MSc Computer Science

Project Report

Ubiquitous Consumer Inventory Management System For Waste Prevention

Supervisor: Professor George Roussos
Author: Keimi Okamoto
2015

BIRKBECK UNIVERSITY OF LONDON
DEPARTMENT OF COMPUTER SCIENCE & INFORMATION SYSTEMS

Contents

1 Introduction

1.1 Abstract

This report details the creation of a home food inventory management system comprised of an RFID embedded fridge and counterpart Android mobile application. The system intends to aid the reduction of food waste generated by the households. Food waste negatively impacts our environment and society, in the UK alone fifteen million tonnes of food is wasted every year, 7 million of which is ac- countable to domestic households.[5] In terms of monetary waste, on average it has been reported that a typical household will waste £470 a year, and £700 for a family with children.[3] By providing the consumer with an ubiquitously accessible system and an appliance capable of automated logging of food products, the user may access their inventory anytime any where and in turn reducing the likelihood of waste occurring. *Supervisor: G.Roussos*

1.2 Structure of the report

The chapter, "Background & Analysis", covers the problem and motivation for the development of the system. Currently used technologies and methods of waste management are documented and analysed, then the proposed system is presented to the reader followed by an analysis of the technologies to be used in the development phase.

The next chapter documents the design of the system using the analysis gathered in the previous stage, the system can be visualised using best practices suggested by Teagarden and Wixom in the book 'System Analysis and Design with UML'. Requirements are presented and organised into functional and non-functional and use cases are presented to understand how the end user is is to use the product and finally the techniques which are to be used in the development phase are presented to prepare the reader for the next chapter.

The fourth chapter entails the implementation of the system. Components are split into sub-chapters so the reader may navigate to the desired section. Each sub-chapter has a detailed description on the method of implementation.

The firth chapter presents the results for the testing of the finished system. Unit tests and mocking frameworks will be discussed and manual testing reports will be shown.

The final chapter concludes the report with a critical analysis of the system.

1.3 Development Methodology

The development methodology used will be the agile development methodology[19]. Although the project will be carried out by a single developer the benefits of an iterative development process can still be acquired. Process visualisation and requirement gathering will utilise the UML workflow, following these and principals and guidelines will help keep the project on track. Additionally, any code written will be documented using the distributed version control service GitHub.

2 Background & Analysis

The first section of this chapter presents the problem and motivation for the creation the application. Then an analysis of currently available processes that are implemented to tackle the problem will be discussed. The last section will provide an analysis of the technologies that will aid the development of the system leading to the design of the application.

2.1 Motivation

2.1.1 Environment

Globally, food waste has exceeded to two billion tonnes annually.[1] This excessive generation of waste has a severe impact on our environment. Spoiling foods emit methane, a particularly harmful greenhouse gas contributing to the warming of the earth's surface and causing the expansion of the hole in the earths ozone layer. The ozone layer is vital to the survival of living organisms as it protects against harmful radiation form the sun. Too much exposure can result to illnesses such as cancer and eye damage.

Resources used during the rearing and plantation process for produce as well as the manufacturing materials for packaging and fuel used for transportation all contribute to the detrimental effect on our environment. Councils are continuously pressured to fuel more funds into waste management recourses and landfill sites are overflowing at such a rapid pace space to accommodate the waste is fast diminishing.[2]

In the United Kingdom alone consumers and households are responsible for at least seven million tonnes of waste, whilst in comparison the retailers contribute a mere two hundred and twenty five thousand tonnes[5]. Although the reported figures implicate the household as the primary culprit, retailers are in fact the catalyst force behind the mass generation of waste. Competition for market share is fierce between the retailers and goods must always be available at a lower price than that of their neighbours. Frequent buy-one-get-one-free and multi-buy discounts hosted by retailers contribute largely to the problem. Farmers and other producers alike are pressured to over produce to accommodate for the probability of a sudden rise in demand or to simply cover the risk of a potentially bad harvest. But when the sales forecast is not met or the harvest is overly fruitful, supermarkets will routinely deduct the price of food nearing expiration as a method of damage control for their investment. Consumers are enticed by the attractive offer of a free item after purchasing two, and encouraged to over purchase food that they do not need and will most likely not consume before the use-by-date. Thus, waste that was originally created by the retailers is pushed down the chain and ultimately residing with the consumers. This aggressive sales strategy not only causes monetary waste to the consumer but also engineers

the blind participation in driving up demands in goods, in turn encouraging the overproduction of food.

2.1.2 Public Health

In addition to the environmental damage, there are growing concerns over public health risks the bargains offers bring. Over purchasing of food can encourage excessive consumption. Discounted foods usually come with a shorter use-by-date, meaning over a shorter span of time an individual must consume more than necessary in order for their investment to be justified. This side effect is detrimental to the health and well being of the public as the national statistics report for the United Kingdom has unveiled. Obesity rates in male adults have increased 13.2% and 7.4% for female adults since 1993. These figures are rising every year and the World Health Organisation (WHO) has predicted that by 2030 74% of male adults and 64% of female adults will be obese if precautionary measures are not implemented.[4]

2.1.3 Aiding the Modern Lifestyle

The final problem is simply human errors. We purchase produce with the good intention of consuming them. But as the demands of our fast paced modern life-style take priority, we often forget all about the existence of what we stocked and inadvertently let our stock expire. Expiry dates for products vary, and keeping track of them all is a near impossible task by relying on human memory alone. A popular study carried out in by George Armitage Miller, a prominent figure in the field of cognitive psychology discovered that the number of objects an average human can hold in working memory is seven, give or take two[6]. With this limited capacity it is no surprise that once the fridge door is closed, it is inevitable that some of the produce is destined for the waste bin.

Additionally many homes are made up of multiple inhabitants. Errors such as double purchasing due to lack of communication is a common occurrence. If two occupiers notice an item is low on stock at different times they may both set out to replace the item, resulting in duplicate items being purchased and increasing the risk of waste occurring.

2.2 Current Waste Reduction Methods

Governments and organisations have taken various measures to tackle the rising figures in waste. Below describe the techniques used and the strengths and weaknesses that each possess.

2.2.1 Manual Labour

Governments have launched Campaigns aimed to educate the public on the implications of waste. The registered charity, part funded by the government, Waste & Resources Action Programme (WRAP) have frequently interacted with communities by setting up stalls and public demonstrations to raise awareness. This form of engagement is inspirational and informative but also a very expensive operation and sustaining the interest of the community is difficult and effects are short lived.[1]

2.2.2 Anaerobic Digestion

Waste management organisation Biffa and the retail giant Sainsbury's have collaborated in an effort to convert waste into energy by recycling food waste using a method called anaerobic digestion. By utilising the gases produced from the decomposing food they are able to generate power which in turn is used to sustain the running of retail stores. Anaerobic digestion creates a circular process where the waste produced by the retailers is pumped back into the production line to fuel the very instrument creating the problem. This method has been questioned as to whether it is a true solution to the problem as the resources used throughout the process also leave a carbon footprint.[10]

2.2.3 Smart Fridge

Smart fridges were introduced in the early 2000's as a home inventory management system. Designed to be integrated into our every day lives and to monitor the inhabitants inventory. The user inputs product details on an embedded screen or mobile device. Some are equipped with barcode scanners as another method of registration. The fridge was designed to monitor inventory and to inform the owner if items are running low on stock. If the fridge detected that inventory was running low an order would automatically made to the retailer. The primary drawback of this product is the cost of the appliance and lack of infrastructure supporting an efficient product registration process. With one unit costing over $20,000 and items still dependant on manual registration or time consuming barcode scanning many were discouraged.[7]

2.2.4 Nano Technology

Most recently Nano technology has been used to monitor the stages of decomposition in foods. A small gel like cube emits a colour corresponding to a particular stage of decomposition and visually representing the shelf life of the product. Nano technology is able to remove the need for printed sell-by-dates but it still requires the inhabitants to actively open the fridge door,

view, process and memorise the colours of the tags for various products.[11]

2.2.5 Mobile Application

Existing waste management applications include the self titled application by 'Love Food Hate Waste' (LFHW). The application include features such as a shopping lists memo maker, recipe suggestions, portion size suggestions. The Netherlands Nutrition Centre Foundation (NNCF) funded by the Dutch government, have released an app 'Smart Cooking' that incorporates similar features to LFHW. TooSkee and LeanPath[2] are other examples of food management apps developed by organisations in the United States. Much like the other apps it will suggest dishes and remind the user to consume products before the expiration date.[2]

But these product logging applications suffer from the same bottleneck, the act of having to manually punch in the product details or scan each barcode with a reader. This arduous process dissuades many users and can result in the application being discarded.

2.2.6 Radio Frequency Identification

NXP Semiconductors, a Dutch research team have collaborated with the Netherlands Packaging Centre (NPC), to develop a sensor enabled RFID tag capable of monitoring the environmental changes though the supply chain. [12] The Pasteur sensor tag has the capacity of measuring shifts in temperature and gas conditions during transportation and various stages of storage. An accurate prediction of the product's shelf life is generated from the data collected, enabling prioritisation of the trading of supplies, reducing the likelihood of waste. At the currents state this is only available for large-scale shipments to suppliers from the producers.

2.3 Item Level Identification

The primary draw back of existing systems such as the smart fridge and the mobile applications is the lack of automation in the item registration process. Convincing consumers to alter habitual behaviours such as stocking the fridge is challenging and would most likely be met with resistance unless the new process yields better results than the current. Thus, the user having to either manually punch in the product name, or scan every item with a barcode reader is an unappealing process and can prevent the user from utilising the technology.

2.3.1 Current Item Level Product Identification Methods

Predominantly, the current methods of item level identification in the supply chain rely on the "picket-fence" style, one-dimensional barcodes [**?**] have

been prevalent in grocery stores since the 1970's and introduced as a means to manage inventory and to accelerate the checkout process. A laser scans the code and the catalogued code is used to retrieve details of the item, such as the name and cost. The scanning of the barcode requires that no objects obstruct the line of sight between the code and laser and even the smallest scratch or dirt can render the code unreadable. Simultaneous reads are not possible and codes must be scanned one by one with precision. Barcodes also have limited capacity; usually can only hold twenty to twenty-five characters. Furthermore, the code is limited to identifying a collection of items rather than the individual item level, omitting crucial fine-grain information such as sell-by-dates and the life-cycle of produce.

2.4 Proposed Approach to the Problem

In attempts to reduce the generation of food waste, the British government has intervened and officials pressured retailers to abolish the multi-buy offers. But the suggestion was met with reluctance. Instead a compromise was reached and the revised promotions allowing consumers a wider variety of products to choose from. Although figures declined periodically it was not maintained. It is evident from this that persuading retailers to prioritise the reduction of waste over the increase in revenue is an up hill struggle. Without constant monitoring, retailers will favour the increase in revenue over the betterment of the public.[1]

For this reason the proposed solution is not to persuade the retailer to change their practice but the consumer. By equipping the consumer with an application that is able to monitor their inventory, overstocking can be discouraged. The consumer will be driven by the incentive of monetary savings and as a side-effect reduce the risk of waste occurring at the consumer level. Theoretically, creating an upstream ripple effect that will keep the waste at bay with the retailers and forcing the termination of over production.

2.4.1 Proposed System

The proposed system is comprised of two key components. The first is the RFID client and the second an Android application. For the RFID application a RFID reader will reside in the fridge, monitoring the presence of any tags in its vicinity and facilitating the automatic registration when loading and unloading products from the fridge. The system will keep an accurate inventory of the users contents and communicate any changes to the user's smartphone.

2.4.2 Mobile Applications

As popularity in smartphones increase many applications have identified the need for ubiquitously accessible counterpart software. Whether it is in the

form of a web applications, such as the popular taxi booking application Uber or by using available mobile platforms such as IOS for Apple or Android, the way in which we utilise software has shifted from the traditional static machines to mobile systems. The convenient size and connectivity has enabled Smart devices to evolve rapidly and daily tasks such as checking emails can now be completed on the move. With many applications with similar functionalities available with a tap of a screen and with a market place where a single bad review can jeopardise the success of an app, quality and functionality is paramount, users have become increasingly intolerant of a poor interface design or performance such as delayed content loading.

2.4.3 Item Level Product Identification Using RFID

RFID utilises electromagnetic waves emitted by a reader. The tag is comprised of a coil and transistor that harvests the energy from the reader. Once the tag has enough power, the data is transmitted and the information may be read from the tag.

NFC registration and stock keeping systems have been successfully installed in retail stores, in particular clothing retailers. Before, logging of stock was a manual task. Employees would walk around the store floor with a laser reader and scan each item one by one. If the barcode had been damaged they would have to note the reference number down to enter into the system at a later time. With the use of RFID the employee now holds a reader and walks by the rack of garments, where the tags are read simultaneously, and the details captured with ease. Some grocery stores also use NFC but the technology is reserved for valuable goods to serve as an anti-theft device. Tags are applied in the form of a sticker and when detected by the readers, normally situated at the store exits, it will trigger an alarm to notify staff.

NFC has a few advantages over the traditional method of product identification. Firstly the tag does not require a clear line of sight. The reader may interrogate any tag within the vicinity of the transmitted electromagnetic field. The energy is passable through objects and some readers can even facilitate simultaneous reads. Secondly, tags are more durable than the traditional printed kind and can be embedded within the packaging or inconspicuously applied to the product. NFC also has more space to hold data.

With the use of RFID the consumer need not alter any behaviour. The item registration process is completely automated, removing the need for the consumer to manually enter product information or line-up lasers with barcodes. Some other benefits can also be experienced through the application of item-level RFID that will be highlighted in the next section.

2.5 Case Studies to Support the Use of RFID at the Item-Level

Remote Amendment of Human Errors Due to human errors such as misprinting information or neglecting to provide adequate information such as allergy information, that abide by food standard regulations perfectly consumable food is recalled and wasted.[**?**] By applying RFID tags to groceries, information could be updated remotely, issuing immediate alerts to consumers informing them of the mistake and the amended error. This provides a different approach to error management.

Food Safety and Traceability In the past there has been numerous incidents when products have been recalled due to the presence of bacteria or other abnormalities. A notable incident is the 2013 meat adulteration scandal in the EU, where traces of horse meat where discovered in various products such as minced meat and ready prepared meals. The time and resources to trace back through the supply chain was estimated to have cost the Food Standard Agency (FSA) £900,000 between 2011 and 2012 and a further £1.6 million between 2012 and 2013 [16]. Other casualties include the reputations and integrity of the blameless producers falsely accused due to limited and inaccurate information that implicated them as the guilty.[9]

With item level identification the contaminated produce could be traced back immediately and the products recalled.[12][13] For example if an infected animal is used in various products, all items holding that particular code can be instantly traceable, effectively compartmentalising the outbreak and maximising efficiency in damage control.

Transparency & Consumer Rights The need for transparency of the origin of the produce is important to better the quality of living. Granular information can help consumers control what they consume to aid a healthy life. Produce information is often documented by the producers and passed to the supplier but the information fails to make it down the chain.[16] Meats in particular have unique backgrounds. Ranging from the rearing environments, type of feed consumed and drugs administered. This information can provide awareness to consumers whether it is for health reasons, environmental or the ethically conscious individual.

Such areas as Japan where vegetables and livestock were exposed to nuclear radiation due to the Fukushima Daiichi disaster raises serious health concerns. Damage caused by radiation exposure by consumption of contaminated foods can surface much later in a persons life and in some cases even be inherited by offspring. This highlights the urgency for transparent detail of the product's life cycle and the importance for the consumer to be fully aware of the risks and responsible for what they consume.[**?**]

The law enforces that the label on fresh meat must contain the country of origin. But this does not apply to the same meat that is processed, such as hamburgers, pies and sausages. Meats may be mixed providing that the animals are slaughtered in the same country, meaning a single hamburger could be made up of several cows.[16] Officials have used over crowding of labels as a reason not to provide the customer with details of a produce and has deemed it unnecessary.

> "*It is clear that many consumers want more information on the origin of meat ingredients in meat products, and in the Agency's consumer research the ingredients in dairy produce also score highly in this respect. The law requires an origin declaration on fresh beef but not on the same product when it has been seasoned. Providing information on the origin of all ingredients in all products would be disproportionately burdensome for industry, and would risk overloading the label with information that is not seen as important by consumers.*"- Food Standard Agency, *'country of origin labelling guidance'*[16]

With the use of RFID the overloading of the label would no longer be a reason to withhold information from the consumer. Allowing the individual to decide what information is of importance.

2.5.1 Efficient Checkout Process

The traditional checkout process requires the cashier to scan each item one by one, even with the new self checkout systems introduced to reduce queuing time, the line is often held up by an unreadable barcode or the numerous failed attempts of the laser initiating a read. Items must be loaded out of the basket or shopping-trolley and places into a shopping bag. As demonstrated by MyGrocer, a system that was designed to modernise the shopping experience, utilises a trolley lined with RFID readers. As tagged products are loaded the readers record the presence of goods and upon checkout all the user needs to do is pay a bill.

2.5.2 Discounting Tags

When products are nearing expiration it is common practice for the retailers to discount the products. A store assistant will usually carry a barcode-printing device that generates a sticker with the discounted code. He or she will manually check the date and if the item is eligible a sticker with a new discounted barcode will replace the original. With the use of RFID the amendment can be made remotely and automatically, filtering and amending only the relevant products.

2.6 Technology Risk Analysis

2.6.1 Mobile Platforms

Currently the most popular mobile platforms are Android and Apple IOS. Android currently has the largest user base, boasting an estimate of one billion users. IOS applications are written in languages such as Objective C, which require the developer to manage the memory of objects. Recently, a new language Swift was introduced by Apple. Swift supports automatic reference counting (ARC), this eliminates large memory leaks that frequently occurred in it's predecessor Objective C. Android on the other hand has a high-performance memory management system that cleans up unused objects, also know as the garbage collector. This feature allows the developer to focus their time and attention on the functionality of the application. Android applications are written in Oracle Java. It is also note worthy to mention that Android has a unique ecosystem unlike that of a typical Java application. The developer must understand and abide by the rules of the environment, failing to do so can result in undesired outcomes. [17][18]

Swift being a relatively young language means that less support is available compared to the others, increasing the risks of encountering difficulties. Objective C will require diligent memory management and can shift focus from the core of the application development and problems may arise from the lack of personal expertise. Having previous exposure to Java, as it was taught as a core module (Programming in Java), the Android development environment was a sensible option. The familiarity of the language and integrated memory management feature minimised the risk of encountering difficulties. Android also has a larger user base making the application accessible to a wider audience.

2.6.2 Cloud Services

Cloud services have gained notoriety due to the I.T giants Google, Amazon and Microsoft. The underlining principals of cloud computing of automated scalability and constant availability of resources have converted the way I.T businesses operate. Enabling small organisations to begin developing with minimal capital expenditure. Traditional in house server farms require thousands of pounds for initial setup, but by delegating the responsibility of upgrades, maintenance and load balancing to a cloud vendor, focus can be kept on developing the application. To further emphasis, more businesses are adopting the agile methodology where change is to be expected and must be embraced[19], cloud services facilitates change by only charging the user for the resources consumed, in effect renting the software and hardware, giving teams agility to react to change with minimal financial risk.

Utility bill style billing and the ability to the respond to peaks and troughs of demand are ideal for this project. Much like most online retailers

where for the most part experience steady traffic, but during the holiday seasons and sale periods could potentially experience a sudden peak when consumers are most active. In the past the unpredictable nature of human activity has resulted in server crashes but with the use of a cloud service the system can scale seamlessly.

Out of the three leading cloud service Google Cloud Platform (GCP), Amazon Web Services (AWS) and Microsoft Azure the most fitting is AWS. Azure is recommended for those who are currently utilising the Microsoft stack such as C# and MySql neither of which is relevant for this project. GAE's data analytics engine has been praised as superior to that of AWS owed to the massive amounts of data harvested by Google.[21] [20] But in terms of services available AWS is without a doubt the leading vendor (see image). AWS also has higher availability with eleven geographical locations that resources can be distributed to and users anywhere can efficiently access, in comparison Google has only three regions. Although a notable disadvantage with AWS's myriad of services is grasping the understanding of the complex network topology, this creating a steeper learning curve than GCP. Although GCP is simpler and user friendly it has been concluded that the positive elements of AWS has outweighed the negative.

2.6.3 RFID Technology

There are many different classifications of RFID readers and tags that differ in the amount of memory it can hold to the range and direction it may transmit data. Radio waves transmitted by the readers differ in frequency from low (LF) to high (HF) to ultra-high frequency (UHF). LF readers typically operate at 125KHz with a read range of around 10cm, as the frequency increased the read range also expands and so may the sensitivity to radio wave interference.

Tags also vary in specifications, passive tags may only be activated by a reader and may not instigate a transaction, where as semi-active and active tags may start communications. For this project the focus will be Near Field Communication(NFC). NFC is most notably used in transport ticketing and contactless payment management systems. NFC has successfully automated the ticketing and payment systems in many countries. Instead of manually punching in digits on a keypad, the user can touch or wave the NFC embedded card near the reader and a transaction can be processed. RFID has sped up various registration and transaction processes. NFC is inexpensive and durable, making it a primary candidate for tagging on mass produced food products that are frequently moved from different containers during the transportation process. Many mobile phones currently available on the market also are equipped with NFC readers further supporting the argument for the utilisation of NFC.

2.6.4 Data Storage Method

For any system the way in which data is stored and received strongly impacts system performance. Relational database management systems (RDBMS) provide a secure, reliable, tried and tested method of data management. Data is organised into schemas that are designed using normalisation techniques and queried using structured query language (SQL). Schemas are predefined and RDBMS comply with the ACID properties.

Although RDBMS provide a robust and secure data management solution, in some cases it's not the best solution. A common problem with RDBMS is the relationship ambiguity that the schemas can present even after the data is normalised. For example, in Figure 1 below, one might argue that that it is describing one person with two qualifications but another may interpret this as two separate entities which happen to have the same name but with different degrees. Schemas are predefined and constraints are applied to disallow invalid values. One way in which RDBMS ensures consistency is with the utilisation of strict locking protocols but heavy locking of tables and rows can case threads to wait and compromises availability to ensure consistency of data.[22]

sname	**course**
Tom	Computer Science
Tom	Biology

Table 1: Schema representing ambiguity in data representation.

The inability to articulate how the data is to be perceived and the need for BigData processing on massively distributed systems was a motivational factor for the NoSQL movement. NoSQL databases are flexible and schemaless, many compromise consistency with availability and operate on the principal of eventual consistency. NoSQL databases can vary from document, graph and key value storage methods. Data can be modelled dynamically and entities are in charge of their own attributes.

For this project high availability of the recourse takes precedence over consistency. Flexibility is needed when modelling the data as the system may need to respond to change rapidly, thus, a NoSQL data storage solution is fitting for the solution.

2.6.5 Distributed Authentication & Access Control

Different security levels and access restrictions must be enforced depending on the user to avoid data corruption and data breach. For cloud services in particular, where each account is billed on a pay-as-you go basis, if resources

and credential are not regulated appropriately the account owner can amass in an unexpectedly large bill. AWS SDK provides an access control service that can be implemented using an API. Different roles and policies may be assigned to users, granting granular level permissions.

3 Design

The first part of this section will present the system requirements and use cases for the application. The second part will discuss the design decisions driven by the analysis of technologies and existing systems made in the previous chapter.

3.1 Requirements

Information gathered in the analysis stage has brought to light the following requirements. Requirements will help understand and organise the core functionality of the application and eliminate the risk of diverting from the main objective.

3.1.1 Automated Product Logging Using RFID

The logging of products must be automated using RFID technology. The RFID interrogator is to be embedded in a fridge were it must listen for incoming and outgoing tagged products. The logging mechanism must provide accurate real-time stock keeping of products.

3.1.2 Offline Inventory Management

The RFID application must have a local cache that keeps stock of items even when an Internet connection is not present. When connection resumes the cache and the database must be consistent.

3.1.3 User Verification

Users must be verified from an Android device in order to use the service. Authentication must be made possible either by email or a popular third-party user authentication API. By keeping a record of the users, if the user happens to remove the application but later decide to reinstall the application. The previous account can be restored, allowing the user to skip time-consuming registration steps. User verification is also a necessity to avoid the wastage of cloud recourses.

3.1.4 Access Control to Resources

The RFID client and Android client must have different layers of access to resources. AWS credentials must not be hard coded into applications as the application is intended for mass distribution across multiple devices.

3.1.5 Data Organisation

Data must be available via wireless communication and must be constantly available to facilitate the displaying of real-time representation of the users inventory. Data must be organised and follow best practice data modelling techniques.

3.1.6 Multiple User Support

A single fridge may have multiple users. Application use must mirror real life scenarios where multiple users exist for one fridge.

3.1.7 Displaying Fridge Contents

The contents must be visible on an Android powered Smartphone. When connected to the Internet the data should be consistent with the current state of the fridge. Items will display the products in priority order with the earliest use-by-date at the head and the latest at the tail.

3.1.8 Offline Content Viewing

For situations where there is no Internet connection or sudden loss of connection the last view of the fridge before the connection is terminated must be visible. To enable this feature a local cache must be made that saves the intermediate data.

3.1.9 Engaging Users with Notifications

When a new item is added to the fridge it must push a notification to all android devices subscribing to a particular fridge. Ensuring all parties is up to date with the alterations.

3.1.10 User Interface

The user interface must be easy to navigate and follow Android best practices. Instantly recognisable icons and screen gestures will be utilised to ensure the user has a stress free introduction to the application.

3.2 Requirements Definition Report

A requirement definition report is provided to support the development of the new system. Here the requirements are organised into categories following the UML recommended presentation as described in the book, 'System Analysis and Design with UML' by Tegarden and Wixom. (Suggestion was taken from the Information Systems module as part of the MSc Computer Science.)

3.2.1 Non-Functional Requirements

Operational Requirements

- The mobile element should be able to operate in an Android environment.
- The system should be portable and accessible from a mobile phone.
- The system should persist and read data from a database.
- The system should persist and read data from the devices internal storage.
- The system should scan and register RFID tags.

Performance Requirements

- The user interface must constantly be active.
- The system should be available for use 24 hours per day, 365 days per year.
- Any communication between the clients and the server must not exceed 2 seconds.
- The system should be durable and preserve data integrity.

Security Requirements

- Only authorised users can use the system.
- Users may only see the content of the subscribed fridge.

3.2.2 Functional Requirements

Multi-user Support

- Any user should be able to join an existing fridge by entering the unique code provided to the primary owner of the fridge.

Information Viewing

- User should be able to view a list of their stocked items.
- User should be able to view the expiry date of the items.
- User should be able to see the quantity for each item.
- User should be notified when an item is added to their inventory.

- Users may only see the content of the subscribed fridge.

User Registration

- Users should be able to login to the system.
- User should be able to logout by pressing the logout button.

3.3 Use Case Reports

3.3.1 User Login

Use Case	Initial start up of Android Application.
Primary Actor	A member of the public who installs the application.
Description	A ew user opens the application for the first time.
Assumptions	1.The application is installed and running on a device running the Android environment. 2. The device is connected to a wireless Internet connection. 3. The RFID client is successfully running.
Success Scenario	1. A user selects the application icon on the home screen. 2. The application does not recognise the user as previously registered and will display the login page. 3. The user will enter valid credentials and grant permissions for the application to access user information. 4. The user will then be directed to a page to enter a unique Fridge code. The user will enter a valid code. 5.The user will successfully subscribe to the and will be presented with the contents of the fridge in order of priority with the latest expiry date at the top.
Alternative Flows	If the user has previously signed up, the sign up process is skipped and the user is presented with the main page. If the user does not have and valid credentials they will not be able to proceed.
Non-Behavioural Requirements	User details must be stored in the database. Only users with a valid fridge ID is able to access the contents of a particular fridge.
Issues	n/a

3.3.2 User Logout

Use Case	User logs out of the application.
Primary Actor	A member of the public who installs the application.
Description	A logged in user decides to log out of the application.
Assumptions	1.The application is installed and successfully running in a Java runtime environment. 2. The component is connected to the Internet. 3.The fridge's unique code persists in the database. 4.User has successfully logged into the application.
Success Scenario	1. A user navigates to the settings menu and presses the logout button. 2 The user is logged out and returned to the login page.
Alternative Flows	n/a
Non-Behavioural Requirements	User details must still be saved so if the user decided to log back in the sign up process is ignored. The user credentials are stored internally until the device case is cleared or the application is deleted.
Issues	n/a

3.3.3 Add item into the fridge.

Use Case	User loads the fridge with items.
Primary Actor	A member of the public who installs the application.
Description	A user who has a RFID embedded fridge stocks the fridge with tagged items.
Assumptions	1.The application is installed and successfully running in a Java runtime environment. 2. The component is connected to the Internet. 3.The fridge's unique code persists in the database. 4.All items that are loaded into the fridge are tagged with RFID tags.
Success Scenario	1. A user opens the fridge and adds the products. 2.The tag is registered and interrogated for it's ID. 3.The item ID is stored in the user's database. 4. When an item is removed the RFID reader will read and the application ID will be removed.
Alternative Flows	If Internet is not present the item ID is persisted locally.
Non-Behavioural Requirements	If there is no Internet connection the application must save the tag ID's locally. Once the connection is present the data must be saved to the database.
Issues	n/a

3.3.4 Refresh Contents.

Use Case	User can refresh content view upon request.
Primary Actor	A member of the public who installs the application.
Description	Using the pull down gesture the contents of the fridge can be refreshed and displayed.
Assumptions	1.The application is installed and running on a device running the Android environment. 2. The device is connected to a wireless Internet connection. 3. The RFID client is successfully running. 4.The user has successfully signed in.
Success Scenario	1. User pulls the list down and releases. 2. If any changes have occurred in the fridge the list will be updated displaying the latest informattion.
Alternative Flows	n/a
Non-Behavioural Requirements	The application must display data as close to real time as possible. Any alterations to the database will trigger a notification to the user.
Issues	n/a

3.4 Business Process Modelling with Activity Diagrams

3.4.1 User Login

Figure 1 represents the potential actions a user may take within the context of the application. The system will request the user for information such as a username and email. The system will then determine whether the user is a new user or and existing user. If the user has previously used the application the fridge registration is skipped. If the user is a new user they will be presented with an activity where they will be asked to enter a unique fridge ID, assumption is made that the 'purchased' appliance is assigned this unique code. Once the code is verified a new user is made in the database.

3.5 Program Design

This section defines the programs that is to be written and the methods that will be utilised to satisfy the requirements.

3.5.1 Distributed Authentication & Access Control

The authentication process will be completed through the Android application. A single AWS account may be used for multiple applications, and so it is important to authenticate users and only grant access to the relevant resources. AWS Cognito and Identity and Access Management (IAM) tool

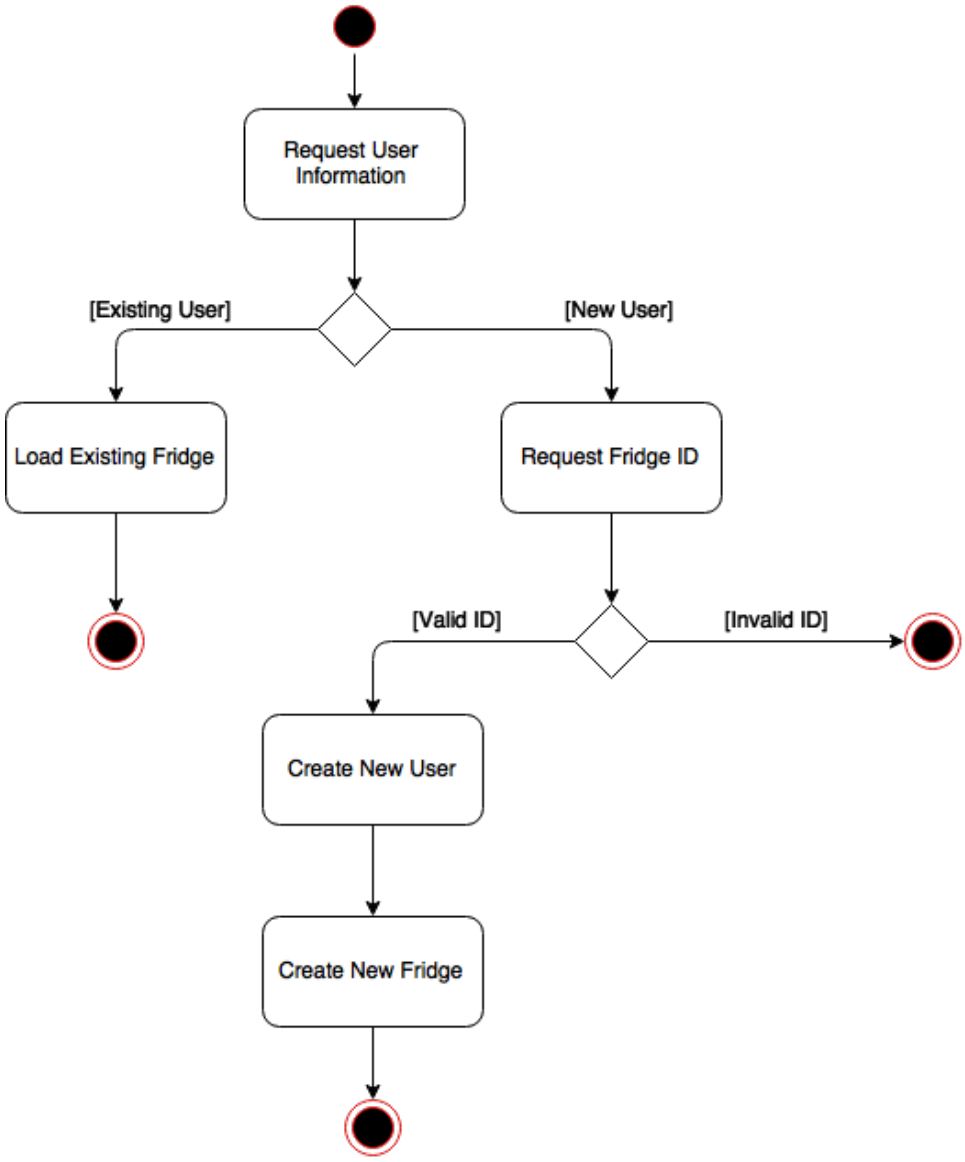

Figure 1: Activity Diagram for the user login flow.

kit provides an API and console to mange users and delegate role policies to restrict resource access.

3.5.2 User Authentication

A user must be authenticated to avoid unnecessary AWS resource consumption. Authentication is to be made possible via email or a third-party service such as Facebook, Google or Twitter. Majority of users will have previously signed up to popular social networking sites and it has become a popular authentication pattern to utilise these services rather than to provide a custom sign up flow. By utilising existing API's repetitive user behaviour can me avoided. Figure 2 represents the inner process when using third party authentication with AWS.

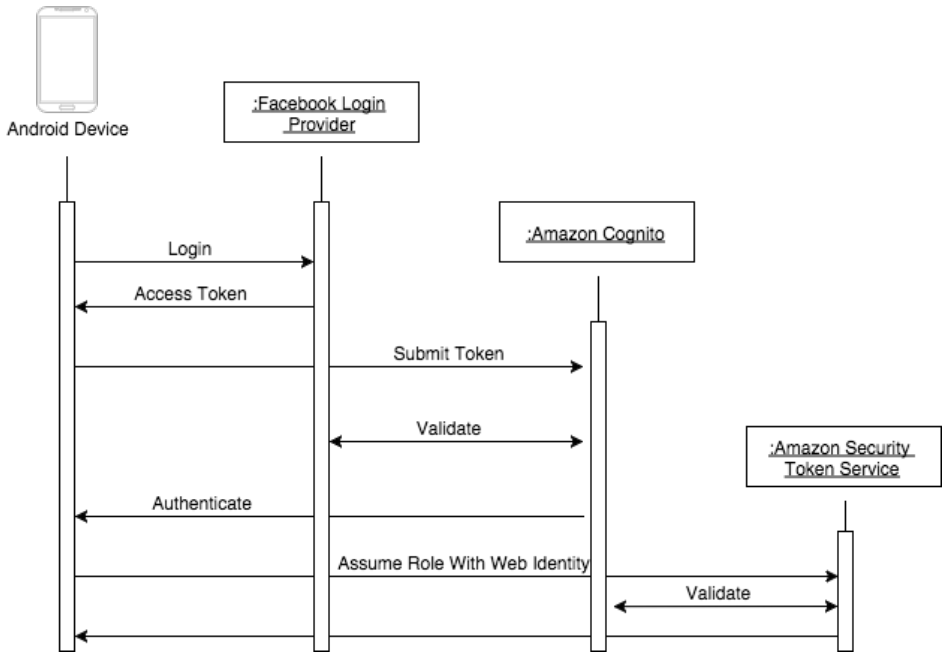

Figure 2: Sequence diagram for the user authentication process.

3.5.3 Delay Tolerant Networking

The system must be able to keep a consistent state by utilising a local cache on both the Android device and the fridge application even in an event where Internet connection does not exist. Heterogeneous networks with multiple dependencies and the reliance on wireless Internet access must be fault tolerant to loss of network connections.

3.5.4 Database & File Specification

AWS DynamoDB Data will be stored in Amazon's key-value NoSQL database DynamoDB. DynamoDB is a managed distributed database built on Solid State Drives (SSD) for efficient, low latency response time. Data is free to store and Amazon will only charge for throughput used. As user demand grows DynamoDB is able to scale seamlessly and traffic is managed and balanced automatically and is backed by a robust fault recovery mechanism. Data is replicated across multiple nodes to ensure that even in a situation where a cluster is damaged, another is always available.

Internal Device Storage The Android application will utilise the devices internal storage to localise retrieved data. The SharedPreference class is a lightweight method of storing data in a key-value pair. For complex objects on both the Android application and RFID application the Java I/O library will be used to serialised and persist date.

3.5.5 Interface Design

Over the years Smartphone users have become accustom to certain navigation patterns. Keeping with popular gestures and icon designs provides the user with a stress free user experience. For example users instinctively recognise certain icons to give a particular functionality such as the 'settings' button' depicted by three vertically aligned dots. It would be ineffective to redesign a settings icon and make the user learn a new pattern of navigation and can even cause the user to reject the application. A common pattern for refreshing a list is to pull down the screen where a loading icon is displayed. This feature is to be implemented for refreshing the screen. The interface must constantly be responsive and provide feedback to the user to avoid the user thinking the application has crashed.

3.5.6 RFID Client

The RFID client will be written using the Java development kit version 8, using the IntelliJ Integrated development Environment (IDE). The AWS SDK will also be used to access DynamoDB resources. Tag ideatefiers will be read and persisted in the relevant fridge where the android application will receive the data. The hardware will be connected to an Internet enabled computer with the Java Runtime Environment installed. A console will display a log to communicate any errors that may occur. Figure 3 is a sequence diagram visulaises the item logging process.

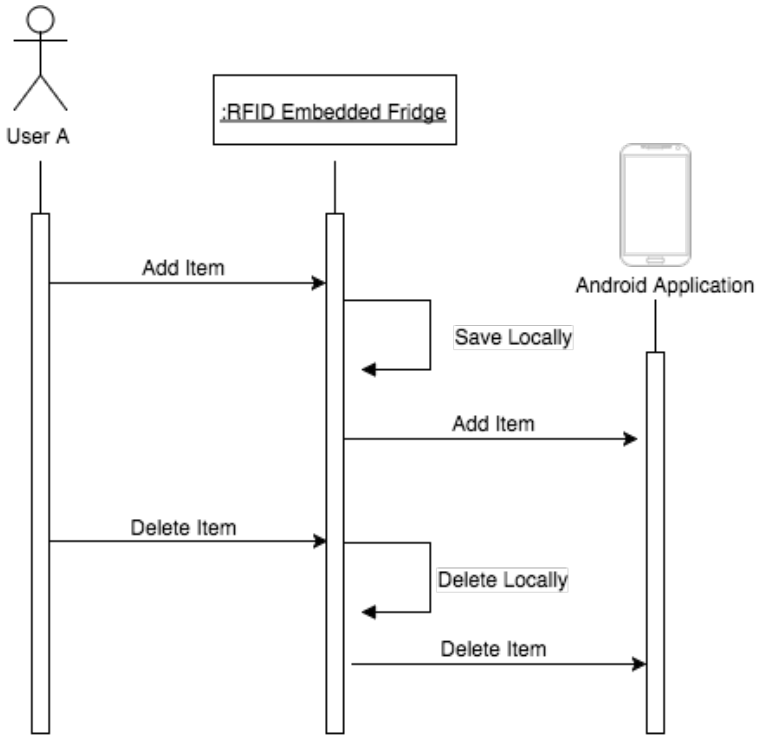

Figure 3: Sequence diagram for the item information persistance flow.

3.5.7 Android Client

The Android application will be written using Java and utilise the Android SDK version 4.4 to target as many devices as possible. The application will be developed using Android Studio IDE and the AWS Mobile SDK. Implementation will take into consideration Android's unique activity lifecycle methods to avoid undesired behaviour. Android development and object oriented best practices will be applied to maximise code reusability and minimise decoupling of classed to avoid strong dependencies between classes and activities.

4 Implementation

4.1 RFID Java Client Development

4.1.1 Hardware

The selected interrogator is a 125khz Mifare technology RFID reader. It has a short read range of up to ten centimetres. The tag type is an ISO/IEC 14443 standard, passive low frequency RFID embedded stickers. Figure 4 is an image of the hardware used.

Figure 4: Image of NFC tags and a RFID reader

4.1.2 Application

The java client continuously polls for incoming tags. When a tag is read the unique code is extracted and queried against the database. Once the item is successfully identified the product is added to the users table. Where the Android client may retrieve the data, thus automating the item registration process. When the same item is read the reader will acknowledge the removal of the item and the product is removed from the table. In the even of a network failure a local cache will store the item identifiers, details on the implementation can be found in the section 4.4.

4.2 Android Development

4.2.1 Android Activity & Life Cycle Methods

Android uses a component called an activity to depict an action a user may accomplish within the application. When an activity class is extended some key methods listed below may be overridden and custom behaviour can be

implemented. The developer has no control over when the methods are called and so given a situation the implementations can vary. Figure 5 taken from the android developers website, shows the life cycle of an activity and the methods that may be called. Careful management and implantation of the methods is needed to avoid unexpected behaviour and system crashes.

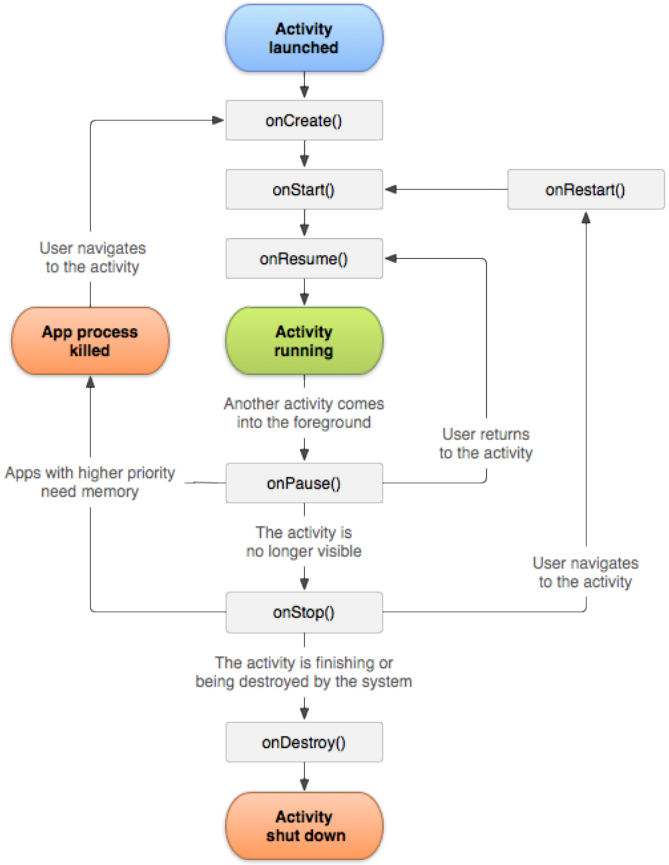

Figure 5: Life cycle of an Android activity.

Activities may be launched as an intent. When activities are launched they are places on top of a stack where the history of the activity is kept, this is called the back stack. Upon the user pressing the back button the current activity is popped off the top of the stack and they may return to the previously running activity.

4.2.2 Login Activity

The login activity is responsible for the user authentication process. Graphical elements such as buttons and text boxes instantly translate the needed actions the user is to take.

Buttons The LoginActivity class implements the `onClickListenter()` interface which enforces the implementation of the `onClick()` method. In this method a switch statement is used to determine which button was pressed. The switch statement checks for the interacted resource ID and executed the necessary block . For example, `R.id.facebook_login_button` corresponds to the Facebook login button.

When the Facebook button is pressed the user must go through Facebook's authentication process, if the user has not previously signed up. If the user is a retuning user then the login process is avoided and the user is directed straight to the main activity.

When the user enters their credentials, a request for an access token is made to Facebook as shown on Figure.2 If the user is a registered member of Facebook a new activity is launched requesting that the user grant the application permission to extract the requested details from their account. In this case the user's profile and email address is needed to use the application, as shown in Figure 6. If the user accepts, a graph JSON object is retuned containing the user's details and the next activity is launched.

```
case R.id.facebook_login_button:
    Log.d("LoginActivity", "facebook button pressed");

    FacebookSdk.sdkInitialize(getApplicationContext());
    callbackManager = CallbackManager.Factory.create();
    facebookloginButton.setReadPermissions("public_profile", "email");
    facebookloginButton.registerCallback(callbackManager, new FacebookCallback<LoginResult>() {
        @Override
        public void onSuccess(LoginResult loginResult) {
            facebookloginButton.performClick();

            SharedPreferences sharedPreferences = getSharedPreferences("LoginActivity", Context.MODE_PRIVATE);
            SharedPreferences.Editor editor = sharedPreferences.edit();
            editor.putBoolean("user_logged_in", true);
            editor.apply();

            FacebookUserInfoInitializer FacebookUserInfoInitializer = new FacebookUserInfoInitializer();
            FacebookUserInfoInitializer.execute(loginResult.getAccessToken());
            Log.d(Constants.LOGIN_ACTIVITY, "login success, token is: " + loginResult.getAccessToken());
        }
        @Override
        public void onCancel() {
            Toast.makeText(LoginActivity.this, "Facebook Login Cancelled", Toast.LENGTH_LONG).show();
        }
        @Override
        public void onError(FacebookException e) {
            Log.e("Facebook", e.getMessage());
            Toast.makeText(LoginActivity.this, "Facebook Login Error", Toast.LENGTH_LONG).show();
        }
    });
```

Figure 6: onClick() method implementation.

If the user is invalid the error will display a toast message, a light-weight message displaying process, communicating the failure to the user. The user will stay on the login page and will not be able progress until valid credentials are provided.

Once the user has been authenticated this state is preserved to avoid the

user needing to go through the signup flow each time the application is closed. Only when the logout button situated in the settings is explicitly pressed the user is logged out and the next time the user launches the application the sign up activity is launched again.

Back Stack Management When launching a new Intent, flags may be added to manage the back stack. The history of the login page should not be kept on the stack. This is to avoid the user navigating back to the activity holding sensitive information such as user name and password. Also if the Back Stack is not managed, a trail of activities become visible when exiting the application and is perceived as bad practice and unprofessional by the android develops guide.[17] Activities that the user will never return to should be disposed of and eliminated form the back stack. To ensure that the user cannot return to the login activity the history must be removed. By calling the Intents `addFlags()` method and passing it the parameters `Intent.FLAG_ACTIVITY_NO_HISTORY`, if this is set then the intent is not saved in the tasks history. Other flags may also be added to control the life-cycle of an intent. Figure 7 is an example taken from the `LoginActivity.java` class.

```
Intent fridgeRegisterIntent = new Intent(LoginActivity.this, EnterFridgeIdActivity.class);
fridgeRegisterIntent.addFlags(Intent.FLAG_ACTIVITY_NEW_TASK | Intent.FLAG_ACTIVITY_CLEAR_TOP |
        Intent.FLAG_ACTIVITY_NO_HISTORY);
```

Figure 7: Implementation of an Intent and the use of flags.

Graceful Application shutdown Commonly, if the user is on the main page and the back button is pressed the application will exit and return to either the device home screen or the previously running application on the back stack. But as the aim of login page is is to get the user to sign up, if the back button is pressed the application will ask for confirmation from the user that the user truly would like to exit. If so the user is prompted to press the back button again. This is achieved by overriding the `onBackPressed()` method. Figure 8 is the implementation of the `onBackPressed()` method taken from the `LogingActivity` class.

4.2.3 Fridge Code Request Activity

Users will be able to subscribe to a fridge using the `FridgeCodeRequestActivity`. This activity displays a text box. The user is prompted to enter a unique ID. If the ID exists in the database, the contents of the fridge will be retrieved

```
private Boolean exit = false;
@Override
public void onBackPressed() {
    if (exit) {
        finish();
        System.exit(0);
    } else {
        Toast.makeText(this, "Press back again to exit.", Toast.LENGTH_SHORT).show();
        exit = true;
        new Handler().postDelayed(() -> exit = false, 3 * 1000);
    }
}
```

Figure 8: Implementation of the onBackPressed method.

and displayed. If the user is an existing user of the application this activity will not be launched and they will be directed to the main activity.

User actions and feedback When the application requires the user to input or edit a field, the actions must be visually translated to the user. If no feedback is provided to the user they may mistake that the application has frozen or has encountered a system crash. Toast messages may be used to provide feedback to the user. Here when an empty field is submitted the user will be prompted of their error. However if the entered id is invalid another message will be displayed so the user is informed of the mistake with accurate detail. Figure 9 is a screen capture of the activity when the user attempts to pocceed without entering any characters.

4.2.4 Main Activity

The main activity is launched after the user is successfully authenticated. This activity will allow the user to remotely access the contents of the fridge. Below describes the construction process of the activity.

Fragments The Main Activity is made up of fragments. In Android development the use of fragments is encouraged to eliminate duplicate code and to create a transportable UI element. Fragments can be composed into activates allowing UI elements to be transported with ease and promote code reusability. Fragments belong inside an activity and also has a life-cycle of it's own, similar to an Activity. The fragment can be declared in two ways. Either statically using the activities layout file, or programmatically set in the `onCreate()` method.

Custom List Adapter and View Just by using the list adapter in the Android library would not be able to display all the elements needed such as

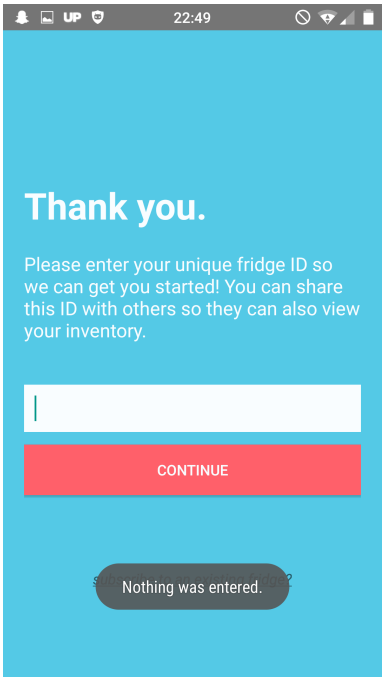

Figure 9: Screen capture of the toast message appearing when nothing is entered.

the expiry date, quantity, product name and image icon. Instead a custom adapter and list view were created. The `InventoryListAdapter` class extends an `ArrayAdapter`. The adapter class is responsible for dynamically setting the graphical elements and text fields in the list elements. As the list is populated the correct icons and text colours are assigned.

In the `onCreateView()` method in the `InventoryFragment.java` class the custom adapter is set to the list view. As the activity's `onCreate()` method is called and in turn the fragments are launched the information will display to the viewer.

The `compare()` method in the products class is overridden and in the adapter, the products are rearranged in priority order. The text will be displayed in red if the product is expiring today and green if there is no urgency.

Settings The settings icon and location have conformed to the Android recommended design practice. As shown in Figure.10 the settings contains the logout button where the user must navigate to in order to logout. To implement this the `onOptionsItemSelected()` must be overridden. Same as the `onClickListener()` method it uses the resource ID to find which field was selected.

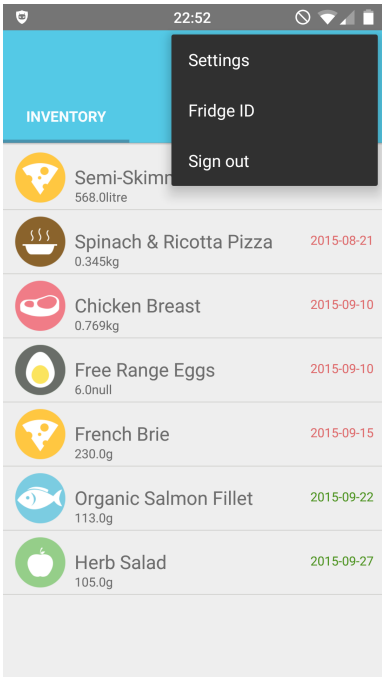

Figure 10: Screen capture of the settings tab.

4.2.5 Graphical User Interface Design & Resources

Interface elements can be created using XML elements that make up the UI. Elements that build the GUI are declared inside the layout files and can be styled within the XML document. Alternatively graphical elements can be set at runtime programmatically using Views and ViewGroups objects, where graphical elements can be extracted and edited within the code.

For each Activity and Fragment the application will style the interface using XML to keep the business logic and GUI logic separated, allowing the code to be more readable and maintainable. Figure 11 is the layout used for the Main Activity.

Resources such as graphical icons, logos, colours and Strings are stored in the values file within the resources. Strings are kept in a separate XML document so others may reference it and avoid hardcoding into the code and created a single access point where modifications can be made. Android operates world wide so this approach can also facilitate the

4.2.6 Manifest File

All Android application has a manifest file. Activities, services, receivers and content providers must all be registered in this file in order for them to be utilised. Permission are also set here to enable privileged features such as

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    tools:context=".MainActivity"
    android:noHistory="true">

    <include
        android:id="@+id/tool_bar"
        layout="@layout/tool_bar"
        android:layout_height="wrap_content"
        android:layout_width="match_parent"
        />

    <com.milk.keimiokamoto.milk.SlidingTabLayout
        android:id="@+id/slidingTabs"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:elevation="2dp"
        android:background="@color/colourPrimary"/>

    <android.support.v4.view.ViewPager
        android:id="@+id/pager"
        android:layout_height="0dp"
        android:layout_width="match_parent"
        android:layout_weight="1">
    </android.support.v4.view.ViewPager>

</LinearLayout>
```

Figure 11: Static layout initialisation using XML

access to the Internet and cross application data sharing. Application name and screen orientation and even setting the start up activity is declared in this document.

4.2.7 Notifications

Android uses notification alerts to communicate with the user. Notifications can be triggered when certain events occur. When a new item is added to the fridge an alert is triggered notifying the user of the change in state. This way even without the user having the application open they are kept informed of the state reducing the likelihood of the user unknowingly purchasing the same product. Figure 12 is the screen a user will see when the event is triggered.

4.2.8 Threads & Concurrency Management

Application component by default run in the same thread called the 'main thread'. If processed that take a long time to complete are ran on the main thread it will cause the UI to become unresponsive and can even block other features of the application. An unresponsive UI can lead the user to believe that the application has encountered and error. Android best practices sug-

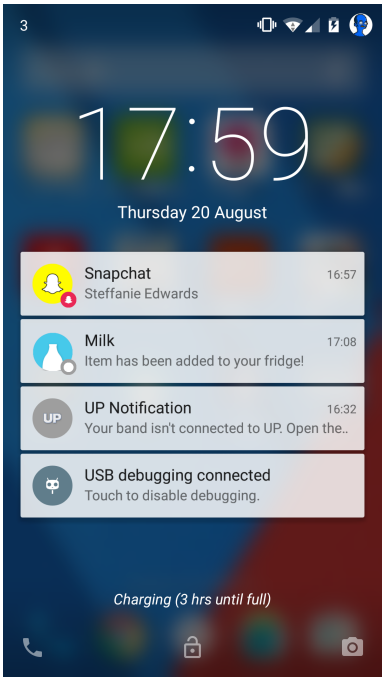

Figure 12: Notification displaying a message.

gest that network and I/O operations are performed in a separate thread to the main thread.

AsyncTask Android provides a thread management class that preforms asynchronous tasks. Using the `AsyncTask` class the management is performed automatically by the Android environment. When extending AsyncTask the `doInBackground()` method is implemented. Once this method has completed the `onPostExecute()` method is called, upon completion any information processed can be displayed in the UI. This class can be executed using the `execute()` method from the UI thread. Figure 13 shows the asynchronous network request using the Facebook API.

Future Task, Runnable and Handlers At times finer control is needed for threads. In some circumstances race conditions can arise, where the main thread attempts to process data that is dependant on a previously executed AsyncTask's completion. Blocking and forcing the main thread to wait is a bad idea as it will cause the UI to become unresponsive. FutureTask is a cancellable asynchronous operation and can also ensures that the results are only retrieved if the computation has completed. It implements the `Runnable` interface so it may be submitted to an executor. Figure 14 is an example implementation of `FutureTask` that will only retrieve the data

```
//Retrieves the user's information using facebook graph api. Variables initialized for
//persisting to dynampdb.
private class FacebookUserInfoInitializer extends AsyncTask<AccessToken, Void, AccessToken> {
    @Override
    protected AccessToken doInBackground(AccessToken... accessTokens) {
        FacebookSdk.sdkInitialize(getApplicationContext());
        GraphRequest request = GraphRequest.newMeRequest(
                accessTokens[0],
                (object, response) -> {
                    try {
                        uid = (String) object.opt("id");
                        name = (String) object.opt("name");
                        email = (String) object.opt("email");
                    } catch (NullPointerException e) {
                        Log.e(Constants.LOGIN_ACTIVITY, e.getMessage());
                    }
                    Log.d("LoginActivity", response.toString());
                });
        Bundle parameters = new Bundle();
        parameters.putString("fields", "id, name, email");
        request.setParameters(parameters);
        request.executeAndWait();

        return accessTokens[0];
    }
```

Figure 13: Implementation of AsyncTask

once the query to the database has completed. Stream of data and execute

```
private List<String> getContentsIdOf(String fridgeID) throws ExecutionException, InterruptedException {
    FutureTask<List<Product>> task = new FutureTask(new ContentsIdRetriever(fridgeID, getApplicationContext()));
    ExecutorService executorService = Executors.newSingleThreadExecutor();
    executorService.submit(task);

    Fridge fridge = (Fridge) task.get();
    List<String> contentsID = fridge.getContents();

    executorService.shutdown();
    return contentsID;
}
```

Figure 14: Implementation of FutureTask

4.2.9 Service and BroadcastReceiver

Android recommends the use of a service for long running operations. Unlike activities, services do not have a interface; services are designed to run in the background and can continue, even if the user happens to switch to another application. For the system to display data as the user loads the fridge, a service was created to periodically poll the database. This operation is able to continue even when the application is not in the foreground. `LoadFridgeContentService` extends the service class. For the service to function the first thing an activity must do is bind to the service. The `onBind()` will return an `IBinder` this way the client may communicate with this service. When `startService` is called by the client the `onStartCommand()`, this method will continue to run indefinitely. The `ScheduleReceiver` class extended the `BroardcastReceiver` class. The

`onReceive()` method is overridden where an alarm service is implemented and set to poll every thirty seconds. Figure 15 is the implementation of the ScheduleReceiver.

```
public class ScheduleReceiver extends BroadcastReceiver {
    // restart service every 30 seconds
    private static final long REPEAT_TIME = 1000 * 30;

    @Override
    public void onReceive(Context context, Intent intent) {

        AlarmManager service = (AlarmManager) context.getSystemService(Context.ALARM_SERVICE);

        Intent StartServiceIntent = new Intent(context, StartServiceReceiver.class);
        PendingIntent pending = PendingIntent.getBroadcast(context, 0, StartServiceIntent, PendingIntent.FLAG_CANCEL_CURRENT);

        Calendar cal = Calendar.getInstance();
        cal.add(Calendar.SECOND, 30);
        service.setInexactRepeating(AlarmManager.RTC_WAKEUP, cal.getTimeInMillis(), REPEAT_TIME, pending);
    }
}
```

Figure 15: Implementation of ScheduleReceiver

4.2.10 Swipe Refresh Layout Function

An alternative to waiting thirty seconds for the service to refresh automatically, the user may manually pull the list view down where a loading icon will appear, once it has completed any new data will be displayed. The `SwipeRefreshLayout` is implemented in the `InventoryFragmet` class. `setOnRefreshListener()` is set and listens for the user to initiate the refresh. This also a common smartphone user pattern for and a nessary feature for any list view. Figure 16

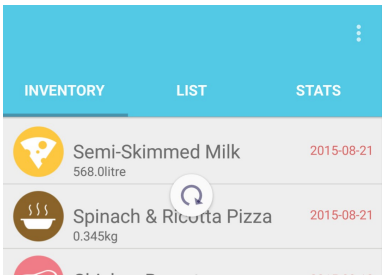

Figure 16: Loading wheel for the swipe refresh layout view

4.2.11 Error Handling

Logging The Android library has a logging class to enable the organisation of errors. Log messages aids the debugging workflow, if the log is for debugging purposes the `Log.d` will be used to explain the process, similarly,

`Log.e` is for errors, `Log.i` is for information, `Log.v` is for verbose messaging descriptions, `Log.w` is a warning message and `Log.wtf` is 'What a terrible failure' message to signify situations that should not happen. [17]

4.3 Distributed Authentication and Access Control

Access control to resources is imperative to avoid security breeches and data loss. When using AWS mobile SDK, AWS enforces the use of Cognito for mobile identity management. Cognito uses Identity Pools to manage the different layers of user privileges and Identity Access Management (IAM) roles.

4.3.1 Identity Pool

An identity pool is specific to the application. As a single AWS account is used by many different applications the segmentation of users can help organise the subscribing users and facilitate the allocation of correct recourses to the user.

By using the Cognito client developers can avoid hardcoding credentials into the application?s source code. This is particularly important for programs intended for mass distribution.

The image below is a method that initialises the Cognito client using the activities context, Identity pool id of the Amazon Resource Name (ARN) and the region code the resource resides on, in this case it accessing the servers in Ireland.

```
private AwsCredentialProvider(Context context) {
    credentials = new CognitoCachingCredentialsProvider(context,
            AwsConfidential.COGNITO_IDENTITY_POOL_ID,
            Regions.EU_WEST_1);
}
```

Figure 17: Initialisation of Cognito Client

4.3.2 Identity Access Management Role

IAM Roles are assigned in identity pools. IAM roles allow fine grain access control to AWS resources such as S3 and DynamoDB. The Cognito client utilises the IAM role?s ARN to specify the access rights of each role. By default AWS creates an authenticated and unauthenticated access role for an identity pool. Authenticated users may only access the specified resources in the policy if the requirements are met in the statement. Unauthenticated roles are for users that may want to try the application before committing to

signing up. Internally, temporary credentials are created for an unauthenticated user granting them temporary access to resources.

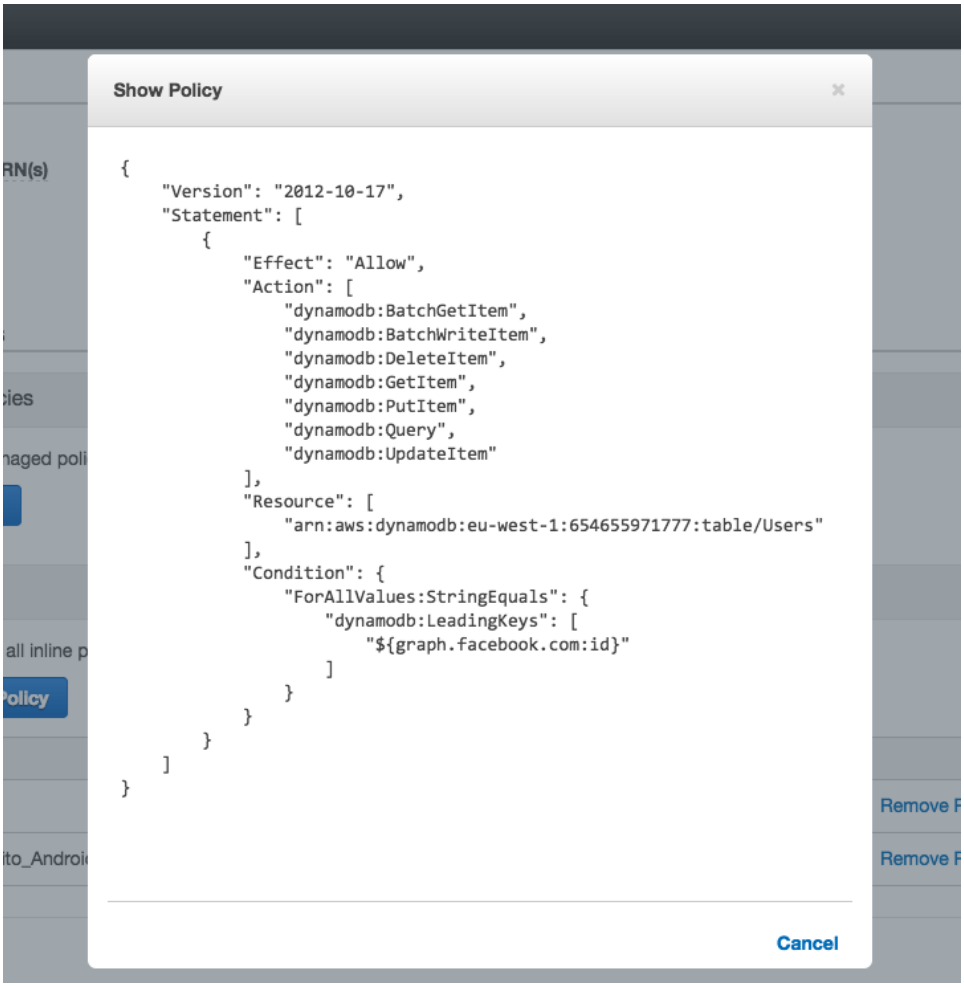

Figure 18: Policy for Authenticated Users

Security Policy Policy statements are written in Java Script Object Notation (JSON). The statement is split into three sections, actions, resource and effect. The action block specifies the actions that are allowed on the service. For example the policy in Figure 18 lists the actions that may be performed on a service, in this case DynamoDB. Actions that are not explicitly specified will be rejected. Resource block specify the AWS recourse that the policy is allowing access to, a resources is indicated by the ARN. The effect block specifies the result after the user requests access to the service. By default the effect is set to ?deny? so this must explicitly set. Additionally,

a condition block may be set to further restrict the type of user gaining access. In the example below only users associated with Facebook are granted access.

4.4 Delay Tolerant Networking

Local storage is needed to facilitate the event of a loss of Internet connection. Local cache is needed to persist the latest data retrieved from the database so the latest data is always available and can manage the dependency on the database. As the service class polls for data it is written to internal memory, if the Internet connection is disturbed the application will still display the last received data until the network connection resumes.

4.5 Data Management

4.5.1 DynamoDB

The first step was to create the products table. Tables may be initialised programmatically within the application using the AWS DynamoDB API or within the AWS console. Throughput is provisioned at table creation time using the console. This figure can be altered if the application needs more resources. The console also estimates monthly costs and displays statistical data for a given table. Figure 19 is an image of the console, at the bottom displays the analysis report for the products table.

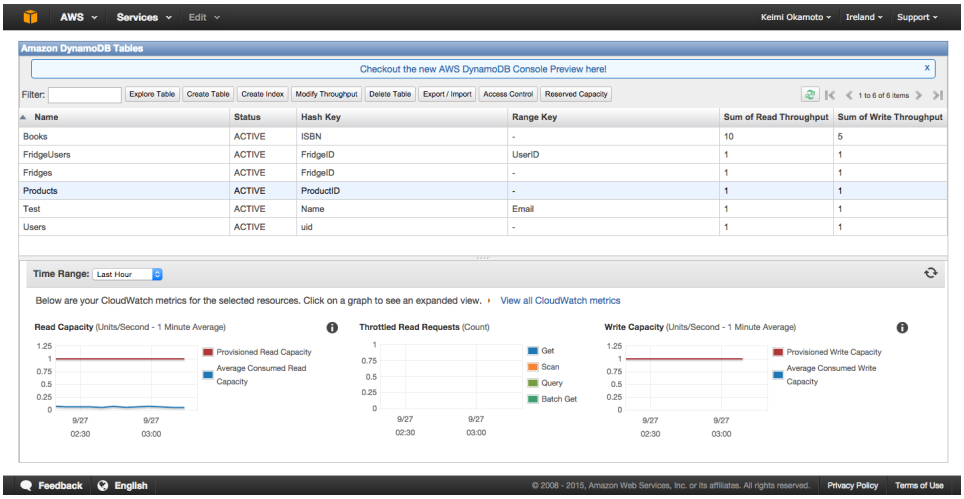

Figure 19: Policy for Authenticated Users

Data modelling Unlike relational databases DynamoDB does not require tables to have a predefined schema. The only requirement is for the primary

key type to be specified, any number of attributes may exist providing the size of a single item does not exceed 400kb. Each attribute can be a single name value pair or the value can be a complex data structure such as Maps and Lists. The value is a set data structure, meaning duplicate values are not allowed.

4.5.2 Primary Key & Range

The primary key consists of a hash value. DynamoDB uses this unique attribute to identify items in the table. Additionally a range key may also be defined when creating the table. The range key acts as an index, when the range key is set efficient retrieval of data can be made. For the products table the primary key was specifies as a string value using the NFC tag's unique identifier. The user fridge table's primary key is also a string value and the corresponding list holds all the product tag IDs.

Object Mapper Once the connection is established using the Cognito client the DynamoDB client can be initialised using the returned credentials. Classes that are intended to be persisted in the tables must be decorated with DynamoDB anotations. The annotated classes are translated into JSON format and added to the specified table. By placing `DynamoDBTable(tableName="Fridge")` before the class declaration the object can be utilised by DymanoDB. The primary key field is also required to be notated with `@DynamoDBHashKey`. Any other field that DynamoDB would like to interact with must also be annotated with `DynamoDBAttribute`. Similarly range keys and secondary indexes are also needed.

Using the object mapper data can be stored and retrieved with ease. Figure 21 shows part of the implementation of the Products class with DynamoDB annotations and Figure 20 shows how the item can be edited in the AWS console.

4.5.3 Android Internal Storage

Shared preferences Shared preference is a lightweight key-value storage that can be accessed by components working in the same context. Firstly the mode of must be set to `Private` or `Public` for this particular use the data is set to `Private` then an editor is needed to write the data into storage.

Internal Storage For more complex objects androids internal storage can be used using the Java I/O library. The `Products` class must implement `Serializable` interface so the class may be serialised. Figure 22 is an implementation of the `InternalStorageManager`.

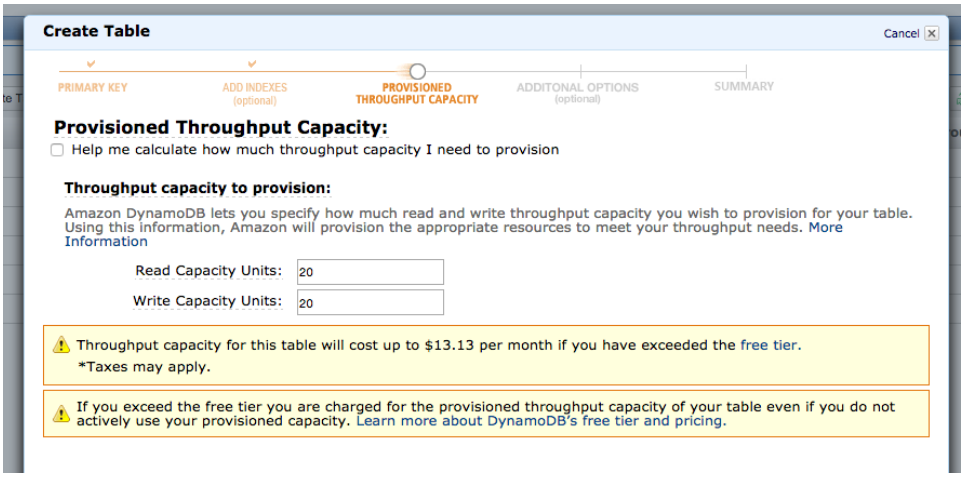

Figure 20: Products item structure in the DynamoDB console.

```java
@DynamoDBTable(tableName="Products")
public class Product implements Serializable, Comparator<Product>{
    private int allergenCode;

    private int foodCode;
    private boolean gm;
    private double netContent;
    private boolean organic;
    private String id;
    private String name;
    private String expiryDate;
    private String measurement;
    private String origin;
    private Date date;
    private List<String> feed;
    private List<Map<String, Double>> nutrition;

    @DynamoDBHashKey(attributeName="ProductID")
    public String getId() { return id; }

    public void setId(String id) { this.id = id; }

    @DynamoDBAttribute(attributeName="Name")
    public String getName() { return name; }

    public void setName(String name) { this.name = name; }

    @DynamoDBAttribute(attributeName="ExpiryDate")
    public String getExpiryDate() { return expiryDate; }

    @DynamoDBAttribute(attributeName="Measurment")
    public String getMeasurement() { return measurement; }
```

Figure 21: Implementation of the Products class with DynamoDB annotations.

```java
public class InternalStorageManager {

    private InternalStorageManager() {
    }

    public static void writeObject(Context context, String key, Object object) throws IOException {
        FileOutputStream fileOutputStream = context.openFileOutput(key, Context.MODE_PRIVATE);
        ObjectOutputStream objectOutputStream = new ObjectOutputStream(fileOutputStream);
        objectOutputStream.writeObject(object);
        objectOutputStream.close();
        fileOutputStream.close();
    }

    public static Object readObject(Context context, String key) throws IOException, ClassNotFoundException {
        FileInputStream fileInputStream = context.openFileInput(key);
        ObjectInputStream objectInputStream = new ObjectInputStream(fileInputStream);
        Object object = objectInputStream.readObject();
        return object;
    }

    public static boolean clearInternalStorage(Context context, String key) {
        File dir = context.getFilesDir();
        File file = new File(dir, key);
        return file.delete();
    }
}
```

Figure 22: Implementation of the InternalStorageManager.

5 Testing & Evaluation

This section will discuss the various testing methods used.

5.1 Unit Tests & Mocking

Unit tests provide a framework for testing the functionality of the code. Tests were written using JUnit testing framework and Mokito mocking frame work was used to create mock objects. Mock objects allow the customisation in behaviour of objects that are composed in the testing class. Figure 23 is a unit test for the `LocalFridgeManagerImpl` class.

```
@Before
public void buildUp() {
    fridge = mock(Fridge.class);
    localFridgeManager = new LocalFridgeManagerImpl(fridge);
    someProductId1 = "00001";
    someProductId2 = "00002";
    someProductId3 = "00003";
    contents = new ArrayList<>();
}


@Test
public void shouldBeAbleToAddItemToExistingContents() {
    contents.add(someProductId1);

    when(fridge.getContents()).thenReturn(contents);

    localFridgeManager.add(someProductId1);
    localFridgeManager.add(someProductId2);
    localFridgeManager.add(someProductId3);

    int expected = 4;
    int actual = contents.size();

    assertEquals(expected, actual);
}

@Test
public void shouldBeAbleToAddNewItemIntoContents() {
    when(fridge.getContents()).thenReturn(contents);

    localFridgeManager.add(someProductId1);

    String expected = someProductId1;
    String actual = contents.get(0);

    assertEquals(expected, actual);
    verify(fridge).getContents();
}
```

Figure 23: Implementation of a unit test for LocalFridgeManagerImpl.

6 Manual Testing

Over the course of development manual testing was carried out to ensure all the components were working. Log messages were used to separate the errors and debug messages. The database items were updated and removed to ensure that the correct items were displaying on the Android application.

6.1 Facebook Testing Platform

Facebook provides a test user generation service where a new test user can be created to use for manual testing when logging in using Facebook. The test user can be configured using the console, once test users sign up the results were checked against the AWS identity pool to make sure the user was correctly accessing the resources. Figure 24 is a screen capture of the Facebook console, the edit button may be used to change the users settings to simulate different senerios.

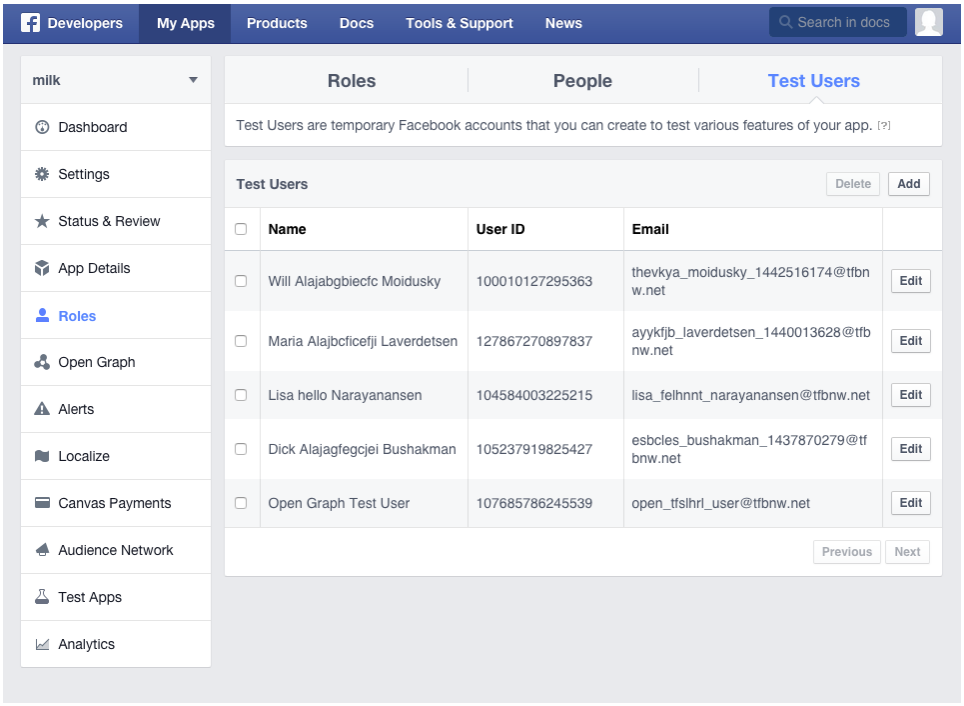

Figure 24: Test users generated by the Facebook console.

6.2 Testing the Service with Toast

As service have no interfaces a toast message was used to ensure the `onStartCommand()` was being called. The toast message appeared every thirty seconds and even when the application was closed it continued to run. Figure 25 is a screen capture of the toast message periodically appearing.

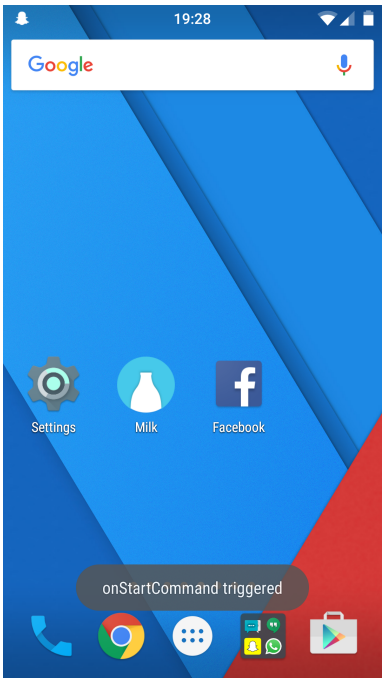

Figure 25: Toast message displayed on the home screen to signify a successful operation.

6.3 Test Reports

6.3.1 User Login

Case ID	Test Case Details	Expected Results	Pass/Fail	Remarks
01	An existing user presses the 'Sign in with Facebook' button.	The user should see their previously created account.	Pass	n/a
02	An new user presses the 'Sign in with Facebook' button.	The user should see the enter fridge code activity.	Pass	n/a
03	An new user presses the back button on the sign in page.	The user sees a message asking for confirmation.	Pass	n/a
04	An new user presses the back button twice on the sign in page,	The user is exited from the application.	Pass	n/a
05	An logged in user opens the application again.	The application displays the main actuvity.	Pass	n/a

6.3.2 User Logout

Case ID	Test Case Details	Expected Results	Pass/Fail	Remarks
01	An existing user presses the 'Sign out' button.	The user should be directed to the login activity.	Pass	n/a
02	An existing user presses the back button.	The user should exit the application.	Pass	n/a

6.3.3 Add Fridge Code

Case ID	Test Case Details	Expected Results	Pass/Fail	Remarks
01	A users enters a valid fridge code.	The user should be subscribed to the account with a corresponding code.	Pass	n/a
02	A users enters a invalid fridge code.	The user should be prompted of the error.	Pass	n/a
03	A users enters an empty string.	The user should be prompted of the error.	Pass	n/a

6.3.4 Adds Items to Fridge

Case ID	Test Case Details	Expected Results	Pass/Fail	Remarks
01	An user adds a tagged item to the fridge.	The user should see the item in the list from their Android device.	Pass	n/a
02	A user removes a tagger item from the fridge.	The user should not see the enter fridge code activity.	Pass	n/a
03	An new user presses the back button on the sign in page.	The user sees a message asking for confirmation.	Pass	n/a

6.3.5 Refresh Contents

Case ID	**Test Case Details**	**Expected Results**	Pass/Fail	Remarks
01	A users pulls the list view down.	The user should see a loading icon.	Pass	n/a
02	A users pulls the list view down and releases.	The user will see any new items added to the fridge.	Pass	n/a

7 Review & Conclusion

For the final chapter critically evaluates the final product and discusses the possible areas of improvement that can be made. Additionally security concerns over the Internet of Things (IOT) movement is discussed.

7.1 Item Weight and Quantity

With the use of RFID the application has successfully automated the logging of the items but the difficulty arose when the quantity of the product was needed. For example, a carton of milk may almost be empty and still have a small amount left. This is not enough for the user and they must be notified, however the fridge is still communicating to the user that milk is present in the fridge. The system is not able to keep track of the mass of the produce. In the past products such as 'Milkey Weight' but ChillHub were built to solve this problem. An item is places on top of a weight measuring surface connected via USB cable. But this would result in every product having a delegated storage location which is highly inconvenient.[24]

7.2 Security Concerns

There has been growing security concern over highly connected smart objects. Recently a test was carried out were an internet enabled vehicle was hacked into and the engine was disabled whilst speeding down a highway. [25] This demonstrated the ease in which hackers are able to tamper with the objects, any object with an IP address is vulnerable to malicious attacks and no matter how secure we try to make it attackers will always exist and is impossible to fully guard against.

7.3 Enhancements

7.3.1 Features

Statistics and shopping list In addition to the inventory list, a statistics report may be presented to the user detailing the monetary savings made and the environmental impact the saving have had to encourage the user. Shopping list generator could be created where suggestions can be made to the user based on past behaviour and purchase patterns.

7.3.2 Technology

Currently a service is implemented on the Android application. The service is set to poll the database every thirty minutes. Instead a trigger can be implemented on the database layer where any alterations to a table triggers a network operation. As the user base grown polling ever thirty seconds for every fridge would be highly ineffective.

7.4 Review

Overall the project has successfully managed to automate the logging of items and the inventory can be displayed to the user in any location via a smartphone. The project has been a highly educational experience where unexpected hurdles were met. Some challenging cases arose where previously taught skills from the MSc Computer Science course were revises and applied. To conclude the project has provided a better understanding of software development cycle and confidence in experimenting with new technologies.

References

[1] W. Carr, E.Downing, *"Food Waste"*, Science and Environment Section, House of Commons, 2 December 2014

[2] European Union Committee, *"Counting the Cost of Food Waste: EU Food Waste Prevention"*, London : The Stationery Office Limited, House of Lords, 6 April 2014

[3] P.Whitehead *et al*, *"Estimates of waste in the food and drink supply chain"*, WRAP, July 2002.

[4] M. White, *"Food access and obesity."* Obesity reviews 8.s1 (2007): 99-107.

[5] Government Statistical Service, *"Uk Statistics on waste 2010-2012"*, Department for Environment, food and Rural Affairs, March 2015

[6] G.A Miller, *"Magical Number Seven, Plus or Minus Two: Some Limits on Our Capacity for Processing Information"*, vol. 63. Cambridge, MA: The Psychological Review, 1956.

[7] Susie Steiner, *"Smart fridge? Idiot fridge, more like"*, The Guardian, January 11th 2014

[8] Graham-Rowe, Ella, Donna C. Jessop, and Paul Sparks.*"Identifying motivations and barriers to minimising household food waste."* Resources, Conservation and Recycling 84 (2014): 15-23.

[9] Environment, Food and Rural Affairs Committee, *"Food Contamination"*,vol. 1, authority of the House of Commons London: The Stationery Office Limited, 2013.

[10] Li, Lei, *et al. "Early warning indicators for monitoring the process failure of anaerobic digestion system of food waste."* Bioresource technology 171 (2014): 491-494.

[11] Handford, Caroline E., et al, *"Implications of nanotechnology for the agri-food industry: Opportunities, benefits and risks"*, Trends in Food Science & Technology 40.2 2014: 226-241.

bibitemgeorgeRG. Roussos, *"Networked RFID System, software & services"*, 1st edition, Springer, August 29, 2008

[12] R. Chen, *"Using RFID Technology in Food Produce Traceability"*,WSEAS Transaction on information science and applications, November 2008

[13] S. Piramuthu, *et al.*, *"RFID-generated traceability for contaminated product recall in perishable food supply networks."* European Journal of Operational Research 225.2 (2013): 253-262.?

[14] G. Roussos, *"Enabling RFID in Retail"*, IEEE Computer Society, 2006

[15] A. Sabbaghi, G. Vaidyanathan, *"Effectiveness and Efficiency of RFID technology in Supply Chain Management: Strategic values and Challenges"*, Journal of Theoretical and Applied Electronic Commerce Research, April 2008

[16] Food Standard Agency, *"Country of Origin Labelling Guidance"*, October 2008

[17] Android Developers,Dashboards.(n.d.). *"https://developer.android.com /about/dashboards/index.html"*, October 2015

[18] IOS Developer Library. (n.d.). *"https://developer.apple.com/library/ios /navigation/#sectionTopics&topicXcode"*, October 2015

[19] M. Fowler *"Writing The Agile Manifesto"*, October 2015

[20] C. Lam, *"Hadoop in Action"*, 1st edition, Manning Publications, 25th Dec. 2010

[21] J. Rosenberg, *"The Cloud at Your Service"*, Manning Publications; 1 edition, 2010

[22] Ramakrishnan, R., & Gehrke, J. *"Database Management Systems"*, Boston: McGraw-Hill,(3rd ed.); 2003

[23] Figure 1. *"http://developer.android.com/guide/components/activities.html"*, September 2015

[24] First Build *"http://market.firstbuild.com/products/milky-weigh"*, October 2015

[25] Tufekci, Z. *"Why 'Smart' Objects May Be a Dumb Idea"*, August 2015