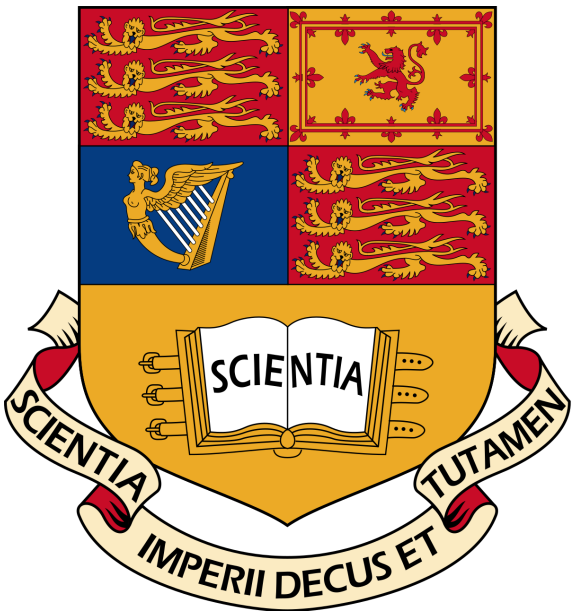
Imperial College London

Department of Electrical and Electronic Engineering

Final Year Project Report 2017



Project Title:	**An Internet-Of-Things Approach To Tackle The Problem Of Waste Management**
Student:	**Naga A. Garigiparthy**
CID:	**00838074**
Course:	**EIE4**
Project Supervisor:	**Professor Esther Rodriguez-Villegas**
Second Marker:	**Dr Pau Herrero-Viñas**

Abstract

The Internet-Of-Things (IOT) is the concept of connecting a wide range of devices to the internet, allowing them to communicate with each other and provide on-demand data that is in turn used for intelligent decision making. Food wastage is a problem that results in over 33% of food produced being thrown away every year, costing the average consumer over £200 annually.

In this project, a backend web service, database and iOS application are designed and implemented for use in an IOT system consisting of a smart bin and fridge. Based on studies concerning the motivations and barriers to engaging in the food waste issue, as well as an evaluation of existing solutions to the problem, this project focusses on delivering the following functionality: automated food and waste monitoring, recipe recommendations and personalized encouragement. This is done in an effort to improve consumers' level of engagement in the issue, improve their food management skills and save consumers money by helping them use a greater proportion of items bought. Following user testing with 9 users, the system was found to be well received by most participants however, it requires a longer period of testing to determine its effect on reducing food waste, and more explicit proof to the user that it is saving them money.

Acknowledgements

I would firstly like to thank Dr Esther Rodriguez-Villegas for her advice, guidance and enthusiasm throughout the project. I would also like to thank Wah Xian for all his hard work and working with me to develop both hardware and software components of the system. I am also very grateful to Sagar Patel and Brian Voong for their coding help. I would like to thank my family and friends for their support throughout my project.

Finally, I would like to thank my parents for their unceasing love, support and encouragement, not just during this project but throughout my time at Imperial College.

Contents

1 Introduction

1.1 Project Context

The concept of food waste has always existed. However, an increase in the governmental efforts to effectively tackle climate change and fight hunger has resulted in greater importance placed on tackling food waste [1] - thereby acknowledging it as a significant contributor to the two issues. Across the world, over 33% of the food produced every year is wasted, equating to approximately 1.3 billion tonnes [2]. This percentage varies between each country but is as large as almost 50% in the United States. From an environmental perspective, food waste makes up the largest contribution to landfill sites as shown in figure 1. As a result it is responsible for 3.3 Gigatonnes of CO2 emissions worldwide. This makes it the third largest emitter of green house gases, behind the USA and China [3]. Additionally, the decomposition of food material releases methane, a greenhouse gas with 25 times greater potency compared to carbon dioxide [4]. Food waste not only affects the environment, but also impacts consumers financially. In the UK alone, such wastage costs the average person around £200, and the average family £700 annually [5]. Based on an average weekly expenditure of £58.80 on food and non-alcoholic drinks, this equates to almost 12 weeks worth of groceries thrown away every year by the average family in the UK [6]. As shown in figure 2, there are six stages in the food supply chain from the point of farming and production to home and restaurant consumption. Whilst food is wasted throughout this supply chain, the study conducted by the Food and Agriculture Organisation of the United Nations [2] showed that wastage is greatest at the consumption stage in medium to high income countries, whereas low income countries suffer greater wastage in earlier stages of the supply chain.

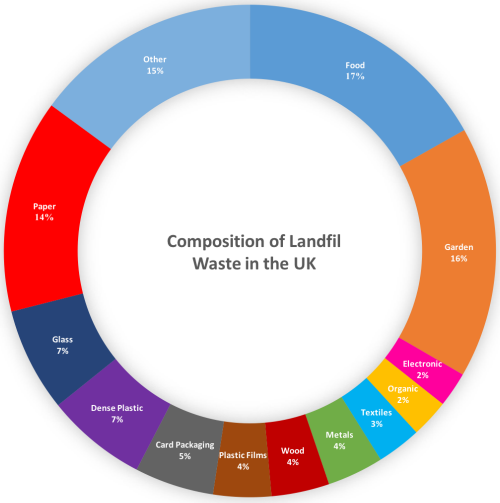


Figure 1: Composition of Landfill Waste In The UK [3]

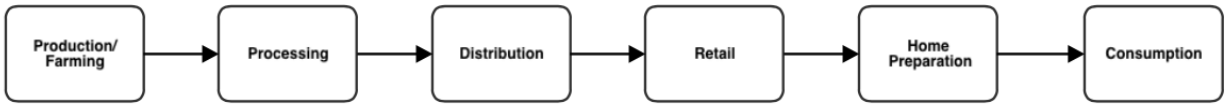


Figure 2: Six Stages of Food Supply Chain [7]

1.2 Project Scope

The main objective of this project is the design and implementation of software components that will be integrated as part of a greater encompassing internet-of-things (IOT) solution to the food waste problem. The software components comprise of both a backend web service and an iPhone application (app). The backend web service serves to collect, store and transfer data between a smart fridge, smart bin and mobile app. The role of the app is to help facilitate better food management skills through the automated monitoring of items stored at home and quantity of food wasted, as well as the recommendation of recipes to help aid usage of a greater proportion of food items bought. In addition, the app will serve to increase engagement with the underlying food waste problem through a well thought out and designed interface as well as features that help to unconsciously alter consumer food planning behaviour.

In this project, the automated monitoring of items stored at home will be limited to those stored in the fridge, placing a greater focus on helping users keep track of most perishable items. As a software focussed project being developed in parallel with a hardware focussed project, the functions that can be provided from the backend service are restricted by the types of data that can be obtained from the hardware sensors being used. During the course of the parallel project, there was no feasible means to install sensors inside the fridge environment. As a result, data such as the amount of an item remaining will not be collected. Identification of what items are wasted is another function not covered by this project, again due to the lack of a feasible hardware implementation in a food waste bin. The latter issue also prevented functionality for equating food wastage to a monetary value.

1.3 Report Overview

The aim of this report is to detail the design making and development behind the IOT centred solution to tackle the food waste problem. This current section introduced the problem and the consequences it poses. Chapter 2 provides a background insight into several areas: the motivations and barriers consumers face for engaging in the food waste issue, recommendation systems, user interface design and existing solutions. This helps to outline a set of requirements for what the solution developed in this project must achieve to improve upon prior efforts. Chapter 3 describes the proposed solution detailing exactly what the system will

do. Chapter 4 expands on chapter 3 describing the entire system architecture in detail, evaluating relevant technologies considered for implementation, as well as explaining other design decisions made. Chapters 5, 6 and 7 detail the implementation of the backend, user interface design and front end components. Chapters 8, 9 and 10 cover testing procedures, results and evaluation respectively. Finally, chapter 11 provides a summary of the project outcomes and limitations, as well as potential improvements that can be made in the future.

2 Background

Before formulating a solution, this section provides a range of background material behind the food waste issue as well as other important areas which may help during the design and implementation stages of the project. The background material covers the consumer mindset, recommendation, user interface design and existing solutions. Exploring these areas not only helps to understand why a new solution is necessary, but presents ideas which can be considered when formulating a concise set of requirements for the new solution.

2.1 Consumer Mindset

The consumer mindset can be viewed in terms of the motivations and the barriers to engaging in the issue. Based on an experiment conducted by Ella Graham-Rowe et al. [8], it was determined that there were several overarching motivations and barriers for consumers to engage in reducing their food waste.

2.1.1 Motivations

The main motivations for reducing food waste were concerns regarding waste in general and to do the 'right' thing. The general waste concerns were focussed towards how food waste equated to money wastage. This is a valid concern considering the fact that households on average still waste around £700 per year on average. Recent economic events have also resulted in significantly reduced income for some families, making money a much greater concern for households. There was also a concern for how food wastage resulted in wasted utility - in the sense that the food had not served its purpose. This was more prominent for meat items, where an animal had been killed for the sake of becoming a food source.

The second motivation of doing the 'right' thing has originated in different ways. Some people have felt that this view has been adopted as a result of societal pressure. This can be triggered from campaigns targeted at raising awareness of food waste, or from the awareness of other people in the world not being as privileged to have access to food. This view is also one that has simply been adopted from ones upbringing.

This is common considering a family's history may have required them to adopt such a way of life and is a behaviour passed down to the subsequent generations.

2.1.2 Barriers

The main barriers preventing a consumer from reducing their food waste the were need to provide, inconvenience, lack of priority and exemption from responsibility.

There was a desire to over-purchase and over-prepare food for the sake of satisfying family or visitors and not facing any embarrassment of not serving enough food. The desire to be a good host can also be cultural. For example, it is common in Indian culture for households to stock more food to provide for any guests who may visit.

A factor leading to over-purchasing of food was the inconvenience of having to make many shopping trips. Instead, people tend to over-purchase food to feel confident that there is enough before their next scheduled shopping trip, typically for the next week. Although expiry dates on packaging are intended to be a guideline and not a strict rule, the fear of illness is a common contributor to throwing away food close to its expiry even though it may seem in good condition.

The lack of priority was said to have come from the incorrect perception that food waste is not a problem to care about, in that it does not affect the environment. It also came from the belief that food waste was an accepted social norm and so was not worth engaging in. Lack of priority can also come from other problems taking precedent for a consumer. An example of this can be placing greater priority on other world issues that a consumer feels more strongly about.

Finally, the exemption from responsibility came from the thought that blame for the food waste issue should be placed with the food industry rather than with consumers. They felt that the lack of quality in food items led to wastage at the consumer level, or that promotions targeted at selling a greater quantity of food for a lower price made it inevitable for there to be wastage.

There was also an additional factor that could be interpreted as both a motivation as well as a barrier. Having good food management skills meant that consumers felt confident that they could plan their meals to waste as little food as possible. A common example of this was the foresight to make a large meal and save left overs for the next day. Those with good food management skills also were more confident in using ingredients in different ways to ensure that they would be used. On the other hand, the lack of such skills means that consumers do not feel confident about their ability to plan meals in the future or to use up remaining ingredients.

2.2 Recommendation Systems

In this project, it was anticipated that a recommendation feature could be useful to incorporate into the final solution. Whilst it is not known what form of recommendation will be implemented, some initial ideas involve the possibility of recommending changes in behaviour or even recipes.

Recommendation systems are used to recommend content that might be of particular interest to a user based on a variety of different metrics. Primarily, there are three types of recommendation system: content-based, collaborative filtering and context aware.

2.2.1 Content-Based

Content based recommendation systems base their recommendations on items that a user has indicated their preference for in the past. Each item is described by a set of features, and each user can have a feature profile consisting of features that are associated to items they have liked. For example, a recipe could be considered as an item and its features could include ingredients, cuisine and its source. New items can then be recommended to a user based on a similarity measure between its features and the features in the user's profile.

2.2.2 Collaborative Filtering

An alternative method of recommendation is collaborative filtering [9]. Instead of focussing on finding similar items to those liked by one user, this method identifies similar users and recommends items based on the ratings and behaviours of these users. The process of finding similar users is done using Pearson's correlation, shown in equation 1. For each item i, the rating of users X and Y are denoted as X_i and Y_i with $\bar{X}$ and $\bar{Y}$ denoting the mean ratings for the respective user. The preference for an unrated item by user X (shown in equation 2) is calculated by taking the weighted average of the ratings for that item by all users deemed in the neighbourhood N, where n denotes the number of such users. Users in the neighbourhood are deemed to be the k-most similar users to user X.

$$r(X,Y) = \frac{\sum_{i \in I_X \cap I_Y}(X_i - \bar{X})(Y_i - \bar{Y})}{\sqrt{\sum_{i \in I_X \cap I_Y}(Y_i - \bar{Y})^2 \sum_{i \in I_X \cap I_Y}(X_i - \bar{X})^2}} \tag{1}$$

$$p(X_i) = \frac{\sum_{Y \in N} r(X,Y) Y_i}{n} \tag{2}$$

2.2.3 Context Aware

Context aware recommendation systems acknowledge the fact that context, such as time of day or time of year, can be just as important as a preference [10]. In the context of this project, a user may not eat certain foods in the morning compared to later in the day. There are a couple of methods for incorporating context aware filtering: Context pre-filtering and context post-filtering. Pre-filtering involves using the context to filter out ratings data before being used with any recommendation technology. However, a downside to this approach is that pre-filtering limits the size of the data set that is used to provide recommendations and has the potential to lead to inaccurate predictions. Post filtering generates recommendations using the entire data set and then uses context to filter the results.

2.3 User Interface Design

As a project that is focussed on the delivery of a consumer-facing solution, the development of a user interface is another area of great importance to consider. This section highlights specific research done in this field.

The importance of guidelines and design principles, and their ability to help solve design challenges is outlined by Ben Shneiderman [11]. Guidelines provide a shared language in terms of terminology, appearance and action sequences to help promote consistency across applications on a platform. An example of a set of guidelines that will be useful in this project are Apple's iOS Human Interface Guidelines. Over the wide range of mobile platforms, each has its own unique design language. The iOS interface guidelines help developers and designers to identify areas of an app that can help it look and feel part of the iOS platform, ensuring that users do not have to adjust to starkly different aesthetics or interactions across their apps. Apple highlights three key themes that differentiates iOS's design language: Clarity, Deference and Depth [12]. Clarity involves the use of legible text and icons, and states that graphics and interface elements must subtly highlight important content or areas of interaction. Deference is to help people understand and interact with the app through a beautiful design whilst ensuring that content is always at the forefront. Depth involves making use of layers to provide context as the user navigates the app.

Whilst guidelines are focussed towards specific elements of user interface design, principles tend to be more widely applicable and can be seen as values for what should be considered when designing an interface. Shneiderman goes on to devise eight design principles that should be considered in interface design:

1. Consistency: A system should make use of similar terminology and action sequences to perform similar tasks.

2. Enable Shortcuts: Frequent users should have a way of performing common tasks with shortcut action

sequences to reduce the number of interactions required. An example of this is the control-c shortcut to copy text.

3. Informative Feedback: Every action with the system should have some type of feedback. This can include an animation or a change of page.

4. Dialogs to yield closure: Sequences of actions should have a way of initiating a task, performing the task, and then completing the task. This helps to inform a user when they have completed a task and can move on to performing another.

5. Prevent Errors: The system should be able to detect errors made by a user and offer simple solutions to resolve them.

6. Easy reversal of actions: A user must feel assured that they can undo any action that they make. A simple example could be mistakenly navigating to a page and using the back button to return to the original page.

7. Make users initiators of action: Nothing should happen in the system without the user performing an action. This helps to make the user feel that they are in control.

8. Reduce short term memory load: Action and navigation sequences should be short enough for a user to remember without too much difficulty.

Additionally, there has been considerable research done to understand the relationship between aesthetics and usability in a design. Don Norman states that attractive things work better [13]. He states that since attractive designs can influence a users emotions and affections, this has an effect on their process of cognition which in term determines whether a user is able to perform certain tasks. In terms of a design, poor aesthetics can invoke negative affect which makes a user less tolerable to seemingly minor problems in the design. As a result, whilst aesthetics does not guarantee that a design is usable, it influences the users perception of its usability. A similar conclusion surrounding the importance of aesthetics is put forward by Wang et. al [14] where aesthetic appeal on web services had an impact on the sample participants perceived quality of the service.

Finally, efforts have been made to understand the importance of colour and its role in design. In the field of colour psychology several researchers state that colours do have psychological affect on humans. Faber Birren gives an example where blue invokes a feeling of calm and pleasant ideation, whereas a colour like red invokes arousal, passion and excitement [15].

2.4 Existing Solutions

This section examines the efforts that have been made so far to address the food waste problem at the consumer level. The existing solutions can be categorised as follows: Planning, Shopping, Storage, Disposal, and Redistribution [16]. These categories highlight areas where the solution attempts to influence a consumer's behaviour.

2.4.1 Planning

Solutions targeting the planning phase focus on users' habits before they start shopping for food. Whilst some of these are ubiquitous in households, they can still be considered as solutions to reducing food waste such as shopping lists, menu plans and recipe books. The concept of a shopping list reduces the chance of buying items not needed. Meal planners help consumers buy only the ingredients needed to suit the meals chosen. The provision of quantities in recipe books helps consumers ensure that they do not buy too much of a particular ingredient. Each of these three products have digital equivalents in apps or websites. Apps and web-based alternatives are created to increase convenience for consumers. There are also apps which help monitor the fridge to prevent a consumer from buying food that is already in the fridge.

At the same time, each of these solutions have some downsides. Recipe books can recommend items which they consumer may rarely use again. Non-digital recipe books, in particular, do not adjust quantities required for the number of servings desired which can result in confusion and extra effort for the user to perform calculations to adjust the suggested ingredient quantities. Shopping lists and meal plans require discipline on the part of the consumer to ensure they stick to buying only what they need. This means they are still very susceptible to impulse purchases which may lead to food not being used. Examples of recipe apps and websites include Food Rescue [17], Yummly [18] and Love Food Hate Waste [5].

The Food Rescue website is part of the Waste Less, Save More initiative launched by Sainsbury's. This initiative experiments with several projects to gauge which are most effective at reducing food wastage. Food Rescue provides a list of recipes based on leftover food items a consumer may enter into its search engine. The recipe recommendation system also allows for users to input more than one primary ingredient. The interface is relatively intuitive to understand, with a single search bar and recipe results displayed in a card-based layout, with a sense of colour being evoked from the images of the recipe on each card.

The Love Food Hate Waste app is part of the first major food waste initiative in the UK started by the Waste and Resources Action Programme (WRAP). The app combines all the functionality of a recipe system, meal planner, shopping list, and fridge monitor. To help with the recipes, it also features a portion calculator to recommend how much of an item should be bought to suit the number of people in a household.

The Yummly app does not directly aim to reduce food wastage but is a good example of a well executed recipe app. The app recommends a wide range of recipes sourced from different websites, based on the consumer's cuisine preferences. The interface is pleasing to use through the use of vibrant images and a logical layout. An example of the later two applications can be seen in figure 3.

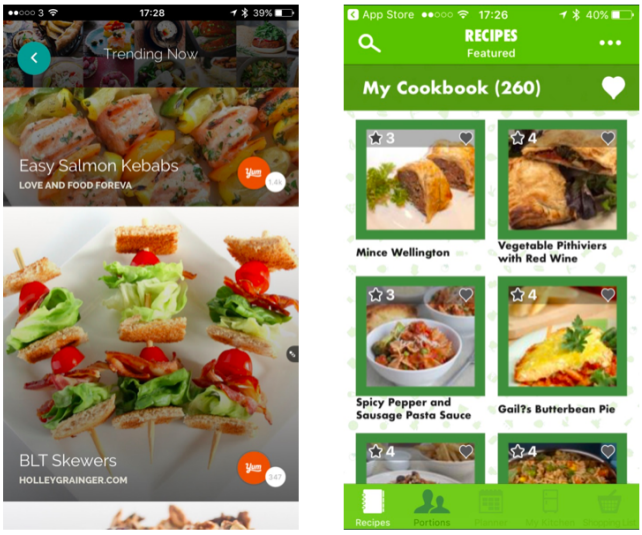


Figure 3: Existing Meal Planning Apps

2.4.2 Shopping

The common idea of grocery shopping involves a consumer going to a market or supermarket and physically taking items off the shelves. Today there are many different ways in which consumers can purchase food. Each of the solutions attempt to influence several behaviours common in the shopping stage.

In addition to convenience, the online shopping concept aims to reduce the urge to make impulse purchases. Online shoppers tend to search for food that they know they want in advance, with less distractions enticing them to buy something not originally intended. However, with online shopping it is possible that consumers will perceive the quality of items selected as worse compared to when they are able to personally select those that are best looking. This may lead to greater wastage if a consumer is not satisfied with the quality of the delivered items. Other solutions that reduce impulse purchasing are grocery box delivery programs such as Hello Fresh [19] and Gousto [20]. These tend to combine the convenience of food delivery with a meal planner and portion control to deliver specific quantities of food required to make select recipes. By delivering the exact quantity of food items required to make a specific recipe, there is less chance of wastage since all the ingredients should be used if the recipe is followed accordingly.

Supermarkets also use discounts and special product ranges to promote fruit and veg with an unusual

appearance. Appearance can be a big factor for why a consumer may not purchase an item, leading to them being wasted. The reduced prices offer an incentive for consumers to overlook this. Examples of special ranges include 'Basics' by Sainsbury's, 'Everyday Essentials' by Aldi and 'Value' by Tesco. Nearly every supermarket chain also discounts food items nearing their expiry date. There is also a Swedish initiative called Resourceful Chef at ICA Malmbords Tuna supermarket where in-store chefs create meals out of foods which have an unusual appearance and may be nearing their expiry dates, that can be purchased. At the same time, these meals are sold at an affordable cost. Since 2007, the store has reduced its food waste by 80% and sells 350 hot lunches a day [21].

Finally, markets and supermarkets alike have tried to tackle the problem of over purchasing food items by selling items unpackaged (e.g loose bananas) and selling single portions of packaged foods such as meat and fish. Both of these solutions ensure that customers can buy only the quantity they need.

2.4.3 Storage

Common storage solutions aim to prolong the life of opened food packaging or help to better communicate how much of a particular food item is left. The conventional method for checking whether food is still in a good condition is to check its expiry date displayed on the packaging. Given that this is only an estimate, there are instances where food is thrown away even though it is still ok to eat. Therefore, there are several solutions which aim to be a better indicator of when food is past its best condition to eat. Stick-on "smart" labels can be placed on items that have been opened. These labels turn from green to red over the period of 1-4 weeks and help to convey the condition of the food inside. This is most helpful to remind consumers of leftovers not finished. More recently, a replacement for the expiry label has been developed in the form of a 'tactile bump' label [22], [23]. Filled with gelatine, the label is bio-reactive and therefore, as the gelatine decays it becomes a liquid allowing a user to feel the bumpy texture underneath. The gelatine is intended to decay at a similar rate to that of food inside.

Aiding a consumers ability to gauge how much of an item is important with regards to food waste. Transparent food packaging allows them to know whether they have enough of an ingredient for a recipe, or may save them from buying a replacement before it is completely finished.

Finally, both Bosch and Samsung have released some of the first smart fridges onto the market. Samsung's Family Hub fridge [24] is connected to wi-fi and has three cameras inside, allowing a user to view its contents from their smartphone. This targets common situations where a consumer is not sure if they need to buy a particular item when out shopping. The product also runs apps that allow consumers to search recipes and order items from a local grocer. The Bosch Home Connect [25] fridge is similar in that it too is connected to the internet allowing interactions through a smartphone app. In addition to cameras, the product provides

alerts if the fridge door has been left open, and provides food management tips through the companion app.

2.4.4 Disposal

Solutions targeted at the disposal stage of the food cycle generally aim at diverting food waste away from landfill sites. The most well known solution is a food bin. Around the UK, councils have implemented a scheme of weekly collections of food waste only, providing residents with a dedicated food bin [26]. Residents are required to put food scraps in a bio-degradable bag, in contrast to the black plastic bags used for general waste collection. The food waste collected is then composted, ultimately preventing it from ending up in a landfill site. A lesser known solution are wormeries, where food waste is anaerobically digested by worms. These solutions, however, do not reduce the food waste generated by a user, only redirecting where it goes.

More recently, the Winnow smart bin [27] is targeted at helping restaurants reduce their food waste. The product incorporates a weight sensor to detect the amount of food waste generated, and makes use of a touchscreen interface so allow employees to quickly identify the specific food wasted at that instant. Given that each bin is tailored to the menu of each restaurant, identification of food items is much quicker. The product connects to a cloud service which analyses and records the waste generated each day. Daily reports can be sent to restaurants of their waste patterns to help identify the value of items being thrown away, ultimately helping to change their disposal behaviour. The company states that customers will typically save between 3-8% on food costs using their smart-bin system.

Finally, a research project [28] was conducted to design a game which would encourage users to compost their food waste and thereby divert it from landfill sites. As part of the project, the game was tested in a university dormitory environment where each user's food waste is weighed and sent to a local farm for composting. The weight recorded was used to calculate a figure for the quantity of CO2 diverted away from landfill sites. Additionally, the weight was also converted into points online to help rank users amongst each other. The project concluded that whilst the game could not be tested in a real world environment outside of the dormitory, the test subjects conveyed that game aspects such as missions and leader rankings made the composting effort more interesting and engaging. At the same time, however, they expressed hesitations about the game design due to its unproven potential to engage users in the long term, and whether the point system based on weight encouraged more food wastage rather than a reduction.

The above research project made use of gamification - a technique making use of gaming design principles and mechanics to make seemingly mundane tasks more fun and engaging. It can involve setting tasks for users, peer-to-peer competition and rewards. Outside of the food waste context, benefit of gamification can be seen in its growing adoption for use in mobile healthcare apps. Mobile healthcare apps help users record and analyse data from wearable or mobile devices. Apps such as Nike+ Fuelband, Runkeeper and

Apple's Activity app are example where gamification has been used as a form of encouragement to help both health and unhealthy users improve and/or be more active. A study conducted at the National University of Singapore [29] concluded that the use of gamification helped to maximise the enjoyment of the app and resulted in increased interest and engagement for both healthy and unhealthy users. Additionally the use of gamification in mobile healthcare apps is effective because of the positive impact it has on users through positive recognition of achievements and social influence [30].

2.4.5 Redistribution

There are many initiatives designed to create a redistribution system where food, that would otherwise have been wasted, is directed to those who may need it most - ie. Homeless hostels, Day care centres for the elderly, women refugee centres. There are small ways in which each of these initiatives differ from one another. The Gleaning network [31] focuses on saving fresh fruit and vegetables from being wasted on farms and directs it to those in need by connecting farmers to food distribution charities and volunteers. Food Cycle [32] not only helps to redistribute food but aims to unite communities by combining volunteers, spare kitchen space and unwanted food to help those in need. Fare Share [33] works with charities and community groups listed above (homeless hostels etc.), forwarding unwanted food to them for them to create meals for vulnerable people. Plan Zheroes [34] is a social network to help build relationships between charities and donors of surplus food. Lastly, Olio is a food sharing app [35] connecting neighbours to each other and to local shops, encouraging the sharing of surplus food and preventing it from being thrown away. Example situations can include food which is close to expiring at a local grocer, or sharing food from a persons fridge before they go away on holiday.

From the analysis of existing solutions, it can be seen that whilst there are many solutions in each of the five stages, most technological innovation appears to be targeting the planning, storage and disposal stages. A common downside to the digital solutions in the planning stage were their need for manual data entries in order for a consumer to experience any benefit from the function or service provided. In the long term, this extra effort may cause users to lose interest and stop interacting with the products. Innovation in the storage stage is taking the form of smart fridges. However, both existing products; Samsung and Bosch, sell for around £5000 and £3000 respectively, making them relatively unaffordable for many households. The disposal stage has been addressed by a smart bin targeted at helping restaurants to decrease their food waste. Its functionality is only possible after inputting the restaurant menu, making it less suitable for a home environment.

2.5 Requirements

This background section covered the landscape of existing solutions, explored the various barriers and motivations for engaging in the reduction of food waste from a consumer perspective, and researched into other factors that can help effect the user experience and increase consumer engagement. As a result of this, it is now possible to identify a capture for the requirements of the new solution.

The aim is to design a solution which not only helps to reduce waste but helps to raise awareness of the issue at hand, exploits the motivations that consumers have about saving money, and helps to develop better food management skills. At the same time, the solution must overcome the barriers of food waste having a lower priority in users' lives and the inconvenience of having to consciously alter behaviour. An important focus for the solution will also be to help automate seemingly tedious tasks such as data entry. Finally, to make the solution accessible for consumers, it must be a solution that is commercially viable in its affordability and scaling to accommodate increased user adoption.

3 Proposed Solution

Following from the background research on existing solutions as well as barriers and motivations for engaging in the reduction of food waste from a consumer perspective, this section formulates the final solution that will be designed and implemented.

The proposed solution will take an internet of things approach to combat the issue of food waste, focussing on the following functions: automated fridge and waste monitoring, recipe recommendations, and encouragement. This functionality will be achieved through the storing and processing of data acquired from sensors placed in or around a fridge and food waste bin. As a whole, the system will comprise of six components: Radio Frequency Identification (RFID) tags on food packaging, a fridge equipped with an RFID sensor, a food bin equipped with a weight sensor, a backend web service, a mobile application and a recipe recommendation system. Automated fridge monitoring will allow users to keep track of perishable items bought, and their expiry dates through the use of RFID detection, eliminating the need to manually enter what items have been bought. Automated waste monitoring will make use of weight sensor data to provide a visualisation for the amount of waste generated by a user and their household over time. Encouragement will seek to improve the consumer's level of engagement over the long term by conveying positive ways to interpret their waste production. An example of this could be highlighting the reductions in waste that have occurred over a month as a result of using the system. Encouragement can also be given through gamification to give users positive recognition of their efforts. The final method of encouragement would involve a notification system to alert users when items are nearing their expiry dates. These kinds of notifications are useful because of the lightweight interaction required by a user to receive useful information. Finally, with a record of food items in stock, recipes can be recommended to help consumers develop better food management skills and use a greater proportion of items bought. Additionally, these recipes can be tailored towards a user's cuisine preferences and prioritise items nearing their expiry dates. For the consumer, all features provided by the system will be experienced from the app component. Given this, the app will be based on an attractive and intuitive interface to enhance the user experience, ensure that tasks are easy to perform, and ultimately help to increase user engagement.

3.1 Assumptions

The conception and development of this solution was made with several assumptions in mind. Firstly, it is assumed that food items will be sold with RFID tags attached as well as their existing barcodes.

An an IOT-based solution, it is assumed that this solution is being developed for consumers with a medium to high income from developed countries, where wireless internet coverage and reliability is greater [36]. The

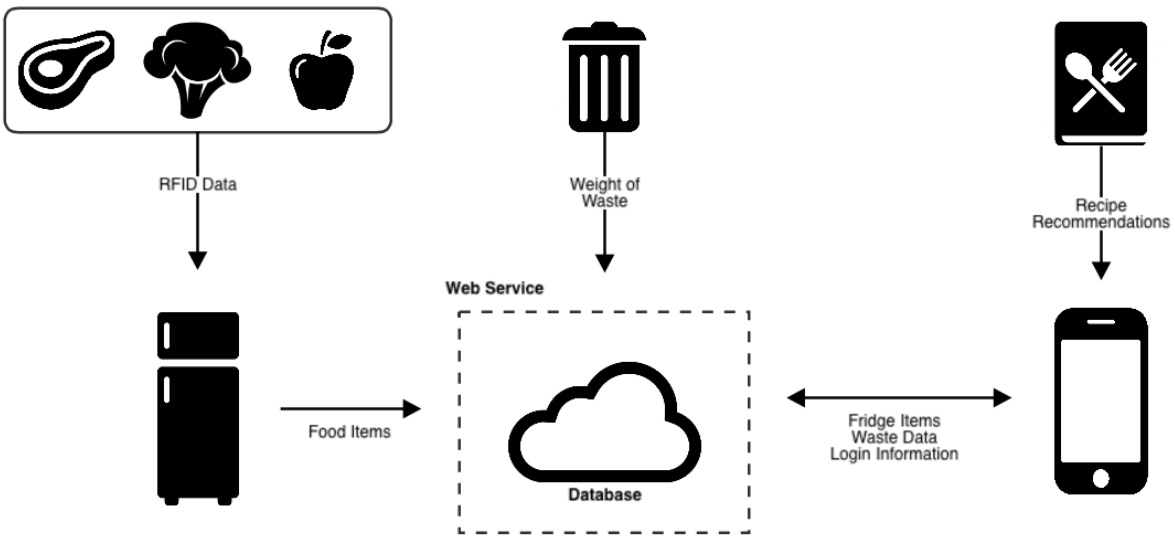


Figure 4: Proposed Solution System Diagram

assumption of a great enough income must be made due to price being a common barrier for most other consumers in the IOT market [37].

Finally, it is assumed that users will own an iOS device capable of running iOS 10 or later.

3.2 Use Case

To understand the role of this solution in the consumer market, it is helpful consider an example scenario of how this solution can be used.

A family of three face a particular struggle with not being able to use all the food they buy on time. This is primarily for two reasons, they are less able to plan meals accordingly to prevent certain items from expiring before they remember to use them, and they often forget what items they have in stock when grocery shopping so tend to over purchase as a precaution and struggle to use all items in time. By making use of this system, after each grocery shop they place their items inside a fridge equipped with the RFID sensor, thereby automatically logging the items they have purchased. By using the app, they see all of their purchased items in order of their expiries and can therefore prioritise which ones need to be used. If there is ever a day when they are not sure what to cook, the app can suggest recipes for them to choose from. In the event they throw food away, the app can display how much food has been thrown to make them more aware and steadily encourage them in the long term to reduce this waste. As a result of using the system for several months, the family have been able to completely stop purchasing items which have already been

bought, develop better skills to prioritise use of certain food items, and save money as a result.

3.3 Abandoned Approaches

This section briefly details other approaches that were considered early on in the development process before being abandoned for various reasons.

3.3.1 Amazon Echo

The Amazon Echo is a smart speaker device commonly used to control other IOT devices in the home through the use of voice commands. Despite the bin recording the amount of waste being produced, it was not possible to identify what items in particular had been thrown away solely based on the weight. A user could choose to throw away 100 grams of chicken, or 100g of tomatoes but the weight sensor cannot discriminate between the two. Instead, an approach that was ultimately abandoned was that users can make short commands to an Amazon Echo device about what item they have thrown away and in what quantity. Ideally, the user shouldn't need to specify the amount in weight but can use generalised quantities if they wish. Example commands could be: "I wasted half of the chicken", or "I threw away a couple of tomatoes". Using the weight data of items in the food item database, the amount of each item wasted can be determined.

Unfortunately, this component was abandoned because of the echo's interaction syntax which takes the form of: "Alexa, tell (app name) + I threw away + (food item and amount wasted)". Whilst the syntax itself is not considered too complex, it adds unnecessary memory load on a user to remember key words that they have to mention for the command to be registered. Additionally, the syntax becomes tedious if a user happens to be throwing away multiple food items.

3.3.2 Monetary Savings

As outlined in the requirements, a key aim of the solution was to exploit consumer concerns about saving money as a form of encouragement to engage in the food waste reduction effort. However, an issue observed was that there was no reliable way to determine what items a user had thrown away without explicitly asking them to perform some form of action. An initial idea was to make an estimate regarding the item thrown away based on the time at which a given item was removed from the fridge, but it was foreseen that this method would not be reliable enough to pursue for this project.

4 System Architecture

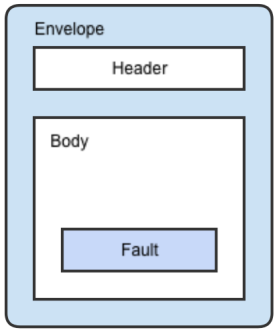
Having outlined the proposed solution, this section will offer a more in-depth view of each of the components involved, and where applicable, will provide an evaluation of relevant technologies considered for implementation. The four components discussed will be the web service, database, recommendation system and mobile application.

4.1 Web Service

The backend web service allows communication between the various devices involved and provides on-demand data to a client device. The design of this component requires consideration for the process of transporting and storing information. From the start, it was decided to make use of Hypertext Transfer Protocol (HTTP) as a transport protocol for web service interaction. Three technologies were considered for the web service: Simple Object Access Protocol (SOAP), Representational State Transfer (REST) and GraphQL.

4.1.1 SOAP

SOAP is an Extensible Markup Language (XML)-based protocol for exchanging information between computers and is transport protocol agnostic. The protocol makes use of XML messaging to exchange complete documents or to call remote procedures. The XML document that forms a SOAP message is made up of the following elements: Envelope, Header, Body, Fault as shown in figure 5 and defines a strongly-typed messaging framework. Each operation provided by a service is explicitly defined, along with an XML structure for the requests and responses involved in the respective operation, using the Web Service Description Language (WSDL). This essentially creates a "contract" between the client and server for how the service will operate.



(a) Diagram

```
<?xml version="1.0"?>
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://www.w3.org/2001/12/soap-envelope"
 SOAP-ENV:encodingStyle="http://www.w3.org/2001/12/soap-encoding">

   <SOAP-ENV:Header>
      //SOAP HEADER
   </SOAP-ENV:Header>

   <SOAP-ENV:Body>
      //SOAP BODY

      <SOAP-ENV:Fault>
        //SOAP FAULT
      </SOAP-ENV:Fault>

   </SOAP-ENV:Body>

</SOAP_ENV:Envelope>
```

(b) XML

Figure 5: SOAP Structure

4.1.2 REST

REST is a style of web service architecture defined by Roy Fielding in his PhD dissertation in 2000 [38]. Though not strictly dependant on any protocol, RESTful services typically make use of HTTP as an underlying protocol.

REST is a resource-based style versus action-based, where a resource can be thought of as a data object. Resource design can be thought of as similar to designing a database by identifying entities and relations. It is important to note that resources are separate from their representations, which are part of the resource state and are how resources get manipulated. Common representations are either JSON or XML and can be decided depending on the type of client. The architecture style is defined by the following six constraints:

1. Uniform Interface: Defines the interface between a client and server, simplifying interactions between the two components. As part of this interface, resources must be identified using URIs (Uniform Resource Identifiers), and whilst a resource and its representation to the client are conceptually separate, a client still holds enough information from the representation to modify or delete a resource on the server.

2. Stateless: The server should not contain any client state between requests. Therefore, all requests from the client side should contain all information necessary for the server to process, and any state information must only be held in the client.

3. Cacheable: Responses from a server to client should be cacheable, helping to reduce network communication for resources which are unlikely to change drastically in short time periods. A server can also specify a duration for how long an item should be cached before it becomes stale or inappropriate to retain.

4. Client-Server: The client and server components are treated as distinct layers, whereby the client component never has direct access to the resource database and so is never concerned with data storage. It must instead make requests to a server for resources. A server component, on the other hand, should not care about the user interface and should only be concerned with listening for client requests and delivering resources requested.

5. Layered System: A client cannot assume that it has direct communication to the end server. It is possible that it is connected to an intermediary which may help improve system scalability. The client need only know the API (Application Programming Interface) and the format of the representation it receives.

6. Code On Demand: An optional constraint that allows a server to temporarily extend a client by transferring logic to the client.

Given that resources that wish to be accessed are identified using URIs, the actions provided by the HTTP protocol helps specify the actions that wish to be executed on the resource. HTTP provides a set of request methods: GET, POST, PUT, DELETE, and more. The specification of one of these methods in the HTTP request helps to identify whether a resource should be returned to a client, a resource should be added or updated, or even deleted entirely.

4.1.3 GraphQL

GraphQL is not just an alternative to REST or SOAP as an architectural style, but is a query language [39]. It enables a client to request exactly what they need rather than being limited to specific responses defined server-side. Tailored responses result in a single request needed to fetch data required versus the possibility of many requests made.

GraphQL is built on the following design principles[40]: hierarchical queries, client-specified queries, backwards compatibility, structured arbitrary code, application-layer protocol, strongly-typed, and introspective. Two important principles from the above are strongly-typed and introspective. Strong-typing means that data must be described using a set of predetermined data types according to the GraphQL type system, ensuring a certain consistency of results returned. Introspection allows a client to query the type system using the GraphQL syntax, removing the need to convert raw JSON into strongly-typed objects when using languages such as Swift or Java on the client side.

Having considered these three technologies, they can be evaluated accordingly. Whilst SOAP provides flexibility in transport protocol, this is not as beneficial in this project given that most web traffic runs over HTTP so presents very little advantage over the other two technologies. Additionally, in SOAP the client and server are tightly coupled as a result of the WSDL rules that are established between the two entities. This means that any changes made on the client or server side will break the entire service. On the other hand, SOAP is sometimes ideal since it supports protocols such as WS-Security and WS-ReliableMessaging which are useful for scenarios like mobile banking where additional security features and a standard messaging system to deal with communication failures are important [41].

In general, REST makes up for what SOAP lacks. For example, the layered system creates a loose coupling between client and server ensuring that changes can be made to either side without breaking the service. As opposed to SOAP, REST carries far less overhead since it does not require a set of rules for the service to be exchanged between the server and client beforehand. In addition, REST has the potential to result in greater performance as a result of having little to no overhead since the service runs on top of HTTP and representations can be encoded in JSON which has far less markup overhead compared to XML.

Unlike GraphQL, REST is restricited to receiving resources in their entire form versus receiving only the

information a client requires. It is possible that the information required by a client requires multiple calls versus a single GraphQL query. As a result, GraphQL helps to reduce traffic between a client and server. According to Gustavsson et. al [42], GraphQL outperforms REST in several performance areas including CPU utilisation, number of bytes sent and received, and response time.

However, despite the benefits that GraphQL provides, it must be noted that REST is still considered a very popular web service architecture. Considering that REST is more established, it is preferred in this project due to its well documented nature and faster development times.

4.2 Database

A database will be used to store information that gets sent from the bin and fridge sensors. The two technologies to consider for the database architecture are SQL and NoSQL.

4.2.1 SQL

Also known as relational databases, SQL databases store data in a structured manner, made up of tables consisting of columns and rows, where each row represents an entry, each column represent a particular data field, and a unique key is used to identify each entry. This simple model structure allows for data to be spread across multiple tables and linked together using foreign keys, keeping data well organised. SQL databases require vertical scaling, meaning that increased demand can be dealt with by increasing the performance of the server hardware.

4.2.2 NoSQL

An alternative to SQL databases, NoSQL databases are document based making it easier to store unstructured data such as articles and photos without needing to categorise into distinct fields. Given the lack of fixed structure, NoSQL databases provided a greater level of flexibility with newer data models. They also have good horizontal scalability, meaning that only a few additional servers are required to support increased demand.

According to Y. Li and S. Manoharan [43], NoSQL databases do generally have better performance characteristics compared to the SQL counterparts. The property of horizontal scaling and data model flexibility also makes this a very attractive technology to implement. However, in this project it was decided to opt for using SQL databases. This was a decision made less with performance in mind but more with the consideration of time required to learn and set up a NoSQL architecture. Finally, given the financial resources

available for this project, it is likely that the database will be held on a single server, so it is not logical to implement a NoSQL solution for this prototype.

Having established that this project will make use of a traditional SQL database architecture, the data model can be designed accordingly. This project makes use of a database consisting of three tables: user accounts, food items, and food waste. An entity-relationship diagram of the database architecture is shown in figure . It can be seen that each of the users will be represented with the following attributes: first name, last name, email, password and userid, with the later being the primary key to distinguish each user. All of these details will be stored as part of a sign up process performed on the app. The second table stores the food items currently being stocked by each user. Each item will be stored with the following attributes: name, expiry, weight, price, userid and itemid, with the later being the unique key associated to the id of each RFID tag attached to the food packaging. The amount of waste accumulated over time will be stored with the following attributes: timestamp, weight and userid. For this table a composite primary key is created using both userid and timestamp to identify each entry.

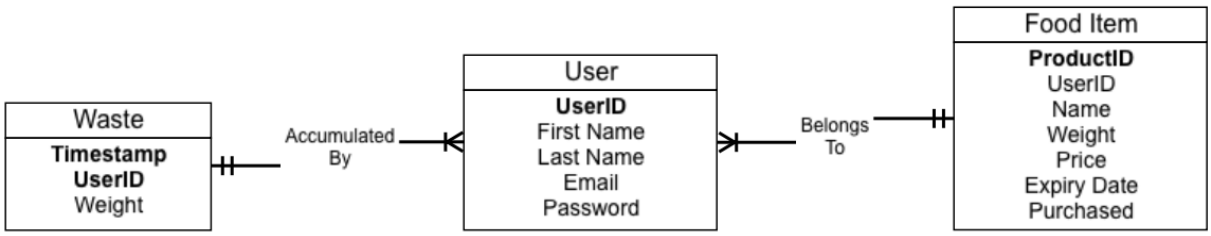


Figure 6: Database Entity-Relationship Diagram

4.3 Recommendation System

Having explored the basics of recommendation systems in the background section, this knowledge can now be used to identify how to implement a recommendation system that best suits the needs for this project. The proposed solution states that users will be recommended recipes based on ingredients they have in their fridge. Considering the time constraints on the development of this project, it was decided to make use of a third-party recipe recommendation API versus developing a unique or existing algorithm from scratch. This decision has its advantages in that 3rd party api's help us connect to a far greater number of recipes than if a recipe database was built from scratch. There were 3 recipe API's considered and evaluated: Yummly, Spoonacular and BigOven.

4.3.1 Yummly

Yummly provides not only an app, as mentioned in the background section, but also an API for developers to access the vast database of recipes stored. The company advertises access to over 2 million recipes from over 70 food blogs and recipe websites. Like the web service architecture used in this project, the Yummly API is based on a REST architecture and so operates over the HTTP protocol. The format of an api call is shown below, with the ability to add additional parameters such as cuisine preferences or allergies.

```
http://api.yummly.com/v1/api/recipes?_app_id=app-id&_app_key=app-key&your_search_parameters
```

4.3.2 Spoonacular

The Spoonacular API is very similar in functionality to Yummly. However, after observing the API documentation, it is evident that the response delivered from a recipe search call contains far less information than that from a Yummly recipe search. An example response is shown in listing 1 containing only 6 fields. This compares to 13 fields delivered by Yummly which includes cooking time information, a full list of ingredients and even cuisine category. Whilst the spoonacular API does provide this information, it requires several different API calls to be made to extract all information we may require. A positive aspect about the spoonacular API is that it provides access to recipe videos in addition to text-based recipes. This is a feature that could appeal to users who are likely to engage more with multimedia content.

```
{
    "id": 641803,
    "title": "Easy & Delish! ˜ Apple Crumble",
    "image": "https://spoonacular.com/recipeImages/
    EasyDelishAppleCrumble-641803.jpg",
    "usedIngredientCount": 3,
    "missedIngredientCount": 4,
    "likes": 1
}
```

Listing 1: Spoonacular JSON Response

4.3.3 BigOven

The BigOven API boasts a similar number of recipes stored to Spoonacular at 350000 and provides a similar amount of information in a recipe search response compared to Yummly. Another important feature of the BigOven api documentation is that it explains how to increase the quality of images returned by appending certain parameters to the image url.

Out of these three possible api's, it was decided to use the Yummly API for this project. Whilst the problems with the Spoonacular api have been detailed, the reason for choosing Yummly over BigOven was simply due to the number of recipes available. Apart from this, it was deemed that both BigOven and Yummly were very similar in their ease of use, quality of documentation and detail in recipe search responses.

4.4 App

From the start of the project, it was decided for the mobile app component to be designed for the iOS platform. This was a decision made due to personal interest in the platform as well as having the necessary software tools and knowledge to embark down this route.

Unlike the other components in the system, iOS apps are restricted to a single Model-View-Controller (MVC) architecture. This design pattern categorises application objects into one of three roles: Models, Views and Controllers, with the assigned roles defining how each object can communicate with one another. Essentially this separates an app's data and logic, and its visual interface elements from each other. Models can be thought of as data objects and their logic, and can represent particular classes in the app, or even data retrieved from the apps local storage. Views pertain to visual elements and objects displayed on screen. Whilst views are primarily used to display data from the model objects, the two are explicitly kept separate because of the desire to be able to reuse and reconfigure these objects for different purposes. The role of a controller is to act as an intermediary between view and model objects, communicating data changes to view objects to display, and interpreting user interactions with view objects to communicate any changed data to the model objects. A diagram of the app architecture in this design pattern is shown in figure 7.

The important modules to highlight are the application delegate, networking and local storage. The application delegate handles events such as app initialisation and local notifications. In iOS, apps are never allowed to run in the background continuously. To preserve system performance, once a user has exited an app, it briefly transitions into a background state, whereby the app is not displayed visually but can perform tasks to save its state and refresh its data from a network. At an unspecified time, the app will be transitioned to a suspended state where it cannot run at all. It is the application delegate which manages the application during these transitions.

Since an app is not kept running continuously, local data storage is important maintain the applications state between uses. More information regarding the specifics of Core Data and Users Defaults is provided in section 7

The networking module handles the launch and preparation of HTTP requests needed to interact with remote web API's such as Yummly and the web service written. Data retrieved from the server response

can then be communicated from the networking module to the necessary view controllers, and as a result be displayed on screen and saved locally. More information about view controllers is also provided in section 7.

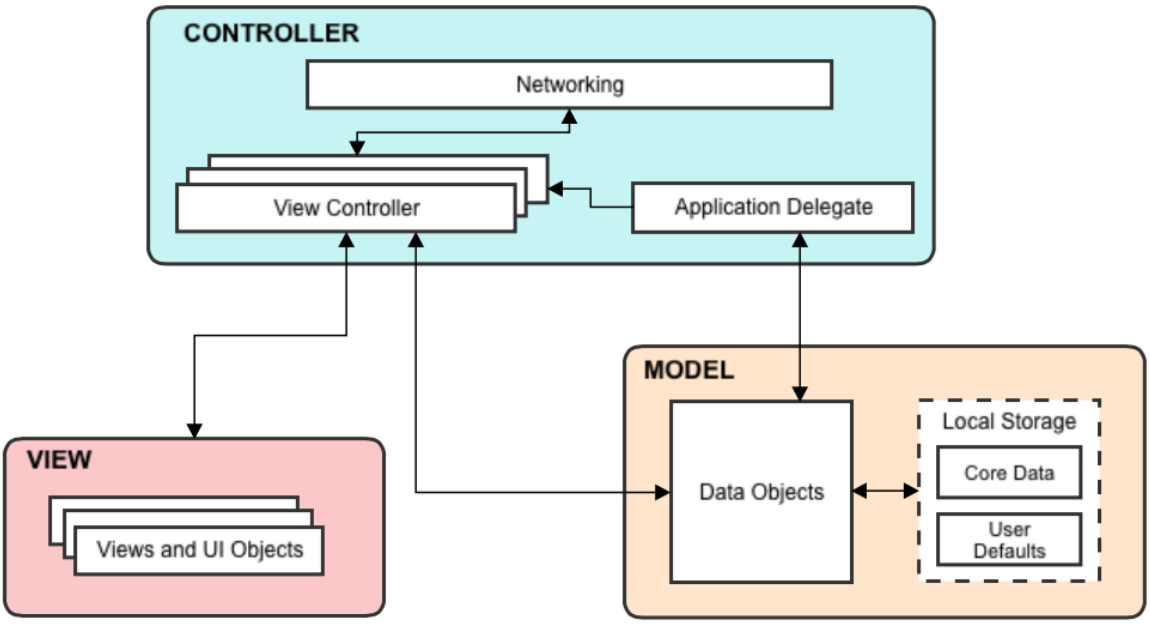


Figure 7: iOS Application Architecture

5 Backend Implementation

In Section 4, it was decided that the backend web service would be built upon a REST architecture, making use of the HTTP protocol to transfer data between the fridge, bin and app components. This section details how the web service component was implemented and features relevant code snippets and flow diagrams to show how particular functions are performed.

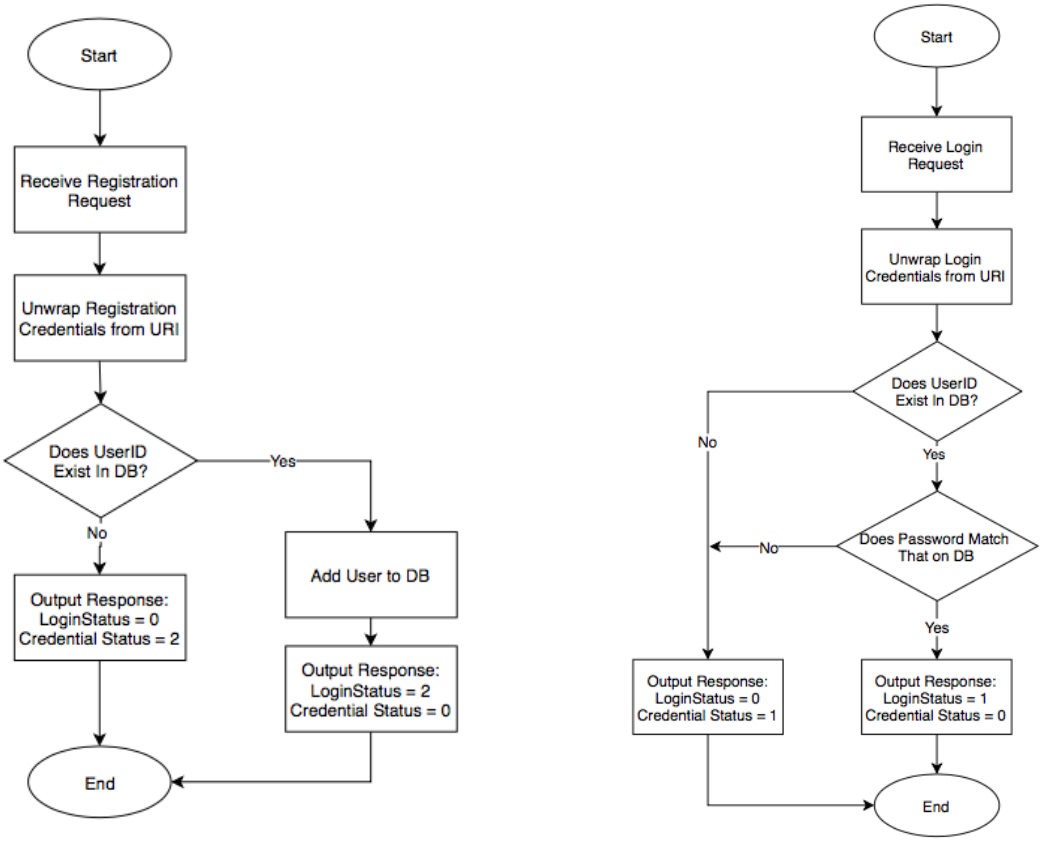
The entire backend was developed in C#, using Microsoft's Visual Studio IDE. This is mainly due to the seamless integration with Microsoft's cloud platform, Azure. In the visual studio environment it is possible to develop both a web API as well as a corresponding database in one project solution and publish straight to Azure in an efficient manner. The database resources provided by Azure are based on an SQL architecture which matches the decision to use this type of database. Additionally, I have personal experience with this language and environment which made development of the backend component viable within the timeframe available.

5.1 Controllers

As stated in Section 4.2, the database is made up of 3 tables: Users, Food Items and Waste. In code, these tables are represented as model objects - similar to the model objects described in section 4.4. In the development of the web API, each of these models are designated a controller to process incoming requests and perform the necessary logic and database interactions to provide a response to the client.

5.1.1 User Controller

The user controller is required to authenticate login requests and process registration requests from the app. The two functions that it performs are represented in a flow diagram in figure 8. As can be seen, the two diagrams look very similar. However, the main difference is in the response sent back to the client. The response is made up of two statuses; a login status and a credential status. The login status can take values of 0,1 or 2 to indicate an unsuccessful login, a successful login, and a successful registration respectively. The credential status can have the same three values to indicate valid credentials, invalid user id or password, and an existing user id respectively. Together, the combination of status helps the app client understand whether the request was successful and if not, gives enough information to understand why.



(a) Registration

(b) Login

Figure 8: User Controller Flow Diagrams

5.1.2 Food Item Controller

The food items controller is designed to interact with both the fridge and app components. For this reason it designed to process requests to add and retrieve food items items to and from the database, respectively. A code snippet of how items are added to the database is shown below in listing 2. An important thing to note is that the post request accepts a list of food items rather than an individual item. This list gets sent in the body of the HTTP request in JSON format. By acknowledging that users may stock a fridge with many items in a short space of time, this was a design decision made to reduce the number of API calls from the fridge component.

```
[Route("AddItems")]
[HttpPost]
public IHttpActionResult BulkAddItems(UserFoodList foodList)
{
    //Adds each item to the db
    foreach(var i in foodList.FoodItems)
    {
        db.FoodItems.Add(new FoodItem(i.ProductId, foodList.UserId, i.ProductName,
        i.Weight, i.Price, i.Purchased, i.Expiry, i.Priority, i.Weight, true));
    }
    db.SaveChanges();
    return Ok();
}
```

Listing 2: Add Food Items Function

```
[Route("GetItems")]
[HttpGet]
public IHttpActionResult GetItems(int id)
{
    UserFoodList userList = new UserFoodList(id);
    var foodlist = from Item in db.FoodItems
        where Item.UserId == id
        where Item.inFridge == true
        select Item;
    foreach(var i in foodlist)
    {
        FoodItem newItem = new FoodItem(i.ProductId, i.UserId, i.ProductName,
        i.Weight, i.Price, i.Purchased, i.Expiry, i.Priority, i.AmountRemaining,
        i.inFridge);
        userList.FoodItems.Add(newItem);
    }
    return Ok(userList);
}
```

Listing 3: Get Food Items Function

Following listing 2 is listing 3 showing how the controller processes a get request from the app. Since the database holds all food items for every user, the get request URI must include an id parameter matching the user id. The controller performs a query on the database, similar to that of an SQL query, using from, where and select clauses to extract all items belonging to the user in question. In addition to post and get requests, there is a 'removeItem' function taking the form of the HTTP delete method to delete entries from the database once the item has been used or thrown away.

5.1.3 Waste Controller

The final controller has two functions to process post and get requests. The code for the get request is shown in listing 4. The goal of the 'getWaste' function is to provide the weight of food waste produced over a specified time period to produce a graph plot on the app. In a design decision similar to that made when posting food items to the database, the 'getWaste' function returns a list of waste entries versus individual records to reduce the number of api calls made. An key part of this function is that it fetches waste entries over a number of past days as specified by the client. This ensures that no duplicate entries are sent to the client, yet that all data is provided to the client even if there is a lengthy gap between get requests. Such a scenario can happen if the user does not open the app for several days. This is a more efficient approach than writing a GET method which returns every waste entry stored in the database.

```
[Route("getWaste")]
[HttpGet]
public IHttpActionResult getWaste(int id, int days)
{
    DateTime today = DateTime.Now;
    TimeSpan t = new TimeSpan(days, 0, 0, 0);
    DateTime d = today.Subtract(t);
    List<Waste> list = new List<Waste>();
    //Returns all waste entries between now and this moment 1 week ago
    var wasteList = from entry in db.Wastes
        where entry.UserId == id && entry.Timestamp >= d
        select entry;
    foreach (var i in wasteList)
    {
        Waste newEntry = new Waste(i.UserId, i.Timestamp, i.Weight);
        list.Add(newEntry);
    }
    return Ok(list);
}
```

Listing 4: Get Waste Function

Above each function in the controllers are two attributes: a route and a HTTP action. The routing attribute is a pattern that gets appended to the web api's base URL and is used by the controller to direct a request to its respective function handler. Under the routing attribute is a HTTP action attribute which helps the controller's routing engine map the functions to particular HTTP action requests.

6 User Interface

This section details the user interface design for the mobile application component of the system. The interface is the medium through which a user will interact with the system and, as stated in section 3, is designed with the intention of helping to increase user engagement with the food waste problem through its attractive and intuitive design. Based on the guidelines and principles explored in section 2.3, the interface has been designed to put colour at the forefront of the app experience. This is done through the use of vibrant edge to edge images and the use of fuchsia accents across the system. This colour was chosen as a differentiator to most other apps, and based on research from Faber Birren [15], it was chosen to invoke a feeling of excitement and positivity - emotions that are desired for this app. The white background found on most pages keeps the interface feeling light and airy, and helps other important UI elements stand out.

The app makes use of a tab bar layout where different pages or 'views' are accessible through icons laid out along the bottom of the screen. Shown in figure 9, as phone sizes grow increasingly larger, it becomes increasingly more difficult for users to reach areas of the screen one-handed. Having such a layout is beneficial from a user experience point of view because it places these navigation elements in an easy-to-reach location, preventing most users from having to adjust their grip when using the app. This design decision is opposed to using a hamburger-menu navigation layout where the user is required to tap a button in the top-left hand side of the screen to reveal a slide out panel containing all other views for the app. Not only would this force users to reach across the screen, but would also create an inefficiency in navigating the app since at least two taps would be required to access any view, versus one tap required using the tab-bar layout. The app also accommodates the use of swipe gestures to replace having to tap back buttons which the iOS system places in the top-left hand corner of the screen.

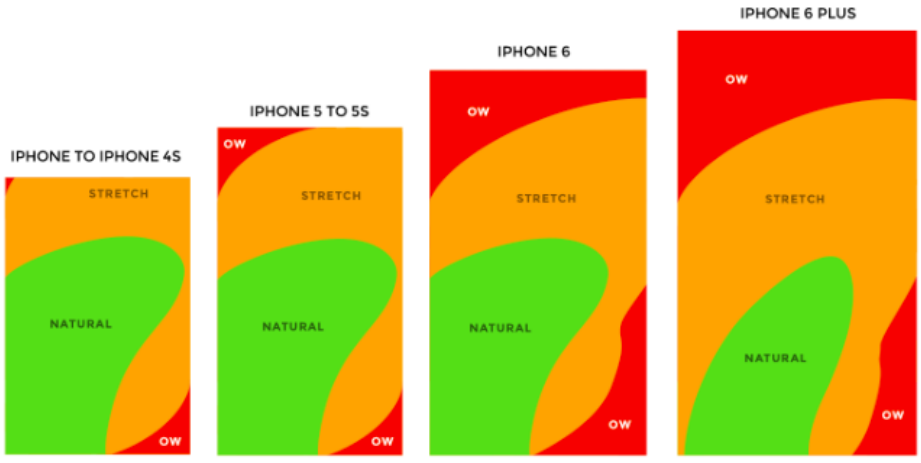
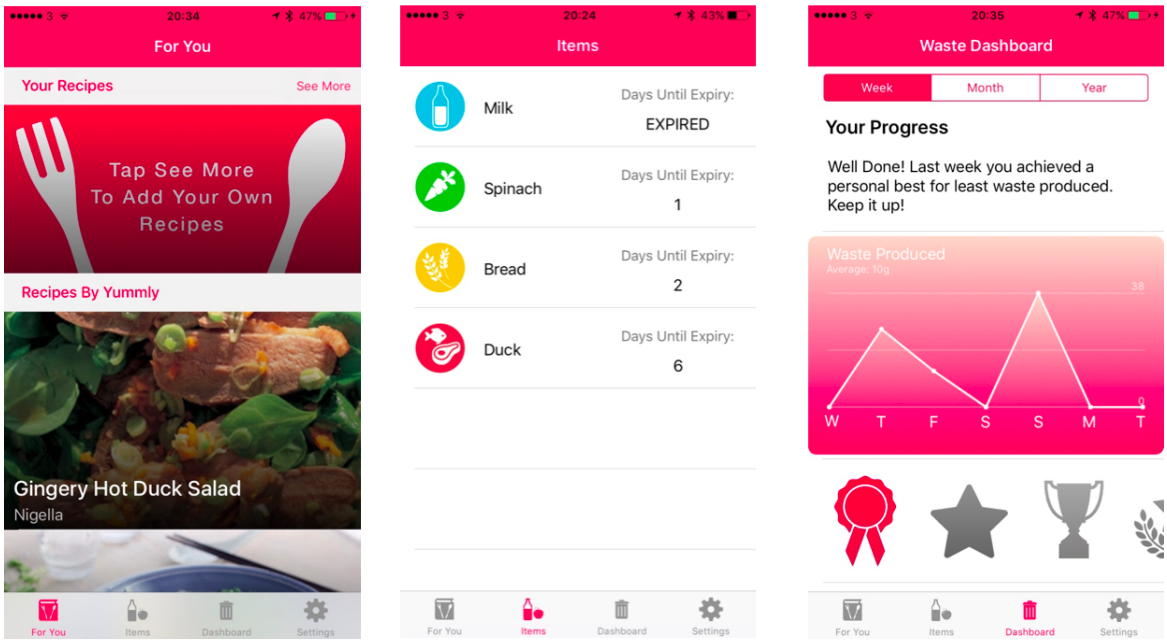


Figure 9: Effect of Single Handed Usage for Increasing Phone Sizes [44]

The app is split into three main 'views': For You, Items and Waste Dashboard. This has been done to clearly represent the three main functions performed by the app and is shown in figure 10.



(a) For You (b) Items (c) Waste Dashboard

Figure 10: Main App Views

6.1 For You

The function of the For You view is to provide users with recipe recommendations using food items that they have in their fridge. In particular, it recommends recipes for items the system has detected are nearing their expiry dates, helping to ensure that fewer items are wasted as a result of going past their expiry.

As stated in section 4.3, the Yummly API is used to pull recipes from over 70 food blogs and recipe websites. Whilst this type of recipe recommendation helps those users who do not know many of their own recipes, the app needs to be useful for those who are enthusiastic cooks and may have their own recipes for cooking certain dishes. To accommodate this type of user, the For You page is split into two sections: Your Recipes and Recipes By Yummly. The former section has a horizontally scrolling set of panels featuring any of the user's own recipes which can be made from the same food item criteria used to generate the Yummly recommendations. Not only does this help accommodate users of varying abilities, but also presents the possibility that a user's own recipe enables them to make a certain dish without having to buy additional ingredients, unlike a third-party recipe. If none of the user's recipes meet the current food item criteria, a placeholder image will be used to instruct users to add their own recipes, as shown in figure 10a The

"Your Recipes" section header also has a "see more" button, allowing users to access a full list of their recipes, including those that do not meet the current food item criteria. More details about tasks that can be accomplished from this view are provided later on in this section.

Aside from their different scroll directions, both sections are based on a similar aesthetic design, making use of large edge-to-edge images with a subtle dark gradient overlay. The use of this gradient is not only aesthetically pleasing, but helps to make text placed on top more legible. In accordance with the terms and conditions of the Yummly API, each recipe features an acknowledgement to its source below the recipe name.

To view recipes in detail, users can simply tap on each item. For recipes shown in the "Your Recipes" section, the app will navigate to a view similar to that shown when the user enters details about that recipe for the first time - consisting of a recipe name, image and list of ingredients. In the case of items from the "Recipes by Yummly" section, the app will navigate to a webpage which displays the full recipe with instructions from the Yummly website.

6.2 Items

The purpose of the items view is to simply provide a list of items which a user has in stock at home. This is a feature which can help to compete with the functionality of smart fridges mentioned in section 2.4.3 which have companion apps connected to cameras allowing users to view what items they have in stock.

The page takes the form of a fairly simple list of item names accompanied by an indicator for the number of days remaining until the item reaches its expiry date. Next to each item in the list is an icon representing one of the following categories: meat/fish, fruit and veg, liquids, and dry food. The use of such icons helps to provide an at-a-glance view of the types of items a user has in stock - helpful for when they may be at the supermarket and have forgotten what types of items they already have at home. This page deliberately makes use of no additional interactions with items in the list. This has been done to facilitate fast and glanceable interaction.

6.3 Waste Dashboard

The waste dashboard view provides the final piece of functionality that was set out in the project aims: making use of automated waste monitoring to provide a visualisation for the amount of waste generated, and to improve a user's level of engagement by conveying encouraging ways to interpret their waste generation.

The page features three key components: a progress statement, a graph and rewards. The progress

statement is a fairy short sentence or two informing the user of how they are doing in their food waste reduction effort. As shown in the figure, the user can be congratulated if they have successfully reduced their wastage from the previous week, or if not they can be provided with a general tip to help them next time. The graph presents the user with a visualisation of how much waste they are generating over three possible time frames; the past week, month or year. The user can use the segmented control at the top of the view to change the time period of waste data. The graph itself has been designed to be in keeping with the general design aesthetic of the app - covering the entire width of the screen so that there are no white borders, and having a background based on the same vibrant fuchsia colour found throughout the app, with a subtle gradient applied for styling purposes. Additionally, the graph features a small indicator for the average amount of waste produced.

The final component of the waste dashboard view is the rewards. As mentioned in section 3, the app will make use of gamification to serve as encouragement to the user, helping them stay engaged. Depending on the progress of a user's waste reduction, certain rewards will show up to make them aware of their achievement. To help motivate them, any remaining rewards not earned will be shown but greyed out. Tapping any of the rewards will display a pop up to explain the achievement in more detail.

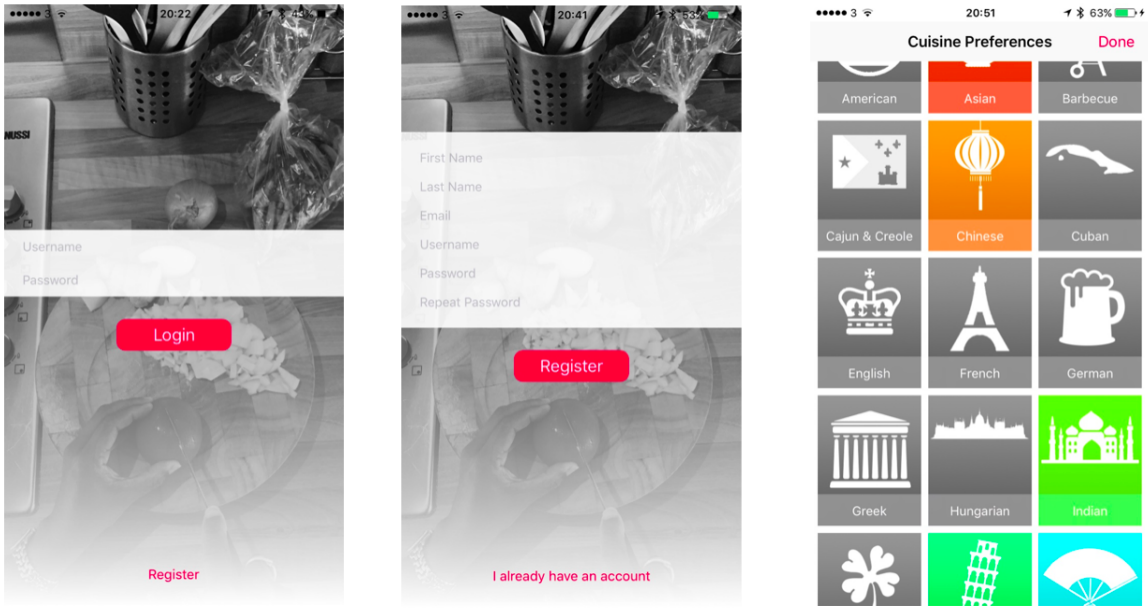
Having detailed the three main views of this app, a description of other important views will now follow.

6.4 Setup

The setup process for a user involves navigating through three views: login, register and cuisine preferences shown in figure 11. The login and register screens are similar and can be accessed interchangeably from each view depending on whether the user has set up an account already. Both views feature fairly simple layouts with a set of text fields overlaying a black and white background. Bright login and register buttons are placed directly beneath the text fields for easy access. The placement of the text fields and the main login/register buttons in either view is done such that they are still visible and accessible whilst an onscreen keyboard is being displayed, removing any action on the users behalf to dismiss the keyboard before they can login or register.

Following login and registration, the user is taken to a cuisine preferences page where they are presented with a scrolling grid of cuisine choices they can choose from. The selections from this page are used to retrieve recipes from Yummly to make recommendations more personal and relevant. Initially the page starts off as a grid of black and white icons for each cuisine however these images switch to vibrant coloured versions as the user selects them. This is not only to fit in with the design aim of using colour to evoke a positive emotion, but having such contrasting appearances helps the user identify which cuisines they have selected versus the ones that have been left unselected. Additionally the gradual introduction of colour

through cuisine selections helps to transition the user from the black and white login and registration pages to the vibrant For You page. To ensure complete clarity, a small text label for each cuisine is featured at the bottom of every grid icon. Tapping the 'Done' button will lead the user to the For You page, where they can start using the app.



(a) Login (b) Register (c) Cuisine Preferences

Figure 11: App Setup Views

6.5 Additional Views

The application comprises of several more additional views: Recipe View, Your Recipes, Recipe Entry and Settings. When the user taps one of the Yummly recommendations, they are navigated to the Recipe View, displaying the respective recipe on the Yummly webpage. As can be seen, the aesthetic design of the webpage is similar to that of the For You page, ensuring that the user has a seamless and consistent experience between pages. From here, the user can scroll further down to view all ingredients and quantities, as well as detailed instructions.

The Your Recipes page not only features all recipes belonging to the user, but also allows them to add new ones by tapping the + icon in the top right hand corner of the screen. Placing the button in a location that is slightly harder to reach is done intentionally to reduce unintentional taps. Tapping this button navigates the user to a recipe entry view which presents itself modally. A modal view is one that moves on top of a user's current view, preventing them from doing other things until a task is completed and helping to create focus. In this case, the user has the choice to either perform the task of entering in their own recipe, or they

can tap the cancel button to dismiss the page. Tapping the done button, will save everything that has been entered by the user. Guidance from Apple Human Interface Guidelines has been used with regards to how this page was designed, such as: "providing an obvious and safe way to exit the modal task, displaying a title that identifies the task, and covering the entire screen for a potentially complex task".

The final view in this application is the settings page which allows the user to adjust their cuisine preferences, log out of their account and connect to the fridge and bin. The latter task is required initially to connect the fridge and bin to a user's home wifi. This is necessary because these devices do not have any screens to be able to perform such functions, so the app acts as a gateway. Since connecting the fridge and bin is not simple, a short set of instructions are presented to the user to guide them through the process.

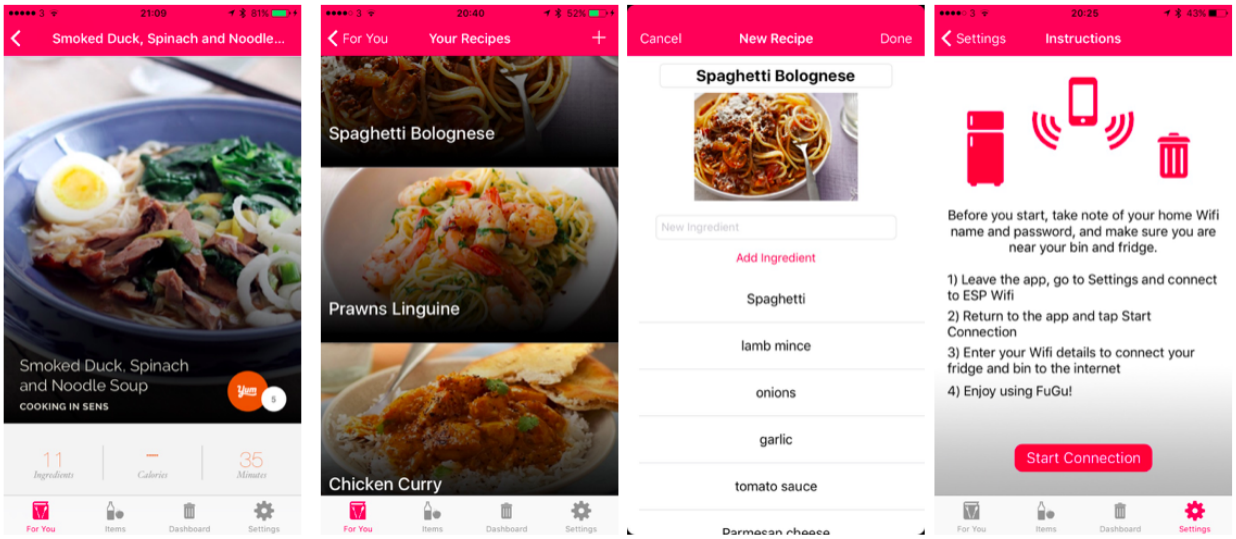


Figure 12: Additional App Views

7 Frontend Implementation

Following the detailed description of the mobile app user interface, this section will cover four aspects of the app: Interface, Local Data Storage, Key Features, and Performance. An in-depth description will be provided for how each of these aspects were implemented.

The application was developed using Swift 3, the third iteration of Apple's open-source programming language. Before the introduction of Swift in 2014, iPhone applications were written in Objective-C, however Swift was ultimately chosen for this project because it is safer, faster and has an easier-to-understand syntax. The Swift language also accommodates an element of backwards compatibility whereby older Objective-C programs and libraries can still run alongside and interact with Swift programs. Development of this app was done in Apple's Xcode IDE.

7.1 User Interface

As stated in section 4.4, the role of controller objects in the MVC design pattern is to act as an intermediaries between view objects and relevant model objects. The controller communicates any changes in model objects to the view and vice versa. Whilst there are several types of Controller object, the most commonly used controller is the View Controller. Every view controller is associated with a particular view in the storyboard, and uses the UIKit framework to help manage that view and any of its subviews, allowing it to manage the entire presentation and handle any transitions to other views. Whilst the inclusion of the UIKit framework allows access to a view object, a view controller can also include frameworks such as core data which allow it to access model data. Whilst a description for every view will not be provided, an in-depth description for views which make use of custom layouts and behaviour will now follow.

7.1.1 For You View

The view controller written to interact with the For You view is the most complex controller in this project because of the vast array of functionality required and the view's unique design layout.

As can be seen from figure in User Interface section, there are two styles in which recipes are presented: a horizontal scrolling set of panels, and a vertical scrolling list. The former is achieved by using a custom table view cell which contains a collection view. A collection view is a class of view used to present items in flexible layout. In this context, it is used to layout a user's recipes horizontally in a single row. Like table views, a collection view is made up of cells. In this case the collection view cell class created contains a single image with a text label to display the name of the dish. The vertical list of recipes from Yummly are implemented

via a custom table view class which consists of an image, two text labels and a gradient layer. The code listing for this custom cell is shown in listing 5. Whilst the image and text labels are featured as outlets - a medium through which storyboard-designed UI elements can be referred to in code - the gradient layer is implemented from scratch programatically. This allow for greater precision with regards to its behaviour. For example we can set the exact colours that get used in the gradient and can set the positioning for the gradient distribution. There is an additional function 'prepareForReuse' which will be explained later on in this section.

```
class YummlyCell: UITableViewCell {
    //Outlets
    @IBOutlet weak var imgView: UIImageView!
    @IBOutlet weak var recipeName: UILabel!
    @IBOutlet weak var sourceName: UILabel!
    var gradient: CAGradientLayer = CAGradientLayer()

    override func awakeFromNib() {
        super.awakeFromNib()
        gradient.frame = imgView.frame
        gradient.colors = [UIColor.clear.cgColor, UIColor.black.cgColor]
        gradient.locations = [0.0,0.85]
        imgView.layer.insertSublayer(gradient, at: 0)
    }
    override func prepareForReuse() {
        super.prepareForReuse()
        self.imgView.image = nil
    }
}
```

Listing 5: Custom Cell Class for Yummly Recipes

As subclasses of the standard UIView base class, both table view and collection view classes require additional functions to achieve their visual layout and data population. The view controller used does not provide this functionality automatically, therefore it must be set as a delegate and a data source to both of these two classes. In iOS development, a delegate is an object that acts on behalf of another to control its user interface. A data source is similar to a delegate, however instead it is delegated control of supplying data to the object. In a table view, for example, the data source provides data to populate each row. In being assigned as a delegate and data source, the view controller must explicitly include several functions. These include declaring how many sections are in the table view and how many rows are to be in each section. One of the most important delegate functions for the table view is shown in listing 6.

The role of this delegate function is to allocate the type of cell that is to be used in each part of the tableview. In this case, to achieve the two styles of recipe presentation the tableview is split into 2 sections. The first section is allocated with 1 row containing a 'Your Recipe' Cell, whilst the second section contains

```
func tableView(_ tableView: UITableView, cellForRowAt indexPath: IndexPath) ->
UITableViewCell
{
    if(indexPath.section == 0){
        let cell = recipeTableView.dequeueReusableCell(withIdentifier: "ForYouCell"
        , for: indexPath) as! ForYouTableViewCell
        return cell
    }
    else{
        let cell = recipeTableView.dequeueReusableCell(withIdentifier: "RecipeCell"
        , for: indexPath) as! RecipeTableViewCell
        cell.recipeName.text = self.recipes?[indexPath.item].name
        cell.sourceName.text = self.recipes?[indexPath.item].source
        cell.imgView.downloadImage(from:
        (self.recipes?[indexPath.item].imageURL!)!)
        return cell
    }
}
```

Listing 6: Tableview Delegate Function

many rows each containing a Yummly Cell. It is in this function, where attributes of the Yummly cell (ie. the image and text labels) can be set.

Finally, at the top of the entire tableview is a search bar. As with the tableview and collection view, the View controller must be assigned as a delegate to the search bar control and include several functions integral to controlling its behaviour. For example, shown in listing 7 is how the search bar control extracts text typed into it and breaks an entire string of ingredients into an array of ingredient words for use with the Yummly API.

```
func searchBarSearchButtonClicked(_ searchBar: UISearchBar) {
    let searchString = searchController.searchBar.text
    searchItems = searchString?.lowercased().components(separatedBy: " ")
    fetchRecipes(items: searchItems!)
}
```

Listing 7: Recipe Search Function

7.1.2 Waste Dashboard

The other view controller which has a somewhat complex implementation is the waste dashboard. For the most part, the view makes use of similar custom layouts as those implemented for the For You view, including custom table view cells and the use of collection views. However, the feature most challenging to implement

was the graph. This is because there are no native charting API's available.

Upon initial research, the graph implementation started by making use of Charts, a third party graphing library. Integration of this library into the app is made possible through the use of a dependency manager called Cocoapods to help manage external libraries. Whilst the library contains a vast array of graph types and appearances, it was found to be limited in its scope for customisation. With further experimentation, it was found that the level of customisation needed to meet the interface design concept would require significant changes to the library and its classes.

An alternative approach which ended up being the chosen method of implementation was to draw the graph programatically, making use of the Core Graphics drawing framework for complete customisation of the graph behaviour and appearance. This removed the need for any external libraries having to be integrated into the application. The programatic process for creating the graph can be outlined in six stages:

1. Apply a gradient onto the graphing area
2. Calculate x and y positions on the graphing area for each data point. This involves ensuring that points have equidistant spacing in the x-axis and that the points are laid out within the confines of the area available.
3. Use the calculated point positions to draw another white based gradient which will appear below the data line
4. Use the point positions to draw the main data line.
5. Draw circles at each of the data points along the line.
6. Draw 3 horizontal lines across the graph: one inline with the maximum datapoint, a second inline with the minimum y position, and a third placed in between the previous two lines.
7. Arrange axis labels on the graph to provide context and meaning to the user.

In the procedure above, each visual element such as the graph line and data points are drawn using the UIBezierPath class. At its core, this class represents a simple path, however its versatility means that a single path can be extended and adjusted in its direction to form almost any shape imaginable. With this procedure for drawing the graph, waste data can be retrieved from the web service and plotted accordingly. To help users interpret their waste generation, the page features a segmented control to filter the time frame of data displayed. Each time the segment changes, the graph is redrawn with the relevant data and adjusted labels.

7.2 Local Data Storage

Whilst most data is stored in the remote SQL database, the app makes use of some local data storage for two reasons: to store data which does not need to be posted to the main SQL database, and as a persistence mechanism to retain the state of the app between uses without having to carry out all functions necessary to reach that state. The app makes use of the Core Data database to store User Recipes (recipes which a user has entered of their own accord) and Waste data for each day. Each User Recipe object contains an image name, a string of ingredients and an image converted into binary data. A local copy of the waste table is saved making any important information more easily accessible than having to make an API call each time it is needed. The food items table is replaced every time the web api is called to ensure that it is always up to date, whereas the waste table is simply updated with new waste entries from the API.

In CoreData, each class of object that is stored is referred to as a managed object. The procedure for interacting with data in the Core Data database involves retrieving a managed object context from the Application Delegate, representing the current state of objects being stored locally. When a user enters a new recipe, a new instance of that managed object is created and inserted into the context.

Apart from Core Data, the app also makes use of the User Defaults database. This is a resource intended to store data that can be used to interpret the apps default state. Data in this store is located using a key for which there is an associated value in the database. Examples of data stored here are the login state of the user, their user id and their cuisine preferences. Saving the state of whether a user is logged into the application enables functions such as auto-login, relieving a user from having to login every time they open the app. Additionally, following the cuisine preferences setup page, these choices can be stored and used in every Yummly API call. The final use of User Defaults data storage is to keep track of the last date that waste data was retrieved. As shown in the backend implementation, the waste controller takes in a parameter for how many days worth of data needs to be retrieved. To avoid duplicate records in the Core Data storage, it is necessary to keep track of the last retrieval date to calculate how many days worth of data is required. The procedure for how this is used is detailed in the next subsection.

7.3 Key Functions

This section explains the implemented for key pieces of functionality in the app.

7.3.1 Waste Retrieval

As mentioned in the previous subsection, the app needs to keep a record of how many days worth of waste data needs to be retrieved from the remote server. Given that new waste data is added to the Waste context in core data, this ensures that no data which has already been retrieved gets stored. The procedure as shown in figure 13 involves retrieving the last date a waste request was made, and calculating the difference between the date and todays date to know how many days worth of data is missing. There are two scenarios when no date will be found in the User Defaults storage: the user has opened the app for the first time, or the user has deleted the app and reinstalled it. In both of these scenarios there will be no waste data stored in Core Data therefore and it will be necessary to request all waste data from the server to have the most up-to-date state again.

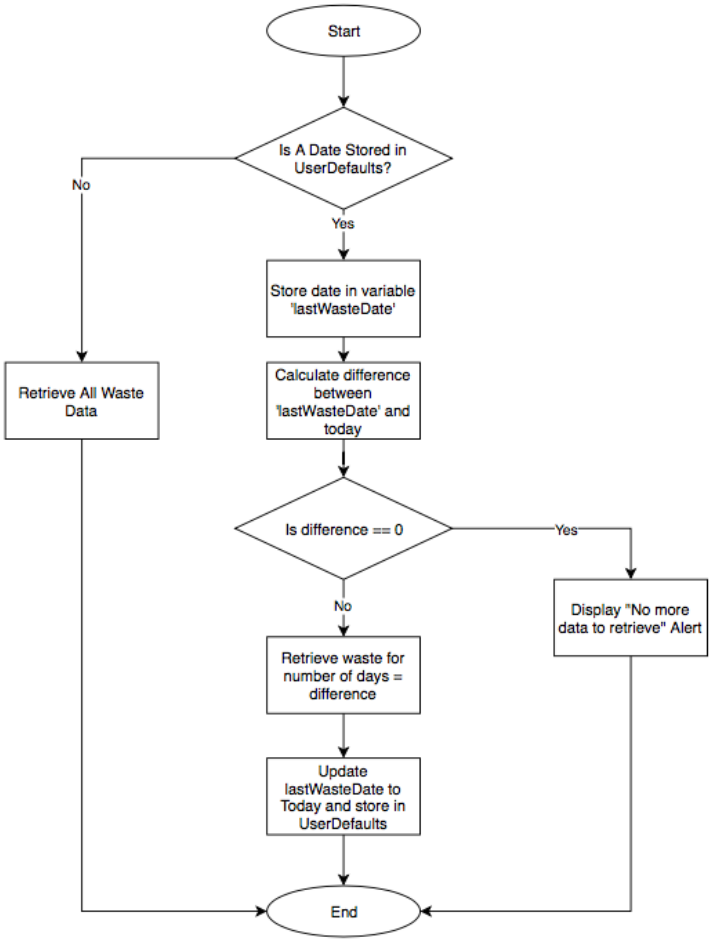


Figure 13: Waste Retrieval Flow Diagram

The next stage of waste retrieval requires processing of the data to input into the graph. As mentioned earlier in the user interface implementation section, a segmented control allows a user to filter the time

period of data that gets displayed on the graph. After updating the context of waste data, a predicate can be added to the fetch request to the filter the results returned. In swift, a predicate takes the form of a query. For retrieval of waste data, three predicates are created to filter data from over the past week, month and year. The retrieval from the Core Data context and the subsequent filtering is shown in listing 8.

```
func getCoreData(days: Int) -> [Waste]{
    let fetchRequest : NSFetchRequest<Waste> = Waste.fetchRequest()
    let entity = NSEntityDescription.entity(forEntityName: "Waste", in: context)
    fetchRequest.entity = entity
    var date = calendar.date(byAdding: .day, value: -(days), to: Date())
    date = calendar.startOfDay(for: date!)
    let predicate = NSPredicate(format: "timestamp > %@", date! as NSDate)
    fetchRequest.predicate = predicate
    do{
        let data = try context.fetch(fetchRequest)
        return data
    }
    catch{
        print("Fetching Failed")
    }
    return [Waste]()

}
```

Listing 8: Waste Retrieval and Filtering

7.3.2 API Interaction

All web api interactions in this app are performed using HTTP Get requests. Using the URLSession class, the relevant API call can be made and during the session, a 'task' is created to retrieve data from the server being accessed. Since all API calls made in this app retrieve data in JSON format, a method of serialisation is required to be able to store and manipulate the data in the swift environment. The JsonSerialization class is used to serialise the JSON response from the servers into a dictionary of strings to objects, representative of the JSON format. With knowledge of the response format, the dictionary can be parsed to extract information of interest and store them in local variables. An example listing for making a Yummly API request and parsing its response is shown in listing 9.

7.3.3 Ingredient Search

Rather than displaying all possible User recipes, the app filters them according to the items which need to be used. As mentioned earlier, the ingredients for each user recipe are stored as a single string instead

```swift
let json = try JSONSerialization.jsonObject(with: data!
, options: .mutableContainers) as! [String: AnyObject]
//gives us json object in form of dictionary of string to anyobject
if let recipesFromJson = json["matches"] as? [[String: AnyObject]]{
    for recipeFromJson in recipesFromJson{
        let recipe = Recipe()
        //extract values we want for each recipe
        if let title = recipeFromJson["recipeName"] as? String,
            let dictToImage = recipeFromJson["smallImageUrls"] as? [String],
            let id = recipeFromJson["id"] as? String,
            let src = recipeFromJson["sourceDisplayName"] as? String{
            recipe.name = title
            recipe.source = src
            recipe.imageURL = self.changeImageRes(url: dictToImage[0],size: 360)
            //need function to extract recipe url
            //Update: recipe url have same format:
            //http://www.yummly.co/recipe/RECIPE_ID?
            recipe.recipeURL = "http://www.yummly.co/recipe/\(id)"
            self.recipes?.append(recipe)
        }
    }
}//array of dictionary
```

Listing 9: Parsing Yummly API Response

of an array to conform to data types that can be stored in Core Data. This function involves having to search through the ingredient string of each user recipe and checking that all items that need to be used are part of this string. To make the implementation concise and readable, a function was written to extend the native string class. The function takes a single items array and loops through each item, checking that the respective ingredient string contains each item. A boolean response is sent back to the main function to indicate whether the recipe matches the ingredient criteria. As a precaution all letters are forced into lowercase to prevent any mismatching due to letters having different cases. This is shown in listing 10.

7.3.4 Notifications

A form of encouragement mentioned in the proposed solution was the use of notifications to prompt and remind users. During development, it was discovered that certain app functionality was restricted because development was carried out without being part of the Apple developer program. Examples of functionality which were prevented from being implemented were push and location-based notifications. Ideally, push notifications would have been used to inform a user if one of their food items was nearing its expiry date. Location-based notifications require access to the Core Location framework which is also limited to members of the Apple developer program. This type of notification could have been useful for reminding users about

```swift
private func fetchUserRecipes(items: [String]){
    do{
        let allRecipes = try context.fetch(UserRecipe.fetchRequest()) as [UserRecipe]
        for f in allRecipes{
            if (f.ingredients?.lowercased().findOccurrencesOf(items: items))!{
                yourRecipes.append(f)
            }
        }
    }
}

extension String{
    func findOccurrencesOf(items: [String]) -> Bool {
        for item in items {
            if self.range(of: item) == nil {
                return false
            }
        }
        return true
    }
}
```

Listing 10: Ingredient Search in User Recipes

their waste progress when approaching a supermarket, to make them conscious about how much food they purchase.

As a compromise, this app makes use of local notifications to provide general prompts to the user. These prompts can help remind the user to use the app in case they may have forgotten about it. As a default setting, these local notifications are scheduled to trigger once a day in the evening, targeting most users who will have come home from a busy day at work.

7.3.5 Security

Given that this project does not handle any sensitive user information, there was less focus on implementing robust security precautions. Despite this, a concious decision was made to make use of HTTPS requests when interacting with the Yummly and web server. Additionally, when passing login credentials to the server, the user's user id and password are hashed using the swift hashing function for the string class, and stored on the server. This prevents anyone intercepting the request from identifying the exact login credentials.

7.4 Performance

During the development of this app, several techniques were implemented to improve the performance of the application and improve the user experience.

7.4.1 Image Resolution

By default the Yummly API provides an image URL which developers can use. However, it was observed that the resolution of the image was 90x60 pixels. All iPhones supported by the latest operating system iOS 10, feature Apple's Retina Screen technology featuring screens with at least 326 pixels per inch. As a result, the quality of the image downloaded was not deemed to be suitable. Whilst Yummly does provide higher resolution imagery, it requires developers to make an additional API request for each recipe returned. This is something which would greatly increase the CPU and network usage considering that 100 recipes are returned from a single search. To resolve this, a function was written to exploit similarities in the URLs between the low and high resolution images. This produces an image of resolution 360x240, large enough to appear clear on high resolution screens, but small enough such that each image does not occupy too much memory. The improvement as a result of this function can be seen in figure 14, which shows a side to side comparison on the high resolution imagery vs the original image resolution.

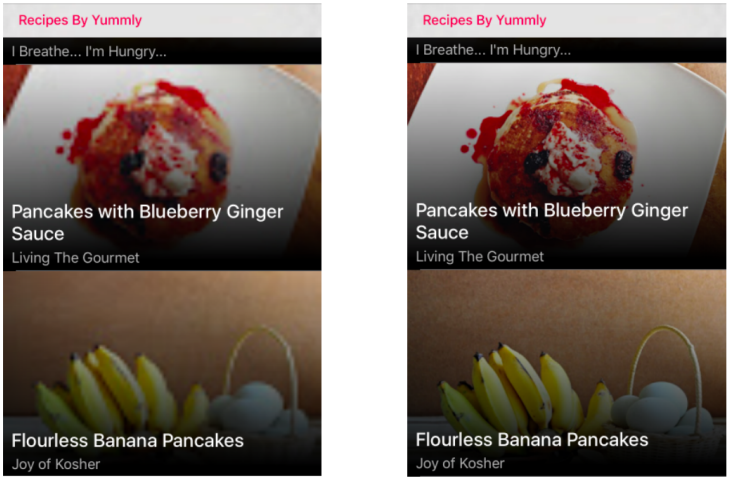


Figure 14: Comparison Between Low (left) and High (right) Resolution Imagery

7.4.2 Cell Reuse

As mentioned, the Yummly API returns around 100 recipes for each search. However, loading 100 table view cells into memory is highly inefficient. Instead, the app makes use of a table view delegate function which reuses cells not currently being displayed on screen. A diagram of how cell reuse works is shown in figure 15.

Such a technique allows the app create only a handful of cell objects and instead change the appearance of each one with the relevant data required. The use of such a technique presents the opportunity to retrieve more than 100 recipes, since the Yummly API has the potential to return thousands in a single response.

Mentioned earlier in this section was a function 'prepareForReuse'. During initial testing of the app, it was found that the cell reuse caused the cell to momentarily display the previous image before switching to the correct image. This creates a poor user experience since the user can become confused by flickering images when they scroll fast. The prepareForReuse function allows for the cell image to be set to nil before it gets reused, thereby disassociating it from the previous image. This ensures that only one image appears in each cell as the user scrolls.

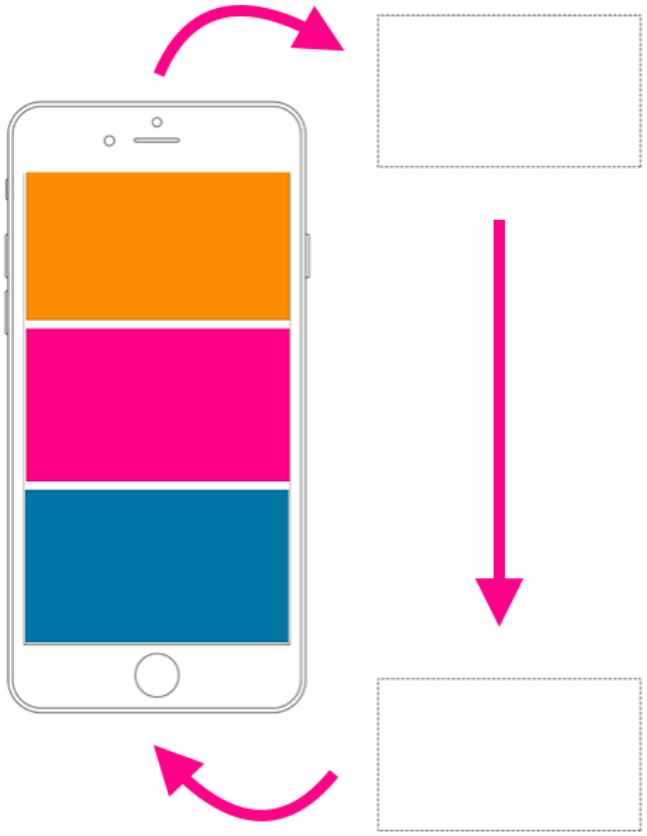


Figure 15: Table View Cell Reuse

7.4.3 Image Caching

Caching is a technique used to save items which are likely to be reused within a short period of time, preventing a program from having to retrieve the item from its original source. In this project, caching is

used for storing images retrieved from the Yummly API. Without caching, the cells that appear on screen are blank momentarily whilst the respective image is downloaded from the internet. As a result of cell reuse, this behaviour would occur for all cells. However, caching results in the same behaviour when an image is being downloaded for the first time, but once loaded it gets cached so that it appears instantly if the cell ever reappears having gone off the screen. Yet again, this technique reduces the number of times an image is downloaded, and saves on data usage for a user when using cellular data.

```
let imageCache = NSCache<NSString, UIImage>()
extension UIImageView{
    func downloadImage(from url: String){
        let urlRequest = URLRequest(url: URL(string: url)!)

        if let imageFromCache = imageCache.object(forKey: url as NSString){
            self.image  = imageFromCache
            return
        }
        let task = URLSession.shared.dataTask(with: urlRequest){
        (data, response, error) in

            if error != nil{
                print(error!)
                return
            }
            DispatchQueue.main.async {
                let imageToCache = UIImage(data: data!)
                imageCache.setObject(imageToCache!, forKey: url as NSString)
                self.image = imageToCache
            }
        }
        task.resume() // fires task
    }
}
```

Listing 11: Image Caching

7.4.4 Asynchronicity

Most of the code in the app runs synchronously - where code is executed line by line. However, for tasks that involve making a network request, asynchronicity is required so that they don't hold up the rest of the program. This is implemented in every function that involves making a remote API call. In the cases where asynchronicity is needed, a separate thread is created to allow interface elements to update independent of the network.

There was one instance in development where the asynchronous execution of network requests proved to be

a problem. This was found when making a request to retrieve items stored in the fridge, and then using these items to make a Yummly recipe search. It is imperative that these two functions run synchronously so that data retrieved from the first function can be passed to the second. To achieve this, the use of semaphores was required. Semaphores can be thought of as a notification to the system that a particular task has finished running. In this scenario, the program is blocked from running momentarily until the semaphore for the items network request is triggered, thereby guaranteeing that there will be data to use in the second request.

8 Testing

This section explains the various testing procedures used to evaluate the functionality of each system component.

8.1 Front End

Throughout development of the app, the iOS simulator provided by Xcode was used to test individual functions and features, and the effect of various interactions. For example, the simulator was useful to test animations and the visual appearance of UI elements. This was necessary for elements implemented programmatically such as the waste graph. Xcode also provides performance analysis tools to use during development, to make sure that the app does not use too much energy or memory. Aside from testing during development on the simulator, the app was tested on real iOS device hardware. Throughout this project, an iPhone 6 running iOS 10.3.2 was used to run the app.

Usability testing was conducted to evaluate the effectiveness of the apps user interface. As explained in section 6, the user interface is the medium through which a user interacts with and experiences benefits from the entire system. For this reason, testing was conducted using a sample set of 9 users: 4 engineering undergraduates from Imperial College London (aged between 21-24), 3 parents (aged between 35-50) and 2 young working professionals (aged 23-30). This age, occupational and ethnic diversity was required to fully evaluate the effect of the app for users of various interests, technical competence and ways of life. Each of the 12 users were asked to perform a set of prescribed tasks with the app. Any difficulties encountered were recorded to highlight areas which were not clear for users.

As highlighted in the use case of the proposed solution, the system is designed to benefit a user in the long term. Given the limited time available for testing, it was not possible to record quantitative data to test whether the system helped to reduce the amount of food waste produced. Instead, a questionnaire was provided to each participant to provide their thoughts on the app, and asked to express their opinions on the overall design, the relevance of recipe recommendations, as well was whether the idea behind the system would make them interested in using it long term.

8.2 Backend

The backend was tested in two stages. The save on financial resources, the web service was tested locally using a separate C# program to test each controller function. Following successful local testing, the web service and database were published to Azure. To test the web service remotely, HTTP POST and GET

requests were sent using Postman, a program enabling easy construction of HTTP requests, with the ability to view the responses and errors in a clear manner. Postman was also used to upload "dummy" data to the remote databases. This was useful when testing the app during development to simulate real data being retrieved.

9 Results

9.1 Usability

Following a brief introduction to the app and its purpose, users were given one minute to explore and familiarise themselves with as much of the system as possible. They were then asked to perform a series of tasks, listed as follows:

1. Perform a search for recipes containing chicken
2. Add your own recipe to the app
3. Connect to the fridge or bin devices
4. Change your cuisine preferences
5. Log out

For the most part, users were able to navigate and perform the prescribed tasks with relative ease. It was observed that Task 2 was by far the most difficult to achieve. Whilst a placeholder image is featured at the top of the For You page instructing users to tap the "see more" button, their instinct was to tap the actual image instead, misreading it to say "tap to see more". Nearly all the test subjects found the "see more" button to be too small. Once provided assistance to find the see more button, every subject was able to enter their own recipe without any further problems. Additionally, a couple of users had trouble with finding the search bar placed at the top of the for you page. They expressed that it could be better to make the search bar text standout in a different colour to the grey colour originally set.

9.2 Questionnaire

Following usability testing, a questionnaire was provided to each subject to receive feedback and their thoughts about their experience. The questions are detailed below.

- What are some things you liked about the app?
- What are some areas of improvement you'd advise?
- Following the overview provided, do you feel this application is lacking in functionality that you would have expected?
- How relevant are the recipe recommendations to your tastes?
- How would you like your waste data to be presented: Amount produced per day, or cumulatively?
- Is the use of gamification something that would motivate you?
- If the entire system of bin, fridge and app cost £200, is this a product that you would likely invest in for the long term?

The questionnaire results for each subject are provided in Appendix B. Analysis of the responses highlighted several common thoughts from participants.

Nearly every subject conveyed that they really liked the aesthetic design of the app, with some further expressing that they liked the level of fluidity and intuition in navigating the app. A few participants even explicitly mentioned how much they liked the general concept of automated monitoring of fridge items and its integration with recipe recommendations. The parent test subjects also expressed their liking for the ability to input their own recipes into the app.

In contrast to how the waste graph was implemented, users expressed that it would be more useful for the them to see their waste production in a cumulative manner for the week versus on a per-day basis.

Most participants did not care about the use of gamification, adding that it wasn't a factor that would motivate them more of less. There were, however, a couple of students that expressed their liking of its inclusion. It turned out that these participants had become used to gamification through the health apps that they use and so felt it was a natural extension to be included in the app.

With regards to improvements, almost half of the participants couldn't think of anything major aside from very small interface tweaks. At the same time, some expressed their desire to be able to see a financial breakdown of how much money was being wasted as a result of their food waste.

With regards to the final question, the price quoted was estimated based on a full cost breakdown of the system, which is explained in the evaluation section. The responses were generally positive or slightly conditional. Most of the students and both young professional subjects expressed that it was a product they would buy, with one student even going on to say that they would buy the product without the bin since they were more interested in the ability to track and monitor food items, track their expiries and receive recipe

recommendations. The parent participants, on the other hand, were slightly more conservative, expressing that if functionality could be implemented to track the exact items being thrown away and the subsequent monetary effects, they would definitely invest in the product.

10 Evaluation

This section evaluates the project deliverable against the desired goals set out in the proposed solution.

Based on the results from testing the app in its usability, as well as the backend in its functionality, it can be determined that the project meets most of the requirements set out. With this solution, the user is not required to perform any manual data entry tasks to experience a benefit from the system functionality. Recipe recommendations for items nearing their expiry, as well as a dedicated list of all items bought help users contribute to the food waste issue without realising, thereby overcoming the barrier of the issue having a low priority in consumers lives. From the testing results, it can be noted that users did not realise that recipe recommendations were targeted at using items near their expiries. This is a feature that helps them develop better food management skills without the need to consciously alter their behaviour. The use of a graph and a reward system helps to raise the user's awareness on their waste habits.

Based on the research done into existing products, it was found that most solutions only targeted a single task in consumer behaviour. This project differed by targeting the planning, storage and disposal behaviours to provide a greater-encompassing solution. The internet of things approach taken, resulted in a solution that made up for many downsides to existing solutions such as manual data entry and high pricing. At the same time, given the limited time available for testing, it cannot be determined whether this solution has a greater effect on reducing consumer food waste. Additionally, the unwillingness to force the user to explicitly detail what items they have thrown away prevents the system from delivering on the aim to engage the user by proving that it can help them save money.

The final criteria to be evaluated is the project's commercial viability for the consumer market. The development of this project incurred costs for both hardware and software resources, with a breakdown shown in table 1

Resource	**Cost**
Microsoft Azure Subscription	£30.29 per month (£363 per year)
Yummly API License	£390 per month (£4680 per year)
Apple Developer Program	£79 per year
Fridge Module (Hardware)	£130 per unit
Bin Module (Hardware)	£10 per unit

Table 1: Cost Breakdown for System Resources

Whilst Yummly recipe recommendations were generated using an academic license which is free of charge, a consideration must be made for the cost of a non-academic license. The cost listed for the Yummly license allows for a maximum of 250,000 API calls per month, equating to just over 8000 calls per day. Assuming an absolute maximum of 15 calls made per day by each user, this license could support a user base of just over 500 users. The azure subscription provides resources to host the web service and database with 250GB of data. During the project, it was not possible to determine how many users this could theoretically support. The Apple developer program fee allows for the app developed to be published on the App Store, allowing consumers to download and use it.

Outside of the scope of this project, but still part of the overall system are the fridge and bin hardware components. These components came to a combined cost of £140 per user.

Taking the assumption of a user base of 500 from the Yummly usage estimate, it can be calculated that an additional £10 would have to be added on top of the total hardware costs to pay off the software resources, resulting in a total price £150 per user. If this price were to increase to £200, it is possible that the the solution can meet its aim of commercial viability by offsetting any other operational costs that may be incurred. Whilst it must be recognised that continual user growth is not guaranteed, there are forms of business model such as monthly or annual subscription payments which can help generate revenue from an existing user base.

Based on this evaluation, it is clear that the project delivered a solution to the food waste issue that met most of its aims. However, it requires additional testing as well as further development to help solve areas that were found to be inconclusive.

11 Conclusion and Future Work

11.1 Conclusion

This report focussed on detailing the design and implementation of the software components to implemented as part of a greater encompassing Internet of Things approach to tackling food waste. These software components are made up of a backend database and web service, as well as an iOS application.

The web service is based on a REST architecture enabling it to receive and send data through the medium of HTTP requests to the fridge, bin and app components. The database is built on a traditional SQL architecture. The app provides automated fridge and waste monitoring, as well as the functionality to recommend recipes provided by Yummly as well as a user's own recipes, based on items nearing their expiry to aid usage of a greater proportion of items over time. The app also focussed on delivering an attractive and intuitive interface to improve the user experience and make tasks easy to perform.

Usability testing of the app showed that the system provided several useful features that engaged users. At the same time, when evaluated against the initial aims set out at the beginning of this report, it was found that more testing and improvements to equate food wastage into a monetary value were required.

11.2 Future Work

Making use of testing feedback, the app can be improved further by firstly, changing the waste graph to display cumulative waste produced on a weekly basis. If enrolment in the developer program is possible, then the app should make use of push notifications to provide more proactive interaction with the user - alerting them to items nearing their expiry and notifying them about their food waste habits before approaching their regular supermarket. A greater level of personalisation can be achieved by applying content or collaborative filtering methods on the Yummly recipe recommendations retrieved. The issue of security is one that was not addressed in this project due to the lack of sensitive information involved. However, more secure communications between the different components of the system can be achieved through the use of a more standardised and robust hashing algorithm versus the built in Swift hashing function. Recipe searches can be improved by providing more filter controls for certain cuisines and even preparation time. Finally, the app can improve by implementing methods to update data in the background. This can involve retrieving the latest fridge items and recipes before the user opens the app, thereby reducing waiting times for the user and enhancing their overall experience.

12 User Guide

This section provides a manual for how the project developed can be installed and used.

As stated in Appendix A, the full code listing can be obtained from Professor Esther Rodriguez-Villegas. The code will consist of two projects: An Xcode project written in Swift called "RESTInteract", and Visual Studio project written in C# called "FoodApp". The former project must be loaded onto a Mac computer capable of running Xcode 8.0, and the later should be loaded onto a Windows computer capable of running Visual Studio 2015.

12.1 Visual Studio Project

There are two options for running the web service: it can be run locally or remotely. To run it locally, the app can be compiled and run from within Visual Studio itself. This local instance can be interacted with through a client such as Postman, running on the same computer.

To run the project remotely, both the web API and database components of the project must be hosted on a remote server. During this project, Microsoft's Azure was used as a hosting platform, where resources for a web app service and SQL server were allocated. Publishing the project to these resources can be done from within Visual Studio itself, as well. During the course of this project, the web service was hosted at the URL shown below. From here, a general list of API methods can be found. Due to limited financial resources, this remote server will be shut down following the end of the academic year (29th June 2017).

```
https://foodappee.azurewebsites.net
```

12.2 Xcode Project

The app is considerably easier to set up. It can be compiled in Xcode without any configuration, and run on the iPhone simulator. The app can also be loaded onto an iPhone, however the device must be running iOS 10.3.2 or later. It cannot be guaranteed that it will run on earlier versions of iOS. Another point to note is that the app has been hard coded to interact with the web service from the above URL and so requires editing to interact with any other endpoint.

A Full Code Listing

A full code listing for all components implemented in this project can be obtained from Professor Esther Rodriguez-Villegas.

B Questionnaire Responses

Questions:

1. What are some things you liked about the app?
2. What are some areas of improvement you'd advise?
3. Following the overview provided, do you feel this application is lacking in functionality that you would have expected?
4. How relevant are the recipe recommendations to your tastes?
5. How would you like your waste data to be presented: Amount produced per day, or cumulatively?
6. Is the use of gamification something that would motivate you?
7. If the entire system of bin, fridge and app cost £200, is this a product that you would likely invest in for the long term?

B.1 User 1 (Student)

1. I think this app looks really nice. I really like the ability to see the items in my fridge without having to do anything.
2. The "see more" button is too small. Maybe it would be better to make the placeholder image a button instead because i thought i had to tap that.
3. I thought the app would have expiry notifications.
4. I like the recipes recommended. I've used Yummly before so i am used to the recipes it provides.
5. Cumulatively
6. Personally, i don't care about the gamification
7. As a student I probably won't buy this, but once I have a job I would definitely consider it.

B.2 User 2 (Student)

1. I love the design of this app, it looks really nice. I also love the ability to use swipe gestures to navigate the interface.
2. I wasn't able to find the "see more" button, so i would advise improvement in that.
3. Not that I can think of.
4. I liked the recipes that were shown.
5. Cumulatively
6. I don't use gamification, but I know that a lot of people do use it so its useful for them.
7. Yes I would buy this.

B.3 User 3 (Student)

1. The styling of the app looks really nice. I love the concept of being able to see recipes based on items that are about to expire, i think it's really cool.
2. The process of connecting the fridge to the internet seems a bit complicated. It would be better if the process was simpler.
3. Once i changed the cuisine preferences to my taste i thought the recipes were nice.
4. Cumulatively on a weekly basis. I don't think i waste enough for the daily view to be of value to me.
5. Gamification wouldn't motivate me
6. Yes, I would buy the solution even without the bin. I'm more interested in the automatic fridge monitoring and recipe recommendations.

B.4 User 4 (Student)

1. I really like this idea of combining fridge and waste monitoring with recipes. Also the ability to search is great because I can look for recipes according to my own criteria. I didn't realise that the recipes recommended were based on the items in the fridge, I think that's really cool.
2. If the app is supposed to target food waste, then maybe it would be better to have the waste page as the first page you see, instead of the recipes.
3. I can't think of anything that is lacking
4. I really like the recipes.
5. Cumulative per week
6. I like the gamification because I'm used to it from my health apps.
7. Yes I think I would buy the product.

B.5 User 5 (Parent)

1. The design looks great. I think this would help me to stay really organised by keeping track of my food expiries. I like the ability to add my own recipes because I really like to cook.
2. It would be nice to see how much money I am saving as a result of using the app.
3. (Same as the above answer)
4. I think the recipes provided were great.
5. I don't mind the waste per day, but i would prefer cumulative for each week.
6. I'm not interested in the gamification
7. If this app could give me information about how much I am saving or tell me how much money I wasted, then yes I would buy the product.

B.6 User 6 (Parent)

1. I think it's a really good idea. I've not seen anything like this. Especially the way the app recommends even your own recipes to you.
2. I wasn't able to find the search bar. Perhaps it would be better to make it more visible than hidden at the top of the list. It would be nice to see both the waste graph and maybe a graph showing how it equates to money wasted.
3. No I can't think of anything that is lacking
4. I like the recipe recommendations but I would probably use my own recipes more.
5. Cumulative.
6. I'm not interested in gamification but maybe younger people like that.
7. If there is proof that I can save money with the app then I will buy the product.

B.7 User 7 (Parent)

1. I like how this app looks. I like the concept of the system as well. The waste page is really nicely laid out, i like the graph and ability to switch time frames.
2. Make the placeholder image the button to add a recipe rather than the "see more" button.
3. It would be nice to have reminders for food items that will expire
4. I really like the recipes that were recommended.
5. Waste per day.
6. No the gamification won't be a motivator.
7. For £200 I would want some information about the money I am wasting from food waste because it would help me change my habits.

B.8 User 8 (Young Professional)

1. It's a great idea and from my limited use it works really well. I really like the layout and splitting the functionality into recipes, item tracking and waste.
2. I cant think of any improvements
3. I thought that I would be able to filter my search based on cuisine.
4. I do like the recipe recommendations
5. I think i would find cumulative waste more useful.
6. Yes I think the rewards system is fun, I like it.
7. Yes I would buy this product because I like how well it seems to work.

B.9 User 9 (Young Professional)

1. I think the app is really easy to use. And it looks different which is nice.
2. It would be nice to see how much money is being wasted.
3. I thought I would be able to add the quantity of ingredients when adding my own recipe.
4. The recipes I saw were ok. I think I would prefer my own recipes.
5. Yes i like the gamification and use of rewards. It's something different to keep me interested.
6. Yes I would buy this product - I don't think £200 is that much for everything I get.

References

[1] S. Goldenberg, Half of all US food produce is thrown away, new research suggests, 2016. [Online]. Available: https://www.theguardian.com/environment/2016/jul/13/us-food-waste-ugly-fruit-vegetables-perfect.

[2] J. Gustavsson, C. Cederberg, U. Sonesson, and R. van Otterdijk, "Global food losses and food waste: Extent, causes and prevention," *Food and Agriculture Organisation of the United Nations, Rome*, 2011.

[3] Initial planning - Planning and estimating potential diversion, 2012. [Online]. Available: http://www.wrap.org.uk/content/planning-estimating-potential-diversion.

[4] Food and Agriculture Organization of the United Nations, 2013, Wastage Footprint.

[5] Love Food Hate Waste, 2017. [Online]. Available: https://www.lovefoodhatewaste.com/why-save-food.

[6] K. Lyons, G. Swann, and C. Levett, Produced but never eaten: a visual guide to food waste, 2015. [Online]. Available: https://www.theguardian.com/environment/ng-interactive/2015/aug/12/produced-but-never-eaten-a-visual-guide-to-food-waste.

[7] The Food Production Chain - How Food Gets Contaminated, 2015. [Online]. Available: https://www.cdc.gov/foodsafety/outbreaks/investigating-outbreaks/production-chain.html.

[8] E. Graham-Rowe, D. C. Jessop, and P. Sparks, "Identifying motivations and barriers to minimising household food waste," *Resources, Conservation and Recycling*, vol. 84, pp. 15–23, 2014. DOI: 10.1016/j.resconrec.2013.12.005.

[9] P. Resnick, N. Iacovou, M. Suchak, P. Bergstrom, and J. Riedl, "Grouplens," *Proceedings of the 1994 ACM conference on Computer supported cooperative work - CSCW 94*, 1994. DOI: 10.1145/192844.192905.

[10] G. Adomavicius and A. Tuzhilin, "Context-aware recommender systems," in *Recommender Systems Handbook*, pp. 217–253.

[11] B. Shneiderman, *Designing the user interface strategies for effective human computer interaction*, 1st ed. Addison-Wesley, 1992.

[12] Apple, iOS Human Interface Guidelines, 2017. [Online]. Available: https://developer.apple.com/ios/human-interface-guidelines/overview/design-principles/.

[13] D. Norman, "Emotion and design attractive things work better," *Interactions*, vol. 9, no. 4, pp. 36–42, 2002. DOI: 10.1145/543434.543435.

[14] Y. J. Wang, M. D. Hernandez, and M. S. Minor, "Web aesthetics effects on perceived online service quality and satisfaction in an e-tail environment: The moderating role of purchase task," *Journal of Business Research*, vol. 63, no. 9-10, pp. 935–942, 2010. DOI: 10.1016/j.jbusres.2009.01.016.

[15] F. Birren, *Color psychology and color therapy*, 1st ed. University Books, 1972.

[16] *Survey of existing consumer products and services which reduce food waste*. Shift Design, 2014. [Online]. Available: http://www.shiftdesign.org.uk/content/uploads/2014/09/shift_Food-Waste_survey.pdf.

[17] Waste Less Save More: Food Rescue, 2016. [Online]. Available: https://wastelesssavemore.sainsburys.co.uk/food-rescue.

[18] Yummly, 2017. [Online]. Available: http://www.yummly.co.uk.

[19] Hello Fresh, 2017. [Online]. Available: https://www.hellofresh.co.uk/tasty/.

[20] Guosto, 2017. [Online]. Available: https://www.gousto.co.uk.

[21] Sustainability Pathways, FAO. [Online]. Available: http://www.fao.org/nr/sustainability/food-loss-and-waste/database/projects-detail/en/c/134603/.

[22] S. Pakstaite, Bump Mark, 2017. [Online]. Available: http://www.designbysol.co.uk/bumpmark/.

[23] S. Hickey, The packaging bump that could cut a 4m tonne mountain of UK food waste, 2015. [Online]. Available: https://www.theguardian.com/business/2015/jan/25/bump-mark-packaging-innovation-cut-uk-yearly-12bn-food-waste-mountain.

[24] Samsung Family Hub. [Online]. Available: http://www.samsung.com/us/explore/family-hub-refrigerator/.

[25] Bosch Home Connect. [Online]. Available: http://www.bosch-home.co.uk/home-connect.html#fridges.

[26] R. Harrabin, Food waste pilot hailed a success, 2008. [Online]. Available: http://news.bbc.co.uk/1/hi/uk_politics/7620109.stm.

[27] Winnow Solutions, 2017. [Online]. Available: http://www.winnowsolutions.com.

[28] J. J. Chun, "A gamified approach to food waste collections at philadelphia university," PhD thesis, 2015. [Online]. Available: http://www.philau.edu/sustainability/inc/documents/theses/JamesChunFinalThesis.pdf.

[29] C. Lee, K. Lee, and D. Lee, "Mobile healthcare applications and gamification for sustained health maintenance," *Sustainability*, vol. 9, no. 5, p. 772, 2017-08. DOI: 10.3390/su9050772.

[30] J. Hamari and J. Koivisto, "Working out for likes: An empirical study on social influence in exercise gamification," *Computers in Human Behavior*, vol. 50, pp. 333–347, 2015. DOI: 10.1016/j.chb.2015.04.018.

[31] The Gleaning Network, 2014. [Online]. Available: http://feedbackglobal.org/campaigns/gleaning-network/.

[32] Food Cycle, 2017. [Online]. Available: http://foodcycle.org.uk.

[33] Fare Share, 2014. [Online]. Available: http://www.fareshare.org.uk.

[34] Plan Zheroes, 2013. [Online]. Available: https://planzheroes.org/#!/.

[35] T. Celestial-One SaashaCook, The food waste revolution. [Online]. Available: https://olioex.com/about/.

[36] P. Biggs, J. Garrity, C. LaSalle, and A. Polomska, *Harnessing the internet of things for global development*. 2017. [Online]. Available: https://www.itu.int/en/action/broadband/Documents/Harnessing-IoT-Global-Development.pdf.

[37] *The internet of things the future of consumer adoption*, ser. Accenture Interactive Point of View Series. 2015. [Online]. Available: https://www.accenture.com/t20150624T211456__w__/us-en/_acnmedia/Accenture/Conversion-Assets/DotCom/Documents/Global/PDF/Technology_9/Accenture-Internet-Things.pdf.

[38] R. T. Fielding, "Architectural styles and the design of network-based software architectures," PhD thesis, University of California, Irvine, 2000.

[39] GraphQL: A Query Language For Your API, 2017. [Online]. Available: http://graphql.org.

[40] N. Schrock, GraphQL Introduction - React Blog, 2015. [Online]. Available: https://facebook.github.io/react/blog/2015/05/01/graphql-introduction.html.

[41] M. zur Muehlen, J. V. Nickerson, and K. D. Swenson, "Developing web services choreography standardsóthe case of rest vs. soap," *Decision Support Systems*, vol. 40, no. 1, pp. 9–29, 2005. DOI: 10.1016/j.dss.2004.04.008.

[42] K. Gustavsson and E. Stenlund, "Efficient data communication between a webclient and a cloud environment," PhD thesis, Lund University, 2016.

[43] Y. Li and S. Manoharan, "A performance comparison of sql and nosql databases," *2013 IEEE Pacific Rim Conference on Communications, Computers and Signal Processing (PACRIM)*, 2013. DOI: 10.1109/pacrim.2013.6625441.

[44] S. Hurff, How to design for thumbs in the Era of Huge Screens, 2017. [Online]. Available: http://scotthurff.com/posts/how-to-design-for-thumbs-in-the-era-of-huge-screens.