

# **Smartphone Recipe Manager**

## **Final Report**

Submitted for the BSc in Computer Science with Study Abroad

May 2016

Word Count: **15775**

By **Craig Matthew Maddocks**

# Table of Contents

1. Introduction.....	1
1.1 Project Comprehension .....	1
1.1.1 Smartphone Usage .....	1
1.1.2 Technology in the Kitchen.....	1
1.2 Project Goals .....	2
1.3 Use Case .....	3
2. Aim and Objectives.....	4
2.1 Objective 1 – Appropriate Management of User Accounts and their Data .....	4
2.1.1 Secondary Objective 1: Online Database .....	4
2.1.2 Secondary Objective 2: Local Database.....	4
2.2 Objective 2 – Create a System UI that adheres to the Developer Interaction/Design Documentation .....	4
2.3 Objective 3 – Create an interactive Recipe Editor / Creator .....	4
2.4 Objective 4 – Implement a method of assisting with Time Management in the Kitchen .....	5
2.5 Objective 5 – Develop an Interactive Hands-free way of communicating with the App whilst cooking .....	5
2.6 Objective 6 – Utilize an External Online Database of Worldwide Recipes.....	5
3. Background .....	6
3.1 Problem Context.....	6
3.2.1 Cooking with Devices.....	6
3.2.2 Programming Languages .....	6
3.2.3 Integrated Development Environments.....	7
3.2 App Store Examples.....	7
3.2.1 Cookpad .....	7
3.2.2 Recipe Keeper .....	8
3.2.3 Yummly.....	8
3.2.4 My CookBook .....	9
3.2.5 Food Network in the Kitchen.....	9
3.3 Comparison of Technologies .....	10
3.3.1 Android .....	10
3.3.2 iOS .....	11
3.3.3 Windows Phone .....	11
3.4 Alternative Solutions.....	12
3.4.1 HTML5.....	12
3.4.2 Xamarin.....	12
3.4.3 Universal Windows Platform.....	12
3.4.4 Final Decisions .....	13
3.5 Processes and Methodology.....	13

3.5.1 Online Database.....	13
3.5.2 Online API .....	14
3.5.3 Hands-free Interaction .....	15
3.6 Potential Problems .....	17
3.6.1 Interaction with SQLite and Parse Database.....	17
3.6.2 Recipe Format .....	17
4. Technical Development .....	18
4.1 System Design .....	18
4.1.1 Class Diagram .....	18
4.1.2 Use Case Diagram .....	19
4.1.3 Program Layout.....	20
4.1.4 API Design.....	21
4.1.4 Database Design .....	23
4.1.5 Coding Convention .....	24
4.2 UI Design .....	25
4.3 Test Design .....	27
4.3.1 Unit Testing .....	27
4.3.2 Acceptance Testing .....	28
4.3.3 Testing Environment.....	28
4.3.4 Review.....	29
4.4 Implementation.....	30
4.4.1 External Dependencies .....	30
4.4.2 Parse Online Database .....	30
4.4.3 SQLite Local Database .....	32
4.4.4 Unit Converter.....	33
4.4.5 Shopping list.....	34
4.4.6 Searching For Recipes .....	35
4.4.11 Timer.....	36
4.4.12 Text-To-Speech / Hands-free Functionality.....	37
4.4.12 Activity Lifecycle .....	38
5. Evaluation.....	41
5.1 Potential Risks .....	41
5.1.1 Project .....	41
5.1.2 Product Design & Deployment.....	41
5.2 Ethical Concerns.....	41
5.2.1 User Health .....	41
5.2.2 User Data.....	41
5.3 Risk Analysis .....	42

5.4 Project Achievements .....	42
5.5 Objective Evaluation .....	43
5.5.1 Objective 1 – Appropriate Management of User Accounts and their Data .....	43
5.5.2 Objective 2 – Create a System UI that adheres to the Developer Interaction/Design Documentation .....	43
5.5.3 Objective 3 – Create an interactive Recipe Editor / Creator .....	44
5.5.4 Objective 4 – Implement a method of assisting with Time Management in the Kitchen.....	44
5.5.5 Objective 5 – Develop an Interactive Hands-free way of communicating with the App whilst cooking .....	44
5.5.6 Objective 6 – Utilize an External Online Database of Worldwide Recipes .....	44
5.5.7 Summary.....	44
5.6 Further work .....	45
5.6.1 Refinements.....	45
5.6.2 Additions.....	46
6. Conclusion.....	47
6.1 Learning outcomes .....	47
7. Appendices.....	48
7.1 Appendix A – Initial Risks Table .....	48
7.2 Appendix B – Initial Time Plan .....	49
7.2.1 Semester 1 (17/10/15 – 18/12/15).....	49
7.2.2 Christmas Break (21/12/15 – 24/01/16).....	50
7.2.3 Semester 2 (01/02/16 – 13/05/16).....	51
7.3 Appendix C – Adjusted Interim Time Plan.....	52
7.4 Appendix D – Initial Task List.....	53
7.5 Appendix E – Acceptance Testing Table.....	55
8. References .....	57

## Table of Figures

Figure 1: Number of Apps across App Stores (Ariel, 2015).....	1
Figure 2: Smartphone Users Worldwide 2014-2019 (Statista, 2016).....	1
Figure 3: Microsoft, Google, Apple App Stores. Search Term "Recipe" .....	2
Figure 4: Use Case Diagram. ....	3
Figure 5: Demy Recipe Reader. ....	6
Figure 6: Android Studio, Xamarin, Visual Studio. ....	7
Figure 7: Cookpad Recipe Creation Page. ....	7
Figure 8: Recipe Keeper Shopping List Page.....	8
Figure 9: Yummly Navigation Drawer. ....	8
Figure 10: My CookBook Text-To-Speech Page.....	9
Figure 11: Food Network's Unit Converter page. ....	9
Figure 12: Smartphone Market Share (IDC, 2015).....	10

Figure 13: Ambient Light Sensor Example.....	16
Figure 14: Proximity Sensor Example.....	16
Figure 15: Project Class Diagram .....	18
Figure 16: Project Use Case Diagram .....	19
Figure 17: Project Architecture .....	20
Figure 18: JSON result for Search Query .....	21
Figure 19: Searched Row Example.....	21
Figure 20: JSON result for Recipe and Design. ....	22
Figure 21: Editing ACL for an Object. ....	23
Figure 22: Class and Layout File Structure of Project.....	24
Figure 23: Initial Design of Application.....	25
Figure 24: Finalised Design of Application. ....	26
Figure 25: JUnit tests for Username and Email Fields.....	27
Figure 26: Nexus 5, Nexus 7, Nexus 10 Emulating different pages.....	29
Figure 27: Shortened Method which handles Action menu.....	30
Figure 28: Async Task that retrieves all Online Recipes. ....	31
Figure 29: Retrieving all Local Recipes for a specific User .....	32
Figure 30: Shared Preferences. ....	33
Figure 31: Unit Converter with History of Session.....	34
Figure 32: Adding Milk, Updating Milk value and adding an incorrect conversion unit.....	34
Figure 33: Methods that Handle Shopping List. ....	35
Figure 34: Adding Time via Step (Left) and manually (Right).....	36
Figure 35: Pattern Matcher for Method Step.....	37
Figure 36: Methods that Handle Position of Step to be Read Aloud. ....	37
Figure 37: Sensor Listener.....	38
Figure 38: Hands-free mode Toggle.....	38
Figure 39: onSaveInstanceState Account Creation Page UI State Save. ....	38
Figure 40: onCreate Pseudo Code of Checking Bundle. ....	39
Figure 41: onPause method for Account Creation Page.....	39
Figure 42: onResume method for Account Creation Page. ....	39
Figure 43: Activity Lifecycle of an Android Application (Google, 2016).....	40

# 1. Introduction

## 1.1 Project Comprehension

### 1.1.1 Smartphone Usage

The use of smartphones in our everyday lives is increasing significantly each year with 68% of Adults in the US owning a smartphone in 2015 up from 35% in 2011 (Mediati, 2015). They can complete almost any task a personal computer can depending on the model and applications installed, as well as providing additional advantages like increased functionality and communication options. These portable computing features have contributed to a substantial rise in users of smartphones each year with predictions of continued growth with every following year (**Figure 2**).

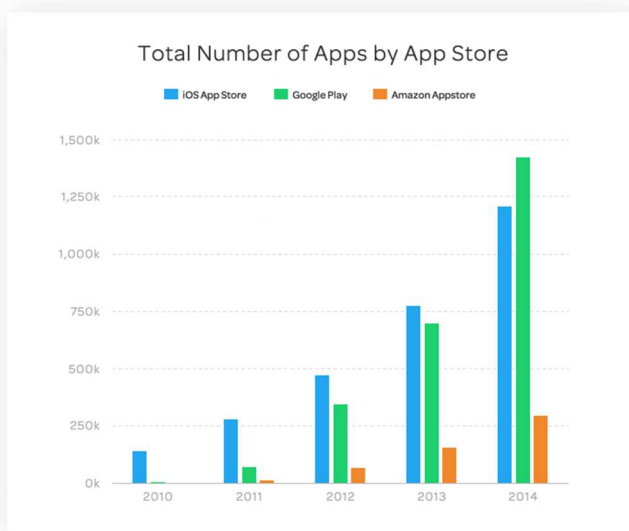


Figure 1: Number of Apps across App Stores (Ariel, 2015).

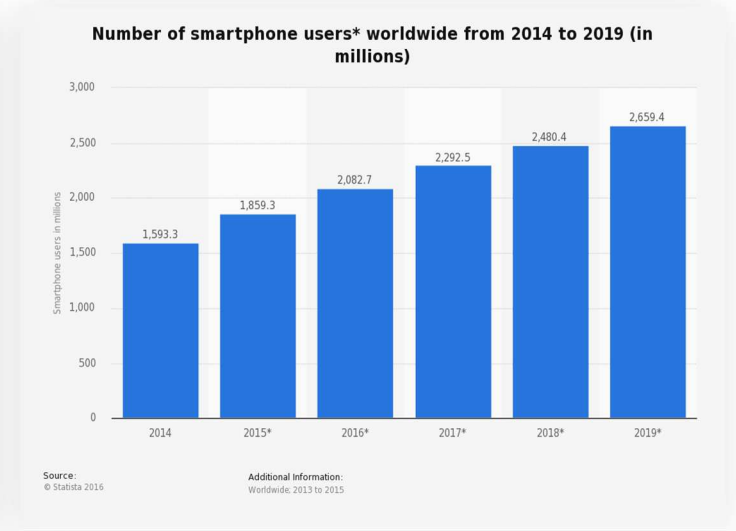


Figure 2: Smartphone Users Worldwide 2014-2019 (Statista, 2016).

There is no denying that smartphones are shaping the world we live in today, with smartphone processor architecture advancing from single to quad-core in 4 years and newer devices containing as much memory as modern day laptops (Triggs, 2015). However it's not just smartphone hardware progressing, the applications that run on devices have also advanced similarly. With the number of Apps available to users on the iOS App Store increasing from 30k to 275k in 4 years (**Figure 1**).

*"Mobile apps are now an integral part of almost every business, irrespective of their size and industry."* (Viswanathan, 2016).

### 1.1.2 Technology in the Kitchen

This project will utilize this ever-changing smartphone technology to create an Application that will assist users with cooking, acting as an interactive cookbook full of recipes. With today's market of technology in the kitchen also advancing over the past several years with new additions to kitchens such as smart fridges, an application like this is relevant now more than ever. The everyday recipe book is soon to be replaced by a more efficient tool that brings more features and versatility to the kitchen environment.

## 1.2 Project Goals

The intended goals of the application are to act as a personal recipe book manager that will help improve culinary proficiency via assisting with time management, organization, and preparation in the kitchen. In addition to managing user-created recipes, it will provide access to an array of online recipes whereby users can edit and try for themselves promoting experimentation in the kitchen with dishes sourced locally or from around the world.

In order to achieve this, research was conducted on the availability of a recipe-managing application across various marketplaces (**Figure 3**).

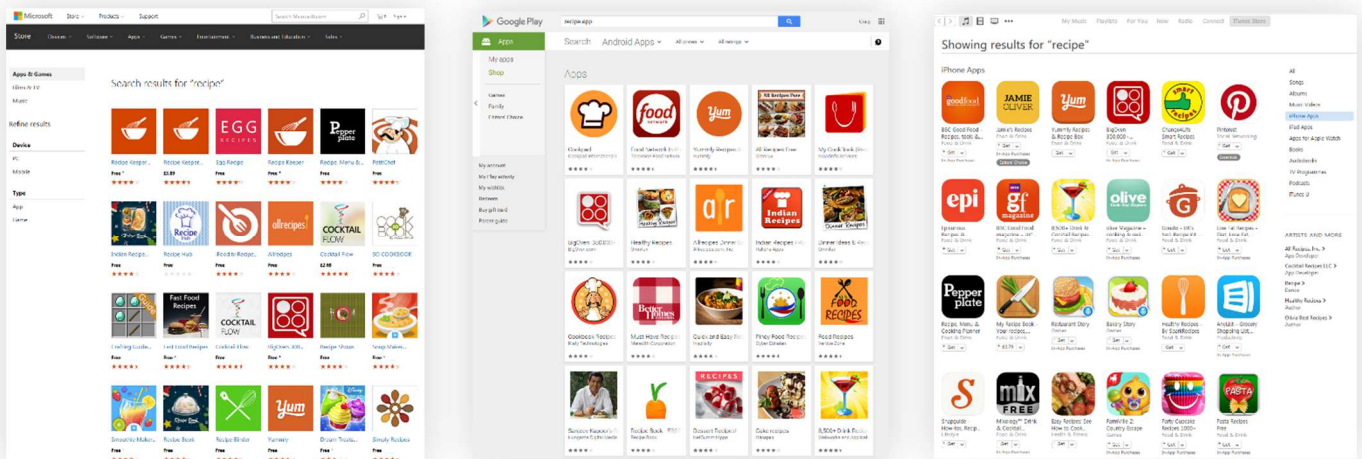


Figure 3: Microsoft, Google, Apple App Stores. Search Term "Recipe".

With several hundred results for a recipe-like app across stores, very few seemed to include all the features one would require for an app that allows users to search for and edit recipes as well as managing their own. The features and functionality that were lacking in a majority of these apps were:

- *Timer function for cooking / preparation steps.*
- *Being able to add photos for each step, not just the final product.*
- *Ability to search for recipes online and add them to user's favourites list.*
- *Ability to create a shopping list of ingredients.*
- *Hands-free functionality.*
- *Text-to-Speech functionality.*
- *Inability to use App in Landscape mode.*

Consequently, a potential market niche was foreseen and ideas of bringing key features seen over several apps together with the lack of features listed above would ultimately meet these project goals.

This report has six sections in total that will aim to provide detail of all phases of the project. They include:

**Aim & Objectives:** - Divide what project aspires to achieve by creating individual objectives made up of tasks.

**Background:** - Describe problem context and comparison of technologies with a summary of chosen features.

**Technical Development:** - UI Design and feature showcase with an explanation of implementation and testing.

**Evaluation:** - Summary of project progress and achievements, with personal reflection and further work explanation.

**Conclusion:** - Overall review of the project and learning outcomes.

### 1.3 Use Case

A user enters their kitchen ready to prepare a meal. They should be able to Login to their personal account on the application where all their saved recipes should sync to form a list. From here they can choose to either search for new recipes accessing a large API of other user submitted meals, or select one of their own self-created recipes (**Figure 4**).

They will see details such as cooking time, serving size, origin of dish and meal type on quick glance with more advanced viewing listing required ingredients and recipe steps on how to prepare the dish. The application will help them with time management and have hands-free capabilities, ultimately trying to provide the user with a solution to the most common issues when it comes to preparing a meal in the kitchen serving as a replacement for the standard cookbook.

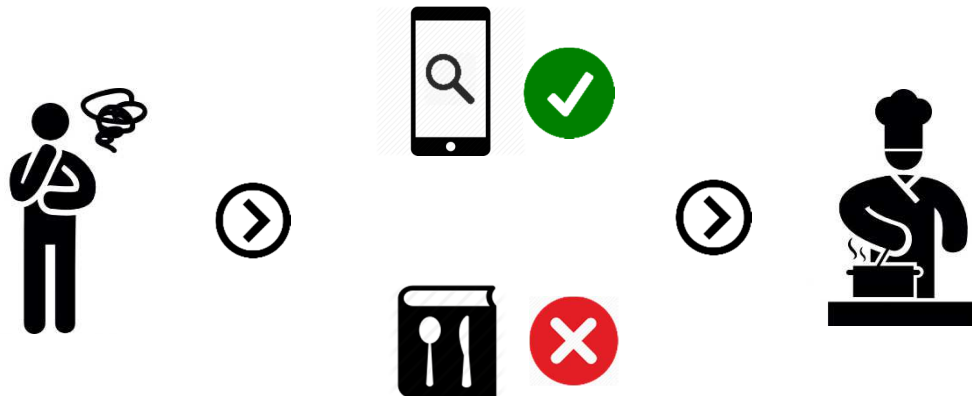


Figure 4: Use Case Diagram.



## 2. Aim and Objectives

***“Creating an Application with ambitions to be a kitchen essential that will enhance cooking preparation, assisting with time management and organization whilst keeping user data secure. Replacing recipe books by means of accessing worldwide dishes that adheres to any dietary restriction”***

### 2.1 Objective 1 – Appropriate Management of User Accounts and their Data

#### 2.1.1 Secondary Objective 1: Online Database

Storing user's data online makes it accessible from any device as long as they have an internet / data connection to log in, increasing portability and widespread use of the app.

Having data that is tied to an online account makes it not only more portable, but in the case of a lost device / app uninstall / broken device, the user's data can be accessed again at a later date. This is useful especially since a third of people lose or damage their phones once purchased (Plaxo, 2011).

#### 2.1.2 Secondary Objective 2: Local Database

Storing user data locally makes it more secure in terms of who can have access to it. With the added benefit of being able to access it offline without a data connection. This increases portability in the sense that the app can be used anywhere, but less portable in that only that device has access to that particular user's data. However not having to worry about server difficulties causing loss of functionality is also useful.

Having data that is stored to a local account makes it more secure and protective of user's private data, as in the case of server hacking, the data could be compromised if implemented incorrectly. This is supported by a report that the number of data records being compromised has increased 78% from 2013 with over a billion records stolen in 2014 (CNBC, 2015).

### 2.2 Objective 2 – Create a System UI that adheres to the Developer Interaction/Design Documentation

The apps layout and overall UI will be expected to provide a consistent experience and should follow standard design and UI rules when being implemented. This is important to the aim, as a design that adheres to formalities will mean the apps navigation will be similar to any other on the device whilst still providing a unique experience through the features.

It has been shown that one of the key features of a successful app is usability, and if users are struggling to navigate due to poor design, then they're much more likely to uninstall it and find a new one. (Inukollu, et al., 2014, p. 19)

### 2.3 Objective 3 – Create an interactive Recipe Editor / Creator

An interactive recipe creator & editor should be present where a user could potentially add their own recipes and additional information so that it can be accessed again in the future. These recipes could then be edited to make revisions of themselves which allows for perfecting or variation of a dish.

This is an objective because one of the major components to a successful app is personalization, it not only increases loyalty but increases purchasing behaviour and time spent using the service (Vaz, 2015). So the ability to allow the user to input their own creations and personalize it to their liking would meet this objective.

## 2.4 Objective 4 – Implement a method of assisting with Time Management in the Kitchen

The app should contain a feature that assists a user with managing time in the kitchen. This feature could help with cooking time, unit conversion etc. It should deal with the majority of minor tasks that slow down the process of cooking in the kitchen.

This is an objective because one of the most essential factors for a successful app is solving a problem, after solving a problem that a lot of people experience, a successful product is born (Varshneya, 2013).

## 2.5 Objective 5 – Develop an Interactive Hands-free way of communicating with the App whilst cooking

A feature that assists the user by allowing them to interact with the app hands-free. As the task of cooking is very hands-on this means that using a touchscreen could cause frustration in terms of registering finger movement and getting the smartphone dirty. This will be done by using the Smartphones hardware to register interaction from the user by means of not using the touch screen. This meets the aim by enhancing the cooking experience by allowing users to use the app but not have to keep washing their hands to use the device.

This is an objective because it solves a problem that most apps on the market do not address. It makes the app unique, which is one of the must-haves for a successful app as unique value differentiates itself from competitors (Nichols, 2015).

## 2.6 Objective 6 – Utilize an External Online Database of Worldwide Recipes

Using an external source for recipes provides users with countless dishes that they could edit and experiment with to their liking. It allows for new tastes with foreign and local cuisine and broadens cooking knowledge by giving the option of new dishes to choose from. This meets the aim by taking the classic recipe book to the next level by providing an ever-changing encyclopaedia of dishes for different meal types.

This is an objective because it allows for all types of users with varying skill levels to use the app, beginners being able to use the online database to start learning how to cook and advanced users inputting their own or modifying existing recipes. This allows for a wider target audience which gives a greater chance of sales success (Samuels, 2015).

## 3. Background

### 3.1 Problem Context

Prior research was conducted into the forming of the aim & objectives. However in order to meet these goals, certain concepts and principles needed to be understood before implementation began.

#### 3.2.1 Cooking with Devices

The current recipe-reader technology available today is dominated by tablet or e-reader devices either specifically tailored for being interacted with in the kitchen, or via an application like this project aims to create.



One of the most popular devices on the market is called the Demy. It's a standalone recipe reader that boasts that its main advantage is being splash / dirt resistant and kitchen safe. This is something that cannot be achieved by an application and therefore gives it an edge on other tablets (**Figure 5**). However, its lack of hands-free interaction, text-to-speech and other features available on other applications in combination with its lack of portability and software updates is its downfall and provided additional considerations to how the implementation of this project would take place (TopTenReviews, 2011).

Figure 5: Demy Recipe Reader.

Some of these considerations were how to handle common problems users might face in the kitchen whilst using a smartphone device, such as:

Dirty hands from cooking → Inability to use Smartphone/Tablet → Hands-free interaction needed.  
 Battery running out during use → Unable to continue with steps → Full charge of device beforehand.  
 Device gets damaged from food / liquid → Unable to access Application → Backup user data online.

The project utilized its advantages over standalone devices such as increased portability and option for software updates to make it more appealing to the market along with the implementations above.

#### 3.2.2 Programming Languages

When implementing the project, a varied selection of platforms was available to develop on, each with their own programming languages which would require the learning and understanding of. Below is a table showing the breakdown of languages available for development along with the developer's personal experience and resources available.

Platform	Language	Experience	Online Resources
Android	Java	Some	Many
iOS	Swift / Objective-C	None	Many
Windows Phone	C#	Experienced	Fewer
Xamarin	C#	Experienced	Least
HTML5	HTML5	Some	Fewer

Ultimately the developer's main concern was the availability of online resources to provide guidance when a problem was encountered, due to having limited knowledge of mobile development combined with the short time period to complete the project. The developer deemed having access to a larger pool of knowledge seemed wiser than selecting a platform with fewer users and inferior documentation for the sake of not having to learn a new language.

In order to deal with the limited time period for implementation, work was started over the summer break practicing different languages and getting confident by creating small projects and familiarizing with the syntax ready for when work on the project began so that important concepts and features were understood.

### 3.2.3 Integrated Development Environments

Like most projects a specific Integrated Development Environment (IDE) will need to be used, the main choices being Android Studio (Android), Xamarin Platform (iOS & Xamarin) and Visual Studio (Windows Phone) (**Figure 6**).

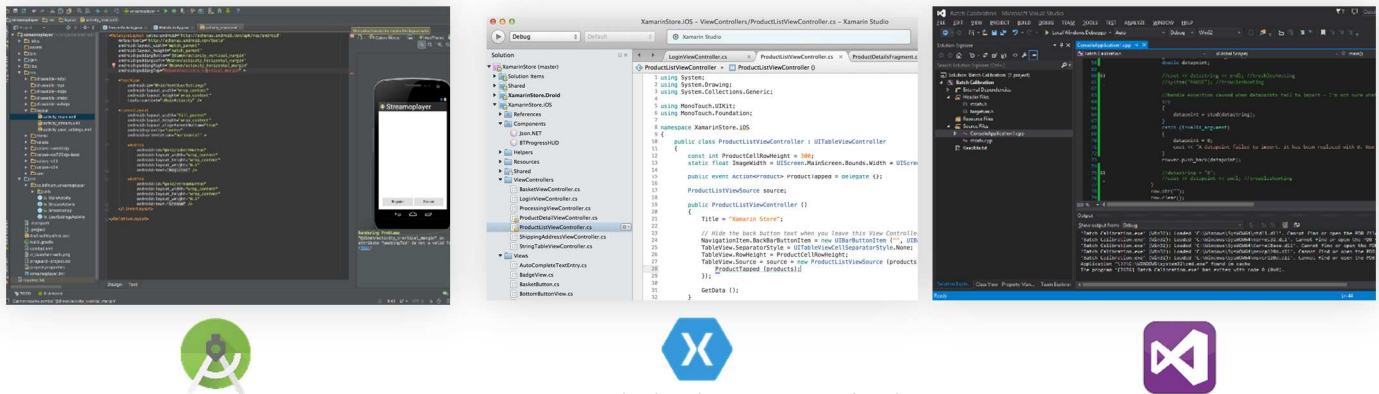


Figure 6: Android Studio, Xamarin, Visual Studio.

In order to familiarize with them, the IDE's were tested by writing small programs, following tutorials and exploring menus to provide additional support to the final decision of platform for the app. Unfortunately, XCode, the preferred IDE for iOS was unable to be tested due to lack of access to an Apple computer hence the use of Xamarin. After using these interfaces briefly the developer found Android Studio would require the most practice learning due to already having prior experience with Visual Studio and C#, however for the most part, all three were similar in design and features.

## 3.2 App Store Examples

In order to establish a prior understanding of competitors and the features used in their applications, research was conducted to investigate apps similar in purpose on both the App Store and Google Play (Apple, 2015) (Google, 2015). The applications which provided inspiration in terms of features and concepts were:

### 3.2.1 Cookpad



Cookpad is currently the no1 app when searching for "recipe" in the Google Play store with 10-50 Million installs. It's a simple application that targets the social side of the market having a large searchable database containing user uploaded recipes which can be downloaded to the device and added to the account. It's also possible to create a recipe and share it with the community so that others can do the same.

There are also options to follow other users and send photo reviews of recipes you have tried, but this is as far as the apps functionality goes.

The main source of inspiration taken from the app was the **Recipe Creation** window. Its layout and details required for the recipe were well thought out and easy to use. Cookpad provides the user with many options when creating a recipe such as Region, Serving size, Cooking Time and Dish Type which enables users to see the vital details of what they are creating but also allows for searching via these parameters (**Figure 7**).

The page also allows the addition of a photo either from the Camera or Gallery and has Editable Text fields for Ingredients and Method Steps which are continuously added to a growing list. The overall design and layout of the creator is modern and easy to use.

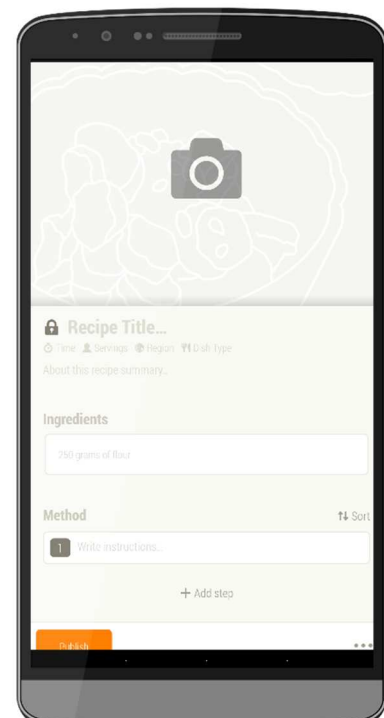


Figure 7: Cookpad Recipe Creation Page.

### 3.2.2 Recipe Keeper



Recipe Keeper is less popular with only 10-50 thousand installs but is the app that's most widespread platform-wise. Available to iOS, Android, Windows phone and as a Windows Desktop App, Recipe Keeper provides basic functionality by giving users the ability to create an account, add recipes and sync it to other devices. It lacks any searchable external database of recipes to add to the device but does contain features that Cookpad and other similar apps lack such as a Favourites / Shopping List and a Meal planner all which sync with the users account.

The main source of inspiration taken from Recipe Keeper was the **Shopping List** feature. The idea of being able to add ingredients to a list that users need to purchase adds a level of functionality that other apps didn't include. Its basic implementation consisted of an edit text field in which users could enter an item. Then once confirmed from either the on-screen keyboard or via the plus button, adds the item to a new row of a checkable list (**Figure 8**).

However, this feature could be greatly improved by connecting the list to the recipe view and allowing automatic population of the Shopping list via a button which would then also add the quantity and unit of ingredient instead of just manually inputting it from a separate window and having to go back and forth.

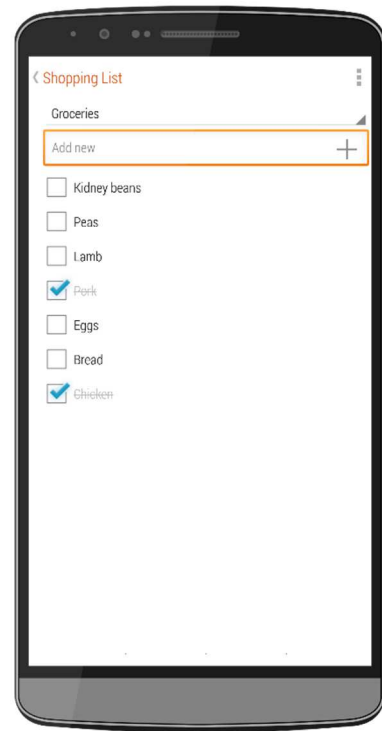


Figure 8: Recipe Keeper Shopping List Page.

### 3.2.3 Yummly



Yummly is the second favourite recipe app on the market with 1-5 Million installs. It acts more of a personal advisor in what recipes to offer the user by creating a very detailed profile about what the user likes/dislikes. It has options to set dietary preferences and filters to choose different tastes for example salty or savoury dishes, tailoring itself to provide the best recipes for the user.

It provided different aspects of inspiration for the project the main feature being the **UI / Design**. Yummy has an easy to use modern interface due to it adhering to Android's UI guidelines. It consists of a sliding navigation drawer and a scrolling list of selectable recipes which expand to show more information once pressed. Another feature which is useful and not seen in similar apps was the use of a Nutrition tab for each recipe displaying the amount of calories etc. which promotes awareness of what users will be creating and hopefully healthy eating. Lastly, Yummly also provides a shopping list feature which unlike Recipe Keeper and others, group's ingredients into categories like Vegetables, Fruit etc. together, which makes shopping easier as food groups are generally stocked in close proximity inside stores (**Figure 9**).

However, Yummly also lacks several features that would be deemed necessary for this project such as the ability to create and view a recipe. Requiring the device to have access to the internet at all times in order to view the steps of each recipe as they are not hard coded but instead contain a link that opens a separate webpage in order to view them.

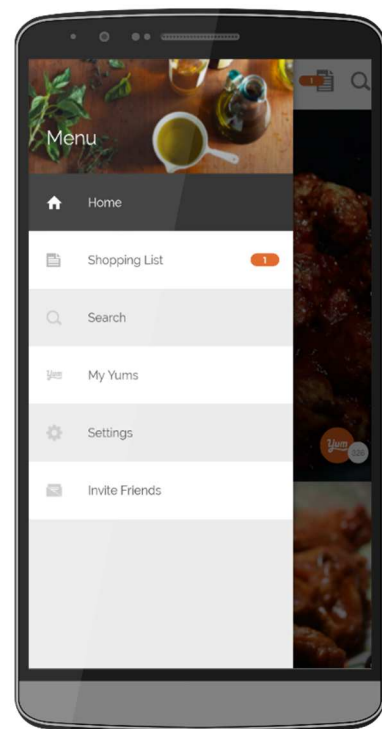


Figure 9: Yummly Navigation Drawer.

### 3.2.4 My CookBook



My CookBook is also popular with 500k-1 Million installs. It is an all-around average application which has a lot of features. Its social aspect is not as good as Cookpad and it doesn't contain any of the personalization settings like Yummly. However, it does contain the most features compared to any other Application tested for this project. A search function, Meal planner, shopping list, file import/export and ability to share recipes with friends to name a few.

The main source of inspiration however came from its **Text-to-Speech** ability to read steps of a recipe aloud with options to read the previous or next instruction. In addition to this, the option to use My CookBook **hands-free** was another source, allowing users to wave over the device in order to progress to the next step of the recipe to be read aloud. This is one of the key objectives that would need to be implemented into this project in order for success (**Figure 10**).

My CookBook also had its downfalls, the primary being a poor recipe search function which instead of utilizing an external database full of recipes just implemented a custom Google Search which made the option of saving recipes difficult and lengthy.

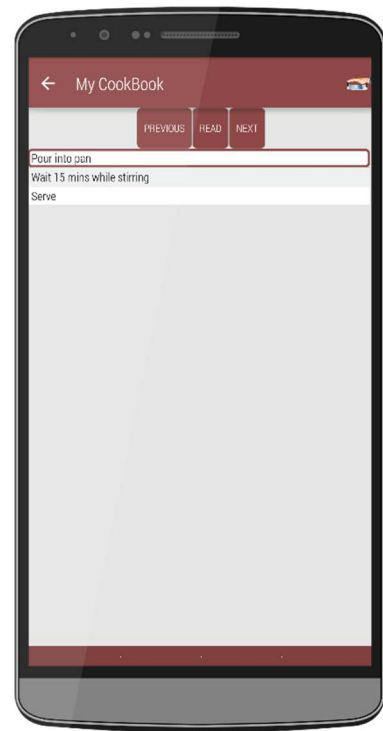


Figure 10: My CookBook Text-To-Speech Page.

### 3.2.5 Food Network in the Kitchen



The Food Network app is the fourth most popular app with 1-5 Million installs. It capitalizes on its TV program and filters the available recipes by certain chefs that present regularly allowing users to search manually as well. It has a limited quantity of available recipes compared to other competitor apps and also doesn't allow the user to create / upload their own personal dishes.

Despite this, the Food Network app has two features that no competitors included that really increase functionality. The first being a **Timer** where users can input a time required for a step in a recipe and it will countdown and play a sound when times up. These timers can also be stacked upon each other to create a total of six simultaneous separate countdowns for different steps making time management simpler. The second feature included was a **Unit Converter** where users can input a value of either Volume or Weight given in an ingredient list and it will convert it to a more familiar or preferred unit. These were both sources of inspiration and meet the objective of improving time management for this project (**Figure 11**).

Food Network did have some other disadvantages, mainly being its lack of functionality offline and poor design in places such as the use of black font over a dark background for example.

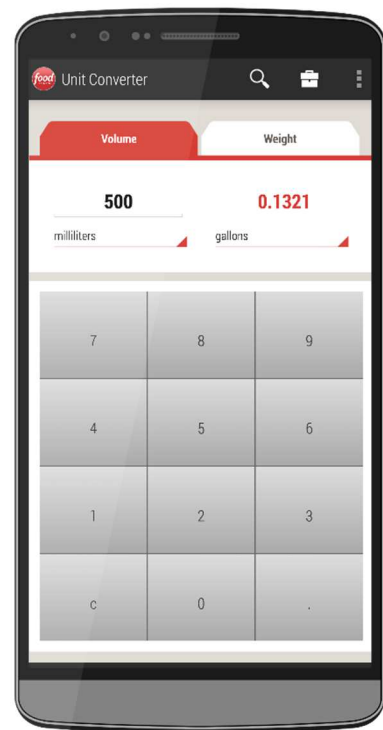


Figure 11: Food Network's Unit Converter page.

Finally, in addition to these features the apps collectively inspired the use of:

- Local / Online Database
- Navigation Drawer
- List of User Recipes Home screen
- External database of Recipes
- Sorting of Lists
- User Accounts

However before considering how to add these features, research into the official developer guides such as the Android API and Apple iOS developer library were performed. For example, when considering UI Design, certain standards have to be adhered to such as not adding an onscreen back button, since Android has on screen controls.

### 3.3 Comparison of Technologies

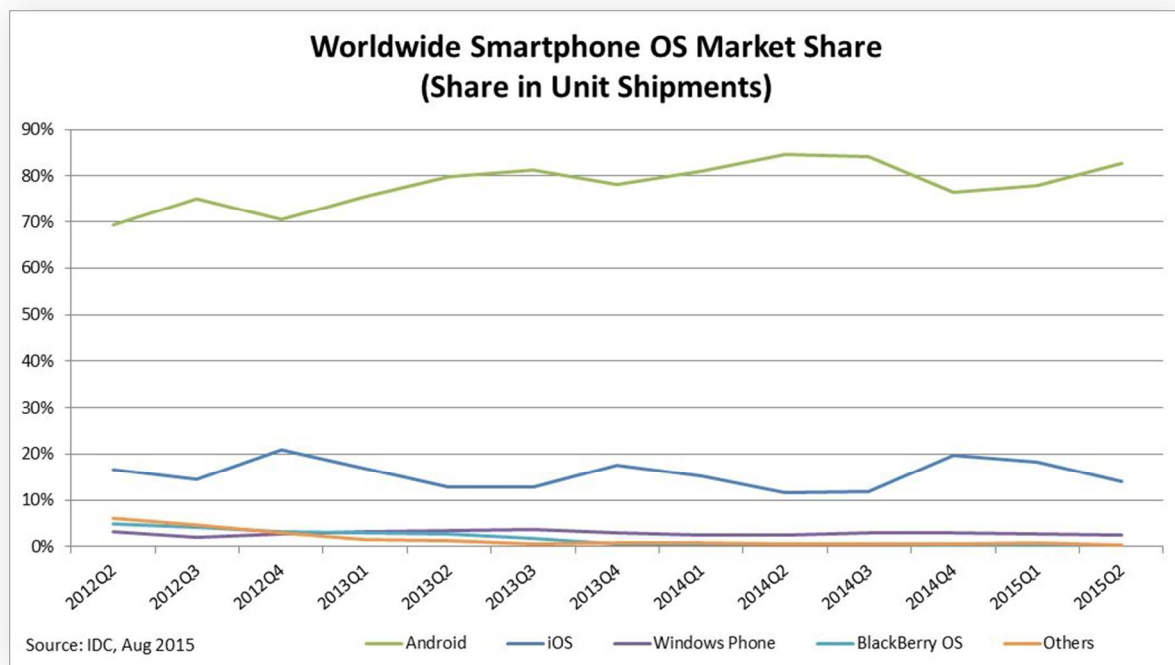


Figure 12: Smartphone Market Share (IDC, 2015).

#### 3.3.1 Android

Android are currently the market frontrunners and have been consistently for several years (**Figure 12**), leading on 82.8% Q2 2015. This puts them ahead of all competitors, meaning a larger scope for new potential users. This means not only a better chance of increased profitability and growth but a means of assisting more people and becoming an essential tool for more kitchens (Healthcare Market Resources, Inc., 2008).

This market success can be largely accredited due to the platform being Open Source (Tiwari, 2014). It gave major companies & manufacturers the ability to tailor Android to their devices without having to spend money on developing an entire Operating system. Similarly, users are also given this option of freedom to customize and tinker, unlike other major platforms at the cost of voiding their warranty.



Developing for the Android platform has also improved, previously apps were created using the Eclipse IDE which was not well received due to bugs and other issues. However, as of September 30<sup>th</sup>, 2015 Google have released a Stable version of the Android Studio IDE which developers can use to create applications. Boasting advantages such as the ability to intercept all build phases of the project, better workspaces, drag and drop GUI designer and superior device emulation (Wikipedia, 2015).

In terms of cost, Android applications can be developed on any operating system for free and charge a one-time fee of \$25 for developing and publishing to the App Store (Android, 2015) (Google, 2015). Overall for students, this is the second cheapest platform and for non-students the cheapest option overall.

### 3.3.2 iOS

iOS are currently in second place with 13.9% total market share, a 68.9% difference from Android (IDC, 2015). Despite this iPhone users have been shown that they're willing to spend more on apps and extra content, with Apple users spending 4 times as much as Google's (Fortune, 2014). This potentially means that even though the amount of users reached is significantly less, the amount they would be willing to spend could make up the loss especially if the app is free.

However, it's been shown that Android is gaining ground in these figures as of Q1 of 2015 with a conversion rate (Transactions / Mobile Sessions) of only 5% compared to Q1 of 2014 which was 20%. Showing that within a year, iPhone users dropped to an average of only 1.2 times more likely to make an online purchase than Android (MoovWeb, 2015).

Nevertheless, iOS is less fragmented across devices (Business Insider, 2015), meaning portability when developing is easier between versions, unlike Android which has a vast selection of models to consider.

Though iOS is also a closed platform, which limits what features that could be utilized when creating the project. For example, the use of NFC is blocked for development and is only permitted for use with Apple Pay, which could cause limitations and cutbacks when it comes to implementation of features (Cult of Mac, 2014). iOS also involves longer approval time for apps to be published on the store, taking on average 72 times longer than Google Play (Fitzgerald, 2014) which would affect how quickly the App will make a profit. Additionally, development for iOS requires a \$99 enrolment fee and access to the OSX operating system for the XCode IDE, making it a pricier alternative and the most expensive of all the platforms excluding middleware (Apple, 2015).

### 3.3.3 Windows Phone

Windows phone not only has a small market share at 2.6% 2015Q2 (IDC, 2015), but also the smallest app marketplace compared to Android & iOS. (Honigman, 2015). This means that although a developed app would stand out more and grow easier in the marketplace, the number of users targeted is very limited. Something that might attribute to this is that as a device, Windows phone was last to enter the market and therefore has a lack of a dedicated developer community and limited social media integration which limits the amount of features an app can utilize (Wilde, 2013).

Windows apps are also limited in that they can only be developed on Windows PC through the use of Visual Studio much like Apple with OSX (Microsoft, 2015). The cost of getting developer status requires a \$19 fee and then an extra \$99 per annual for being able to publish to the Windows App Store. However for students that are a member of DreamSpark, these fees are waived, making it the cheapest option out of all available platforms (Microsoft, 2015).



## 3.4 Alternative Solutions

### 3.4.1 HTML5

HTML5 can be used to develop apps for all three major platforms. Its main advantages include portability, being able to update and still be usable across different devices instantly. It can also utilize hybrid apps whereby an app is created and then wrapped for each platform, making updates quick and easy. This also reduces costs when it comes to development as only one language is required (Android Authority, 2015) (EnterpriseAppsToday, 2015).

However, HTML5 has its downfalls in that the speed and efficiency of apps are much slower than native development techniques as they aren't optimized for the platform. Another problem is the limited access to the majority of hardware available on the device, meaning functionality is decreased too. (Android Authority, 2015) (EnterpriseAppsToday, 2015).

### 3.4.2 Xamarin

Xamarin is a middleware solution to developing apps. It uses a C# shared codebase to write Android, iOS and Windows Apps in their native format keeping the performance and optimization that you would have if developing on them individually (Xamarin, 2015).

It is available to Windows and Mac OS through the use of Xamarin Studio or Visual Studio and has access to run the same app on 2.6 billion devices through sharing the same C# code on all major platforms (Xamarin, 2015). It can share on average 75% of App code (Language, API's, Data Structures) and nearly 100% of User interfaces making it extremely versatile with the portability to multiple platforms seamlessly without having to learn Objective-C, Swift etc. (Xamarin, 2015).

The only drawback is the price as it is rather expensive compared to all other options, the lowest package being \$300 per Annual for access to deploy to App Stores. Which ranges up to \$1899 per Annual for access to all the features available. However similar to Windows Phone, Xamarin offers its product for free to any Student with a DreamSpark account making it one of the cheapest options for this project.

### 3.4.3 Universal Windows Platform

Universal Windows Platform (UWP) is somewhat of a middleware solution developed by Microsoft on the release of Windows 10. Its primary use is being able to create Windows 10 Apps and being able to port to Windows phone / tablets vice versa without having to rewrite any code (Wikipedia, 2016). This makes it easier to support a number of screen sizes and devices regardless of interaction type (touch, mouse/keyboard) as well as saving time rewriting code, meaning apps can work the same way on multiple families of devices with different form factors (Hale, 2016).

Another advantage of UWP is that it's flexible in the approach the developer takes in implementation supporting an array of different languages such as C# or Visual Basic with XAML, JavaScript with HTML or C++ with DirectX. This allows developers of different skill levels to work on apps for a wide range of devices.

UWP applications are also distributed and updated using the Windows Store in the same shared method, meaning that updates are applied to devices in the entire family instead of having to individually modify the application for all devices (Neagu, 2016).

However, UWP is also limited in that it in addition to being unable to run on Android and iOS devices, it is only compatible with devices running Windows 10. Which is currently only 8.8% of all Windows Phones on the market today making it a very small target audience (Saleem, 2015).

### 3.4.4 Final Decisions

Finally with all points considered **Android** was the chosen platform of development.

With Xamarin being the second most favoured platform, as although its ability to port to other platforms easily is a great feature along with having previous understanding of C#. Access to an Android device and the development tools aided the final decision, along with the challenge of using a new programming language to develop with and expand on the existing knowledge of Android and its Operating System.

In addition to a lack of a device that runs iOS or XCode, developing with iOS was not appealing due to its closed source software and lack of market share. Windows Phone and UWP also fell short due to its limited OS and access to wide range of features compared to other platforms.

## 3.5 Processes and Methodology

With the platform of development chosen, research was also needed to be conducted for some of the most technical elements and major features of the project before any implementation could take place.

### 3.5.1 Online Database

In order for the app to be portable and enable use on multiple devices, an online database must be implemented to handle User accounts and their data.

#### 3.5.1.1 Parse

Parse is a company owned by Facebook which provides mobile developers with an API capable of handling local & online databases without any backend setup required. It handles data securely and efficiently in the cloud making server-side logic simpler for any platform (Parse, 2015).

- |                          |                               |
|--------------------------|-------------------------------|
| ✓ Free                   | ✗ No Customization            |
| ✓ Simple API             | ✗ Limited requests per Minute |
| ✓ Good Documentation     |                               |
| ✓ Multi-Platform Support |                               |

Parse allows developers to handle multiple apps each with multiple tables for either data or User Accounts. It's also possible to store Files in the database directly unlike other databases such as SQLite, meaning user recipe images easier to handle. It also supports 'relation' between tables which would mean easier handling of multiple tables connected to one user, for example, the Shopping List and the users saved recipes.

#### 3.5.1.2 Firebase

Firebase offers similar features to Parse and is currently owned by Google. It offers the same benefits of giving developer's access to an Online and Local database without any backend setup required. However, the main difference is its use of JSON to format data back and forth with the app allowing real-time cloud data service (Firebase, 2016).

- |                          |   |
|--------------------------|---|
| ✓ Free                   | ✗ No Push Notifications                     |
| ✓ Real-time Database     | ✗ Limited requests per Minute               |
| ✓ Multi-Platform Support | ✗ JSON architecture will need time learning |
| ✓ Good Documentation     |   |

However, Firebase also allows automatic authentication to most sites including Facebook, Twitter, Google etc. meaning less work developing a back-end, unlike Parse. This could also make connecting user accounts with the

project easier by allowing them to use their existing Facebook/Google accounts to supply information instead of creating new tables and setting up a creation of account page.

### 3.5.1.3 Custom Backend



The last option considered was creating a backend from scratch which would be managed and developed by hosting self-created code. The benefits primarily would be having complete control of the data that is passed to the server and back as well as the amount that can be used. There are no limitations that have to be considered unlike using an API service like Parse/Firebase.

- |  |  |
|--|--|
| ✓ Fully customizable                     | ✗ Basic skill level with server-side development |
| ✓ Option for any Language/framework      | ✗ Extra time needed for research                 |
| ✓ No limitations on requests or features | ✗ Additional costs for hosting server            |
| ✓ Learn more about back-end development  |  |

Another advantage of having a custom backend is eliminating the possibility of an API service shutting down its services and leaving users to have to migrate as the developer manages everything themselves. Despite the freedom in implementation, it would require a lot of research and extra funds to make possible.

### 3.5.1.4 Verdict

Overall Parse was selected to be the service that would be used to host the Online Database in this project as although Firebase was a service which provided similar features, its documentation was not as extensive as Parse's meaning more time would have to be spent on implementation. Which is also why choosing to create a custom backend was not selected as this project has a limited time frame and in addition to having to learn about mobile development, Parse was the most time beneficial option.

## 3.5.2 Online API

In order for the project to meet the objective of providing a searchable external database of recipes, an API service which provides access to recipes would have to be selected.

### 3.5.2.1 BigOven



One of the most popular services available to developers is BigOven. Their Kitchen Cloud platform follows REST-best standards and is delivered in either JSON or XML format and then can be parsed and modified to the users liking allowing developers to take information that only needed for use in the project (BigOven, 2016).

- |                           |  |
|---------------------------|--|
| ✓ 350,000+ Recipes        | ✗ Knowledge of JSON required                     |
| ✓ Reviews of Recipes      | ✗ Steps in recipes are occasionally not accurate |
| ✓ Ingredients Provided    | ✗ Recipe format can be inconsistent              |
| ✓ Steps Provided (Mostly) | ✗ Average Documentation                          |
| ✓ Images of recipes       |  |

BigOven's minimum pricing is usually \$99 per month, however for the purpose of this project they distributed a Key free of charge for a year. The only downsides from using BigOven's database is due to the fact that its recipes are all user uploaded, there isn't one format that is conformed to in terms of ingredient measurements and steps. Some recipes contain links to websites for example where the steps can be found instead of them being listed in that particular recipe. Ingredient measurement units are also inconsistent for example ml is sometimes written as Millilitres or m/l etc. meaning future difficulties could be encountered when parsing the recipes for the project.

### 3.5.2.2 Spoonacular

Spoonacular is another service that provides recipe API to developers although less popular than BigOven. It provides the same basic functionality of searching for recipes along with additional features like the ability to search via ingredients and giving more information like Nutrition analysis and cost breakdown of ingredients. Spoonacular also provides the ability to extract recipes from websites and convert ingredients into specified formats (Spoonacular, 2016).

- |                                 |                                     |
|---------------------------------|-------------------------------------|
| ✓ 330,000+ Recipes              | ✗ Documentation very limited        |
| ✓ Large number of features      | ✗ Recipe format can be inconsistent |
| ✓ Search by multiple parameters | ✗ Smaller Community                 |
| ✓ Nutrition information         |                                     |

Spoonacular's has a freemium pricing strategy that starts at \$0 and then increases if more additional features are required, although they also would provide an academic key giving full access similar to BigOven. Spoonacular's downsides were the format of the recipes that were stored in the database were also inconsistent in terms of ingredients and steps making it difficult to parse for the project.

### 3.5.2.3 Yummly

In addition to Yummly's App, they also provide a recipe API to developers. They currently have the largest database of recipes aggregated from several different sources online. The API is tailored to be personal to the user introducing new concepts such as searching via fat, calorie, carb count etc. to promote weight loss and filtering searches via dietary preferences too (Yummly, 2016).

- |                                    |                                    |
|------------------------------------|------------------------------------|
| ✓ 1,000,000+ Recipes               | ✗ Recipe Step format not hardcoded |
| ✓ Tailored Search Results          | ✗ Knowledge of JSON required       |
| ✓ Large informative Documentation  |                                    |
| ✓ Ability to Share to Social media |                                    |

Yummly provides custom plans to tailor its API for its users but also were willing to provide a Key for Academic use for this project. Yummly's major disadvantage was the lack of hard-coded Steps in its Recipe JSON data, providing only a website link in which you'll be directed to the site in which the recipe is sourced from. Whereas all other recipe data is in a very consistent format which contains a lot of information including nutrition which can be parsed very easily.

### 3.5.2.4 Verdict

Overall BigOven was the API that was selected for this project. It has the second largest selection of recipes and basic documentation to get started with. Yummly overall would have the best API if it wasn't for its unique implementation of recipe steps, having to develop a built-in webpage in which users would have to navigate to in order to read steps. This would disconnect users from the App and more importantly inhibits the use of reading recipes offline and saving to the local database.

Spoonacular and BigOven were very similar in design with Spoonacular having more features available for use, but with the limited time frame for this project in-depth documentation of the API was deemed a more important factor.

## 3.5.3 Hands-free Interaction

In order to implement a hands-free way of communicating with the app, research was conducted on the available hardware in a modern smartphone and how it could be used to detect a command given by a user and then perform a task from the application. More specifically hardware on the front facing side of the device in order to trigger a response, as when in a cooking environment the device will most likely be propped against a stand or on a surface being read from.

### 3.5.3.1 Ambient Light Sensor

The Ambient Light Sensor works by constantly measuring how bright the ambient light around the phone is. Its primary use is normally for adjusting the display's brightness automatically in relation to the outside light in order to save battery life and prevent eye strain (PhoneArena, 2014). For example, if there if the luminosity is low then the sensor will detect this and lower the brightness as less is needed in a darker environment and vice versa. Utilizing this for the project for hands-free interaction would require the user to cover the sensor on the front of the phone, stopping light from being detected by the sensor and therefore triggering a method that for example would read out the next step of the recipe currently being viewed (**Figure 13**).

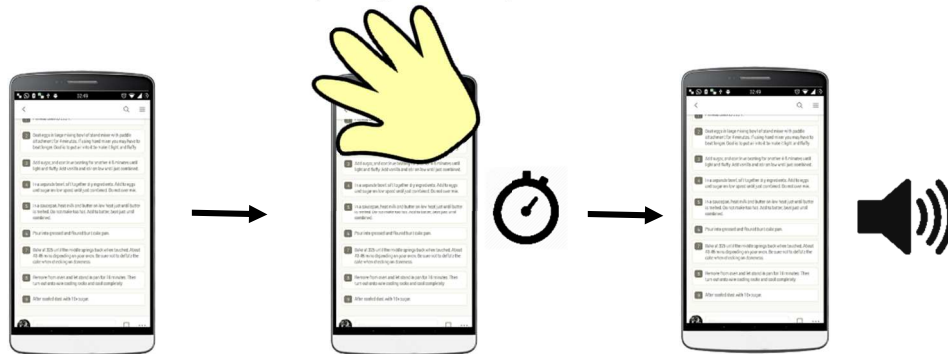


Figure 13: Ambient Light Sensor Example.

### 3.5.3.2 Proximity Sensor

The Proximity Sensor works by utilising an infrared LED constantly illuminating an infrared beam of light invisible to human retinas and an IR light detector. This beam is measured by returning a distance value returned by the IR light detector when the infrared beam is reflected back from a surface covering it. Its primary use is usually to turn off smartphone displays when users are taking an incoming call to prevent unwanted interaction with the device (PhoneArena, 2014). For example, when the phone is next to the user's head the distance returned will be very small triggering the smartphone to turn off the display until normal distance returns. Utilizing this for the project would require the sensor to detect a change in distance in short periods of time to catch a hand wave motion over the device. From here the sensor could trigger a method that for example would read out the next step of a recipe (**Figure 14**).

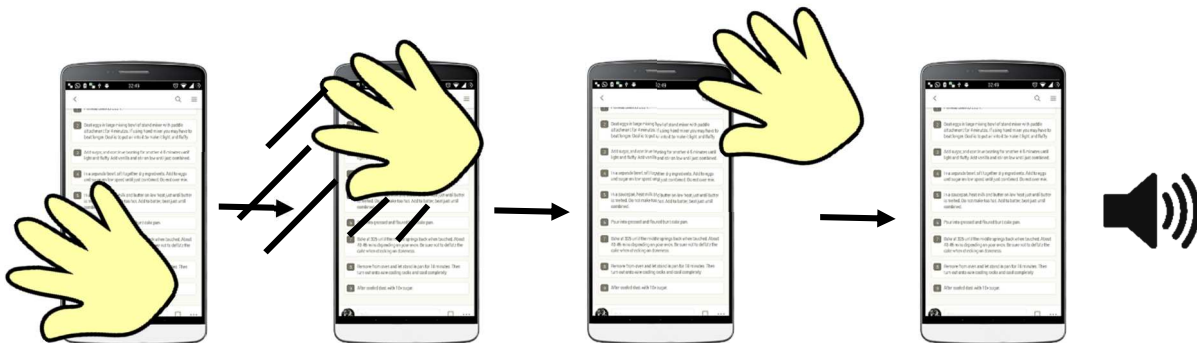


Figure 14: Proximity Sensor Example.

### 3.5.3.3 Verdict

Overall the proximity sensor was selected as although both would work for the application, the ambient light sensor had some foreseen complications. The first being conflicting data with the Android OS which also uses the ambient light sensor constantly, causing the system to prioritise the hardware for the application. Limiting the screen brightness adjustment feature when using the app. Another complication was that in order to reduce luminosity the user's hand would occasionally have to cover the sensor completely which if their hands were dirty causes inconvenience in comparison to a simple wave over the device to activate the proximity sensor.

## **3.6 Potential Problems**

### **3.6.1 Interaction with SQLite and Parse Database**

A foreseen problem is the interaction between the local SQLite and Parse database so that recipes are only displayed specific to the user that's logged in when the application has no access to an internet connection. As in order to check the current user, Parse connects to the database online.

### **3.6.2 Recipe Format**

Another problem is how to deal with the inconsistency of recipes downloaded from the API database. As there are various different ways to represent quantities such as time and quantity, making sure that the application parses the data correctly into a consistent format similar to the applications might cause difficulty.

## 4. Technical Development

### 4.1 System Design

#### 4.1.1 Class Diagram

The Class diagram (**Figure 15**) describes the basic design of the project before major implementation took place and how each class relates to each other in the application. This was to show simple app design as not every feature had been decided on and basic functionality was seen as providing management of user accounts, recipe data and providing the ability to create and search a recipe.

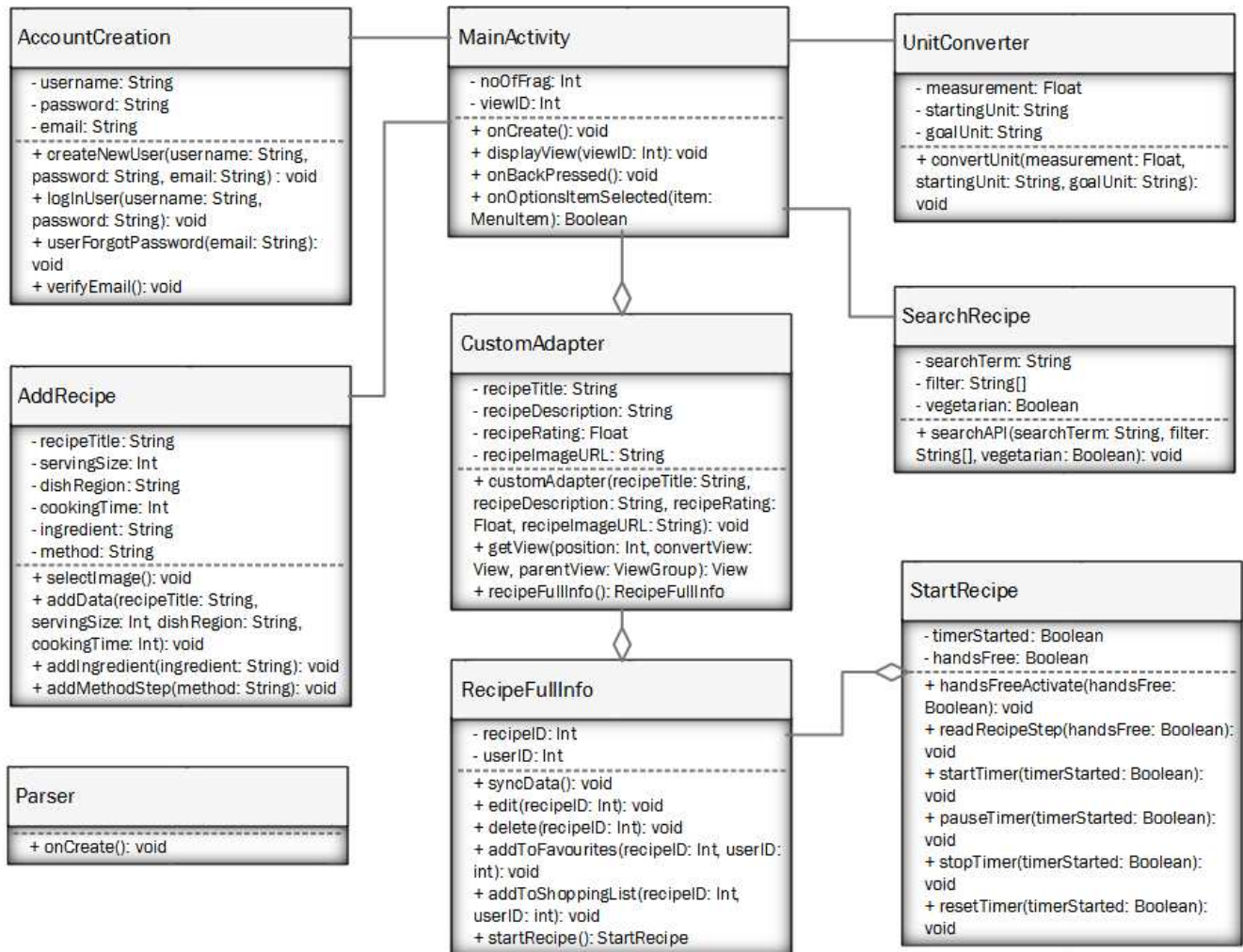


Figure 15: Project Class Diagram

The main Class is MainActivity which contains the home screen and the navigation drawer. Due to this, the majority of other fragments (Pages) are accessed from this Class. With AccountCreation, UnitConverter, SearchRecipe and eventually all other navigation drawer options being accessible as a fragment that is started from the drawer itself and inflated over the MainActivity.



### 4.1.2 Use Case Diagram

The Use Case diagram (**Figure 16**) lists steps a system needs to follow in order to reach a certain goal. For this project there are three actors in the system. The first being the user who is interacting with the app and the remaining being the databases that deal with user accounts and their data (Parse) & searchable external recipes (API).

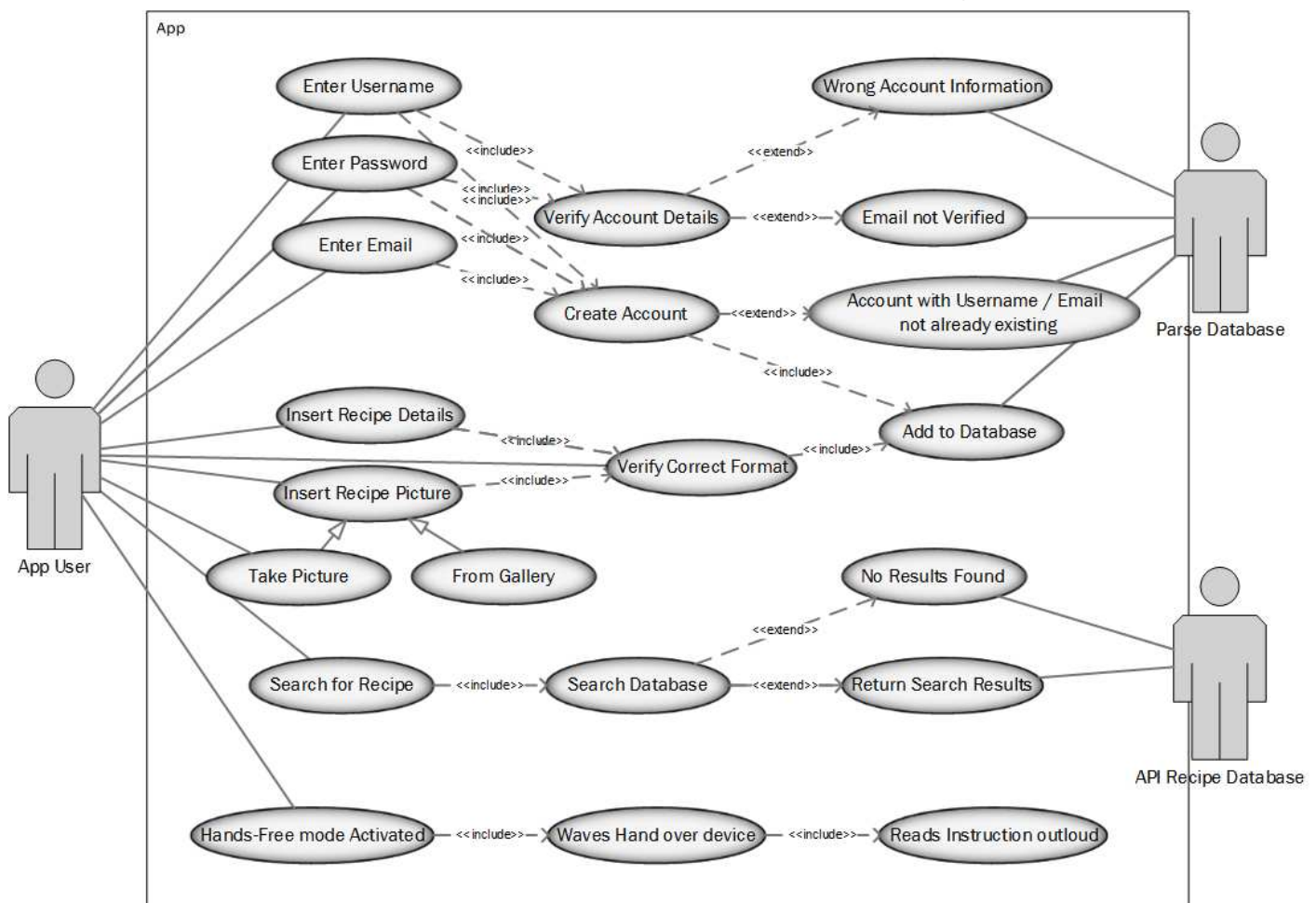


Figure 16: Project Use Case Diagram

The triggers for this diagram are:

- "Create Account" or "Log In" buttons are pressed.
- "Create Recipe" Button is pressed.
- "Search Recipe" Button is pressed.

The Goals for this diagram are:

- Create / Log in User Accounts securely.
- Add recipes to database in the correct format.
- Return searched recipes quickly and accurately.

The Alternatives for this diagram are:

- If Account is already existing → Display message asking to amend entered details.
- If Account has an unverified email → Display message asking to check their inbox before logging in.
- If Recipe details are in an incorrect format → Display messages showing the correct accepted format for the fields.
- If the Search for a Recipe yields no results → Display a message asking to search again.

The preconditions for this diagram are:

- Secure Connection to Internet.
- Correct details have been entered to Account Log In / Account Creation Screen.
- Account with entered Username / Email not already existing in Database.
- Email has been activated attached to account.
- Correct format of information have been entered into recipe creation.
- Recipe searched for is stored in API recipe database.



The Use Case diagram was useful for understanding how the 3 main components which cannot be fully controlled will interact with each other and what would need to be thought about during implementation in order to deal with as many situations as possible.

#### 4.1.3 Program Layout

The architecture of the application was overall designed to utilize one Activity which contains an empty FrameLayout and then to inflate each Page on top of that Layout. Then by utilizing Androids Fragment Manager and specifying how the back-stack is managed, users could repeatedly open different pages without causing as large of a background memory footprint as there would be if each page was its own activity (**Figure 17**).

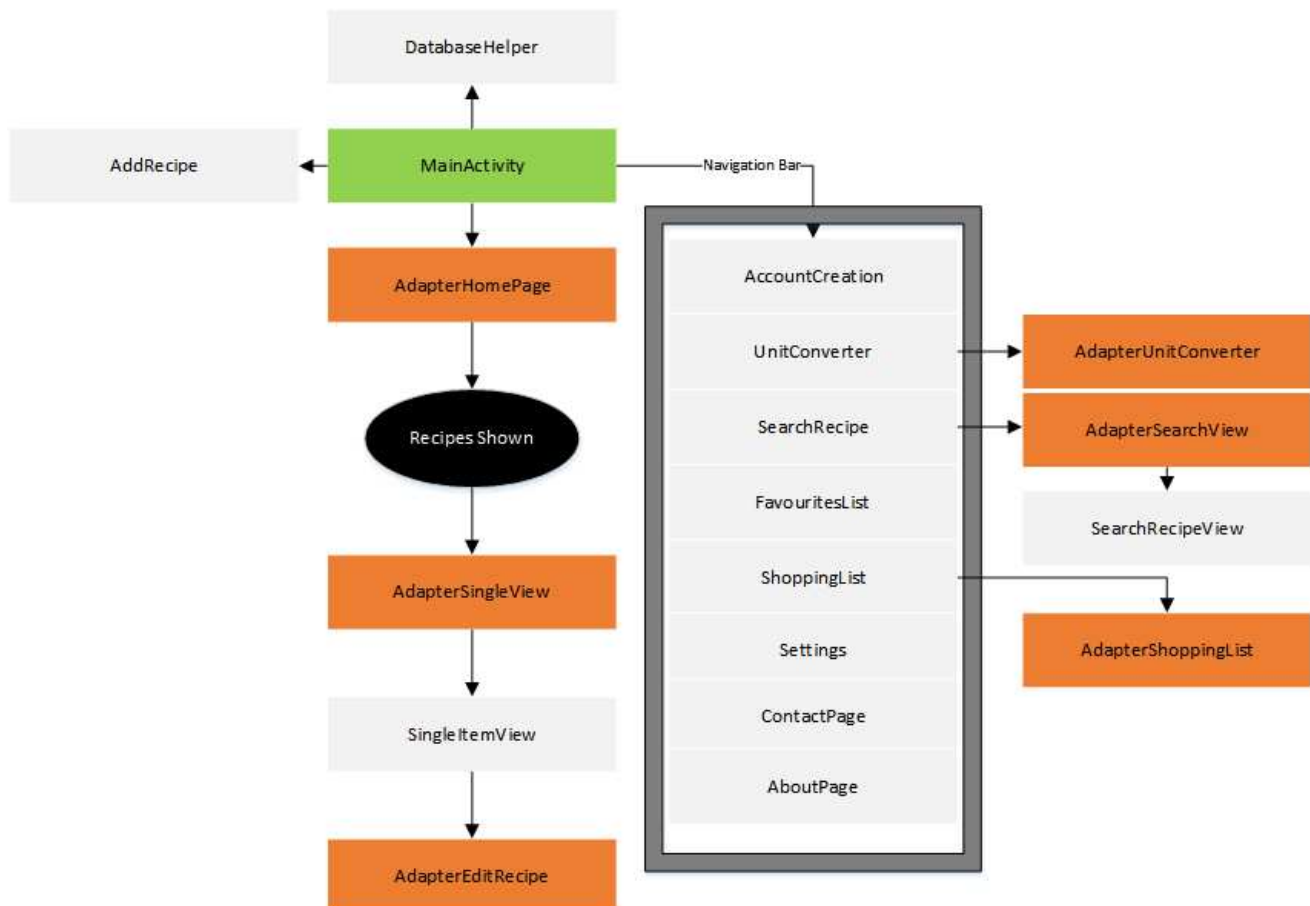


Figure 17: Project Architecture

The defined management of the back-stack had to be declared otherwise expected interaction would not have taken place. The implementation allows for pages to be opened once and then remember their place in the stack so that when the "Back" button is pressed it will navigate to the previously opened fragment, or return to the MainActivity if that fragment isn't present. The back-stack also detects if a page has been opened previously, so multiple selections of the same page will not pile up fragments of the same page. These features are both useful when navigating the application as it adheres to the specification of handling fragments, stated in the Android Documentation. Meaning that users will be familiar with the navigation if they have used other applications that also adhere to this format.

#### 4.1.4 API Design

The design of the BigOven API recipe lies in the interpretation of the JSON data returned when either searching for a particular dish or opening a recipe from a query. Understanding of JSON was required before any parsing could begin in the application but the basic concept was that data is stored as either an Object (Curly Brackets) or as an Array (Square Brackets) and that both data types can be combined i.e. an Array of Objects. Inside these data types are Key:Value pairs of information.

When searching for a recipe, only basic information about each dish was returned like the title, category, and rating (**Figure 18**). So in order to create a list of results in the application, the design was to store only data that seemed necessary to display to the user about that particular dish

```
{
  "ResultCount": 77815,
  "Results": [
    {
      "RecipeID": 432218,
      "Title": "Pistachio Cinnamon Chicken Salad",
      "Cuisine": null,
      "Category": "Salad",
      "Subcategory": "Meat and Seafood",
      "Microcategory": "",
      "StarRating": 4.5,
      "WebURL": "http://www.bigoven.com/recipe/pistachio-cinnamon-chicken-salad/432218",
      "ReviewCount": 2,
      "Poster": {
        "UserName": "darrinrich",
        "UserID": 1721301,
        "FirstName": null,
        "LastName": null,
        "PhotoUrl": "http://photos.bigoven.com/avatar/photo/hwdnrvpumeathey2mfj.jpg"
      },
      "IsPrivate": false,
      "Servings": 6,
      "CreationDate": "2013-01-23T16:30:26.000Z",
      "IsRecipeScan": false,
      "IsBookmark": false,
      "TotalTries": 389,
      "PhotoUrl": "http://photos.bigoven.com/recipe/hero/pistachio-cinnamon-chicken-salad.jpg"
    },
    {
      "RecipeID": 171326,
      "Title": "Skewered Honey-Balsamic Chicken",
      //Additional information like previous.
    },
    //Repeated for each Recipe that matches query.
  ]
}
```

Figure 18: JSON result for Search Query

The data selected was PhotoUrl, StarRating, Title, Cuisine, and Category. These were chosen as they were the data values that described the recipe the most without providing clutter to the display like if the recipe creators account details were included. A mockup design of this can be seen in **Figure 19**.

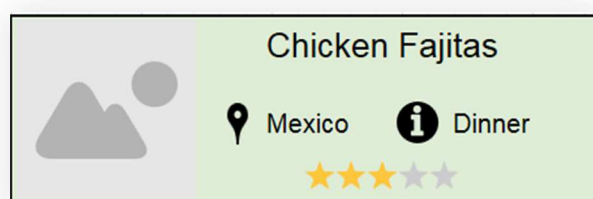


Figure 19: Searched Row Example.

When opening a recipe from the queried list, the JSON returned was similar to the one seen when searching but provided a lot more information about the recipe including an array of ingredients which listed not only the quantity but also the nutritional details. The design for this was to utilize everything about the recipe but leave the nutritional information, miscellaneous details such as LastModifiedDate and Poster information. The choice to leave out nutritional information was mainly due to the unpredictability of recipes including that data, leaving a large majority of dishes showing null information. **Figure 20** shows a mock-up design of how the chosen parsed data would be utilized.

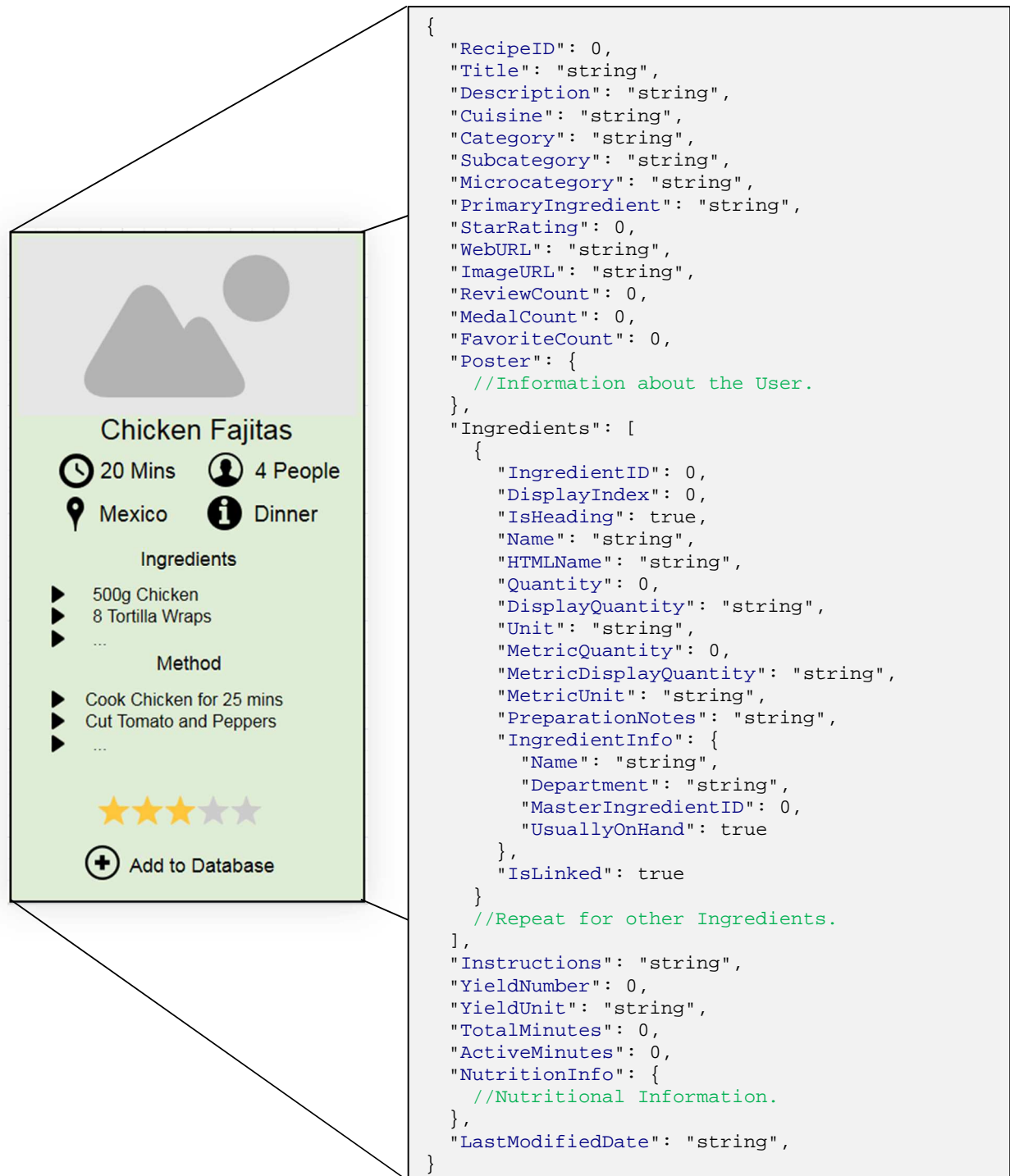
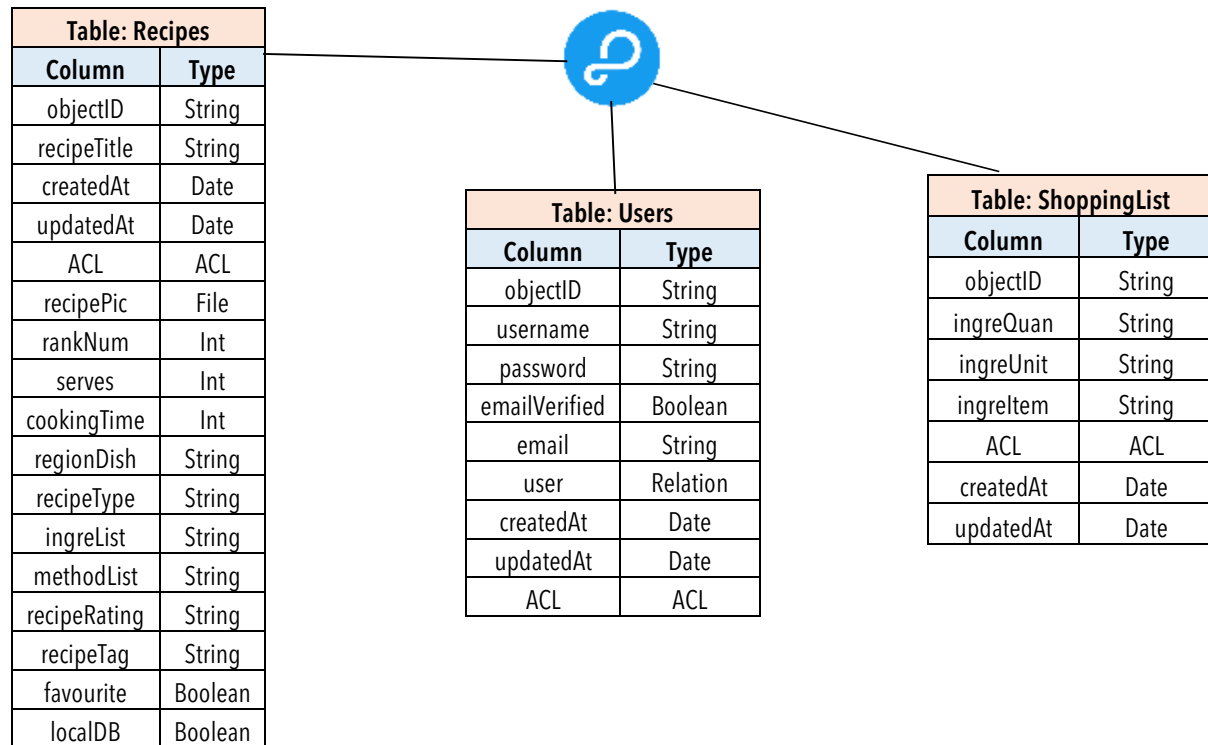


Figure 20: JSON result for Recipe and Design.

#### 4.1.4 Database Design

The design of the Parse database consisted of having three tables. The first storing the User account information, the second keeping all user created / saved recipes and the third having all user ShoppingList's. All of these tables were designed with the idea of reducing redundant data and implementing normalization techniques attained from the University of Hull's Data Mining module. Normalization is the process of organising attributes and tables to minimize data redundancy (Beaubouef, et al., 2011) (Hillyer, 2005). An example of this was storing attributes like CookingTime as an integer instead of a float as a large majority of recipes would not require a cooking time to the exact second, as most are given as a rough estimate. This helps towards keeping the table's dimensionality to a minimum. The overall layout can be seen below.



Each row in every table has a unique randomly generated ID assigned to them, making it simple to perform queries for a specific Recipe or ShoppingList item. Parse also generates other attributes like createdAt and updatedAt, allowing the functionality of sorting items by last created or last updated a lot simpler than having to manually input the date/time.

However, the attribute which has provided the best functionality throughout the implementation was the ACL or Access Control List seen in **Figure 21**. It allows developers to specify which users are able to Read and Write to objects in Tables. This was used for handling the syncing of recipes and shopping lists from the database by querying objects who's ACL matches the User that is currently signed into the application. Then to complete this functionality all that was needed was to update the ACL of any created or externally saved recipe to the current User ID and then it would be tied to that account indefinitely.

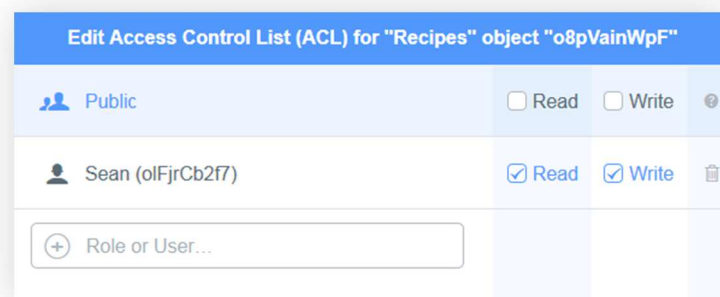
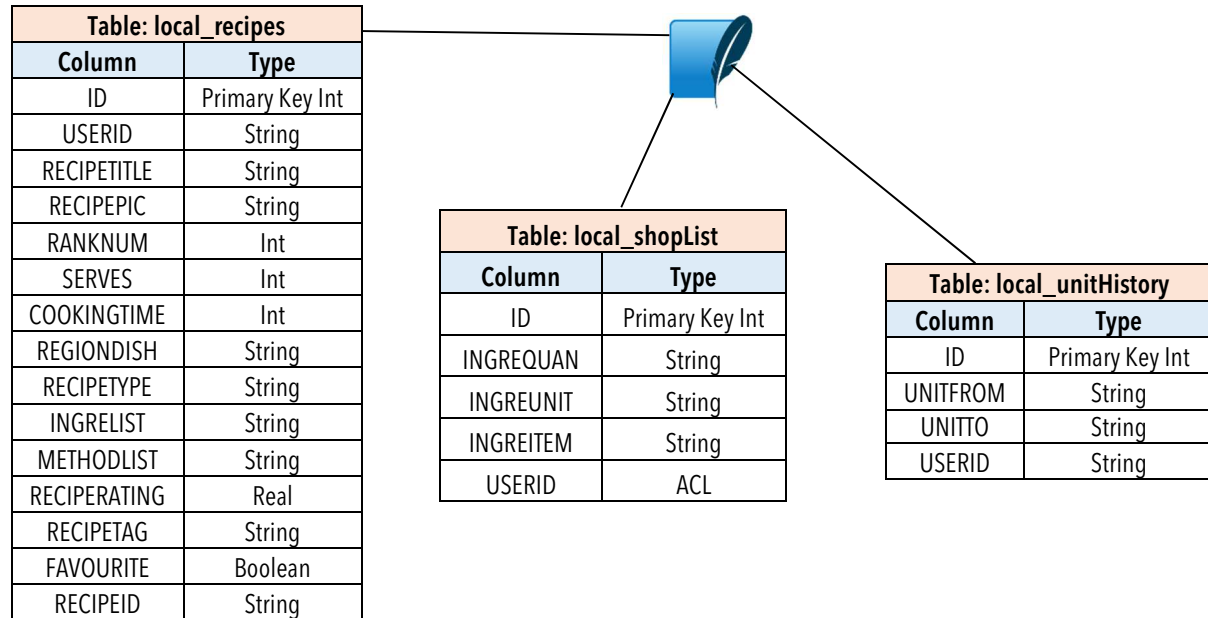


Figure 21: Editing ACL for an Object.

The design of the local database was implemented using SQLite also consisting of three tables. The first storing local recipes on the device, the second storing local shopping lists, and the third acting as temporary storage of any unit conversion history a user might have during a session of using the application. Again these tables were designed with normalization techniques in mind, providing users with access to their data when no available internet connection is present.



No foreign keys have been utilized in this database as none of the tables are related to each other or need access to each other at any point when using the application. Instead, when each object is being saved to any of the tables the UserID is stored by calling a method that stores the current user's ID that is signed in. Then whenever data from the tables needs to populate a view, the current user ID is passed as a parameter to retrieve data that matches that string.

#### 4.1.5 Coding Convention

Due to the size of this project, it was important to adhere to correct coding techniques to ensure readability and understanding for present and future development.

Almost all Java conventions were implemented into this project to allow for other developers to be familiar with the layout, handling and naming of variables if the application ever expanded or requires additional help. Some examples being to not ignore exceptions or catch generic ones. Another convention is providing Javadoc comments to every file or major method, unfortunately due to time constraints this project has limited comments and would be something that would be refined later on (Google, 2016).

Another technique was packaging Classes and Layout files into separate directories using an Android Studio plugin. It makes understanding the projects structure easier without creating new folders or actually moving any files (Dmytro, 2015) (**Figure 22**).

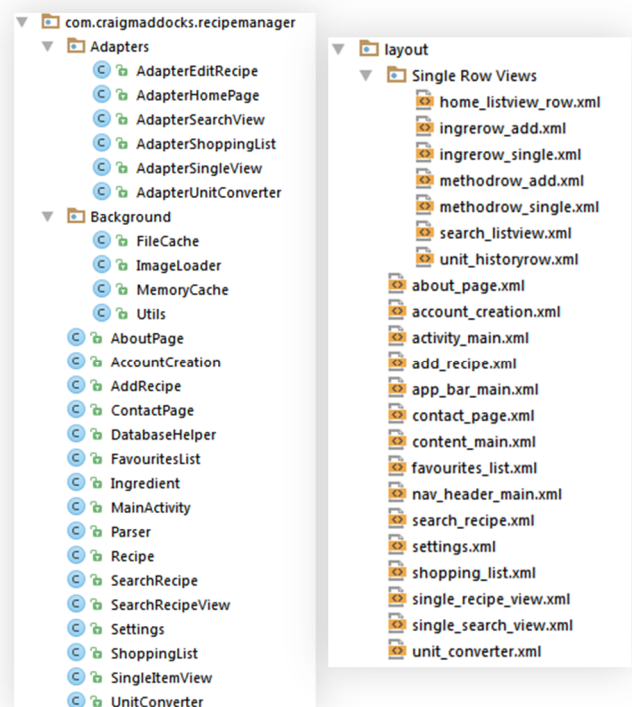


Figure 22: Class and Layout File Structure of Project.

## 4.2 UI Design

The UI design for the application was purely digital and no paper-based drawings were created. This was due to time limitations but mainly because the developer had more experience in using design software and found it easier to portray ideas through these methods. The UI designs goals were simplicity and minimalism. The design reflects this as all key features are accessible via the main home screen or the navigation drawer. It also adheres to Androids UI Design standards meaning that it should provide no difficulty in use for the average user (Figure 23).

One of the key affordances in the design is the recipe creation page where if users want to add ingredients / Steps they can press the Plus icon and a new editable text row will appear extending the page in a ScrollView so that they can keep adding items. Additionally, in the Accounts page when the user selects Create Account, new fields will appear asking the user to enter an email along with the button text changing to "Create Account".

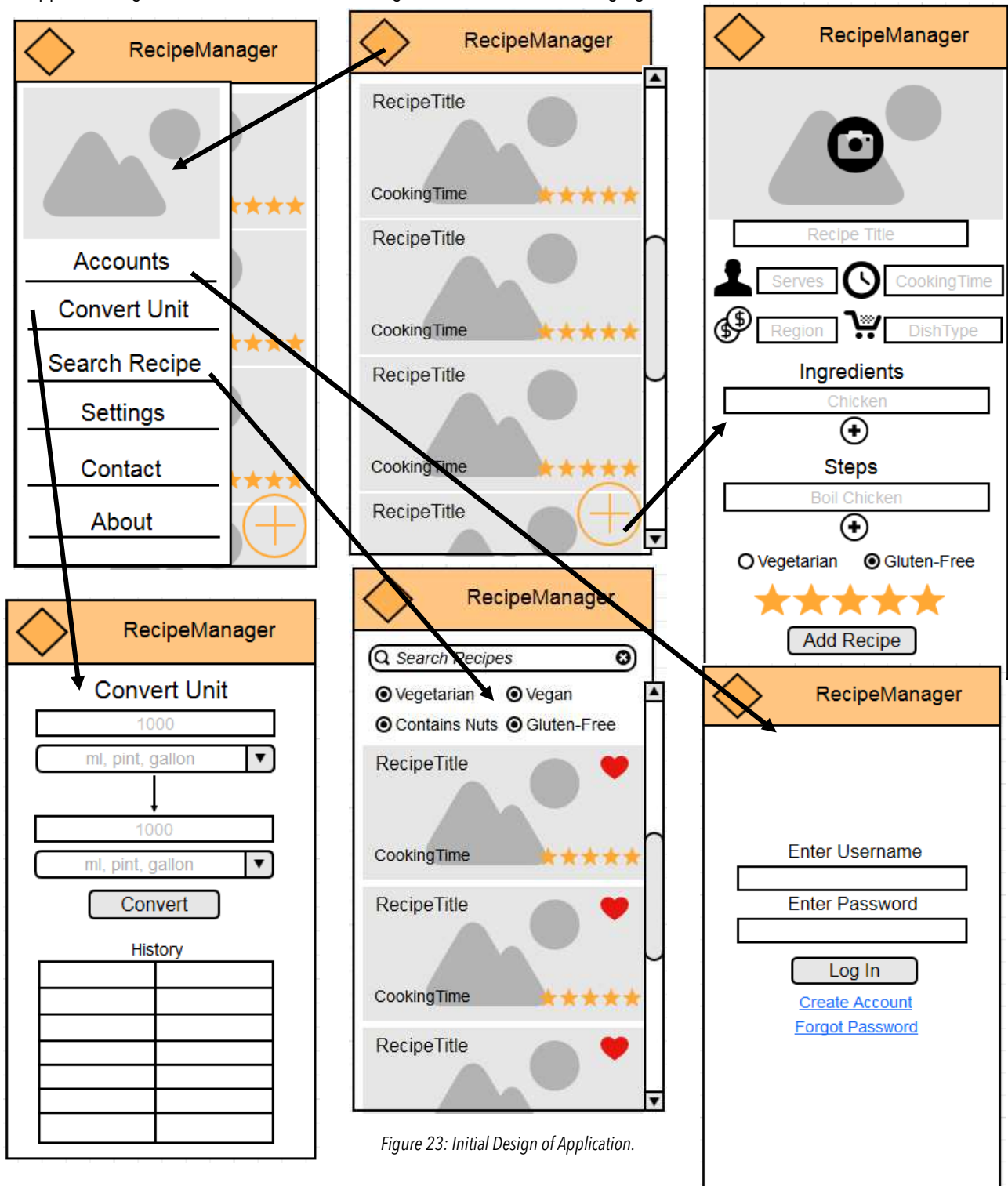


Figure 23: Initial Design of Application.



As this was the initial design, not all features had been established and so were not included along with the decided colour scheme, logo, and name of the application. As this was the developer's first application, the designs were trying to provide a good aesthetic but more importantly could be feasibly created in the time period given. The only major differences from the UI Designs and the end project was the radio buttons to search by dietary preference and the way recipe tags were entered.

**Figure 24** is the finalised design of the project including logo and primary colours used throughout. Overall the project followed the initial design well and made improvements on areas where screen space was empty along with meeting standards of Android and overall UI design.

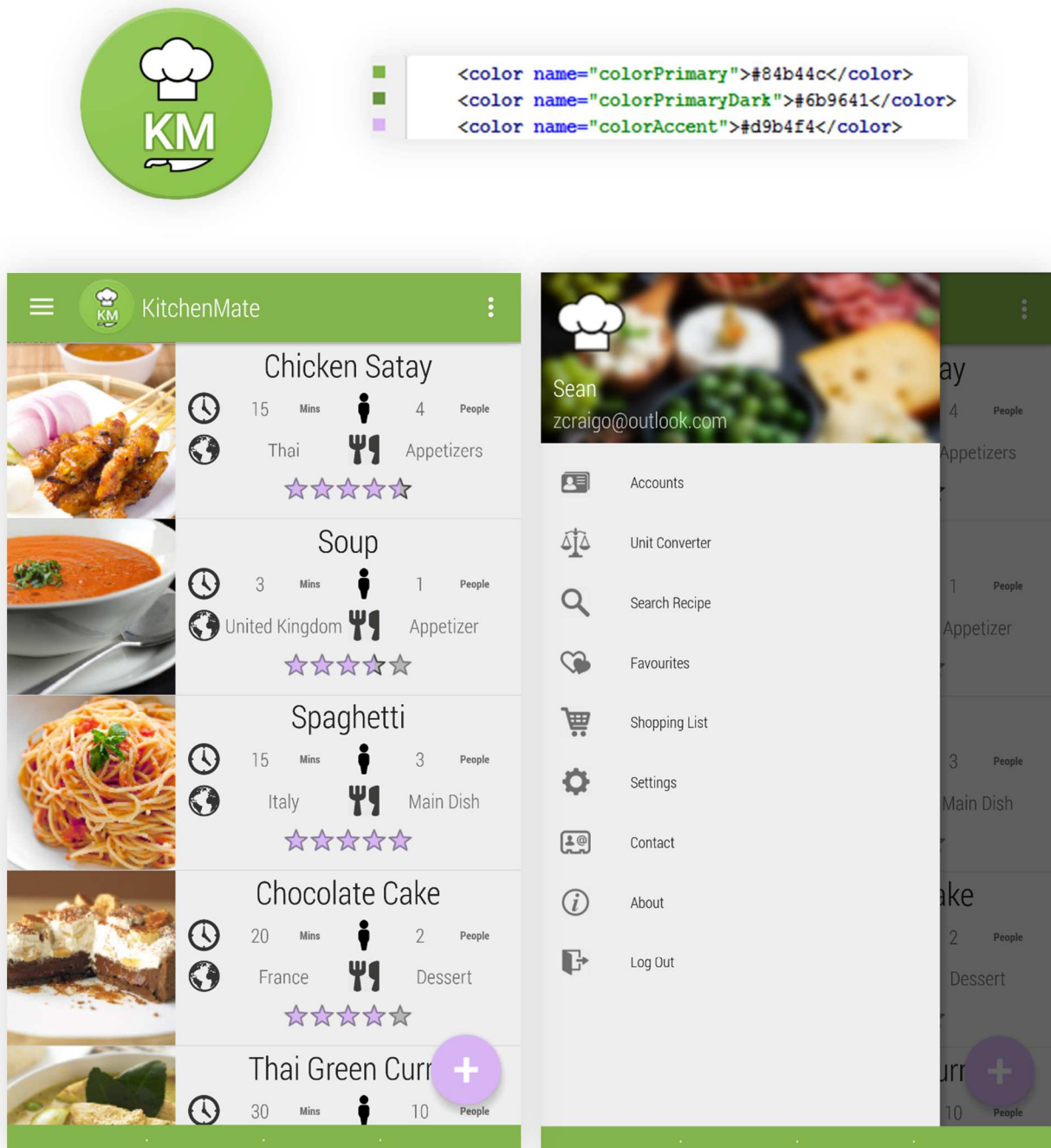


Figure 24: Finalised Design of Application.

## 4.3 Test Design

### 4.3.1 Unit Testing

Unit testing is where the smaller testable methods of an application are independently subjected to automated or manually inputted queries to see if they will behave as intended or break during use (Link, 2003). This project mainly implemented Unit Testing in areas of the application where a large variation of user input could occur for example the AccountCreation and RecipeCreator pages. As only certain text would be valid for use in the databases like Usernames being less than 12 Characters etc. Unit testing made validating these factors easier than performing individual Acceptance tests.

Local Unit Tests were the only type of unit testing implemented in this project through the use of the built-in platform JUnit. It's automatic testing framework made generating and running scripts painless and quicker than manually inputting queries into fields, closing the application and repeating. This is due to JUnit testing being able to run on the local machine testing small methods without the use of an emulator making testing quicker once the tests have been initialised first. **Figure 25** shows an example of the JUnit tests implemented in the project, these tests pass a different combination of input that could be entered on the device into a method that checks whether or not it is valid for that Field. The assertTrue will make sure the True Boolean value is returned and vice versa.



```
public void testUsername() {
    assertFalse("Blank Username", AccountCreation.validUsername(""));
    assertFalse("Username with special char", AccountCreation.validUsername("@Craig^&"));
    assertFalse("Username with underscores", AccountCreation.validUsername("_Craig"));
    assertFalse(">12 char Username", AccountCreation.validUsername("VeryLongUsername"));
    assertFalse("Number Username", AccountCreation.validUsername("34354"));

    assertTrue("Alphanumeric Username", AccountCreation.validUsername("Craig94"));
    assertTrue("Normal Username", AccountCreation.validUsername("Craig"));
}

public void testEmail() {
    assertFalse("Blank Email", AccountCreation.validEmail(""));
    assertFalse("Email without ending", AccountCreation.validEmail("email@gmail"));
    assertFalse("Email with extra char", AccountCreation.validEmail("email@gmail.com"));
    assertFalse("Email with extra char", AccountCreation.validEmail("email@gmail..com"));
    assertFalse("Email without Start", AccountCreation.validEmail("@gmail.com"));

    assertTrue("Alphanumeric Email", AccountCreation.validEmail("craig934dfd@gmail.com"));
    assertTrue("Long Email", AccountCreation.validEmail("VeryLongEmailAddress89@gmail.com"));
}
```

Figure 25: JUnit tests for Username and Email Fields.

Instrumented tests are another subtype of unit testing which runs via an emulator which can be programmatically set up to perform tasks that Local Unit Tests cannot. This is due to instrumented tests having access to the apps Context and other information making tasks like testing UI or mock interaction which sensors possible. However, this was not implemented in the project as the developer found that acceptance testing was better suited to test features compared to writing tests to obtain feedback. Also with the limited time limit given for implementation, this was foreseen as a faster method. (Google, 2016).



### 4.3.2 Acceptance Testing

Acceptance testing is where actions are carried out on the system manually to check whether it has met the intended functionality, for example swiping the screen from the left to the right will display the navigation bar. This form of testing was the primary method used throughout the course of the project, after implementing a feature it was immediately checked via emulation and performing acceptance tests. The developer found this very useful as it provided immediate feedback about how a feature not only performs but how it interacts with the entire environment, this helped to support later design decisions throughout the application.

Test	Segment of App	Pass/Fail
Recipes are added to the Favourites List when Heart is selected.	<i>Edit View</i>	<i>Pass</i>
Ingredients are added to the Shopping List when Selected.	<i>Full Recipe View</i>	<i>Pass</i>
The timer starts and finishes at the right time when used.	<i>Full Recipe View</i>	<i>Pass</i>
Text-To-Speech reads Steps out in order.	<i>Full Recipe View</i>	<i>Pass</i>
Waving Hand over phone starts next step to be read out aloud.	<i>Full Recipe View</i>	<i>Pass</i>
Pressing Back button navigates to the correct previous page viewed.	<i>All</i>	<i>Pass</i>

Above is a segment taken from the full testing table seen in Appendix E. Every feature within a page in the application was subjected to tests in both orientations to confirm operational status.

Executing Acceptance testing became an iterative process that took place during every change in relevant code, tests would need to be repeated in addition to performing the tests each new feature would bring. This is to make sure that new additions are working correctly but that they also do not negatively affect the functionality that was already in place.

### 4.3.3 Testing Environment

The application has been tested in several different environments emulated via Android Studio and on the developer's personal device. This was to check that functionality and layout was still preserved and that user experience was not affected. This preservation was mainly achieved by designing views using RelativeLayouts or by the use of the Weight parameter in XML objects to define how the layout should look *relatively* in comparison to the device instead of manually entering values for size. The devices used to test this are listed below.

Device	Screen Size	Screen Resolution	Operating System	Pixels per Inch (PPI)
LG G3	5.5"	1440 x 2560	CloudyG3 2.5 (Android 5.0 Lollipop)	538
Nexus 5	4.95"	1080 x 1920	Stock (Android 6.0 Marshmallow)	445
Nexus 10	10.1"	2560 x 1600	Stock (Android 6.0 Marshmallow)	299
Nexus 7	7.0"	800 x 1280	Stock (Android 5.0 Lollipop)	216

**Figure 26** shows how the application contrasts visually with phones and tablets of different screen resolutions, the views changing to adhere to different devices. The applications design was primarily for use with a smartphone but as seen can be extended for use with a tablet also. SmartTV's was a platform which was not tested/considered due to the audience being limited to a minority of users who own a device in their kitchen. Alternatively utilising Smartwatches to provide integrated functionality was considered but ultimately was not included due to time constraints.



Figure 26: Nexus 5, Nexus 7, Nexus 10 Emulating different pages.

#### 4.3.4 Review

Overall acceptance and personal testing with different devices provided the most help throughout the course of the project. Unit testing however, proved to help very little due to it being implemented during the end of the project lifecycle retesting elements of the project that were already put through acceptance testing. Unit testing would have been useful to the project if it was implemented from the beginning and throughout implementation instead of at the end, serving as confirmation of functionality. This would be something that the developer would change if the project was restarted.

## 4.4 Implementation

### 4.4.1 External Dependencies

Dependency Name	Company	Version	Use in Project
Parse	Facebook	1.13	Online Database which stores and handles User accounts, recipes and shopping lists.
Picasso	Picasso	2.5.2	Made transforming, editing and scaling of ImageViews quicker and easier with the ability to add placeholder images for when images are loading or if there was a connection error.

### 4.4.2 Parse Online Database

The Parse database was implemented firstly by adding the SDK to the applications Gradle file. This file acts as a build script that tells Android basic information about how the application should be run and which external APIs to include in the source code.

Calls to the Parse database are performed by using ParseQuery methods that are able to be filtered or ordered by a certain column or ID to produce an individual or list of objects. This is used throughout the project to find, update or delete recipes but its primary use is syncing pages across the app in order to populate a ListView and display the current user's recipes, display a list of recipes generated from a search query or display the current user's favourite recipes.

The default behaviour is to sort recipes in the order they were created or added to the database, however, this can be changed via the action menu by selecting a different parameter from the drop down list. This will then change a global variable so that when the list is synced again the order of recipes passed into the adapter is one the user selected (**Figure 27**).

From here the syncRecipes method is called. This AsyncTask populates an array list of recipe objects that are given data from the returned Parse Objects. Then using a BaseAdapter Class, this ArrayList is passed into the adapter which assigns the objects data to the corresponding field in the recipe row and returns a fully populated list of recipes (**Figure 28**).

```
@Override
public boolean onOptionsItemSelected(Item mItem)
{
    boolean mLocalsync = false;
    if(!(haveNetworkConnection())) {
        mLocalsync = true;
    }
    switch (mItem.getTitle().toString()) {
        case "Alphabetical":
            if(!mLocalsync) {
                mOrderList = "recipeTitle";
                new syncRecipes().execute();
            } else {
                mOrderList = "RECIPETITLE";
                new localSync().execute();
            }
            break;
        case "Most Recent":
            if(!mLocalsync) {
                mOrderList = "ranknum";
                new syncRecipes().execute();
            } else {
                mOrderList = "RANKNUM";
                new localSync().execute();
            }
            break;
        case ... // Continued.
    }

    return super.onOptionsItemSelected(mItem);
}
```

Figure 27: Shortened Method which handles Action menu.

These methods were used in similar ways to populate all lists throughout the app.

```

public class syncRecipes extends AsyncTask<Void, Void, Void> {

    protected void onPreExecute() {
        super.onPreExecute();
        // Create a Progress Dialog
        mProgressDialog = new ProgressDialog(MainActivity.this);
        mProgressDialog.setTitle("Your Recipes are being Synced!");
        mProgressDialog.setMessage("Loading...");
        mProgressDialog.setIndeterminate(false);
        mProgressDialog.show();
    }

    protected Void doInBackground(Void... params) {
        // Populate ArrayList with Recipes
        recipeList = new ArrayList<Recipe>();

        ParseQuery<ParseObject> query = new ParseQuery<ParseObject>("Recipes");
        if (orderList.equals("recipeTitle") || orderList.equals("regionDish") ||
        orderList.equals("recipeType")) {
            query.orderByAscending(orderList);
        } else {
            query.orderByDescending(orderList);
        }

        ob = query.find(); // Perform Search

        for (ParseObject recipe : ob) {
            ParseFile image = (ParseFile) recipe.get("recipePic");
            Recipe r = new Recipe();
            r.setRank((int) recipe.get("ranknum"));
            r.setRecipeTitle((String) recipe.get("recipeTitle"));
            r.setRecipeCookingTime((int) recipe.get("cookingTime"));
            r.setRecipeServes((int) recipe.get("serves"));
            r.setRecipeType((String) recipe.get("recipeType"));
            r.setRegionDish((String) recipe.get("regionDish"));
            r.setRecipeRating((String) recipe.get("recipeRating"));
            r.setRecipePic(image.getUrl());
            r.setObjectID(recipe.getObjectId());
            r.setRecipeFav((Boolean) recipe.get("Favourite"));
            recipeList.add(r);
        }
        return null;
    }

    protected void onPostExecute(Void result) {
        //Sets data to adapter.
        homeAdapter = new AdapterHomePage(MainActivity.this, recipeList, false,
        false);
        // Binds the Adapter to the ListView
        homeRecipesLSTV.setAdapter(homeAdapter);
        // Close the progressdialog
        mProgressDialog.dismiss();
    }
}

```

Figure 28: Async Task that retrieves all Online Recipes.

Parse was also responsible for the handling of user accounts in the application. Working in a similar way, utilising the ParseUser methods such as SignUpInBackground and LogInBackground for creating accounts and logging in. The main challenge here was confirmation the validity of data entered into edit text fields so that they adhere to the database requirements. Most other functionality was handled by Parse, including the ability to reset Account Passwords and only granting accounts which have been activated the ability to log in. Thus leaving more time available to work on other features in the project.

#### 4.4.3 SQLite Local Database

The local database was responsible for providing functionality when the application is unable to connect to the internet. An SQLiteOpenHelper class was used for the implementation for both the local recipes and shopping lists for users. Several basic methods were used which allowed Inserting, Deleting, Updating of items along with getting all or getting individual objects via a recipeID. However, the biggest functionality came from populating all pages containing ListView's when the app was offline.

Firstly the application checks whether it can connect to the internet either via WIFI or Data when a call to populate a ListView is made. This is done via a method that utilises the ConnectivityManager, this then attempts to retrieve network information about the phones WIFI and Data connection. If this network information returns data and is not null, then the device is online and will return true. However, if this method returns false, then a separate AsyncTask similar to **Figure 28** is performed where instead of making a call to Parse, a method within the SQLiteOpenHelper class getAllRecipesUser is called. This will perform in the same way by populating an ArrayList of Recipes taken from the SQLite database that matches the UserID given. The ability to specify the order of returned recipes is also possible in the same way by passing the global mOrder string and using the "ORDER BY" query in SQLite (**Figure 29**).

These methods were used in similar ways to obtain local data for user shopping lists and favourite recipes too, with only minor changes in the getAll methods in the Database class.

```
public ArrayList<Recipe> getAllRecipesUser(String mUserID, String mOrder) {
    ArrayList<Recipe> Recipes = new ArrayList<Recipe>();
    String ascOrdsc;

    if(mOrder.equals("RECIPETITLE") || mOrder.equals("REGIONDISH") ||
mOrder.equals("RECIPETYPE")) {
        ascOrdsc = "ASC";
    } else {
        ascOrdsc = "DESC";
    }

    String selectQuery = "SELECT * FROM " + TABLE_NAME + " WHERE "
        + COL_2 + " = '" + mUserID + "'" + " ORDER BY " + order + " " +
ascOrdsc;

    SQLiteDatabase db = this.getReadableDatabase();
    Cursor c = db.rawQuery(selectQuery, null);
    if (c.moveToFirst()) {
        do {
            Recipe p = new Recipe();
            p.setUserID(c.getString((c.getColumnIndex(COL_2))));
            p.setRecipeTitle(c.getString((c.getColumnIndex(COL_3))));
            p.setPicLocal(c.getString((c.getColumnIndex(COL_4))));
            p.setRank(c.getInt((c.getColumnIndex(COL_5))));
            p.setRecipeServes(c.getInt((c.getColumnIndex(COL_6))));
            p.setRecipeCookingTime(c.getInt((c.getColumnIndex(COL_7))));
            p.setRegionDish(c.getString((c.getColumnIndex(COL_8))));
            p.setRecipeType(c.getString((c.getColumnIndex(COL_9))));
            p.setIngreList(c.getString((c.getColumnIndex(COL_10))));
            p.setMethodList(c.getString((c.getColumnIndex(COL_11))));
            p.setRecipeRating(c.getString((c.getColumnIndex(COL_12))));
            p.setRecipeTag(c.getString((c.getColumnIndex(COL_13))));

            p.setRecipeFav(Boolean.valueOf(c.getString((c.getColumnIndex(COL_14))));
            p.setObjectID(c.getString((c.getColumnIndex(COL_15))));
            Recipes.add(p);
        } while (c.moveToNext()); //Populates Recipe ArrayList.
    }
    return Recipes;
}
```

Figure 29: Retrieving all Local Recipes for a specific User



Originally a local user account system was going to be implemented, however in doing this a problem arose during development as security issues and the ethical concern of storing all user data came into place. This meant that in order to provide any functionality to a specific user offline a method of confirming which user was the default user was adopted. Then by using this UserID, all the user's recipes and other data could be retrieved for use without any calls to the Parse database.

In order to provide this UserID to the Databasehelper Class, shared preferences have been utilised, as in order to establish the current user normally `ParseUser.getCurrentUser()`; is called, however as there is no internet connection this wasn't possible. Instead, this was done by setting the shared preference for `savedUser` when a user logs into an account for the first time on the application or if the Default User checkbox has been selected. This is then stored indefinitely until a new user is signed in with the checkbox ticked and thus overwrites the value. A temporary user can also be saved to the device for one-time usage also if the checkbox isn't selected. This feature can be seen in **(Figure 30)**.

```
mUserCurrent = user.getObjectId();

if (myPrefs.getString("savedUser", "noUser").equals("noUser") ||
defaultCheck.isChecked()) {
    SharedPreferences.Editor e = myPrefs.edit(); //Saves Default user.
    e.putString("savedUser", mUserCurrent);
    e.commit();
} else if (!(mUserCurrent.equals(myPrefs.getString("savedUser", "noUser")))) {
    SharedPreferences.Editor e = myPrefs.edit();
    e.putString("tempUser", mUserCurrent); //Saves Temp user.
    e.commit();
}
```

Figure 30: Shared Preferences.

The catch to this method however, is that on the applications first run, an internet connection is required to save the initial default user ID in order to load the local database data. Additionally, any temporary user that was signed in previously would not be able to be used offline even if it was the last account logged in as only the `savedUser` ID is used.

#### 4.4.4 Unit Converter

The unit converter was implemented simply by showing an EditText field where users can enter the quantity of the measurement they want converted, and then two spinners which display the units of the measurement they want converted from and to.

One consideration with this was whether to invalidate the conversion between Volume and Weight units, as they are not interchangeable in their base format and would have to include additional information about the item. It was eventually decided against due to the complexity of having to include a dictionary of common ingredients along with what they individually weigh at different physical forms i.e. chopped, diced etc. This was seen as a minor feature and more time was spent implementing different things.

The concept of the basic converter was to send the value of the measurement into a method along with both spinner positions to return the new conversion value. Depending on the position passed into this method and what the desired converted to unit was, would decide if the value was multiplied or divided by the base value to return the correct figure.

The last feature on this page was the ability to store a User's unit conversion history during their current session. This was decided as a time-saving feature if users are preparing multiple dishes and the same units required for different ingredients need to be converted to a more familiar format, saving them from having to re-enter information.

This was done simply by creating another table in the SQLite database and storing the conversion values. Then by calling `NotifyDataSetChanged` on the `ListView` shown in **Figure 31**, update the current list of previous conversions.

Clearing the history was done by calling a method in the database helper class that drops the table containing all the information in the `onCreate` of `MainActivity` on the applications start up. This data along with many other session dependent variables like deleting saved temporary user strings adopted this concept of deleting at startup instead of when `onStop` or `onDestroy` methods are called was important for accurate results. As methods like `onDestroy` which are called when the application is at the end of its life cycle are not guaranteed to execute every time as it's vastly system dependent, whereas `onCreate` is executed every time the app starts.

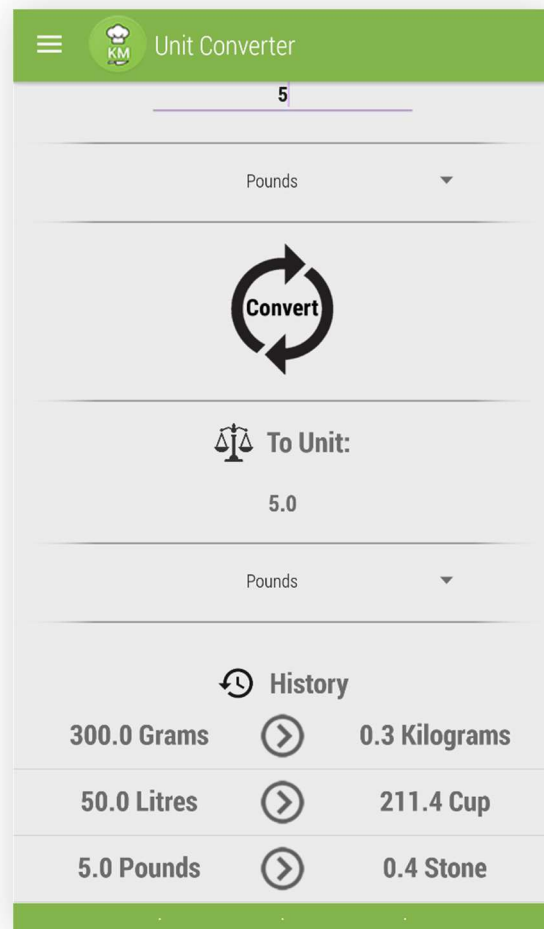


Figure 31: Unit Converter with History of Session.

#### 4.4.5 Shopping list

This feature aspired to improve on the existing shopping lists currently seen in similar applications on the market today. Usually, these lists were very basic containing just an editable text field and an add button. This application tried to provide additional functionality by allowing users to automatically insert ingredients listed in recipes to their personal shopping lists. This will also insert the quantity of the ingredient and the unit of measurement. Then from here if users were to insert the same ingredient instead of adding a new row or showing a duplication error, the application will add the quantity of the ingredient to the value that already exists and update the list. This also works if the units of the ingredients are not the same, by converting the value of the inserting data to the ingredient unit that already exists in the database. The aim of this was to make creating a shopping list for several recipes painless as ingredients will add up and the application will perform all the maths for the user (**Figure 32**).

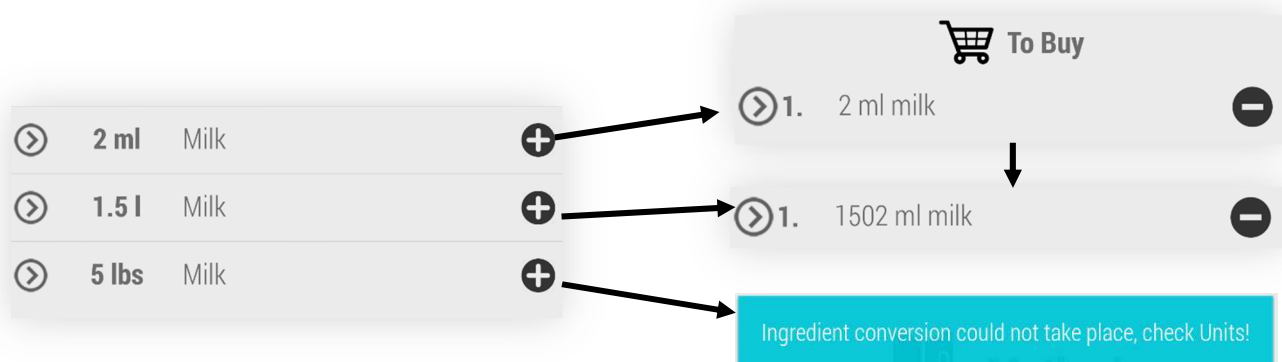


Figure 32: Adding Milk, Updating Milk value and adding an incorrect conversion unit.

The shopping list was implemented firstly by using an `onClickListener` in the `ListView` adapter which fires when the plus icon is pressed on an ingredient row the user wants to add to their list. From here the ingredients quantity, unit and name are passed through several methods explained in pseudo-code in **Figure 33**.

```
Public boolean checkIngredientExists(float ingreQuan, String ingreUnit, String ingreName){
    ArrayList shoppingList = new ArrayList();

    if(haveNetworkConnection()) {
        shoppingList = ParseQuery.whereEqualTo("ingreItem", ingreName);
    } else {
        shoppingList = DatabaseHelper.getIngre(ingreName, savedUserID);
    }

    if(shoppingList.isEmpty()) {
        Return false;
    } else {
        Return true;
    }
}
```

```
if(checkIngredientExists) {
    convertUnit(ingreQuanOld, ingreUnitOld, ingreQuanNew, ingreUnit New);

    if(haveNetworkConnection) {
        ParseQuery.findInBackground(ingreName) // Update Online Database.
    } else {
        ParseQuery.saveEventually // Update Online Database once connection returns.
    }

    DatabaseHelper.updateIngre(ingreName) // Update Local Database.
} else {
    if(haveNetworkConnection) {
        ParseObject.saveInBackground // Create new Ingredient Online Database.
    } else {
        ParseObject.saveEventually // Create new Ingredient once connection returns.
    }

    DatabaseHelper.insertIngre // Create new Ingredient Local Database.
}
```

Figure 33: Methods that Handle Shopping List.

The idea was to initially check if the ingredient exists in the database, if so then convert the inserting value to the format that already exists and add to both databases. Otherwise, create a new object and add to both databases. Parse provided functionality that allowed objects to be created and updated even when the device is offline and then update once a connection was re-established through the use of their local database which temporarily stores objects using the `saveEventually` method. It was decided that every user's shopping list would be stored locally as well as online as the memory footprint was very small compared to the functionality provided for added offline use.

#### 4.4.6 Searching For Recipes

In addition to the implementation techniques discussed in API design, correct parsing of API recipes proved troublesome. This was mainly due to the randomness of the format in which dishes were added by users of the API. Instructions sometimes had newlines or tabs encoded in the string which had to be removed along with step numbers, both which are already handled automatically in this application. Ingredients also sometimes had quantities which included values to unnecessary decimal places (5.00) or sometimes not included at all (0). Parsing



such inconsistent data was a long process but not everything could be resolved such as instructions containing a URL in which users should navigate to, for the full list of steps.

#### 4.4.11 Timer

The timer was implemented with the goal of being able to handle both manual and automatic input of times. Automatic input was done by allowing users to select a step which contains a time and then press start timer, which would input the time listed in the instruction. Manual input just allows users to input their desired minutes / seconds and start the timer. This is shown in more detail in **Figure 34**. Once time has run out a sound clip will play even if the device is sleeping.



Figure 34: Adding Time via Step (Left) and manually (Right).

The timer's logic was executed via Java's built-in `CountDownTimer` method which takes an integer of total time and an integer of what increment it should count down by. With this, manual input was accomplished by inserting the user entered integer after converting it to the correct format using:

$$TotalTime = ((60 * Minutes) + Seconds) * 1000$$

Automatic timing was used in the same way, however retrieving the time from the step required more calculations. The first being able to determine which step had been selected. This was done through the use of an `onClickListener` which sends an intent containing the position / string of the instruction in the `ListView` selected. Then by using this string against a matcher containing a regular expression, the ability to extract the time in a step was possible. This can then be passed into the `CountDownTimer` as normal (**Figure 35**).

One concern with this was that it was only possible to detect Minutes listed inside a step. However after viewing a large selection of recipes during the implementation of this project, no mention of hours or seconds in a dishes instructions was recorded and therefore was not seen as a major priority. However, this is something that would be a future addition to the project as extra functionality.

```

Pattern p = Pattern.compile("[0-9]+\\s*(Minutes|minutes|mins|Mins|Minute|minute|min|Min)");
Matcher m = p.matcher(stepText);

if (m.find()) {
    String time = m.group();
    String[] mfinalTime = time.split("\\s*(Minutes|minutes|mins|Mins|Minute|minute|min|Min)");
    mfTime = (60 * Integer.parseInt(mfinalTime[0]) * 1000);
} else {
    Toast.makeText(getActivity(), "Step doesn't contain any Time!", Toast.LENGTH_LONG).show();
}

```

Figure 35: Pattern Matcher for Method Step.

#### 4.4.12 Text-To-Speech / Hands-free Functionality

Text to speech was implemented firstly by setting up a TextToSpeech Listener which is packaged within Android Studio. This is triggered when the user presses the "Read Next" or "Read Prev" buttons, passing the string of that particular step into this listener so it can be spoken aloud via the Language setting specified. The logic that needed to be implemented was making sure the correct position within the methodList was passed into the listener every time. This was done through the use of two methods seen in **Figure 36**.

```

public void readNextStep() {
    if(prevRead){
        methodProgress++;
    }

    if(methodProgress >= methodList.size()) {
        methodProgress = 0;
    }

    tts.speak(methodList.get(methodProgress), TextToSpeech.QUEUE_FLUSH, null);
    methodProgress++;
    nextRead = true;
    prevRead = false;
}

```

```

public void readPrevStep() {
    if(nextRead){
        methodProgress--;
    }

    if(methodProgress <= 0) {
        methodProgress = (methodListSV.size() - 1);
    } else {
        methodProgress--;
    }

    tts.speak(methodListSV.get(methodProgress), TextToSpeech.QUEUE_FLUSH, null);
    nextRead = false;
    prevRead = true;
}

```

Figure 36: Methods that Handle Position of Step to be Read Aloud.

The logic was achieved through the use of flags and making sure that when the position reached the end / start of the methodList that it was changed to the start / end correspondingly.

The Hands-free functionality aimed to utilise these methods to also perform text to speech for recipe steps, however the issue with this was managing the proximity sensor listening capabilities and finding a way to detect different gestures that would fire a call to these methods. After trying several different ways of implementation, it was

discovered that Proximity sensors in smartphones usually adopt a binary method of returning whether or not an object is near or far from the device. This is usually because its primary use is to disable the screen when phone calls are taking place (Google, 2016).

The original idea was to detect a hand wave around a 5-15cm distance away from the device to trigger the next step to be read aloud and a held hand around 0-5cm distance to indicate the previous step to be read aloud. However after analysing the results produced from the sensor this was not possible to replicate due to only ever returning two values of 0 (Near) or 5 (Far) with no in-between. The compromise for this was to detect if the returned range was equal to the maximum range and if not read the next step in the list as seen in **Figure 37**.

```
public void onSensorChanged(SensorEvent event) {
    if(event.sensor.getType() == Sensor.TYPE_PROXIMITY)
    {
        float maxRange = mProximity.getMaximumRange();

        if (!(maxRange == event.values[0])) {
            readNextStep();
        }
    }
}
```

Figure 37: Sensor Listener.

To activate this sensor a flag inside the hands-free mode check box was created that registered/unregistered the sensor listener on interaction. The issue with this was to make sure to unregister the sensor at any point the user navigates away from the recipe page either manually or automatically (User receives a phone call). This was resolved by performing checks and calls to unregister the sensor at important parts of the applications lifecycle and in the fragment manager (**Figure 38**).



Figure 38: Hands-free mode Toggle.

#### 4.4.12 Activity Lifecycle

Another vital function of the application was its ability to correctly manage the different states it could be in and how it would handle any data that was being used. This meant that any orientation change, incoming phone call or pressing of the home button would trigger methods that would need to be edited in order to manage app state and data. This overall life cycle can be seen in **Figure 43**.

There were two approaches employed to solve all changes in app state. The first change in state that was solved was changing the orientation of the device. When this happens the lifecycle runs through methods seen via path 1 (**Figure 43**), here the activity running is destroyed and restarted calling methods onCreate and so on. Through this, the transient UI state was saved using the onSaveInstanceState method which is called right before onStop is performed (Google, 2016). This method uses a Bundle where it's possible to save variables, however should only be utilised to store preferences of the current layout and not long-term persistent data as onSaveInstanceState is not guaranteed to be called during every change in the application's lifecycle (**Figure 39**).

```
public void onSaveInstanceState(Bundle outState) {

    outState.putBoolean("DefaultUser", mDefaultCheck);
    outState.putBoolean("LoginPage", mLogInView);

    super.onSaveInstanceState(outState);
}
```

Figure 39: onSaveInstanceState Account Creation Page UI State Save.

Now when onCreate is called it is passed this Bundle, meaning that the state change can be checked (**Figure 40**).

```
protected void onCreate(Bundle savedInstanceState) {
    if (savedInstanceState == null) {
        //First Launch, Set up as Normal.
    } else {
        //Change in Orientation, Load Data from Bundle → Apply to Layout.
    }
}
```

Figure 40: onCreate Pseudo Code of Checking Bundle.

The second approach was to handle path 2 (**Figure 43**). This is where the activity isn't destroyed or recreated but is no longer visible, occurring when users receive phone calls or navigates to the home page. This is where persistent data is saved through the use of the onPause and restored with onResume as these methods are called at any change of the life cycle. Saving data within the onPause method could have been completed in different ways depending on the data that was being saved. These methods can be seen below.

Method	Type of Data
Shared Preferences	Small amounts of Key:Value Data.
Internal Storage	Small / Medium amounts of Private Data.
External Storage	Large amounts of non-Private Data.
Database (SQLite)	Small / Large amounts of structured Data.

The method of saving utilised for this project was through Shared Preferences. This was due to the small amount of data that needed to be saved at any time in the application. An example of this can be seen in **Figure 41**.

```
protected void onPause() {
    sharedPref = getSharedPreferences("sharedPref", MODE_PRIVATE);
    SharedPreferences.Editor e = sharedPref.edit();
    e.putString("emailET", mEmailText);
    e.putString("usernameET", mUsernameText);
    e.putString("passwordET", mPasswordText);
    e.commit();
    super.onPause();
}
```

Figure 41: onPause method for Account Creation Page.

Here variables that have been edited during the course of the applications lifecycle are saved as key-value pairs through shared preferences so that any changes the user has made are still there once they return. In order to restore this data similar logic is performed in the onResume (**Figure 42**).

```
protected void onResume() {
    string savedEmail = sharedPref.getString("emailET", "");
    string savedUsername = sharedPref.getString("usernameET", "");
    string savedPassword = sharedPref.getString("passwordET", "");

    emailET.setText(savedEmail);
    usernameET.setText(savedUsername);
    passwordET.setText(savedPassword);

    super.onResume();
}
```

Figure 42: onResume method for Account Creation Page.

Both of these approaches of handling data & layout allowed for increased usability throughout the application handling every stage of the lifecycle. However a future addition to this would be the ability to save data whilst it's being entered into fields, allowing for functionality like displaying search results as they are being typed.

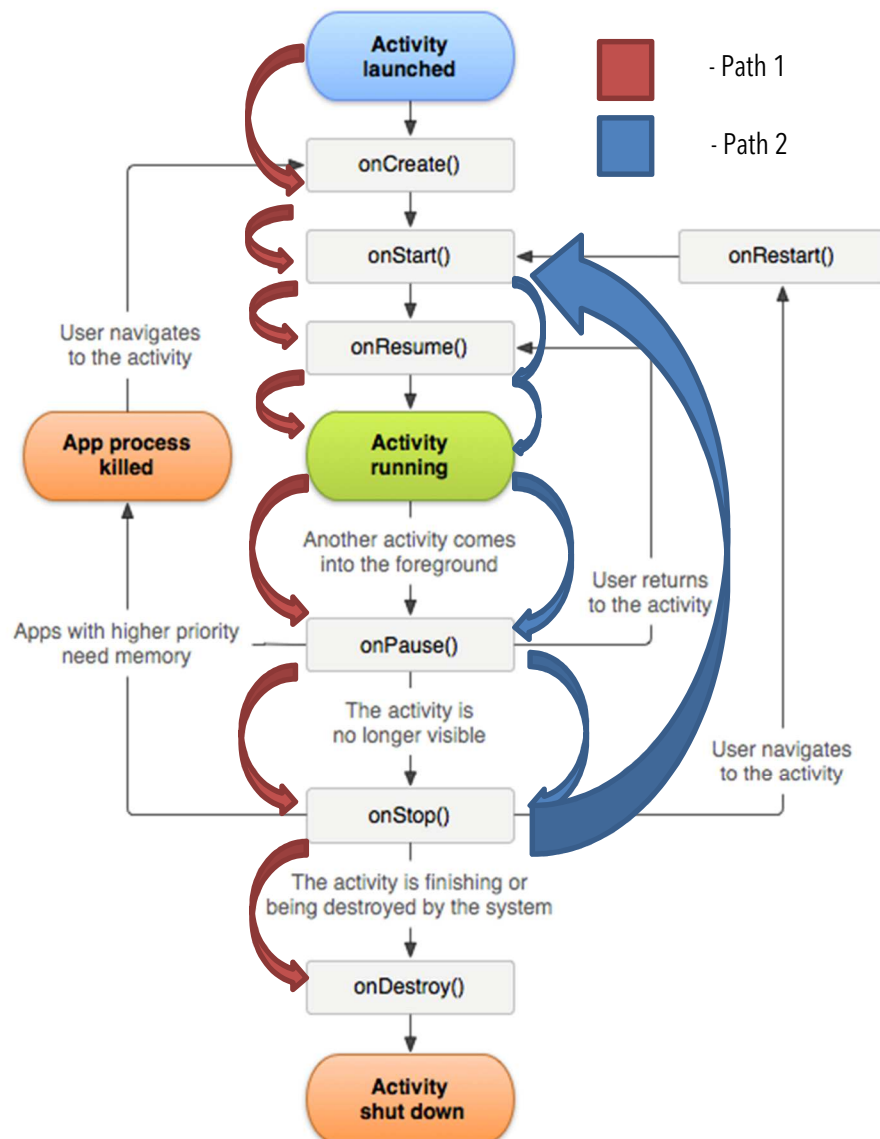


Figure 43: Activity Lifecycle of an Android Application (Google, 2016).

## 5. Evaluation

### 5.1 Potential Risks

#### 5.1.1 Project

In addition to the initial risks (Appendix A), a risk for the project was not adhering to the updated time plan due to other work commitments or inaccurately predicting time to complete tasks. Therefore, as the project was due to be completed around 2 months before the due date, this allowed for minor revisions and improvements if necessary. Another risk was spending too much time on perfecting features to the last detail (tweaking colour schemes, creating graphics etc.) when all the key features were yet to be implemented. As this is where time would have overrun and finishing on time would have been difficult.

#### 5.1.2 Product Design & Deployment

A foreseen risk with Deployment was not implementing all the features originally planned for the project due to time constraints or difficulties in implementation. To overcome this the time plan allowed extra time for tasks predicted to be challenging in order to allow for extra research. If this was still insufficient then realistic cuts were made to consider leaving a half implemented feature in the App, meaning that features which depended on each other would be implemented at the same time as stated in the adjusted Task list.

A risk for product design was that users wouldn't be able to understand how to use the App the way it's designed for. In order to overcome this, the Android Developer rules have been adhered to in terms of App Design meaning that the overall layout and feel should be very familiar to a normal user of a smartphone.

### 5.2 Ethical Concerns

#### 5.2.1 User Health

A concern that has been identified is that as this App is involving users with food and the preparation and cooking of meals, it's important to note that the promotion of a healthy balanced diet should be made.

Suggesting meals that are unhealthy to users consistently will have a detrimental effect on their health if they choose to use these recipes and consume them regularly. In order to overcome this, the research on the Recipe Database API where users can search for recipes has been thorough, making sure that there are a wide variety to choose from that adhere to most if not all dietary restrictions. Then to overcome this these recipes of varying nutrition are used in the search function of the App allowing users to pick and choose themselves.

#### 5.2.2 User Data

Another concern was the security of users data stored on both the local and online databases. Data should only be accessed if it belongs to the user that is currently logged in and in order to overcome this some measures were taken. Parse already handles user data quite effectively by utilising Access Control Lists, whereby each Object (row) in the database is assigned an ACL with the same authentication as the user who creates it. Meaning that when the application, for example, retrieves data to sync the homepage recipe list, only objects that match that same ACL will be downloaded.

The local database works in a similar way by storing a recipes ACL value as well as the data when downloading from Parse. Then when the local recipes are synced, a call to check what user is signed in is made and the recipes that match that user are loaded. In addition to this User Accounts are handled entirely by Parse which hashes user's passwords using bcrypt. This allows for users to securely log in with their passwords but also eliminates any possibility of their accounts being compromised if for example the parse servers were hacked as the password is not stored anywhere on the database.

### 5.3 Risk Analysis

In addition to the risks predicted for this project which can be seen in Appendix A. There were some risks that were established during the implementation of this project that was not thought of, which can be seen below.

Risk	Severity (L/M/H)	Likelihood (L/M/H)	Significance (Sev. x Like.)	How to Avoid	How to Recover
API Provider Shutting down Service	H	L	HL	Out of Users Hands.	Be familiar with alternatives available. Have a backup provider to switch to if primary closes services.
Health Issues	M	L	ML	Eat Healthily, Exercise.	See Doctor, take medication.
Feasibility	M	M	MM	Try not to set impossible goals for the projects time frame.	Evaluate the goal if it's taking too long or the complexity is higher than expected.

### 5.4 Project Achievements

After evaluating the original time plan for this project in comparison to how implementations for this project took place, I can say that the project was a success. With the estimation on total project completion being accurate as the initial time plan (Appendix B) stated that the project would hope to be completed 22<sup>nd</sup> Feb with the adjusted time plan (Appendix C) stating 14<sup>th</sup> March. In reality, the project had implemented all major features leaving only minor changes to be made the week starting 14<sup>th</sup> March.

However when it came to following the time plans given time scale for certain features, it was mostly inaccurate. This is mainly due to the fact that they were estimations based on how long the developer thought each feature would need in order to be implemented without having prior knowledge of the complexity. For example, the shopping list was given 2 days for completion when in reality it took almost a week due to complications with the local and online database. The opposite effect also happened where for example the hand over recognition was given 7 days to complete when in reality it took only a day. This made for very erratic implementation of all the features and although the order in which they were added was accurate to the time plan, the completion times were not.

In terms of consistency, time spent on development was highest around the Christmas break and February where work commitments from other areas were minimal. This is where most of the tasks were completed as seen from the overall progress of the adjusted interim task list, seen below and initial task list (Appendix D). Where only minor tasks like a Tutorial Screen, help options and the search bar for recipes linked with that user were left unimplemented.

Task Number	Priority	Task Name	Completed / Started / Not Started	Duration (days)
1	1	Implement User Account Saving/Loading	Completed	2
2	2	Implement Full Recipe View	Completed	2
3	3	Implement Text-to-Speech on Steps in a Recipe	Completed	3
4	4	Add Database API of Recipes	Completed	7
5	5	Implement Editing of recipes	Completed	3
6	6	Implement Deletion of recipes	Completed	1
7	7	Implement Conversion Tool	Completed	3
8	8	Implement Hand hover Recognition	Completed	7
9	9	Implement a Search function for Recipes	Completed	3
10	10	Implement Search function for API online Recipe database	Completed	3



11	11	Implement Sorting of recipes in Lists and Search bar	Started	2
12	12	Correctly convert API downloaded recipes to consistent format	Completed	7
13	14	Implement Built-in Timer for Recipe cooking Steps	Completed	2
14	15	Implement a Favourites List	Completed	2
15	16	Implement a Shopping List	Completed	2
16	17	Design Logo	Completed	4
17	18	Design Buttons	Completed	4
18	19	Implement Settings option for sidebar	Completed	3
19	20	Implement About/Contact option for sidebar	Completed	2
20	21	Implement Tutorial Screen on Startup	Not Started	3
21	22	Implement a Help option	Not Started	2
22	13	Write Final Report	Completed	50

As the developer had never completed a project of this size before, combined with having limited knowledge of the platform. Time management and prioritising workload were crucial factors that lead to the success of this project. Creating weekly objectives along with guidance from the supervisor helped the progress of this project especially during crucial periods like February where workload was highest.

## 5.5 Objective Evaluation

### 5.5.1 Objective 1 – Appropriate Management of User Accounts and their Data

#### 5.5.1.1 Secondary Objective 1: Online Database

With the utilization of the Parse API, the project has successfully implemented a working online database. It handles User accounts along with having two separate tables for recipes and shopping list ingredients. Data is also saved to Parse's local database when the app is offline and uploaded to online storage when a connection is re-established.

**Percentage Complete: 100%**

#### 5.5.1.2 Secondary Objective 2: Local Database

Using Androids built in SQLite database class, the project has successfully implemented a local database. This database serves to save recipes or shopping lists from a specific Parse account for use on the app if no internet connection can be found. Successfully handles saving, editing and deleting of data and updates the Parse database when action has been taken.

**Percentage Complete: 100%**

### 5.5.2 Objective 2 – Create a System UI that adheres to the Developer Interaction/Design

#### Documentation

Reading the official documentation provided by Android before implementing a feature for this project helped to create a system UI which adheres to developer guidelines. With the exception of the Recipe Creation / Single view pages where in order to create listviews which update dynamically, an unofficial method was used to update the displayed height of the list, by calculating the standard height of each row and then multiplying by the total length of the list. The default behaviour for listviews is to allow scrolling if the data set is greater than the total specified height. However as the page was already encapsulated by a ScrollView which allows scrolling of the page, the page would be very stiff and not scroll correctly.

**Percentage Complete: 90%**

### 5.5.3 Objective 3 – Create an interactive Recipe Editor / Creator

The project successfully allows the user to create and edit recipes which will save and update on each specified database. The recipe creator allows the addition of images from either the phones gallery or camera and dynamically updating ListViews for ingredients and steps. There are also options to add tags, rating and miscellaneous data for users to upload. The recipe editor also allows users to change any of these parameters and update the same recipe to their liking.

The only feature lacking is the addition of being able to add images to recipe steps was not completed due to recipe API format not containing any images for steps.

**Percentage Complete:** 95%

### 5.5.4 Objective 4 – Implement a method of assisting with Time Management in the Kitchen

The project has implemented several methods of helping with time management. A timer has been created allowing users to select a step of a recipe where the time will be inserted automatically and started with the press of a button, then an alarm will be played once the timer has finished. Another time-saving feature is the unit converter which allows users to enter a value and get conversion rates to an array of other figures they request. These both assist with time management as instead of backing out of the app to search the internet or open other apps, the functionality is included.

**Percentage Complete:** 100%

### 5.5.5 Objective 5 – Develop an Interactive Hands-free way of communicating with the App whilst cooking

The project has successfully developed a hands-free method of working with the user. A checkbox has been placed next to the "Read Steps" button which once checked enables the proximity sensor to take readings of change in distance. This allows the user to wave their hands over the device and activate the readStep method, reading aloud the recipe step.

However ideally there should have been a way to navigate going back a step, for example, holding the hand over the sensor for a longer period of time. This functionality is still available manually but hardware constraints meant that this minor part was unimplemented.

**Percentage Complete:** 90%

### 5.5.6 Objective 6 – Utilize an External Online Database of Worldwide Recipes

The utilization of a searchable database has been successfully implemented through the use of BigOven. Users can search for a recipe by typing an ingredient or a recipe name and a list of results will be produced. Recipes can then be viewed and saved to that user's personal account.

However the ability to search with other parameters like dietary preferences were not possible due to the limited features given through the academic API key this project received.

**Percentage Complete:** 100%

### 5.5.7 Summary

Overall the project's objectives have been very successful with every objective having a working feature which performs the task specified. The only drawbacks being minor faults caused by either the API supplied for the project or time constraints.

**Overall Percentage Complete:** 96%

## 5.6 Further work

### 5.6.1 Refinements

#### 5.6.1.1 Different Recipe API / Upgraded BigOven Key

The biggest refinement for this project would be the API. As although BigOven's API worked correctly and has a large database of recipes as well as being the most superior API for this project out of the ones tested, the academic key that was supplied for this project lacked in features that could be implemented such as searching by ingredient, dietary preference etc. Additionally, the recipe format did vary quite significantly at times making it very hard to parse the data into the format for this project so that it consistently displayed Recipes.

#### 5.6.1.2 Search via Tag

The ability to search for recipes via their tag would be another refinement. The recipe tag implementation as it stands serves to show information about the recipe about dietary preferences etc. when viewing a recipe in single view mode. However being able to search for recipes with these specific tags would increase the usability of the application by suggesting recipes users have in their databases instead of having to check each one individually.

#### 5.6.1.3 Graphic Design

One of the key aspects of any application is how it looks and is presented to the user. This project made an effort to look appealing by using graphics / custom images, being more presentable than some of the applications on the market today. But due to the time constraints of this project and the importance of varied feature implementation, time spent on graphic design was minimal and could be improved.

#### 5.6.1.4 Recipe Step Text Wrapping

Another refinement would be changing the way the recipe step handles text that is too long for the device to display. Currently, the text will marquee across the screen making it hard to see the full step without waiting. But ideally the text should wrap displaying the full step. However the problem of including a ListView inside a ScrollView and the method used for calculating total height would not allow this functionality. So a way of wrapping text and displaying the ListView dynamically would solve this issue.

#### 5.6.1.5 Hands-free Navigation Control

As explained previously a way to navigate the Text-to-Speech functionality hands-free would be useful for when users might want to repeat or navigate to previous steps. Although this was attempted to be done via the use of the proximity sensor, a future refinement would be to include several sensors that can detect different things. For example, the proximity sensor could still manage its current functionality and the ambient light sensor could detect hands covering the sensor at different ranges to perform other tasks.

#### 5.6.1.3 Automatic Timer Detection

The addition of the timer allowing detection of hours and seconds in recipe steps would allow more functionality to the timer. As mentioned previously these time formats were not seen across any dishes during implementation but should be expected to work if minutes are able to be extracted for use.

#### 5.6.1.3 Search Bar Loading Results on Keypress

Including the ability to display search results on keypress would provide quicker feedback to users about what recipes are available. This was not included due to basic functionality being the priority at the time of creation, but would be a superior alternative to refine the applications search.

## **5.6.2 Additions**

### **5.6.2.1 Xamarin**

One addition to this project would be the use of Xamarin and its ability to port this application to other platforms like Windows Phone and iOS. This way the project could target a larger audience without the extra work of having to redevelop in different environments. This would require either manually converting Java code to C# which would require more time or use a code conversion tool such as Sharpen. However as the developer has experience in both languages either solution is possible.

### **5.6.2.2 Image Option to Recipe Steps**

Another addition to the project would be the ability to attach an image to a recipe step when creating / editing a recipe. This would provide more detail to the process if users could see what the dish looks like at each stage as well as acting as a guideline on their progress. This wasn't added to the project due to constraints with the ListView within Android as well as the format of recipes provided by the API whereby steps did not include images.

### **5.6.2.3 Multiple Timers**

The ability to have multiple active timers at once was another addition that would benefit the project. As sometimes multiple steps are being carried out at the same time within a kitchen, sometimes requiring a timer. It would be beneficial to track each step simultaneously and not be limited to having to wait for one step to finish as multitasking is common when preparing a dish.

### **5.6.2.3 Nutrition Tab**

A useful addition to the recipe view would be a tab or segment displaying the nutritional information for that dish. This is useful for promoting healthy eating by highlighting what effects the recipe will have on the user's diet. Yummly first highlighted this as a useful feature, but this was seen as a secondary objective to implement once everything else had been completed. This was, unfortunately, unable to happen due to time being spent on other features.

### **5.6.2.4 Smart Watch Implementation**

The addition of adding smart watch functionality would be beneficial for minor tasks. An example of this would be the addition of syncing the current timer to the user's smartwatch. This way they could leave the kitchen and perform other tasks while things are cooking without the inconvenience of having to take the device with them or listen out for the timer from other rooms. The watch would notify them of the timer finishing from within the house and they could continue.

## 6. Conclusion

Ultimately the project has been very successful even though it failed to fully include all the features initially planned for each objective. Despite this the final product has managed to utilize the Android platform, follow developer guidelines and create an application which provides users with a familiar, feature rich experience which has genuine real-world use in and outside of the kitchen.

The main contributions for this project have been:

- |                           |                              |
|---------------------------|------------------------------|
| ✓ Local / Online Database | ✓ Recipe Creator / Editor    |
| ✓ Text-To-Speech          | ✓ Favourites / Shopping List |
| ✓ Hands-free interaction  | ✓ Search Function            |
| ✓ Timer Functionality     | ✓ Access to External Recipes |
| ✓ User Accounts           | ✓ Sort by Recipes Details    |

The developer has confidence that if the further work stated previously was carried out, that the application would be a contender for some of the applications on the market available today. Offering more features or a better implementation of existing ones, would really set it apart from others, with the only major drawback being the disadvantages that come with using an existing API where Developers are limited to the format they provide for recipes.

### 6.1 Learning outcomes

This project has been challenging, fun and enriching. Many different skills have been developed over the period of its duration with the primary examples being listed below.

- |                    |                    |
|--------------------|--------------------|
| ✓ Java Development | ✓ SQLite Databases |
| ✓ XML              | ✓ Report Writing   |
| ✓ JSON / Parsing   | ✓ Time Management  |

## 7. Appendices

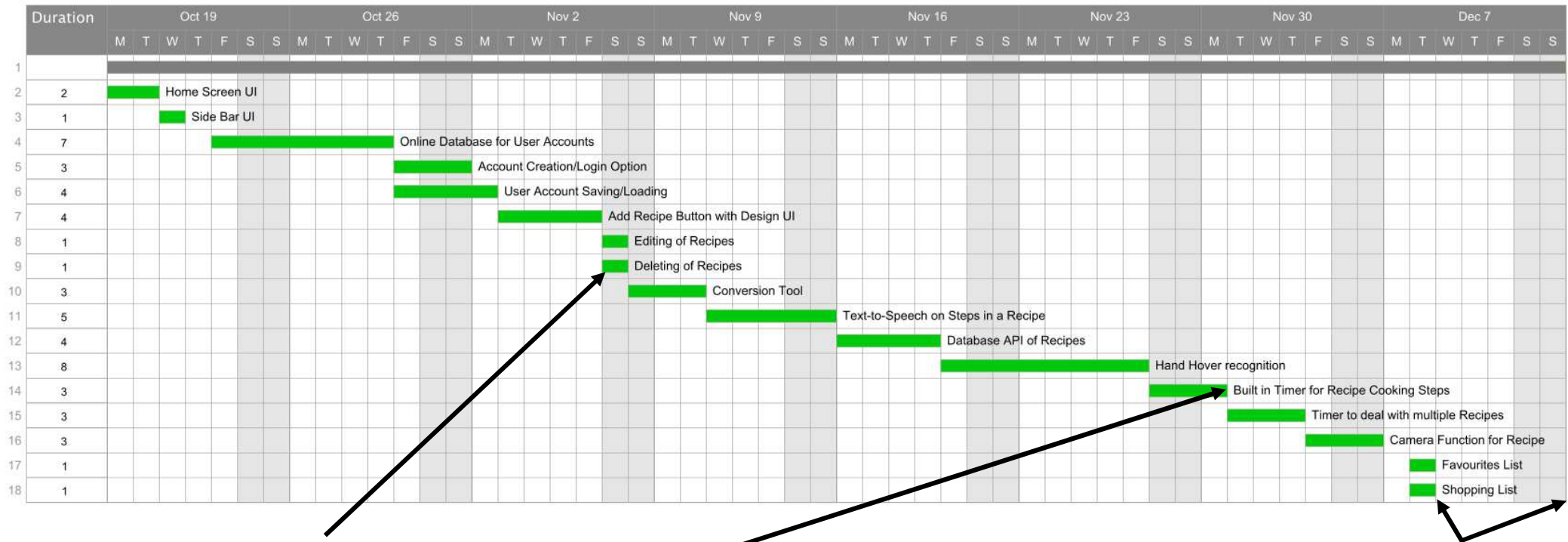
### 7.1 Appendix A – Initial Risks Table

Risk	Severity (L/M/H)	Likelihood (L/M/H)	Significance (Sev. x Like.)	How to Avoid	How to Recover
Data loss of Project	H	M	HM	Keep Backups via SVN and external Hard drive.	Reinstate from backups.
Loss of backups Project/Accounts	H	L	HL	Multiple Backups with cloud storage.	Reinstate from backups.
Skill risk (Not having previous experience)	M	M	MM	Conduct research into skills needed prior to development.	Assign extra time to learn skills needed for the project.
Not compatible with users Android Version	L	L	LL	Assign Apps settings for Android versions (Target Majority).	Edit settings in Android studio so that it targets modern versions.
UI not compatible with screen resolution	L	M	LM	Create icons for all types of screen DPI	Redesign icons or use standard icons but be aware of reduced quality for larger resolutions.
Inconsistent UI	M	L	ML	Make sure to use similar fonts and follow consistent design patterns with buttons, logo etc.	Redesign UI and change minor things such as fonts or colour schemes or follow a built-in template if time constraints.
Interactive features not working as intended i.e. buttons	H	M	HM	Make sure implemented features are working before moving on to creating next task.	If it is a minor feature and have limited time, remove. If it is a key task, edit until working correctly.
Data loss for User Accounts	H	M	HM	Create another online backup which copies the database at timely intervals.	Reinstate from backups.
Not all users will have a phone with an ambient light sensor	M	M	MM	List an ambient light sensor as a requirement to be able to use the app to its fullest.	Create alternative option so that users can perform the same task the hardware would perform.
External Database Legal issues	H	L	HL	Make sure to read the terms and conditions of the company that will be providing the database and check if it can be implemented.	Find another database online that will accommodate the app legally.
Fail to meet Time Plan Deadline for Task	M	H	MH	Keep working to schedule and if needed put in extra work to try and meet deadline on time.	Put it extra time to try to meet Task taking into account the priority, if it is an extra feature and Key tasks still need implementing, move on.

## 7.2 Appendix B – Initial Time Plan

### 7.2.1 Semester 1 (17/10/15 – 18/12/15)

Developmental Task



Milestone: Basic App Implemented

Milestone: All key Tasks Completed

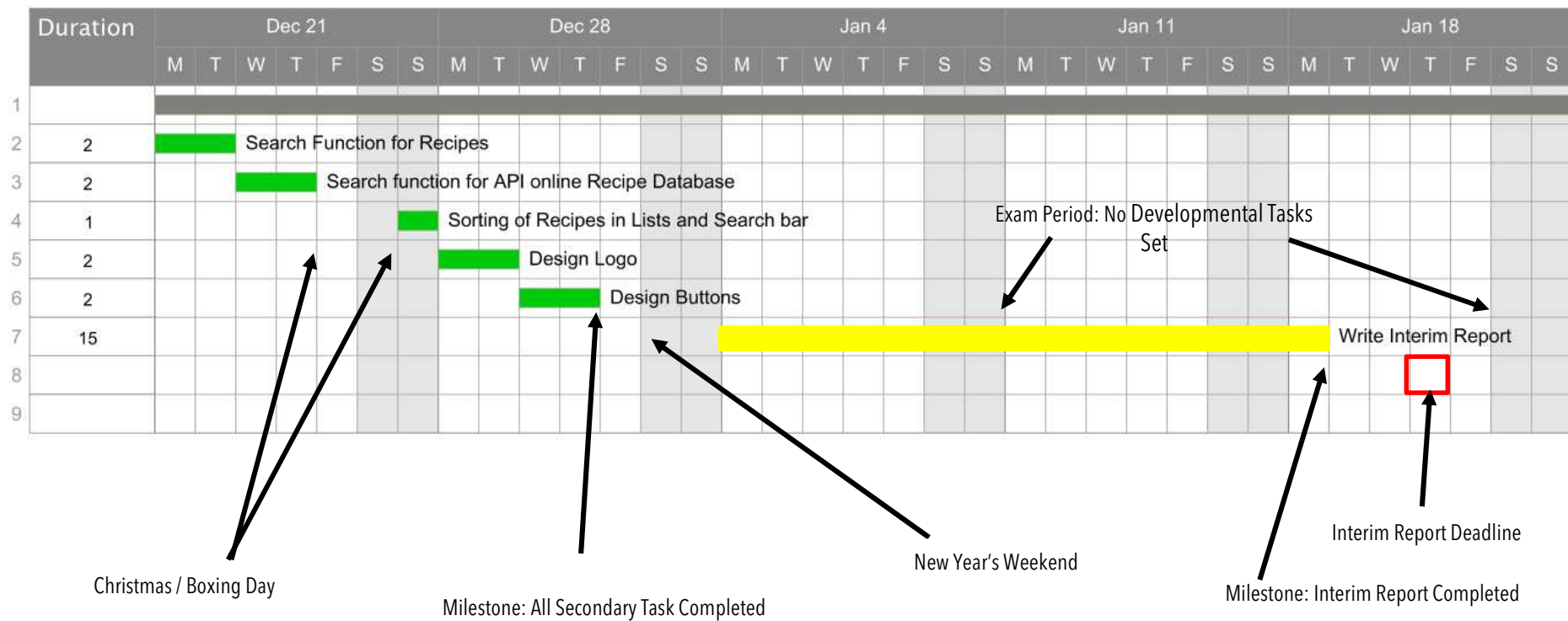
10 days given for any coursework completion to be made at end of Semester 1.


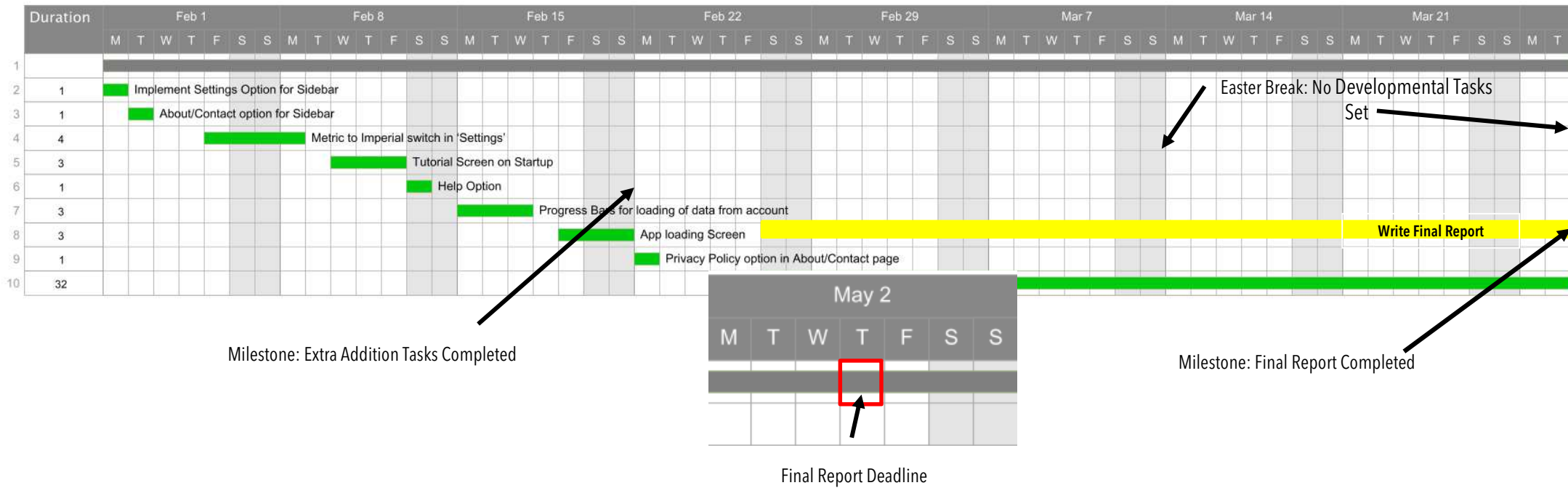


### 7.2.2 Christmas Break (21/12/15 - 24/01/16)

Developmental Task

Management Task

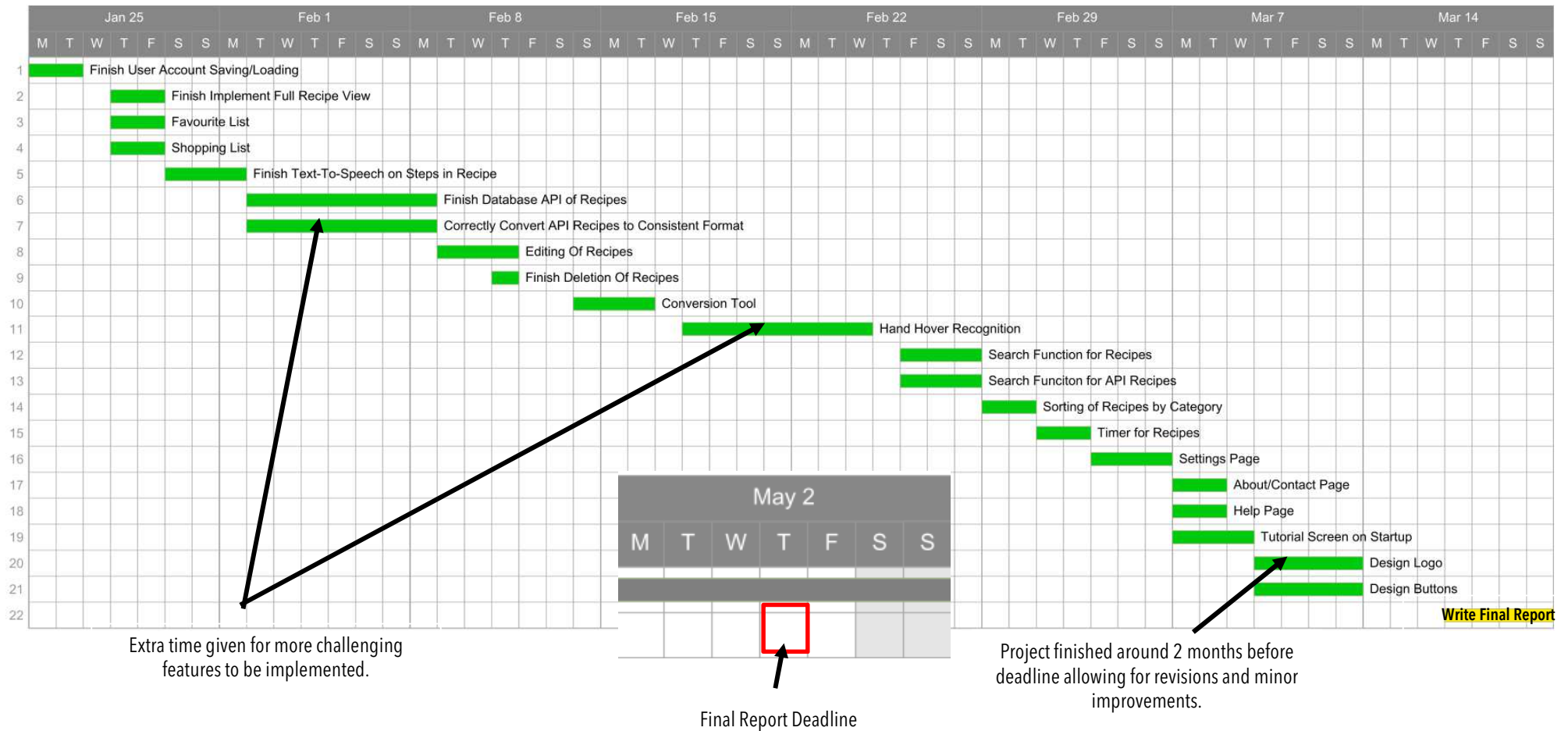


**7.2.3 Semester 2 (01/02/16 – 13/05/16)** Developmental Task Management Task

### 7.3 Appendix C – Adjusted Interim Time Plan

Developmental Task

Management Task



## 7.4 Appendix D – Initial Task List

Task Number	Priority	Task Name	Description	Duration (days)
1	1	Implement Home screen UI	Create a working scrollable image grid that will eventually hold recipes finished image with brief information. Each image should be able to be viewed in a separate window once selected. <b>Objective 2</b>	2
2	2	Implement Sidebar UI	Create a sidebar for the home screen, displaying a list of all available options to be later implemented. <b>Objective 2</b>	1
3	3	Implement Online Database for User Accounts	Create an online database that will store usernames, passwords and each individual 'My Recipes' List. <b>Objective 1</b>	7
4	4	Implement Account Creation/Login Option	Create the Sidebar option for 'My Account', implementing prompts to be able to create an Account and be able to log in by connecting to the database. <b>Objective 1</b>	3
5	5	Implement User Account Saving/Loading	Correctly implement the feature to Save the account to the database when any changes are made as well as loading the correct updated information if the app is run again. <b>Objective 1</b>	4
6	6	Implement an 'Add Recipe' Button with Design UI	Create an option to select to create own recipe and new window UI in which one can be created with appropriate headings (Title, Preparation time, Ingredients etc.) <b>Objective 4</b>	4
7	7	Implement Editing of recipes	Create an option to be able to edit recipes that will bring up a similar window as that of a creation of a Recipe and correctly save the edited parts to the database and on screen. <b>Objective 4</b>	1
8	8	Implement Deletion of recipes	Create an option to be able to delete recipes that will be correctly updated on screen and in the database. <b>Objective 4</b>	1
9	10	Implement Conversion Tool	Create the Sidebar option for 'Conversion Tool', a tool that will convert different measurements to another i.e. Cups to Millilitres. <b>Objective 3</b>	3
10	11	Implement Text-to-Speech on Steps in a Recipe	Create a Text-To-Speech feature that will read off a step of a recipe individually, once pressed. <b>Objective 5 &amp; 3</b>	5
11	12	Add Database API of Recipes	Connect a database of recipes available online via an API to the app, so the user can view, add them to their list. <b>Objective 6</b>	4
12	13	Implement Hand-hover Recognition	Create Hovering hand detection on the phone and connect it to the text-to-speech feature for reading out of steps in a recipe. So that overall, once the first step is pressed to be read out aloud, hovering over the phone will continue the next step. <b>Objective 5</b>	8
13	14	Implement Built-in Timer for Recipe cooking Steps	Create an option on the steps in a recipe to set an inbuilt timer for the duration of cooking time. <b>Objective 3 &amp; 5</b>	3
14	9	Write Interim report	Write the interim report deliverable and upload it.	15

15	16	Implement Timer to deal with multiple recipes	Make multiple timers work across multiple recipes so that they keep counting down no matter what page of the app you are on. <b>Objective 3 &amp; 5</b>	3
16	17	Implement Camera function for recipe	Create camera / 'choose from gallery' option in Add/Edit windows of recipes so that users can upload their own pictures of what they have created. <b>Objective 4</b>	3
17	18	Implement a Favourites List	Create an option for users to select a recipe as their favourite and display it on a separate list under 'My Favourites' in the Sidebar. <b>Objective 3</b>	1
18	19	Implement a Shopping List	Create an option for users to select ingredients in a recipe and add it to the 'Shopping List' in the Sidebar. <b>Objective 3</b>	1
19	20	Implement a Search function for Recipes	Create a search bar that will be able to search through users Recipes and display them in a separate window. <b>Objective 3 &amp; 2</b>	2
20	21	Implement Search function for API online Recipe database	Update the Search function to search in the recipes that are connected to the app via the API database. <b>Objective 3 &amp; 2 &amp; 6</b>	2
21	22	Implement Sorting of recipes in Lists and Search bar	Create options to sort the Recipes by different factors such as Rating, shortest Preparation time/Cooking time, Vegetarian etc. <b>Objective 3</b>	1
22	23	Design Logo	Design and implement an iconic Logo for the app and add it the UI. <b>Objective 2</b>	2
23	24	Design Buttons	Design and implement consistent and related buttons to be used in the Sidebar and throughout the app that will match the design UI / Logo. <b>Objective 2</b>	2
24	25	Implement Settings option for sidebar	Create the Sidebar option for 'Settings', opening a new window which will display different options. <b>Objective 3</b>	1
25	26	Implement About/Contact option for sidebar	Create the Sidebar option for 'About the App' and 'Contact', giving correct version numbers and contact details of the App Creator. <b>Objective 3</b>	1
26	27	Implement Metric to Imperial switch in 'Settings'	Create the option for inside 'Settings', so that throughout the app measurements will be in Metric or Imperial depending on the choice for each recipe. <b>Objective 3</b>	4
27	28	Implement Tutorial Screen on Start-up	Create an interactive tutorial window that will show to the user on first start-up of the app, explaining how to use the 'basics' of the app. Making sure not to show again if the app is restarting again. <b>Objective 2</b>	3
28	29	Implement a Help option	Create the 'Help' option for the Sidebar by connecting the interactive tutorial screen to the Help button. <b>Objective 3</b>	1
29	30	Implement Progress Bars for loading of data from account	Create a progress bar to be placed on the window to represent loading of data throughout the app to notify users that activity is happening. <b>Objective 2</b>	3
30	31	Implement App loading Screen	Create a loading screen to be displayed for when app is started from the launcher. <b>Objective 2</b>	3
31	32	Implement Privacy Policy option in About/Contact page	Create a brief privacy policy text view containing legal information and link it to the About/Contact Page. <b>Objective 3</b>	1
32	15	<b>Write Final Report</b>	Write the Final report deliverable and upload it.	32

## 7.5 Appendix E – Acceptance Testing Table

Test	Segment of App	Pass/Fail
Edit Text boxes have a limit of inputted characters.	All	Pass
Account Logging in/Creation only works if all fields are filled.	Accounts	Pass
Forgot Password Links sent to correct email.	Accounts	Pass
Logging in works only if email is verified.	Accounts	Pass
On screen Back button works as intended and exits App correctly.	All	Pass
App Resumes without errors after being exited.	All	Pass
Going back to the App keeps information stored on exited page. (i.e. Creating a recipe)	All	Fail
App works on multiple Android devices of different screen resolutions.	All	Pass
Works as intended in Landscape mode without bugs.	All	Pass
Pages don't stack upon one another when opened repeatedly.	All Pages	Pass
History for Current session conversions save.	Unit Conversion	Pass
All Unit conversions are accurate.	Unit Conversion	Pass
Searched Recipe results are accurate to search term.	Search Recipes	Pass
Recipes are added to the Favourites List when Heart is selected.	Edit View	Pass
Ingredients are added to the Shopping List when Selected.	Full Recipe View	Pass
The timer starts and finishes at the right time when used.	Full Recipe View	Pass
Text-To-Speech reads Steps out correctly.	Full Recipe View	Pass
Waving Hand over phone starts next step to be read out aloud.	Full Recipe View	Pass
Pressing Back button navigates to the correct previous page viewed.	All	Pass
Selecting a picture from the Gallery scales correctly and inserts itself into the creation screen.	Recipe Creation	Pass
Taking a photo and accepting the picture inserts it in the correct orientation into the creation screen.	Recipe Creation	Pass
Create a recipe when not connected to the internet.	Recipe Creation	Fail
All Data inserted to the recipe creation is correctly inserted into the database.	Recipe Creation	Pass
Text Marquees for long names recipes.	Main Page	Pass
Sorting recipes by different parameters works correctly.	Main Page	Pass
Images are loaded quickly and smoothly on start up.	Main Page	Pass
Created recipes are added to Recipe list automatically.	Recipe Creation	Pass
Navigation bar show User signed in currently along with email address.	Nav Bar	Pass
Logging in with "Save as Default User" ticked, saves user as Primary User in shared Preferences.	Accounts	Pass
Logging in without ticked saved user as temp User in shared Preferences.	Accounts	Pass
Unit Converters history is wiped after the session is closed.	Unit Conversion	Pass
Not able to perform conversion between Mass and Volume units.	Unit Conversion	Pass
Null data from searched recipes are formatted correctly within the single view.	Search Recipes	Pass
The format of recipes is as desired every time.	Search Recipes	Fail
Adding a searched recipe to the database works correctly.	Search Recipes	Pass
Favourites list correctly displays user's favourite recipes.	Favourites	Pass
Shopping list correctly displays items that were added from recipes.	Shopping List	Pass
Adding an ingredient that already exists in the list updates quantity if the same unit.	Shopping List	Pass
Adding an ingredient that already exists in the list updates quantity if different unit and converts.	Shopping List	Pass
Deleting all Recipes in settings performs expected function.	Settings	Pass
Deleting all shopping lists in settings performs expected function.	Settings	Pass
Deleting User Accounts performs expected function.	Settings	Pass
Contact Page correctly displays information.	Contact Page	Pass
About Page correctly displays information.	About Page	Pass

Pressing the Logout button logs the user out of the session.	<i>Nav Bar</i>	<i>Pass</i>
Recipes load information correctly when in single view page.	<i>Full Recipe View</i>	<i>Pass</i>
Pressing back with hands-free mode still activated doesn't continue reading.	<i>Full Recipe View</i>	<i>Pass</i>
Timer Still runs when not on Full recipe view page.	<i>Full Recipe View</i>	<i>Pass</i>
The timer still runs when the screen is turned off.	<i>Full Recipe View</i>	<i>Pass</i>
Automatic Time insert works when a recipe step containing time is selected.	<i>Full Recipe View</i>	<i>Pass</i>
Manual time insertion works correctly.	<i>Full Recipe View</i>	<i>Pass</i>
Editing a recipes data updates in the databases correctly.	<i>Full Recipe View</i>	<i>Pass</i>
Deleting a recipe updates in the databases correctly.	<i>Full Recipe View</i>	<i>Pass</i>
Saving a recipe to the local database works correctly.	<i>Full Recipe View</i>	<i>Pass</i>
Adding the same recipe twice displays error.	<i>Full Recipe View</i>	<i>Pass</i>
Updating a recipe offline updates when the connection is re-established.	<i>All</i>	<i>Pass</i>
Deleting a recipe offline updates when the connection is re-established.	<i>All</i>	<i>Pass</i>
Creating a recipe offline displays an error message.	<i>All</i>	<i>Pass</i>
Local recipes assigned to the correct user are shown when switching accounts.	<i>All</i>	<i>Pass</i>
Online recipes assigned to the correct user are shown when switching accounts.	<i>All</i>	<i>Pass</i>
Logos and graphics are correctly shown on pages.	<i>All</i>	<i>Pass</i>
Navigation drawer functions without lag.	<i>Nav Bar</i>	<i>Pass</i>



## 8. References

- Android Authority, 2015. *HTML5 vs Native Android App*. [Online]  
Available at: <http://www.androidauthority.com/html-5-vs-native-android-app-607214/>  
[Accessed 14 October 2015].
- Android, 2015. *Download Android Studio*. [Online]  
Available at: <https://developer.android.com/sdk/index.html>  
[Accessed 14 October 2015].
- Apple, 2015. *App Store*. [Online]  
Available at: <https://search.itunes.apple.com/WebObjects/MZContentLink.woa/wa/link?mt=8&path=appstore>  
[Accessed 14 October 2015].
- Apple, 2015. *Apple Developer Program*. [Online]  
Available at: <https://developer.apple.com/programs/enroll/>  
[Accessed 14 October 2015].
- Ariel, 2015. *App Stores Growth Accelerates in 2014*. [Online]  
Available at: <http://blog.appfigures.com/app-stores-growth-accelerates-in-2014/>  
[Accessed 16 March 2016].
- Beaubouef, T., Petry, F. E. & Ladner, R., 2011. Normalization in a Rough Relational Database. In: D. Ślęzak, et al. eds. *Rough Sets, Fuzzy Sets, Data Mining, and Granular Computing*. Berlin: Springer Berlin Heidelberg, pp. 275-282.
- BigOven, 2016. *BigOven Kitchen Cloud Platform*. [Online]  
Available at: <http://api.bigoven.com/>  
[Accessed 18 March 2016].
- Business Insider, 2015. *Google giving developers reason to choose Android vs. iPhone - Business Insider*. [Online]  
Available at: <http://uk.businessinsider.com/google-developers-android-vs-iphone-2015-5?r=US&IR=T>  
[Accessed 10 October 2015].
- CNBC, 2015. *Year of the Hack? A billion records compromised in 2014*. [Online]  
Available at: <http://www.cnn.com/2015/02/12/year-of-the-hack-a-billion-records-compromised-in-2014.html>  
[Accessed 15 January 2016].
- Cult of Mac, 2014. *NFC on iPhone will only be used for Apple Pay*. [Online]  
Available at: <http://www.cultofmac.com/296093/apple-confirms-iphone-6-nfc-apple-pay/>  
[Accessed 10 October 2015].
- Dmytro, D., 2015. *Android File Grouping Plugin v1.1*. [Online]  
Available at: <https://github.com/dmytrodanylyk/folding-plugin>  
[Accessed 13 April 2016].
- EnterpriseAppsToday, 2015. *Mobile Apps: HTML5, Native or Both?*. [Online]  
Available at: <http://www.enterpriseappstoday.com/management-software/mobile-apps-html5-native-or-both.html>  
[Accessed 14 October 2015].
- Firebase, 2016. *Firebase*. [Online]  
Available at: <https://www.firebase.com/>  
[Accessed 18 March 2016].
- Fitzgerald, B., 2014. *How Long Does it Take for your App to be Approved?*. [Online]  
Available at: <https://www.appmakr.com/blog/how-long-app-approved/>  
[Accessed 10 October 2015].

Fortune, 2014. *Apple's users spend 4x as much as Google's - Fortune*. [Online]  
Available at: <http://fortune.com/2014/06/27/apples-users-spend-4x-as-much-as-googles/>  
[Accessed 14 October 2015].

Google, 2015. *Google Play*. [Online]  
Available at: <https://play.google.com/store?hl=en>  
[Accessed 14 October 2015].

Google, 2015. *New to the Google Play Developer Console? Learn the Basics*. [Online]  
Available at: <https://support.google.com/googleplay/android-developer/answer/6112435?hl=en>  
[Accessed 14 October 2015].

Google, 2015. *SQLiteDatabase*. [Online]  
Available at: <https://developer.android.com/reference/android/database/sqlite/SQLiteDatabase.html>  
[Accessed 28 December 2015].

Google, 2016. *Activity Lifecycle*. [Online]  
Available at: <https://developer.android.com/reference/android/app/Activity.html>  
[Accessed 30 April 2016].

Google, 2016. *Code Style for Contributors*. [Online]  
Available at: <https://source.android.com/source/code-style.html>  
[Accessed 13 April 2016].

Google, 2016. *SensorEvent*. [Online]  
Available at: <https://developer.android.com/reference/android/hardware/SensorEvent.html#values>  
[Accessed 24 April 2016].

Google, 2016. *Testing Concepts*. [Online]  
Available at: [https://developer.android.com/tools/testing/testing\\_android.html](https://developer.android.com/tools/testing/testing_android.html)  
[Accessed 20 April 2016].

Hale, D., 2016. *Guide to Universal Windows Platform Apps*. [Online]  
Available at: <https://msdn.microsoft.com/en-us/windows/uwp/get-started/universal-application-platform-guide>  
[Accessed 18 March 2016].

Healthcare Market Resources, Inc., 2008. *Why is Large Market Share Important? - Healthcare Market Resources, Inc.*. [Online]  
Available at: <http://www.healthmr.com/january-feature/>  
[Accessed 10 October 2015].

Hillyer, M., 2005. *An Introduction to Database Normalization*. [Online]  
Available at: <http://users.dcc.uchile.cl/~mnmonsal/cc42a/guias/intronorm.pdf>  
[Accessed 17 April 2016].

Honigman, B., 2015. *Which Operating System should I choose for my App?*. [Online]  
Available at: <http://getappcase.com/blog/beginners-tips/which-operating-system-should-i-choose>  
[Accessed 14 October 2015].

IDC, 2015. *IDC: Smartphone OS Marketshare 2015, 2014, 2013, and 2012*. [Online]  
Available at: <https://www.idc.com/prodserv/smartphone-os-market-share.jsp>  
[Accessed 7 October 2015].

Inukollu, V. N., Keshamoni, D. D., Kang, T. & Inukollu, M., 2014. Factors Influencing Quality of Mobile Apps: Role Of Mobile App Development Life Cycle. *International Journal of Software Engineering & Applications*, 5(5), pp. 15-34.

Link, J., 2003. *Unit Testing in Java: How Tests Drive the Code*. 1st ed. Burlington: Morgan Kaufmann.

Spoov, N., 2015. *Pew survey shows 68 percent of US adults now own a smartphone*. [Online]  
Available at: <http://www.pcworld.com/article/2999631/phones/pew-survey-shows-68-percent-of-americans-now-own-a-smartphone.html>  
[Accessed 16 March 2016].

Microsoft, 2015. *Account types, locations, and fees*. [Online]  
Available at: <https://msdn.microsoft.com/en-us/library/windows/apps/jj863494.aspx>  
[Accessed 14 October 2015].

Microsoft, 2015. *Windows Dev Centre*. [Online]  
Available at: <https://dev.windows.com/en-us>  
[Accessed 14 October 2015].

MoovWeb, 2015. *Are iOS Users Still More Valuable Than Android Users?*. [Online]  
Available at: <https://www.moovweb.com/blog/are-ios-users-still-more-valuable-than-android-users/>  
[Accessed 28 December 2015].

Neagu, C., 2016. *What are Universal Windows Platform (UWP) apps?*. [Online]  
Available at: <http://www.digitalcitizen.life/simple-questions-what-are-universal-windows-platform-uwp-apps>  
[Accessed 18 March 2016].

Nichols, J., 2015. *9 Must-Haves for a Successful App*. [Online]  
Available at: <https://apsalar.com/blog/2015/08/9-must-haves-for-a-successful-app/>  
[Accessed 27 December 2015].

Parse, 2015. *Parse Core*. [Online]  
Available at: <https://parse.com/products/core>  
[Accessed 28 December 2015].

PhoneArena, 2014. *Did you know how many different kinds of sensors go inside a smartphone?*. [Online]  
Available at: [http://www.phonearena.com/news/Did-you-know-how-many-different-kinds-of-sensors-go-inside-a-smartphone\\_id57885](http://www.phonearena.com/news/Did-you-know-how-many-different-kinds-of-sensors-go-inside-a-smartphone_id57885)  
[Accessed 28 December 2015].

Plaxo, 2011. *Plaxo Mobile Trends Study*. [Online]  
Available at: <http://pressroom.plaxo.com/plaxo-mobile-trends-study-infographic/>  
[Accessed 15 January 2016].

Saleem, H., 2015. *Adduplex: Windows 10 Mobile running on 8.8 percent of Windows Phone devices*. [Online]  
Available at: <http://www.winbeta.org/news/adduplex-windows-10-mobile-running-8-8-percent-windows-phone-devices-several-unreleased-phones-spotted-well>  
[Accessed 18 March 2016].

Samuels, L., 2015. *Why a Great App Isn't Limited to One Kind of Audience*. [Online]  
Available at: <https://www.appmakr.com/blog/app-audience/>  
[Accessed 27 December 2015].

Spoonacular, 2016. *Spoonacular Food and Recipe API*. [Online]  
Available at: <https://spoonacular.com/food-api>  
[Accessed 18 March 2016].

Statista, 2016. *Number of Smartphone Users worldwide from 2014 to 2019 (in millions)*. [Online]  
Available at: <http://www.statista.com/statistics/330695/number-of-smartphone-users-worldwide/>  
[Accessed 16 March 2016].

Tiwari, N., 2014. *How open sourcing Android made it a mobile market leader*. [Online]  
Available at: <https://opensource.com/business/14/7/how-open-sourcing-android-made-it-mobile-market-leader>  
[Accessed 28 December 2015].

TopTenReviews, 2011. *Demy Review*. [Online]  
Available at: <http://recipe-reader-review.toptenreviews.com/demy-review.html>  
[Accessed 17 March 2016].

Triggs, R., 2015. *How far we've come: a look at smartphone performance over the past 7 years*. [Online]  
Available at: <http://www.androidauthority.com/smartphone-performance-improvements-timeline-626109/>  
[Accessed 16 March 2016].

Varshneya, R., 2013. *4 Essentials for Successful Mobile Apps*. [Online]  
Available at: <http://www.entrepreneur.com/article/226732>  
[Accessed 27 December 2015].

Vaz, A., 2015. *Why Mobile App Personalisation Matters*. [Online]  
Available at: <https://blog.onliquid.com/why-app-personalisation-matters/>  
[Accessed 27 December 2015].

Viswanathan, P., 2016. *Why You Necessarily Need a Mobile App for Your Small Business*. [Online]  
Available at: <http://mobiledevices.about.com/od/additionalresources/tp/Why-You-Necessarily-Need-A-Mobile-App-For-Your-Small-Business.htm>  
[Accessed 16 March 2016].

Wikipedia, 2015. *Android Studio*. [Online]  
Available at: [https://en.wikipedia.org/wiki/Android\\_Studio](https://en.wikipedia.org/wiki/Android_Studio)  
[Accessed 10 October 2015].

Wikipedia, 2016. *Universal Windows Platform*. [Online]  
Available at: [https://en.wikipedia.org/wiki/Universal\\_Windows\\_Platform](https://en.wikipedia.org/wiki/Universal_Windows_Platform)  
[Accessed 18 March 2016].

Wilde, B., 2013. *iOS vs Android vs Windows: The Battle of the Mobile OS*. [Online]  
Available at: <https://blog.udemy.com/ios-vs-android-vs-windows/>  
[Accessed 28 December 2015].

Xamarin, 2015. *Xamarin Platform Page*. [Online]  
Available at: <https://xamarin.com/platform>  
[Accessed 28 December 2015].

Yummly, 2016. *Yummly Recipe API*. [Online]  
Available at: <https://developer.yummly.com/>  
[Accessed 18 March 2016].