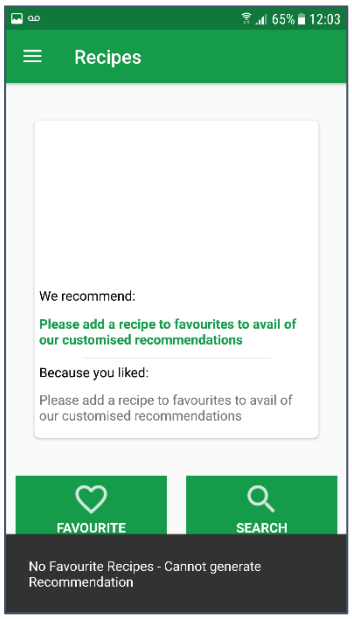


Figure 72 – Cannot Generate Recipe Recommendation

Figure 73 - No Favourite Recipes

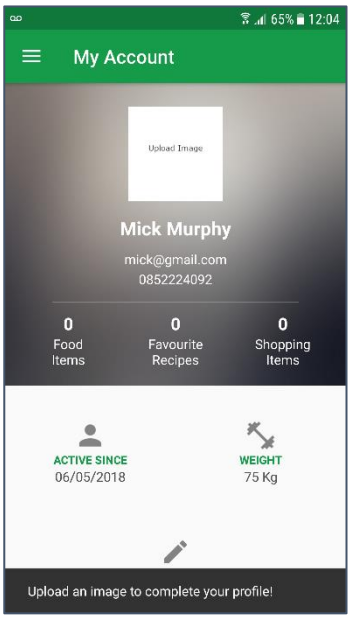


Figure 74 - Complete Profile Prompt

4.3.18 Raspberry Pi Scanning GUI

This GUI element relates to **Functional Requirements No 2 & 3.**

Although the end users will not see this section, it has still been included in the GUI. The scanning GUI is part of the Python script which waits for tags to come in contact with the RFID controller. Once the tags come in contact, the system will either add or remove the associated food items from the Firebase database. The output terminal has been configured to give the developer an overview of what changes have occurred. This can be useful in the future – especially for troubleshooting.

```
[u'-LBVQMXo8pRtGu8E5w80', u'-LBWf0-Uqvo379Tth3M3', u'-LBVRECcWTG0231DhX5P', u'-L
BVPD8wa1CZPpRzkEet', u'-LBVRDblyDcEeGFmWpAF', u'-LBWevhn9JREx8HMDQbE', u'-LBVRCu
OPj4Jve5R7duP', u'-LBVPEHoVDKH0XXeJUqO', u'-LBVQqMaeHGywawrl4cu', u'-LBVPCGFQm4I
aXTV_4Ba']
[431700793825L, 443142265157L, 1038012471643L, 640811882914L, 1055366688828L, 28
8037427540L, 942868945243L, 476814661996L, 901303339356L, 840665321767L]
Tag Id: 443008571717
Food Type: Milk
Expiry Date: 29/05/2018
Category:Dairy


No Match
No Match
No Match
No Match
No Match
No Match
No Match
No Match
No Match
No Match
Item Added to Firebase!
```

Figure 75 - Raspberry Pi Scanning GUI

5 Application Programming Interfaces

API's or Application Programming Interfaces were utilized extensively throughout the project. Initially, it was planned that only the Spoonacular API would be used, although throughout development the Nutritionix API was also included.

The Rapid API marketplace was utilized to analyse and test each API. The Rapid API Marketplace is a huge framework which encompasses thousands of API's in a single location. Rapid API allows developers to search through the extensive listing of available API's. Many are free including examples such as Google Maps, Facebook etc. whilst others may incur a monthly or yearly subscription fee. Once an API key has been received from the respective API owners, users can log in and link their newly generated API key to the Platform. Rapid API provides users with the ability to test each endpoint via the a in Terminal Window. The Terminal is broken into two sections – Sample Request and Response. The Sample Request section allows users to view API preview code in various languages including Java, Python and Ruby. The Response sections displays the result that the request will return.

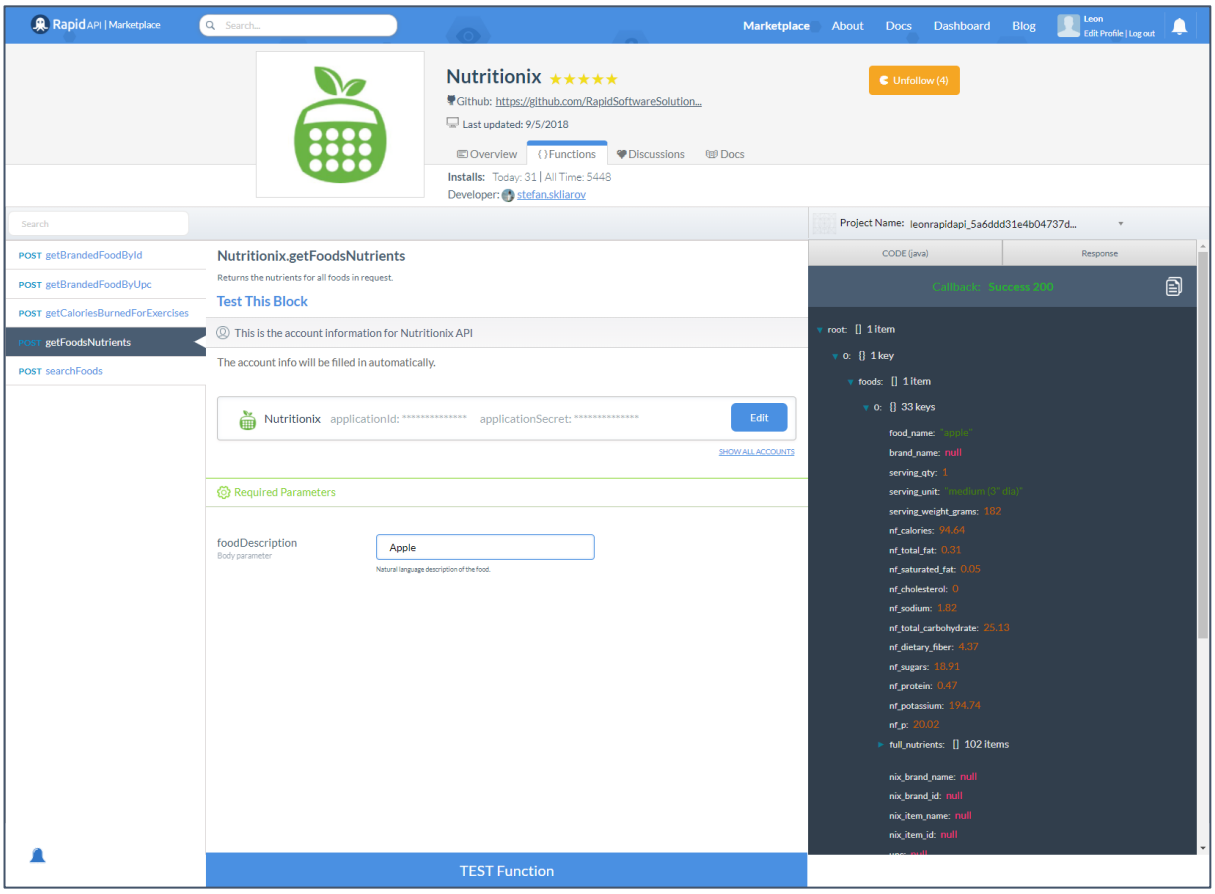


Figure 76 - Rapid API Terminal Window

5.1 Nutritionix API

The Nutritionix API was found via the Rapid API marketplace. The Nutritionix API is free for all users, therefore all that was required was to create a new account on the Nutritionix website, generate a new API key and link Rapid API to the newly generated account. The Nutritionix API was first tested using the Rapid API built in console before it was ported to the Android project. This negated the need to use an external source such as Postman for the API testing.

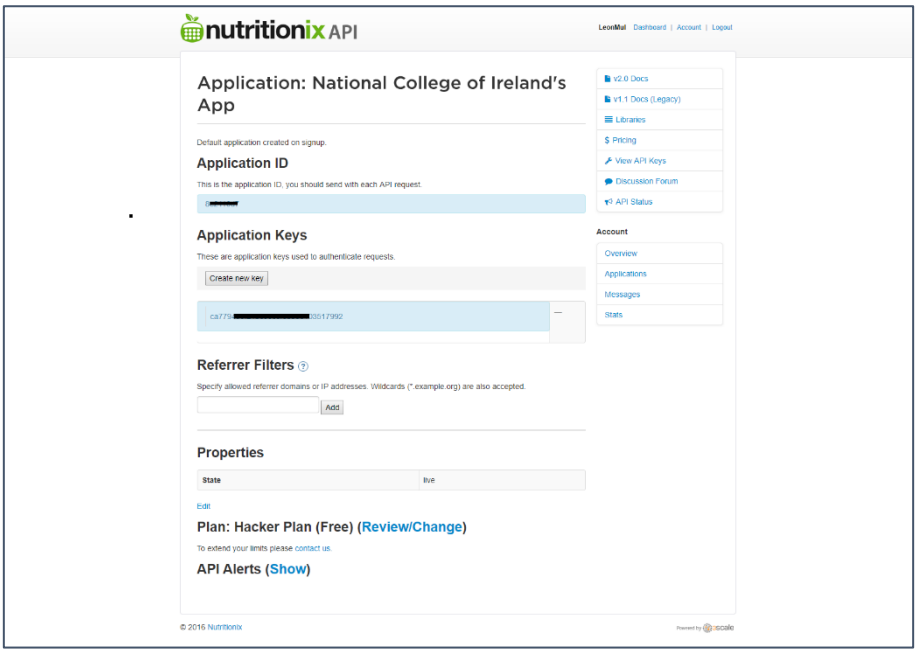


Figure 77 - Nutritionix API Keys

5.1.1 Endpoints Used

GET Nutritional Data via Food Name	
Description:	*This endpoint takes in a food name. Once the food name is received, it will return the corresponding Nutritional Values associated with that food type. I.e. the API will be passed "Apple", then return the Nutritional Data associated with the food "Apple".*
URL:	***http://rapidapi.io/connect/Nutritionix/getFoodsNutrients/{FoodName}***
HTTPVerb:	*GET*
Parameters:	*Food Name*
Response:	*The Android Application pulls the data from the API as a HashMap which is then saved as objects before parsing to UI.*

Figure 78 - Nutritional Data via Food Name Endpoint

5.2 Spoonacular API

The Spoonacular API was found online but was also present within the API marketplace. This was a great advantage as a single platform – Rapid API could be used to manage all project API's. Spoonacular is a massive food-oriented platform consisting of a huge database of recipes, ingredients, products and food menus. The API is very powerful and facilitates developers, students and anyone whom wishes to use it. The platform is used by various Corporations and Universities across the globe.

In contrast to the Nutritionix API, Spoonacular requires monthly instalments. The Co-Founder, Crystal Schlegelmilch was contacted via email requesting Student access to the platform. After some emails back and forth, a confirmation email providing integration with Spoonacular and Rapid API was received. The price for API usage was $10 per month. The Student plan was much better than even the basic plan - offering up to 5000 daily requests as opposed to 500. The basic plan was also 3 times more expensive costing $30 instead of $10. Once the confirmation email was received, the Spoonacular API was linked to the Rapid API account. The endpoints were then tested using the built in Rapid API Terminal.

Figure 79 - Spoonacular API Overview

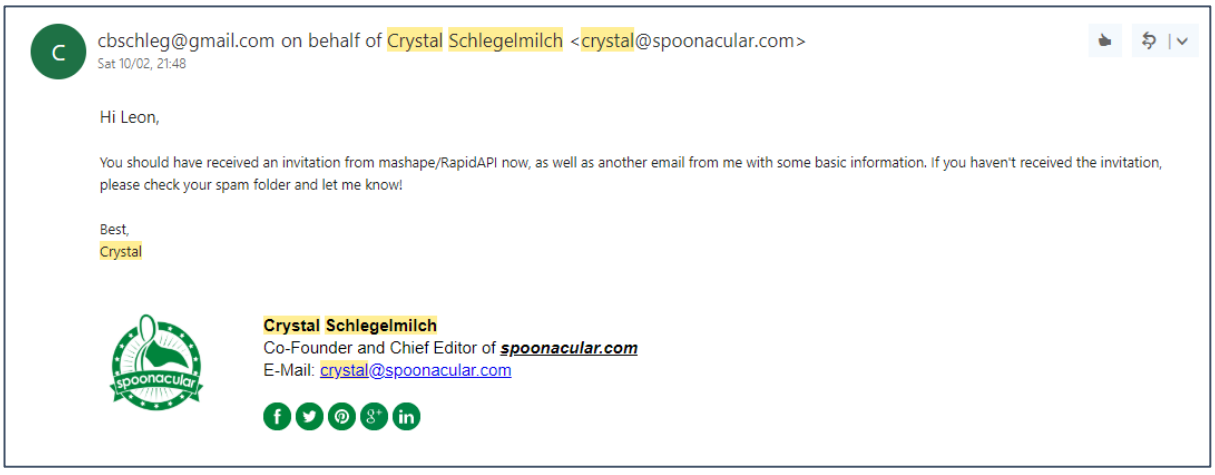


Figure 80 - Confirmation of API Subscription

5.2.1 Endpoints Used

GET Recipes via Ingredient	
Description:	*This endpoint takes in an ingredient i.e. Chicken and returns a list of recipes that can be prepared using this ingredient. The number of recipes can be limited via the URL.*
URL:	***https://spoonacular-recipe-food-nutrition-v1.p.mashape.com/recipes/findByIngredients?fillIngredients=false&ingredients={ingredient}&limitLicense=false&number={numberOfRecipes}&ranking=1***
HTTPVerb:	*GET*
Parameters:	*Ingredient, NumberOfRecipes*
Response:	*The Android Application pulls the data from the API into a JSON Formatted String. The data is then saved to objects before it is parsed to the UI.*

Figure 81 - Recipes via Ingredient Endpoint

GET Recipe via ID	
Description:	*This endpoint takes in a Recipe ID and then returns all the Recipe details associated with that ID.*
URL:	***https://spoonacular-recipe-food-nutrition-v1.p.mashape.com/recipes/{recipeId}/information***
HTTPVerb:	*GET*
Parameters:	*recipeId*
Response:	*The Android Application pulls the data from the API into a JSON Formatted String. The data is then saved to objects before it is parsed to the UI.*

Figure 82 - Recipe via ID Endpoint

GET Similar Recipes via ID

Description:	*This endpoint takes in a Recipe ID and then returns a list of similar Recipes associated with the ID.*
URL:	***https://spoonacular-recipe-food-nutrition-v1.p.mashape.com/recipes/{recipeId}/similar***
HTTPVerb:	*GET*
Parameters:	*recipeId*
Response:	*Android Application pulls the data from the API into a JSON Formatted String. The data is then saved to objects before it is parsed to the UI.*

Figure 83 - Similar Recipes via ID Endpoint

6 System Architecture

This section will outline the fundamental system architecture that has been implemented to parallel the initial project aims.

6.1 Class Diagram

The final system design incorporates a range of different sections; therefore, it has been segregated into 4 primary categories.

- **Raspberry Pi**
- **Firebase**
- **Android Application**
- **API**

The class diagram encapsulates all the fundamental aspects of the System - The diagram shows the degree of association between each section, how they are linked, and the variables and methods associated with each class. Each section was allocated its own colour coded box (labelled up the top left-corner). This Architecture was derived from other IOT Architectures that feature similar components and hardware such as the RFID Controller and Raspberry Pi. The Android Application was designed to work in parallel with the Firebase platform - right from initial start-up the Application interacts with the Firebase back-end in some way or form. Therefore, instead connecting each individual Android class to the Firebase components - the Entire Android Application container has been connected to represent inheritance.

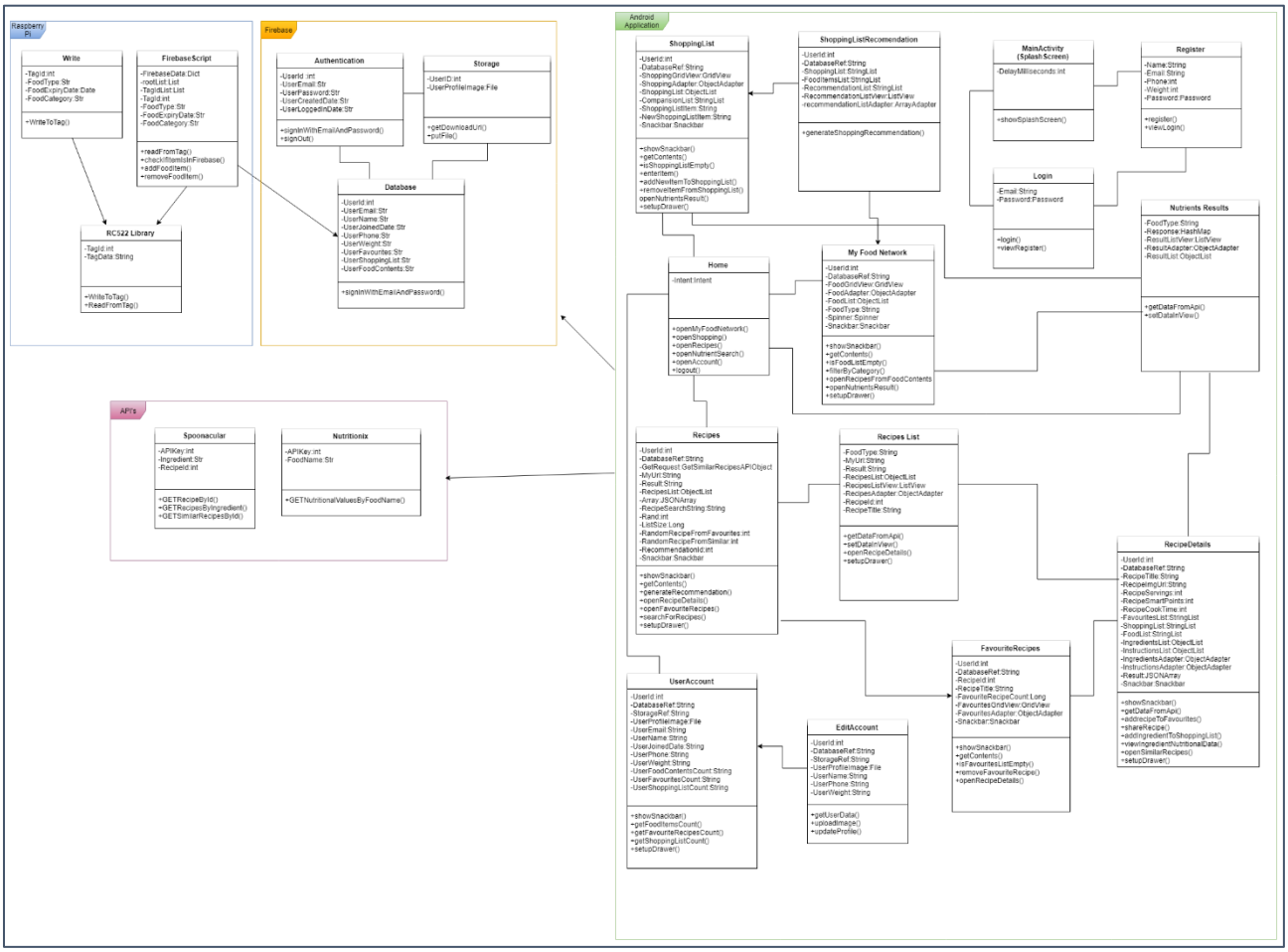


Figure 84 - System Class Diagram

6.2 System Architecture Diagram

A System Architecture Diagram has also been developed in conjunction with the Class Diagram. Each individual functionality will be outlined in a further section titled "***Implementation***".

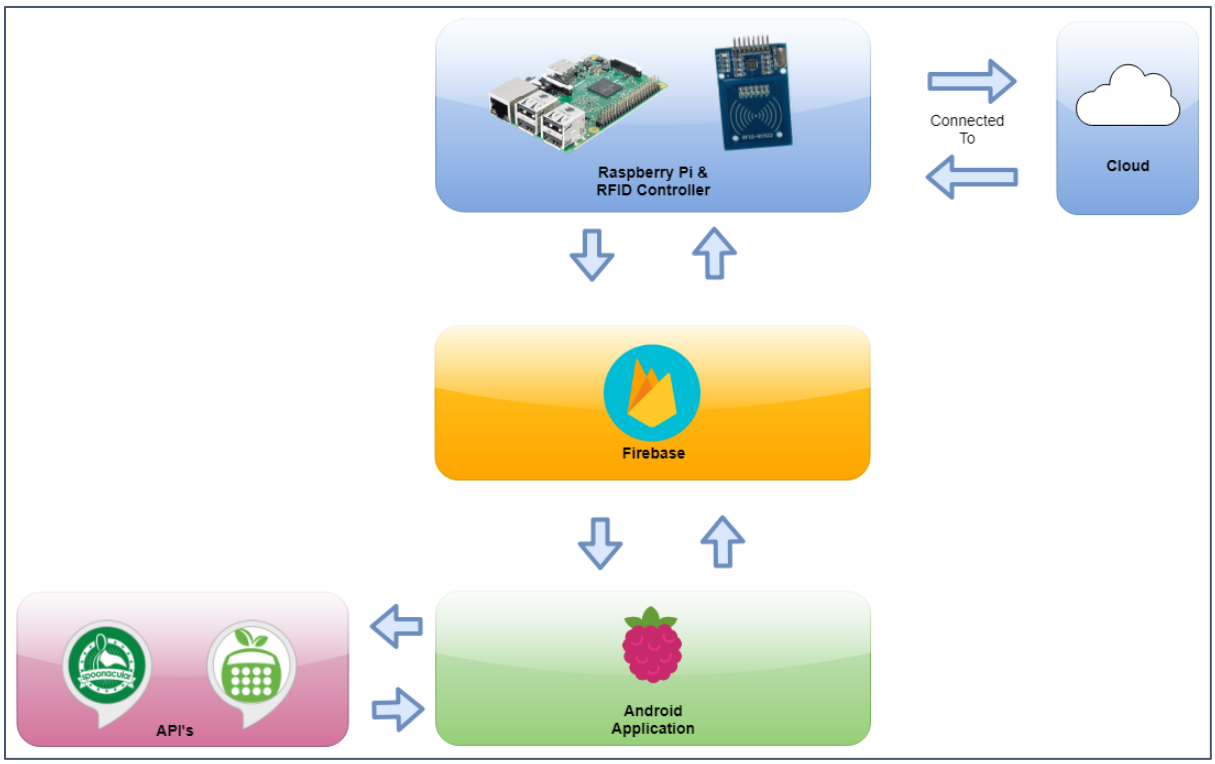


Figure 85 - System Architecture Diagram

7 Implementation

This section outlines the primary system functionalities, how they were developed, the underlying requirements and any software and hardware components that were utilized throughout development. The system was developed across the entire academic year - therefore there has been many changes and alterations to code and underlying methodologies. There are 4 primary development sections that the system is separated into:

1. Firebase
2. Raspberry Pi
3. Android Application
4. API's

7.1 Firebase

As outlined in the previous sections, Googles Firebase platform was selected as the primary form of system data storage. Almost every section in the application utilizes the Firebase platform in some way or form. Firebase is composed of a plethora of different branches - although the primary branches utilized for this project were:

- Firebase Authentication
- Firebase Database
- Firebase Storage

7.1.1 Firebase Authentication

Firebase Authentication is utilized as soon as the user logs in or creates an account. The Authentication branch is used for secure login and access to user account data. Only logged in users may access any system data - with the data limited to their own account. Separation of profiles is crucial as each user will have different items in their home, favourites and shopping lists etc. Upon registration, users are required to enter some mandatory details including Name, Email, Phone and Password. The option to enter weight is also available - although is optional and may be left blank or changed at any time after registration via the "***Edit Profile***" activity. After successful registration, the Firebase Authentication branch will generate a new unique id for the user. This unique id is vital as it is used extensively throughout the application to link users to their Database files (Outlined in the following section "***Firebase Database***"). The Firebase platform supports a very useful Administrator GUI which enables admins to view an overview of the many branches it supports. The Authentication branch in particular allows the admin to view all users connected to the application, their associated user id and other information such as email, account creation date and last login date.

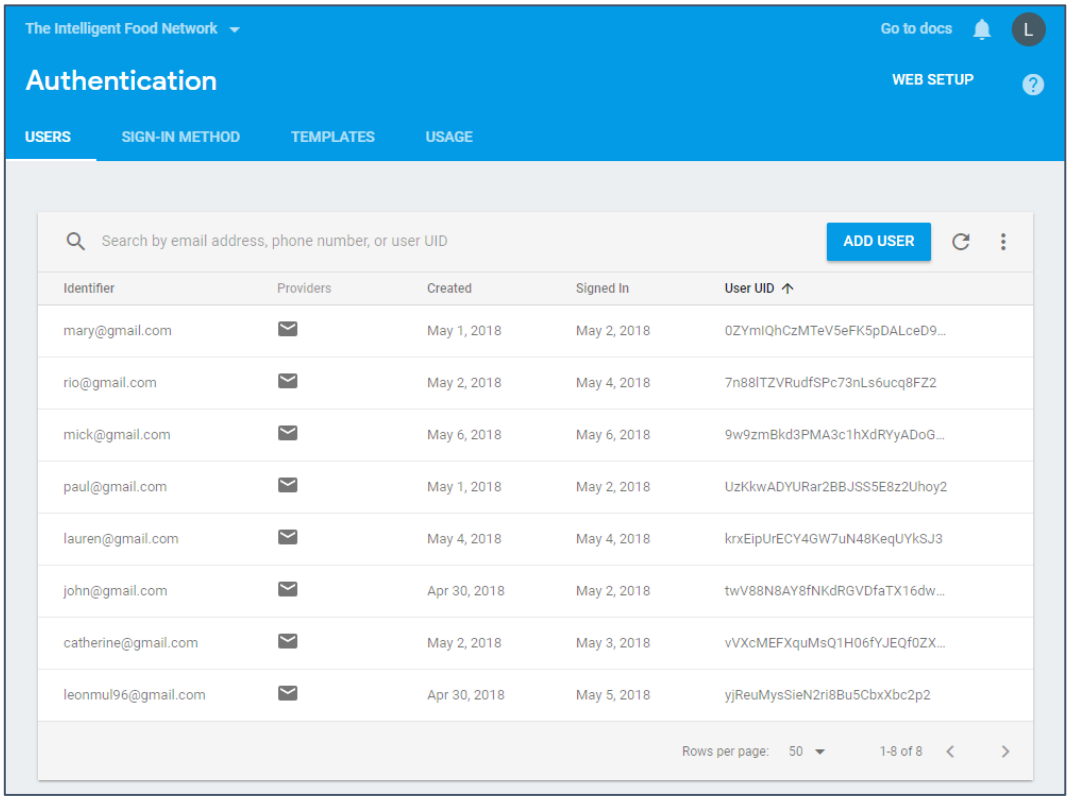


Figure 86 - Firebase Authentication Admin Page

7.1.2 Firebase Database

The Firebase Database is the primary form of data storage for the system. Firebase Database was chosen as it is very lightweight and provides native support in Android Studio. It also inhibits Real-Time functionality which was a crucial design element that ideally suited the project. Firebase Database is a noSQL database, it uses JSON to hold objects and data in contrast to SQL. This made the database even easier to develop with – negating the need for an SQL server or equivalent.

As described in the above section "***Firebase Authentication***" - when a user creates a new account, the Authentication branch will generate a unique user id. On successful registration, this user id is also utilized to create a new entry for the user in the Firebase Database. The new entry is created within the "***Users***" table with the root node being the newly generated user id. Everything linked to each user account can be found within this newly generated entry/object. Initially, the only details that will be present are the user registration details, although favourite recipes, food items and shopping list items may be populated and will update as the user uses the application.

The Firebase Authentication branch can also be utilized to store similar user information such as Name, phone etc. although storing all user data under the one root node in the database seemed a more appropriate design choice. Ultimately, this meant that the Authentication branch could be used for the login/registration and the Database branch for any associated user data. Of course, passwords cannot be accessed by the admin and are not saved in the database.

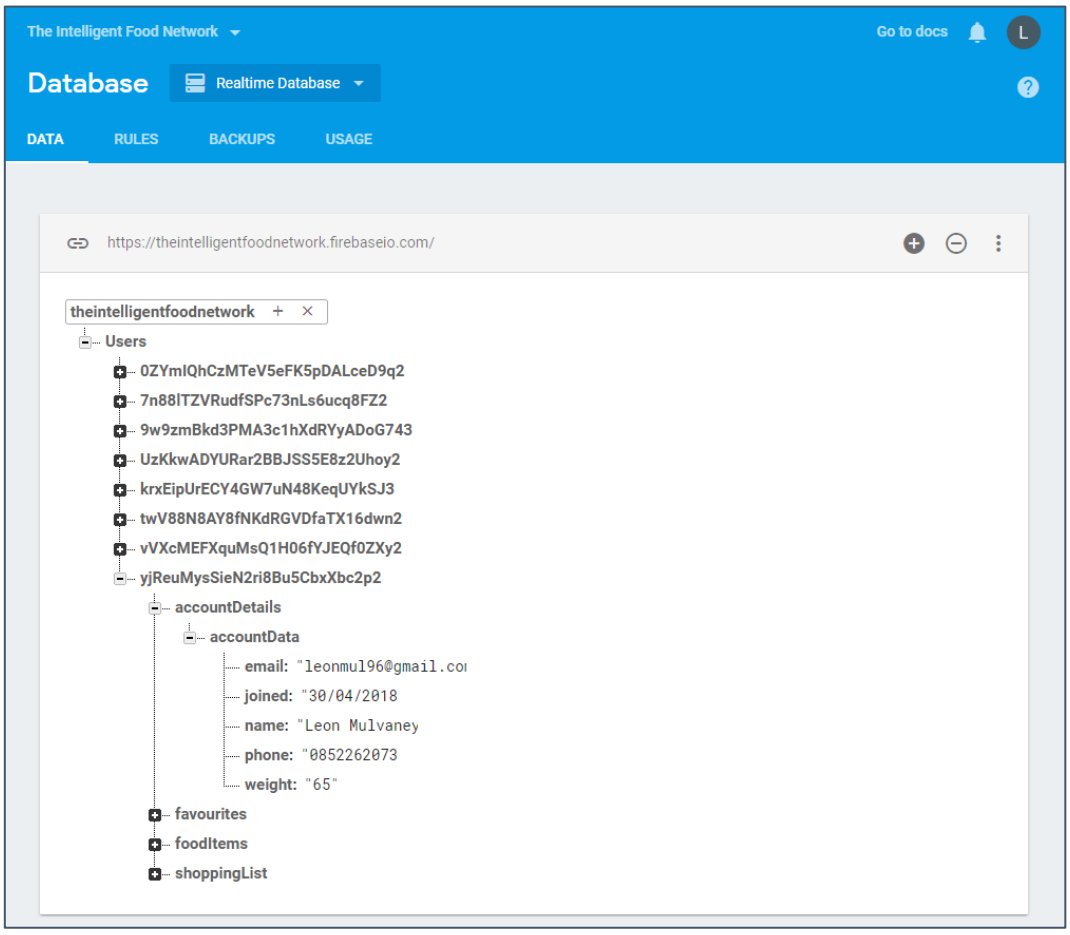


Figure 87 - Firebase Database Admin Page

7.1.3 Firebase Storage

The final branch of the Firebase framework to be utilized was Firebase Storage. Firebase storage inhibits the capabilities of holding a plethora of file types including images, videos, music, pdfs and much more. The Firebase storage branch was specifically used within the application to hold user profile images. When a user creates a new profile, they may navigate to the "***Edit Profile***" activity to upload a profile image and complete their profile. This image will then be stored in Firebase Storage. The unique user id is also used here to generate a new folder to save the profile image into. For example, the default path for a user profile image in Firebase Storage is as follows:

profileImages/"userId"/profileImage.jpeg

If the user has just recently registered for a new account or they have not yet uploaded a profile image, a default image - "*profileImages/uploadImage.png*" will appear. This is set as the default until a change is applied. Users may update their user profile image as much as they require – with the changes being applied instantly.

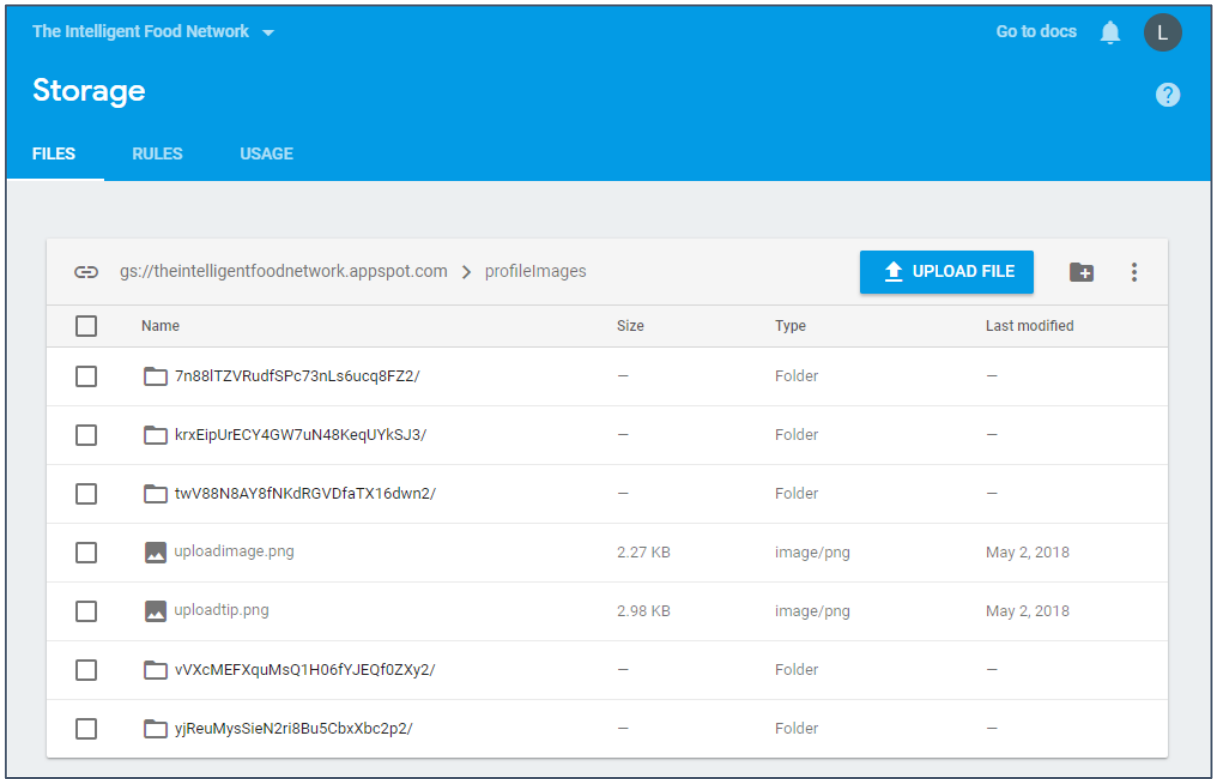


Figure 88 - Firebase Storage Admin Page

7.2 Raspberry Pi

The Raspberry Pi is one of the most crucial components of the entire system. The Raspberry Pi acts as a mediator between the RFID tags, the cloud Database and the Android Application. The Pi was an integral inclusion within development as without it, the entire concept could have been negated. The Pi utilized in this project was the "Raspberry Pi 3 Model B". This microcontroller features a Quad core ARM CPU clocked @ 1.2Ghz accompanied by 1GB DDR2 RAM. Wi-Fi and Bluetooth are build in as standard. A Micro USB cable is used to power the Pi. There is no onboard storage – instead the device supports Micro SD cards.

Figure 89 - Raspberry Pi 3 Model B

7.3 RFID Controller

The RIFD controller of choice was the model "***Mifare RC522***". This particular controller is part of many starter kits and is very accessible to beginners whom are new to the concepts of RFID and NFC. The Controller communicates with RFID chips via the 13.56 Mhz Frequency band.

Figure 90 - RC522 RFID Controller

7.4 Raspberry Pi Code

7.4.1 Write to Tags

Associated Files

Python
Write.py

Code Snippet

```
import RPi.GPIO as GPIO
import SimpleMFRC522
reader = SimpleMFRC522.SimpleMFRC522()
while True:
    try:
        foodType = raw_input('Food Type: ')
        expiryDate = raw_input('Expiry Date: ')
        category = raw_input('Category:')
        data = str(foodType+","+expiryDate+","+category+",")
        print ("Now place your tag to write")
        reader.write(data)
        print("Written")
        id,text = reader.read()
        print(text)
        print "--------------------------"
        print "---------New Item---------"
        print "--------------------------"

    finally:
        GPIO.cleanup()
```

Figure 91 - Write to Tag

Overview

When the RFID tags were purchased, they contained no data. After some research, a library called "***SimpleMFRC522***" was found. This library effectively acted as an API between the Raspberry Pi Python code and the RFID controller. This library was first imported into the "***Write.py***" script. Once imported the data could be written to the tags. The library supports various methods, although the two that were most relevant to the project were "***Read()***" and "***Write()***".

When the Script is run, the user will be prompted to enter the data that should be written to the tag. In this case, the command terminal would first prompt the user to enter the "***Food Type***", followed by the "***Expiry Date***" and finally the "***Category***". This is still a Proof of Concept project; therefore the tags must be manually written with information. The project assumes that in the future, manufactures will make barcodes redundant, and replace them with RFID or NFC technologies. Instead of barcodes being written within the manufacturing plant, RFID tags will instead be written and placed onto the corresponding food items before they are shipped to the Shopping Markets.

Once the user has finished inputting the details, the details will be written to the tag in a comma separated fashion. The library only supports the writing and reading of Strings from the Tags. To combat this issue, the values are first saved as a single comma separated string during "Writing" stage. They can then be pulled back out into individual variables during the "Reading" stage. Finally, the result will be printed to the screen.

7.4.2 Read from Tags

Associated Files

Python
Read.py

Code Snippet

```
import RPi.GPIO as GPIO
import SimpleMFRC522
import time
reader = SimpleMFRC522.SimpleMFRC522()


while True:
    try:
        id,text = reader.read()
        print("TAG ID: " + str(id))
        foodType,expiryDate,category,empty= text.split(",")
        print("Food Type: " + foodType)
        print("Expiry Date: " + expiryDate)
        print("Category: " + category)
        print("\n")
        time.sleep(3)


    except KeyboardInterrupt:
        GPIO.cleanup()
```

Figure 92 - Read from Tag

Overview

The “Read.py” Python file is simply used to test the tags to ensure they are correctly reading. This file once again imports the “***SimpleMFRC522***” library, except it calls upon the “***read()”*** method**.** Once the values are read from the tag, they are split into separate variables based on the order they were written. A redundant variable called “empty” was used here to combat an issue with the separation process (The “***read()***” method was adding the last comma to the “category” variable). Once the tag has been successfully read, the system will wait for 3 seconds before reading another tag. This gives the RFID controller time to re-initialize.

7.4.3 Add & Remove Tag Data from Firebase

Associated Files

Python
FirebaseScript.py

Overview

The "***FirebaseScript.py***" is one of the most important files in the entire project. It allows the Tags to be added and removed from the Firebase Database.

First, the required imports are declared and instantiated. The Firebase database is then targeted. The userId associated with the Pi must be declared at the top of the file. (This tells the Pi which user the it belongs to – As part of the idea, the RFID Controller will be mounted in a user's cabinet or Refrigerator). A while True loop is then called to infinitely loop and listen for changes.

Code Snippet 1

```
from firebase import firebase

import json

import RPi.GPIO as GPIO

import SimpleMFRC522

import time


firebase =
firebase.FirebaseApplication('https://theintelligentfoodnetwork.firebaseio.com/',None)

reader = SimpleMFRC522.SimpleMFRC522()

userId = "yjReuMysSieN2ri8Bu5CbxXbc2p2"
```

Figure 93 - Imports and Firebase Declaration

Next, the users "***foodItems***" table is pulled from the Firebase Database and saved.

Code Snippet 2

```
#Get the values from foodItems and save into dict

 result = firebase.get('Users/'+userId+'/foodItems',None)
```

Figure 94 - Pulling from Firebase

If the "***food Items***" dictionary is blank, then the system will simply listen for a new tag to be passed to the controller. The system knows that the users table is empty and that there is no possibility of duplicate tags, as such the next item to be scanned will immediately be added to the Database.

Code Snippet 3

```
    if bool(result) == False:
        print "No Items linked to user account"
        id,text = reader.read()
        foodType,expiryDate,category,empty= text.split(",")
        print ("Tag Id: " + str(id))
        print("Food Type: " + foodType)
        print("Expiry Date: " + expiryDate)
        print("Category:" + category)
        print("\n")
        firebase.post('Users/'+userId+'/foodItems',{'tagId':id,
                            'foodType':foodType,
                             'expiryDate':expiryDate,
                             'category':category})
        print "Item Added to Firebase!"
```

Figure 95 - If Table is Blank

If the "***foodItems***" dictionary is not empty, then the system must check and see if a Tag ID already exists in the Database. When a tag is scanned via the RFID controller, two lists are generated. The first list called "***rootList***" is used to store all the unique "***foodItem***" root ids. The next list "***tagIdList***" is used to store all the tag ids currently in the Database. The lists are populated in the same order, i.e. element at [2] in the "***rootList***" corresponds to element at [2] in the "***tagIdList***".

The "***tagIdList***" is then iterated, with the currently scanned tag id being compared to each element in the list. If it matches any element, the location of the match is saved and the "***action***" variable is declared as "***Match***". The match location is used to grab the corresponding root from the "***rootList***". The root element is then removed from the Firebase Database using the "***delete()"*** function. Python Keys and root elements cannot be accessed via a value. This is the reasoning behind the generation of the two lists. (i.e. when iterating through the list, the only value that is grabbed is the value "tagId"). If the currently scanned tag id does not match any tag ids in the "***tagIdList***", then the item is not in the Database and can be added using the Firebase "***post()***" function. N.B Scanning GUI can be seen in the previous section titled "***Raspberry Pi Scanning GUI***".

Code Snippet 4

```
        for x in range(0,len(tagIdList)):
            listTagId = tagIdList[x]
            if id == listTagId:
                print "Match"
                action = "remove"
                firebaseRoot = rootList[x]
            else:
                print "No Match"
        if action == "remove":
            print "Item Removed"
            firebase.delete('Users/'+userId+'/foodItems',firebaseRoot)
        else:
            firebase.post('Users/'+userId+'/foodItems',{'tagId':id,
                                'foodType':foodType,
                                 'expiryDate':expiryDate,
                                 'category':category})
            print "Item Added to Firebase!"
```

Figure 96 - Tag Comparison Method

7.5 Android Application

The Android Application is the most extensive section of the entire project. The application ties all sections together and is the primary entry point between the user and the IoT system. As expected, most of the development time was allocated to development of the application. There were many changes, design choices and issues which arose during development. All UI elements described in this section can be seen in the previous listing "***GUI***". The below outlines the final results derived from the development process.

7.5.1 Splash Screen Activity

Associated Files

Java	XML
MainActivity.java	activity_main.xml

Code Snippet 1

```
public class MainActivity extends AppCompatActivity {

    private static final int DELAY_MILLISECONDS = 3000;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        //Disable Action bar From:
https://stackoverflow.com/questions/8456835/how-to-disable-action-bar-
permanently
        ActionBar actionBar = getSupportActionBar();
        actionBar.hide(); //Hide Action Bar On Splash Screen

        showSplashScreen(); //Call the Splash Screen Method
    }

    public void showSplashScreen(){
        //SplashScreen from Lecture Notes Dr Anu Sahni--------------
        Handler handler = new Handler();
        handler.postDelayed(new Runnable() {
            @Override
            public void run() {
                Intent intent = new Intent(MainActivity.this, Login.class);
                startActivity(intent);
                finish();
            }
        }, DELAY_MILLISECONDS);
    }
```

Figure 97 - Splash Screen

Overview

The Splash Screen is a vital UI element that is present in almost all high-end application on the Google Play and Apple App Store. The Splash Screen has been set as the "***MainActivity*** "as this is the first activity that loads when the application is launched. (Specified in the "***AndroidManifest.xml***" file). The splash screen method has been declared within the class and is called upon at the end of the "***OnCreate()***" method. The action bar has also been programmatically hidden for this view. The splash screen lasts for approximately 3 seconds – declared as "***DELAY_MILISECONDS"*** at the top of the class. Once the 3 seconds has elapsed, the "***Login***" activity will load and the Splashscreen will terminate using the "***finish()"*** command.

7.5.2 Login Activity

Associated Files

Java	XML
Login.java	activity_login.xml

Code Snippet

```
        //Pass the parameters email and password to the function
mAuth.signInWithEmailAndPassword(email, password)

            .addOnCompleteListener(this, new OnCompleteListener<AuthResult>() {
                public static final String TAG = "";
                @Override
                public void onComplete(@NonNull Task<AuthResult> task) {
                    if (task.isSuccessful()) {
// Sign in success, update UI with the signed-in user's information
                        Log.d(TAG, "Successful Login. \n Welcome " + email);
//Welcome the user on successful login
                        FirebaseUser user = mAuth.getCurrentUser();
                        Intent intent = new Intent(Login.this, Home.class);
                        startActivity(intent);

                    } else {
                        // If sign in fails, display a message to the user.
                        Log.w(TAG, "signInWithEmail:failure",
task.getException());
                        Toast.makeText(Login.this, "Login failed.",
Toast.LENGTH_SHORT).show();
                    }
                }
            });
}
```

Figure 98 - Login

Overview

The login activity is loaded directly after the Splash Screen timer has elapsed. The Login activity allows users to Login to their account using the credentials Email and Password. Users may also navigate to the Registration section at the bottom of the UI. This class creates an instance of Firebase Authentication and establishes a connection with the associated Firebase Authentication branch once the user attempts to login. The primary method here is "***Login***", which utilizes the built in classes supported by Firebase Authentication. Once a user enters their credentials and attempts to login, their inputs will first be checked to see if they meet the underlying credential criterial i.e. if an email entry does not contain an @ character or if password is blank then the user will be prompted to enter the correct credentials.

If these credentials meet the required criteria, a connection with Firebase will then be established to sign the user into the app. The Firebase function "***signInWithEmailAndPassword()***" is called and is passed the parameters "***Email***" and "***Password***". This will return a successful or unsuccessful response. If the successful response is received the user will be provided access to their account – progressing onto the homepage("***Home.java***"). If an unsuccessful response is returned, the user will be notified and asked to reattempt to sign in.

7.5.3 Register Activity

Associated Files

Java	XML
Register.java ModelUser.java	activity_register.xml

Code Snippet

```
mAuth.createUserWithEmailAndPassword(email, password)
        .addOnCompleteListener(Register.this, new
OnCompleteListener<AuthResult>() {
     @Override
      public void onComplete(@NonNull Task<AuthResult> task) {
 // If sign in fails, display a message to the user. If sign in succeeds
 // the auth state listener will be notified and logic to handle the
 // signed in user can be handled in the listener.
if (!task.isSuccessful()) {
Toast.makeText(Register.this, "Registration failed. \n" + task.getException(),
Toast.LENGTH_SHORT).show();
                }
 else {
Toast.makeText(Register.this, "Successfully Registered. Welcome \n " + email,
Toast.LENGTH_SHORT).show();

//Pushing Data to Firebase From:
https://firebase.google.com/docs/database/admin/save-data
 String userId = mAuth.getUid().toString(); //Get User ID to string

);//Create new User Model when new user is created
ModelUser newUser = new ModelUser(name,weight,email,joined,phone

//create new Hashmap of type user model
Map<String, ModelUser> newUserDetails = new HashMap<>();

newUserDetails.put("accountData", newUser); //Define title and values of Hashmap

usersRef.child(userId).child("accountDetails").setValue(newUserDetails); //Push
the created Hashmap to the Database (Using the Declared Reference to the Users
Table)
  startActivity(new Intent(Register.this, Home.class)); //Open the Home Activity
(As user has successfully signed in)
                    finish(); //Finish/Close the Registration Activity
                }
            }
        });
```

Figure 99 - Register

Overview

The Registration Activity can be accessed via an option at the bottom of the "***Login***" activity. The registration activity allows new users to create an account. This section asks users to enter some required information such as Name, Email, Phone and Password. The email and password will be used as part of the Login process in the future. The Registration activity utilizes both Firebase Authentication and Firebase Database.

When a new user enters their details, a call to the Firebase Authentication branch is undertaken. The Authentication function "***createUserWithEmailAndPassword()***" is passed the newly entered email and password. Similar to the login activity - a successful or unsuccessful response will be returned. Of course, this section also inhibits the capability to interpret incorrect entries such as emails missing characters @ and passwords being blank. This section ignores the weight input as it is entirely optional.

If a successful response is returned, a new user account will be created in the Firebase Authentication branch. The user id will then be grabbed and used to create a new entry in the Firebase database. (An overview of the structure of this database can be seen in the above section titled "***Firebase Database***"). The fields Name, Email, Phone and Weight will be grabbed from the Text Inputs, casted to a Model called "***ModelUser.java***" and then placed in a Hashmap. This hashmap is then pushed to the Firebase Database. This effectively creates a new user table with the root node being the newly generated user id.

If an unsuccessful response is returned, the user will be notified that the Registration has failed. The input error handling will be applied before any contact is established with Firebase Authentication, therefore this response would indicate that the user is attempting to use an already in use email.

7.5.4 Home Activity

Associated Files

Java	XML
Home.java	activity_home.xml

Code Snippet

```
public void openMyFoodNetwork(View view){
    Intent intent = new Intent(this,FoodContents.class);
    startActivity(intent);
}

public void openShopping(View view){
    Intent intent = new Intent(this,Shopping.class);
    startActivity(intent);
}
```

Figure 100 - Home Activity

Overview

On successful login or registration, users will be directed to the homepage. The homepage does not hold much functionality as it is only used as a mediator to allow users to navigate to other sections. There are a couple of methods which are called when the icons are clicked. These methods have been titled "***open + SectionName***" for example "***openShopping()***" etc. The sample snippet above displays two of these methods. These are basic intents that are used in Android to do something, i.e. pass content, open another class etc.

7.5.5 My Food Network Activity

Associated Files

Java	XML
FoodContents.java ModelFoodItem.java AdapterFoodItem.java	activity_food_contents.xml adapter_food_item_layout.xml

Code Snippet 1

```
public void getContents() {
    //Get contents from Firebase into String From :
https://www.youtube.com/watch?v=WDGmpvKpHyw
    keyRef.addValueEventListener(new ValueEventListener() { //SingleValueEvent
Listener to prevent the append method causing duplicate entries
        @Override
        public void onDataChange (DataSnapshot dataSnapshot){
            foodList.clear(); //Clear foodlist before adding items again
            //Get ID From: https://stackoverflow.com/questions/43975734/get-
parent-firebase-android
            for (DataSnapshot ds : dataSnapshot.getChildren()) {
                String type = ds.child("foodType").getValue().toString();
                String expDate = ds.child("expiryDate").getValue().toString();
                String category = ds.child("category").getValue().toString();

                ModelFoodItem newItem = new ModelFoodItem(type,
expDate,category);
                foodList.add(newItem);
            }
```

Figure 101 - Pulling Data from Firebase

Overview

The "***My Food Network***" activity is used to display all the food contents within a user's home. This information is all pulled from the Firebase Database. The table "***foodItems***" nested within the users table (Root = user id). When a user is signed in, the user id will be used to grab the corresponding data from the Database associated with their account. This activity is composed of many functions:

Grabbing Values from Firebase

A method called "***getContents***" is used to grab the values from the database. This method is called within the "***onCreate***" method so it can parse data as soon as the UI loads. An instance of the Firebase database is created, then the method "***addValueEventListener***" is called. This is a proprietary Firebase function which allows the Android device to grab any required data. This method also acts as an active listener – once the method is called it will infinitely listen to changes. As such, any updates to the database can be seen live on the Mobile application. (One specific use case would be when a user adds or removes items via the RFID controller).

After the listener is called, a "Snapshot" of the database will be pulled down into the application. A Snapshot is effectively a duplicate image or instance of the database at the current moment in time. This "Snapshot" will only hold data associated with the signed in user. Once the "Snapshot" has been pulled from Firebase, the data is looped through and added to an object called "***ModelFoodItem()***". This model is saved into an arraylist of objects. The arraylist is then passed to a custom adapter called "***AdapterFoodItem.java***". The custom adapter utilizes the "***adapter_food_item_layout.xml***" file to display the data. This process of grabbing data from Firebase, saving it to a new object and passing the object list to the custom adapter has been used extensively throughout the project - as

such the associated processes have been covered in detail. As almost all the custom adapters follow the same methodology.

Another event listener titled "***addValueEventListenerForSingleValueEvent()***" has also been used throughout development. This function only pulls data from Firebase once.

Filter By Category

This section also supports filtering of food items via Category. There are 4 primary categories titles "***Poultry***", "***Fruit and Veg***", "***Dairy***" and "***Misc.***". Once any of the categories from the drop-down menu are selected, the list is searches through - with all items that do not hold the specified category being removed. The updated list is then passed once-more to the custom adapter so it can be displayed on the UI.

Code Snippet 2

```
for(int i =0;i<foodList.size();i++){
    if(foodList.get(i).getCategory().equals("Poultry")){
        //Do nothing
    }
    else{
        foodList.remove(i);
    }
}
```

Figure 102 - Filtering by Category

7.5.6 Recipes List Activity

Associated Files

Java	XML
RecipesFromFoodContents.java ModelRecipeFromIngredient.java GetRecipesByIngredientApi.java AdapterRecipeByIngredients.java SimilarRecipe.java	activity_recipes_from_food_contents.xml adapter_recipes_by_ingredient_layout.xml activity_similar_recipe.xml

Code Snippet 1

```
//Get Data From API
myUrl = "https://spoonacular-recipe-food-nutrition-
v1.p.mashape.com/recipes/findByIngredients?fillIngredients=false&ingredients="+f
oodType+"&limitLicense=false&number=10&ranking=1";
try {
    result = getRecipesByIngredientRequest.execute(myUrl).get();
} catch (InterruptedException e) {
    e.printStackTrace();
} catch (ExecutionException e) {
    e.printStackTrace();
}
```

Figure 103 - Making API Call

Overview

The recipes list activity is used to display a listing of recipes returned from the Spoonacular API. This UI is utilized for two endpoints:

1. Recipes via Ingredient
2. Similar recipes via id

Both classes ("***RecipesFromFoodContents.java***" and "***SimilarRecipe.java***") operate in the exact same manner - the only difference being that one uses the ingredient to make an API call, whilst the other uses a recipe id. To describe this example, the "***RecipesFromFoodContents.java***" class will be used.

The "***RecipesFromFoodContents.java"*** class is called, and it is passed the corresponding data – in this case it will be passed the ingredient the API is to search recipes for. Once the class has retrieved this ingredient, it will create a new string url. This ingredient will be combined to this string url.

This url is then passed as a parameter to the "***GetRecipesByIngredientApi.java***" class. This class is used to make GET requests using the Android device. Any networking operations on Android must run on an alternative thread as the UI and other components are allocated the main thread. This reduces latency and errors within the application. As such, this get Request class extends the android AsyncTask library.

```
public class GetRecipesByIngredientApi extends AsyncTask<String, Void, String> {
    public static final String REQUEST_METHOD = "GET";
    public static final int READ_TIMEOUT = 15000;
    public static final int CONNECTION_TIMEOUT = 15000;
    @Override
    protected String doInBackground(String... params){
        String stringUrl = params[0];
        String result;
```

Figure 104 - Extending ASyncTask

Once the API call has successfully executed, the result will be sent back to the "***RecipesFromFoodContents.java"*** class where it is saved as a string ("***result***"). The result from the API must be parsed to the correct format for display. Initially, the result is in JSON format - although it is saved as a string. The first thing to be done is to parse the String to JSON. This was done by passing the String as a parameter to the JSONArray class. Next, the JSON array is looped and the required variables are saved into strings. The Strings are then passed into the object "***ModelRecipeFromIngredien.java***". Next, the model is saved to a list of objects to which it is finally parsed to the UI using the custom adapter "***adapter_recipes_by_ingredient_layout.xml***".

N.B Almost every API GET request performed through the application using the Spoonacular API utilizes the same regime wherein a string uri is passed to the "get" class, then it is passed back to the original class saved as a string and parsed to the UI.

```
//Convert String to JSON in Java From:
https://stackoverflow.com/questions/35722646/how-to-read-json-string-in-java
try {
    array = new JSONArray(result); //Create JSON Array
} catch (JSONException e) {
    e.printStackTrace();
}

for(int i=0; i<array.length(); i++){
    try {
    JSONObject jsonObj;
        jsonObj = array.getJSONObject(i);
        int id = (int) jsonObj.get("id");
        String title = (String) jsonObj.get("title");
        String imageUrl = (String) jsonObj.get("image");
        ModelRecipeFromIngredient recipe = new
ModelRecipeFromIngredient(id,title,imageUrl);
        recipesList.add(recipe);

    } catch (JSONException e) {
        e.printStackTrace();
    }
}
```

Figure 105 - Getting API Result and Parsing to Object

7.5.7 Recipe Details Activity

Associated Files

Java	XML
RecipeDetails.java GetRecipeFromIdApi.java AdapterIngredients.java AdapterInstructions.java ModelIngredient.java ModelInstruction.java GeSimilarRecipesApi.java	activity_recipe_details.xml adapter_ingredients_layout.xml adapter_instructions_layout.xml

Overview

The Recipe Details activity is one of the most extensive sections in the entire application, it makes use of a large number of elements and links to a host of different sections encompassing varied functionalities. Therefore - to adequately cover everything, this section will be segmented into multiple subsections.

Pulling Recipe Details from API

The "***RecipeDetails.java***" class utilizes a unique recipe id to make a call to the API. The API call here follows an almost identical regime of as outlined in the above section titled "***Recipes List Activity***". The only difference is that the url holds a unique recipe id and targets a different spoonacular API endpoint. In short, the "***RecipeDetails.java***" class is passed a recipe id. This id is parsed to the url. The url is then passed as a parameter to the "***GetRecipeFromIdApi.java***" class. The result is returned as a string, then casted to JSON to which it can be manipulated before it is parsed for display to the UI.

Code Snippet 1

```
//Get Data From API
myUrl = "https://spoonacular-recipe-food-nutrition-
v1.p.mashape.com/recipes/"+recipeId+"/information";
try {
    result = getRecipeFromIdApiRequest.execute(myUrl).get();
} catch (InterruptedException e) {
    e.printStackTrace();
} catch (ExecutionException e) {
    e.printStackTrace();
}
```

Figure 106 - Making API Call

Parsing Data from API Result to UI

The API result returns a significant amount of information, therefore it had to be segregated into different sections. The sections in the "***RecipeDetails.java***" UI are:

- The Top Section - (Holds the Recipe Image, Title, Icons and lists details such as preparation time, cook time, servings and weight watcher smart points). These values are the first to be pulled from the API result. When developing the application using this endpoint, some crashes occurred. This was primarily because some recipe results contained different key names. Most of the recipes returned contained the keys "***preparationMinutes***" and "***cookingMinutes"*** - although a small amount contained "***readyInMinutes***" instead. To combat this error, a simple if-else clause was formulated. This resolved the issue immediately. There were many occasions throughout development wherein similar issues occurred. As most issues were with the API, there was very little to be done except implement either if-else or try-catch clauses into the Android code.

Code Snippet 2

```
try {
    object = new JSONObject(result); //Create JSON Object so values can be
retrieved
    title = object.getString("title");
    imageUrl = object.getString("image");
    instructions = object.getString("instructions");
    spoonacularSourceUrl = object.getString("spoonacularSourceUrl");
    servings = object.getString("servings");
    wwSmartPoints = object.getString("weightWatcherSmartPoints");

    if(object.has("preparationMinutes") && object.has("cookingMinutes")){
//Error Handling - Some API Responses contain different data - App was not
loading...
        preparation = object.getString("preparationMinutes");
//...data if it could not find element in JSONObject
        cookTime = object.getString("cookingMinutes");
    }
    else{
        preparation = object.getString("readyInMinutes");
        cookTime = "-";
    }
```

Figure 107 - Handling Errors from API Result

- The Ingredients Section – Lists all the ingredients associated with the current recipe. The next section to be parsed to the UI was the ingredients. The result from the API was composed of multiple nested JSON arrays, with some objects holding even more arrays. The ingredients array had to be looped through so a new JSON array was created. This JSON array was then looped through to grab the required data. The data was saved into variables, these variables were then saved as a new object called "***ModelIngredient.java***". Each new object was saved to a list of objects. The list was then passed to the custom adapter labelled "***adapter_instructions_layout.xml***" so it could be parsed to the UI.

Code Snippet 3

```
//Get Ingredients, Store in Object called ModelIngredient, save objects to
Arraylist
JSONArray extendedIngredientsArray = object.getJSONArray("extendedIngredients");
for(int i=0;i<extendedIngredientsArray.length();i++){
    JSONObject object = (JSONObject) extendedIngredientsArray.get(i);
    String originalString = object.getString("original");
    double amount = object.getDouble("amount");
    String unit = object.getString("unit");
    String name = object.getString("name");

    ModelIngredient ingredientModel = new
ModelIngredient(originalString,amount,unit,name);
    ingredientList.add(ingredientModel);
```

Figure 108 - Saving Ingredients to Object

- The Instructions Section – Lists the procedures which should be followed for the recipe. The instructions section works in the exact same manner as the above "***Ingredients Section***" – the only exception being different variables pulled from the JSON array and passed into a different model, list and custom adapter. ("***InstructionModel.java***", "***instructionsList***", "***adapter_instructions_layout.xml***").

Code Snippet 4

```
//Get Instructions, Store in Object called ModelInstruction, save objects to
Arraylist
JSONArray analyzedInstructions = object.getJSONArray("analyzedInstructions");
JSONObject instructionsObject = (JSONObject) analyzedInstructions.get(0);
JSONArray instructionsObjectArray = (JSONArray) instructionsObject.get("steps");
for(int i=0;i<instructionsObjectArray.length();i++){
    JSONObject object = (JSONObject) instructionsObjectArray.get(i);
    String stepNumber = object.getString("number");
    String instruction = object.getString("step");
    ModelInstruction instructionModel = new
ModelInstruction(stepNumber,instruction);
    instructionsList.add(instructionModel);
```

Figure 109 - Saving Instructions to Object

Add recipe to Favourites

A heart icon at the top of the Action bar allows users to add a recipe to their favourites. This functionality grabs the current Recipe "***Title***", "***id***" and "***Image URL***". These details are then saved as variables, then passed to a Hashmap. The Hashmap is then pushed to the users favourite table in the Firebase Database. This method will first check if the user has the recipe in their favourites. The method compares the new recipe id to all the ids already in their favourites. If it matches (recipe is already in favourites), then the user will be notified, and a duplicate entry will not be added. If the recipe is not in the user's favourites, the new entry will be added.

Code Snippet 5

```
if(addRecipeToFavourites.equals("true")){
    String itemId = favouritesRef.push().getKey();
    Map<String,String> ingredientHmap = new HashMap<>();
    ingredientHmap.put("recipeId",recipeId);
    ingredientHmap.put("recipeTitle",title);
    ingredientHmap.put("recipeImgUrl",imageUrl);

    favouritesRef.child(itemId).setValue(ingredientHmap);

    View view = findViewById(R.id.recipeDetailsLayout);
    String message = title + " added to your favourites."; //Capitalize Using
StringUtils From: https://stackoverflow.com/questions/5725892/how-to-capitalize-
the-first-letter-of-word-in-a-string-using-java
    int duration = Snackbar.LENGTH_SHORT;
    showSnackbar(view, message, duration);
}

else{
    View view = findViewById(R.id.recipeDetailsLayout);
    String message = title + " is already in your favourites."; //Capitalize
Using StringUtils From: https://stackoverflow.com/questions/5725892/how-to-
capitalize-the-first-letter-of-word-in-a-string-using-java
    int duration = Snackbar.LENGTH_SHORT;
    showSnackbar(view, message, duration);
}
```

Figure 110 - Added to Favourites Snackbar Notification

Share Recipe

The share recipe icon is located at the top of the UI as part of the action bar. The share option grabs the current recipe URL from the result and passes it to a new intent. The new intent enables the user to choose what platform they would like to share the recipe on. Options include, Whatsapp, Email, SMS etc.

Code Snippet 6

```
public void shareRecipe(){
    //Share Intent From: https://stackoverflow.com/questions/19683297/how-to-
send-message-from-android-app-through-viber-message
    Intent shareIntent = new Intent(android.content.Intent.ACTION_SEND);
    String recipeData = "Hey, check out this cool recipe from The Intelligent
Food Network App: " + spoonacularSourceUrl;
    shareIntent.putExtra(Intent.EXTRA_TEXT, recipeData);
    shareIntent.setType("text/plain");
    startActivity(Intent.createChooser(shareIntent, "Please choose an
application to share your recipe on..."));
}
```

Figure 111 – Share Recipe Intent

Add item to shopping List

The add item to shopping list method allows users to add an item from the ingredients list to their shopping list. When a user clicks on any ingredient, an AlertDialog menu will display and give the option to add to shopping list. The add to shopping list method works similar to the Add to favourites in the way it compares the Firebase data and the proposed newly added data – in this case the ingredient string. Both strings are first concatenated to lowercase and then trimmed before the comparison. If the strings match, then the item is already in the shopping list and is not added. If the strings do not match, the new item is added to the shopping list.

Code Snippet 7

```
if(addItemToShoppingList.equals("true")){
    String itemId = shoppingListRef.push().getKey();
    Map<String,String> ingredientHmap = new HashMap<>();
    ingredientHmap.put("title",StringUtils.capitalize(ingredient));
    shoppingListRef.child(itemId).setValue(ingredientHmap);

    View view = findViewById(R.id.recipeDetailsLayout);
    String message = ingredient + " added to Shopping List."; //Capitalize Using
StringUtils From: https://stackoverflow.com/questions/5725892/how-to-capitalize-
the-first-letter-of-word-in-a-string-using-java
    int duration = Snackbar.LENGTH_SHORT;
    showSnackbar(view, message, duration);
}
```

Figure 112 - Adding New Item to Shopping List

Ingredient List Icon and Green Ingredient Text

Another feature associated with the ingredients list is the applications ability to parse an icon at the side of the ingredient. This icon is an image of a shopping trolley and indicates that a user already holds the ingredient in their shopping list. This feature is achieved by passing a string list to the custom adapter. When the adapter is parsing the values it first checks if the value it is passing matches that already in the database. The custom adapter not only checks if the item is in the shopping list, it also checks if the item is currently in the user's home – if so, the ingredient will be highlighted in green. If the ingredient is in both the users home and in the shopping list, it will be highlighted green, with a shopping trolley icon placed at the right side.

Code Snippet 8

```
// Populate the data into the template view using the data object
//Android Set color programmatically From:
https://stackoverflow.com/questions/4602902/how-to-set-the-text-color-of-
textview-in-code
if(foodList.contains(ingredientList.get(i).getIngredientName().toLowerCase().tri
m())){
    //If match, highlight in Green
    tvIngredient.setTextColor(Color.parseColor("#159B4A"));
    tvIngredient.setText(ingredientList.get(i).getOriginalString());
}
else{
    //Just Parse the value
    tvIngredient.setText(ingredientList.get(i).getOriginalString());
}

if(shoppingList.contains(ingredientList.get(i).getIngredientName())){
    //Set Image porgramatically

alreadyInShoppingListIcon.setBackgroundResource(R.drawable.ic_shopping_cart);
}
else{
    //Do nothing
}
```

Figure 113 - Setting the Icon and Colour through the Adapter Class

Similar Recipes

The similar recipes button at the bottom of the page allows users to view similar recipes to the one they are currently viewing. This once again makes use of the Spoonacular API. The processes are identical to those outlined above in "***Recipes List Activity***" – the only difference being the fact that a recipe id is used to search for recipes as opposed to an ingredient.

Code Snippet 9

```
public void openSimilarRecipe(){
    Intent intent = new Intent(this,SimilarRecipe.class);
    intent.putExtra("similarRecipeId",recipeId); //Pass String from one Activity
to another From: https://stackoverflow.com/questions/6707900/pass-a-string-from-
one-activity-to-another-activity-in-android
    startActivity(intent);
}
```

Figure 114 - Similar Recipes Method

7.5.8 Shopping List Activity

Associated Files

Java	XML
ShoppingList.java AdaoterShoppingList.java ModelShoppingListItem.java	activity_shopping_list.xml adapter_shopping_list_layout.xml

Code Snippet

```
@Override
public void onDataChange(DataSnapshot dataSnapshot) {
    shoppingList.clear();
    comparisonList.clear();
    //Get ID From: https://stackoverflow.com/questions/43975734/get-parent-
firebase-android
    for (DataSnapshot ds : dataSnapshot.getChildren()) {
        String item = ds.child("title").getValue().toString();
        ModelShoppingListItem shoppingListItem = new
ModelShoppingListItem(item);
        shoppingList.add(shoppingListItem);
        comparisonList.add(item);
    }

    shoppingListGridView.setAdapter(null); //Clear adapter so the information is
not duplicated
    shoppingListGridView.setAdapter(adapter);
    isShoppingListEmpty();
}
```

Figure 115 - Getting Shopping List and Parsing to UI

Overview

The shopping list activity is used to show users what items they have in their shopping list. This section works very similar to the "***My Food Network***" activity as it uses the exact same Firebase method to grab a snapshot of the database. The pulled data is saved as an object called "***ModelShoppingListItem***". This object passed to an object list, then passed to the custom adapter to be parsed to the UI. The custom adapter utilises a grid view as opposed to the listview seen in other sections. Users may also add items to this section through the "***RecipeDetails Activity***" or manually by clicking on the plus icon on the Action-bar. If they click the icon, a AlertDialog will pop up. When a user attempts to add a new item manually, the method will first check to see if the item is a duplicate before posting to the Firebase Database.

7.5.9 Shopping List Recommendation Activity

Associated Files

Java	XML
ShoppingListRecommendation.java	activity_shopping_list_recommendation.xml

Code Snippet

```
for(int i=0;i<shoppingList.size();i++){
    String shoppingListItem = shoppingList.get(i).toString();
    if(foodItemsList.contains(shoppingListItem)){
        //Do nothing
    }
    else{
        recommendationList.add(StringUtils.capitalize(shoppingListItem));
    }
```

Figure 116 - Generating Shopping Recommendation

Overview

The shopping list recommendation class once-more utilizes the "***addValueEventListener***" method from the Firebase Database library. A snapshot of the users "***foodItems***" and "***shoppingList***" table is pulled from Firebase. These values are then saved into two separate lists titled "***foodItemsList***" and "***shoppingList***". The two lists are looped through and compared to see what matches. The example food item"***Carrost***" will be used to describe the comparison. If "***Carrots***" are present in both lists, then this food item is not added to the "***recommendationList***". The method evaluates and decided "*Okay, the user has carrots in their shopping list, but also already has carrots in their home*" – therefore they do not require Carrots on the next shopping trip. Of course, if an item is on the weekly shopping list, yet not in the home - then it is saved into the recommendation list. The recommendation list is then parsed to the UI in the "***ShoppingList Activity***".

7.5.10 Recipes Activity

Associated Files

Java	XML
Recipes.java FavouriteRecipes.java ModelFavouriteRecipe.java ModelRecipeFromIngredient.java	activity_recipes.xml activity_favourite_recipes.xml adapter_favourite_recipes_layout.xml

Overview

The Recipes Activity section can be accessed via the homepage. The recipes activity has 3 primary functions:

1. To generate a Recipe Recommendation (This will be listed in its own section below and titled "***Recipe Recommendation***")

2. To allow users to navigate to their Favourite Recipes
   When a user clicks "***View Favourite Recipes***" they are brought to the "***Favourite Recipes***" activity. This activity pulls down a snapshot of all the users favourite recipes. The recipes are saved as objects, then saved into an object list. This object list is passed to the custom adapter "***adapter_favourite_recipes_layout.xml***" so it can be displayed on the UI. The "***Favourite Recipes***" activity allows users to View and Remove recipes from their favourites.

Code Snippet 1

```
public void removeFavouriteRecipe(){
    //Firebase Remove Files From:
https://firebase.google.com/docs/storage/android/delete-files
    keyRef.addValueEventListener(new ValueEventListener() {
        @Override
        public void onDataChange(DataSnapshot dataSnapshot) {
            for (DataSnapshot ds : dataSnapshot.getChildren()) {
                String firebaseId = ds.child("recipeId").getValue().toString();
                String recipeTitle =
ds.child("recipeTitle").getValue().toString();
                if(firebaseId.equals(recipeId)){
                    String key = ds.getKey().toString();
                    keyRef.child(key).removeValue();
                }
            }
        }
```

Figure 117 - Removing a Favourite Recipe

3. To allow users to search for a Recipe via ingredient
   When a user clicks on the "***Search Recipes***" option, an AlertDialog will pop up to provide search functionality. After the user submits the ingredient they would like to view recipes for, an API GET request is made. This API request utilizes the endpoint "***Recipes via Ingredient***" once-more. The user will then be routed to the "***Recipes List***" activity (outlined previously in section titled "***Recipes List Activity***").

Code Snippet 2

```
//Android Alertbox with EditText From:
https://stackoverflow.com/questions/18799216/how-to-make-a-edittext-box-in-a-
dialog
searchLinearLayout.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        AlertDialog.Builder alert = new AlertDialog.Builder(Recipes.this);
        final EditText searchEt = new EditText(Recipes.this);
        alert.setMessage("Enter an ingredient...");
        alert.setTitle("Recipes Search");
        alert.setView(searchEt);
        alert.setPositiveButton("Search", new DialogInterface.OnClickListener()
{
            public void onClick(DialogInterface dialog, int whichButton) {
                //What ever you want to do with the value
                 recipeSearchString = searchEt.getText().toString();
                 openRecipesFromFoodContents(); //Open class to make API call
            }
        });
```

Figure 118 - Recipes Search Dialog

7.5.11 Recipe Recommendation

Associated Files

Java	XML
Recipes.java ModelFavouriteRecipe.java	activity_recipes.xml

Overview

When the "***Recipes.java***" class is loaded, a recommendation is automatically generated. This functionality is based off the recipes a user currently has in their favourites.

- First, a snapshot of the user's favourite recipes is pulled from the Database
- The data is then saved as an object called "***ModelFavouriteRecipe***"
- This object is then added to an object list "***favouriteRecipesList***"
- The Math.random() function is used to pick a random number
- This random number will be between 0 and the list size
- The random number is selected and the recipe object corresponding to this number is grabbed from the list
- The recipe id is taken from the randomly selected object
- The id is passed to the API (Endpoint: Similar Recipes via ID)
- The API will return 10 similar recipes saved into another list (This can be changed but is currently limited to 10 results)
- The Math.random() function is again applied (with the number between 0-9 or 0 and the API result list size)
- The similar recipe corresponding to the random number is pulled from the list
- Finally, the result are displayed in the UI

In short, a random recipe from the user's favourites is selected. An API GET request is made to grab similar recipes to the selected favourite. A random similar recipe is once more grabbed from the similar recipes list and presented in the UI.

Code Snippet

```
    listSize = dataSnapshot.getChildrenCount();//Firebase Get Children Count
From: https://stackoverflow.com/questions/43606235/android-firebase-get-
childrens-count
    rand = new Random();
    randomRecipeFromFavourites = rand.nextInt((int) listSize)+0;
    String idForSimilarRecipeApi =
favouriteRecipesList.get(randomRecipeFromFavourites).getRecipeId();
    String titleForSimilarRecipeApi =
favouriteRecipesList.get(randomRecipeFromFavourites).getRecipeTitle();
    recommendationCv.setClickable(true);

    //Get Data From API
    myUrl = "https://spoonacular-recipe-food-nutrition-
v1.p.mashape.com/recipes/"+idForSimilarRecipeApi+ "/similar";
    try {
        result = getRequest.execute(myUrl).get();
    } catch (InterruptedException e) {
        e.printStackTrace();
    } catch (ExecutionException e) {
        e.printStackTrace();
    }

    //Convert String to JSON in Java From:
https://stackoverflow.com/questions/35722646/how-to-read-json-string-in-java
    try {
        array = new JSONArray(result); //Create JSON Array
    } catch (JSONException e) {
        e.printStackTrace();
    }

    for(int i=0; i<array.length(); i++){
        try {
            JSONObject jsonObj;
            jsonObj = array.getJSONObject(i);
            int id = (int) jsonObj.get("id");
            String title = (String) jsonObj.get("title");

            String imageUrl = "https://spoonacular.com/recipeImages/" + (String)
jsonObj.get("image");
            ModelRecipeFromIngredient recipe = new
ModelRecipeFromIngredient(id,title,imageUrl);
            recipesList.add(recipe);

        } catch (JSONException e) {
            e.printStackTrace();
        }
    }
    randomRecipeFromSimilar = rand.nextInt(recipesList.size())+0;
    recommendationId = recipesList.get(randomRecipeFromSimilar).getId();
    String similarRecipeImageUrl =
recipesList.get(randomRecipeFromSimilar).getImageUrl();
    String similarRecipeTitle =
recipesList.get(randomRecipeFromSimilar).getTitle();

Picasso.with(Recipes.this).load(similarRecipeImageUrl).into(recommendationImg);
//Use picasso library to load images instead of setImageResource
    recommendationTitleTv.setText(similarRecipeTitle);
    basedOnTv.setText(titleForSimilarRecipeApi);
}
```

Figure 119 - Recipe Recommendation Method

7.5.12 Nutrients Results Activity & Views

Associated Files

Java	XML
NutrientsResult.java ModelNutrientsResult.java GetNutrientsApi.java	activity_nutrient_result.xml adapter_nutrients_result.xml

Overview

The "***Nutrients Results Activity***" is used to present the user with nutritional values based on their search criteria. This section of the application utilizes the "***Nutritionix***" API – another API which is used solely for making GET requests to return nutritional values. It was decided that a second API would increase variation within the system. The endpoint in use here is "***Nutritional Values via Food Type***". As expected, the response from the API is different from that of Spoonacular.

When a user opens the "***Nutrients Results Activity***" they are presented with a search bar. Users may search for any food item they can imagine, e.g. Apples, Oranges, Chicken, Curry etc.
The search string will be grabbed and passed to the "***GetNutrientsApi.java***" class. This class will make a call to the "***Nutritionix API***" and then return a response. This API response differs from Spoonacular as it returns an object saved into a Hashmap as opposed to a JSON formatted string. Initially, this caused some issues as the multiple maps and objects needed to be created in order to pull out the required data. Once the data is pulled out from the result HashMap it is casted to an object called "***modelNutrientsResult***". This model is saved into an object list. The object list is then passed to the custom adapter "***adapter_nutrients_results.xml***" for display in the UI.

This section can be not only accessed from the homepage, but also in many other places within the application i.e. Users can view nutritional data of the food in the "***My Food Network***", "***Shopping List***" and "***Recipe Details***" sections. These sections will display an AlertDialog when the user clicks on the corresponding food item. This Dialog will provide the option to "***View Nutritional Info***". Once clicked, the same processes outlined above will be applied, except the user will be routed to the "***NutrientSearch.java***" activity – an identical UI to the "***Nutrients Results.java***" activity, minus the search bar. The below example shows the AlertDialog shown when the user has clicked on the Horseradish ingredient located in the "***RecipeDetails.java***" activity.

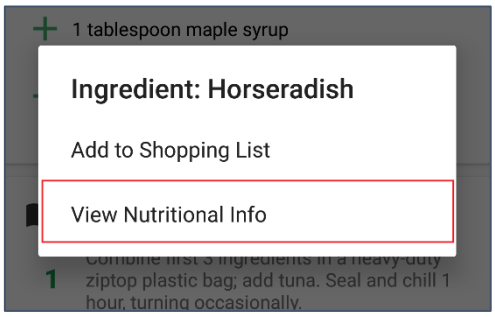


Figure 120 - Ingredient Options

Code Snippet 1

```
try {
    response = new GetNutrientsApi().execute(foodType).get();
    setData();
} catch (InterruptedException e) {
    e.printStackTrace();
```

Figure 121 - Making API Call

Code Snippet 2

```
public void setData(){
    ArrayList<Object> foods = (ArrayList<Object>) response.get("success");
    Map<Object,Object> contents = (Map<Object, Object>) foods.get(0);
    ArrayList<Object> item = (ArrayList<Object>) contents.get("foods");
    Map<Object,Object> foodItem = (Map<Object, Object>) item.get(0);

    Map<Object,String> imageUrlObj = (Map<Object, String>)
foodItem.get("photo");
    String imageUrl = imageUrlObj.get("thumb");

    String food_name = foodItem.get("food_name")+"";
    String serving_qty = foodItem.get("serving_qty")+"";
    String serving_unit= foodItem.get("serving_unit")+"";
    String serving_weight_grams = foodItem.get("serving_weight_grams")+"";
    String calories = foodItem.get("nf_calories")+"";
    String total_fat = foodItem.get("nf_total_fat")+"";
    String saturated_fat = foodItem.get("nf_saturated_fat")+"";
    String cholesterol = foodItem.get("nf_cholesterol")+"";
    String sodium = foodItem.get("nf_sodium")+"";
    String total_carbohydrate = foodItem.get("nf_total_carbohydrate")+"";
    String fibre = foodItem.get("nf_dietary_fiber")+"";
    String sugars = foodItem.get("nf_sugars") + "";
    String protein = foodItem.get("nf_protein")+"";
    String potassium = foodItem.get("nf_potassium")+"";



    ModelNutrientsResult modelNutrientsResult = new
ModelNutrientsResult(imageUrl,food_name,serving_qty,serving_unit,serving_weight_
grams,

calories,total_fat,saturated_fat,cholesterol,sodium,total_carbohydrate,fibre,sug
ars,protein,potassium);
```

Figure 122 - Saving Data from API to Object

7.5.13 View User Account Activity

Associated Files

Java	XML
UserAccount.java	activity_user_account.xml

Overview

The "***UserAccount.java***" activity is used to display all the key information associated with the currently signed in user. This activity provides an overview of the entire user account – listing information:

- Profile Image
- Name
- Email
- Phone
- Count of Food items in home, Favourite recipes & Shopping List Items
- Date Joined the platform
- Weight

This class utilizes the 3 branches of Firebase:

Firebase Authentication is utilized to grab the user id. By grabbing this id, the users table in the database can then be referenced before a snapshot is taken.

Code Snippet 1

```
//Get Instance of Firebase Authentication
mAuth = FirebaseAuth.getInstance();
userId = mAuth.getUid().toString();
```

Figure 123 - Getting User ID

Firebase Database is used to pull down key information such as Name, email, phone and the counters (Shopping List, Favourite Recipes) etc. Once again, and identical to other sections the "***valueEventListener***" function is utilized to pull down a database snapshot so the information can be passed to the UI. One minute difference here being the absence of Listviews. Listviews were not required for this section as there is effectively only one object in this view (The user). Once the values are grabbed, they are placed in their designated placeholders. The counters were formulated using the same methodology, although instead of grabbing values, a counter was created and incremented for each value in the snapshot.

Code Snippet 2

```
for (DataSnapshot ds : dataSnapshot.getChildren()) {
     name = ds.child("name").getValue().toString();
     email = ds.child("email").getValue().toString();
     phone = ds.child("phone").getValue().toString();
     weight = ds.child("weight").getValue().toString();
     joined = ds.child("joined").getValue().toString();
}
```

Figure 124 - Getting User Details

Code Snippet 3

```
for (DataSnapshot ds : dataSnapshot.getChildren()) {
    //The DataSnapshot will iterate through all elements within the specified
table(JSON Object), add to the counter for each item iterated
     numberOfFoodItems = numberOfFoodItems +1;
}
```

Figure 125 - Popularing Counters

Firebase Storage is used here to pull the user profile image into the ImageView. An instance of the Storage branch is created. A proprietary function "***getDownloadUrl()"*** is provided by the Firebase Storage class is then used to grab the currently signed in users profile image. Once again, this section is heavily dependent on the User id provided by the Firebase Authentication branch. If the user has not yet uploaded an image, the default image will be pulled from the Storage branch and the user will then be notified to "Complete their profile" via the Android Snackbar view.

Code Snippet 4

```
public void getUserImage() {
    //Android Getting Image From Firebase Storage From:
https://stackoverflow.com/questions/38424203/firebase-storage-getting-image-url

storageReference.child("profileImages/"+userId+"/profileImage").getDownloadUrl()
.addOnSuccessListener(new OnSuccessListener<Uri>() {
        @Override
        public void onSuccess(Uri uri) {
            Picasso.with(getApplicationContext()).load(uri).into(profileIv);
        }
    }).addOnFailureListener(new OnFailureListener() {
        @Override
        public void onFailure(@NonNull Exception exception) {
            String uploadTipImage =
"https://firebasestorage.googleapis.com/v0/b/theintelligentfoodnetwork.appspot.c
om/o/profileImages%2Fuploadimage.png?alt=media&token=6dadee2f-cfcd-43ff-8d54-
035153d0c12c";

Picasso.with(getApplicationContext()).load(uploadTipImage).into(profileIv);

            View view = findViewById(R.id.userAccountLayout);
            String message = "Upload an image to complete your profile!";
            int duration = Snackbar.LENGTH_SHORT;
            showSnackbar(view, message, duration);
        }
    });
```

Figure 126 - Pulling a User Profile Image and Parsing to UI using Picasso

7.5.14 Edit User Account Activity

Associated Files

Java	XML
EditAccount.java	activity_edit_account.xml

Overview

Users have the option to edit and customise their account. This functionality can be accessed through the "***UserAccount.java***" activity. The loading of data into this section is identical to the section outlined above ("***UserAccount.java***") – the only difference being that the information is pulled into EditText fields as opposed to TextView fields. Firebase Authentication, Database and Storage are used once-more to grab the user id, data and profile image. This information is then parsed to the UI.

From here, a user may change their profile image. This functionality can be utilized by clicking on the current profile image (If the user has not yet uploaded an image, the default image will be used as a placeholder). When the click is registered, a the "***chooseImage()***" method is called. This method creates a new intent which allows the user to select a new image to upload from their Android device. The image can be uploaded via the in build File Storage Navigation pane, or the Device Gallery. In either case, when the image is selected, it will be passed back to the application and parsed to the ImageView so the user can preview the changes.

Code Snippet 1

```
//Method which allows user to access device Gallery to upload a Profile Image
public void chooseImage(View view){
    Intent intent = new Intent();
    intent.setType("image/*");
    intent.setAction(Intent.ACTION_GET_CONTENT);
    startActivityForResult(Intent.createChooser(intent, "Select Picture"),
PICK_IMAGE_REQUEST);
}
```

Figure 127 - Upload Image Intent

Code Snippet 2

```
if(requestCode == PICK_IMAGE_REQUEST && resultCode == RESULT_OK
        && data != null && data.getData() != null )
{
    filePath = data.getData();
    try {
        Bitmap bitmap = MediaStore.Images.Media.getBitmap(getContentResolver(),
filePath);
        profileIv.setImageBitmap(bitmap); //Set Image in View
    }
```

Figure 128 - Setting Image into Preview

One the user has edited their profile, then can save the changes by clicking on the "***Update Profile***" button. This button makes a call to the method "***updateProfile()".*** All corresponding information is grabbed from the view – this includes the image and updated user details. The Image is uploaded to Firebase Storage where it overwrites the old image. The Data is grabbed and saved to an object called "***ModelUse.java***". The object is saved into a Hashmap wherein finally the data is pushed to the Firebase Database under the root userId.

Code Snippet 3

```
StorageReference ref =
storageReference.child("profileImages/"+userId+"/profileImage".toString());
ref.putFile(filePath)
        .addOnSuccessListener(new OnSuccessListener<UploadTask.TaskSnapshot>() {
            @Override
            public void onSuccess(UploadTask.TaskSnapshot taskSnapshot) {
                Toast.makeText(EditAccount.this, "Profile Image Uploaded",
Toast.LENGTH_SHORT).show();
            }
        })
```

Figure 129 - Uploading Image to Firebase Storage

Code Snippet 4

```
ModelUser updatedUser = new ModelUser(name,weight,email,joined,phone);//Create
new User Model when new user is created
Map<String, ModelUser> updatedUserDetails = new HashMap<>(); //create new
Hashmap of type user model
updatedUserDetails.put("accountData", updatedUser); //Define title and values of
Hashmap
usersRef.setValue(updatedUserDetails); //Push the created Hashmap to the
Database (Using the Declared Reference to the Users Table)
Toast.makeText(getApplicationContext(),"User Account Successfully
Updated",Toast.LENGTH_LONG).show();
```

Figure 130 - Updating Details in Firebase

7.5.15 Misc. Functionalities

Associated Files

Java	XML
Multiple Classes	Multiple Layouts

Overview

Throughout development, there have been various smaller tools and methods used to create the various functionalities seen in the final project. Although small, some of the external libraries and methods help mould the project together. Therefore, they have been outlined in this section.

Loading Images From URL

Picasso – An extremely useful library was utilized in various sections of the project to load images. This library takes an image URL and view id as parameters, then parses the image to the targeted view. This library was used in sections such as "***RecipesList.java***", "***RecipeDetails.java***" ,"***ShoppingList.java***" and "***UserAccount.java***".

Code Snippet 1

```
storageReference.child("profileImages/"+userId+"/profileImage").getDownloadUrl()
.addOnSuccessListener(new OnSuccessListener<Uri>() {
    @Override
    public void onSuccess(Uri uri) {
        Picasso.with(getApplicationContext()).load(uri).into(profileIv);
```

Figure 131 - Loading Images with Picasso

Firebase Instances

Almost every section of the application depended on the Firebase platform. An instance of each Firebase branch was usually declared at the top of each class

Code Snippet 2

```
//Firebase Authentication
private FirebaseAuth mAuth;
private String userId;

//Firebase Database
private FirebaseDatabase firebaseDatabase;
private DatabaseReference databaseReference;
private DatabaseReference usersRef;

//Firebase Storage
private FirebaseStorage storage;
private StorageReference storageReference;
```

Figure 132 - Declaring Instances of Firebase

Setting Action Bar Title

The Action Bar title was set programmatically at the top of each onCreate method. The Associated String was pulled from the Strings.xml resource file.

Code Snippet

```
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    //Change Action Bar Title From:
https://stackoverflow.com/questions/3438276/how-to-change-the-text-on-the-
action-bar
    setTitle(R.string.edit_profile_action_bar_string);
```

Figure 133 - Programmatically setting Action Bar Title

Hamburger Navigation Menu

A simple Hamburger Navigation menu was created to allow easier navigation throughout the application. This menu utilizes a drawer layout to allow the menu to flow in and out from the side. The Hamburger menu is declared and listens for changed using the OnClickListener method. Once the changes are detected, it acts accordingly – using the location in the list to call upon the required methods.

Code Snippet

```
mDrawerList = findViewById(R.id.navList);
mDrawerList.setOnItemClickListener(new AdapterView.OnItemClickListener() {
    @Override
    public void onItemClick(AdapterView<?> parent, View view, int position, long
id) {
        if(id == 0){
            finish();
            openHome();
        }
        else if(id == 1){
            finish();
            openFoodContents();
        }
        else if(id == 2){
            finish();
            openShoppping();
```

Figure 134 - Hamburger Menu OnClick Listener

Android Snackbar

The Android Snackbar was utilized to display messages to the user. The Snackbar requires specific parameters such as a layout file id, message to display and duration time.

Code Snippet

```
//Android Snackbar From: https://spin.atomicobject.com/2017/07/10/android-snackbar-tutorial/
public void showSnackbar(View view, String message, int duration) {
    Snackbar.make(view, message, duration).show();
}

View view = findViewById(R.id.userAccountLayout);
String message = "Upload an image to complete your profile!";
int duration = Snackbar.LENGTH_SHORT;
showSnackbar(view, message, duration);
```

Figure 135 - Android Snackbar Notification

Android Toast

The Android Toast is another element that was used to display messages to the user. The Toast is usually a little easier to display as it does not require a set view id. Nonetheless, both were utilized throughout development.

Code Snippet

```
Toast.makeText(EditAccount.this, "Profile Image Uploaded",
Toast.LENGTH_SHORT).show();
```

Figure 136 - Android Toast Notification

7.6 Application Programming Interfaces

Associated Files

Spoonacular API	Nutritionix API
GET Recipes via Ingredient GET Recipe via ID GET Similar Recipes via ID	GET Nutritional Data via Food Name

Overview

Both API's were utilized extensively throughout the entire project. The Spoonacular API was primarily utilized to provide the users with recipes. A total of 3 API endpoints were used from this API. Of course, each endpoint produces a plethora of information including ingredients, instructions, images and much more.

The Nutritionix API was chosen to provide the users with Nutritional Data based on any of their search criteria. This particular API was primarily due to its extensive database of food items and nutritional data. Only one endpoint was used from this API.

As outlined above, both API's returned different results. Spoonacular returns a JSON formatted String, whilst the Nutritionix API returned a HashMap. In either case, the information had to be looped through and pulled out into Objects before it could be parsed to the UI. The below snippets feature sample responses returned from both API's printed in the RapidAPI testing console.

Figure 137 - Spoonacular Recipe via ID Result

Figure 138 - Nutritionix Nutrients via Food Name Result

8 Testing

One of the fundamental aspects of any system is usability & reliability. The underlying system architecture and all associated functionalities were derived from the original user survey ("***Section 2***").

To ensure the project aims were achieved the system was tested in a variety of ways. The testing process was broken down into 2 primary categories:

1. Internal Testing
2. External Testing

8.1 Internal Testing

Similar to many software development companies and organisations across the globe - internal testing was conducted to phase out any potential errors, bugs and overlooked system issues. Internal testing is a vital component of any software application - therefore it was mandatory that a significant amount of internal testing was undertaken before any external testing could commence. The Android Application is the entry point between the user and the back-end system, as such the internal testing process was primarily focused on this aspect of the system.

8.1.1 Multiple Device Support

As outlined in sections "***3.2.8 – Portability Requirement***" and "***3.2.10 – Resource Utilization Requirement***", the system was designed to be interoperable with a range of different hardware devices and Android OS versions. Throughout development, the built in Android Studio emulator was used to test the Application on a variety of virtual devices. A physical device – "***Samsung Galaxy S7 Edge***" was also used.

Once the application had been completed, a range of different physical Android devices were gathered for testing. A total of 5 physical devices were used throughout this process. First, the APK was created in Android Studio and installed on all 5 target devices. A labelling machine was then used to title each of the devices. The title consisted of Device Name, Android OS version and API Version. All system use cases were then individually tested on each device. A comprehensive overview of the devices utilized for this testing can be seen in the table below. These devices were specifically selected as they inhibited different specifications, screen sizes, android versions and API levels.

Name	Manufacturer	Android Version	API Version	Screen Size (Inches)	Screen Resolution	CPU (GHz)	RAM
Galaxy Tab 4	Samsung	5.1.1	API 22	10.1	1280 x 800	Quad Core 1.2	1.5GB
Smart Ultra 6	Vodafone (ZTE)	5.1.1	API 22	5.5	1920 x 1080	Octa Core (4x1.5, 4.1.1)	2 GB
Moto G4	Motorola	6.0.1	API 23	5.5	1920 x 1080	Octa Core (4x1.5, 4x1.2)	2 GB
Galaxy S6	Samsung	7.0	API 24	5.1	2560 x 1440	Octa Core (4x2.1, 4x1.5)	3 GB
Galaxy S7 Edge	Samsung	7.0	API 24	5.5	2560 x 1440	Octa Core (4x2.3, 4x1.6)	4 GB

Figure 139 - Test Device Specifications

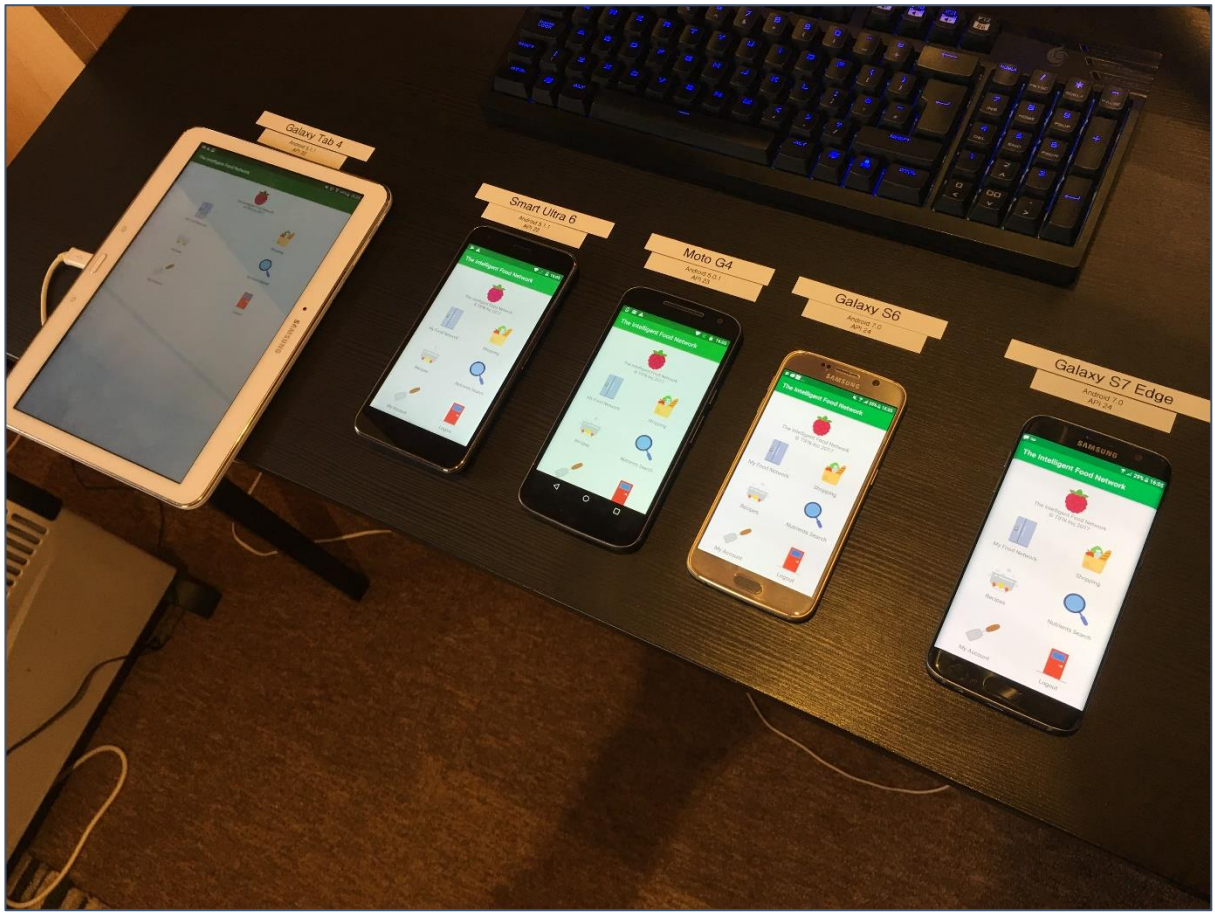


Figure 140 - Test Devices Labelled on Desk

8.1.2 API Testing

API's are extensively used within the Android Application; therefore it was mandatory that all responses returned to the application would match the API call. All API endpoints were first tested using the Rapid API built in console before they were ported to the Application. This would ensure that the application was receiving exactly what was expected.

8.1.3 Security Testing

One of the biggest currently in IT is Cyber Security. With more and more hacking and internet thefts being reported every day, software systems must implement adequate security protocols to protect their userbase. Although not a major requirement it the IoT stream, the application still had to be made secure. After discussions with how to approach this a fellow student in NCI studying Cyber Security recommended "htbridge.com" as the platform of choice for testing mobile applications. This site is free and provides developers with the option of uploading an APK file or specifying an application currently on the Google Play Store. The online platform will then take roughly 20-30 minutes to scan the Application and return the results. The Results of this testing methodology can be seen in section "***8.4.1 – Internal Testing Results***"

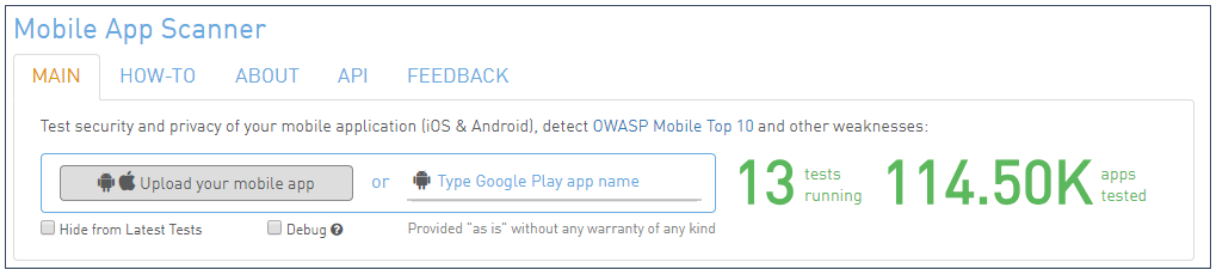


Figure 141 - APK Upload Dialog

8.2 External Testing

Once all required internal testing was complete, external testing could take place. A broad range of different candidates were selected for the external testing process. A total of 10 candidates were selected. All surfaced from different backgrounds. The candidate population consisted of both males and females varying in age. The testing took place in an Engineering Company throughout the course of the day (The Company I am currently employed for). The Controls office was selected as the location wherein the testing could take place. This office is located near the bottom end of the building near the storage and board rooms (Which are usually quite on Fridays)– therefore it was an ideal place to conduct the user testing.

The "*Test Bed*" consisted of A Mini Refrigerator, Plastic Foods, Raspberry Pi, RFID Controller and Android device which were setup on a desk in the office. Candidates were first approached to receive verbal confirmation that they would be willing to take part. Next, they were asked to sign the "***Ethics Disclosure Form***". This document was signed by both myself and the candidates with an appropriate Date Stamp beside each signature. N.B This form can be seen in "***Appendix C***".

The external testing process was broken down into two categories:

1. Usability Testing
2. Application Review

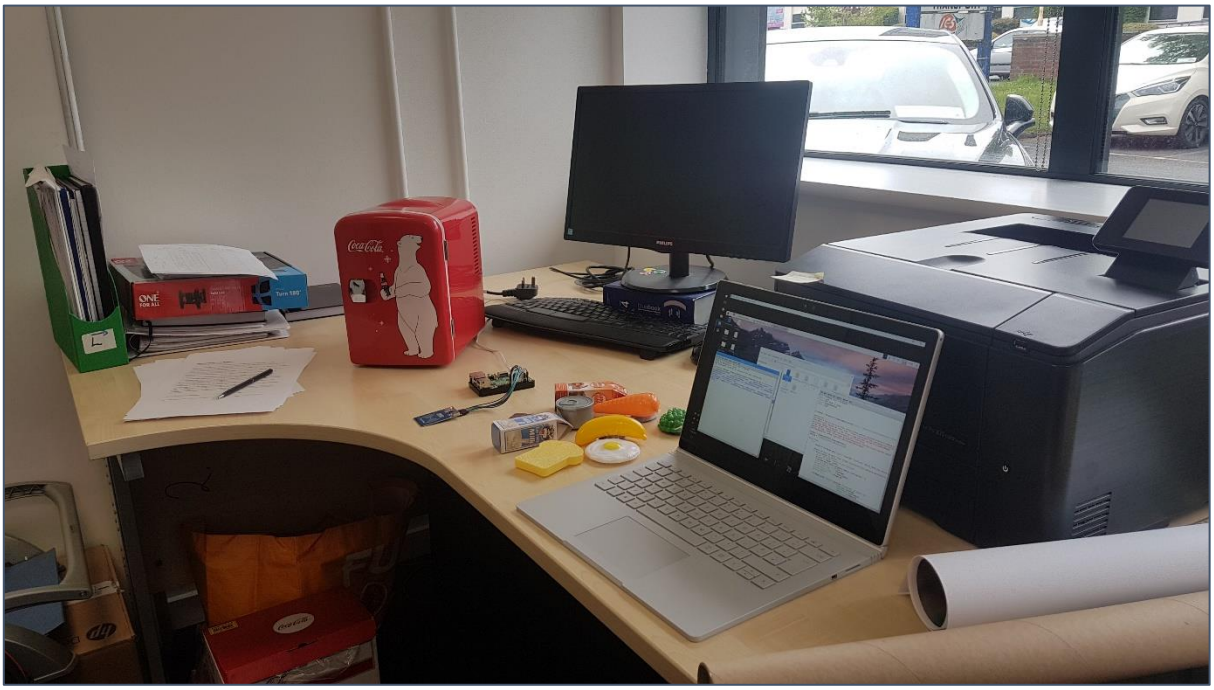


Figure 142 - Testing Setup in Controls Office

8.2.1 Usability Testing

Usability testing was used to test the primary functionality and User Interface of the Intelligent Food Network platform. Usability ties together overall UI with back end functionality. The ultimate aim was to design a system which would inhibit a competent back-end but also be able to flow from section to section with a clean and easy to use UI. Each Functional Requirement was tested here.

A total of 21 “tasks” were created for this testing stage and were based around the Use Cases. The “tasks” were then printed and laid out on the desk. The Users were then asked to take a seat at the “*Testing Desk*” and work through the list of tasks. All required resources for the testing were provided, therefore the user did not need any of their own equipment. The Application was tested on a “***Samsung Galaxy S7 Edge***”. A second sheet was created in Excel. This sheet was used by the testing facilitator to time each task. An option to record issues with any of the tasks and additional comments box were also added.

Both Task and Time Documents can be seen below.

Task Sheet

External Testing – System Usage

Please follow the below steps, taking as long as required to complete each action.

1. Register for a new Account
2. Log Out
3. Login with the newly created account
4. Scan 5 Food items of your choice into the Refrigerator
5. Remove one item from the Refrigerator
6. Place the same item back into the Refrigerator
7. Open the "***My Food Network***" section and find nutritional values for one of the items previously placed into the Refrigerator
8. Return to the "***My Food Network***" section. Locate a Recipe for one of the items in the Refrigerator
9. Now, favourite the Recipe you just found
10. Share the recipe you just found
11. While still looking at the Recipe, add one of the ingredients to your shopping list
12. While still looking at the Recipe, view the nutritional data for one of the ingredients
13. Return to the Homepage, then locate "***My Shopping List***" section
14. Manually add an item to the Shopping List
15. Return to the Homepage. Locate the "***Favourite Recipes***" section
16. From the "***Favourite Recipes***" section, view the Favourite recipe that you added earlier
17. From the "***Favourite Recipes***" section, return back to the "***Recipes***" section, then search for a recipe of your choice
18. Select a recipe from the list of results, then add it to your favourites
19. Return to the Homepage. Now, find nutritional values for any item of your choice
20. Return to the Homepage. Navigate to the "My Account" section, then change your profile image. When complete, update your profile to save the new profile image.
21. Finally, in the "My Account" section, contact the developers to provide some feedback on the application. (Examples include:
    - How easy was the scanning process
    - How easy the app was to navigate
    - Thoughts on app colour scheme
    - What features were most useful
    - If you would use the app day-to-day)

Thanks for your participation,
Leon.

Figure 143 - Task Sheet

Time Sheet

External Testing - System Usage
Time Taken

Leon Mulvaney 14445618 BSHC4IOT

Candidate Number: 1 **Date:** 11/05/2018

Step Number:	Time Taken:	Issues with step:
1		
2		
3		
4		
5		
6		
7		
8		
9		
10		
11		
12		
13		
14		
15		
16		
17		
18		
19		
20		
21		

Additional Comments/Observations:

Figure 144 - Time Sheet

8.2.2 Application Review

Once the Users were finished the Usability testing, they were asked to fill out a quick Review from their experience of using the entire system. This Review was kept limited to 10 questions - with each answer rating a system component. The below example can be seen below.

External Testing – System Review

1	On a scale of 1-10 (10 being very easy), how easy did you find the "***Login and Registration***" process?	1 2 3 4 5 6 7 8 9 10 Very Hard Very Easy
2	On a scale of 1-10 (10 being very easy), how easy did you find the "***Scanning and Removal***" process?	1 2 3 4 5 6 7 8 9 10 Very Hard Very Easy
3	On a scale of 1-10 (10 being very useful), how useful was the "***Nutritional Values***" section for you?	1 2 3 4 5 6 7 8 9 10 Not Useful Very Useful
4	On a scale of 1-10 (10 being very easy), how easily did you "***find a Recipe***" based on an item in the Refrigerator?	1 2 3 4 5 6 7 8 9 10 Very Hard Very Easy
5	On a scale of 1-10 (10 being very useful), how useful was the "***Favourite Recipe***" option for you?	1 2 3 4 5 6 7 8 9 10 Not Useful Very Useful
6	On a scale of 1-10 (10 being very useful), how useful was the "***Add Ingredient to Shopping List***" option for you?	1 2 3 4 5 6 7 8 9 10 Not Useful Very Useful
7	On a scale of 1-10 (10 being very useful), how useful was the "***Share Recipe***" section for you?	1 2 3 4 5 6 7 8 9 10 Not Useful Very Useful
8	On a scale of 1-10 (10 being very often), how often would you be willing to use the "***Recipe Recommendations***" based on your experience so far?	1 2 3 4 5 6 7 8 9 10 Not Often Very Often
9	On a scale of 1-10(10 being very often), how often would you change your profile image on Social Media and similar platforms?	1 2 3 4 5 6 7 8 9 10 Not Often Very Often
10	Could you please rate your overall experience (1 being very bad, 10 being very good)?	1 2 3 4 5 6 7 8 9 10 Very Bad Very Good

Figure 145 - Review Sheet

8.3 Testing Results

8.3.1 Internal Testing Results

As outlined above, the system was tested on 5 physical target devices. The results are as follows:

8.3.1.1 Resource Error

After initially installing the APK onto the Galaxy Tab 4 and Smart Ultra 6, the application would open and attempt to load the Splash Screen. The system would then immediately crash. To figure out what the issues were, both devices were connected to the PC. The Android Studio Logcat console was used to uncover the errors. N.B – The application had previously worked fine through the Emulator and Galaxy S7 Edge throughout development.

The issue was listed as "***Binary XML file line #0: Error inflating class ImageView***". After some research - it was uncovered that some of the Image assets were saved into the "***Drawable v-24***" directory. This wasn't an issue on the Galaxy S7 (Android 7.0), but greatly affected older Android versions. All images were refactored to the "***Drawable***" directory before the APK was re-created. Once this process was complete, the Application booted perfectly – the Splash Screen displayed exactly as designed, with the user then being routed to the Login Activity.

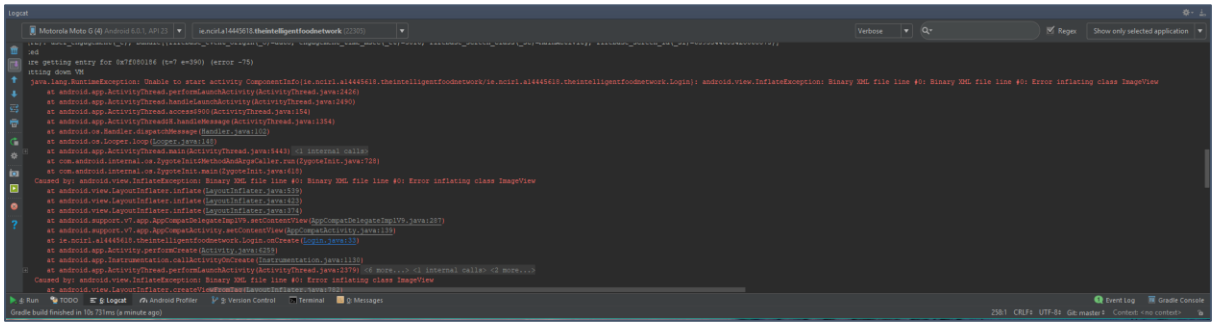


Figure 146 - Resouce Error

8.3.1.2 Soft Touch Buttons

Another issue which arose during the internal testing process was with the Motorola Moto G4. Previously the system had been tested on devices that utilized physical navigation buttons.

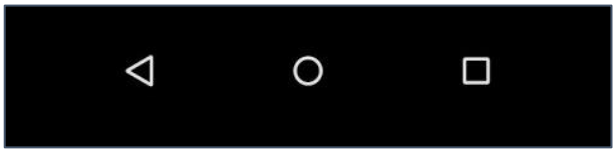


Figure 147 - Soft Touch Buttons

When the APK was installed onto the Moto G4, some sections were cut off. To resolve this issue, various sections in the application were encapsulated with "***ScrollView***" parents. The swiftly resolved the issues with devices that inhibit soft-touch buttons.

8.3.1.3 Use Case Testing Results

As outlined above - all system use cased were tested individually on each test device. Once the initial issues had been resolved, the system ran flawlessly on each target device. This testing process ensured at all the major functionalities would run on lower powered and slower hardware. The application GUI was also designed with portability in mind – the results can be seen in the following images wherein the application scales to each of the 5 devices – all inhibiting different screen sizes and resolutions.

Home Activity

The Home Activity scaled perfectly to each of the target devices. Initially, there was some concern with the Galaxy Tab 4 as it was quite an outlier in terms of screen resolution and size, although the application scaled very well to suit this device.

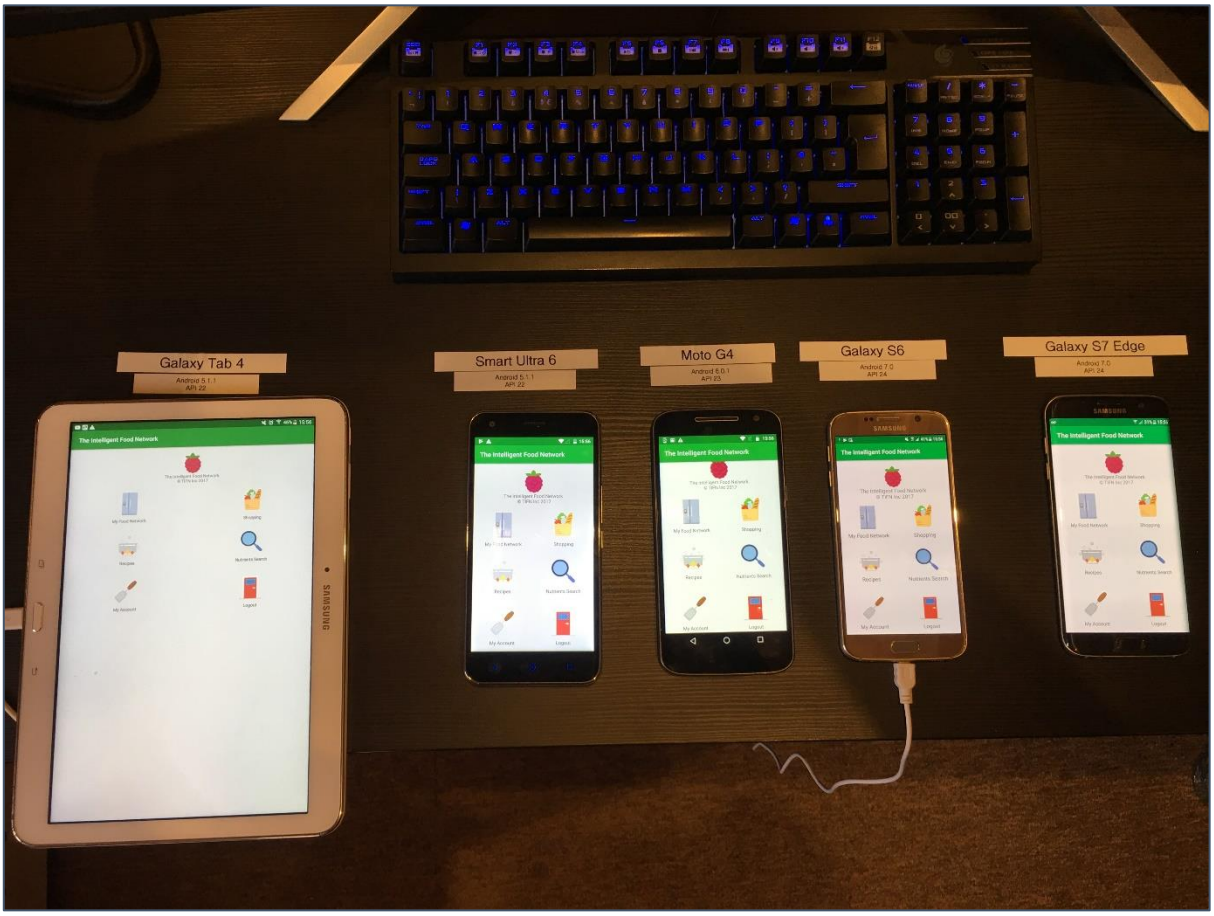


Figure 148 - Home Activity Running on Test Devices

My Food Network Activity

The next section to be tested was the “My Food Network” activity. This section again scaled as designed across all devices. All food contents, text and all UI elements were clearly visible.

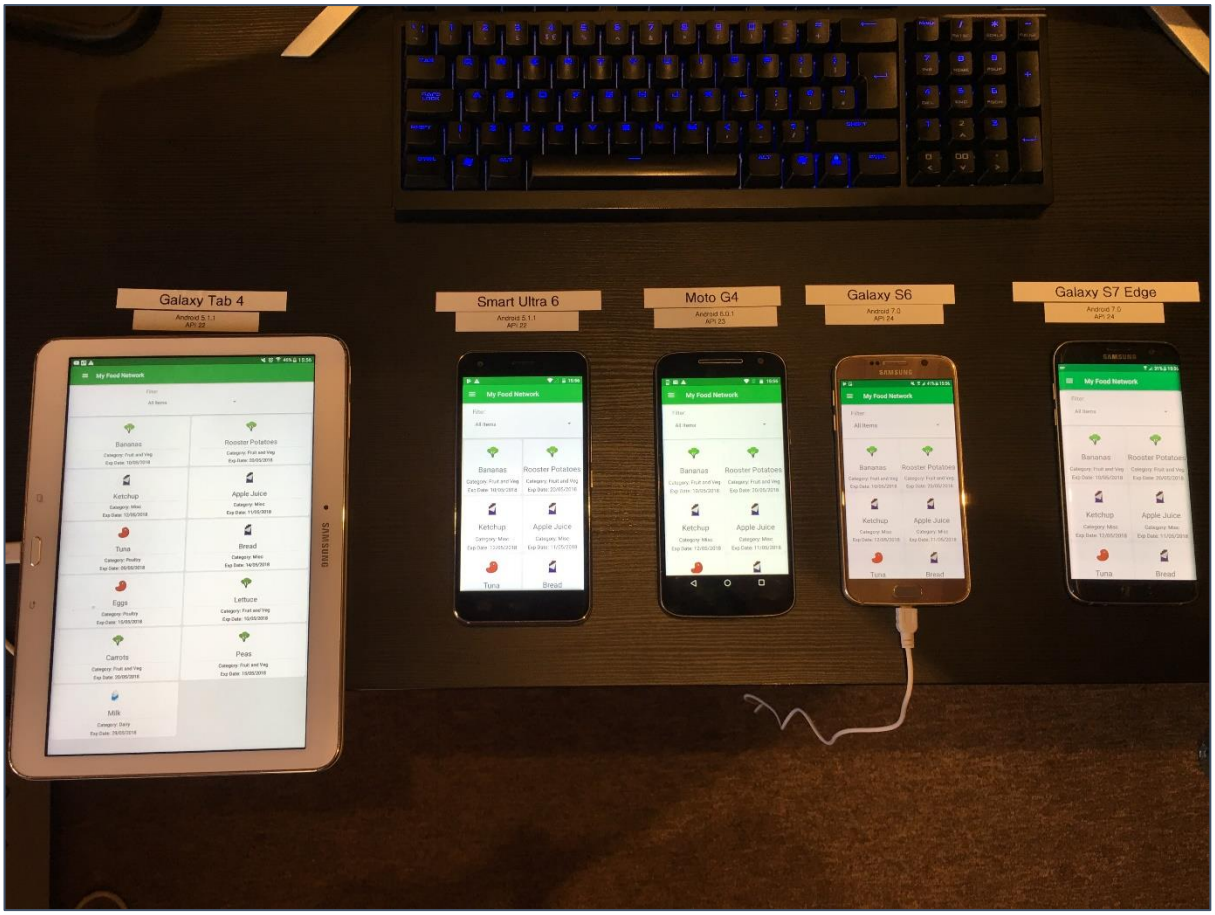


Figure 149 - My Food Network Running on Test Devices

Recipes List Activity

The Recipes List section is one of the most vital sections in the Application. This section scaled all images from the API to each device, keeping all UI elements in their respective positions. There was no delay when loading the Recipes, even on the lower powered devices such as the Galaxy Tab and Smart Ultra 6.

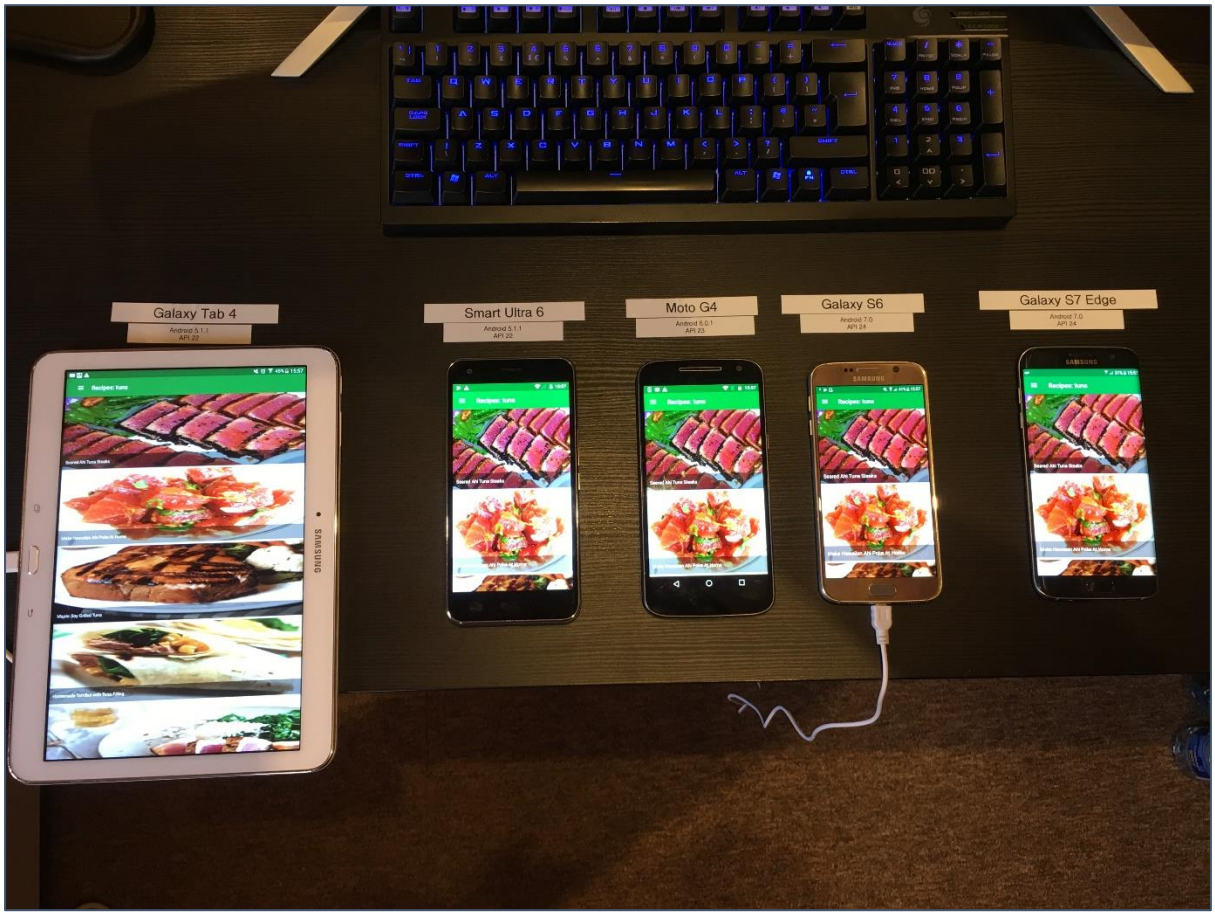


Figure 150 - Recipes List Running on Test Devices

Recipe Details Activity

The Recipe Details activity is the most extensive section in the entire application. Almost every component of the system is loaded here in some way or form. Therefore, this section would prove to be the most demanding in terms of CPU power and Memory requirements. After testing this sections functionality i.e. Adding the Recipe to Favourites, adding items to Shopping List, viewing Nutritional data etc. there was no slow-down or significant errors. This was a great sign and demonstrated that the application was capable of being ported to all targeted devices.

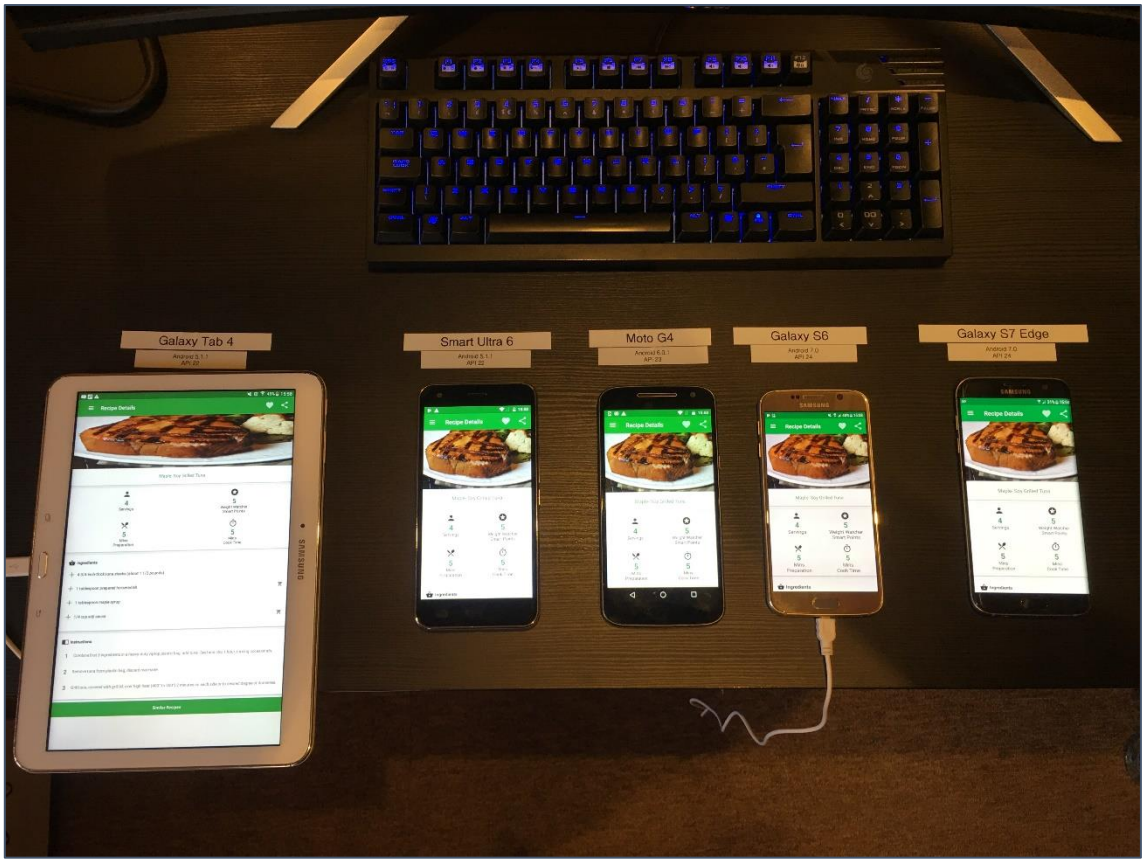


Figure 151 - Recipe Details Running on Test Devices

Shopping List Activity

The shopping list section grabs multiple image URL's and parses them to the UI using the Picasso library. As such, this section can become quite resource-hungry if there are a lot of Shopping List items.

Before this section was loaded, the Shopping List size was drastically increased. This would put additional strain on each device's resources. Once opened, this section loaded all images with relative ease. One thing that was noted was that the Galaxy Tab 4 took slightly longer to finish loading all food images. One of the primary resources utilized for this section is RAM – the Tab 4 inhibiting the lowest amount. Whilst not "slow" by any means, the Tab did take an additional second or two to finish loading the UI elements. This was particularly evident when all devices were laid out on the table.

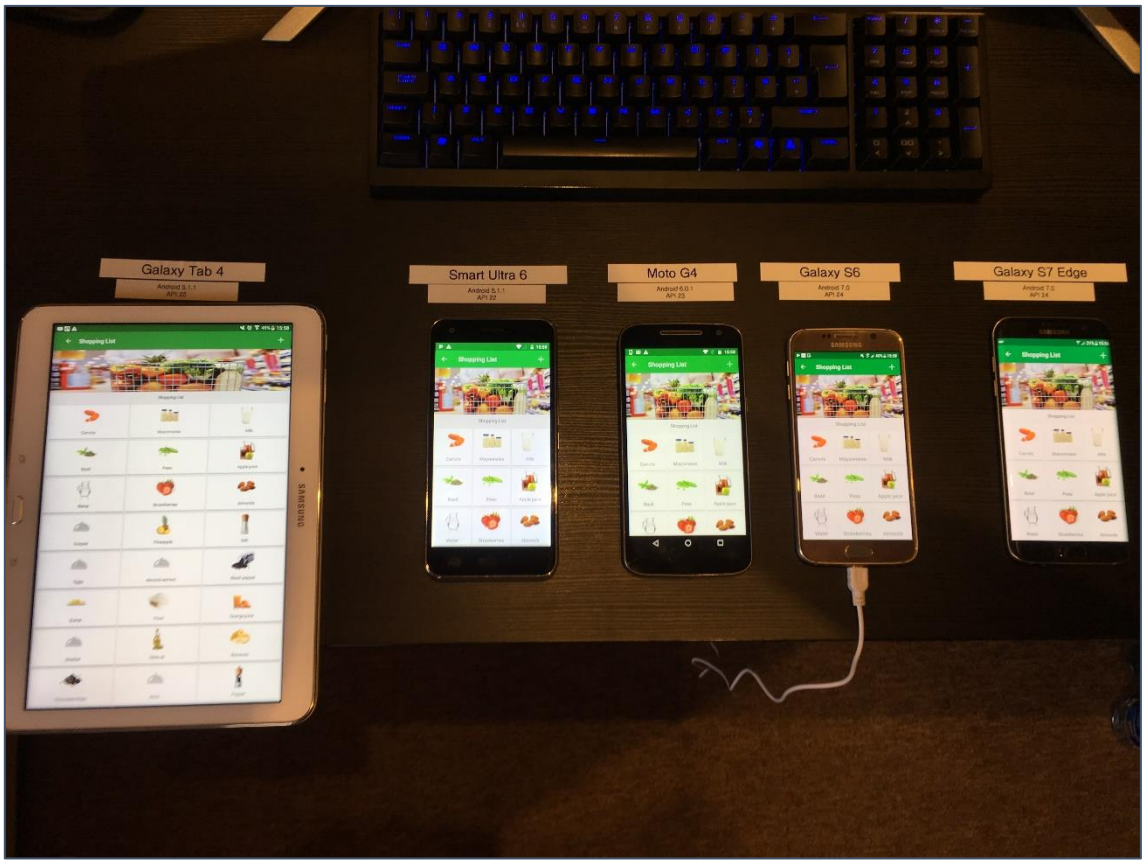


Figure 152 - Shopping List Running on Test Devices

Recipes Activity

The Recipes section was tested next. This section again features a significant amount of different background tasks – the primary being the creation of a Recipe Recommendation. This section loaded and parsed all UI elements on each device perfectly during the testing process.

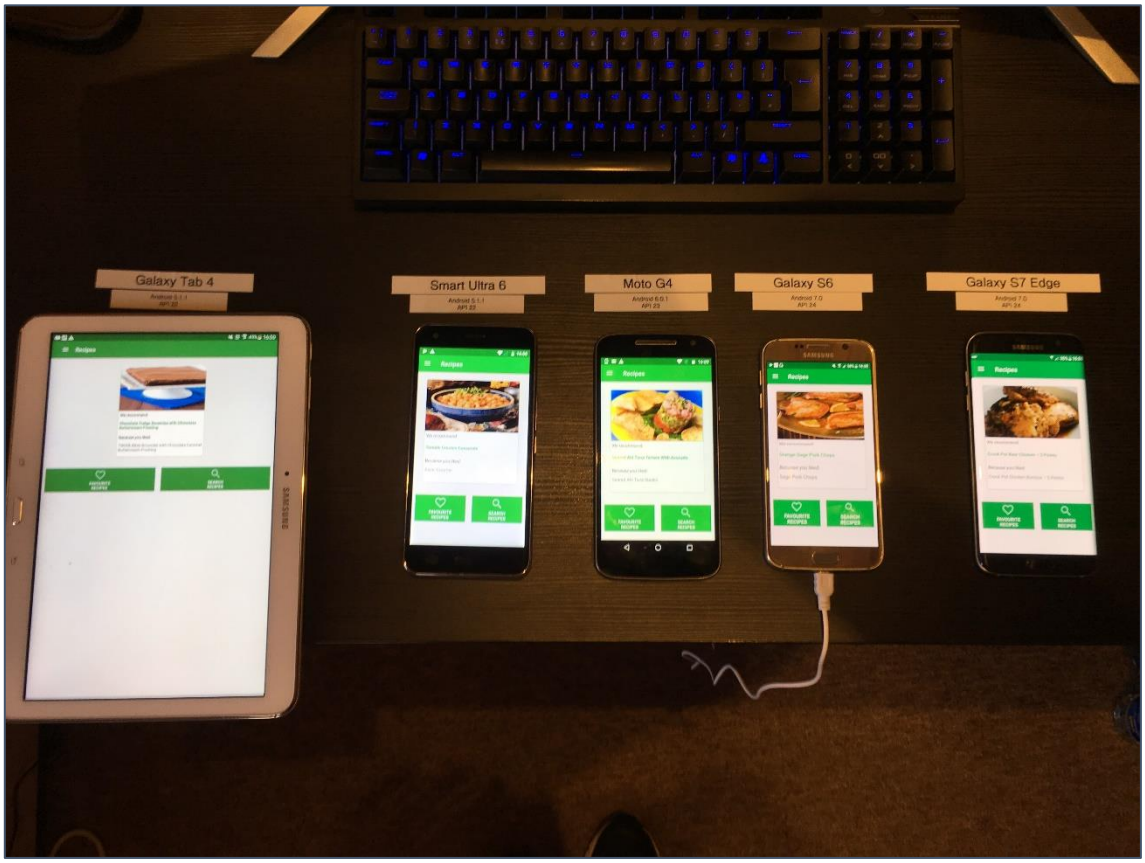


Figure 153 - Recipe Recommendation Running on Test Devices

Favourite Recipes Activity

This section also utilizes the Picasso library to parse images from URL's. All elements loaded as designed, with no significant slow-down or issues arising.

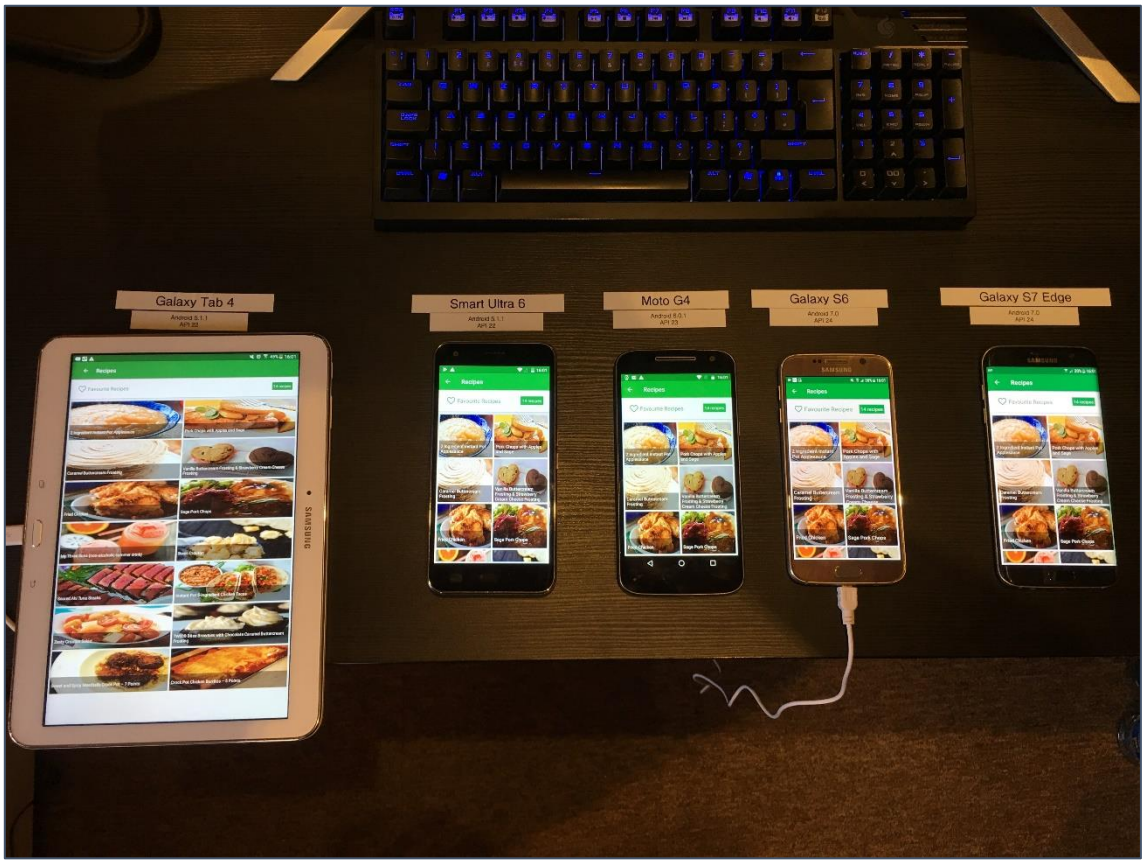


Figure 154 - Favourite Recipes Running on Test Devices

Edit Profile Activity

This section pulls a lot of profile data, calculates counters for Favourite Recipes etc. It is also one of the only sections of the Application to pull data from Firebase Storage. The profile image loaded and scaled perfectly, the counters populated, and all UI elements were displayed in their respective positions.

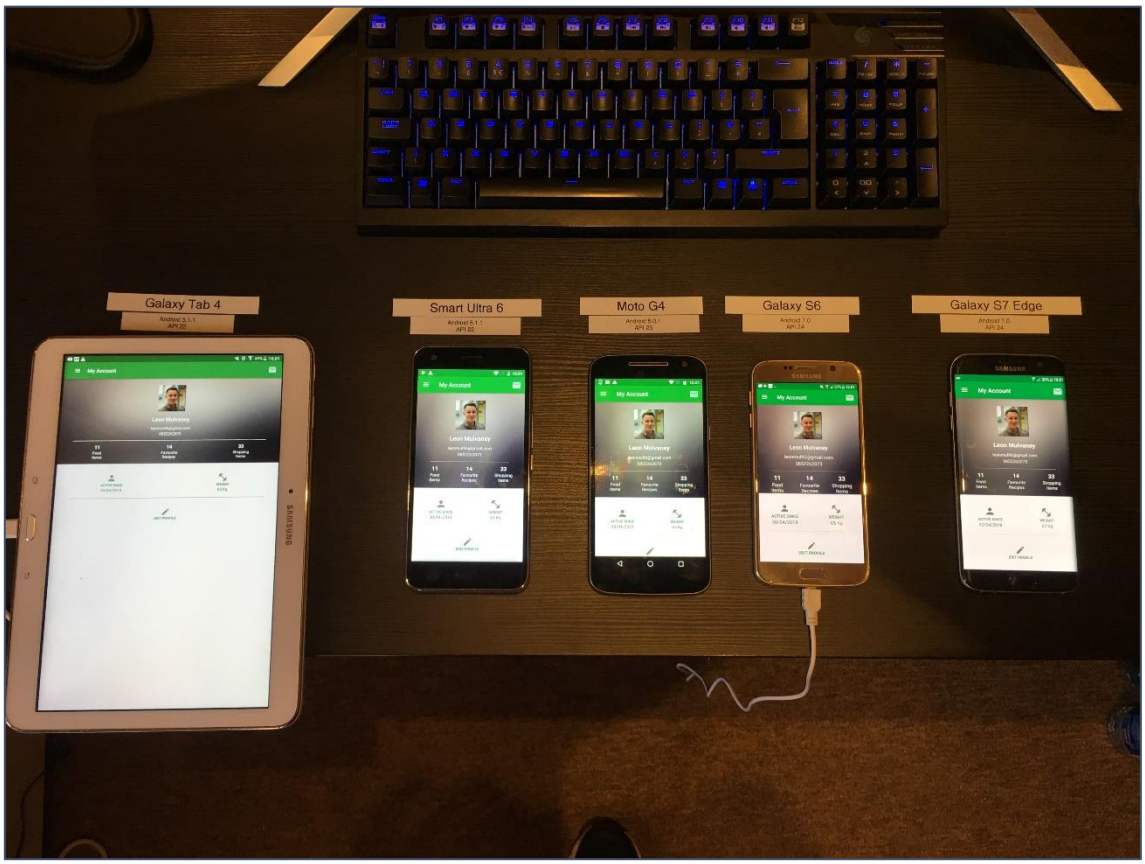


Figure 155 - Edit Profile Running on Test Devices

Edit Profile & Upload Image

The final functionalities to be tested were Edit Profile and Upload image. These sections worked perfectly with all devices populating the UI and loading all associated data swiftly. Each device could select an image from any of the build in Device file managers/ Gallery application.

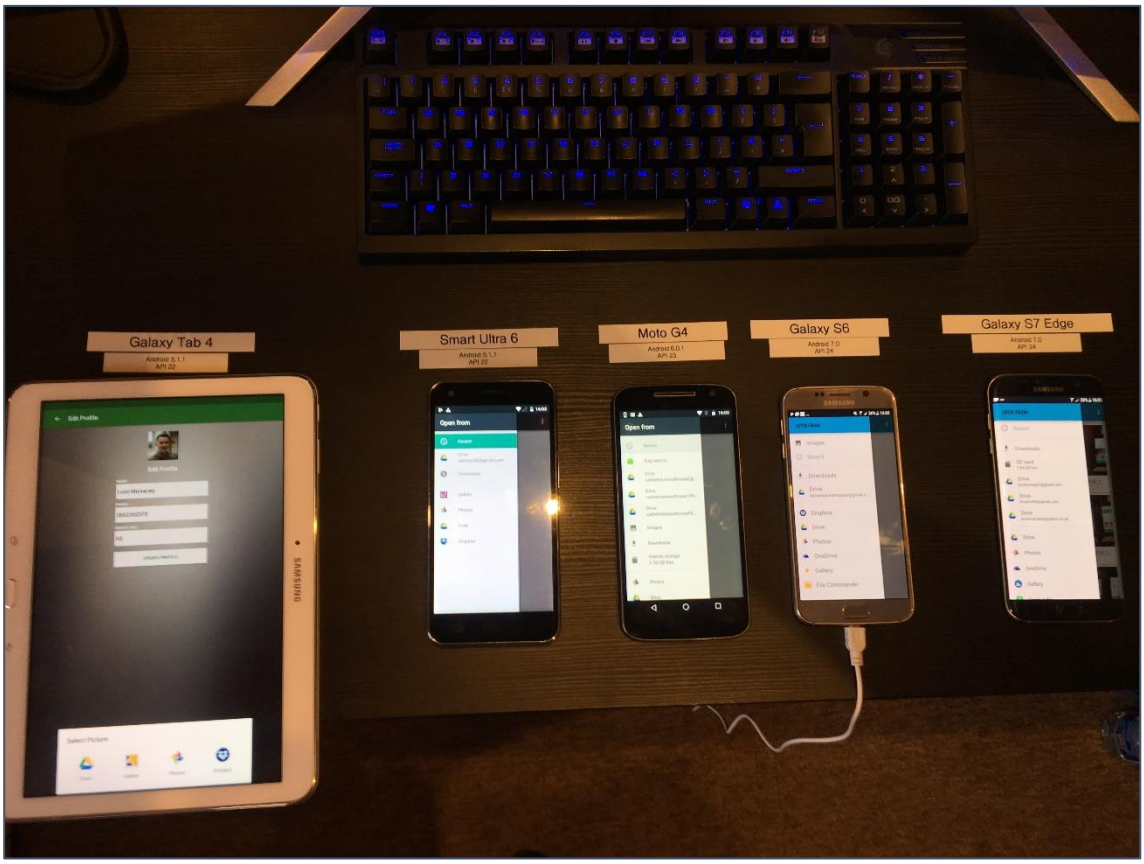


Figure 156 - Profile Image Upload Running on Test Devices

8.3.1.4 API Testing Results

Each API endpoint was called in the application multiple times. This was to test and see if there was any considerable slow down. Local 5Ghz Wi-Fi and 4G Mobile Networks were utilized for this test. Both returned successful results with all information pulling from all endpoints at a comparable time. The Endpoint results were set so they printed in the Android Logcat Terminal. This result was compared to the same result via the Rapid API console. One such example was the “GET Recipe via ID” request. The ID was printed to the Android terminal, then entered into the Rapid API console for comparison.

8.3.1.5 Security Testing Results

The Android APK file was uploaded to "***htbridge.com/mobile***". The test took 11 minutes and after approximately 33 minutes the results could be accessed. The results are as follows:

- 5 Warnings
- 6 Low Risk Threats
- 2 Medium Rick Threads
- 0 High Risk Threats

An overview of each threat was described. After analysing these threats, it was uncovered that most did not affect the overall system security – this was indicated as there were 0 High Risk threats - a very successful result. As mentioned previously, the system design was derived from a "Thin-Client" architecture. (See section "***3.2.1***"). The utilization of the Google Firebase Framework for data storage inevitably meant that the system would be more secure.

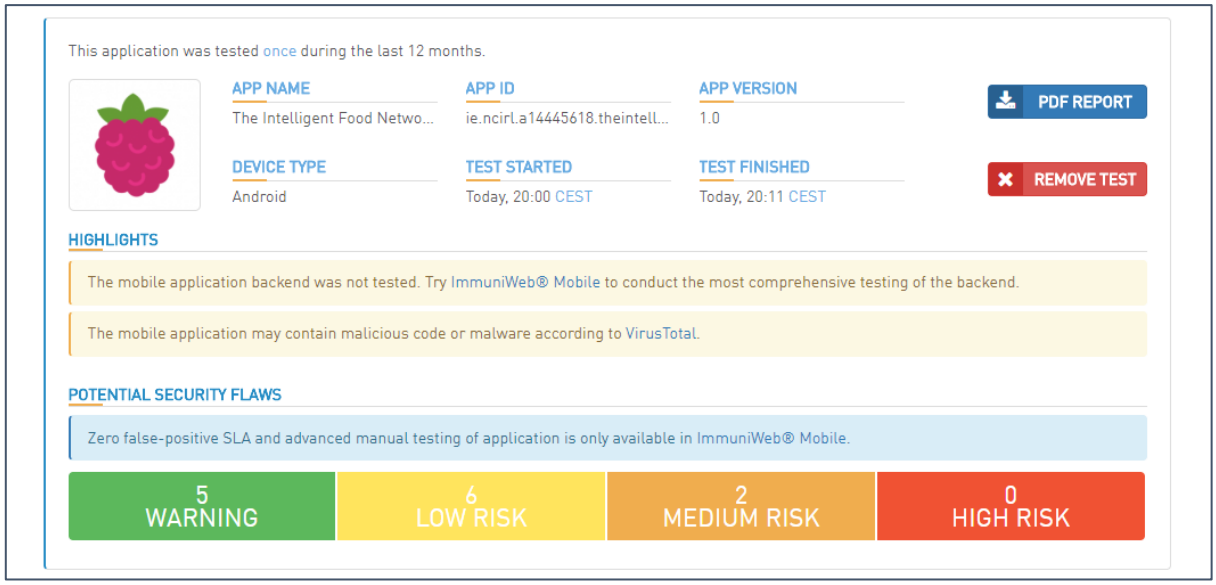


Figure 157 - Security Results Overview

Many of the risks highlighted from the testing process touched upon elements such as the use of a "Predictable Random Number Generator" – i.e. to generate recipes recommendations from the API results and other minor gripes such as "Enabled Debug Mode" which is required for development of the application. Perhaps the most formidable risk on this list would be Medium risk no 1 – "Usage of Unencrypted Http Protocol". As outlined in section "***3.2.6 – Security Requirement***", the system is primarily designed to function on a secure local or 4G network – this completely negates the medium risk. Also, the

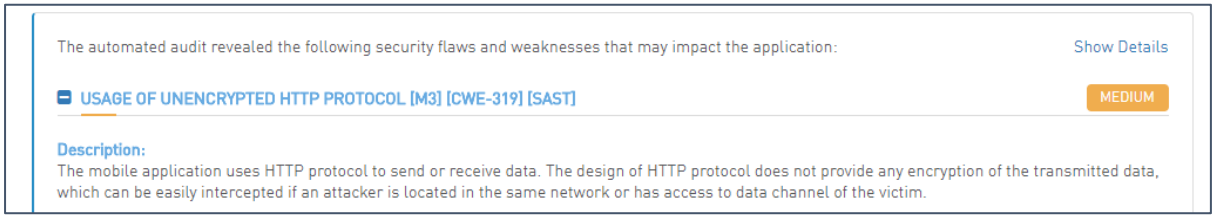


Figure 158 - Unencrypted HTTP Protocol Threat

The complete Mobile Security Audit can be seen below:

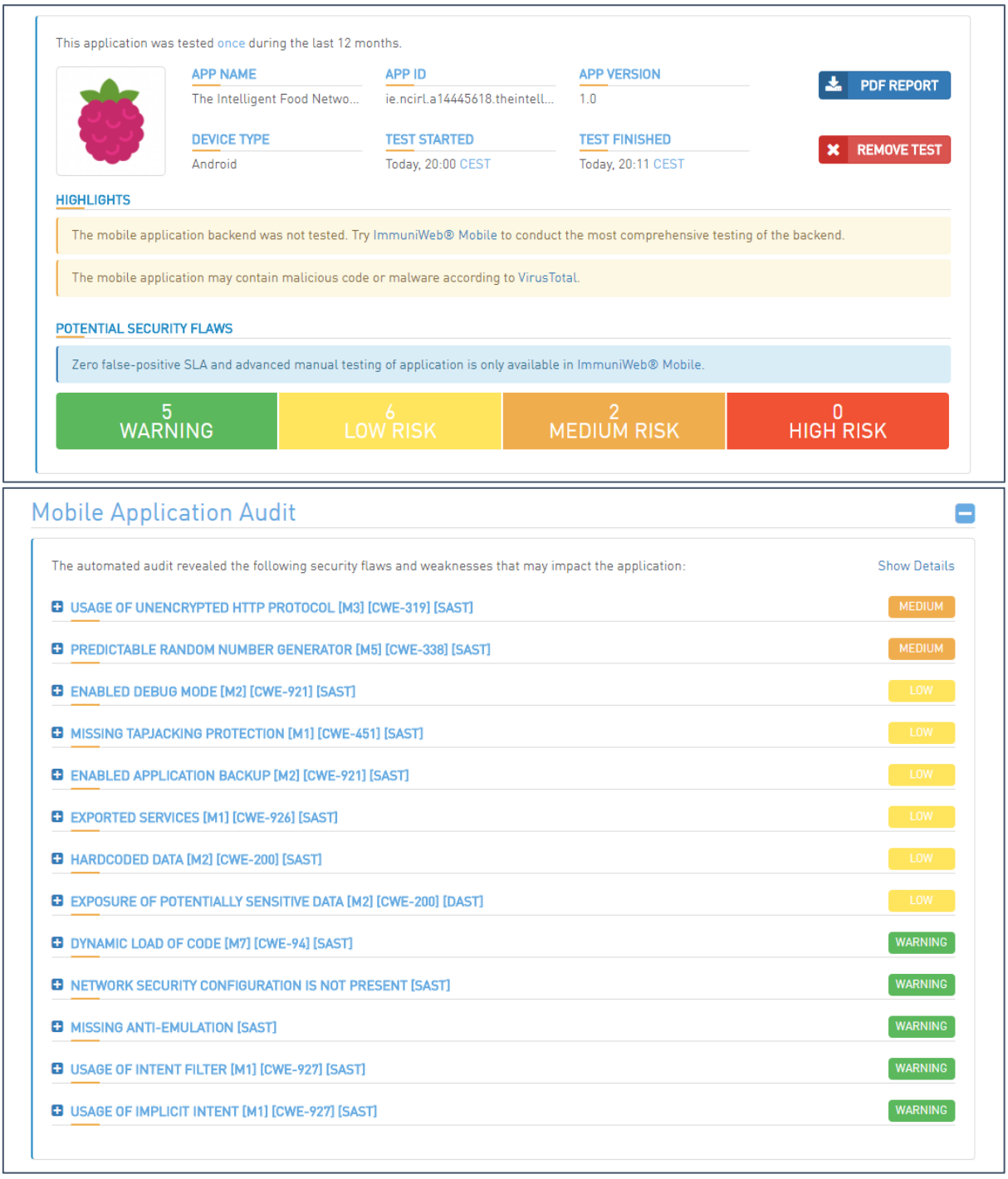


Figure 159 - Complete Security Audit

8.3.2 External Testing Results

All 10 candidates were timed throughout Usability Testing and all filled out the Review form thereafter. As mentioned previously, the Application was tested on a "***Samsung Galaxy S7 Edge***" Android device, although there was one exception. The first test candidate had recently purchased a "***Samsung Galaxy S8 Plus***". This was a great opportunity to test the Application on another alternative form of hardware. The Galaxy S8 was running on Android 8.0 API 26 and possessed a 21:9 screen aspect ratio. This was significantly different from any other devices that were tested during the Internal Testing process. The candidate was asked if they would like to use their own mobile device for the testing. They agreed and the APK was installed on the S8 Plus. Testing on this device was captured using the S7 Edge. This testing process has been outlined below with accompanying images.

8.3.2.1 Timing Results

After all the tests were completed, the times were transcribed into an excel spreadsheet to work out the averages. For comparison, a personal timing was done to see how long the actual developer would take – this was approx. 237 seconds. The results were all quite similar across the board – the outliers being candidates 3 and 8 taking over 400 seconds to complete the tasks. This was primarily down to the "***Contact Developers***" section being not clearly labelled. This issue was solved by adding the text "Contact" instead of the email icon. Also, one user had slight difficulty finding the "***Manually add to Shopping List***" button. This has also been updated to include the text "Add" instead of the plus icon. Overall, results were very good – as expected the developer timing is always going to be significantly faster, although the results were all within the same time range which suggests that the application and overall system appeals to its targeted user base. N.B All Time Test Sheet can be seen in "***Appendix C***".

B20 | 10

	A	B	C	D	E	F	G	H	I	J	K
1	Candidate Number	1	2	3	4	5	6	7	8	9	10
2	**1**	60	50	68	54	62	57	65	64	59	61
3	**2**	3	4	5	4	6	5	5	5	6	5
4	**3**	12	10	16	14	15	14	14	16	10	12
5	**4**	38	28	37	40	34	41	38	42	41	32
6	**5**	5	3	3	4	4	4	4	4	4	4
7	**6**	3	3	3	3	3	4	3	4	4	4
8	**7**	10	15	16	12	16	15	16	16	21	15
9	**8**	7	14	10	11	12	13	13	12	11	14
10	**9**	10	8	11	5	7	8	8	9	9	9
11	**10**	7	12	11	9	5	8	7	7	6	6
12	**11**	3	4	5	4	6	4	5	6	5	5
13	**12**	4	3	4	5	7	5	15	5	9	6
14	**13**	8	10	14	12	13	14	14	15	11	14
15	**14**	7	5	20	18	16	17	17	16	16	14
16	**15**	5	8	4	3	5	5	5	9	4	7
17	**16**	3	7	9	5	6	5	9	6	9	8
18	**17**	14	17	17	16	15	16	15	14	14	15
19	**18**	14	7	9	10	12	11	10	11	11	9
20	**19**	10	14	14	16	14	14	16	16	29	12
21	**20**	40	45	55	62	51	55	49	60	57	52
22	**21**	40	70	80	57	50	60	51	84	61	47
23	***Total Time***	***303***	***337***	***411***	***364***	***359***	***375***	***379***	***421***	***397***	***351***
24	***Overall Average***	***370***									
25											

Figure 160 – Usability Result Calculations in Excel

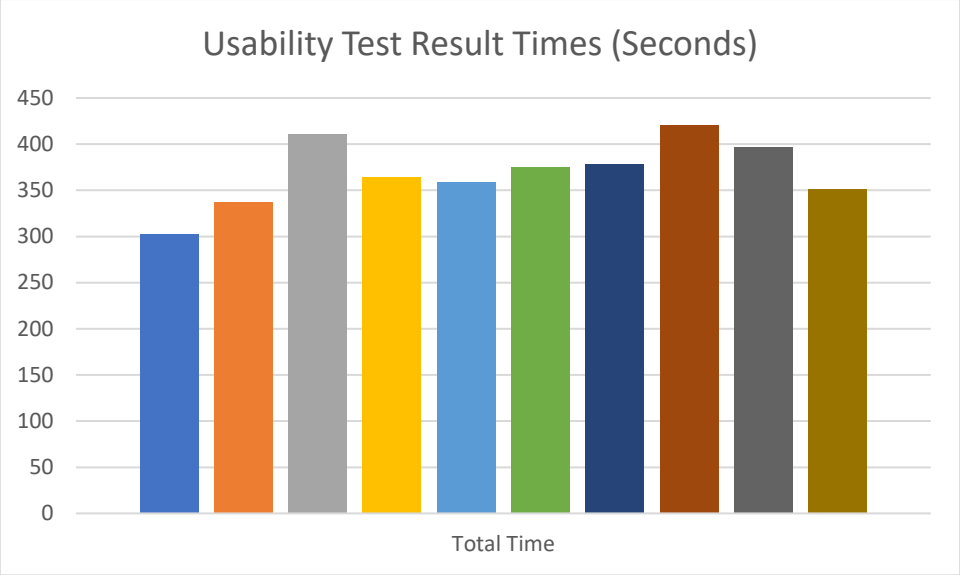


Figure 161 - Usability Results Bar Chart

Changes to UI based on issues during testing:

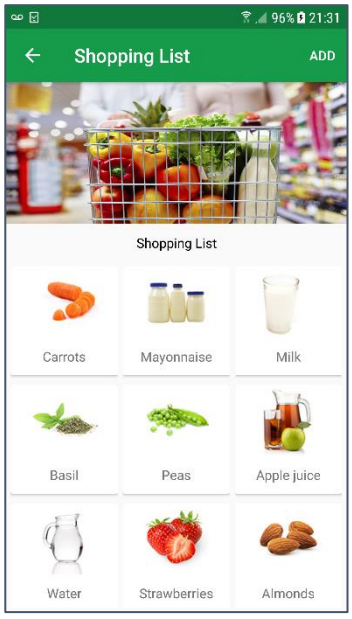


Figure 163 - Add symbol replaced with "Add"

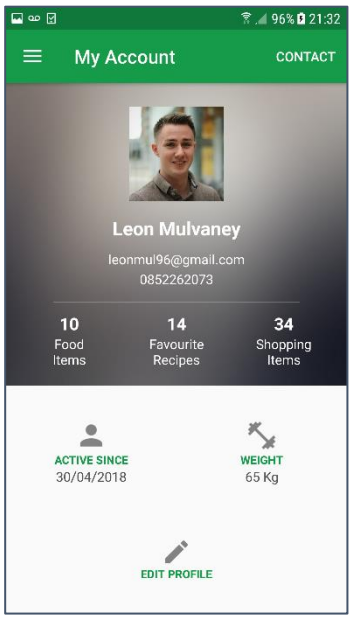


Figure 162 - Email symbol replaced with "Contact"

Candidate 1: Testing Using Galaxy S8 Plus

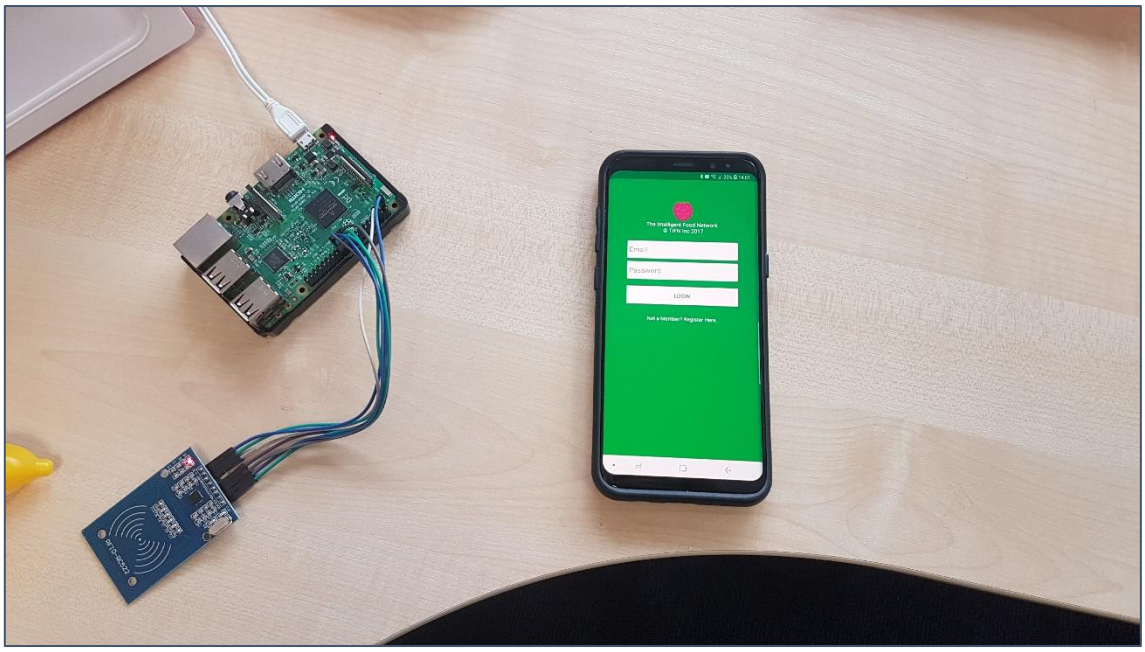


Figure 164 - App Running on Candidate 1 Device (Galaxy S8 Plus - Android 8.1 API 26)

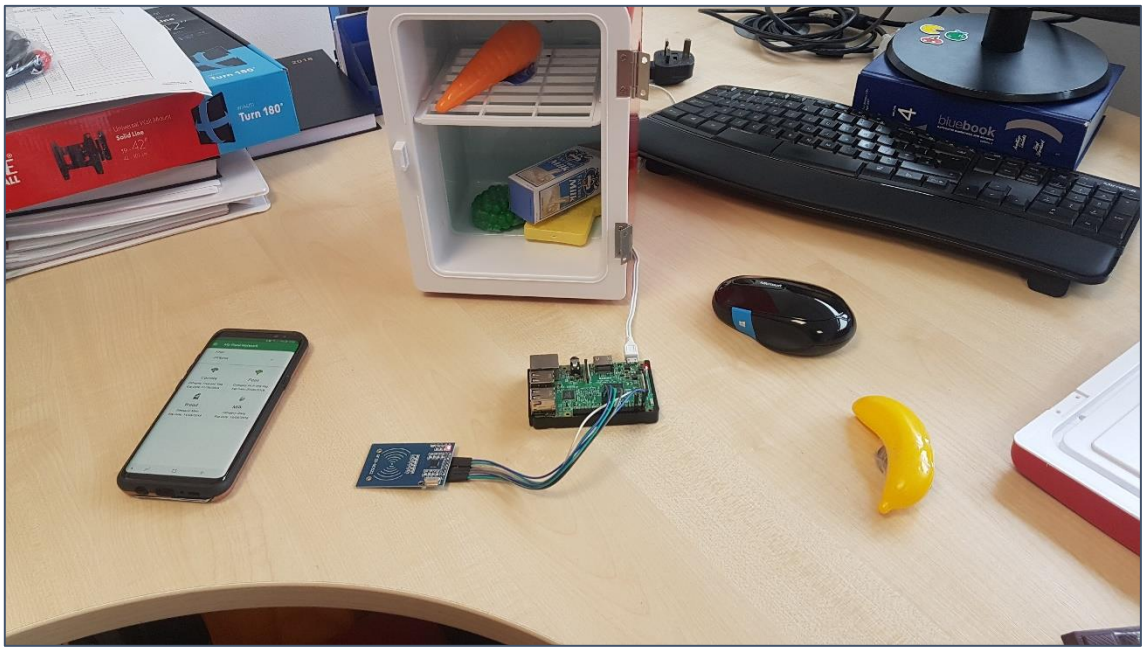


Figure 165 – Test Candidate Scanning Items into Refrigerator during Testing

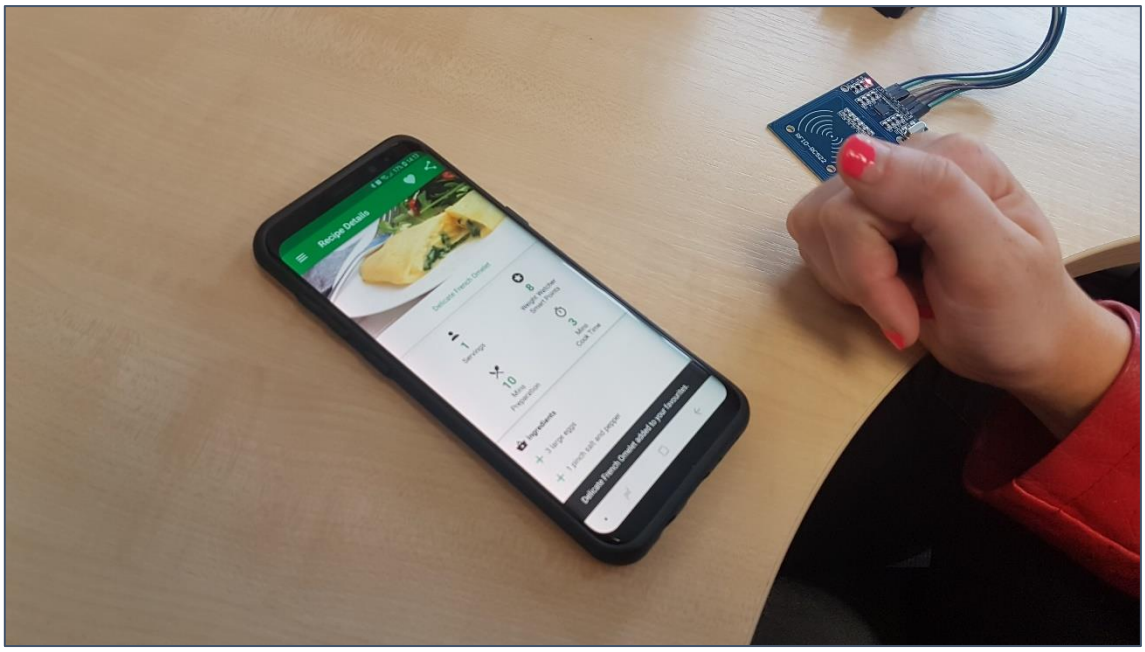


Figure 166 - Test Candidate Viewing Recipe

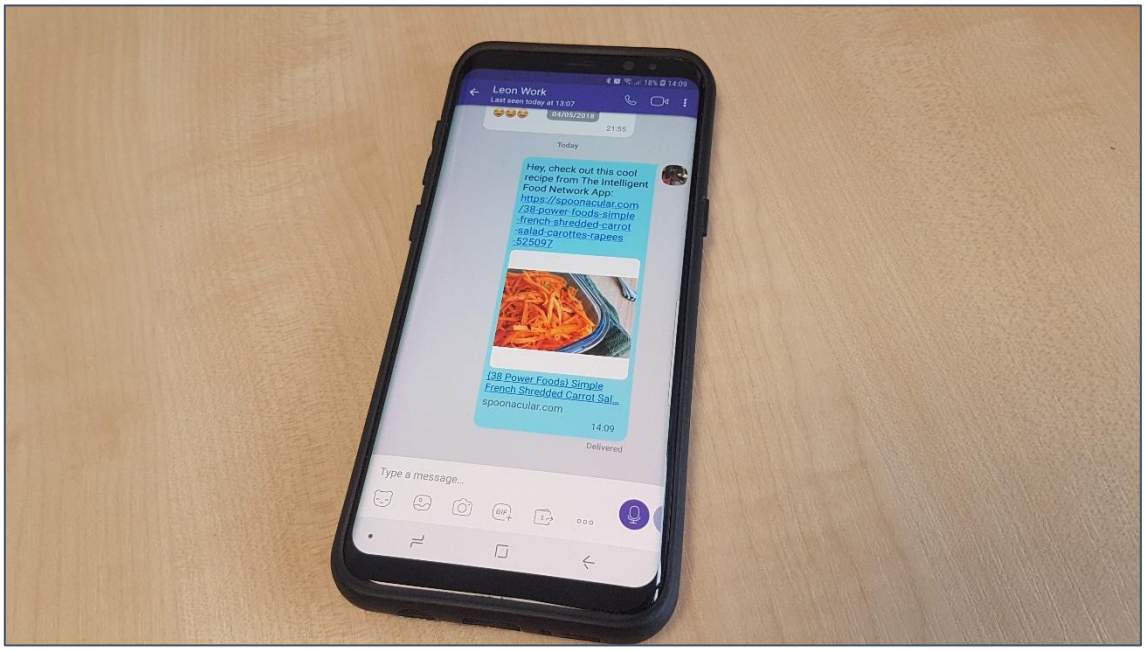


Figure 167 - Test Candidate Sharing Recipe

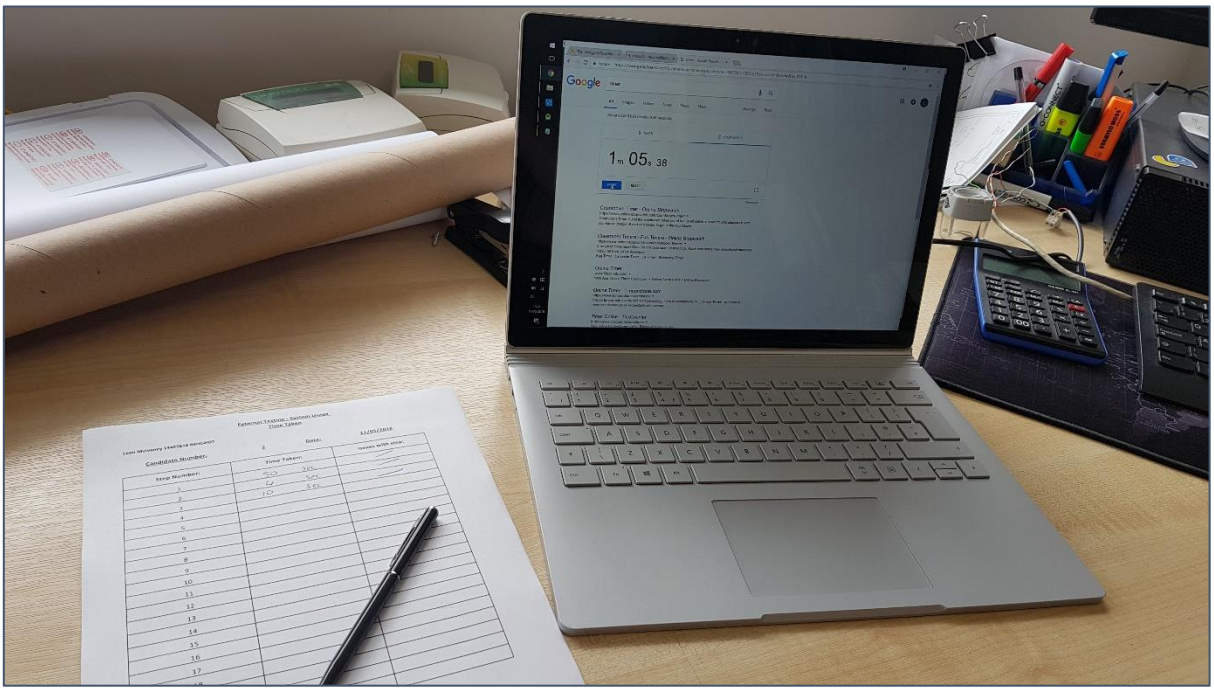


Figure 168 - The Opposite Desk where the Times were noted

8.3.2.2 Review Results

The Review Form Results were taken and again transcribed into Excel to calculate the averages and create some accompanying charts. Although the test candidates were all work colleagues they were requested to be as honest as possible when filling out the Review. The target average was 80% as this would represent a 4* application on the Google Play Store. The Application received an 87% average from all the ratings – 7% above the target. This result alongside the usability results display that the application has been able to appeal to its target audience.

Question 5 – "How useful was the Favourite Recipe Section" achieved the highest average for a single question scoring an average of 93%. The Favourite Recipes section is one of the primary features of this application, therefore this was a very good indication once again.

Question 9 – "How often would you change your profile image on Social Media and similar platforms" scored the lowest average with 76%. This question was expected to be an outlier as the test candidates ranged from ages 22-55. The Review Results can be seen in "***Appendix C***".

	Candidate Number	1	2	3	4	5	6	7	8	9	10	Total For Section
Section	**1**	10	8	9	8	8	8	8	9	9	9	86
	2	9	8	8	9	7	9	9	9	10	10	88
	3	10	9	9	10	8	8	9	8	10	9	90
	4	10	9	7	10	9	9	9	8	9	8	88
	5	9	10	9	10	9	8	10	9	10	9	93
	6	10	10	9	8	7	10	10	8	9	10	91
	7	10	7	9	8	8	7	8	8	8	8	81
	8	10	9	8	10	8	9	10	9	8	10	91
	9	10	8	9	5	8	7	7	5	9	8	76
	10	9	9	9	8	7	10	9	9	10	10	90
	Rating	***97***	***87***	***86***	***86***	***79***	***85***	***89***	***82***	***92***	***91***	
	Overall Average Rating	***87***										

Figure 169 - Review Results

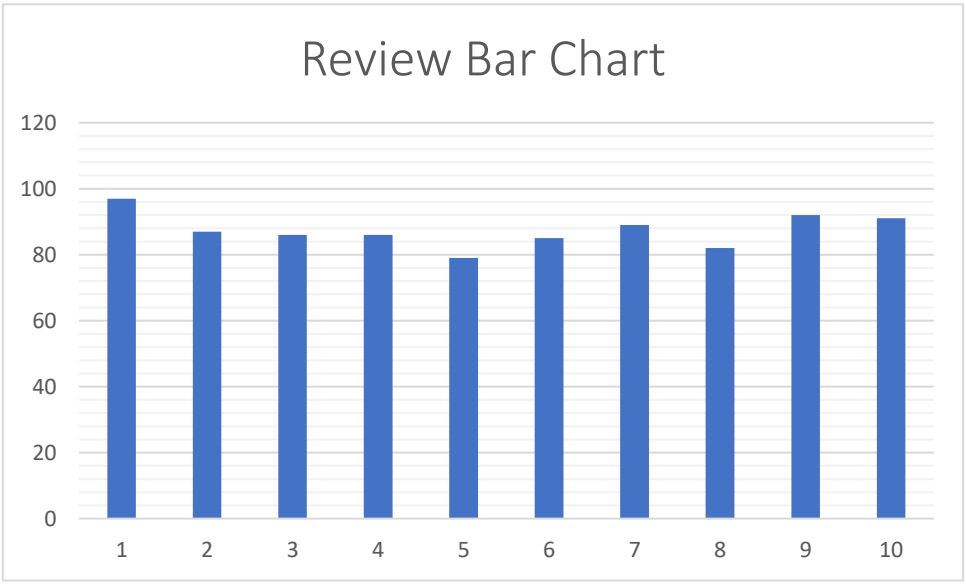


Figure 170 - Review Bar Chart

9 System Evolution

Of course, development time is always an issue – even in industry with many organisations choosing to roll out additional features in later versions of their software packages. If this could have been the case, there are a few additions that would have been made to the final system.

9.1 More Powerful RFID Controller

At present, the MC522 RFID Controller is very useful, it was easy to develop with and works extremely well with the Raspberry Pi. Although, one shortcoming which would have ideally been addressed is the limited range. The Controller more-less requires a User to bring the food item and Controller into direct contact. An ideal scenario would have been to utilize a more powerful industry-standard controller. This aspect of the system had been investigated at a very early stage, although the sheer cost of some of the industrial-controllers was too high. Once particular controller costing over €500 Including Vat in the Radionics store, Rialto, Dublin.

Figure 171 - Industrial-Grade RFID Controller

9.2 Alternative RFID Chips & Technologies

Back in September, a plethora of different RFID Controller types and Chips were purchased. This was primarily for testing purposes to see which solution would best suit the project. It was initially planned to utilize Stickers which could be stuck to the individual food items, although after purchasing many of these it was uncovered that many were simply not compatible with the RDIF Controller. Another NFC Controller was also purchased in an attempt to utilize these stickers, although time constraints meant that this could not occur.

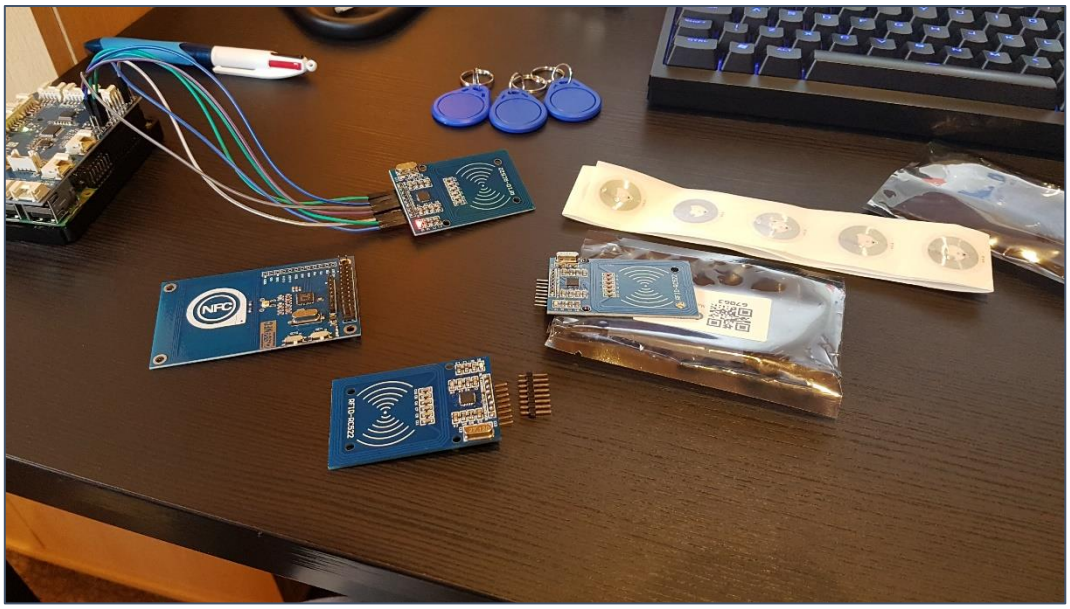


Figure 172 - Assortment of Purchased Equipment

9.3 Additional API Endpoints

The API's utilized in the Project were very useful and held a wide variety of endpoints. The Spoonacular API in particular held quite and extensive number of endpoints which could have been used. Some examples include "GET Random Recipes", "GET Generate Nutritional Plan for the Week" etc.

9.4 GUI Enhancements

The Application GUI was design was designed around Activities. This was useful and assisted greatly in the Development and Debugging processes. If the project could have been started over, more emphasis would be on better UI elements such as Fragments and a more dynamic looking UI. Time was quite limited, as such most was spent on the back-end. It would have been nice to have some more flowing animations, floating pop-up menus and other Material Design UI elements.

10 Bibliography & References

10.1 Firebase

1. *Firebase. 2018. Add Firebase to Your Android Project | Firebase. [ONLINE] Available at: https://firebase.google.com/docs/android/setup. [Accessed 13 May 2018].*
2. *Captech Consulting, Inc.. 2018. Firebase Realtime Database - Android Tutorial. [ONLINE] Available at: https://www.captechconsulting.com/blogs/firebase-realtime-database-android-tutorial. [Accessed 13 May 2018].*
3. *Stack Overflow. 2018. android - Read Data From Firebase database - Stack Overflow. [ONLINE] Available at: https://stackoverflow.com/questions/39800547/read-data-from-firebase-database. [Accessed 13 May 2018].*
4. *YouTube. 2018. Getting Started with Android Firebase - Part 2 - Read Data from Firebase Database - YouTube. [ONLINE] Available at: https://www.youtube.com/watch?v=WDGmpvKpHyw. [Accessed 13 May 2018].*
5. *Firebase. 2018. Saving Data | Firebase Realtime Database | Firebase. [ONLINE] Available at: https://firebase.google.com/docs/database/admin/save-data. [Accessed 13 May 2018].*
6. *Firebase. 2018. Delete Files on Android | Firebase. [ONLINE] Available at: https://firebase.google.com/docs/storage/android/delete-files. [Accessed 13 May 2018].*
7. *Firebase. 2018. Read and Write Data on Android | Firebase Realtime Database | Firebase. [ONLINE] Available at: https://firebase.google.com/docs/database/android/read-and-write. [Accessed 13 May 2018].*
8. *Stack Overflow. 2018. android firebase - get childrens count - Stack Overflow. [ONLINE] Available at: https://stackoverflow.com/questions/43606235/android-firebase-get-childrens-count. [Accessed 13 May 2018].*
9. *AndroidHive. 2018. Android Getting Started with Firebase - Login and Registration Authentication. [ONLINE] Available at: https://www.androidhive.info/2016/06/android-getting-started-firebase-simple-login-registration-auth/. [Accessed 13 May 2018].*
10. *Code Envato Tuts+. 2018. How to Upload Images to Firebase from an Android App. [ONLINE] Available at: https://code.tutsplus.com/tutorials/image-upload-to-firebase-in-android-application--cms-29934. [Accessed 13 May 2018].*
11. *Stack Overflow. 2018. firebase storage - getting image URL - Stack Overflow. [ONLINE] Available at: https://stackoverflow.com/questions/38424203/firebase-storage-getting-image-url. [Accessed 13 May 2018].*

10.2 API's & HTTP Requests

12. *RapidAPI. 2018. RapidAPI. [ONLINE] Available at: https://docs.rapidapi.com/docs/java-android. [Accessed 13 May 2018].*

13. *RapidAPI Marketplace | Recipe - Food - Nutrition. 2018. RapidAPI Marketplace | Recipe - Food - Nutrition. [ONLINE] Available at: https://rapidapi.com/spoonacular/api/Recipe%20-%20Food%20-%20Nutrition/functions/Get%20Recipe%20Information. [Accessed 13 May 2018].*

14. *RapidAPI Marketplace | Recipe - Food - Nutrition. 2018. RapidAPI Marketplace | Recipe - Food - Nutrition. [ONLINE] Available at: https://rapidapi.com/spoonacular/api/Recipe%20-%20Food%20-%20Nutrition/functions/Get%20Similar%20Recipes. [Accessed 13 May 2018].*

15. *RapidAPI Marketplace | Recipe - Food - Nutrition. 2018. RapidAPI Marketplace | Recipe - Food - Nutrition. [ONLINE] Available at: https://rapidapi.com/spoonacular/api/Recipe%20-%20Food%20-%20Nutrition/functions/Search%20Recipes%20by%20Ingredientsecipes. [Accessed 13 May 2018].*

16. *RapidAPI Marketplace | Nutritionix. 2018. RapidAPI Marketplace | Nutritionix. [ONLINE] Available at: https://rapidapi.com/stefan.skliarov/api/Nutritionix/functions. [Accessed 13 May 2018].*

17. *Medium. 2018. Android AsyncTask HTTP GET request Tutorial – Jason Cromer – Medium. [ONLINE] Available at: https://medium.com/@JasonCromer/android-asynctask-http-request-tutorial-6b429d833e28. [Accessed 13 May 2018].*

18. *Stack Overflow. 2018. AsyncTask Android example - Stack Overflow. [ONLINE] Available at: https://stackoverflow.com/questions/9671546/asynctask-android-example. [Accessed 13 May 2018].*

10.3 Listview & Adapters

19. *ListView Tutorial With Example In Android Studio. 2018. ListView Tutorial With Example In Android Studio. [ONLINE] Available at: http://abhiandroid.com/ui/listview. [Accessed 13 May 2018].*

20. *Stack Overflow. 2018. java - Populating a ListView using an ArrayList? - Stack Overflow. [ONLINE] Available at: https://stackoverflow.com/questions/5070830/populating-a-listview-using-an-arraylist. [Accessed 13 May 2018].*

21. *AndroidExample.com. 2018. Create A Simple Listview - Android Example. [ONLINE] Available at: https://androidexample.com/Create_A_Simple_Listview_-_Android_Example/index.php?view=article_discription&aid=65. [Accessed 13 May 2018].*

22. *GitHub. 2018. Using an ArrayAdapter with ListView · codepath/android_guides Wiki · GitHub. [ONLINE] Available at: https://github.com/codepath/android_guides/wiki/Using-an-ArrayAdapter-with-ListView. [Accessed 13 May 2018].*

23. *The Developer World Is Yours. 2018. ListView inside ScrollView • The Developer World Is Yours. [ONLINE] Available at: http://thedeveloperworldisyours.com/android/listview-inside-scrollview/. [Accessed 13 May 2018].*

10.4 Hamburger Menu

24. *Treehouse Blog. 2018. How to Add a Navigation Drawer in Android - Treehouse Blog . [ONLINE] Available at: http://blog.teamtreehouse.com/add-navigation-drawer-android. [Accessed 13 May 2018].*

10.5 Other

25. *Mobile App Scanner. 2018. Mobile App Scanner. [ONLINE] Available at: https://www.htbridge.com/mobile/. [Accessed 13 May 2018].*

26. *Picasso. 2018. Picasso. [ONLINE] Available at: http://square.github.io/picasso/. [Accessed 13 May 2018].*

27. *Stack Overflow. 2018. Pass a String from one Activity to another Activity in Android - Stack Overflow. [ONLINE] Available at: https://stackoverflow.com/questions/6707900/pass-a-string-from-one-activity-to-another-activity-in-android. [Accessed 13 May 2018].*

28. *Stack Overflow. 2018. How to capitalize the first letter of word in a string using Java? - Stack Overflow. [ONLINE] Available at: https://stackoverflow.com/questions/5725892/how-to-capitalize-the-first-letter-of-word-in-a-string-using-java. [Accessed 13 May 2018].*

29. *Stack Overflow. 2018. java - Compare two String[] arrays and print out the strings which differ - Stack Overflow. [ONLINE] Available at: https://stackoverflow.com/questions/36122735/compare-two-string-arrays-and-print-out-the-strings-which-differ. [Accessed 13 May 2018].*

30. *Stack Overflow. 2018. java - Sorting arraylist in alphabetical order (case insensitive) - Stack Overflow. [ONLINE] Available at: https://stackoverflow.com/questions/5815423/sorting-arraylist-in-alphabetical-order-case-insensitive. [Accessed 13 May 2018].*

31. *Stack Overflow. 2018. How to read json string in java? - Stack Overflow. [ONLINE] Available at: https://stackoverflow.com/questions/35722646/how-to-read-json-string-in-java. [Accessed 13 May 2018].*

32. *Stack Overflow. 2018. Change Image of ImageView programmaticallyâ Android - Stack Overflow. [ONLINE] Available at: https://stackoverflow.com/questions/16906528/change-image-of-imageview-programmatically-android. [Accessed 13 May 2018].*

33. *Stack Overflow. 2018. android - How to change the text on the action bar - Stack Overflow. [ONLINE] Available at: https://stackoverflow.com/questions/3438276/how-to-change-the-text-on-the-action-bar. [Accessed 13 May 2018].*

34. *Stack Overflow. 2018. android - Add back button to action bar - Stack Overflow. [ONLINE] Available at: https://stackoverflow.com/questions/12070744/add-back-button-to-action-bar. [Accessed 13 May 2018].*

35. *Android Practices: Alert Dialog : Dialog with Item List Example in Android. 2018. Android Practices: Alert Dialog : Dialog with Item List Example in Android. [ONLINE] Available at: http://rajeshvijayakumar.blogspot.ie/2013/04/alert-dialog-dialog-with-item-list.html. [Accessed 13 May 2018].*

36. *Stack Overflow. 2018. android - How to make a edittext box in a dialog - Stack Overflow. [ONLINE] Available at: https://stackoverflow.com/questions/18799216/how-to-make-a-edittext-box-in-a-dialog. [Accessed 13 May 2018].*

37. *Atomic Spin. 2018. Android Snackbar Tutorial: Setup, Action Handling, and UI Customization. [ONLINE] Available at: https://spin.atomicobject.com/2017/07/10/android-snackbar-tutorial/. [Accessed 13 May 2018].*

38. *Stack Overflow. 2018. android - How to set the text color of TextView in code? - Stack Overflow. [ONLINE] Available at: https://stackoverflow.com/questions/4602902/how-to-set-the-text-color-of-textview-in-code. [Accessed 13 May 2018].*

39. *Stack Overflow. 2018. java - Replace Space to Hyphen - Stack Overflow. [ONLINE] Available at: https://stackoverflow.com/questions/5262554/replace-space-to-hyphen. [Accessed 13 May 2018].*

40. *Stack Overflow. 2018. android - How to disable action bar permanently - Stack Overflow. [ONLINE] Available at: https://stackoverflow.com/questions/8456835/how-to-disable-action-bar-permanently. [Accessed 13 May 2018].*

41. *Stack Overflow. 2018. Android: Force EditText to remove focus? - Stack Overflow. [ONLINE] Available at: https://stackoverflow.com/questions/5056734/android-force-edittext-to-remove-focus. [Accessed 13 May 2018].*

42. *Stack Overflow. 2018. java - How to send message from Android app through Viber message - Stack Overflow. [ONLINE] Available at: https://stackoverflow.com/questions/19683297/how-to-send-message-from-android-app-through-viber-message. [Accessed 13 May 2018].*

43. *Stack Overflow. 2018. Getting random numbers in Java - Stack Overflow. [ONLINE] Available at: https://stackoverflow.com/questions/5887709/getting-random-numbers-in-java. [Accessed 13 May 2018].*

44. *Stack Overflow. 2018. Android:java.lang.OutOfMemoryError: Failed to allocate a 23970828 byte allocation with 2097152 free bytes and 2MB until OOM - Stack Overflow. [ONLINE] Available at: https://stackoverflow.com/questions/32244851/androidjava-lang-outofmemoryerror-failed-to-allocate-a-23970828-byte-allocatio. [Accessed 13 May 2018].*

45. *Stack Overflow. 2018. android - How to add org.apache.commons.lang3 to AndroidStudio with gradle - Stack Overflow. [ONLINE] Available at: https://stackoverflow.com/questions/32835143/how-to-add-org-apache-commons-lang3-to-androidstudio-with-gradle. [Accessed 13 May 2018].*

46. *Stack Overflow. 2018. java - Binary XML file line #0: Error inflating class ImageView - Stack Overflow. [ONLINE] Available at: https://stackoverflow.com/questions/47526417/binary-xml-file-line-0-error-inflating-class-imageview. [Accessed 13 May 2018].*

47. *Samsung Galaxy Tab 4 10.1 (2015) - Full tablet specifications. 2018. Samsung Galaxy Tab 4 10.1 (2015) - Full tablet specifications. [ONLINE] Available at: https://www.gsmarena.com/samsung_galaxy_tab_4_10_1_(2015)-6980.php. [Accessed 13 May 2018]*

48. *Vodafone Smart ultra 6 - Full phone specifications. 2018. Vodafone Smart ultra 6 - Full phone specifications. [ONLINE] Available at: https://www.gsmarena.com/vodafone_smart_ultra_6-7313.php. [Accessed 13 May 2018].*

49. *Motorola Moto G4 - Full phone specifications. 2018. Motorola Moto G4 - Full phone specifications. [ONLINE] Available at: https://www.gsmarena.com/motorola_moto_g4-8103.php. [Accessed 13 May 2018].*

50. *Samsung Galaxy S6 - Full phone specifications. 2018. Samsung Galaxy S6 - Full phone specifications. [ONLINE] Available at: https://www.gsmarena.com/samsung_galaxy_s6-6849.php. [Accessed 13 May 2018].*

51. *Samsung Galaxy S7 edge - Full phone specifications. 2018. Samsung Galaxy S7 edge - Full phone specifications. [ONLINE] Available at: https://www.gsmarena.com/samsung_galaxy_s7_edge-7945.php. [Accessed 13 May 2018].*

10.6 Python, RFID & Firebase

52. *Pi My Life Up. 2018. How to setup a Raspberry Pi RFID RC522 Chip - Pi My Life Up. [ONLINE] Available at: https://pimylifeup.com/raspberry-pi-rfid-rc522/. [Accessed 13 May 2018].*

53. *PyPI. 2018. python-firebase · PyPI. [ONLINE] Available at: https://pypi.org/project/python-firebase/1.2/. [Accessed 13 May 2018].*

54. *GitHub. 2018. GitHub - thisbejim/Pyrebase: A simple python wrapper for the Firebase API.. [ONLINE] Available at: https://github.com/thisbejim/Pyrebase. [Accessed 13 May 2018].*

55. *Stack Overflow. 2018. Accessing dictionary value by index in python - Stack Overflow. [ONLINE] Available at: https://stackoverflow.com/questions/15114843/accessing-dictionary-value-by-index-in-python. [Accessed 13 May 2018].*

56. *YouTube. 2018. Python firebase adding data using POST - YouTube. [ONLINE] Available at: https://www.youtube.com/watch?v=ZAOTe7MhNSo. [Accessed 13 May 2018].*

57. *Python For Beginners. 2018. How to use Split in Python. [ONLINE] Available at: http://www.pythonforbeginners.com/dictionary/python-split. [Accessed 13 May 2018].*

PROJECT PROPOSAL

BSHC4IOT

Leon Mulvaney
14445618
BSc (Hons) in Computing - IoT
X14445618@student.ncirl.ie
October 2017

11 Appendix A - Project Proposal

11.1 Objectives

The Internet of Things (IoT) is an integrated network of smart devices which interchange information between each other. These devices are embedded within already developed platforms. Internet of Things enabled devices do not require human interaction and instead, can perform actions based on their surroundings. IoT has grown exponentially within recent years – this is due to many factors including, availability of networking, cost of technology alongside a massive growth within the mobile market.

The primary objective of my project is to develop an “Intelligent Food Network”/” Smart Fridge” using RFID technology alongside an Interactive Mobile Application to track what foods are entering and leaving a user’s fridge. My idea will build upon and utilize current IoT principles.
A broken-down analysis of my objectives is featured below;

Scanning using Raspberry-Pi & RFID
The Raspberry-Pi - a single board computer with embedded microcontroller capabilities will be a crucial component throughout development. My initial objective is to connect the Raspberry-Pi to an RFID reader/writer via a breadboard in order to track what foods are entering and leaving the Fridge. RFID tags containing embedded information will be attached to these food items – the RFID reader will then record the addition or removal of an item from the fridge. The Raspberry-Pi will act as the mediator - processing all data associated with each use case, then traversing the information to a database.

Android Application
Once I am pleased with the reading and writing using RFID, I will aim to begin development of my application to go alongside my Intelligent Food Network. Users will be able to use this application to interact with their fridge. The application will be developed throughout the course of the project, starting with basic GUI, progressing on to become a fully-functioning application - including a real-time database, push notifications, recipe recommender and much more.

Use of Firebase (Backend Functionality)
My next objective is to implement Firebase – a real time database and a very powerful platform. Firebase will act as an intermediary between my Pi RFID reader and my Android application - allowing all information to be traversed from the reader up to the cloud database and then back into the application thus, creating my “network". I believe Firebase will work well with my project and its real-time capabilities will prove very beneficial. The Firebase platform also integrates well with Android – this will enable me to send push notifications to my Android Application. I aim to store all food details in a Firebase database, which users may view through the app. Information will be sent from the RFID reader to the Pi then to the application.

Use of Current Technologies & Trends

Throughout the development process one of my major objectives is to use current technologies and trends. These include;

- The use of various programming languages i.e. Java & Python etc.
- The use of Mobile Technologies (Android Studios, Raspbian, Firebase)
- The use of Current Trends (IoT, Raspberry Pi, Smart Living etc.)

11.2 Background

My idea to create a Smart Food Network stems from my interest in electronics and technology. Since the recent emergence of IoT I have always been interested in the idea of a "Smart Home". I regularly receive emails featuring new developments in technology - with many of these including smart gadgets or products which improve home life. I have always wanted to create my own "smart thing" but never really knew what to create.

One of the biggest developments I have noticed within the IoT world is the addition of Smart Heating systems. (Tado is a good example). These smart heating systems have taken the industry by storm. 10 years ago, the idea of controlling your heating from a mobile phone would have been "bizarre" – why would you want to do that? But within the past decade things have drastically changed. People are starting to adopt these new advancements in technology and I believe there will be more and more smart gadgets and embedded systems in domestic homes within the next few years.

One of the major changes within today's world is the addition of Mobile phones. Everyone has a Mobile Phone, period. There is also an abundant array of applications to suit everyone's needs. One of the most popular type of applications on the app stores I noticed (after games) were productivity and leisure. I also noticed that food recipe apps were plentiful. I've downloaded some of them myself in the past, with the enthusiastic idea of making something extraordinary – only to find that I didn't have the ingredients nor the cooking skills.

Another persistent issue I noticed within my home was that we were always low on bread, milk, butter and some other basic food items. I would often return home after a day at work longing for a cup of tea - only to find there was no milk. What if I could check on my way home? or could check my fridge when no one else is home? Within my home every day the same question is tossed about… "What's for dinner?" there is always a debate and then when everyone agrees, it turns out that the correct ingredients are not there.

Then the idea sparked – my interest in electronics and the smart home could come together. I realised that the whole food network within a home could become "Intelligent". The idea of being able to see what foods are present in the home seemed really appealing. Also, a recipe recommendation based on what food items were present seemed like a very good idea.

I thought about the idea more and more and eventually came up with a plan, I could implement this idea into my final year project. Below are some of the most desirable characteristics which will glue my idea together into a finalized product.

Android Application

To tie the whole idea together, users will need a stable platform to use. An Android Application will work as the mediator between the user and their food items. I chose Android as it is one of the most popular mobile operating systems. It also supports a wide range of functionality which I was looking for. Android works seamlessly with Firebase- a fundamental inclusion within my project.
An Android Application will be developed. It will include some of these features;

- Users will be able to see what's present in their fridge/press
- Check what recipes they can use for meals
- Receive push notifications on their mobile regarding meal choices
- Check expiry dates and frequently purchased items
- Access their digital shopping list based off frequently used food items

Raspberry Pi & RFID

The Raspberry Pi will act as the glue between the Kitchen and the database. An RFID reader/ sensor will be connected to the Pi and any food items entering and leaving will be recognised via RFID stickers. This information will then be recorded and posted to the database.
I chose the Raspberry Pi over any other devices (i.e. Lattepanda, Arduino, Asus Tinkerboard) due to the wide range of support and compatible sensors. The newest model of Pi also features built in Wi-Fi & Bluetooth which will allow it to function wirelessly via SSH. The Raspberry Pi is also a very efficient device – only sipping on a small amount of power. Python - a programming language which is relatively easy to pick up is the main language used on the Pi. It works well with JSON and Firebase – the database I have decided to use. The readings from sensor data will also take advantage of current IoT technologies.

Firebase

Firebase is a very powerful platform developed by Google. It features a real-time database, provides push notification services alongside much more. Firebase will effectively undertake all the "heavy lifting" within my project. Data will be traversed from the Pi to my Firebase server. The database will dynamically update and push notification will be provided based on pre-defined rules. Firebase works well with Android and I will implement various API's to traverse data from Python up to my cloud storage solution.

The Intelligent Food Network – Sample Use Case
Below is a sample, high level Use Case. There will be much more use cases which I will discuss in a later section/ document.

- 9am - User Checks what's in Fridge via Android Application (Firebase RTDB accessed)
- 10am - User Purchases Goods in Food-store
- 11am – User returns home and places new items into Fridge (Making contact w/ RFID reader – Firebase Updated)
- 2pm – User receives push notification sent from Firebase (Recipe Recommendation for dinner at 5pm)
- 4pm – User prepares dinner (Removing recipe contents from Fridge – Firebase update applied)

For the idea to work, users will simply have to fit the small device (Pi & RFID Scanner) then download the application.

11.3 Technical Approach

Throughout my project, I will follow a step by step approach – implement specific procedures before continuing onto another section;

Conducting sufficient research

This will ensure I approach my project in the most effective and efficient manner. I will use a wide variety of reputable sources for any queries or issues which may arise. These will include the National College of Ireland library, The Summon platform, books, and videos alongside reputable websites i.e. Stack overflow, W3Schools and Tutorials Point. Any resources I do find helpful will be recorded and referenced via the Harvard Referencing Methodology. Research is a vital component and by doing so, I can develop my project to a very high, industry standard.

Requirements Capture

Before progressing, I will ensure I am fully confident that I understand all the proposed requirements. This will reduce the chances of confusion and enable me to successfully complete my set goals. I will understand each section, what it will be composed of and how it will function. Requirements capture will prove a vital concept prior to development as I will understand the system features and requirements – these will include both Functional & Non-Functional requirements. I also aim to attain feedback from potential users by conducting surveys – this will improve any potential features.

Class, UML and Architecture Diagrams

I will make use of Microsoft Visio alongside Draw.io to create my required diagrams. I aim to create these prior to development as they will assist greatly. These will be featured within my Requirements Specification document. These diagrams will consist of Class & System Architecture etc.

Mock-ups & Prototypes

Before any development begins, I will create some high-level mock-ups then progress to develop some basic prototypes. I will create a variety of prototypes to test and select the best one before progressing. Mock-Ups will be created by first pen and paper, then following onto using the Balsamiq Mock-ups. Early prototypes of the proposed application will be created using Android Studio – I will develop a “test” application here to ensure everything works to a sufficient standard.

Testing

Once a section is completed, I will vigorously test it to ensure it is error free. I will test my application on many devices to ensure all features function correctly with the required OS version. The Raspberry-Pi will also be tested on various network and configurations to ensure an error free project.

I aim to undertake my project to an industry standard - implementing a logical, step-by-step approach throughout development. I will also break my project into smaller tasks to increase manageability.

11.4 Special Resources Required

My project will require some mandatory resources to obtain a successful outcome. Outlined below are some of the most crucial and assistive resources I will use;

Raspberry-Pi (Hardware)

The raspberry-Pi is probably the single most important pieces of equipment I will use throughout the development process. It is effectively the "bread and butter" of my whole project. Without it, my project would not be possible. One of the most beneficial aspects of the Pi is its inclusion of 40 GPIO pins which can be used with various sensors – in my case a breadboard and RFID reader.

Firebase (Software)

Firebase will prove to be a valuable asset throughout development process.

Android Studio (Software)

Android Studio is an IDE designed by Google, specifically targeted at Android Development. It features integrated support for Firebase, Github and contains a wide array of debugging features.

Python Books

1. The Raspberry Pi User Guide (Eben Upton and Gareth Halfacree)
2. Learning Python with Raspberry Pi (Alex Bradbury and Ben Everarad)

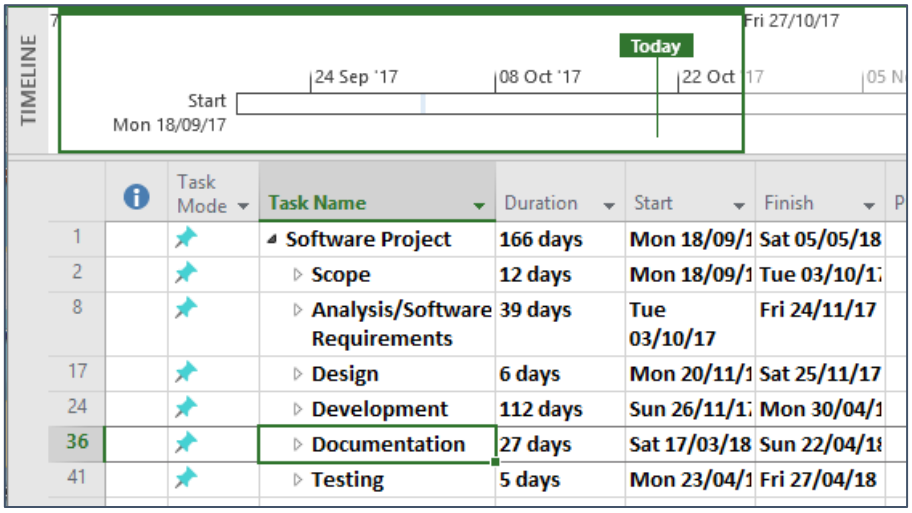
Throughout the summer and during the internship I was learning about the Raspberry Pi. During this period, I purchased 2 Raspberry Pi books. I have since read book 1, although have not finished book 2. I believe these books will prove to be beneficial additional resources.

11.5 Project Plan

Below is my Project Plan featuring a Gantt Chart. At these early stages, the dates below are still preliminary - although I aim to follow this plan as best I can.

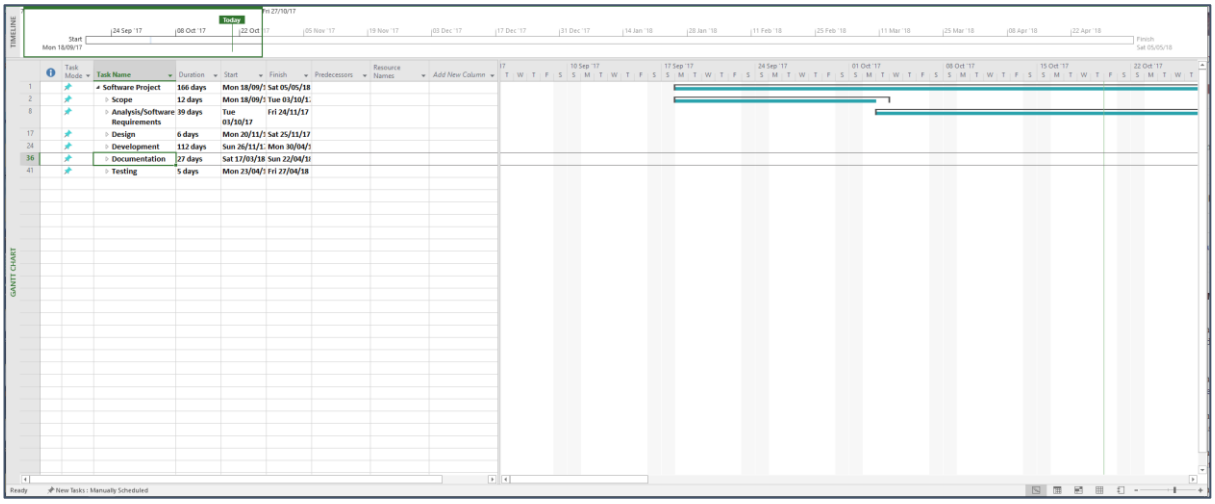
I have broken the project plan down into 6 main categories;

- Scope
- Software Requirements
- Design
- Development
- Documentation
- Testing



	Task Mode	Task Name	Duration	Start	Finish
1		Software Project	166 days	Mon 18/09/1	Sat 05/05/18
2		Scope	12 days	Mon 18/09/1	Tue 03/10/1
8		Analysis/Software Requirements	39 days	Tue 03/10/17	Fri 24/11/17
17		Design	6 days	Mon 20/11/1	Sat 25/11/17
24		Development	112 days	Sun 26/11/1	Mon 30/04/1
36		Documentation	27 days	Sat 17/03/18	Sun 22/04/18
41		Testing	5 days	Mon 23/04/1	Fri 27/04/18

Project Plan



Project Plan with Gantt Chart

		Scope	**12 days**	**Mon 18/09/1**	**Tue 03/10/1**		
		Develop Project Idea	9 days	Mon 18/09/17	Thu 28/09/17		
		Prepare for pitc	2 days	Fri 29/09/17	Sat 30/09/17		
		Finalize idea for pitch	2 days	Fri 29/09/17	Sat 30/09/17		
		Project Pitch	1 day	Mon 02/10/1	Mon 02/10/1		
		Project Pitch Complete (Milestone)	1 day	Mon 02/10/17	Mon 02/10/17		

Scope

		Analysis/Software Requirements	**39 days**	**Tue 03/10/17**	**Fri 24/11/17**		
		Further Research	10 days	Tue 03/10/17	Sun 15/10/17		
		Project Proposal	10 days	Sun 15/10/17	Thu 26/10/17		
		Project Proposal Upload (Milestone)	1 day	Fri 27/10/17	Fri 27/10/17		
		Requirements Gathering	7 days	Sat 28/10/17	Sat 04/11/17		
		Early Feasability Testing	7 days	Sat 28/10/17	Sat 04/11/17		
		Requirements Specification Write-up	19 days	Sun 29/10/17	Wed 22/11/17		
		Supervisor Feedback on Req Spec	1 day	Thu 23/11/17	Thu 23/11/17		
		Requirements Spec Upload (Milestone)	1 day	Fri 24/11/17	Fri 24/11/17		

Analysis/ Software Requirements

		Design	**6 days**	**Mon 20/11/1**	**Sat 25/11/17**		
		Mockups	1 day	**Mon 20/11/1**	**Mon 20/11/1**		
		Develop Layouts for App	1 day	Tue 21/11/17	Tue 21/11/17		
		Further testing Using RFID & Pi	1 day	Wed 22/11/17	Wed 22/11/17		
		Supervisor Meeting for next steps	1 day	Thu 23/11/17	Thu 23/11/17		
		Implement necessary changes	2 days	Fri 24/11/17	Sat 25/11/17		
		Design Complete (Milestone)	1 day	Sat 25/11/17	Sat 25/11/17		

Design

		Development	**112 days**	**Sun 26/11/17**	**Mon 30/04/1**		
		Use Requirements Spec & Proposal as	112 days	**Sun 26/11/17**	**Mon 30/04/18**		
		Early Prototype Development	7 days	**Sun 26/11/17**	Sat 02/12/17		
		Mid-Point Presentation (Milestone)	1 day	Mon 04/12/17	Mon 04/12/17		
		Develop Pi 8 RFID Features	20 days	Tue 05/12/17	Sun 31/12/17		
		Develop Mobile Application	23 days	Mon 01/01/18	Wed 31/01/18		
		Implement Back-End Functionality	21 days	Wed 31/01/18	Wed 28/02/18		
		Testing	6 days	Wed 28/02/1	Wed 07/03/1		
		Advisories from Supervisor	1 day	Thu 01/03/18	Thu 01/03/18		
		Alterations based on advisories	6 days	Fri 02/03/18	Fri 09/03/18		
		Further Testing	6 days	Sun 11/03/18	Fri 16/03/18		
		Development Complete (Milestone)	1 day	Fri 16/03/18	Fri 16/03/18		

Development

		Documentation	**27 days**	**Sat 17/03/18**	**Sun 22/04/18**		
		Project Report	27 days	**Sat 17/03/18**	**Sun 22/04/18**		
		Make alterations to older documentation if required	7 days	Sat 17/03/18	Sat 24/03/18		
		Guide Creation	9 days	Wed 21/03/1	Sun 01/04/18		
		Documentation Complete (Milestone)	1 day	Sun 22/04/18	Sun 22/04/18		

Documentation

		Testing	**5 days**	**Mon 23/04/1**	**Fri 27/04/18**		
		Test application	5 days	**Mon 23/04/18**	**Fri 27/04/18**		
		Testing Complete (Milestone)	1 day	Fri 27/04/18	Fri 27/04/18		

Testing

This Project Plan will assist me throughout the development process. I will follow a step-by-step approach throughout.

11.6 Technical Details

Throughout the development of my project, I will implement a professional – industry standard approach. Below is a listing of the technical details with regards to my project;

Hardware

The primary hardware components that will be featured within my project are:

- Raspberry-Pi
- Breadboard
- RFID Reader & Writer
- Jumper Cables
- RFID/NFC Stickers and Tags
- Multiple Android Devices (For testing – Each Featuring different OS Versions)

Software

I will be using a variety of software throughout the development lifecycle. Software components will include:

- Microsoft Windows – My Primary Development OS
- Raspbian – Pi OS
- Android Studio – For the creation of my Android Application
- Google Firebase Platform – Database, Backend & Push Notifications etc.
- IDLE – For Python Coding & Scripts
- Command Line Variants - Windows CMD & Linux Terminal
- Putty – Providing SSH access to the Pi (Via Command Line)
- VNC Viewer – Remote Access to Pi (Via GUI)

Programming Languages

For my project, I will be utilizing many different types of hardware – this will require me to use multiple platforms for development, especially a varied set of programming languages. These will include:

- Python – The primary language of the Pi
- Java – To develop the functionality of my Android Application
- XML – To develop the design, layout and GUI of my Android Application
- JSON – A powerful platform providing inter-language traversal
- Various Libraries – For use within my project e.g. Python GPIO Import/ Android Intent etc.

API's

I aim to include various API's throughout development:

- Firebase API
- Pi to Firebase API – Transmission of data from Pi to Firebase
- Food API – To grab types of foods from the current online databases
- Misc. API's – I will also expect to be incorporating additional API's not yet on this list as they are still to be confirmed.

11.7 Evaluation

The overall aim of my project is to become a one for all solution to a common household problem. It will build upon current IoT technologies under the category of “Smart Home”. I will incorporate current technology and design trends to ensure my project is successful and is of an Industry Standard. Once each milestone is completed, I will move onto the next.

Software testing is a vital measure which much be implemented to produce a successful outcome. Throughout the course of development, I will continually test my project to ensure all sections are functioning adequately. Once I approach the final stages of development, I will implement a rigorous testing routine to ensure each section error-free. I will use Android Studio to test my project on a physical device – making use of the proprietary debugger. I can also use some automated testing software for Android i.e. Junit and Monkey. These pieces of software will enable me to mimic extensive user interaction with my application - uncovering any potential faults or weaknesses.

Alongside testing via Software, I will also test the application myself. Finally, when I believe my project cannot be tested any further, I will begin testing with potential end-users. This will strain out any potential faults whilst highlighting any GUI or usability enhancements I could improve upon.

MONTHLY JOURNAL

BSHC4IOT

Leon Mulvaney
14445618
BSc (Hons) in Computing - IoT
X14445618@student.ncirl.ie
2017-2018

12 Appendix B – Monthly Journal

12.1 Introduction

My name is Leon Mulvaney and I'm currently in my final year of the BSHC programme. I came to NCI back in 2014 immediately after finishing my leaving certificate. I had always had a great interest in computers, technology and electronics in general and wanted to study in these areas. When first year began I was very confused to be honest. Yes, I had been acquainted enough with general computing i.e. desktop applications, operating systems, basic usage etc - but had never been exposed to programming. The whole concept of syntax, semi-colons and variables were indeed, quite a mystery to me.

Most modules featured in Semester 1 of First year seemed to ease us in gradually. After all, we were all in the same boat (well apart from the guys that had previously done PLC courses - their concept of programming was as expected, a little more developed). Some of the modules were Introduction to Computers, HTML and Web Design and PPD which were all fine.

Then came Introduction to Programming.

This module really was the bane of my very first semester at NCI. We started with Java and for the first few weeks I really couldn't get my head around it. The whole idea of semi-colons, variables, classes and constructors in my mind were, effectively from Mars. Going into each class was like getting lost and lost, over and over again. I remember spending absolutely ages trying to figure out how to get TextPad to compile with the Java JDK. When you eventually did manage to get some form of code running, whom knew that something (at the time) as small as a bracket or semi-colon could turn your perfectly functioning piece of code into an absolute wreck. In an instant, you went from zero errors to 78. Ohhh, the frustration...

Eventually, after watching Frances Sheridan's Programming videos over and over (Which were like a godsend to most of us in fairness), I slowly began to come to grips with it all. After a couple of weeks, I started to enjoy programming. If there was only one thing everyone only took from the module, it was this;

Eggs myEggs = new Eggs (Most of the lads in the class will know laugh at the term - kind of like some sort of inside joke if you will).

I digress, I tend to go on tangents from time to time.

In the past few years, I've learned a lot at NCI. The past 3 years have flown in and I've developed a variety of skills which I hope to apply to my final year project. I've decided to specialize in Internet of Things as I'm very interested in the topic and believe it is the way towards the future. The internet has only been around for such a relatively short period of time, yet it is now part of our everyday lives. Computers, smartphones, smart devices – we now live in a digital world which has grown exponentially.

This document will feature my personal reflections as I undertake my final project.

12.2 September 2017

18th September

After spending the past 9 months working a 40-hour week, it was nice to return to NCI. We had a short lecture with Eammon describing the ins and outs of the year ahead. We were informed of the work that was needed and how to manage ourselves throughout the year.

20th September

Today we had our first IOT class with Dominic Carr. He introduced us to the module and what it consisted of. As my specialization is IOT, I wanted my final project to be based around that area. At this stage, I had a couple of ideas although I had still not solidified my decision. Today was a nice short day finishing at 1pm. It was quite a contrast to usually starting at 8 and finishing at 4. I got home early at about 2pm and done some more research.

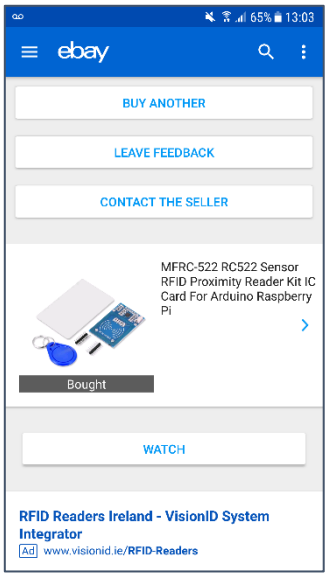
21st September

Today we had some more module introductions. We had another class with Dominic today. He requested anyone with IOT project ideas please see him to discuss. At the end of class I met with him in his office. I explained my idea of the "Ultimate Smart Home". This initial idea would bring together many concepts within a typical domestic dwelling. My aim was to feature 4 main aspects:

- Security (PI Camera/PIR sensor with email sent on intruder alert)
- Lifestyle (A smart fridge using RFID)
- Pets (Pet food dispenser using relays, actuators etc)
- Health (Heating controlled by RPI using relay and temperature sensor)

After speaking with Dominic, he suggested I focus in on the Smart Fridge as I would have a good bit to work with.

I returned home and conducted some more research on the topic.



25th September

Today I conducted some more research on my topic of choice. I ordered some equipment from eBay for testing (RFID Reader/Writer and RFID/NFC Stickers). These should arrive within the next few days. I discovered that more powerful RFID readers/writers allow for increased range. This is something that is not exactly necessary now, but it would be ideal.

27th September

Today we got our Raspberry Pi's in class. I had been learning and using my own Pi over the past couple of months, so I knew I was able to set it up with relative ease, albeit a little differently via Dominic's slides. I went home and set the new Pi up and could access it over my local network using SSH. I also got an old mobile phone with the touch screen malfunctioning and could remote access into the desktop via VNC Viewer. I left my old Pi set up but swapped the new SD card I had set up in an alternative manner. This meant I didn't have to go reconnecting wires, sensors and resistors to the breadboard.

30th September

Today my NFC stickers arrived in the post. I am now just waiting for the RFID reader. After conducting some more research, I decided that there were some very desirable features I wanted embedded within my project:

SQLite Database

- Used to hold all my data, i.e. food type, expiry date contents
- Works well with lightweight devices and Android
- I would ideally like this to be hosted on another Pi (Such as NAS Server)

Android Application

- Push notifications recommending recipes based on fridge/press contents
- Option to View what's in the fridge
- Recipe recommender
- Expiry Dates
- List of what needs to be bought when shopping

I have still to prepare my pitch on Monday which I hope will go well.

12.3 October 2017

2nd October

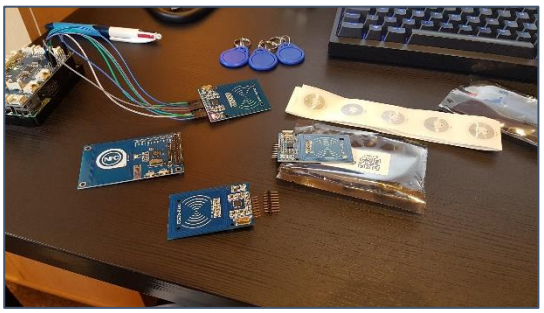
Today I had my pitch at 16.30. I arrived a little early at 15.30 so I done what any student would do… got a burrito… – After all I didn't want to go in nervous and as well as having an empty stomach. When I went in I explained my idea and went on a tangent as usual explaining all the concepts I wanted to implement. I thought it went quite well, being honest and Dominic was there too so that helped as he already had an idea as to what the overall project was about. I was in talking for much less time than I had expected. All went well and I was very happy overall with the way the viva went. My idea got approved. Now I'm just waiting for my RFID Reader/Writer to arrive!

The rest of this Month went well, albeit there was quite a workload coming from Modules such as IOT, Mobile App Development and especially Artificial Intelligence. Here is an overview of the month of October:

Achievements

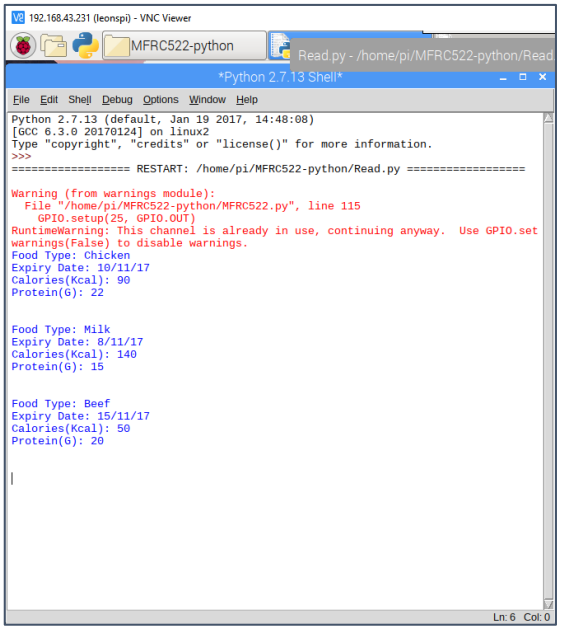
To be fair, I was quite busy this month – there seemed to be a constant influx of CA's to be completed although I did manage to get some major features working regarding my Food Network.

I ordered a load more equipment for my project, 4 readers (Sort of an insurance policy in case some of them didn't work), 20 more RFID tags and some more RFID stickers. I also ordered a soldering Iron as the readers needed to be soldered. Once this arrived, I began to solder the readers. Unfortunately, the "Brand New" soldering iron malfunctioned halfway through the 3rd reader. I had to just make use of the first 2.



I installed the required libraries on my raspberry pi and began to prepare a script for reading and writing data through RFID. I plugged the first reader in – no result it did not work. Then thankfully the second one worked immediately and I could now read and write data to the tags. For testing, I added 4 variables Food Type, Expiry Date, Calories and Protein content – these would give me a good basis to work off.

Next, I began to work on my mobile application. I set up Firebase and connected it to Android Studio. Firebase took a little effort to set up, but I got it working in the end.



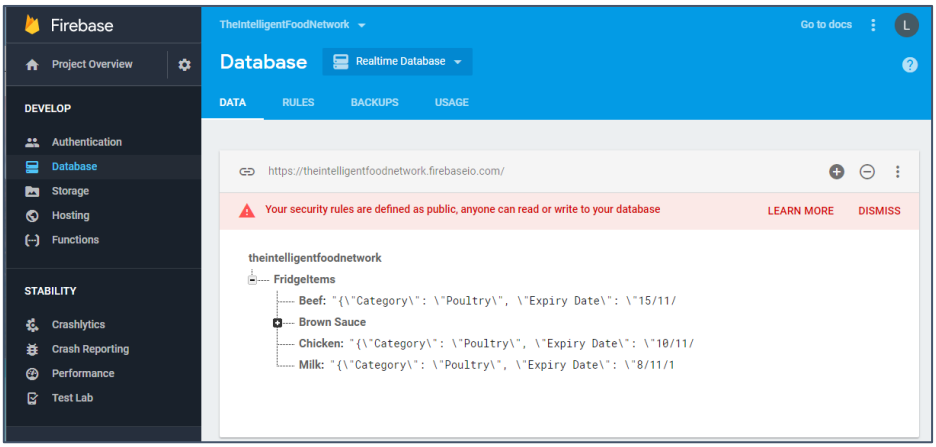
I added a Firebase connection to my Python "Read" script. Then I created a function to post the values read in from the Tag up to Firebase. I got his working, although it is not fully structured in "Parent-Child" format. I will need to test this in the coming weeks.

A brief overview of my achievements for October:

- RFID Reader/Writer soldered and functioning properly
- RFID Tags reading correctly (4 Variables held on each)
- Creation of Mobile application – Basic GUI
- Link Python and Android Application to Firebase
- Python Script to Read/Write
- Python Script to Read & Post Values to Firebase

Reflection

Although I didn't get as much time this month to work on my project, I think that I made some good progress. One of my primary goals was to Read data and post to Firebase which I achieved this month. There were a few setbacks, especially the soldering iron failure – but that's all part and parcel I suppose. Also, only my Key-Fob and Credit Card type RFID Tags would properly Read/Write. Whilst I did prove my Proof of Concept, I would ideally like the stickers I purchased to work too!



Intended Changes

The next major part of the project is preparing for the mid-point presentation. I have a couple of ideas which I think will really demonstrate my idea (in a humorous way too). The next steps are to better structure the data posted from Python into my Firebase Database. For the Prototype, I also would like to have the mobile application reading Firebase.

Supervisor Meetings

Thursday 18th October
Thursday 26th October

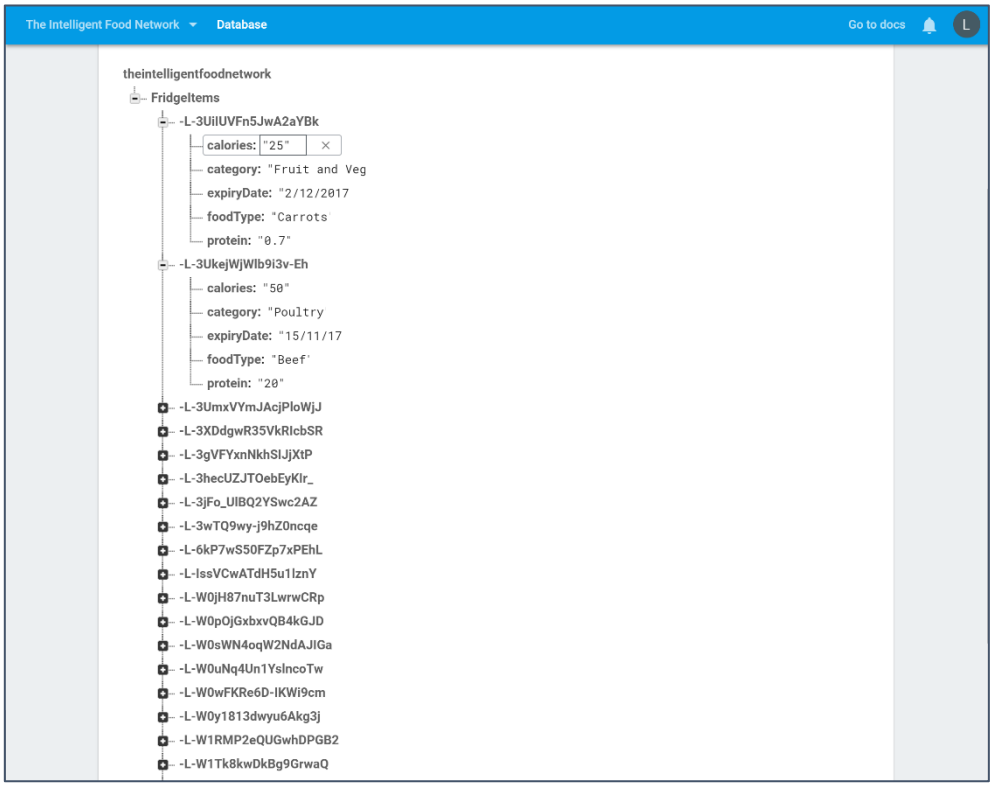
I met with Glen Ward Multiple times this month – we have agreed that Thursdays at 12pm is an appropriate time. This month we chatted about further developments. One of the aspects of my application is to provide recipe recommendations based on food items within the home. Glen suggested I look at some Recipe API's as this would prove a good starting point. He also provided me with some links to reliable API webpages.

12.4 November 2017

Achievements

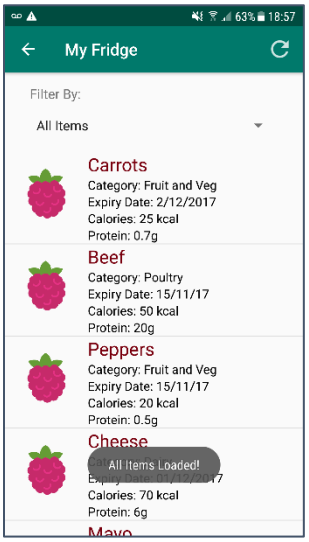
This Month was once again quite busy. Many CA's and project submissions were due week after week. This Month was primarily focused on my project Requirements Specification and Technical Report document. The Requirements took most of the time, it took a few weeks to develop this document as it featured, surveys, mock-ups, class diagrams and a lot of typing to be honest.

I also worked on my prototype. The primary aims this month were to get the Firebase reading from Python in a structured format i.e. child key:value pairs with the unique id as the parent for each data set. This may be easier to understand using the below screenshot of my Firebase database. I figured out the problem why the data was not uploading properly was that I was using the PUT method where I should have been using POST. Of course, this is all part of the learning curve.

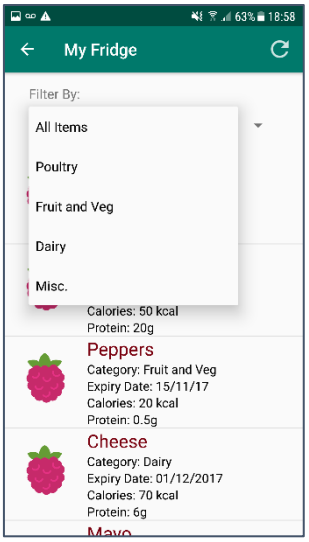


Firebase Database Organisation

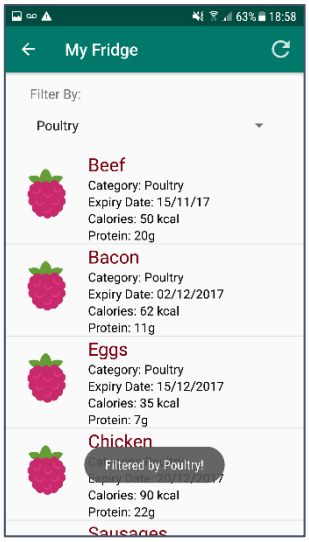
The next section was to get the data pulling into the Android Application. I needed to learn how to pull the data from Firebase into Android as I had never done it before. After searching through various tutorials, I finally was able to pull a simple line item called “Chicken” from my database. Whilst this didn’t seem like much, it was a great stepping stone within the prototype – I finally was able to Post data to Firebase, then pull it back down into my Android Application, thus creating my “Network”.



All Firebase Contents



Filter Menu



Contents Filtered by Poultry

I developed this section further, incorporating a list view which was able to hold all the data grabbed from Firebase. Once the custom list view was complete, I added a filter menu to the application. This also required me to change my Firebase database to include an extra key:value pair – “Food Category”.

This was as far as I had got before the mid-point presentation, although no other sections worked I thought this prototype would provide a good Proof of Concept regarding my Intelligent Food Network.

I then began to prepare for the Mid-Point Presentation. I had decided earlier on in the Month to use a mini-fridge alongside children’s plastic food. I thought these items would be humorous and allow the project idea to be better demonstrated during the mid-point presentation. I wrote all the required information to my RFID tags before taping them to the plastic foods.

Mini Fridge and Kids Play Food from Argos

Finally, the last thing to prepare before the presentation were the presentation slides. I created my presentation through Prezi as it is easy to use and looks very well if used correctly.

The last thing was the Mid-Point presentation. Although technically part of December (December 4th to be exact I am going to include it within November as all my work associated with the project in November was in preparation for the Mid-Point Presentation).

On the day I brought all my equipment which included my Raspberry Pi, RFID Controller, RFID Tags, Android Mobile Device, Mini Fridge & Plastic Foods. I went in and talked about my idea. I was nervous to be honest as I hate presentations, although after a few minutes what was a presentation turned into a friendly conversation about the idea. I referred to my slides to keep structure throughout presentation, but once I started talking about the idea I felt I didn't need them. I had worked hard in fairness on the idea, so I knew exactly what the ins and outs were. The presentation ended before I knew it. I was quite happy with the constructive criticism and feedback I received from my Supervisor Glen and the second examiner Dermott.

Brief Overview of Achievements for November:

- Firebase Data Structured
- New Variable – Food Category Added to Database Items
- App GUI Enhancements
- App now pulls contents from Firebase (This data automatically updates and refreshes)
- App now allows Filtering
- Requirements Specification/Technical Report
- Mid-Point Presentation Complete

Reflection
This month was a very busy month with a lot of time focused on developing the Requirements Specification document. When I started the doc, I didn't realise how much work was required. Unfortunately, this meant that I could not focus as much on my prototype - although I still did manage to achieve my aims for this month –with the primary aim being able to POST data to Firebase in a structured format, then pull it back down into the android application. I felt that the Mid-Point presentation went well, and I was happy with my overall progress considering there were so many other deliverables too this month. I appreciated the feedback that I received throughout the Mid-Point as it allowed me to further concise my overall idea.

Intended Changes
The next big step is to develop the system even further using the feedback I was given within the Mid-Point. My primary aims are to develop the primary sections of the application which are: Recipes, Shopping Lists and User Account sections. I look forward to further development of the application as usually find the documentation the most tedious part of any project.

Supervisor Meetings
Thursday 9th November
Thursday 23rd November
Friday 1st December

I met with my supervisor Glen multiple times throughout this Month. Meetings usually take place each Thursday at 12pm. I find these meetings very helpful as I am provided with great feedback on the project progress and what routes to follow. This month was quite a busy one and I had many CA's due. Week ending the 19th was especially busy and I didn't get much time to work on software project - so I asked Glen if I could push the meeting scheduled on the 16th to the following week. He had no problem with this and it allowed me to make some more substantial progress to present the following week. This month's meetings were primarily based around preparation for the Mid-Point presentation, the final being before the 1st December.

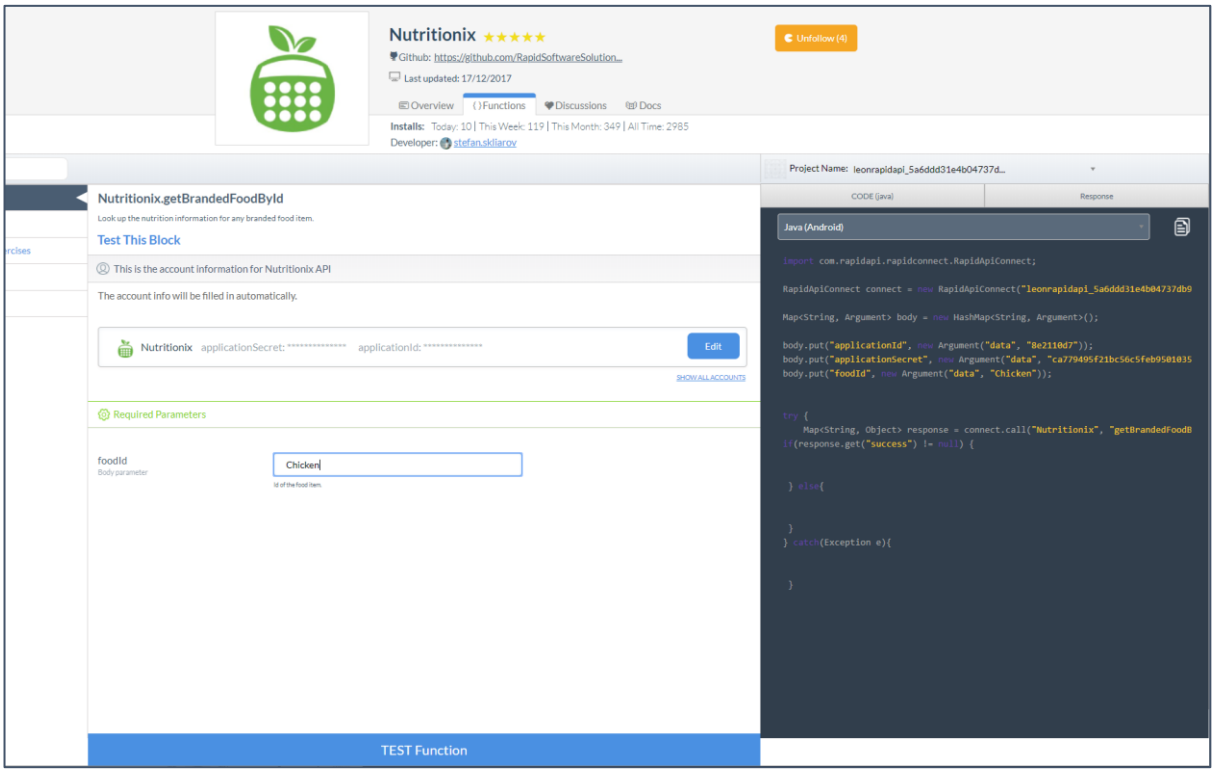
12.5 December/January 2017-2018

Achievements

These two months were quite hard in fairness. After the Mid-Point presentation at the start of December I had many other projects to complete – primarily IOT, Android and API. Over the Christmas I primarily focused on studying for the Christmas Examinations. Working part time over the Christmas took some time away from the project also, although upon returning back to college in January I started to work on some of the fundamental aspects of my "Intelligent Food Network".

I had never used API's before asides from the Web Services & API module in Semester 1. I researched for a week how to incorporate the API into my application. Finally, I found a very useful website known as RapidAPI. RapidAPI is essentially a hub for developers to upload and promote their API's. The platform features a very useful section which generates the requests for many languages. This proved to be extremely useful as I wasn't 100% sure how to make requests to every API. From here, I could also test the requests with the result displaying in a IDE-looking placeholder.

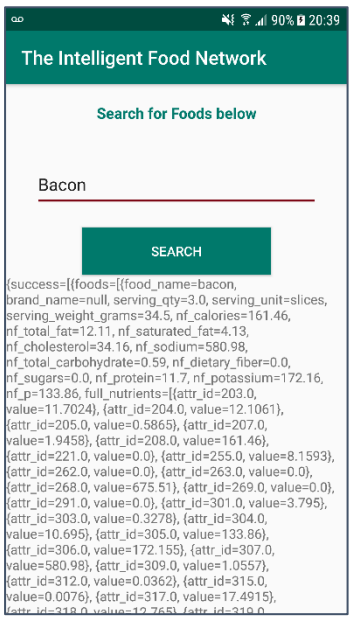
Although Spoonacular is the primary API that I specified in the Technical Report, I found another useful one called "Nutritionix". Nutritionix allowed me to grab food nutritional values from a String search. This perfectly suited my application, it was also free which was a bonus. (I contacted Spoonacular about a student plan and they told me the charge was 10$ per month, so I wanted to make sure my application could handle API's before I subscribed to the plan).



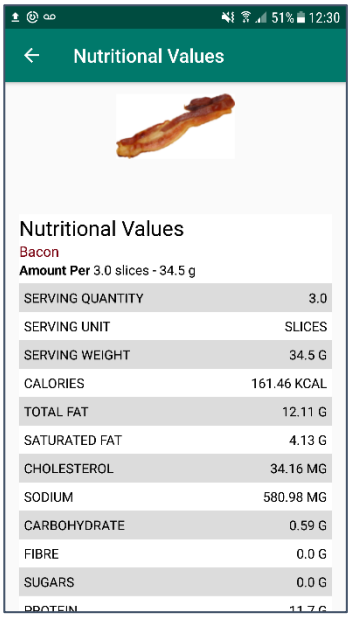
The Section on RapidAPI which generates the requests calls and displays the response

Before I could make use of this API, I had to create an account on their website – Nutritionix.com, then request an API key. I then focused on the Android application. One of the primary issues with the application was running network threads. By default, Android prevents the system from running networking tasks on the main thread. To combat this, I made use of ASyncTask. This was probably the most time-consuming part. I eventually got ASyncTask working correctly, this will prove useful throughout further development.

I created the Android views and layouts to further test the API. A simple search view with button was used to make requests to the API. Initially, the API only returned a Hashmap with lots of content- a lot which I didn't need. I filtered the content, placed it into an Object called FoodSearchItem and then parsed this object to a custom Adapter (Using Android ListView). The Application now allows users to search for food nutrients, with all values pulling from the Nutritionix API. I also used a new library called Picasa – this was really useful as it allowed me to pull an image into an imageview from a URL.



The Response before filtering



The Final Design using OOP and Custom Listviews

Reflection

Overall, I was quite happy with the progress I made between December and January. I was working about two weeks over the Christmas break before and after exams too, so this obviously took a little time away from the project. In fairness, Semester 1 was very though, and I remember in week 7 writing down all the deliverables I had due over the next 5 weeks. Now its February and only 10 weeks to go. I think that I've learned the most this year and my programming competency has gotten better overall from doing so many different projects. I especially liked the Mobile Application Development and IOT modules. I've also thoroughly enjoyed working on the Software Project as I can work on it at my own pace. The fact I'm not rushing to meet deadlines means I can actually try and perfect it even more. I was happy this month especially getting the API working.

Intended Changes

Over the next month I hope to get a good amount of primary functionality completed. The Main sections are the Shopping Lists and the Recipe Generator. I am aiming to take about a week for each section – this is an optimistic estimate, but I am going to try and achieve it. Furthermore, I will also try build upon the Python Scripts I have created to record food coming in and out of the home. At present, food is only logged when it enters.

Supervisor Meetings

Thursday 7th December
Thursday 14th December
Thursday 1st February

After the Mid-Point I met with Glen a couple of times before the Christmas break. We discussed where the project was going, how I got on in the mid-point and what I should aim for next. I try to meet Glen each week as it's great to get feedback, I also really enjoy the meetings. Glen has advised that I try and get all the major functionality working by the end of the Mid-Term two-week break. This will allow me to focus on the Document and any tidying up of the system. This is my new aim and I hope to work hard on the project each week to achieve this aim.

12.6 February 2018

Overview

This month did not have much deliverables apart from IoT, so I decided to dedicate a lot of time to Software Project. I love working on the GUI sections of applications, so a lot of time was spent at night listening to music and designing the UI as best I could. Personally, I felt that up until 4th year the GUI of almost every project has been neglected – may it be time constraints or otherwise. I made a significant amount of progress this month and was really pleased overall.

Achievements

I chose to specialise in the IoT stream, as such most of my deliverables are due the end of the semester. I decided to dedicate a lot of time in February to the Software Project. My aim was to get most of the primary functionality and some more GUI enhancements completed by the end of February.

To do this, I primarily focused on Software Project, putting other project to the side for a while. This month went well, and I got a lot of functionality working. I will break down the achievements via each application section. Some of the sections that I completed/improved were:

Spoonacular API

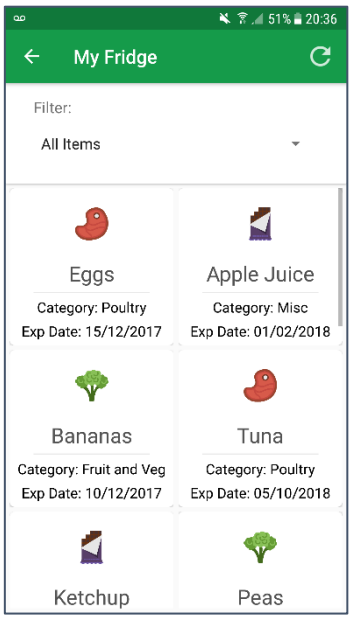
This was the primary API I wanted to use, as outlined in my proposal and Technical Report. I got in contact with the student support team at Spoonacular, to which they provided Student access to the API at 10$ per month - which I thought was very fair. Once I had attained access, I created the views and other elements in my application which could utilize the API. This enabled me to pull in recipes, food data, cooking instructions and much more.

My Food Network

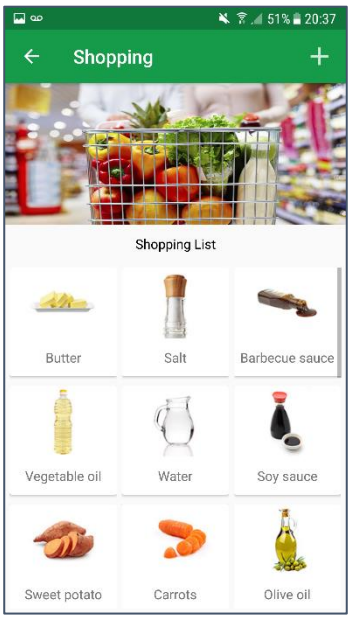
Originally, users could view food items scanned in via RFID. After extensive work on this section, users can now view recipes and nutritional data based on the food produce they have in their home.

Shopping Lists

I got a lot of work done in the shopping list section. From here, users can now add items to their shopping list, view nutritional data and also add items to the list via the recipe instructions screen.



My Food Network Section

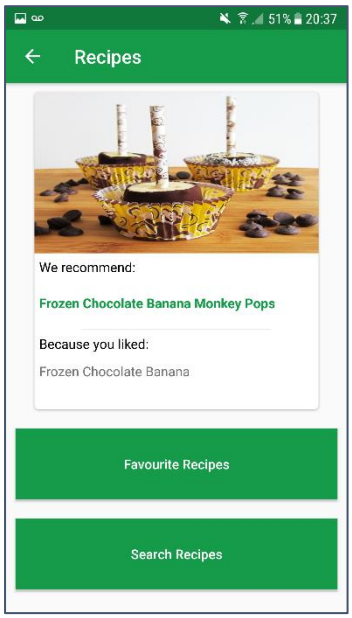


Shopping List Section

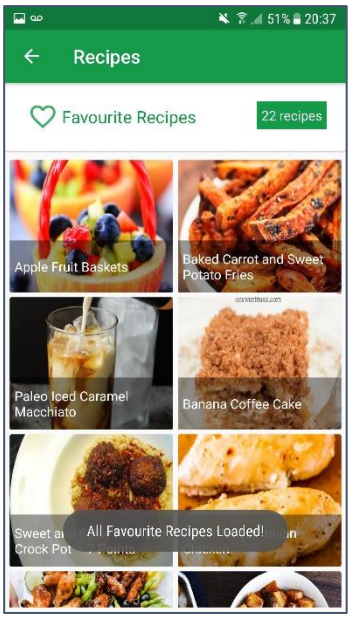
Recipes Section

I spent a lot of time on this section, as such it is the most populated and featured within the application. This entire section is focused around the Spoonacular API. Users can now;

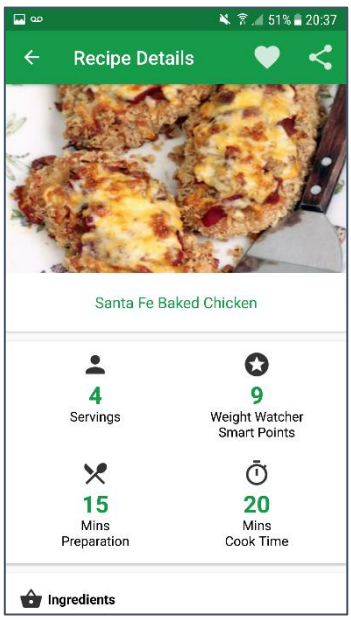
- View recipes based on food items they have in their home
- Search for recipes via ingredient
- Add recipes to their favourites
- Remove recipes from their favourites
- View similar recipes
- Share recipes via social networking/email/sms
- Users will also be recommended a recipe recommendation based on their favourite recipes



Recipes Section – Featured Recommendation



Favourite Recipes



Recipe Details

Nutrients Search

Users can search any food item and will be returned nutritional values via the Nutritionix API. I completely re-vamped this section

GUI Enhancements

I also focused much of my attention to GUI. I wanted to create a more professional looking application so looked at other recipe and food apps. The colour scheme, buttons, layouts and much more changed this month. Overall, I was very happy – I put a lot of work into the Recipe Details section and in fairness I was very happy with the results as I thought it looked very minimalistic and clean.

Reflection

Overall, I was very happy with how well the project came together. Within about a two-week period I bashed out a load of hours into the project and it turned out to be well worth the time. I was happy to get the API working and pulling data. I really enjoyed doing the project this month as I usually like designing the GUI. There were a few hiccups along the way but after some research and more testing I could solve any issues. I had aimed to get a large portion of the application functionality completed by the end of February and I think a lot of it went perfectly to plan.

Intended Changes

I hope to work on the application some more, add some more features such as the Profile Section, finish off the shopping list section and also add the user account functionality.

Supervisor Meetings

Thursday 8th February
Thursday 15th February

I met with Glen on the usual days which is Thursday. It is a great help especially getting solid feedback as it helps in evaluating my current progress. I usually like to meet with Glen each week to review progress, although some IoT deliverables were due and my time was pulled away to focus on that near the end of February. Nevertheless, Glen was still helping me via email.

12.7 March/April/May 2018

Reflection

The past 3 months have been very busy. Unfortunately, I slipped up and never got a chance to individually write up March, April and May's Monthly Reports. Therefore, I'm going to combine them into one final summary.

About 5 minutes ago I just finished compiling the references for the Technical Report. At this stage, and to be pure honest I don't want to look at another word document for a long time...

The Months March, April and May had many deliverables due – Data Mining & Visualisation, Cloud Application Development and IoT Application Development final CA's were all within a week of each other towards the end of April. I had to put the Software Project aside from around the start of March until all the CA's were uploaded. The work done in February really benefitted me as almost all Primary functionalities I wanted to implement had been done already. This took a lot of pressure off.

IoT Application Development was due the 29th of April, since then I've solely worked on the Software Project. The final touch ups for the Application took about a week and brought me up to last Saturday (5th May). Since then, I've worked day and night on the Technical Report Document. Looking at it now I'm personally very happy with the overall Result. It's quite funny the biggest issue was formatting the Figures. The numbering completely messed up, so I had to put any images that were horizontally aligned into tables, then re-add the figure reference. This was actually one of the most tedious parts of the entire project. Oh, and the Shopping List Icon – well who knew that would take so long…

Overall, it's been a tough year, trying to balance work, college and social life was difficult but I suppose you get out what you put in. Looking forward to starting the Apprenticeship in June – get away from the PC for a while.

Intended Changes

Proof Read the Technical Report, Upload final deliverables and Prepare for the Final Presentation.

Supervisor Meetings

Thursday 8th March
Thursday 5th April
Thursday 12th April

I met with Glen over the past few months, although there was more of a break between meetings due to deliverables and the 2-week break. Contact was still made via email regarding the project. Glen sent back recommendations to make for the final Technical Report, all of which were very useful and have been completed.
Thanks for the help throughout the year Glen, it's been a pleasure and a good bit of craic.

13 Appendix C – User Testing Documentation

13.1 Ethics Disclosure Form

The Intelligent Food Network

Consent to take part in research

- I..... voluntarily agree to participate in this research study.
- I understand that even if I agree to participate now, I can withdraw at any time or refuse to answer any question without any consequences of any kind.
- I understand that I can withdraw permission to use data from my survey/ interview at any time, in which case the material will be deleted.
- I have had the purpose and nature of the study explained to me in writing and I have had the opportunity to ask questions about the study.
- I understand that participation involves... using the Raspberry Pi RFID reader and Android application
- I understand that I will not benefit directly from participating in this research.
- I understand that all information I provide for this study will be treated confidentially.
- I understand that in any report on the results of this research my identity will remain anonymous. This will be done by changing my name and disguising any details of my interview which may reveal my identity or the identity of people I speak about.
- I understand that signed consent forms and original audio recordings will be retained in a secure location (NCI Student Servers) until September 2018.
- I understand that under freedom of information legalisation I am entitled to access the information I have provided at any time while it is in storage as specified above.
- I understand that I am free to contact any of the people involved in the research to seek further clarification and information. Creator: Leon Mulvaney, Supervisor: Glen Ward, Stream Co-Ordinator: Dominic Carr

Signature of research participant ---------- Date ----------

Signature of researcher---------- Date ----------

13.2 Time Scans

External Testing - System Usage
Time Taken

Leon Mulvaney 14445618 BSHC4IOT

Candidate Number: 1 **Date:** 11/05/2018

Step Number:	Time Taken:	Issues with step:
1	60 sec	—
2	3 sec	—
3	12 sec	—
4	38 sec	—
5	5 sec	—
6	3 sec	—
7	10 sec	—
8	7 sec	—
9	10 sec	—
10	7 sec	—
11	3 sec	—
12	4 sec	—
13	8 sec	—
14	7 sec	—
15	5 sec	—
16	3 sec	—
17	14 sec	—
18	14 sec	—
19	10 sec	—
20	40 sec	—
21	40 sec	—

Additional Comments/Observations:

Step 21 time will vary based on users typing speed / amount of content they put into email.

External Testing - System Usage
Time Taken

Leon Mulvaney 14445618 BSHC4IOT

Candidate Number: 2 **Date:** 11/05/2018

Step Number:	Time Taken:	Issues with step:
1	50 sec	—
2	4 sec	—
3	10 sec	—
4	28 sec	—
5	) sec	—
6	3 sec	—
7	15 sec	—
8	14 sec	—
9	5 sec	—
10	12 sec	—
11	4 sec	—
12	3 sec	—
13	10 sec	—
14	5 sec	—
15	8 sec	—
16	7 sec	—
17	17 sec	—
18	7 sec	—
19	14 sec	—
20	45 sec	—
21	70 sec	—

Additional Comments/Observations:

User currently owns Iphone, yet still navigated app with ease.

External Testing - System Usage
Time Taken

Leon Mulvaney 14445618 BSHC4IOT

Candidate Number: 3 **Date:** 11/05/2018

Step Number:	Time Taken:	Issues with step:
1	68 sec	—
2	5 sec	—
3	16 sec	—
4	37 sec	—
5	3 sec	—
6	3 sec	—
7	16 sec	—
8	10 sec	—
9	11 sec	—
10	11 sec	—
11	5 sec	—
12	4 sec	—
13	14 sec	—
14	20 sec	User hesitated before item added (Longer)
15	4 sec	—
16	9 sec	—
17	17 sec	—
18	9 sec	—
19	14 sec	—
20	55 sec	—
21	80 sec	—

Additional Comments/Observations:

User took extra time to add item to shopping list.
Also hesitated when "contacting Developers".

External Testing - System Usage
Time Taken

Leon Mulvaney 14445618 BSHC4IOT

Candidate Number: 4 **Date:** 11/05/2018

Step Number:	Time Taken:	Issues with step:
1	54 sec	—
2	4 sec	—
3	14 sec	—
4	40 sec	—
5	4 sec	—
6	3 sec	—
7	12 sec	—
8	11 sec	—
9	5 sec	—
10	9 sec	—
11	4 sec	—
12	5 sec	—
13	12 sec	—
14	18 sec	—
15	3 sec	—
16	5 sec	—
17	16 sec	—
18	10 sec	—
19	16 sec	—
20	62 sec	—
21	57 sec	—

Additional Comments/Observations:

N/A.

External Testing - System Usage
Time Taken

Leon Mulvaney 14445618 BSHC4IOT

Candidate Number: 5 **Date:** 11/05/2018

Step Number:	Time Taken:	Issues with step:
1	62 sec	—
2	6 sec	—
3	15 sec	—
4	34 sec	—
5	4 sec	—
6	3 sec	—
7	16 sec	—
8	12 sec	—
9	7 sec	—
10	5 sec	—
11	6 sec	—
12	7 sec	—
13	13 sec	—
14	16 sec	—
15	5 sec	—
16	6 sec	—
17	15 sec	—
18	12 sec	—
19	14 sec	—
20	51 sec	—
21	50 sec	—

Additional Comments/Observations:

N/A.

External Testing - System Usage
Time Taken

Leon Mulvaney 14445618 BSHC4IOT

Candidate Number: 6 **Date:** 11/05/2018

Step Number:	Time Taken:	Issues with step:
1	57 sec	—
2	5 sec	—
3	14 sec	—
4	41 sec	—
5	4 sec	—
6	4 sec	—
7	15 sec	—
8	13 sec	—
9	8 sec	—
10	8 sec	—
11	4 sec	—
12	5 sec	—
13	14 sec	—
14	17 sec	—
15	5 sec	—
16	5 sec	—
17	16 sec	—
18	11 sec	—
19	14 sec	—
20	55 sec	—
21	60 sec	—

Additional Comments/Observations:

N/A.

External Testing - System Usage
Time Taken

Leon Mulvaney 14445618 BSHC4IOT

Candidate Number: 7 **Date:** 11/05/2018

Step Number:	Time Taken:	Issues with step:
1	65 sec	—
2	5 sec	—
3	14 sec	—
4	38 sec	—
5	4 sec	—
6	3 sec	—
7	16 sec	—
8	13 sec	—
9	8 sec	—
10	7 sec	—
11	5 sec	—
12	15 sec	—
13	14 sec	—
14	17 sec	—
15	5 sec	—
16	9 sec	—
17	15 sec	—
18	10 sec	—
19	16 sec	—
20	49 sec	—
21	51 sec	—

Additional Comments/Observations:

N/A.

External Testing - System Usage
Time Taken

Leon Mulvaney 14445618 BSHC4IOT

Candidate Number: 8 **Date:** 11/05/2018

Step Number:	Time Taken:	Issues with step:
1	64 Sec	—
2	5 Sec	—
3	16 sec	—
4	42 Sec	—
5	4 Sec	—
6	4 Sec	—
7	16 Sec	—
8	12 Sec	—
9	9 Sec	—
10	7 Sec	—
11	6 Sec	—
12	5 Sec	—
13	15 Sec	—
14	16 Sec	—
15	9 Sec	—
16	6 Sec	—
17	14 Sec	—
18	11 Sec	—
19	16 Sec	—
20	60 Sec	—
21	84 Sec	Contact Developers Issue

Additional Comments/Observations:

User could not locate the "contact developers" section. Took extra time clicking multiple options before it was found.

External Testing - System Usage
Time Taken

Leon Mulvaney 14445618 BSHC4IOT

Candidate Number: 9 **Date:** 11/05/2018

Step Number:	Time Taken:	Issues with step:
1	59 sec	—
2	6 sec	—
3	10 sec	—
4	41 sec	—
5	4 sec	—
6	4 sec	—
7	21 sec	—
8	11 sec	—
9	9 sec	—
10	6 sec	—
11	5 sec	—
12	9 sec	—
13	11 sec	—
14	16 sec	—
15	4 sec	—
16	9 sec	—
17	14 sec	—
18	11 sec	—
19	29 sec	Issue with search.
20	57 sec	—
21	61 sec	—

Additional Comments/Observations:

Nutritional values - user attempted to go back into "Food Network" Before realising "nutrients" search was part of "Hamburger Menu".

External Testing - System Usage
Time Taken

Leon Mulvaney 14445618 BSHC4IOT

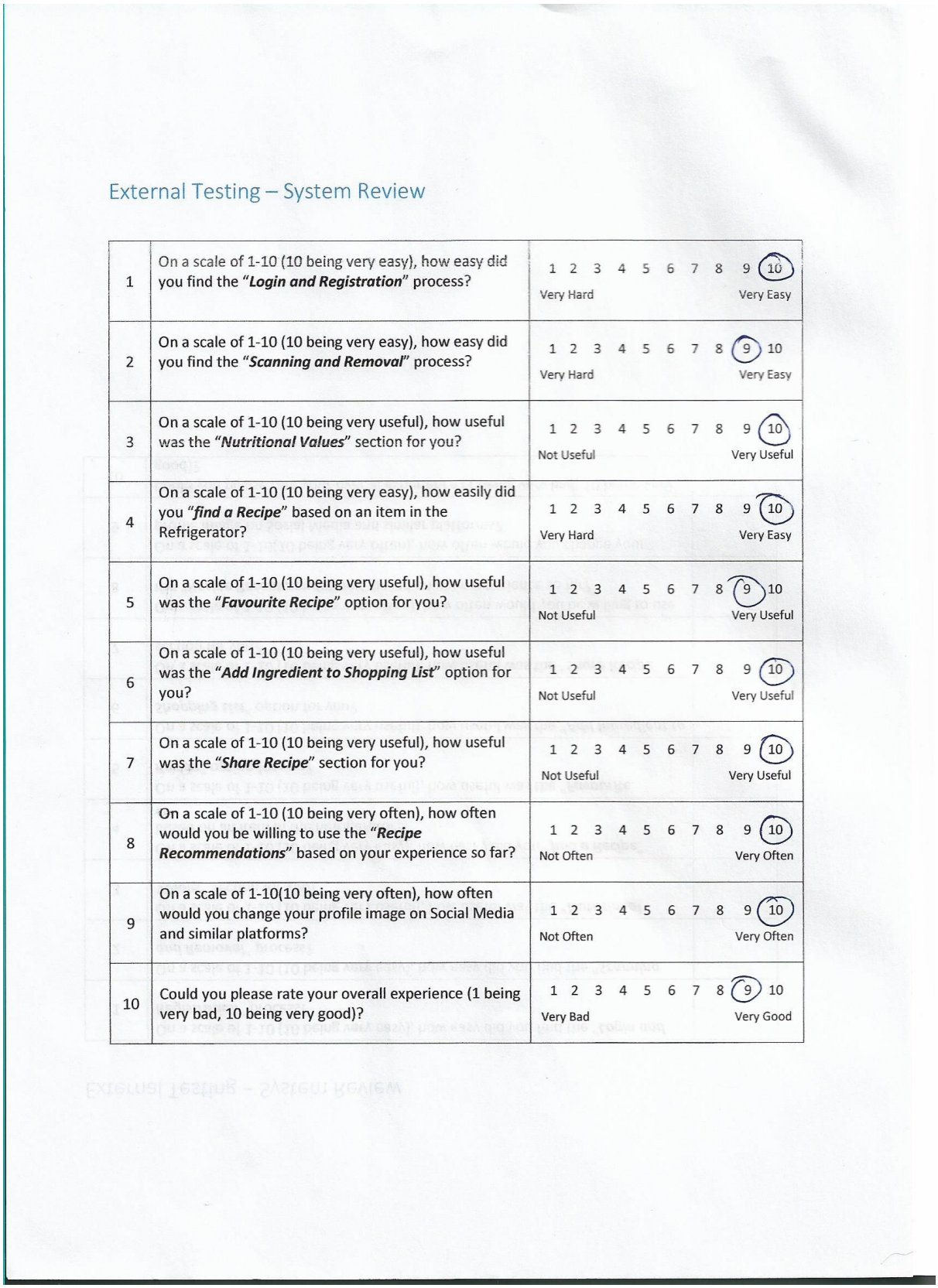
Candidate Number: 10 **Date:** 11/05/2018

Step Number:	Time Taken:	Issues with step:
1	61 Sec	—
2	5 sec	—
3	12 sec	—
4	32 sec	—
5	4 sec	—
6	4 sec	—
7	15 sec	—
8	14 sec	—
9	9 sec	—
10	6 sec	—
11	5 sec	—
12	6 sec	—
13	14 sec	—
14	14 sec	—
15	7 sec	—
16	8 sec	—
17	15 sec	—
18	9 sec	—
19	12 sec	—
20	52 sec	—
21	47 sec	—

Additional Comments/Observations:

N/A-

13.3 Review Scans



External Testing – System Review

1	On a scale of 1-10 (10 being very easy), how easy did you find the "***Login and Registration***" process?	1 2 3 4 5 6 7 8 9 (10) Very Hard — Very Easy
2	On a scale of 1-10 (10 being very easy), how easy did you find the "***Scanning and Removal***" process?	1 2 3 4 5 6 7 8 (9) 10 Very Hard — Very Easy
3	On a scale of 1-10 (10 being very useful), how useful was the "***Nutritional Values***" section for you?	1 2 3 4 5 6 7 8 9 (10) Not Useful — Very Useful
4	On a scale of 1-10 (10 being very easy), how easily did you "***find a Recipe***" based on an item in the Refrigerator?	1 2 3 4 5 6 7 8 9 (10) Very Hard — Very Easy
5	On a scale of 1-10 (10 being very useful), how useful was the "***Favourite Recipe***" option for you?	1 2 3 4 5 6 7 8 (9) 10 Not Useful — Very Useful
6	On a scale of 1-10 (10 being very useful), how useful was the "***Add Ingredient to Shopping List***" option for you?	1 2 3 4 5 6 7 8 9 (10) Not Useful — Very Useful
7	On a scale of 1-10 (10 being very useful), how useful was the "***Share Recipe***" section for you?	1 2 3 4 5 6 7 8 9 (10) Not Useful — Very Useful
8	On a scale of 1-10 (10 being very often), how often would you be willing to use the "***Recipe Recommendations***" based on your experience so far?	1 2 3 4 5 6 7 8 9 (10) Not Often — Very Often
9	On a scale of 1-10(10 being very often), how often would you change your profile image on Social Media and similar platforms?	1 2 3 4 5 6 7 8 9 (10) Not Often — Very Often
10	Could you please rate your overall experience (1 being very bad, 10 being very good)?	1 2 3 4 5 6 7 8 (9) 10 Very Bad — Very Good

External Testing – System Review

1	On a scale of 1-10 (10 being very easy), how easy did you find the "***Login and Registration***" process?	1 2 3 4 5 6 7 (8) 9 10 Very Hard Very Easy
2	On a scale of 1-10 (10 being very easy), how easy did you find the "***Scanning and Removal***" process?	1 2 3 4 5 6 7 (8) 9 10 Very Hard Very Easy
3	On a scale of 1-10 (10 being very useful), how useful was the "***Nutritional Values***" section for you?	1 2 3 4 5 6 7 8 (9) 10 Not Useful Very Useful
4	On a scale of 1-10 (10 being very easy), how easily did you "***find a Recipe***" based on an item in the Refrigerator?	1 2 3 4 5 6 7 8 (9) 10 Very Hard Very Easy
5	On a scale of 1-10 (10 being very useful), how useful was the "***Favourite Recipe***" option for you?	1 2 3 4 5 6 7 8 9 (10) Not Useful Very Useful
6	On a scale of 1-10 (10 being very useful), how useful was the "***Add Ingredient to Shopping List***" option for you?	1 2 3 4 5 6 7 8 9 (10) Not Useful Very Useful
7	On a scale of 1-10 (10 being very useful), how useful was the "***Share Recipe***" section for you?	1 2 3 4 5 6 (7) 8 9 10 Not Useful Very Useful
8	On a scale of 1-10 (10 being very often), how often would you be willing to use the "***Recipe Recommendations***" based on your experience so far?	1 2 3 4 5 6 7 8 (9) 10 Not Often Very Often
9	On a scale of 1-10(10 being very often), how often would you change your profile image on Social Media and similar platforms?	1 2 3 4 5 6 7 (8) 9 10 Not Often Very Often
10	Could you please rate your overall experience (1 being very bad, 10 being very good)?	1 2 3 4 5 6 7 8 (9) 10 Very Bad Very Good

External Testing – System Review

1	On a scale of 1-10 (10 being very easy), how easy did you find the "***Login and Registration***" process?	1 2 3 4 5 6 7 8 (9) 10 Very Hard Very Easy
2	On a scale of 1-10 (10 being very easy), how easy did you find the "***Scanning and Removal***" process?	1 2 3 4 5 6 7 (8) 9 10 Very Hard Very Easy
3	On a scale of 1-10 (10 being very useful), how useful was the "***Nutritional Values***" section for you?	1 2 3 4 5 6 7 8 (9) 10 Not Useful Very Useful
4	On a scale of 1-10 (10 being very easy), how easily did you "***find a Recipe***" based on an item in the Refrigerator?	1 2 3 4 5 6 (7) 8 9 10 Very Hard Very Easy
5	On a scale of 1-10 (10 being very useful), how useful was the "***Favourite Recipe***" option for you?	1 2 3 4 5 6 7 8 (9) 10 Not Useful Very Useful
6	On a scale of 1-10 (10 being very useful), how useful was the "***Add Ingredient to Shopping List***" option for you?	1 2 3 4 5 6 7 8 (9) 10 Not Useful Very Useful
7	On a scale of 1-10 (10 being very useful), how useful was the "***Share Recipe***" section for you?	1 2 3 4 5 6 7 8 (9) 10 Not Useful Very Useful
8	On a scale of 1-10 (10 being very often), how often would you be willing to use the "***Recipe Recommendations***" based on your experience so far?	1 2 3 4 5 6 7 (8) 9 10 Not Often Very Often
9	On a scale of 1-10(10 being very often), how often would you change your profile image on Social Media and similar platforms?	1 2 3 4 5 6 7 8 (9) 10 Not Often Very Often
10	Could you please rate your overall experience (1 being very bad, 10 being very good)?	1 2 3 4 5 6 7 8 (9) 10 Very Bad Very Good

External Testing – System Review

1	On a scale of 1-10 (10 being very easy), how easy did you find the "***Login and Registration***" process?	1 2 3 4 5 6 7 8 9 10 Very Hard Very Easy
2	On a scale of 1-10 (10 being very easy), how easy did you find the "***Scanning and Removal***" process?	1 2 3 4 5 6 7 8 9 10 Very Hard Very Easy
3	On a scale of 1-10 (10 being very useful), how useful was the "***Nutritional Values***" section for you?	1 2 3 4 5 6 7 8 9 10 Not Useful Very Useful
4	On a scale of 1-10 (10 being very easy), how easily did you "***find a Recipe***" based on an item in the Refrigerator?	1 2 3 4 5 6 7 8 9 10 Very Hard Very Easy
5	On a scale of 1-10 (10 being very useful), how useful was the "***Favourite Recipe***" option for you?	1 2 3 4 5 6 7 8 9 10 Not Useful Very Useful
6	On a scale of 1-10 (10 being very useful), how useful was the "***Add Ingredient to Shopping List***" option for you?	1 2 3 4 5 6 7 8 9 10 Not Useful Very Useful
7	On a scale of 1-10 (10 being very useful), how useful was the "***Share Recipe***" section for you?	1 2 3 4 5 6 7 8 9 10 Not Useful Very Useful
8	On a scale of 1-10 (10 being very often), how often would you be willing to use the "***Recipe Recommendations***" based on your experience so far?	1 2 3 4 5 6 7 8 9 10 Not Often Very Often
9	On a scale of 1-10(10 being very often), how often would you change your profile image on Social Media and similar platforms?	1 2 3 4 5 6 7 8 9 10 Not Often Very Often
10	Could you please rate your overall experience (1 being very bad, 10 being very good)?	1 2 3 4 5 6 7 8 9 10 Very Bad Very Good

External Testing – System Review

1	On a scale of 1-10 (10 being very easy), how easy did you find the "***Login and Registration***" process?	1 2 3 4 5 6 7 (8) 9 10 Very Hard Very Easy
2	On a scale of 1-10 (10 being very easy), how easy did you find the "***Scanning and Removal***" process?	1 2 3 4 5 6 (7) 8 9 10 Very Hard Very Easy
3	On a scale of 1-10 (10 being very useful), how useful was the "***Nutritional Values***" section for you?	1 2 3 4 5 6 7 (8) 9 10 Not Useful Very Useful
4	On a scale of 1-10 (10 being very easy), how easily did you "***find a Recipe***" based on an item in the Refrigerator?	1 2 3 4 5 6 7 8 (9) 10 Very Hard Very Easy
5	On a scale of 1-10 (10 being very useful), how useful was the "***Favourite Recipe***" option for you?	1 2 3 4 5 6 7 8 (9) 10 Not Useful Very Useful
6	On a scale of 1-10 (10 being very useful), how useful was the "***Add Ingredient to Shopping List***" option for you?	1 2 3 4 5 6 (7) 8 9 10 Not Useful Very Useful
7	On a scale of 1-10 (10 being very useful), how useful was the "***Share Recipe***" section for you?	1 2 3 4 5 6 7 (8) 9 10 Not Useful Very Useful
8	On a scale of 1-10 (10 being very often), how often would you be willing to use the "***Recipe Recommendations***" based on your experience so far?	1 2 3 4 5 6 7 (8) 9 10 Not Often Very Often
9	On a scale of 1-10(10 being very often), how often would you change your profile image on Social Media and similar platforms?	1 2 3 4 5 6 7 (8) 9 10 Not Often Very Often
10	Could you please rate your overall experience (1 being very bad, 10 being very good)?	1 2 3 4 5 6 (7) 8 9 10 Very Bad Very Good

External Testing – System Review

1	On a scale of 1-10 (10 being very easy), how easy did you find the "***Login and Registration***" process?	1 2 3 4 5 6 7 **8** 9 10 Very Hard — Very Easy
2	On a scale of 1-10 (10 being very easy), how easy did you find the "***Scanning and Removal***" process?	1 2 3 4 5 6 7 8 **9** 10 Very Hard — Very Easy
3	On a scale of 1-10 (10 being very useful), how useful was the "***Nutritional Values***" section for you?	1 2 3 4 5 6 7 **8** 9 10 Not Useful — Very Useful
4	On a scale of 1-10 (10 being very easy), how easily did you "***find a Recipe***" based on an item in the Refrigerator?	1 2 3 4 5 6 7 8 **9** 10 Very Hard — Very Easy
5	On a scale of 1-10 (10 being very useful), how useful was the "***Favourite Recipe***" option for you?	1 2 3 4 5 6 7 **8** 9 10 Not Useful — Very Useful
6	On a scale of 1-10 (10 being very useful), how useful was the "***Add Ingredient to Shopping List***" option for you?	1 2 3 4 5 6 7 8 9 **10** Not Useful — Very Useful
7	On a scale of 1-10 (10 being very useful), how useful was the "***Share Recipe***" section for you?	1 2 3 4 5 6 **7** 8 9 10 Not Useful — Very Useful
8	On a scale of 1-10 (10 being very often), how often would you be willing to use the "***Recipe Recommendations***" based on your experience so far?	1 2 3 4 5 6 7 8 **9** 10 Not Often — Very Often
9	On a scale of 1-10(10 being very often), how often would you change your profile image on Social Media and similar platforms?	1 2 3 4 5 6 **7** 8 9 10 Not Often — Very Often
10	Could you please rate your overall experience (1 being very bad, 10 being very good)?	1 2 3 4 5 6 7 8 9 **10** Very Bad — Very Good

External Testing – System Review

1	On a scale of 1-10 (10 being very easy), how easy did you find the "***Login and Registration***" process?	1 2 3 4 5 6 7 **8** 9 10 Very Hard — Very Easy
2	On a scale of 1-10 (10 being very easy), how easy did you find the "***Scanning and Removal***" process?	1 2 3 4 5 6 7 8 **9** 10 Very Hard — Very Easy
3	On a scale of 1-10 (10 being very useful), how useful was the "***Nutritional Values***" section for you?	1 2 3 4 5 6 7 8 **9** 10 Not Useful — Very Useful
4	On a scale of 1-10 (10 being very easy), how easily did you "***find a Recipe***" based on an item in the Refrigerator?	1 2 3 4 5 6 7 8 **9** 10 Very Hard — Very Easy
5	On a scale of 1-10 (10 being very useful), how useful was the "***Favourite Recipe***" option for you?	1 2 3 4 5 6 7 8 9 **10** Not Useful — Very Useful
6	On a scale of 1-10 (10 being very useful), how useful was the "***Add Ingredient to Shopping List***" option for you?	1 2 3 4 5 6 7 8 9 **10** Not Useful — Very Useful
7	On a scale of 1-10 (10 being very useful), how useful was the "***Share Recipe***" section for you?	1 2 3 4 5 6 7 **8** 9 10 Not Useful — Very Useful
8	On a scale of 1-10 (10 being very often), how often would you be willing to use the "***Recipe Recommendations***" based on your experience so far?	1 2 3 4 5 6 7 8 9 **10** Not Often — Very Often
9	On a scale of 1-10(10 being very often), how often would you change your profile image on Social Media and similar platforms?	1 2 3 4 5 6 **7** 8 9 10 Not Often — Very Often
10	Could you please rate your overall experience (1 being very bad, 10 being very good)?	1 2 3 4 5 6 7 8 **9** 10 Very Bad — Very Good

External Testing – System Review

1	On a scale of 1-10 (10 being very easy), how easy did you find the "***Login and Registration***" process?	1 2 3 4 5 6 7 8 (9) 10 Very Hard — Very Easy
2	On a scale of 1-10 (10 being very easy), how easy did you find the "***Scanning and Removal***" process?	1 2 3 4 5 6 7 8 (9) 10 Very Hard — Very Easy
3	On a scale of 1-10 (10 being very useful), how useful was the "***Nutritional Values***" section for you?	1 2 3 4 5 6 7 (8) 9 10 Not Useful — Very Useful
4	On a scale of 1-10 (10 being very easy), how easily did you "***find a Recipe***" based on an item in the Refrigerator?	1 2 3 4 5 6 7 (8) 9 10 Very Hard — Very Easy
5	On a scale of 1-10 (10 being very useful), how useful was the "***Favourite Recipe***" option for you?	1 2 3 4 5 6 7 8 (9) 10 Not Useful — Very Useful
6	On a scale of 1-10 (10 being very useful), how useful was the "***Add Ingredient to Shopping List***" option for you?	1 2 3 4 5 6 7 (8) 9 10 Not Useful — Very Useful
7	On a scale of 1-10 (10 being very useful), how useful was the "***Share Recipe***" section for you?	1 2 3 4 5 6 7 (8) 9 10 Not Useful — Very Useful
8	On a scale of 1-10 (10 being very often), how often would you be willing to use the "***Recipe Recommendations***" based on your experience so far?	1 2 3 4 5 6 7 8 (9) 10 Not Often — Very Often
9	On a scale of 1-10(10 being very often), how often would you change your profile image on Social Media and similar platforms?	1 2 3 4 (5) 6 7 8 9 10 Not Often — Very Often
10	Could you please rate your overall experience (1 being very bad, 10 being very good)?	1 2 3 4 5 6 7 8 (9) 10 Very Bad — Very Good

External Testing – System Review

1	On a scale of 1-10 (10 being very easy), how easy did you find the "***Login and Registration***" process?	1 2 3 4 5 6 7 8 9 10 Very Hard Very Easy
2	On a scale of 1-10 (10 being very easy), how easy did you find the "***Scanning and Removal***" process?	1 2 3 4 5 6 7 8 9 10 Very Hard Very Easy
3	On a scale of 1-10 (10 being very useful), how useful was the "***Nutritional Values***" section for you?	1 2 3 4 5 6 7 8 9 10 Not Useful Very Useful
4	On a scale of 1-10 (10 being very easy), how easily did you "***find a Recipe***" based on an item in the Refrigerator?	1 2 3 4 5 6 7 8 9 10 Very Hard Very Easy
5	On a scale of 1-10 (10 being very useful), how useful was the "***Favourite Recipe***" option for you?	1 2 3 4 5 6 7 8 9 10 Not Useful Very Useful
6	On a scale of 1-10 (10 being very useful), how useful was the "***Add Ingredient to Shopping List***" option for you?	1 2 3 4 5 6 7 8 9 10 Not Useful Very Useful
7	On a scale of 1-10 (10 being very useful), how useful was the "***Share Recipe***" section for you?	1 2 3 4 5 6 7 8 9 10 Not Useful Very Useful
8	On a scale of 1-10 (10 being very often), how often would you be willing to use the "***Recipe Recommendations***" based on your experience so far?	1 2 3 4 5 6 7 8 9 10 Not Often Very Often
9	On a scale of 1-10(10 being very often), how often would you change your profile image on Social Media and similar platforms?	1 2 3 4 5 6 7 8 9 10 Not Often Very Often
10	Could you please rate your overall experience (1 being very bad, 10 being very good)?	1 2 3 4 5 6 7 8 9 10 Very Bad Very Good

External Testing – System Review

1	On a scale of 1-10 (10 being very easy), how easy did you find the "***Login and Registration***" process?	1 2 3 4 5 6 7 8 **9** 10 Very Hard — Very Easy
2	On a scale of 1-10 (10 being very easy), how easy did you find the "***Scanning and Removal***" process?	1 2 3 4 5 6 7 8 9 **10** Very Hard — Very Easy
3	On a scale of 1-10 (10 being very useful), how useful was the "***Nutritional Values***" section for you?	1 2 3 4 5 6 7 8 **9** 10 Not Useful — Very Useful
4	On a scale of 1-10 (10 being very easy), how easily did you "***find a Recipe***" based on an item in the Refrigerator?	1 2 3 4 5 6 7 **8** 9 10 Very Hard — Very Easy
5	On a scale of 1-10 (10 being very useful), how useful was the "***Favourite Recipe***" option for you?	1 2 3 4 5 6 7 8 **9** 10 Not Useful — Very Useful
6	On a scale of 1-10 (10 being very useful), how useful was the "***Add Ingredient to Shopping List***" option for you?	1 2 3 4 5 6 7 8 9 **10** Not Useful — Very Useful
7	On a scale of 1-10 (10 being very useful), how useful was the "***Share Recipe***" section for you?	1 2 3 4 5 6 7 **8** 9 10 Not Useful — Very Useful
8	On a scale of 1-10 (10 being very often), how often would you be willing to use the "***Recipe Recommendations***" based on your experience so far?	1 2 3 4 5 6 7 8 9 **10** Not Often — Very Often
9	On a scale of 1-10(10 being very often), how often would you change your profile image on Social Media and similar platforms?	1 2 3 4 5 6 7 **8** 9 10 Not Often — Very Often
10	Could you please rate your overall experience (1 being very bad, 10 being very good)?	1 2 3 4 5 6 7 8 9 **10** Very Bad — Very Good

14 Appendix D – Project Poster