Project Title

Tanvi Gavali

20th April 2022

A project report submitted in partial fulfilment for the degree of

BSc (Hons) Software Engineering

School of Psychology and Computer Science

University of Central Lancashire

Abstract

This project approaches the problem of food wastage and helps in time management by creating meal plans which can reduce the time for making meals but also make users more aware of what food they have bought and how to use it most efficiently (without throwing out expired ingredients.

Food wastage has been a major issue in the world, with estimates of 930 tonnes of food being wasted per year (McCarthy, 2021) and 17 percent of total food production wasted (United Nations, n.d.). This is a tremendous waste of both money but also people's efforts and labour.

This solution provides incentives to the user (saving money, saving time on planning meals, not having to throw away food, more control over their diet, etc.) using convenience, but also provides advantages that could benefit everyone such as a lack of waste, and healthier living.

My solution has been successful in producing meal plans that outline ways to control a user's diet, and automatically process what they can consume, whilst also taking into account expiry dates of products so that meal plans produced do not have unintended wastage. Meal plans that cannot be complete are notified to users. Further testing on a larger scale could reveal how this affects household spending in customer's lives, as well as the longer term effects on what they eat.

Attestation

I understand the nature of plagiarism, and I am aware of the University's policy on this.

I certify that this document reports original work by me during my University project. I also confirm that I adhere to the University's legal and ethical guidelines for undergraduate projects in Computing.

Signature: ____  _____

Date: 04/04/2022

Acknowledgements

I would like to thank the academic staff especially to my supervisor, Matthew Paul Leslie Horton, who not only guided me and gave me feedback throughout stages of this project but also motivated me to complete my project within the time frame. I would also like to thank the module leader for this project, Nicky Danino and the course leader, Nicholas Philip Mitchell.

I would also like to thank, Mrunal Gavali, for believing in my project at time where I lost hope and helping me with my time management and Aaron Armstrong for listening to me patiently every time I was having difficulty with perseverance, believing in me and pushing me to completing this project.

I would also like to express my thanks to all my peers and friends.

I would also like to express my gratitude towards my parents who have supported me, encouraged me and believed in me through all these years and helped me pursue in software engineering field.

And finally, I would like to say thank to all the online websites, documentations and blogs explaining the concepts.

Table of Contents

List of Figures

List of Tables

List of Listings

1 Introduction

1.1 Background

This project aims to keep track of each customer's fridge inventory and generate personalised meal plans each week. As a student, I generally find it very challenging to sort my fridge inventory from remembering each item's expiry dates to thinking what meal I can make today from the items I already have. The application purpose is to reduce the amount of food waste and avoiding over buying stock, as well as save customer time so they don't have to create meal plans themselves. It also pushes customers to create a healthy diet plan and control the portions in their lifestyle by giving them personalised options like calories intake, diet category, etc. To save time, the application includes user preference total cooking time and will be recommended personalised recipes. This application would be a desktop application deployed into a windows wizard installer. There are not many applications which personalise meal plans and those that do cost extortionate subscriptions per month (over £50). I am hoping to fill a gap in the market, in this respect, creating an affordable option.

1.2 Scope and Objectives

- A desktop application which takes customer information from a local grocery database
- Able to store favourite customer recipes and other personalised options.
- The API used in this project should integrate efficiently with the app and retrieve appropriate recipe data.
- The recommender system should find similar users and accurate suggested tags
- The application should provide a suitable personalised meal plan
- Easy to understand, functional and appealing User Interface (UI)
- Deployed in an installer properly

1.3 Achievements

This project has achieved all its major features discussed above. The application I have created was able to connect with a mock local database I created, which stores in the user details and items with storing new recipes from the API. The application was also able to use this data and generate a unique meal plan depending on the ingredients the user has, with considering the best before dates, calories and quantities. The app takes the user details from the database since the main plan of this project is to collect ingredients each user had bought from any grocery store and connect their data directly through the app to allow generation of a suitable meal plan. Users can also check if the ingredients they have would last them for a week and manage calories and cooking time as they want. The application allows user to get more recipe data from the TastyAPI and allows them to favourite the recipes they liked. Based on the favourites, the app also recommends similar recipes they might like based on similar user and based on cuisines. There is a huge scope for adding more features to this project like controlling the meals with servings, adding multiple users so to make a very personalised meal plan for either a family or couples, and technical based like adding cloud databases like AWS and Azure.

1.4 Report Overview

Chapter 1: Introduces the basic understanding of the project and the objectives.

Chapter 2: Evaluates using thorough research on the background for this type of project and any related work such as recommender systems and those that use APIs

Chapter 3: Detailed planning taken before starting this project. Includes methodology this project will follow on, user stories and MoSCoW Analysis to understand the requirements this project will meet, as well as an evaluation of the Agile methodology and Extreme Programming. Any potential solutions thought before considering the tools and techniques which will be used in project. And the last, any other legal and ethical issues I should be mindful before.

Chapter 4: Design section will cover detailed design of the system, including the Java application, deployment method, and database, as well as the UI of the project and designing for disabled users

Chapter 5: Implementation section will showcase the implementation of all features in the project with detailed explanation on how it was executed, including logic of algorithms, and use of libraries within the classes

Chapter 6: Testing will cover all test cases for each of the functionalities demonstrated in the project and AB testing and ableness testing for users

Chapter 7: The last section will estimate on the whole project with detailed cover of any future work planned for this project.

2 Background and Related Work

2.1 Introduction

The purpose of this project is the generation of meal plans based on individual preferences and items people have in their fridge to facilitate time management. It is possible with the use of a grocery database to keep track of bought items and check those items with ingredients used in an API recipe caller. This section explores on the use of APIs, recommender systems and user interface design. It also reviews relevant literature about the user interface and colour space.

2.2 Application Programming Interfaces

2.2.1 What is an API?

This section will cover some research on APIs, how they work, their usefulness and implementation. Before heading into how this research is going to help in the prospect of this project, it is important to understand what APIs are in general. APIs, also known as Application Programming Interfaces, allow two applications to talk to each other. These applications can be in any programming language regardless of the platforms they use and hence APIs have been popular for a long time and have found use on large websites such as Twitter, Netflix, and Facebook (Jacobson et al., 2012). They often connect with a database and provide an easy access to the developers. Therefore, they can save developers a lot of time. In this project, it is required for me to access recipes from the APIs so as to manipulate that data to generate meal plans.

2.2.2 The use of APIs

APIs are efficient since they can be useful to companies, developers, clients, and customers. I will be writing about its usefulness briefly in this chapter.

APIs can be useful for **companies** since it allows them to give some data to its users without exposing the confidential information, as well as develop apps faster without spending as much time populating products with data (Jacobson et al., 2012). It also allows them to keep their algorithms and techniques secret, which can also help reduce dependencies (de Souza & Redmiles, 2009; Jin et al., 2018). Companies can get analytics of their users by checking the number of people using their APIs and getting information from their API calls. Some companies have private formats and keys to access their APIs and thus information on the internet can be protected and used only for their clients. Thus, it helps in preventing the misuse of information by external factors (McLellan et al., 1998).

From the perspective of the **developer**, APIs make it easier to implement new complex algorithms and save time in implementing existing algorithms by hiding the complexity underneath the API (de Souza & Redmiles, 2009). It also takes out the strain of storing a lot of data by themselves and makes searching for the data easier.

Since APIs store data, it allows every **user** to access info without the delay of their device looking through it. In the case of my project, it makes running the program effective as the webserver ends up taking the heavy tasks like accessing bulk recipe lists. Users don't have to put efforts while searching and putting in their own recipes. Customers are also not constrained and thus get all google recipes allowing them a variety.

2.2.3 How APIs work?

APIs have sets of rules on how computers/applications/software can talk to each other. These rules pave ways for getting the data from the webserver to the program. APIs usually have a database

which stores the relevant information. Private keys are needed to set a communication with an API and are accessible through methods and requests (GET, POST, etc.) (Cloud Endpoints, n.d.). Many APIs online provide documentation for these methods to make it easier to access.

APIs also involve sending data from and to the webserver. This is often done using file formats such as JSON and XML which can encode and decode data from within the programs and databases.

To check the input and output with the API, many frameworks have been created. In this project I will be using Postman as a testing API tool framework, which allows me to connect to the API easily and pass parameters on method calls to get response. This allows to separate code and checks input factors like the origin code, private key, etc., to get connected to the API.

Figure 1: below demonstrates the workings of the APIs integration with the project.

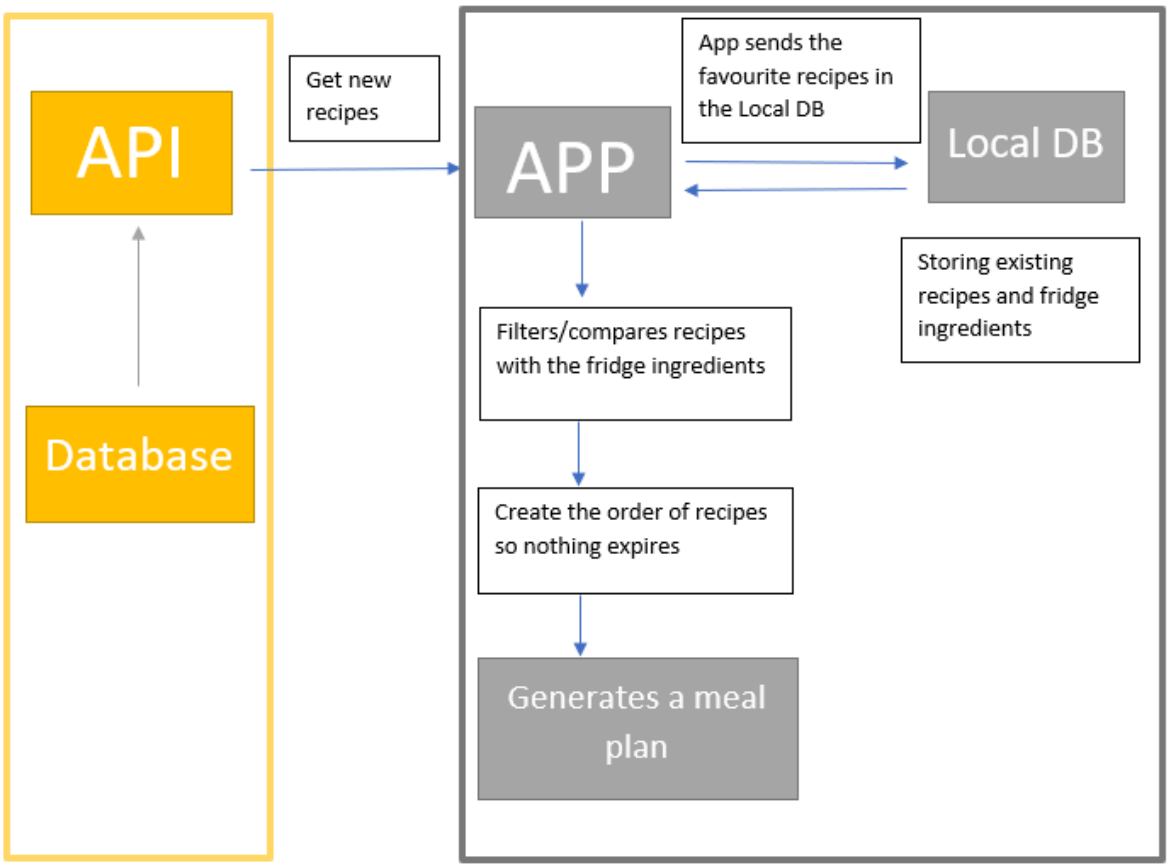


Figure 1: API work in my project

2.2.4 Implementation of API (connection)

Connect to the API - Usually, programming languages require a connection to be made in the form of a connection objects. For example, in Java there is the HttpUrlConnection class, or since Java 11, this has been replaced with HttpClient (Baeldung, n.d.). This connection/client will typically be used to create an initial connection as well as execute requests in future.

Send requests - Requests can be built separately and will include information such as the URL being accessed, parameters, request type (POST, GET, etc.), and other information about how the request will be run. In Java, this is done using the RequestBuilder object (which implements the builder design pattern), to set these details with functions such as RequestBuilder.setHeader() and

RequestBuilder.setUri() (The Apache Software Foundation, 2021)). Finally, these will be built using .build(), and the request can be run by the client using HttpClient.execute().

Receive requests - The response returned by the website (if any) will be captured by a HttpResponse object. This class has been made to handle HTTP responses and has two important fields, HttpResponse.body, and HttpResponse.statusCode. The status code will give information such as whether the request was successful and can be handled using if statements or a switch statement. The body will include the information returned by the webservice. This could, for example, be a JSON file which could then be decoded for the program.

2.2.5 Types of API

There are different types of APIs which will be categorised into protocol types and main types. Protocols /Architecture types are different approaches based on structure and methods to build APIs. Main types include public, partner, private and composite APIs.

2.2.5.1 Protocol/Architecture Types

2.2.5.1.1 REST

REST stands for Representational State Transfer Approach and is one API paradigm. It is considered the most popular API paradigm and is used by many high-profile companies such as Facebook, YouTube, Twitter, and Google.

REST APIs are considered less structured than APIs following other paradigms, and this is because the REST paradigm provides an approach to architecture, but does not have a specific protocol, unlike SOAP, meaning that the messages themselves are less structured due to them not following a set protocol. The stateless nature of REST APIs also further reduces how structured the format of REST APIs is (Jin et al., 2018).

The REST API does have other requirements, including the implementation of CRUD, standing for "Create, Read, Update, and Delete", which requires the implementation of each of these verbs in the API regarding its data. This means that the functionality provided by the REST API must provide the ability to use requests and methods to perform all these verbs on the data sent between the server and client. Data can be created and deleted, but also accessed and changed.

A REST API also uses lower bandwidth than other APIs because of how messages are structured. Firstly, the lack of structure means that the messages themselves are smaller than SOAP API requests. Not only this, but also a REST API must support caching, which means that the client can remember some data, meaning fewer API requests are sent to the server (Techolution, 2019).

A REST API is considered "stateless". This means that the REST server does not keep track of its clients, and clients can access resources from anywhere in their usage, so long as they have the correct access. This contrasts to RPC and SOAP, which will interpret requests differently based on previous packages sent (Ranga & Soni, 2019).

2.2.5.1.2 SOAP

The SOAP API framework is more rigid and structured than REST and has more specific rules regarding the structure of requests sent between the client and server. This includes a set file type (XML extension), fixed fields such as Header and Body, and added security requirements (Techolution, 2019). It was the first paradigm to standardise how applications manage network connections for their services, and is an official protocol defined by the World Wide Web Consortium (W3C).

The SOAP API framework only uses XML and is stricter and more structured than REST. Its name stands for Simple Object Access Protocol. It has several main necessary XML style elements, four of which are necessary for the API to be considered SOAP (IBM, n.d.). One of these elements is the "Envelope", which is a mandatory root which contains all other elements within it. The "Header" element is an optional sub-element of Envelope, which passes application-related information to be processed by SOAP nodes along the message flow. The "Body" is an element which contains the main information of the message and is used to store the data which will end up being opened by the user. The "Fault" element is a sub-element of the body and is part of the SOAP API framework's enhanced error reporting abilities. These contrast to REST which has less of a rigid structure and is more used as a guideline of rules (Ranga & Soni, 2019).

2.2.5.1.3 RPC

RPC is another form of API paradigm which is primarily focused on actions. RPC stands for Remote Procedure Call, and this relates to the procedural nature of the paradigm, where methods are more important than resources. The main rule of the paradigm is that the server contains names of methods to be executed, and the client calls these (Jin et al., 2018). This is a lightweight form of API which is still used today by some popular companies, such as Slack (Slack, n.d.).

RPC APIs are the oldest form of API, and often considered the most simple and easy to understand APIs due to being (similar to REST) not rigidly structured and are more about general rules than defining a protocol. Also similar to REST APIs, RPC can implement either JSON or XML, meaning that the format of the API is more flexible and will be familiar to more developers. Unlike REST, RPC APIs do not have rules such as adhering to CRUD, or caching, and instead primarily focus on implementing methods (Techolution, 2019).

Due to the procedural nature of RPC APIs, the requests sent in RPC are highly lightweight as they are procedure-oriented instead of focusing on data. However, there are some drawbacks to this as well, as they can be subject to "function explosion" (Emmersberger, 2019) due to the reliance on procedures, as well as being hard to structure due to a lack of standardisation.

2.2.5.2 Main Types of APIs

Open/Public APIs are where the data is not restricted and can be used freely. For example, Google Maps. Sometimes these APIs cost per search. Example, SerpAPI used in this project is a public API and starts its prices when there are more than 100 searches a month. These APIs allow developers to enhance their skills in using APIs, make free open-source projects and help small companies.

Partner APIs are APIs which can only be used when given access to. The developer needs consent or a certain license to use this API. These can be used between different companies or partnered groups. For example, take two companies A and B hypothetically exists. Company A holds data of a private API. This API is given to Company B in the form of partnership at a certain price. Company B makes a profit as well since other competitors or people will not have access to this data. Thus, Partner APIs benefits both the companies/groups (Boyd, 2014).

Internal/Private APIs are secret APIs usually used within companies internally and are more prevalent than public APIs (Jacobson et al., 2012). These types are usually dependent on the kind of information the APIs have and who gets the right to use them. The data in these APIs are confidential and are given access to only certain members of that company/group/organisation. This permits a company to modify the data in the API to their needs and does not put strain on external security risks. Example, Amazon Web Services provides private APIs using Amazon API Gateway which are isolated from the public internet (Amazon Web Services, 2021).

Composite APIs combine two or more APIs together and act as a single API to increase the tasks called and to save time. It can make multiple calls in a single API request and return response and status in a single response body (Salesforce, 2021). It is efficient in desktop/web applications in compressing the number of tasks into a single task to reduce the complexity and time for the user.

2.3 Recommender system

2.3.1 What is a recommender system?

Recommender systems are algorithmic approaches which personalise each user's preferences. It makes the user feel like the software understands them and remembers their likes and dislikes.

They were originally called "Collaborative Filtering" until the name Recommender System was first introduced in the 1990s. The first workshop on recommender systems was run in 1995 (Konstan, 2004), but the methods used in these systems can still be referred to as CF, especially for those not based on neural networks.

Two common types of collaborative filtering are item-based CF, and user-based CF (Park et al., 2012). The former works by comparing similar items in a database and recommending to a user based on which items they prefer (by rating, clicks, etc.) and this can be done locally. The latter ("user-based") requires multiple users and will recommend items based on what similar users have preferred. Figure 2 demonstrates the difference between comparing the users (Collaborative Filtering) and the content (Content-Based Filtering).

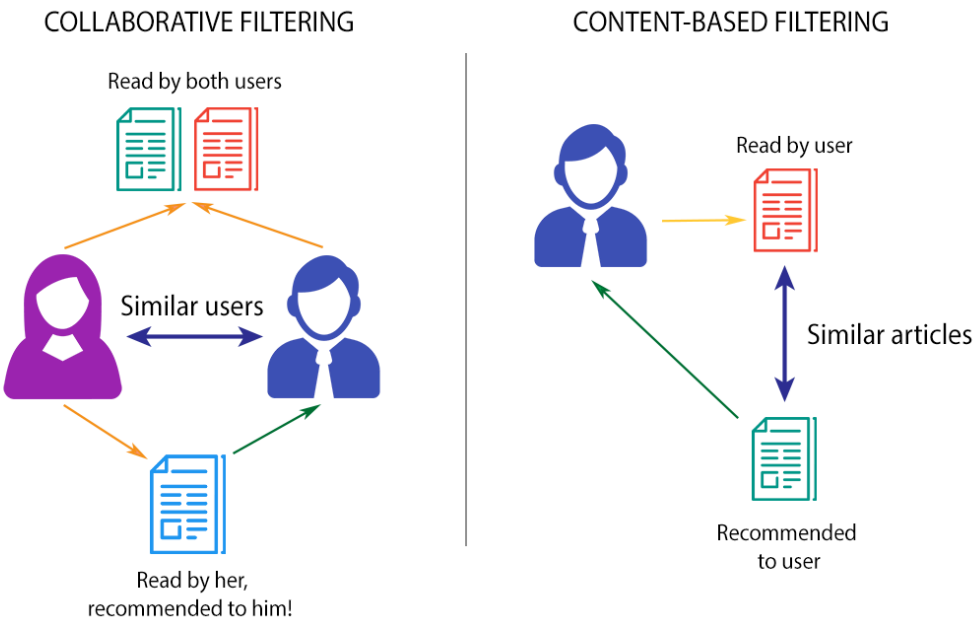


Figure 2: Collaborative Filtering and Content-Based Filtering from

(Liao, n.d.)

2.3.2 Why use a recommender system

These systems have been useful for almost a decade now used by big companies like Amazon, Netflix, etc., to make people realise that they identify their customers on personal levels. In this sub section, we would be covering about how recommender systems have been useful to the companies and customers.

Even early recommender systems have proved useful and appreciated by users in research groups with the earliest versions of algorithms to find neighbouring data (user-based and item-based with nearest-neighbour and ratings) (Konstan, 2004).

Recommender systems help **companies** to keep track of useful user data (for example, what products are popular) which could be sold. Thus, helping to build relationship with increase profitable sales. These also improves the user interaction with the software by making them spend more time and buy more subscriptions, etc.

These systems as per related to the topic of my project, it helps the **users** to stick to a nutrition plan as it will recommend better healthy food. It will also help users remember what they enjoy and help suggest new food they might like.

2.3.3 Who use recommender systems?

Nowadays, big, and small companies have been using recommender systems. Amazon, Netflix, LinkedIn are well known companies known for using recommender systems. This has allowed them to communicate with their customers and increase their customer satisfaction.

2.3.4 How they work

There are different approaches to make these systems work depending on the output needed. The general idea of these systems is to collect their user and content data and later compare with related data or other know users to find similarities. They score their content based on these similarities and show possible new content to the users. The image below demonstrates how the working of a recommender system works. There are several forms of recommender system, and while all of them share in common the process of comparing data and finding similarities, there are differences in the data and implementations.

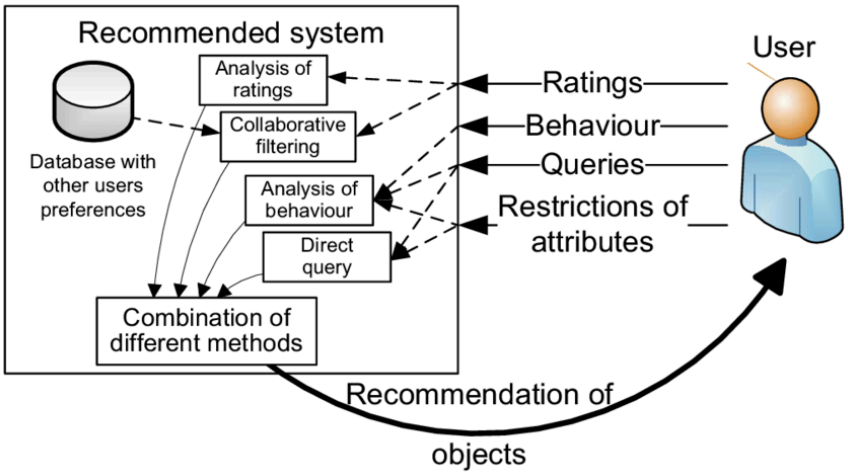


Figure 3: Working of the recommender system

(Eckhardt, n.d.)

2.3.4.1 Data used

The foundation of recommender systems is to compare data and find similarities. This is used to decide which content should be suggested to users. However, the data which is used will vary

between different systems. When users are compared, this will often be similar, but the company-owned data could be anything from movies to books to restaurants. In my case, I will be using recipes and ingredients as my primary data.

Content data can be varied depending on the type of software and its customers. Not only is the data itself variable (Netflix will compare movies, TripAdvisor might compare hotels and holiday destinations), but the data belonging to those items can vary too. There are some systems that have the same data, such as likes and favourites.

2.3.4.2 What is done with the data

Different metrics can be used to evaluate data. Firstly, there are ways of categorising the data itself so that item-lead recommendations will recommend similar items. For example, if a user likes the movie "Terminator", they might be recommended the movie "Rambo", as both are action movies. Similarly, if a user likes the movie "Terminator", they may be recommended the movie "Twins", because both movies star Arnold Schwarzenegger. These two movies may be given "weights", or "scores" based on how likely the user is to enjoy them. Their score will be increased because of the similarities in genre and actor. However, these scores can also be cumulative, and because of this, the movie "Predator" may be recommended because it is both an action movie and also stars Arnold Schwarzenegger, therefore it may have a higher ranking. This would fall under the category of "Tagging" recommendation systems, as the tags of genres and actors will be applied to the data.

Another way of finding similarities would be to find common items between users, and to suggest new items based on those. For example, if User A likes the movie "Hunger Games", and User B also likes this movie, the system may recommend "Squid Game", another item that User B has watched. This is not only limited to items that both users have watched but could also be used for movies that both users have enjoyed in some way, and this could be represented through metrics such as "favourited" or "liked" movies. This is especially prevalent on social media, where metrics of engagement can be measured, including also "comments" and "clicks". Other metrics may also be measured, for example seeing how long a user will watch a video for on Facebook or Twitter and giving it an engagement rating based on that (Baltrunas & Ricci, 2009; Symeonidis et al., 2007).

These recommendations can also be controlled by the user in some cases. For example, User A on Twitter might be interested in User B. User A can then view who User B is following, which posts they have liked, and even search for the tags they are interested in.

2.3.5 What could work with my project

This research is important for my project since based on their most favourited and rated recipes I would have to recommend recipes in their meal plan based on the recommender system. Because of the time constraints and other features that need to be implemented in my project, I will not be using a neural network as this would take too much time to develop. However, as my project is going to already involve a database, I will use that database to store metrics such as "favourite recipes" and can use this in conjunction with the user database. Because these fields are already useful due to them being needed to remember recipes for convenience, it will be useful for me to use these in a recommender system which will suggest recipes based on users who have both favourited the same recipes.

I can also implement a tagging system for offline users. This would be convenient in scenarios where there are not multiple users, and I can let the system suggest recipes to users based on if the recipes they already like share similar tags. For example, recipes could include tags such as "Snack food" or

"Korean cuisine" or "Healthy eating". If a user has eaten a salad and favourited that recipe, "Red Lentil Dahl" might be suggested in future as another recipe which includes "Healthy eating" as a tag.

2.4 User interface design

User Interface design is a much-researched topic which provides methodologies to developers based on what makes a good design to give customer satisfaction to their users. Its user goal is to help those using the program to communicate with the application easily and intuitively, and for the program to communicate back in a way that as many users as possible can understand (McKay, 2013). There are several key areas of user interface design, and the study of user interface design aims to make programs better for all people, including those of different culture and ableness.

2.4.1 Rules/guidelines of interface design

2.4.1.1 What is a guideline and areas of guidelines?

Guidelines are a bunch of rules which have been formatted through research. Some areas of guidelines are as follows:

Typography (font) - Typography includes the font size, colour, and shape. It plays a role in this project as it is important for the users to understand the right text displayed and without strain in their eyes. Inability to understand the letters will lack user interaction with the application. One important aspect of typography is choosing the correct font family for a given subject. For example, formal products such as the website of a law firm would use a serif font, but a computer-science oriented application, especially one with code as text, would use a monospace font (Young, n.d.).

Colours - Choosing the right colours for an application is essential to maintain the colour balance. Too bright colours have possibility of straining the users eyes and difficult combinations can make it hard for colour blind users. Taking all the different account it is very important to have the right palette of colours. I will cover about this in bit more depth in section 2.4.2.

Design of components (size/shape of buttons, textfields, icon, etc.) - Allowing a lot of open space with bad button sizes is not what users will be comfortable and neither the testing would be efficient. It is important to have sensible placed components and well tested functionalities of them to get customer satisfaction. Placing components like the buttons, text fields and icon with consistency size and shape allows the application to look more professional and understandable (Google, n.d.).

Spacing - Inappropriate spacing of objects and sections can lead to bad layout in the program and thus will be tricky to navigate. Consistent spacing is also important for readability. Pixel sizes are ideally powers of 2 (2, 4, 8, 16, 32, etc.). Some studies suggest that while white space is necessary and keeping whitespace consistent is important, using too much white space can also negatively affect usability, so a correct balance is needed (Coursaris et al., 2012). However, sometimes a large amount of whitespace can have benefits, such as how Google uses whitespace to make its homepage stand out (Schlatter & Levinson, 2013, p. 108).

Integration and Behaviour (of components) - Smooth interaction of components with suitable action listeners and events are important to have a well functioned app. Non consistent behaviours can confuse the customers and hence usability would be jeopardised.

Error and System Status - It is very important to have error and system status in logfile managers to keep track of the connections made with the app. It is best if users always know if something has

malfunctioned. When errors are displayed, they should be simple, intuitive, and ideally help users fix their problems (Wong, 2020).

2.4.1.2 Main aims/objectives for this project

Appealing design - To attract the users to use the program, productivity of the app alone won't be enough. Appealing designs trending with modern times are also very important (Schlatter & Levinson, 2013, p. 60).

Intuitiveness - Having right intuitiveness on what makes an application better in terms of better human interaction is essential. This includes being consistent by following standards of other programs so users will not be surprised.

Accessibility - It is important to make the application accessible to a wide range of users. If less abled users are not considered, this will reduce how many people can use the application.

2.4.1.3 Examples of common guidelines

I will be focusing on the different aspects of UI as listed above during development but will also compare my work against certain guidelines to evaluate my work.

Table I.1 The Two Best-Known Lists of User Interface Design Guidelines

Shneiderman (1987); Shneiderman and Plaisant (2009)	Nielsen and Molich (1990)
• Strive for consistency • Cater to universal usability • Offer informative feedback • Design task flows to yield closure • Prevent errors • Permit easy reversal of actions • Make users feel *they* are in control • Minimize short-term memory load	• Consistency and standards • Visibility of system status • Match between system and real world • User control and freedom • Error prevention • Recognition rather than recall • Flexibility and efficiency of use • Aesthetic and minimalist design • Help users recognize, diagnose, and recover from errors • Provide online documentation and help

Figure 4: Comparison between two know guidelines

(Johnson, 2014)

2.4.1.3.1 Golden rules by Schneiderman

Schneiderman's Eight Golden Rules is one UI guidelines which goes into detail regarding creating a good user experience, and I will be using it in my work. The guidelines involve rules such as having consistency in design, keeping a user engaged with features such as shortcuts and dialogues, and letting a user reverse their actions (Wong, 2020).

2.4.1.3.2 Nielsen and Molich's Ten User Interface Design Guidelines

I will also be using Nielsen and Molich's guidelines for UI which is similar, but is more extensive and also has further information on error handling as well as minimalist designs. Figure 4 above is a table which shows the comparison between these two known guidelines. There are also other guidelines such as Norman, Stone and Johnson.

2.4.2 Colour theory

2.4.2.1 What is colour theory?

Colour theory is a study to understand colours and set an overview of rules and guidelines, which facilitates in user interfaces (Fine, 2021). Over the time, people have come up with colours and shaders that suits well in any place. These often use “colour spaces” and rules to choose colours from them. It is useful because people are comfortable using this colour theory and hence allows the use of accepted colour patterns. Below are some features of colour theory which are covered in the guidelines.

- **Colour spaces** - Colour spaces are ways of defining ranges of colour. This includes variables such as hue, saturation, brightness. Rules and guidelines can be applied to these to pick colours.
- **Palettes** - Palettes is a very broad range to talk about. Sometimes palettes are all about different shades of each colour to about different colours going well with each other
- **Readability** - Allowing the components to be more readable and understandable is vital. Vibrant colours (bright or saturated) can show a user which buttons are more important. It is also very important to choose colours which would not be an inconvenience to colour blind people.

2.4.2.1.1 Complementary colours/Colour schemes

Complementary colours are easy to use and hence allows a wide range of options. Mixture of two colours usually cool and warm colours opposite each other in a colour wheel with correct proportions gives a complementary colour (Zakia, 2013). Most of the modern applications use these schemes in their user interface to draw their users. There are different colour schemes such as complementary, triadic, and analogous. Figure 5 shows different colour schemes on a hue colour wheel including complementary colours

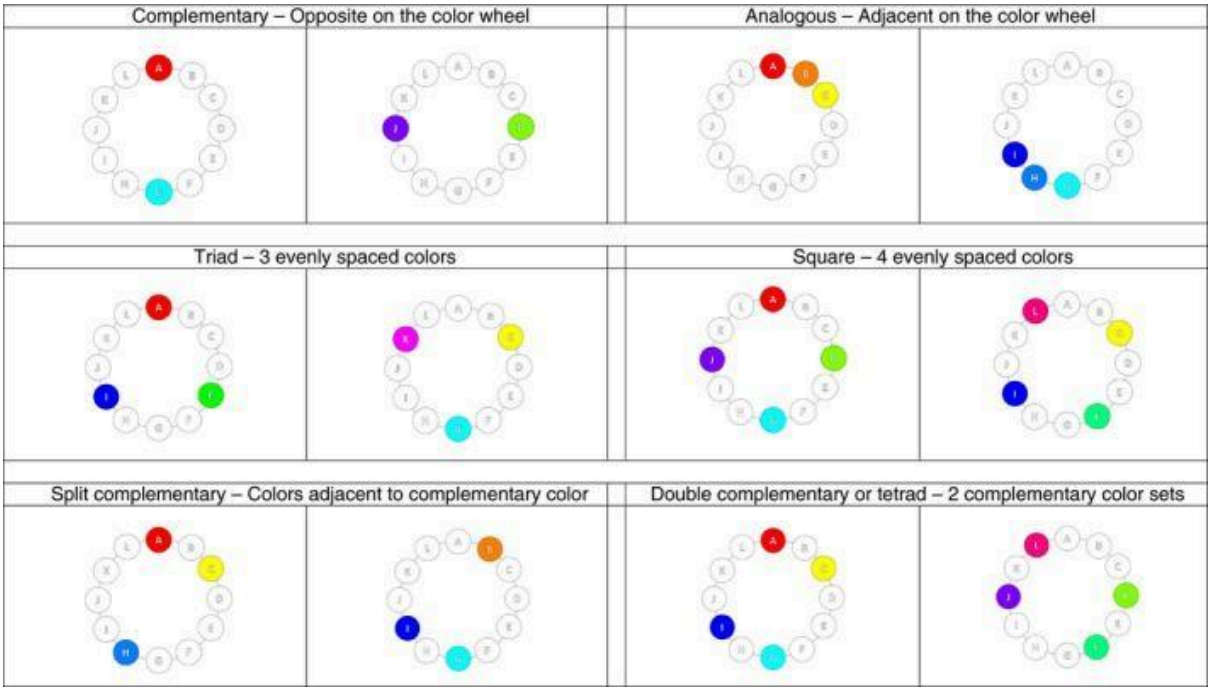


Figure 5: Colour Schemes

(Plante & Cushman, 2020)

2.4.2.1.2 Colour space coverage

The coverage of these schemes in a colour space can be measured, such as with the tool “DBToolbox” with its Palette Analysis feature. Figures 6 demonstrates insights into how a limited palette covers the colour spectrum.

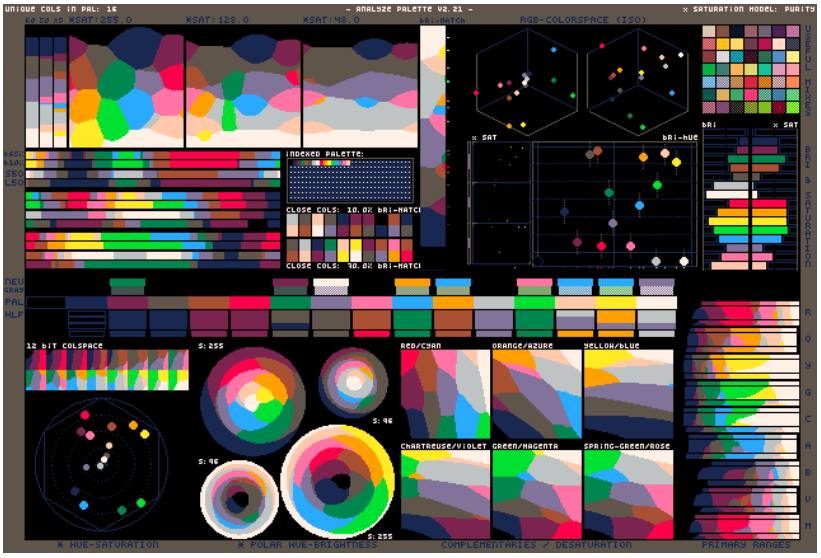


Figure 6: DB’s ToolBox Palette Analyser

(Fhager, 2017)

2.5 Summary

This literature review has given insights on information related to APIs, recommender systems and user interfaces. The use of APIs for companies, developers and users with their workings and implementations have been discussed. Findings on different types of APIs with examples have been formulated, which includes topics like REST, SOAP and RPC with main types such as public, private, partnered, and composite APIs. These allow enough understanding based on the API I will be using in this project. This study also covers an understanding of what recommender systems are and why they are useful to companies, developers, and users and how they work based on types of recommender systems. Deploying an understanding of these systems in my project and analysing how the data is being used has also been considered in this chapter. Finally, this section has also researched on the understanding of user interface with popular guidelines and colour theory including colour schemes and colour spaces.

3 Project Planning

3.1 Introduction

This chapter will cover the planning process and considerations taken before starting on this project. This includes methodology used, requirements that should be in the project and tools and techniques needed to make this project a success. It will also consider legal, social, and ethical issues so to understand the legal implications of this project before developing.

It is important to pre-plan iterative projects with techniques found in methodologies such as Agile and Extreme Programming, as it is proven that planning can increase the chance of completing a project cycle within dedicated time frames(Pinto & Prescott, 1990) (Bryson & Bromiley, 1993).

This chapter will go through the planning process of my project, including information on Extreme Programming (XP), the methodology which I have used, the requirements I have gathered in relation to my methodology (including User Stories).

This chapter will also discuss potential solutions for the project, including the language and frameworks used, as well as software to aid the production process (such as software to aid the database modification and deployment of the project). This will be followed by a discussion of the tools I have currently chosen for the project.

Finally, I will discuss the legal, social, and ethical issues of the project and how I will navigate them. This includes how user data will be used, and contractual obligations such as that of the university's intellectual property agreement, and the licensing of different software used for commercial projects. Project Proposal can be found in *Appendix 1* and Technical Plan can be found in *Appendix 2*.

3.2 Methodology

3.2.1 Introduction

I chose Extreme Programming (XP) because it is a light-weight, versatile and iterative methodology that can scale well with smaller teams (including of one person (Jeffries, 2006b)).

It is an agile-software-development methodology which focuses on shorter development cycles with incremental designs (Beck & Andreas, 2005) and planning to adjust to changing requirements (Novoseltseva, 2020). Methodologies like waterfall are not iterative and hence might be a drawback since my project needs constant changing as the tasks are adapted to user stories and iterative designs.

I also used several techniques from the Agile methodology and since XP is a part of Agile methodology adopting XP is much more optimal in time constraint deadlines (Digité, n.d.). adjusted my usage of it as Agile is best used for teamwork, where this was a solo project. Because of this I felt that XP is a methodology that will suit my project better, but I used methods from both. XP also reduces the risk because of the constant feedback allowing to make changes needed early before deployment.

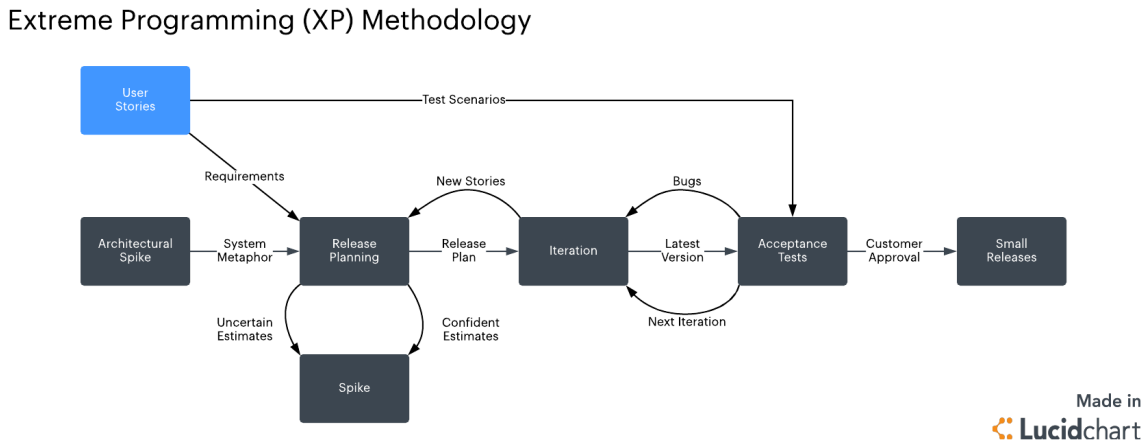


Figure 7: Extreme Programming overview

(Lucid Content Team, 2022)

3.2.2 Explanation

Extreme Programming (XP) starts with a release plan which I have shown in the *Appendix 3*. The release plan describes the overall project, and includes several User Stories, which describe the need for features. From these User Stories, programming tasks are created, and these are put into Iterations (Wells, 1999).

User Stories describe the need for different features in the project. They define real needs of users and how these needs will be met with a complete project (O'hEocha & Conboy, 2010). I have listed several of these in the section 3.3.1.

Iterations are part of the iterative methodology of Extreme programming, and part of why I chose this methodology. The iterations usually happen in periods of 2-3 weeks which worked well for me because I usually need 2 or 3 weekends to fully implement a new set of features. Each new Iteration will take User Stories and put them into programming tasks. Then I work on these until I can plan a new Iteration. As a student, discussion with my supervisor will be analogous to discussing a project as a group in a company.

The iterative nature of XP is good for this project as well, because my requirements are likely to change over time. Because of this, Iteration Planning will help me to adapt if the needs of the project change (Fowler & Taber, 2001).

3.2.3 XP For Smaller Teams

As explained above, part of the reason I chose XP is because it is versatile and can work with smaller teams, including teams of 1. One of the creators of Extreme Programming methodology, Ron Jeffries, explains this on the Extreme Programming Roadmap pages maintained by WikiWikiWeb (http://xp.c2.com/ExtremeProgrammingForOne.html) (Jeffries, 2006b).

He describes that although Pair Programming does not work for one person, most of the ideas are suited for small teams and can still apply to one person. I will use this to guide me when translating the ideas of XP into solo development.

I will use Continuous Integration Testing which can be used to continuously check that the program is suitable for the needs of a customer (as defined by my user stories). These tests will either be performed as test cases by me using the program, or by writing code that will test the classes (Jeffries, 2006a).

The extreme programming will be based on User Stories which are shown in the requirements section. Followed with a release plan shown in the *Appendix 3*. I will be planning my work in two-week iteration plan to get feedback from my supervisor. This way the basic values of XP, simplicity, communication, feedback, courage and respect are covered (Jeffries, 1998). I personally will be follow this website - http://xp.c2.com/ExtremeProgrammingForOne.html where one of the founders of XP, Ron Jeffries, give guidance on how one person projects can be built following this methodology. I have also constructed unit test and functional tests which are covered in *section 6. Testing*.

3.3 Requirements

I created user stories to understand what the user might need to run this application smoothly. Also due to the limited time given to complete this project, I created MoSCoW Analysis approach to prioritise the features in this project.

3.3.1 User Stories

The user stories have been created to take into account the user needs while using this application.

1) **As a user**, **I want to** be able to put login details **so that** I can log in the application.
2) **As a user**, **I want** the username and password the same as the one used to login for the grocery app **so that** I can see my recent bought grocery items
3) **As a user**, **I want to** view all the items with calories, quantity and best before date of each item bought from the grocery store, and items they have currently in their kitchen **so that** I can keep track of my items
4) **As a user**, **I want to** add and delete the items from the fridge table **so that** I can update my app when I throw out food or add new ingredient in my fridge.
5) **As a user**, **I want to** sort my ingredients in the fridge table with the soonest expiry date **so that** I know which ingredients are getting expired soon.
6) **As a user**, **I want to** sort the items from the table in the Fridge tab alphabetically **so that** I can search for things easily
7) **As a user**, **I want to** generate a meal plan from the items I have **so that** I do not have to worry about creating a meal plan myself.
8) **As a user**, **I want to** know if the items I have are not enough to create a weekly plan and the application should tell me to buy more ingredients **so that** I am settled for a week's food.
9) **As a user**, **I want to** get the recipe details when I double click on any recipe in the meal plan **so that** I do not have to search for the recipe and find the descriptions manually.
10) **As a user**, **I want to** save the current meal plan and generate a new meal plan for the next week **so that** I can roughly estimate how many food items I must buy.
11) **As a user**, **I want to** view a list of general recipes and be able to favourite any recipe I liked **so that** I can read other recipes which takes my interest
12) **As a user**, **I want to** create my own recipe and save or favourite them for later **so that** I have my own personal recipes to use.
13) **As a user**, **I want to** see my favourited recipes more in my meal plan generation **so that** the meal plan has the recipes I like.

14) **As a user**, **I want to** view all favourited recipes in a favourite tab and for it to suggest new recipes I might like **so that** I find more recipes of my taste.
15) **As a user**, **I want to** see my suggested recipes as per my cuisine **so that** the recipes shown are suited to my taste.
16) **As a user**, **I want to** search for any recipe or ingredient which should be displayed and remembered **so that** I can later read those recipes.
17) **As a user**, **I want to** able to update my password in my settings tab **so that** I do not have to worry in case I forgotten or need to change my password details.
18) **As a user**, **I want to** able to change any preferences **so that** my meal plan is customised based on how I like it.
19) Atlas, **as a user**, **I want to** able to log out of the application **so that** nobody else use my application.

3.3.2 Release Plan

Creating a release plan for the project included splitting the user stories into individual programming tasks and making estimates for these. After this, I created priorities for the different tasks and decided which ones to work on. I have also created a breakdown of Priorities, Values, Risks, and Estimates and shown this in bar-chart format to show the distribution of what kind of tasks have been created from user stories.

Usually, a release plan would involve group activities to estimate the time a task will take and also what tasks should be prioritised, but I did this individually (Wells, 1999). A detailed release plan is showcased in *Appendix 3*.

3.3.3 Moscow Analysis

Following on from the literature review, I realised some requirements that would be needed for the project, based on what similar projects require as components, and have revised my MoSCoW analysis to account for this knowledge.

For example, I learned that for a recommender system, I will not only need a way for the database to store user preferences when it comes to content (favourites) for collaborative filtering, but for content-based filtering I will also need to have data for items. For this I will add cuisine tags to the recipes so that there is a way to compare similar items when suggesting new recipes.

For UI, I learnt more about colour schemes and the requirements for both generally appealing colour schemes as well as colour schemes that work for blind people. For this I will make sure to include a requirement for comparing colour schemes and simulating if they will be visible for all groups of people.

Must have:

1) A working prototype demonstrating the conversion of ingredients into a meal plan

2) Meal plans separated into reasonable breakfast/lunch/dinner meals, and ordered from monday-sunday

3) A database hosting customer, ingredient, and recipe information, supporting CRUD (Create, Read, Update, Delete) operations

4) An SQLManager component of the Java application which serves as a library to run all relevant queries to the SQL database

5) An extensive layout using the Swing graphics framework for Java

6) An APIManager which can retrieve new recipes from an API using HTTP responses, so they can be stored locally

7) Consistent, well organised code with appropriate use of design patterns

8) Libraries maintained so that they can be expanded upon and re-used in future

9) Project successfully deployed in a Runnable JAR for easy installation

10) A Class Domain Diagram to demonstrate the complete scale, hierarchy, and structure of the project

11) A UI and colour scheme that has been thoroughly tested for different ableness for colour blind people

Should have:

1) Database stores expiry dates of ingredients, and this is used when calculating meal plans

2) Change recipe option to switch which recipes are chosen in a meal plan

3) Feature to allow users to add their own recipes to the database, which can be generated in a meal plan

4) Use the Graphics2D library to create new versions of Swing components which look more modern and professional

5) The project should be deployed using an installer, via jpackage to create a Runnable JAR

Could have:

1) Multiple users supported (for friends and family), supported each with their own ingredients and favourites

2) Shared meal plans generated per household

3) Allow users to enter preferences such as diet (calorie count), desired recipe difficulty, and religious needs (no beef, no pork, etc.)

4) Create new versions which can be deployed to mobile or on web

5) A recommender system to provide new suggestions for recipes based on favourites (based on recipe tags)

5) A recommender system to provide new suggestions for recipes based on favourites (based on similar users)

Would have:

1) Information of user preferences, favourites, saved recipes, etc. could be stored on a cloud service such as AWS for backup

2) Usage of AWS QuickSight or similar to create analytics and provide neural network machine learning for user recommendations

AWS S3 data can further be converted into graphs using the AWS Quicksight to help the recommender system to learn from its users. This will also allow the developers to understand how many hours users spend on the program, is the program recommending the right meal plans and how many users prefer intercontinental meal plans.

3.4 Potential Solutions

3.4.1 Language

I evaluated multiple languages for this project. The languages I have used and considered were C++, C#, Java, and Python.

I have decided to use Java for multiple reasons. It is the language I am most comfortable with after my industrial placement. Because of this, I also know some details about it which are good reasons to use it for this project.

Firstly, Java has a good GUI framework for me to work with, which will save me a lot of time when implementing the many different screens for this project. Also, because this project does not have a graphical focus, it is not necessary for me to spend a lot of time working on the display. Java has a framework called Swing, which handles several aspects of GUI such as having graphical components (JFrame, JTextBox, JScrollBar, etc.), concepts for arranging components (called Layouts), and event handling (capturing button clicks, keyboard presses, etc. in components). This wraps the Graphics2D functionality of Java, which uses the DirectDraw graphics API of DirectX. These settings can also be changed if necessary (Oracle, 2020b). This is maintained in the standard Java packages, by Oracle. C++ and Python do not have graphics libraries maintained by the owner of the language. C# has the WPF framework, which is similar to Swing, and I considered this, but decided I was more comfortable with Swing overall.

I also looked at efficiency between languages. Most of the runtime in the project I estimate would be spent inside of the indexing algorithms (comparing ingredients and recipes to find viable recipes). Overall, C++ has a reputation for being faster than Java. For example, studies have shown indexing algorithms to be faster on average (Prechelt, 1999). However, optimised Java code will still be faster than the average C++ code. I also believe that the project will not be very intensive for the CPU. This might be different if I were using a more complicated system for recommendations – for examples if I had to recalculate weights this could take a lot of CPU power based on different permutations as the search space increases exponentially (Fournier-Viger, 2017). For the current designs I have, I think Java will be enough to make the program run quickly.

3.4.2 IDE

For the IDE, I compared three different pieces of software: Eclipse, IntelliJ, and NetBeans. I have chosen Eclipse, as it is what I have used during industrial placement and am familiar with. Along with this, Eclipse offers the Perspective feature for switching between debugging and development mode among others, which I have found helpful when testing. Eclipse also offers good support for plugins, and has the plugin WindowsBuilder which provides a useful WYSIWYG editor for GUI. Even though IntelliJ is a more modern IDE, it is not cost free. I could have got a student free edition but I won't be

able to work on this project after university if I would had wanted to make changes. Eclipse is known commercially but is also an open-source IDE.

3.4.3 GUI Framework

I have thought of three approaches to using graphics in Java to create a GUI. These are writing directly with Graphics2D to make components, using the Swing framework, and using the JavaFX framework. Although I have used Graphics2D before, on its own it does not have enough functionality to implement menus without extra work. Because of the extra work that would be required to implement the threading and event handling to replicate a robust menu system, I decided to use a higher-level framework such as Swing or JavaFX. Due to the fact I have used Swing more than JavaFX, I decided to use Swing.

I chose Swing and WindowsBuilder so that I can create basic GUI project quicker without having to learn extensive frameworks like Node.js and Angular.js. Since the project can further be made into a more complex project, I wanted to keep the foundations simple. The focus on this project is more about the algorithms and the features than the design aspect of it.

3.4.4 Database

I created the database in MySQL Workbench because of easy implementation without complicated connections. Also, MySQL in my project allowed me to execute triggers like SELECT, UPDATE, DELETE and it gives a comprehensive application development advantage and is the most known open-source relational database (Kofler, 2004). Since I am building this application in Eclipse, MySQL provides plug-in internal and external libraries like mysql-connector-java and java.sql.DriverManager drivers which will allow me to create a database driven information system (DBQuest, 2012). I could have used other MySQL GUI based software to work with but I personally felt that the workbench satisfies most of the tools I needed and have been comfortable to use for complex big projects because of its high performance and platform flexibility (Letkowski, 2015).

3.4.5 Deployment Platform

I weighed different deployment platforms as options for this project. Desktop, web, and mobile were the main platforms I considered.

Firstly, I decided to deploy on one platform. Whilst I will be showing some of my deployment skills regardless, it is not the focus of this project, and therefore I did not want to spend extra time developing for multiple platforms as some of these might have unique issues and would take away time where I could be working on the features of the program.

I have decided to deploy for Windows desktop, instead of web or mobile. This is because I can easily run the project in-editor to test in the same environment that it would be deployed for. If the project were deployed for mobile, I would have to package it frequently and test it on the mobile platform, which would take extra time. This is also true for deploying for web. Furthermore, deploying for web would require using other languages I am less familiar with such as PHP and Javascript. It is possible I could use Java using something like WebAssembly, but this technology is new, and does not have much documentation.

3.4.6 Deployment Method

I decided to use JPackage for deploying the project as an installer and using InnoSetup to create a script for the installer wizard.

The reason I chose to make an installer instead of a standalone .exe file is so that customers can download new versions of the project and update their installation instead of keeping track of many individual builds of the project themselves. I wrote the Inno Setup scripts in Pascal for Windows.

I could have used programs such as Launch4J to create an installer for the project. I did consider this, as some programs have GUIs which are fast to use for deploying projects, but in my experience these can sometimes fail depending on the libraries that have been included. Because of this, I decided to use a command line incubator module. Since these are developed by Oracle, they are more stable and kept up to date. I decided to use JPackage as it is newer than JavaPackager and is Oracle's currently suggested method of deployment (Oracle, 2020c). JPackage also supports .dmg files for Mac OS, which will help if I ever want to expand the platforms I will deploy for.

3.5 Tools and Techniques

Along with the language, IDE, and deployment method/platform mentioned above, there are several other tools I have decided to use for this project.

For class domain diagrams and other UML that I will be using to plan the project, I will be using Draw.io, which is a free open-source platform to create flowcharts and class diagrams. I will use this to model my system as I will not have to pay for it, and I can run it locally on my system without it being stored on a webserver which could stop being supported mid-development. It saves files locally in the .drawio file extension, which is XML format.

I will be using Figma for creating UI mockups, which is a free online platform to create prototypes and designs for software UI. I will use it because it has features that support modern designs, such as gradients, dropshadows, etc. so that I am not limited in what my mockups will look like. https://www.figma.com/ui-design-tool/

I will be using GitHub for source control which provides version control using Git. It helps to keep track of all the projects with different software versions of the project. Along with this I will also use GitHub Desktop which I am most familiar with. *Appendix 7* showcase GitHub commits of my project. https://github.com/

I will be using the JPackage command line incubator module – It is a tool used for Java applications to convert the jar into and .exe on windows or dmg on macOS with all necessary dependences https://docs.oracle.com/en/java/javase/14/docs/specs/man/jpackage.html .

Inno setup will be used to create installation wizard scripts. It is a compiler used to create installers for windows. It uses the .iss file extension and the script is written in Pascal. https://jrsoftware.org/isinfo.php

3.6 Legal, Social, and Ethical Issues

Legal Issues

If I were to make this commercial, I should be able to have my intellectual property rights which allows me to legally get a full control of my own project which include automatic protection like

copyright and design right (GOV.UK, n.d.-c). I can also apply for other intellectual property rights protection like trademarks, registered designs and patents (GOV.UK, n.d.-b). I can also create non-disclosure agreement in case I want to discuss my idea with someone on business level and don't want them to share this with anyone (Intellectual Property Office, 2015). I should be also careful of the intellectual property rights as a student (UCLan, 2019).

In terms of using external systems like RapidAPI which allows me to access third party services like the TastyAPI (RapidAPI, 2021b). I must be careful to not modify any content on the API provided and have strict instructions on not to license, sublicense, resell or lend the API provided in any way (RapidAPI, 2022). I should also make sure I ask permission of the party before making it viable commercially. I should also be aware I am not using the server without paying when needed.

In terms of using MySQLWorkbench, because it is an open-source (GPL license) software, the database created is viable to use commercially (MySQL, 2022c) (Lischke, 2022)

Since I created the database on my own, I have to make sure to have my copyright protection of my database and database rights as per the Copyright and Rights in Database Regulations 1997 (Intellectual Property Office and Government Digital Service, 2020). This will allow me to protect my database created and won't be used in any illegal manner.

While using this application it is important that the use of the user data like the username, user password, user address, etc., must obey GDPR laws which falls under the Data Protection Act 2018 (GOV.UK, n.d.-a) and Computer Misuse Act 1990 to protect personal data from malicious unauthorised personnel (legislation.gov.uk, n.d.). I also have to make sure the user is okay for the application to use their database to create suggested recipes for them.

I also must be careful not to use any external libraries which might commercially not feasible.

Ethical Issues

I will be responsible if the meal plan generated is not being hazardous and does not include recipes with excess fats, salts, etc., I could use other health fitness related APIs to curb this problem.

I will be responsible if the recommended recipes are not inappropriate and unfit to eat.

It is important to note, in case of multiple users using the application, I also must take the dietary needs like glucose free, gluten free, lactose intolerance, allergies, religious diets, etc., into concern. I could create more user preference which ensures these ethical issues are met.

3.7 Summary

I have chosen a methodology to follow when iteratively planning my project and have started the release plan process.

Java has several benefits such as a suitable framework which obfuscates the low level processes of graphical programming, supported by Oracle, and this will be beneficial for me when developing this project, as well as having good native support for deploying to multiple platforms.

I have also researched the different legal, social, and ethical issues of the project to ensure that I am within my right to deploy this project both as a personal project, and also potentially as a commercial one in future, as I have ownership of my own intellectual property, and am not using any restricted frameworks when developing.

4 Design

4.1 Introduction

It is important to design aspects of the program before starting, so that I can think about what my program needs to do before I spend time implementing features.

In this chapter, I will discuss the designs I have made for the system (the program and its manager classes), the database (the data that will be remembered across sessions), and the user interface (the GUI components, how they are positioned, and how the user will use them).

I have been guided both by user interface principles such as Schneiderman's Golden Rules, as well as system based design ideas such as SOLID principles and Design patterns, and lastly data normalisation for my databases.

I have also used modelling such as the UML Class Domain Diagram for my system, and ERR diagrams to help model my database.

There are several main domains (larger systems) that will be involved in this project, which communicate to each other and the user. These are the Java application, the MySQL database, and the Tasty API.

I am responsible for the Java application, and the database. The Tasty API will be accessed by the Java application, through RapidAPI. RapidAPI is an API Marketplace which provide third party API services to connect and discover APIs. It will also allow me to track usage of all the APIs through their dashboard (RapidAPI, 2021a). More information about RapidAPI will be covered in Implementation section.

The Java application will have several responsibilities relating to interfacing, and logic/algorithms:

Interface with the user

Generate meal plans

Interface with the database for products, ingredients, user data, etc.

Interface with the API to access new recipes

The database will have several responsibilities relating to storing data for users:

Storing user data such as name, password, etc.

Storing fridge data to remember which items a user has and when they will go out of date

Storing recipe and ingredient data for creating meal plans

Finally, Tasty API will be responsible for delivering new recipes to the user. This was not created by me but will be interfaced with using the Java application.

4.2 System Design

A detailed diagram of this system is shown below, which describes how the system should be structured. This covers all the main areas of the program and database, as well as the external API.

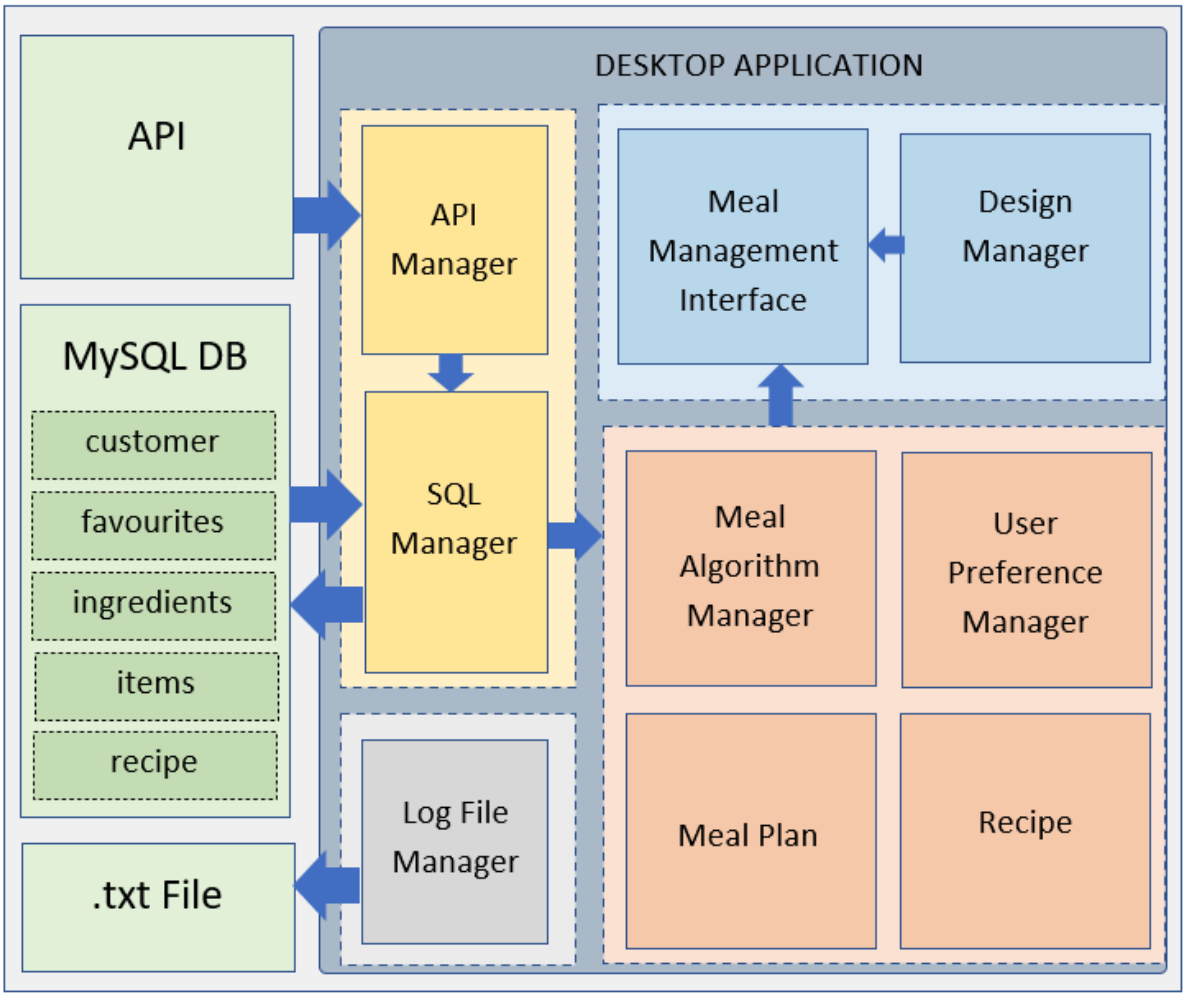


Figure 8: System Design

4.2.1 Java Application

In this section I will write about the different classes I will implement in the program, and their responsibilities within the Java application. This includes which classes interface with other domains such as the API and the MySQL database.

4.2.1.1 Managers

MealManagementInterface

I will have a meal management user interface which will be handled in the main class of the system. This will include the declaration and initialisation of the Swing components. This main class will interface with the user by integrating with the GUI side of the application. The initialise() method in this class will display all the swing GUI components like JButton, JLabel, JTextField, cardLayouts. These components will handle all the action and mouse listeners, event handlers and adapters. The declaration of most of the components will be reverse engineered into code when components are put in the WindowsBuilder which will be used to design the GUI for the system. All the manager classes eventually will meet this java file and will facilitate in the GUI experience.

DesignManager

I will have a design manager class which will be given the responsibility of handling either any components which are not provided by Swing or any helper design methods. The helper methods could include populating data in Swing components like JTable with data from the stored database. In other words, this file will be created to store methods and data relating to the GUI. This file will also include other classes like extends of GUI components which I will discuss in further sections. The design manager will be closely used in the MealManagementInterface file.

SQLManager

An SQLManager class will be designed to designate all the SQL related stuff from connecting the database, running queries to both for reading the data (such as favourites when generating recommendations), as well as inserting (inserting new recipes from the API). This file (a .java class) will be using the maven external library of MySQLConnector to connect the external database. Most of the SQL query methods in this file, will generally have a pattern of iterating through the result set depending on the state of the method. This ResultSets will include things like favourites and recipes, and the class will be able to insert data too. Since this file handles all the data complexity this will be used for all the algorithms from base to complex in this project.

APIManager

The APIManager file and class will be handling the role of communicating to the external API (TastyAPI). This class will have functionality to send HTTP requests and receives responses in JSON format. Later, this manager might have to handle parsing all the JSON data received from the API to send it to the MySQL database through the SQLManager. In summary, this class will be used in conjunction with SQLManager and MealManagementInterface, to store new recipes from the user queries.

MealAlgorithmManager

The MealAlgorithmManager file will be designated to handle the complexity of the meal plan generation algorithm. It will produce meal plan objects which will be used to populate the UI so that the users can plan their meals and find recipe instructions. This manager in future might also have to account for various user details like user favourites, diet restrictions, and ingredient details such as quantity and expiry date. This class will be in close conjunction with SQLManager and MealManagementInterface, to get possible recipes and generate meal plans for the users based on the items they have.

Importantly, the MealAlgorithmManager will be one of the most important classes, and have many indexing algorithms related to producing the meal plan. This includes checking recipes against the ingredients to work out which recipes will be possible, but also dynamically creating a meal plan where these ingredients are taken away so that quantities are taken into account for all the different combinations of meals that could be created.

LogFileManager

The LogFileManager will manage all the exception handlers of this system. This will be used to store

debug logs for users so they could work out if anything went wrong. This information will be dumped in a .txt file. This could also be used later to send to the company if a client has an issue. This class file will be used in all the classes so as to debug logs in the .txt file.

UserPreferenceManager

The user preference manager will store all the user preferences such as max calories, recipe difficulty, etc., This will also be recalled later when generating meals and that is why it would be in conjunction with both MealAlgorithmManager and MealManagementInterface.

Data types

There will also be several smaller classes which are used as data types, for storing information temporarily (such as recipes and ingredients taken from the database or API), as well as perform algorithms (such as meal plan generation which uses all of the following data types):

Recipe - For storing individual recipe data

Ingredient - For storing individual ingredient data

MealPlan - For storing meal plan data, with several days (MealPlanDay), each with different meals

MealPlanDay - An individual day with a Calendar object to keep track of the date (used for expiry when generating), part of MealPlan

4.2.1.2 Swing Components

As mentioned above in the DesignManager section, there will be several GUI components which will be extensions of existing Swing Components. These will be used to create a unique and more modern feel for the application and will have overrides on their paint methods. These classes will be referenced and guided from this source website:

https://www.shredzone.org/maven/jshred/jshred-swing/apidocs/net/shredzone/jshred/swing/JGradientPanel.html

https://localcoder.org/java-swing-rounded-border-for-jtextfield

https://docs.oracle.com/javase/7/docs/api/java/awt/RadialGradientPaint.html

https://jalbum.net/api/se/datadosen/component/JGradientButton.html

https://code-examples.net/en/q/6c9398

This will be for Swing components such as Panels, and Buttons to give them features such as rounded corners, and colour gradients.

4.2.1.3 Class Domain Diagram

Appendix 4 demonstrates the complicated version of the UML Diagram. Here is attached a smaller version of the UML Class Domain Diagram showing the key classes and their relations, as well as several main methods of each.

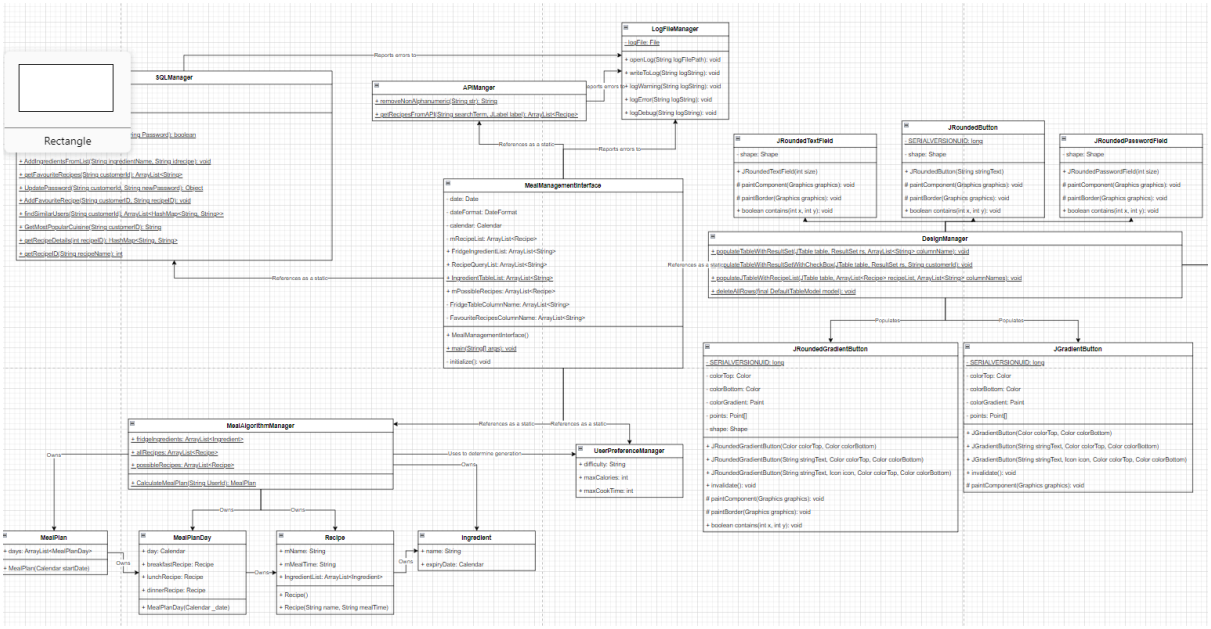


Figure 9: Simple UML Diagram

4.2.2 Coding Style

I have also used several coding conventions to make my code readable. I have tried to be consistent in my naming conventions, such as having camelCase for local variables, _underscoreCase for parameters, and PascalCase for functions. Part of this is making it fit with the "Self-Documenting Code Principle". This includes conventions such as:

- Verb names for functions
- Having variable names that indicate type (eg. "has" or "is" for booleans or boolean functions)
- Using normal english words such as "from" and "with" to explain data flow (eg. populateJTableWithResultSet for using ResultSet in parameters)

4.3 Database Design

I have divided my designing database stages into smaller simpler parts. For this project the database is supposed to be accessed from the actual grocery stores so whenever the customer buys an item from the grocery store it should save those items in their database. But currently because it is a student project and the stores do not have open-source databases, I will have to make this database myself with my own perspective of how the dataset would be. For example, an amazon or Ocado grocery store with details such as product name, calories and sell-by-date.

To create and design my database I decided to use MySQL Workbench 8.0 CE. This is an Oracle's graphical user interface for managing local MySQL Server (Krogh, 2020). SQL Workbench provides a modelling interface and an editor to create models for a database (MySQL, 2022d). This might help me a lot since it will allow me to keep up with my database design from the start to end. These EER diagrams provided can also be reverse engineered if one has started making the tables and columns already and further wanted to build the design approach.

MySQL Workbench will allow me to add, create, update and delete the data in the database without any complicate procedures. This will save a lot of my time and will allow me to start working on my basic prototype right away.

From this, I will be creating an EER diagram model first in MySQL workbench to understand the relationship between the tables and could also later help me formulate the code algorithms around these bases. This model might keep changing at later stages to understand, maintain and handle the data with keeping the base design of the data model.

My second stage will be creating the database as per the model diagram so that the application could start populating the fridge items of each customer in the application. This will make it convenient for me as this will help me start setting up the fridge panel. I will create all the basic SQL tables like the customer table, recipe table, ingredient table, and fridge item table. I will also manually add some data in each of these tables as it will become more convenient to add simple data myself first rather than a complicated API data from the start. The customer and items table will have one to many relationships since each customer will have many fridge items. Relatively, the ingredients and recipe will have one to many relationships as well as each recipe contains many ingredients. The favourite table will have relationship with both customer and recipe tables as each customer could have one or many favourite recipes and there will be many recipes in the favourite table.

This database will play a very important role in the application. It will not only be used to get the customer and customer item data but will also be used to save each user's favourited recipes, recipes and ingredients used in each recipe. All this data would be needed in the application to base algorithms on, finding possible recipes so that the user could create, generate the meal plan and get recommended. This database will act like a third-party involvement between a customer and the grocery stores to help the users formulate a better management in food and costs. Therefore, designing and creating the database correctly is a must have requirement before any basic prototype of the application.

4.4 User Interface Design

4.4.1 Paper Designs

Early in the design process I have been creating sketches with pen and paper to demonstrate my idea of what the application should look like. This will help me to understand what GUI components I will later need to know how they would be arranged on each screen, as well as working out the navigation between pages. These designs will later be used when creating my designs on computer.

This will include a lot of different diagrams for where components could go, and include variations for the same page. I will compare these and decide on which ones are most easy to use. In some cases, this will involve AB testing with others, and seeing which designs have the most positive feedback. I have put my sketches made for this application in *Appendix 8*.

4.4.2 Digital Designs (Figma)

When designing the app digitally, I have used Figma, an online, web-based application on browser which can be used to design the GUI for a desktop application or website. This will be helpful because it comes with many tools for precisely measuring the components so I can recreate the same scale in my real Java application using Swing. Figma also has many tools for implementing visual effects such as gradients and colour schemes, so I will be using this to mock-up my ideas before using them in code. I have also used UIGradients.com to find colour schemes, and have used the hex codes from these to experiment with colour in Figma. In future, the hex codes will be taken

and used in java to define colours in the java.awt.Color class. I have created a Figma design for this application in *Appendix 9*.

4.4.3 Swing Designs

When implementing these designs in Swing, I will be using WindowsBuilder WYSIWYG editor for Eclipse, and follow the proportions I created in Figma to define new components. I will have to use several nested components to define the behaviour to switch between different components. In particular, the "CardLayout" component will likely very important as I could be used this to switch between which component is active at any one time (for example, switching between the main panels such as the recipe details page, the API recipe finder, and the meal plan page, etc.). Using WindowsBuilder will generate new code in the main class of my program, and I will able to further modify this code later when populating components (for example to fill in user and recipe details).

As previously stated in 4.2.1.2 I will also be creating a special class to handle many of these details called DesignManager. This class will be used to store important data such as the colours used in the colour palette, as well as generic functions to populate Swing components. I have also created variations of this to populate the JTable with a checklist, which will be useful when using a checkmark and Booleans for favourite recipes.

4.4.4 Schneiderman's Golden Rules

I have incorporated Schneiderman's Golden Rules as described in the background work section, by evaluating the choices in the design and changing designs to match these to make the application more intuitive and easy to use for users.

This includes allowing users to reverse their actions (allowing users to insert but also delete entries to all the databases they have access to, such as the recipes and fridge ingredients), and also having consistency in design throughout, having a consistent colour scheme, and also consistent use of components and layouts (such as repeated table styles, and a CardLayout that is consistent).

4.4.5 Designing for Disability

I will also focus on designing for disabilities by following important guidelines for those who are less abled, especially relating to sight. This includes having larger font sizes for much of the project as well as having strongly contrasting colour schemes for those with varying eyesight.

I will also incorporate disability simulation to determine which colour schemes are compatible with people who have impaired eyesight, which has been proven as an effective measure for designing user friendly interfaces (Aytac, 2017)

4.5 Summary

I have created designs for several different main areas of the project, including System Design, Database Design, and User Interface Design.

For each of these areas, I tried to be guided by both visual designs (diagrams such as class domain, ERR, and UI mockups), as well as guidelines such as Schneiderman's Golden Rules, and principles such as the Single Responsibility Principle.

I have especially tried to understand the different areas of the program and how they relate to eachother, and hope this will prevent me from having to redesign the system. However, this might still happen with changing requirements, but iterative planning with XP should help this.

5 Implementation

5.1 Introduction

This project was deployed as an executable Java application, along with a MySQL database for storage, with an external requirement of connecting to the Tasty API via HTTP (which the program can function without, but will require manual addition of new recipes).

In this section, I will explain and discuss my methods of implementing the functionality to support my user stories and requirements with classes and algorithms inside of the Java solution. This includes manager classes such as APIManager, and MealPlanAlgorithmManager, as well as data types such as Recipe and MealPlan.

This section will contain multiple code snippets with full versions found in the appendix, along with explanations of how the algorithms work, and where I found relevant material to implementing these.

Finally, I will explain my steps to making a standalone .exe to be deployed on Windows desktop platforms using the JPackage incubator module, as well as an installer with a script written in Pascal using Inno Setup Script.

5.2 Database

5.2.1 What did you work on?

I created an EER diagram model from MySql Workbench which is shown in *Appendix 5*. This appendix shows a completed finished diagram updated with the end of the project as it guides me and saves my time with increasing complexity of the database rather than looking at each entity(table) and remembering their relationships with other entities.

Based on the EER diagram I will create all the tables with their column names and simple data in their rows. I also connected my database to the Eclipse Project using the external libraries. The image below is my code I used to connect the database with Eclipse. I had to use in-built libraries java.sql libraries which are shown in Listing 2. It also includes logging errors caught as exceptions and throwables, and printed to a .txt file via the LogFileManager.

```
// a member function to connect the sql database
public static Connection getConnection() throws Exception {

	try {
		String driver = "com.mysql.cj.jdbc.Driver";
		String url = "jdbc:mysql://localhost:3306/store_db?serverTimezone=UTC";
		String username = "root";
		String password = "root";
		Class.forName(driver);

		Connection conn = DriverManager.getConnection(url, username, password);
		System.out.println("Connected successfully");
		return conn;
	} catch (SQLException e) {
		e.printStackTrace();
		LogFileManager.logError(e.getMessage());
	} catch (Exception ex) {
		System.out.println(ex);
```

```
                LogFileManager.logError(ex.getMessage());
        } catch (Throwable ex) {
                System.out.println(ex);
                LogFileManager.logError(ex.getMessage());
        }
        return null; // couldn't return a connection
}
```

Listing 1 - [SQLManager.java] Connection to SQL Server

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
```

Listing 2 - [SQLManager.java] Libraries imported for SQL server

5.2.2 How does it work

MySQL Connector/J (mysql-connector-java maven dependency) is implemented with the Java Database Connectivity (JDBC) API which provides a way to access data using Java. With the external maven dependency, using the JDBC DriverManager interface, it gives mysql connection to the java application (MySQL, 2022a)(MySQL, 2022b)(Progress, 2022).

With this library I created connections within the SQLManager class, using the Java SE 8 class Connection (Oracle, 2020a).

I also exported an external maven dependency for this to support the inclusion of this java connector to MySQL.

```
<!-- https://mvnrepository.com/artifact/mysql/mysql-connector-java -->
<dependency>
  <groupId>mysql</groupId>
  <artifactId>mysql-connector-java</artifactId>
  <version>8.0.21</version>
</dependency>
```

Figure 10: Maven External Library -sql-connector-java

5.3 API

5.3.1 What did you work on?

The project uses the Tasty API extensively. The API is used via links on the RapidAPI service, to make queries, and find recipe data. This is for the feature of finding new recipes for the user, and putting these inside of the local database.

This feature can be accessed by a user, by entering the Recipe Finder page, where they can enter queries to find related recipes (ie. User can enter “pasta” to receive pasta recipes), which will return

new recipes filtered by both their query as well as their user preferences. These recipes will be added to the local database so their details can be accessed and they can be added to favourites later.

The code below is the connection parameters given by the RapidAPI site for the TastyAPI. It uses a builder factory pattern to get the header parameter. The rapidapi-key will be kept between the host and the user.

```
// Get response from API
HttpRequest request = HttpRequest.newBuilder()

.uri(URI.create("https://tasty.p.rapidapi.com/recipes/list?from=0&size=20&q="+searchTerm))
        .header("x-rapidapi-host", "tasty.p.rapidapi.com")
        .header("x-rapidapi-key", "1fe57c4133msh7efab93e804ef92p1e2bb3jsneafe38f2b6dd")
        .method("GET", HttpRequest.BodyPublishers.noBody()).build();

HttpResponse<String> response = HttpClient.newHttpClient().send(request,
HttpResponse.BodyHandlers.ofString());

System.out.println(response.body());

label.setText(response.body());
```

Listing 3 –[APIManager.java] Connection of API

Listing 3: Connection of API in APIManager.java

5.3.2 How does it work

The API features are accessed mostly through a class called API Manager. I created this to enforce the Single Responsibility Principle (similar to SQLManager for database handling, UserPreferencesManager for settings, etc.).

This class has functions which are used to set up and test a connection with the API, as well as retrieve new recipes with API HTTP requests if this test was successful.

The function getRecipesFromAPI() takes a query as well as a JLabel reference to populate with the results. The query is a string which is added to a request made with the builder pattern, and this request is sent to the API to get new recipes in JSON data format. This JSON data is decoded and used to populate the JLabel referenced.

The decoded JSON data is also passed to the SQLManager class so it can be stored locally for later usage (such as looking at recipe details or adding it to favourites). This used the org.json.JSONArray and org.json.JSONObject classes to have nested decoding of each object to find the correct data, as well as checking for null data to avoid Exceptions that could prevent the decoding from happening. This nested decoding is listed in *Appendix 12*.

I have created a simple diagram in *Appendix 6* to show case the workings of the APIManager.

5.4 Recommender Systems

5.4.1 Similar users function/query – collaborative filtering

5.4.1.1 What did you work on?

I created a simple recommender system to suggest new recipes to users based on the recipes they have already favourited.

The system works in two primary ways. Firstly, it uses content-based filtering, matching similar items to the ones users have already taken a preference to, similar to Netflix recommending action movies to people who like action movies (in this case I used cuisines).

Secondly, the program uses collaborative filtering (user based), which works out similar users based on users who share the same favourited items, and then recommends new items to each based on items they do not share in common as favourites. Similar users are weighted based on which ones share common items, ordered by the count of common items, and the highest value is chosen.

5.4.1.2 How does it work?

The collaborative filtering works with two main functions: FindSimilarUsers() and PopulateSuggestedRecipesInTable().

Firstly, the FindSimilarUsers() function is very important, as it is used to find the similar users whose items to suggest. Primarily this is done with the following SQL request where 01234 is requested customerid:

```
SELECT a.customerId AS current_customer_id, b.customerId AS other_customer_id, COUNT(*) AS
shared_recipe_count
FROM store_db.favourites AS b JOIN store_db.favourites AS a ON a.recipeId = b.recipeId AND
a.customerId <> b.customerId
WHERE a.customerId = 01234
GROUP BY a.customerId, b.customerId
ORDER BY a.customerId, shared_recipe_count DESC;
```

Listing 4 - [SQLManager.java] SQL query to find similar users

This query is used to find all users who share items in common, and for each of these pairings, make a count of how many items are shared in common. A customer ID is passed to the function and added to the string to decide who to compare to.

After this, the first value from the returned result set is used (the one with the highest count), and then the algorithm will find which items this user likes which the current user does not. These are suggested as new recipes in the recommender system.

PopulateSuggestedRecipesInTable is similar to PopulateJTableWithResultSet in that it takes a JTable as a parameter and uses that to put the recipe data into. It also takes a customerID as a string parameter, and uses this to find a customer. It calls the FindSimilarUsers function first, and then uses this to get the user which shares the most in common.

When the most similar user is found, their favourite recipes are retrieved using another SQLManager function, getFavouriteRecipes(). These recipes are then compared with the current user using an indexing algorithm, to find recipes they do not share in common, and after comparing, add them.

```
ArrayList<String> existingFavourites = getFavouriteRecipes(customerId);
for (int i = 0; i < existingFavourites.size(); i++) {
        // Get suggested
```

```
            String existingFavourite = existingFavourites.get(i);
            // Look for existing
            for (int j = 0; j < SuggestedRecipes.size(); j++) {
                        // Get existing
                        String suggestedFavourite = SuggestedRecipes.get(j);

                        // Compare
                        if (existingFavourite.equals(suggestedFavourite)) {
                                    // Remove
                                    SuggestedRecipes.remove(j);
                                    j--;
                        }
            }

}
```

Listing 5 – [SQLMaanger.java] populat suggested recipes in table

Finally, these are added with the results of the item-based recommender system, and the final recommendations are populated into the JTable.

5.4.2 Find similar cuisine recipes – content – based filtering

5.4.2.1 Introduction

I created a simple item/content-based recommender system where it takes in the most popular cuisine of recipes favourited by the user and recommends the user other recipes related to the same cuisine recipes. It is a type of content-based filtering which compares items based on the item data instead of the user data. This type of filtering is often called “tag-based filtering”.

The tags used were retrieved from the API, and stored locally. These tags are then compared locally when recommending new recipes.

5.4.2.2 How it works

This feature, like the collaborative filtering, is run inside of the populateSuggestedRecipesInTable() function, and also uses the getMostPopularCuisine() function to find the most popular cuisine of a user before suggesting new recipes of that tag. Lastly, RecipeQueryCuisine() is used to query recipes of a specific cuisine.

The query used to get the most popular cuisine is the following where 01234 is the required customerid:

```
SELECT p.customerId, p.idFavourites, a.recipeCuisine, COUNT(a.recipeCuisine) AS cusine_count
FROM store_db.favourites p LEFT JOIN store_db.recipe a ON p.recipeId = a.idrecipe
WHERE p.customerID = 01234
GROUP BY a.recipeCuisine
ORDER BY cusine_count DESC;
```

Listing 6 – [SQLManager.java] SQL query to find the most popular cuisine

Then, inside of populateSuggestedRecipesInTable(), the other function, RecipeQueryCuisine() is called, which takes in the cuisine as well as several user preference details to return recipes of the given cuisine that also match the user preferences.

Lastly, these results are combined with the results given by the collaborative filtering so they can be returned as recommended items.

5.5 Find possible recipes algorithm

5.5.1 What did you work on?

It takes in recipes from the user's fridge items and checks if those items can be used in making any of the recipes taken from the api. It basically creates a list of possible recipes each user can create from the items they currently have. This list would be used to create meal plan for the user.

The UpdatePossibleRecipes() function is a function found in the MealPlanAlgorithm class. It is used to update the recipes that could possibly be made at any one time given the ingredients in the fridge (data maintained throughout the algorithm as a member variable of MealPlanAlgorithm), and the recipes the user would like to create.

At the end of each execution of UpdatePossibleRecipes, the SortMealPlan() function is used to re-order the recipes so that the favourites are on the top. This is done so that during the execution of MealPlanAlgorithm's CreateMealPlan() function, only favourite meals are selected first and removed.

5.5.2 How does it work

It used a function call possible recipes which iterates through user fridge items and the recipe list ingredients. It is nested thrice first recipe list inside get each recipe and then further nested to get each recipes ingredients and further nested with the fridge ingredients the user has and check if the fridge ingredient name is equal to the recipe ingredients.

The function UpdatePossibleRecipes() primarily works using an indexing algorithm which iterates through all of the recipes that are known first, and then iterates through all of the ingredients in each of those recipes. Lastly, it iterates through the items a user has in their fridge, and sees if all of the relevant ingredients for a recipe are found in the fridge. If all of the ingredients are available, the recipe is added to the possible recipes list.

Finally, after all of the possible recipes are found, they are sorted based on favourites and non favourites. This is so that when items are subtracted from the list, the favourite ones are taken first, and therefore prioritised over regular recipes. This is done repeatedly when making a meal plan.

5.6 Meal Plan from possible recipes

5.6.1 What did you work on?

I created a function in the MealAlgorithmManager class called CalculateMealPlan(), which takes in the possible recipes the user can created and makes a weekly planner to use the ingredients based on the best before dates, quantity of the item and calorie intake each day.

It uses several other functions inside the class to repeatedly add new recipes to the meal plan and them remove the relevant items from the temporary ingredient list. If the meal plan is successful, it will remove these ingredients from the real fridge database and return the meal plan to the user.

Also, all dates on the meal plan have a "Calendar" variable, so that ingredients will expire throughout the week, and expired ingredients will not be used

5.6.2 How does it work

First, the MealAlgorithmManager initialises its fridge and recipe data, using the functions InitialiseRecipeList(), and InitialiseFridge().

Then, the MealAlgorithmManager generates a new instance of the MealPlan class. This takes a Calendar instance for the constructor, which is used to set the days of the meal plan (this is used for expiration date comparisons and also to show the date when displaying the meal plan to the user.

Then, the algorithm iterates through each of the days inside of the meal plan. When it does this, it updates the fridge, using the Calendar instance inside of the MealPlanDay class (contained inside the body of the MealPlan class) to get rid of expired ingredients. Then, with the updated fridge, it uses UpdatePossibleRecipes() to update the list of possible recipes based on expired ingredients and ingredients used for new meals. Finally, it adds these to the different meal times for that MealPlanDay using the getRecipeAtMealTime() function.

If any of the meal time variables inside of MealPlanDay are null, an error is thrown and the meal plan is not generated.

If no errors are thrown, the meal plan is completed, and returned by the function so it can be used to populate the UI in the app.

5.7 UI Implementation

5.7.1 Swing Components

5.7.1.1 What I worked on

To start up with the project setup I made sure that I had installed the WindowsBuilder plugin which is well integrated with my project in the Eclipse IDE. WindowsBuilder is a plugin composed of SWT and Swing Designer and use WYSIWYG (Eclipse Foundation, n.d.). It creates bidirectional code and is composed of components to create easy GUI components using an IDE GUI instead of writing initialisation code. This plugin uses Eclipse Graphical Editing Framework (GEF) to display and manage the design section. It is bidirectional, meaning that it can read and write any format and reverse engineer most handwritten code thus allows me to make changes in the swing libraries (Rubel et al., 2012). I will talk more about the Design Manager and Graphics2D used in Eclipse.

This also helps to separate the source code section and design section differently and helps me to input designs quicker to build the basic prototype. I started with creating a basic card layout so that I can add multiple tab functionality like fridge panel (to check user's fridge items), settings panel (to edit or save account settings) and so on. This allowed me to expand my project without having the worry about the complexity of design implementation once I would start adding more components in it. The reason I started with this stage of implementation so as the integration between basic design and code is smooth and later it is not too complex to reformat the code around of the design.

After adding the card layout, I started with the basic login functionality feature. As it is not only the least complicated way to start the project, but it also lets me test if I can connect to each user's information from the database.

5.7.1.2 How it works

I translated the measurements of my Figma designs into the project using the properties pane of the Windows Builder editor in Eclipse.

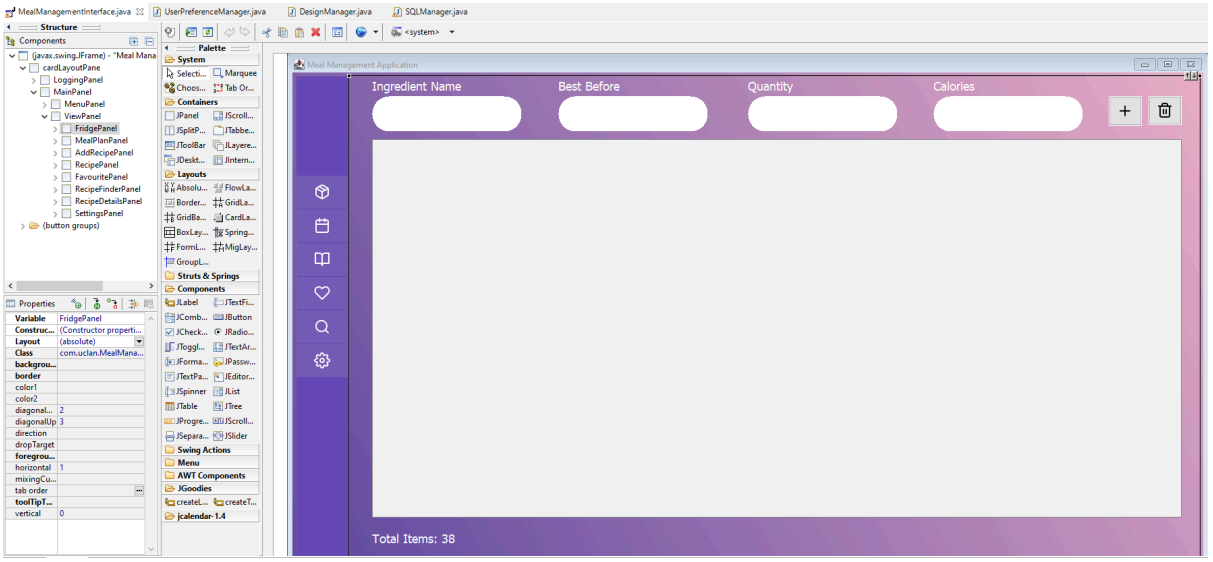


Figure 11: Design view of WindowsBuilder within Eclipse

When a new Swing project is created in Eclipse using Java, code is auto-generated to show the initial components, using the root component JFrame, which is extended by the main application class.

Another benefit of Swing in particular is that it has automatic handling of events which is an improvement on the Graphics2D pipeline it uses. This includes adding EventListeners which can be overrided to then implement unique behaviour. For example, this is the code I have used to pick up a an event to switch to the fridge menu when clicking a button with the mouse:

```
FridgeMenuButton.addMouseListener(new MouseAdapter() {
                                @Override
                                public void mouseClicked(MouseEvent e) {
```

Listing 7 – [MealManagementInterface.java] Mouse Listener

5.7.2 Populating the fridge table

5.7.2.1 What did you work on?

Populating the fridge table was done using the DesignManager class. Because the SQL statements by default return a ResultSet, and most of my methods in SQLManager return ResultSet, I made a general function in the DesignManager called PopulateTableWithResultSet() which can take a ResultSet and iterate through it, adding the information to each row.

I also created a function called PopulateTableWithResultSetWithCheckBox(), which does the same but also adds a Boolean field at the end for SQL ResultSets which have true/false information. This was used when getting the recipe data which could have favourited recipes. I used the checkbox cell to show if the user favourited the recipe or not.

I added more functionality to this page as well, such as the user ticking the checkboxes so they can add or remove rows. The UI also has a section to add new recipes to this list if needed, and they will be inserted into the database for future use.

5.7.2.2 How does it work

PopulateTableWithResultSet() works by iterating through a result set (using ResultSet.Next()), and adding each of the objects as a new row. It first gets the column names by getting the metadata of the ResultSet, and then creates a Vector<Vector<Object>> to make an abstract list of lists of objects. This is because the data stored in a resultset can vary (eg. They may store integers or Booleans or strings).

This data format using vectors is needed, because after being populated with a while loop using ResultSet.next(), it is used to call the constructor for the table model DefaultTableModel(Vector<Vector<Object>>, Vector<String>). The second vector (of strings) is given the column names retrieved from the metadata.

Finally, the JTable is given the new table model, and the design is also changed using new colours that conform to the style of the project.

populateTableWithResultSetWithCheckBox() is a variation on this function which also creates a custom column for Booleans. This is used when making favourites. It works similar, but adds a final Vector<Boolean> to the columns, to create a list of Booleans which is used for the favourites when displaying recipes in a table. It also overrides the getColumnClass() function, so that it returns a Boolean class type for the last column.

PopulateJTableWithRecipeList() is a function specifically for this task which gets the names of the columns and takes an arraylist and is used for the API manager.

5.7.3 Recipe Details

5.7.3.1 What did you work on?

I created a recipe detail panel which takes in a specific recipe from the database and provides more detail on the recipe. This is so that when looking at the recipes available to a user, the user can double-click one on the table, and go to a page specifically for it, so they can read the details (ingredients, instructions, etc.) and easily follow them to make the meal. The page has several details at the top, such as recipe name, description, nutrition details, serving times, calorie intake, etc.

Underneath this, important preparation notes such as ingredients needed and instructions are listed. The GUI is made using JSplitPane and JTextArea to break up the different areas of text.

5.7.3.2 How does it work

The double clicking of rows in the JTable inside of the recipe panel will open the details page. This is done by overriding the mouseClicked() event for the JTable used to display the meal plan. It takes the MouseEvent object, and checks details such as the click count and the button clicked. On the event that it is the left button, and it is clicked twice, the CardLayout component will switch the shown component

If double clicked on each recipe the recipe displays another panel which shows the details of that recipe. It uses JSplitPane to show case each of details, like description and instructions. It also uses text area to put the information in each splitPane. It gets the information through the database by using the getRecipeDetails() function from the SQLManager class taking the recipeID as the parameter.

5.8 Favourite System

5.8.1 Introduction

The favourite system is used so that the database has memory of which users like which recipes. It is used to users can revisit their favourite recipes, the meal plan can be generated so that it has a preference to include favourite recipes, and also to recommend new recipes via the recommender system.

Users can see their favourited recipes inside of the recipe menu. They have a marked checkbox to show which ones are favourites. The "favourites" page will also exclusively show only the favourites to the user so they can see which recipes they liked in the past.

The same screen also has recommendations listed at the bottom, based on both content-based recommendation as well as user-based recommendations (collaborative filtering).

The meal plan will choose randomly from the recipes that can be made, but it will have a preference for favourited recipes and suggest those randomly if they are available.

5.8.2 How it works

The function populateTableWithResultSetWithCheckBox() generates a unique table model for displaying the recipes in the recipe panel. This model adds an extra column on the right side of the JTable, with a boolean type. This is done by adding a Vector<Boolean> into the data, as well as overriding the getColumnClass() function to return a boolean for the last column.

The meal plan has a preference for favourites, and this is executed by using the sortMealPlan() function inside of the MealAlgorithmManager class. The code iterates through the recipes which are possible, and then checks each if they are a favourite inside of the database. Based on this, each result is put into one of two new lists, favourites and non-favourites. The non-favourites is prioritised when making the meal plan.

```
// Get favourite recipe from db
ResultSet rsFavouriteRecipeQuery = SQLManager.getFavouriteRecipesResultSet(customerId);

// Put favourite recipes from db into favourite arraylist
ArrayList<String> favouriteRecipeList = new ArrayList<String>();
while (rsFavouriteRecipeQuery.next()) {
        favouriteRecipeList.add(rsFavouriteRecipeQuery.getString(1));
}

// Create new meal plan for sorted list
ArrayList<Recipe> newMealPlanList = new ArrayList<Recipe>();

// Iterate through the meal list // add favourites first
for (int i = 0; i < mealPlan.size(); i++) {
        // Set variables
        Recipe recipe = mealPlan.get(i);
        boolean recipeIsFavourite = false;

        // Check if recipe is favourite
        for (int j = 0; j < favouriteRecipeList.size(); j++) {
                if (mealPlan.get(i).mName.equals(favouriteRecipeList.get(j))) {
                        recipeIsFavourite = true;
```

```
                }
        }

        // Add if favourite
        if (recipeIsFavourite) {
                possibleFavouriteRecipes.add(recipe);
        } else {
                newMealPlanList.add(recipe);
        }
}
```

Listing 8 - [MealAlgorithmManager.java] sort meal plan

The favourites themselves are inside of the database, and have three values: favouriteID (the primary key), userID, and recipeID. The system is very simple and is used to connect the two databases: Users, and Recipes. These entries are added and removed whenever the user favourites or unfavourites an item. This is done using SQLManager.

5.9 User Preference Manager

5.9.1 Introduction

I have created a panel which is used for customisation of the user preferences and settings. It is made to input details such as the user password and dietary information such as daily calorie intake, diet, and also preferences such as desired recipe difficulty.

The settings the user chooses can be saved with a button press, and then these are used to filter the things they see, such as only giving easy, low-calorie meals if the user has specified this.

5.9.2 How it works

There is an enum used for difficulty, which is saved to the UserPreferences class when the user selects a corresponding RadioButton.

The UserPreferences class also has static integers which are used to remember the maximum value for the calorie intake per meal for users. When meal plans are generated, the recipes collected will be based on this detail, as well as the difficulty filter being used.

5.10 Deployment

5.10.1 JPackage

I used the JPackage incubator module command created by Oracle to create a native package for the Windows operating system in the form of an .exe. JPackage is used for these native packages as well as installers, and can also deploy to other systems such as Mac OS in the form of a .dmg file.

JPackage is the successor to Java Packager and was taken into production as early as the Java 16 version of the JDK (Java Development Kit) (Sadogursky, 2021). I first made sure that I had the correct version of the JDK installed on my machine before using it, as well as checking that my environment variables were set up for Java.

After this, I used the command line argument and designated several commands, such as the original Runnable JAR path, the root class path, and also the path of where the .exe file should be created.

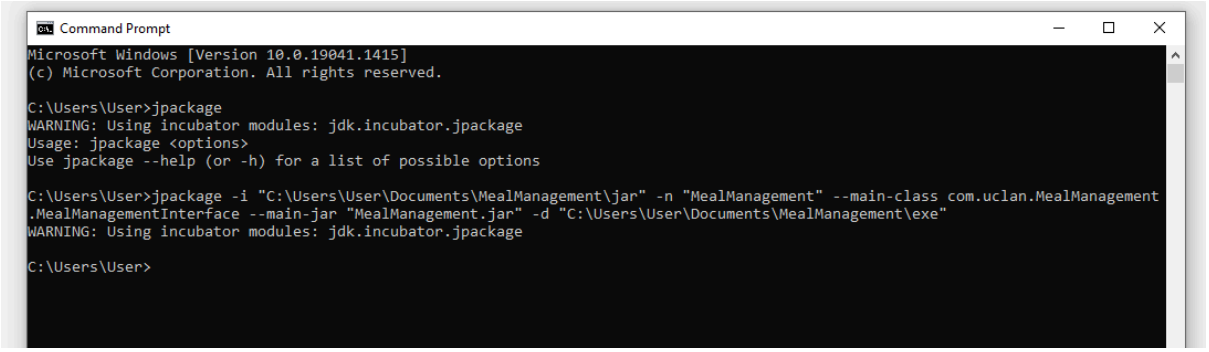


Figure 12: JPackage Command

5.10.2 Inno Setup Script

After creating the standalone .exe, I wanted to make an installer for the project, and used Inno Setup Script for this. I created a script for making the installer using Pascal and ran this script with the correct pathing to create an installer. I did this because it creates an installation wizard for Windows when installing and can also allow for more options added to the installation wizard later on if needed. *Appendix 10* shows the Inno Setup Script written for this project.

5.11 Summary

I have created a runnable Java program deployed as an installer on Windows desktop, with a dependency on a local MySQL server for user and recipe/ingredient data, as well as an external API which I use to run queries using HTTP requests.

This project has included a multitude of skills in my programming, including understanding of recommender systems, and indexing algorithms to calculate many permutations of meal plan to find one which is suitable for the user, and adheres to their personal user preferences, as well as their preferred food types based on collaborative filtering of multiple users.

This project could also serve as a base for further work if I wish to pursue it further, as I could integrate it more directly with the shopping process if I find a relevant API for retailing companies to create a full experience of shopping for food and generating a meal plan directly.

I have also had much experience with developing a user interface for the user, with a variety of skills shown to control the components of Java's Swing library to display relevant information and take user input to change the data of the project.

6 Testing

6.1 Introduction

I have employed a variety of both functional and non-functional testing for this project, with test cases being used to show functionality, and AB and ableness testing to show that the program is suitable for all audiences visually.

This section will discuss first the functional testing and how I have created test cases for multiple systems, such as SQLManager accessing the database, APIManager creating HTTP requests for the API, GUI elements registering user events, and algorithms successfully handling data to produce a meal plan, among other systems.

I have also performed non-functional user testing, to both develop a colour scheme and component layout that users will appreciate using AB testing, as well as usability testing for different amounts of ableness to ensure that the program is suitable for less abled users. This includes testing the colour scheme for varying amounts of sightedness for people with monochromacy, dichromacy, or anomalous trichromacy.

6.2 Error Handling

I created a class named LogFileManager to handle errors encountered during runtime of the app when it is compiled as a standalone .exe or runnable .JAR file. This is so that users of the app can track the issues they encounter without having to debug the program formally using an IDE and breakpoints, etc.

Exceptions are caught using try and catch blocks (and sometimes using throw Exception for functions), and inside of the catch blocks, I have called the static instance of LogFileManager to log error information in a text file. The following is an example of an output to the .txt file for the meal plan manager when an error has been encountered:

```
[2022/03/22 12:14:44] Meal Management started.
[2022/03/22 12:22:11] ERROR: Data truncation: Data too long for column 'recipeName' at row 1(INSERT INTO store_db.recipe (recipeName, mealTime, recipeDescription, recipeTime, dietCategory, recipeCalories, recipeDifficulty, recipeServings, recipeInstructions, recipeCuisine) VALUES ('Cafemaddy Shows You 3 Creative Ways Of Using Kimchi', '', '', '0', 'none', '0', '', '0', '', '') )
```

Listing 9 - [LogFileManager.java] output in .txt file

This has been quite a good process as Java has a lot of support for Exception handling. I also take the stack trace information and output to the editor so it can be debugged in the IDE as well.

Relevant Exception names are used for situations like SQLException in the SQLManager, or HTTPResponse exceptions for the API. I also catch general exceptions and use the relevant name in the debug log.

When exceptions are thrown, Java provides an object to capture the information in this Exception, which inherits from the Throwable class. I use Throwable.printStackTrace(), which prints the information and call stack of the Exception object into the console when using an IDE, which is useful for debugging as a developer (Oracle, 2020d)

For customers, I use the Throwable.getMessage() method and pass this to my LogFileManager to print these errors in the logfile. The getMessage() method returns a string, which is taken by the static instance of LogFileManager inside of the writeToLog() method (Oracle, 2020d).

6.3 Functional Testing

6.3.1 Database Test

The tables in the database and their relationships with each other in the MySQL Workbench can be tested by adding raw queries in File -> New Query Tab. Below is an image of creating a one – many relationship-based query between the customer table and the items table used in my application. This query gets each customer's fridge items and can be tested if the data displayed is relevant with the information we need.

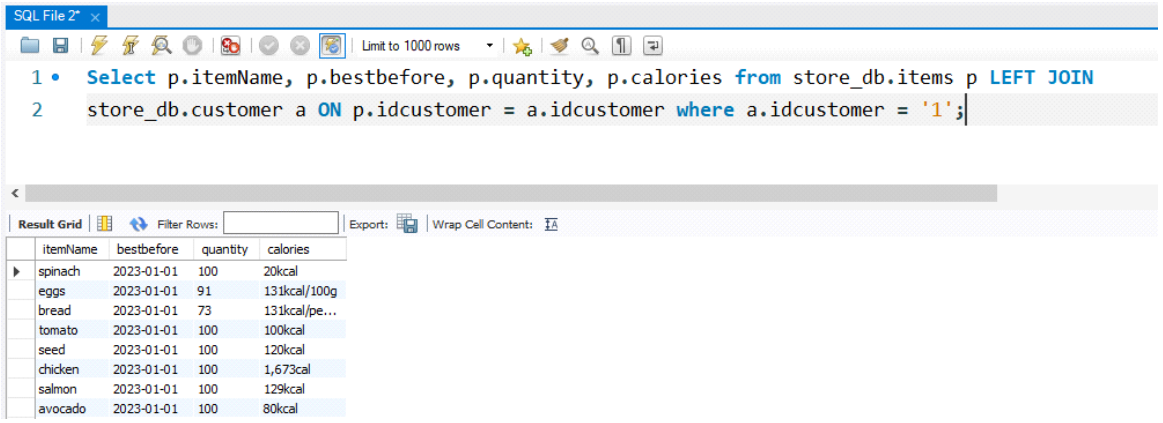


Figure 13: MySQL WorkBench Query Tab

It also sends console output to check if the query is executed, if not then it displays relevant SQL syntax exceptions or error codes.

Figure 14: Check query has executed

The below image throws an error code when table name is inputted incorrectly.

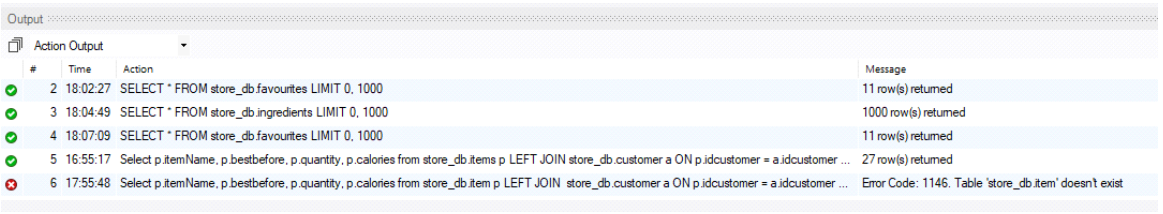


Figure 15: Output to check query errors

6.3.2 API Test

I tested the API endpoints using the RapidAPI website which both documents the API endpoints available for a website, shows example responses for given code snippets in different languages, and also runs these queries live to show the user how the responses will look for their queries.

Once testing each endpoint using RapidAPI, I ran the same queries inside of my program using Java's HTTP responses to check that the response was consistent.

The following is a test of the Tasty endpoints to guarantee that the endpoints are working before further verifying it with test cases using the Java API connection code using HTTP requests:

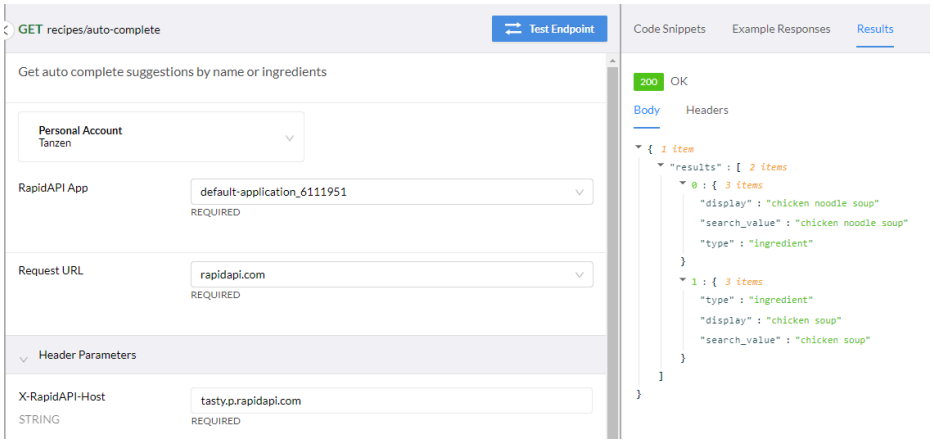


Figure 16: Testing endpoints of the TastyAPI using the RapidAPI site

The following is a dashboard checking the amount and frequency of API calls to make sure the calls do not go over the limit:

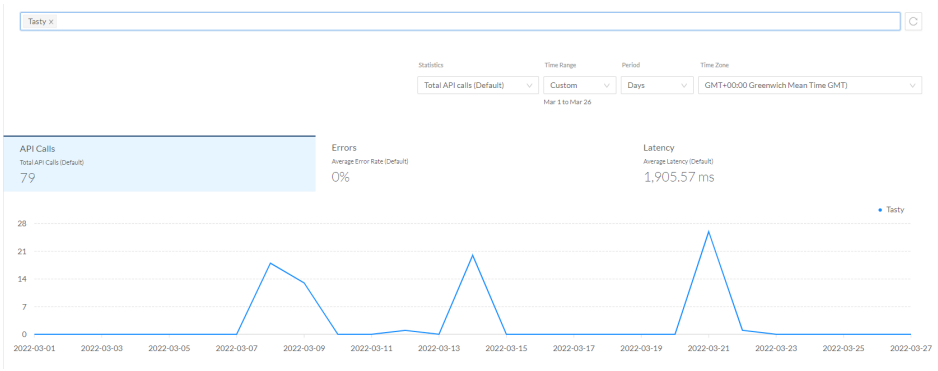


Figure 17: Graph of API calls the application made

I have also included an example of an API response tested on RapidAPI, with code snippet and a partial section of the response included in *Appendix 13*.

6.3.3 User Interface Testing

I included the compilation of each Swing component to be displayed in my test cases, to ensure that all Swing components were visible and displaying correctly to the user. This also included checking the population of UI elements and was recorded in my test cases, such as the following:

Table 1: Test Case for populating the query in a JTable

Populating the query in a JTable	1	Start app from .exe	PASS	
	2	Navigate to the fridge page	PASS	
	3	Check the fridge table	PASS	Populated the query in a table
	4	Check for any exceptions in the log .txt file	PASS	No exceptions output

6.3.4 Recommender Test

I tested the recommender system by including test cases which checked if similar users would be included when running a similarity test. I also tested if when similar users were present, the program would recommend recipes which were new, but only from users which shared some recipes already.

Table 2: Test Case for similarity score in database

Favourite	Similarity score in database	1	Create main user with individual favourite set	PASS	
		2	Create users 1 and 2 with similar items	PASS	
		3	Check favourite recipes of main user from the database	PASS	
		4	Check favourite recipes of user 2 from the database	PASS	
		5	Check favourite recipes of user 3 from the database	PASS	
		6	Run the similar user count raw query in the database	PASS	
		7	Check the similarity score	PASS	Successful similarity score of 2 given

The SQL function used to find similar users would return a count (which is used as a similarity score) and I could check in MySQL Workbench to see that the user sorted at the top was the one being used to suggest new recipes.

After checking in MySQL WorkBench it was important to see that this was also displayed in the GUI.

More test cases can be found in *Appendix 14*.

6.3.5 Deployment Test

Most deployment tests were run simply by testing if the deployed .exe was working, and would re-run test cases using the .exe built by jpackage in the same way that I did when using the Eclipse IDE. I also checked the manifest meta-inf files to see that the correct main JAR class file was referenced in the Runnable JAR before using jpackage.

After this, I would test using Inno Setup Script to see that no errors were thrown when creating an installer. Finally, I would run the installer and fully install a fresh version of the meal management app and then run further test cases as before.

I also could check the LogFileManager .txt logs with each of these deployed versions to check if any new errors occurred. This can be important because unique Throwables may be encountered only after installing if the JRE version is wrong. This is why Throwable exceptions were used in the SQLManager.

6.4 Non-Functional Testing

6.4.1 Disability Testing

To test the proportions of the project, I will compare the component sizes against the Figma designs. This will ensured that the original design specifications are met, including those that are important

for accessibility such as large proportions and colour schemes which suit people who are colour blind.

I also used the Coblis (**C**olor **BLI**ndness **S**imulator) colour blindness simulation tool found on Color-Blindness.com (Wickline & Human-Computer Interaction Resource Network, 2001)to detect if the application has non-visible elements for people with varying versions of anomalous trichromacy (cone weakness), dichromacy (cone blindness), and monochromacy (total colour blindness).

For example, here is a simulated image of Colour Scheme B (the chosen colour scheme) with Blue-Blind Tritanopia

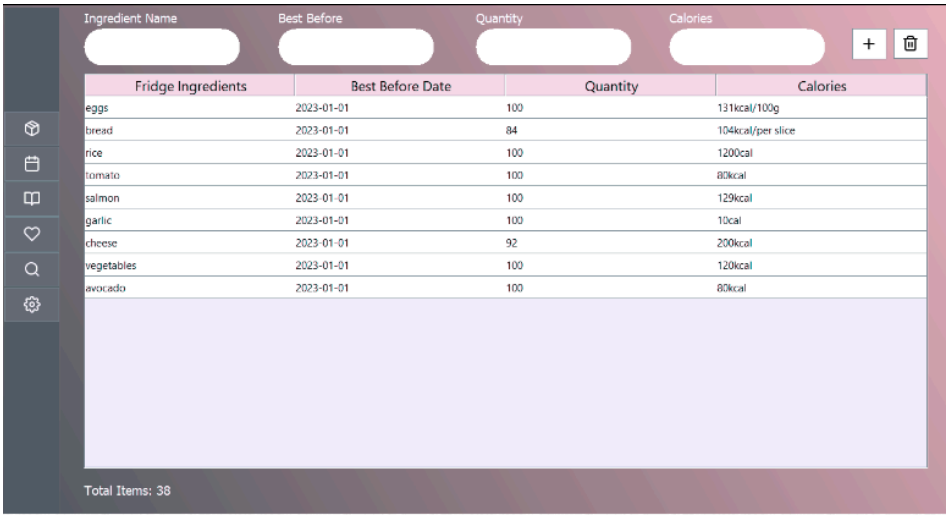


Figure 18: Colour Scheme B with Blue-Blind Tritanopia

Further examples of colour blindness have been included in *Appendix 11*.

6.4.2 AB Testing

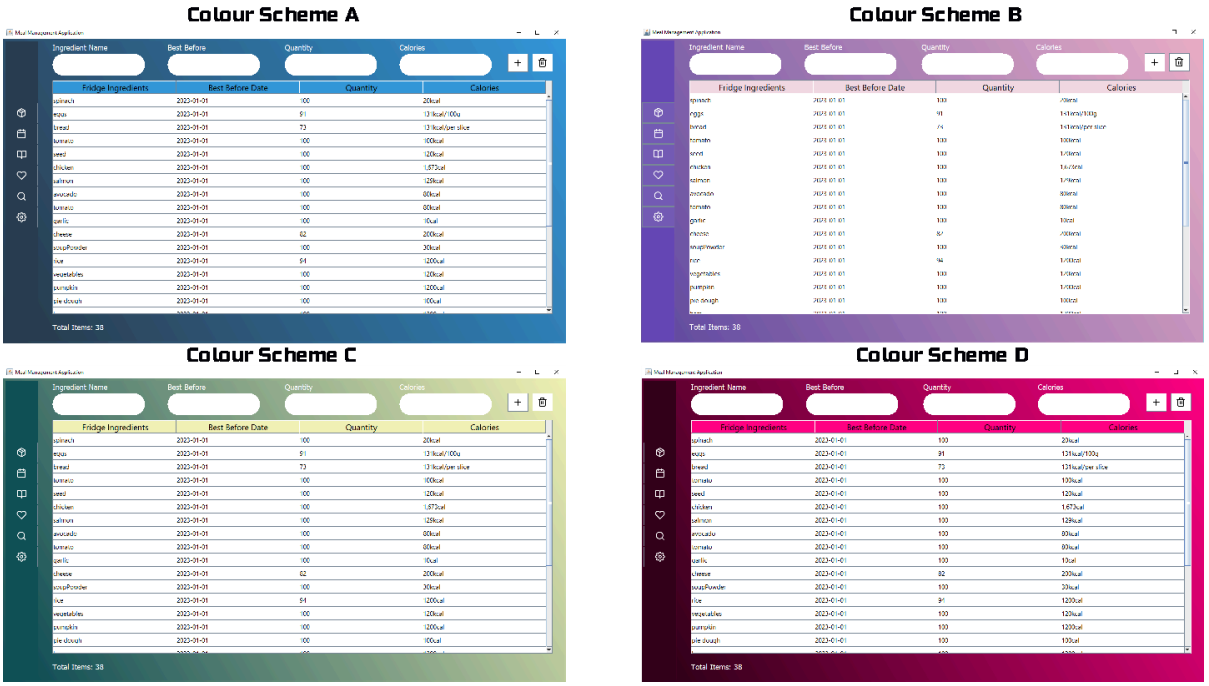


Figure 19: AB Testing with different colour schemes

After performing AB testing on the above colour schemes, the votes revealed that Colour Scheme B was the most popular with 43.5% of the votes, but Colour Scheme A was close in second place with 30.4% of the votes.

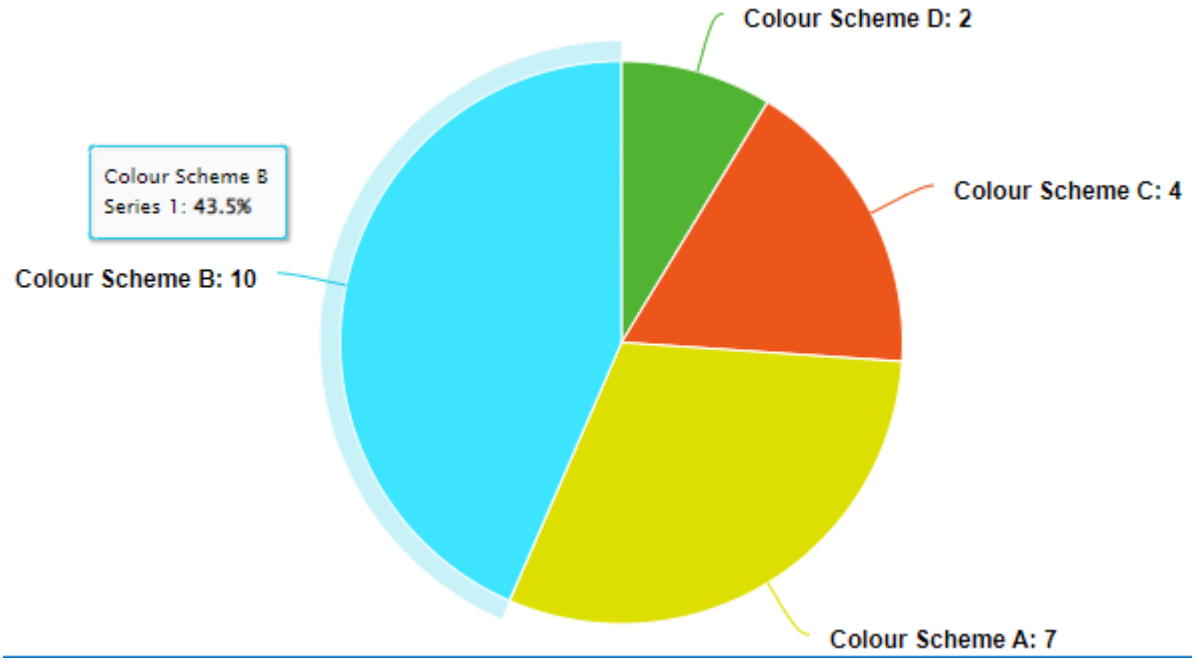


Figure 20: Pie Chart score from the four colour schemes tested using the AB testing

Because of the close score given to Colour Scheme A, it could potentially be considered as an alternate colour scheme, and this is a consideration for future. However, this would require more disability testing to see that the alternate scheme is suitable for all users.

6.5 Summary

My testing has been successful in proving that the most important systems of the program are functional and not error-prone, as well as revealing any forms of user input that could cause error, which can be dealt with.

The testing has also been able to reveal the colour schemes and component layouts which are most liked by users through AB testing, and these layouts and colour schemes have been chosen as a result, making the app more aesthetically pleasing.

Potentially, more AB testing could be used to reveal more specific preferences if I can find more factors to change, such as different fonts, and effects such as drop-shadows. This could make the app more modern. I would also like to experiment with more features of “flat-design” UI.

Lastly, the testing has proven that the colour scheme chosen is suitable for different abledness in users. This could be further improved by checking for other types of abledness, such as shortsightedness, but using different font sizes. Potentially, these features could be adjusted by user preferences.

7 Evaluation, Conclusions and Future Work

7.1 Project Objectives

Deployed a meal management application which produces a meal plan using a user's ingredients and recipes taken from an API.

Includes:

- Meal plan generation algorithm
- User customisation
- Content based recommendations
- Collaborative filtering recommender system
- Dynamic API usage for custom queries and storage within local database

7.2 Self-Evaluation

The project has been challenging and has pushed me a lot. Though I did not finish each task as per the Gantt chart I made, I did manage to make time at the end to finish all my must and should haves which I feel a bit proud of. I also proceeded to do some things I did not expect, such as my recommender system.

From my experience during my placement year, I have been careful of not being too ambitious and add too many features which would have been difficult for me to push them in the constraint time limit. I planned features I believed I would be capable of and made sure they worked well.

At first, I was a bit overwhelmed with the number of features I wanted to add in the project. But while writing my technical plan and I compressed and made sure I get one thing at a time and have the foundation base for this project all ready.

At the end I still felt like I could have added simple things like getting a text file of a generate meal plan so that the customer and later view or reformat the code and divide my large chunks of code into more class files. But I am overall satisfied with how the project ended within the constraint time.

I felt, over the university years I was able to use all my skills from software development to databases to writing advance algorithm and code. I learned a lot more in my third year and I wished I knew those skills before starting the project like multithreading, design and architectural patterns and creating web services so that I could have implemented those concepts in my project. Even though I have used builder factories and some other patterns through either java handling them like that. I did not know a lot about them to get practice with.

In terms of time management, even though at start it was challenging I am really glad that my supervisor kept meetings after every two weeks' time, which encouraged me to get features done one by one and follow his feedback thoroughly. It was helpful for code reviews and constant feedbacks while still working on the project.

This project is just the beginning of its foundation and I want to keep working on this and make the could and would haves in the MoSCoW Analysis into must haves. If I keep dedicating enough time in this project as I thought, in the near future it should take me another one year to get a professional application.

I have much better confidence in myself than the last years and feel positive about completing this project in the near future.

7.3 Project Evaluation

As an overall system I think the project has demonstrated all the technicality I have studied till now. It has also completed the key core objectives the project needed. Although, there could be more changes made in future. Since this project has a lot of scope to add more features and make it more complex with what I learned from this year. Below are some notes I think I could do in the future.

7.3.1 UI design

This stage of the project was successful, but it is possible I could have used a different graphics framework if I had more time. I have not used JavaFX, but there are more resources for it regarding how to make modern interfaces.

7.3.2 API

In terms of the API, I think it worked well. It was a shame the SerpAPI was expensive, and I had to change to another API. It took some research time from me. If I could have gotten more time I think it would be better long term to create my own web API and that way I won't have to use the local-database. For this to be possible, I could learn about web API frameworks in Java and make use of frameworks such as spring or springboot. In future, instead of saving the recipes in the local database and increase the database size and having to overwrite the API data in a database. I could also add the data in a JSON formatted/csv file and used it as a cache data.

With more time I could integrate with other APIs. For example, access to an API which has databases for products inside of grocery retailers could mean that the items a customer bought could go directly into the meal plan manager. This would be similar to a case where a developer used the Dunzo API to create an app which could find half loaves of bread in nearby grocery retailers (Howell, 2022). Similarly, being able to connect to other APIs could also work, such as Amazon's API to get users' deliveries (Lord, n.d.).

7.3.3 Recommender sys

The project could have used more tags for the recommendation system. Currently, the recommendations and filtering focus on cuisine and difficulty of recipe, but this could extend to more tags such as "homely foods" or "party food", etc.

With more time, I could also store weights for item sets, so the recommendations could be more fine-tuned, and remembered for future. However, this could also be quite intensive for the computer as weights need to be recalculated often. This could be used both for finding similar users and also similar items.

I could have used other types of recommender system as well, such as rating-based recommendations, which would simply take the most favourited (or highest average rated in a star system) and recommend these items to a user.

If there were more types of recommender system used, I could try to blend these together for a final weight, instead of using all types of recommenders separately.

7.3.4 Meal Plan from possible recipes

Even though the meal plan generator works really well in the project, it could make further improvements like giving an option to change recipes in the meal plan and show other suggested recipes around the same ingredients. For example, if the user doesn't want to have a French toast in the morning as generated by the meal plan, the user can change the option in the meal plan and instead go for some other recipe like egg sandwich which takes up almost same time and have ingredients. This would give much more flexibility to the user.

7.3.5 Error handling

My methods of logging errors has worked well so far, although I think I could potentially split my logging into multiple sections in future, and use different files, for different levels of expertise. This way, a general log file could be used by customers to understand what went wrong, but more complicated stack traces could be output to a larger logfile to be sent to a developer if needed.

The system works well , and I could have added more exception catching any uncaught or runtime exceptions, but since my program is not using any multithreading for now, I don't want to make the code look very long, as I have already put many exceptions and throwables in the code.

7.3.6 Code base and UI

Looking at the code in general I do feel the need to reformat the code more and divide a lot of my large classes into smaller simple files. I also feel I might make the code more consistent with methods. For example, some populating table methods are divided between DesignManager and SQLManager class. I could have also tried to moved the colour code of the JTable population somewhere else to enforce single responsibility pattern. And divided more of the algorithmic side of the recommender system into a different new recommender class. I think it would have been better to add a search bar for the recipe tab since it would be hard to navigate every single recipe. In terms of UI, I think I could have used more icons. For example, the checkbox used inside of the recipe table to show which recipes are favourites could be replaced with a texture for a heart, with a filled and non-filled version to show favourite and non-favourite, as this is common with most similar applications.

7.3.7 User preferences

I think the User Preference Manager worked well, but I feel if I had more time I would have added more features like cuisine selection, diet options like veg/non-veg or religious options like a Hindu meal plan, or kosher (which will be strictly on which days we could eat meat or have a tick on the days we might need to fast on occasions such as Ekadasi) or any other religion meal plan. It could have also added recipes food based on festivals each user celebrates. It could have many more customised meal features which I could have added if I got time.

7.3.8 Single Responsibility Principle

I think I could have enforced single responsibility more strongly in my classes. For example, I kept colour data types inside of the MealManagementInterface class, where I could have stored them inside of the DesignManager. I also had some UI population inside of my SQLManager, where queries directly populated some JTables. Instead, I think I should have made the SQLManager consistently return general data types such as ResultSet, or ArrayList<String> and then used these separately to populate JTables using the DesignManager. Lastly, I could have made a RecommenderSystem class which handles the logic of creating recommendations based on tags and similar users. It could also have functions for settings, so I could try balance different importance of content based and user based filtering.

7.3.9 Storage in Database

In terms of storage, I think I could have utilised the database more. Meal plans could have been stored in the database, attaching ingredients to individual meals and meals to users, and these could be removed or archived as used when the week passes. This could be done so the user can close the application and re-open it to find their current meal plan. I also could have similarly stored the user preferences inside of the database.

7.4 Applicability of Findings to the Commercial World

This project is fit for commercial use as I have specifically designed it to be continually and iteratively improved on. The classes are organised to be modular and have future improvements. I have also deployed it in such a way that it can be continuously delivered to customers using installers without customers having to manage multiple versions. As well as this, the error log functionality provided by the log manager and exception handling should allow customers to relay problems back to the developer so that improvements can be made.

I believe the algorithms and data structure handling in this project could form the base of many other projects if needed. For example, the recommender system could work for a shopping website, or a streaming service if needed. The indexing algorithms could also be used for other projects such as a household DIY app to work out which tools are needed to perform which services in the house, and provide instructions similar to how recipes are given for this project. This could also extend to making computers or mechanical projects. I believe generally the recommender systems and indexing algorithms have general use for planning aids for customers.

7.5 Conclusions

I have successfully planned, developed, and deployed a meal management application to help users manage the products they buy and generate custom-made meal plans that account for their preferences and dietary needs.

My solution has web connectivity and delivers new recipes taken from live data via a public API, and stores these locally, keeping track of which users own which items, and suggesting new recipes using a custom recommender system, using both collaborative and content-based filtering.

This solution is delivered with a custom made modern flat UI interface and has been tested to conform to popular UI practices as well as be suitable for less abled audiences using colour blind simulations for the palettes and component design.

The final project is deployed as a custom installer programmed in Pascal using Inno Setup Script, which can create fresh installations using a wizard for the Windows desktop OS, and initially packaged from a Runnable JAR using the jpackage incubator module.

7.6 Future Work

For this project to continue as a commercial product, it would need the help of a grocery retailer to provide data and integration so that when customers purchase products or order deliveries, these orders can automate with the meal manager. This data could also help store product information using barcodes, so that individual products could be registered to the system using photos, possibly taken on the app.

As written in the evaluation section, more types of recommender system could be used, and these could be blended together using a weighting system.

For the API, two changes noted were that there could be more APIs used, such as Dunzo for finding products in grocery retailers, and that the API could be hosted by me personally (although this would take a lot of time writing data).

8 References

Amazon Web Services. (2021). *Creating a private API in Amazon API Gateway*. Amazon Web Services, Inc. https://docs.aws.amazon.com/apigateway/latest/developerguide/apigateway-private-apis.html

Aytac, S. (2017). Using Color Blindness Simulator During User Interface Development for Accelerator Control Room Applications; Using Color Blindness Simulator During User Interface Development for Accelerator Control Room Applications. *Accelerator and Large Experimental Control Systems*. https://doi.org/10.18429/JACoW-ICALEPCS2017-THSH103

Baeldung. (n.d.). *Exploring the New HTTP Client in Java | Baeldung*. Retrieved November 24, 2021, from https://www.baeldung.com/java-9-http-client

Baltrunas, L., & Ricci, F. (2009). Item Weighting Techniques for Collaborative Filtering. *Studies in Computational Intelligence*, *220*, 109–126. https://doi.org/10.1007/978-3-642-01891-6_7

Beck, K., & Andreas, C. (2005). *Extreme Programming Explained , Second Edition*. Addison Wesley.

Boyd, M. (2014, February 21). *Private, Partner or Public: Which API Strategy Is Best For Business?* . ProgrammableWeb. https://www.programmableweb.com/news/private-partner-or-public-which-api-strategy-best-business/2014/02/21

Bryson, J. M., & Bromiley, P. (1993). Critical factors affecting the planning and implementation of major projects. *Strategic Management Journal*, *14*(5), 319–337. https://doi.org/10.1002/SMJ.4250140502

Cloud Endpoints. (n.d.). *Why and when to use API keys | Cloud Endpoints with OpenAPI*. Retrieved November 24, 2021, from https://cloud.google.com/endpoints/docs/openapi/when-why-api-key

Coursaris, C. K., Hassanein, K., Head, M. M., & Bontis, N. (2012). The impact of distractions on the usability and intention to use mobile devices for wireless data services. *Computers in Human Behavior*, *28*(4), 1439–1449. https://doi.org/10.1016/J.CHB.2012.03.006

DBQuest. (2012). *Top Reasons to Use MySQL*. DBQuest, Inc. http://www.databasequest.com/index.php/product-service/mysql-dbquest

de Souza, C. R. B., & Redmiles, D. F. (2009). On The Roles of APIs in the Coordination of Collaborative Software Development. *Computer Supported Cooperative Work (CSCW) 2009 18:5*, *18*(5), 445–475. https://doi.org/10.1007/S10606-009-9101-3

Digité. (n.d.). *What Is Extreme Programming (XP)? - Values, Principles, And Practices*. Digité. Retrieved May 3, 2022, from https://www.digite.com/agile/extreme-programming-xp/

Eckhardt, A. (n.d.). *Various aspects of user preference learning and recommender systems*. Retrieved November 24, 2021, from https://www.researchgate.net/publication/220827211_Various_aspects_of_user_preference_learning_and_recommender_systems

Eclipse Foundation. (n.d.). *WindowBuilder | The Eclipse Foundation*. Eclipse Foundation. Retrieved May 4, 2022, from https://www.eclipse.org/windowbuilder/

Emmersberger, C. (2019, August 18). *Is GraphQL the Resurgence of RPC Style API'S?* Emmersberger.Org. https://emmersberger.org/tag/rpc/

Fhager, R. (2017, December 14). *Pixel Joint Forum: DB's TOOLBOX V1.4 (GrafX2)*. Pixeljoint. https://pixeljoint.com/forum/forum_posts.asp?TID=26080

Fine, A. (2021). *Color theory : a critical introduction*. Bloomsbury Academic.

Fournier-Viger, P. (2017, August 1). *Introduction to the Apriori algorithm (with Java code) | The Data Mining Blog*. The Data Mining Blog. https://data-mining.philippe-fournier-viger.com/introduction-to-the-apriori-algorithm-with-java-code/

Fowler, M., & Taber, C. (2001, January). *Planning and Running an XP Iteration*. MartinFowler.Com. https://martinfowler.com/articles/planningXpIteration.html

Google. (n.d.). *Components - Material Design*. Google. Retrieved November 29, 2021, from https://material.io/components

GOV.UK. (n.d.-a). *Data protection: The Data Protection Act - GOV.UK*. GOV.UK. Retrieved May 4, 2022, from https://www.gov.uk/data-protection

GOV.UK. (n.d.-b). *Intellectual property and your work: Protect your intellectual property - GOV.UK*. GOV.UK. Retrieved May 4, 2022, from https://www.gov.uk/intellectual-property-an-overview/protect-your-intellectual-property

GOV.UK. (n.d.-c). *Intellectual property and your work: What intellectual property is - GOV.UK*. GOV.UK. Retrieved May 4, 2022, from https://www.gov.uk/intellectual-property-an-overview

Howell, D. (2022, April 13). *Man creates search engine that only searches for half-loaves of bread. - Neowin*. Neowin LLC. https://www.neowin.net/news/man-creates-search-engine-that-only-searches-for-half-loaves-of-bread/

IBM. (n.d.). *The structure of a SOAP message - IBM Documentation*. IBM Corporation. Retrieved November 25, 2021, from https://www.ibm.com/docs/en/integration-bus/10.0?topic=soap-structure-message

Intellectual Property Office. (2015, March 12). *Non-disclosure agreements - GOV.UK*. GOV.UK. https://www.gov.uk/government/publications/non-disclosure-agreements

Intellectual Property Office and Government Digital Service. (2020, January 30). *Sui generis database rights - GOV.UK*. GOV.UK. https://www.gov.uk/guidance/sui-generis-database-rights

Jacobson, D., Brail, G., & Woods, D. (2012). *APIs: A Strategy Guide* . O'Reilly Media, Inc.

Jeffries, R. (1998). *What is Extreme Programming?* Ronald E Jeffries. https://ronjeffries.com/xprog/what-is-extreme-programming/

Jeffries, R. (2006a, April 29). *Continuous Integration Relentless Testing*. ExtremeProgrammingRoadmap. http://xp.c2.com/ContinuousIntegrationRelentlessTesting.html

Jeffries, R. (2006b, April 29). *Extreme Programming For One*. ExtremeProgrammingRoadmap. http://xp.c2.com/ExtremeProgrammingForOne.html

Jin, B., Sahni, S., & Shevat, A. (2018). *Designing Web APIs* . O'Reilly Media, Inc.

Johnson, J. (2014). Designing with the Mind in Mind: Simple Guide to Understanding User Interface Design Guidelines. In *Designing with the Mind in Mind: Simple Guide to Understanding User Interface Design Guidelines: Second Edition*. Elsevier Inc.

Kofler, M. (2004). What Is MySQL? In *The Definitive Guide to MySQL*. Apress, Berkeley, CA. https://doi.org/10.1007/978-1-4302-0669-9_1

Konstan, J. A. (2004). Introduction to recommender systems: Algorithms and Evaluation. *ACM Transactions on Information Systems (TOIS)*, *22*(1), 1–4. https://doi.org/10.1145/963770.963771

Krogh, J. W. (2020). MySQL Workbench. In: MySQL 8 Query Performance Tuning. In *MySQL 8 Query Performance Tuning*. Apress, Berkeley, CA. https://doi.org/10.1007/978-1-4842-5584-1_11

legislation.gov.uk. (n.d.). *Computer Misuse Act 1990*. Legislation.Gov.Uk. Retrieved May 4, 2022, from https://www.legislation.gov.uk/ukpga/1990/18/contents

Letkowski, J. (2015, January). *Doing database design with MySQL*. ResearchGate. https://www.researchgate.net/publication/271910489_Doing_database_design_with_MySQL

Liao, K. (n.d.). *Prototyping a Recommender System Step by Step Part 1: KNN Item-Based Collaborative Filtering | by Kevin Liao | Towards Data Science*. Retrieved November 24, 2021, from https://towardsdatascience.com/prototyping-a-recommender-system-step-by-step-part-1-knn-item-based-collaborative-filtering-637969614ea

Lischke, M. (2022). *MySQL :: Re: Is the Workbench Community edition free for commercial usage?* Oracle. https://forums.mysql.com/read.php?152,606940,606963

Lord, R. (n.d.). *API Reference*. Slate. Retrieved May 4, 2022, from https://clickpost.github.io/slate/?hsCtaTracking=337d2141-e3c6-4f0d-9253-da1685f6deee%7C74742ad9-27cf-4300-8d0c-6e0169bba81d#introduction

Lucid Content Team. (2022). *What Is the Extreme Programming Methodology? | Lucidchart Blog*. LucidChart Software Inc. https://www.lucidchart.com/blog/what-is-extreme-programming

McCarthy, N. (2021, March 5). *The Enormous Scale Of Global Food Waste [Infographic]*. Forbes. https://www.forbes.com/sites/niallmccarthy/2021/03/05/the-enormous-scale-of-global-food-waste-infographic/?sh=6d47abd626ac

McKay, E. N. (2013). *UI is communication : how to design intuitive, user centered interfaces by focusing on effective communication*. Elsevier Inc.

McLellan, S. G., Roesler, A. W., Tempest, J. T., & Spinuzzi, C. I. (1998). Building more usable APIs. *IEEE Software*, *15*(3), 78–86. https://doi.org/10.1109/52.676963

MySQL. (2022a). *MySQL :: MySQL Connector/J 8.0 Developer Guide :: 1 Overview of MySQL Connector/J*. Oracle. https://dev.mysql.com/doc/connector-j/8.0/en/connector-j-overview.html

MySQL. (2022b). *MySQL :: MySQL Connector/J 8.0 Developer Guide :: 7.1 Connecting to MySQL Using the JDBC DriverManager Interface*. Oracle. https://dev.mysql.com/doc/connector-j/8.0/en/connector-j-usagenotes-connect-drivermanager.html

MySQL. (2022c). *MySQL :: MySQL Workbench Features*. Oracle. https://www.mysql.com/products/workbench/features.html

MySQL. (2022d). *MySQL :: MySQL Workbench Manual :: 9.1.1.3 EER Diagrams*. Oracle. https://dev.mysql.com/doc/workbench/en/wb-eer-diagrams-section.html

Novoseltseva, E. (2020, February 3). *Extreme programming: tips & advantages | Apiumhub*. Apiumhub. https://apiumhub.com/tech-blog-barcelona/extreme-programming-tips-advantages/

O'hEocha, C., & Conboy, K. (2010). The Role of the User Story Agile Practice in Innovation. *Lecture Notes in Business Information Processing*, *65 LNBIP*, 20–30. https://doi.org/10.1007/978-3-642-16416-3_3

Oracle. (2020a). *Connection (Java Platform SE 7)*. Oracle. https://docs.oracle.com/javase/7/docs/api/java/sql/Connection.html

Oracle. (2020b). *System Properties for Java 2D(TM) Technology*. Oracle. https://docs.oracle.com/javase/7/docs/technotes/guides/2d/flags.html

Oracle. (2020c). *The jpackage Command*. Oracle. https://docs.oracle.com/en/java/javase/14/docs/specs/man/jpackage.html

Oracle. (2020d). *Throwable (Java Platform SE 7)*. Oracle. https://docs.oracle.com/javase/7/docs/api/java/lang/Throwable.html#printStackTrace()

Park, D. H., Kim, H. K., Choi, I. Y., & Kim, J. K. (2012). A literature review and classification of recommender systems research. *Expert Systems with Applications*, *39*(11), 10059–10072. https://doi.org/10.1016/J.ESWA.2012.02.038

Pinto, J. K., & Prescott, J. E. (1990). PLANNING AND TACTICAL FACTORS IN THE PROJECT IMPLEMENTATION PROCESS. *Journal of Management Studies*, *27*(3), 305–327. https://doi.org/10.1111/J.1467-6486.1990.TB00249.X

Plante, T. B., & Cushman, M. (2020). Choosing color palettes for scientific figures. *Research and Practice in Thrombosis and Haemostasis*, *4*(2), 176. https://doi.org/10.1002/RTH2.12308

Prechelt, L. (1999). Comparing Java vs. C/C++ Efficiency Differences to Interpersonal Differences. *Communications of the ACM*, *42*(10), 109–112. https://doi.org/10.1145/317665.317683

Progress. (2022). *What Is a JDBC Driver?* Progress Software Corporation. https://www.progress.com/faqs/datadirect-jdbc-faqs/what-is-a-jdbc-driver

Ranga, V., & Soni, A. (2019). API Features Individualizing of Web Services: REST and SOAP. *Article in International Journal of Innovative Technology and Exploring Engineering*, *8*(9S), 2278–3075. https://doi.org/10.35940/ijitee.I1107.0789S19

RapidAPI. (2021a, September). *What is RapidAPI?* RapidAPI. https://docs.rapidapi.com/docs/what-is-rapidapi

RapidAPI. (2021b, September 17). *Privacy Policy | RapidAPI*. RapidAPI. https://rapidapi.com/privacy/

RapidAPI. (2022, April 29). *Terms of Service | RapidAPI*. RapidAPI. https://rapidapi.com/terms/

Rubel, D., Wren, J., & Clayberg, E. (2012). *The Eclipse Graphical Editing Framework (GEF)* . Addison-Wesley. https://books.google.co.uk/books?hl=en&lr=&id=Mo0beOLdmLMC&oi=fnd&pg=PR9&dq=windowbuilder+eclipse&ots=114VLbxe78&sig=J9sCPX88rCgrYH62s_zJgFRUsTQ&redir_esc=y#v=onepage&q=windowbuilder%20eclipse&f=false

Sadogursky, B. (2021, March 16). *Java 16 Release: jpackage Is Now Production Ready*. JFrog Ltd. https://jfrog.com/blog/java-artifacts-just-got-better-jpackage-is-production-ready-in-java-16/

Salesforce. (2021). *Composite | REST API Developer Guide | Salesforce Developers*. Salesforce.Com, Inc. https://developer.salesforce.com/docs/atlas.en-us.api_rest.meta/api_rest/resources_composite_composite.htm

Schlatter, Tania., & Levinson, Deborah. (2013). *Visual Usability : Principles and Practices for Designing Digital Applications.* Elsevier Inc.

Slack. (n.d.). *Using the Slack Web API* . Slack Api. Retrieved November 25, 2021, from https://api.slack.com/web

Symeonidis, P., Nanopoulos, A., & Manolopoulos, Y. (2007). Feature-Weighted User Model for Recommender Systems. *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, *4511 LNCS*, 97–106. https://doi.org/10.1007/978-3-540-73078-1_13

Techolution. (2019, July 17). *Different API Types*. Techolution. https://techolution.com/types-of-apis/

The Apache Software Foundation. (2021). *RequestBuilder (Apache HttpClient 4.5.13 API)*. The Apache Software Foundation. https://hc.apache.org/httpcomponents-client-4.5.x/current/httpclient/apidocs/org/apache/http/client/methods/RequestBuilder.html

UCLan. (2019). *Intellectual Property Regulations*. University of Central Lancashire. https://www.uclan.ac.uk/assets/student-contracts/2022-23/intellectual-property-regulations-2223.pdf

United Nations. (n.d.). *Food Loss and Waste Reduction | United Nations*. United Nations. Retrieved May 4, 2022, from https://www.un.org/en/observances/end-food-waste-day

Wells, D. (1999). *Release Planning*. ExtremeProgramming.Org. http://www.extremeprogramming.org/rules/planninggame.html

Wickline, M., & Human-Computer Interaction Resource Network. (2001). *Coblis — Color Blindness Simulator – Colblindor*. Colblindor. https://www.color-blindness.com/coblis-color-blindness-simulator/

Wong, E. (2020). *Shneiderman's Eight Golden Rules Will Help You Design Better Interfaces | Interaction Design Foundation (IxDF)*. Interaction Design Foundation. https://www.interaction-design.org/literature/article/shneiderman-s-eight-golden-rules-will-help-you-design-better-interfaces

Young. (n.d.). *Lecture #25: Human-Computer Interaction and Web Design CS106E, Young*. Stanford University. Retrieved November 29, 2021, from https://web.stanford.edu/class/cs106e/lectureNotes/L25NHCIWebDesign.pdf

Zakia, R. D. (2013). *Perception and imaging : photography - a way of seeing*. Focal Press.

Appendix 1 – Project Proposal

Student Name	Tanvi Prakash Gavali
Course	Software engineering, BSc (Hons)
Project Title	Shopping Management Desktop Application

Project Context

The project is a desktop application for individual users. It is a small business solution which takes in the information of the items bought by the user in an online grocery store and manages these items as per the best before dates, calories, etc. and compiles them into a meal plan. This information would be taken in from the store database. Since this is a university project, I would be creating a simple store mock database system instead. Moreover, it allows the application to create a meal plan depending on the items bought by the customer. This meal plan would be constructed by creating simple recipes found using the Google API *SerpApi*.

There is a big scope for this project to go ahead with advance features like being able to give options to choose different recipes, favourite the recipes recommended, being able to see your most common used and accepted recipes, being able to write your own recipes, delete, add, modify and share as per required. Each user will have their own local database created on their desktop enabling the personalised data to be saved.

The benefits of taking this project are the possibilities of further development into a mobile or a web platform, getting multiple grocery online stores integrated all in one app with multiple user functionality for families, couples and house roommates.

Though there have been applications introduced with a management concept before, there haven't been many implementations of an integration between the grocery stores online allowing customers to manage the items bought directly. The original concept from this idea is taken from the Ocado receipts which provides best before dates on paper. These receipts can be used to plan meals. The idea of this project is to provide that functionality electronically and automatically. Recently noted ASDA started to generate recipes in their website and allow its customers to make meal plans. However, they are not generated automatically and therefore, consume a lot of time. Moreover, the customers end up buying more items during that process and the application does not make meals from what customers have already bought.

Specific Objectives

Objectives:

- Able to generate a meal plan from the items available with recipes recommended from the Google API
- Constructs a functional and appropriate meal plan for its users
- Offers a variety of customization options to the user
- Simplest, clean, and modern user interface design for easy navigation
- Maintainable and well documented code using appropriate design patterns
- Able to form and maintain a local database/server to save favourites, modified recipes, most used recipes, etc.

References

1. Freyne, J., & Berkovsky, S. (2010). Recommending Food: Reasoning on Recipes and Ingredients. *User Modeling, Adaptation, and Personalization*, 381–386. https://doi.org/10.1007/978-3-642-13470-8_36

2. Freeman, E., Robson, E., Bates, B., & Sierra, K. (2008). Head First Design Patterns. In *Google Books*. "O'Reilly Media, Inc." https://books.google.co.uk/books?hl=en&lr=&id=NbCNAQAAQBAJ&oi=fnd&pg=PR25&dq=design+patterns&ots=JjaZ9pUupr&sig=ot-Rc5MQ8HsesxNMYbagBR96qms#v=onepage&q=design%20patterns&f=false

3. Oikonomou, T., Votis, K., Tzovaras, D., & Korn, P. (2010). Designing and Developing Accessible Java Swing Applications. *Lecture Notes in Computer Science*, 186–188. https://doi.org/10.1007/978-3-642-14097-6_30

4. Daniel, F., Yu, J., Benatallah, B., Casati, F., Matera, M., & Saint-Paul, R. (2007). Understanding UI Integration: A Survey of Problems, Technologies, and Opportunities. *IEEE Internet Computing*, *11*(3), 59–66. https://doi.org/10.1109/mic.2007.74

5. Freyne, J., & Berkovsky, S. (2010). Intelligent food planning. *Proceedings of the 15th International Conference on Intelligent User Interfaces - IUI '10*. https://doi.org/10.1145/1719970.1720021

6. *Development of a web-to-mobile program that generates personalized meal plans for athletes - ProQuest*. (n.d.). Www.proquest.com. Retrieved October 11, 2021, from

https://www.proquest.com/openview/1b68996349c748c775da73d6eea3e88e/1?pq-origsite=gscholar&cbl=18750

7. *Download Limit Exceeded*. (n.d.). Citeseerx.ist.psu.edu. Retrieved October 11, 2021, from https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.11.3632&rep=rep1&type=pdf

Potential Ethical or Legal Issues

Legal Issues:

- The use of the Google API recipes should be free to use.
- Use of data must obey GDPR laws.

Ethical Issues:

It might not generate the recipes from the API as expected

- Meals that are too complex
- Meals that are unfamiliar

I would be responsible for the recipe's not being hazardous:

- dietary needs (glucose free, lactose intolerance, etc.)
- allergies
- religious diets
- unhealthiness
- etc.

Not only could use of data be considered a legal concern (GDPR), but it may also be considered an ethical concern as people may not want their data being sent to construct a meal plan from the Google API

Resources

- IDE - https://www.eclipse.org/
- Language – Java
- Swing – GUI widget toolkit for Java
- MySQL Workbench 8.0 CE – https://www.mysql.com/products/workbench/
- Google API - https://serpapi.com/recipes-results

Potential Commercial Considerations - Estimated costs and benefits

In the preliminary search, similar software was found. The similar software was free but did not offer all of the same features, such as generating a meal plan and making recipes from a shopping list. Other software also does not connect with the Google API. To use this software to its full potential in its advanced stages, this software would need permissions of grocery companies for a full database of products and convenience for the user. These companies need to support online shopping.

The software could also support offline shopping similar to Amazon Fresh systems which keeps track of the customer ID and names even in offline market visits. This will allow the software to use the database of the customers so that the software can automatically update the items and recommend meal plans accordingly no matter which grocery shop they went to.

As this is a new concept in the technology it would be more economically viable to sell this software now then it would in two years before other companies develop the same technology. The software could be sold to a company as part of a bigger project or this project at later stages if successful can get revenue from the grocery companies it is getting its databases from. The project can be expanded in many different categories and hence the price would depend on the advancement and development done in this project.

Proposed Approach

Milestones for the project and their estimated time:

- **Make a basic prototype in the start weeks**
  - **Data and UI setup - 1-2 weeks**
    - creating the client mock database of the online grocery store (create UML diagram to get the table relationships, create the database and the tables, add simple examples inside database, connect the database to the program)
    - create SQLManager library to run the function queries of that database, this should be able to sort the table in asc-dsc as per the individual needs
    - Create a basic swing UI layout using the WindowBuilder.
  - **Algorithm - 2 weeks**
    - Create test data in the program and a simple algorithm to make recipes from the test ingredients (stored in strings for ingredients and Recipe objects)
    - List unused ingredients to the user
- **Create a more polished product with API functionality**
  - **API setup - 4 week**
    - understand the methods and functions used in the API, connect the API to the program
    - Make a mock API-application conversation first, then get simple recipes to display in the program
  - **API with algorithm - 4 weeks**
    - Get recipes from the API based on ingredients displayed (from the database),
    - Later get ingredients based on the best before dates so that it will display the recipes which will have to use up the ingredients before they go off.
    - Generate a simple meal plan with breakfast, lunch, and dinner, displaying the recipes received from the API
  - **Extra Features - 4 weeks**
    - Create a change option in the meal plan allowing users to see list of recipes recommended or an option of adding their own recipes.

- Able to write recipes and save in their web localhost either as a json file or in their local folder where the application is downloaded creating json files which could be shared on social media.

- **Reformatting and OOP**
  - **Evaluate current design – 2 weeks///**
    - Look into which design patterns would be appropriate
    - Decide which features could be re-used
  - **Reformat structure – 3 weeks**
    - Implement design patterns where appropriate
    - Create libraries for ease of expansion later

- **Polish User Interface**
  - **1 - 2 weeks**
    - Use graphics libraries (Graphics2D etc.) to make more professional looking Swing components

Appendix 2 – Technical Plan

Name: Tanvi Prakash Gavali

Course: Software engineering, BSc (Hons)

Supervisory Team: TBC

Title

Shopping Management Desktop Application

Summary

This project is a Java desktop application which provides a Swing UI to manage individual user's grocery items and develop a meal plan using a Google Recipes API. This is to help them to manage meal plans and prioritise food items to cook and budget in less time. Items bought by the user will automatically update in the program since the userID will be connected to the grocery shop database which will be managed using MySql Workbench. After Covid-19, more people have been comfortable getting online groceries and hence, this program further facilitates them in planning their meals.

Users will be able to check all their items they currently have in their kitchen including the items they recently bought from the store. They could categorise their items either alphabetically or based on calories, best before dates or weight. They would also be able to generate meal plans through the SerpApi API to access Google Recipes, favorite recipes which have been recommended and create/edit new recipes. This will allow all the users to have more control and manage their budget and items well.

Challenges

Since this is a university project, getting a database from actual grocery stores is not possible. Hence, I will be creating a mock database for this to demonstrate the project's purpose.

Due to ease of quarantine restrictions, more people have started visiting markets and hence there is no guarantee of keeping the database up to date. Still, stores like AmazonFresh, who keep an update of their customers on visits and online can benefit from this program.

Regarding technical approach, this project has a scope of advancing a lot even after the final year and therefore, it is very important to have a maintainable code with right design pattern implementations. From a software perspective, it's necessary for the program to adapt to the changing requirements over time.

There is a chance of having API connection issues and generating the appropriate meal plans which I would have to be prepared for in future. The mock database created might not be exactly the way the grocery stores store their customer data and therefore, the lack of consistency could be seen.

The solution to this would be to first make a mock up localhost server which could act like a test API and input and output information through it. I should be able to demonstrate the foundations of the sending and receiving information and this could help further stages of using an external Google API.

One issue might be choosing the correct design patterns for the project. As the design might change over time it is not clear what the most appropriate patterns would be. Because of this, the project will take a more modular approach at first and use design patterns when needed. This will work well as design patterns will be used when required and can be used in combinations as the project gets more complex. (Kuchana, 2004)

Objectives

- Able to generate a meal plan from the items available with recipes recommended from the Google API
- Constructs a functional and appropriate meal plan for its users
- Offers a variety of customization options to the user
- Simplest, clean, and modern user interface design for easy navigation
- Maintainable and well documented code using appropriate design patterns
- Able to form and maintain a local database/server to save favourites, modified recipes, most used recipes, etc.

Methodology Summary

Since my project has a large scope of adding more features in the future; I will be taking inspiration from some software methodologies and frameworks where the methods suit small teams. I will be using a Kanban board like Trello and add MoSCoW Analysis so as to not compromise and distract myself with other features and prioritise the essential features first. I will set deadlines in each sprint so as to manage the time constraints during this period. Adapting to changing requirements may also be an issue and that is why the agile medhodology would be use. I have also been taking inspiration from the Rapid Application Development (RAD) methodology as this is a project with a short time frame which suits the processes involved in RAD.

Deliverables

I will be creating a Java program which can be executed in the Java Runtime Environment on multiple platforms. I will be using the jpackage incubator module, a java packaging tool to convert the Runnable JAR into a .exe and Inno Setup Script, to convert the executable into an installer packaged with relevant libraries and folders. I would also be creating a server to store relevant information like the customer's saved/used recipes, new recipes, modified recipes, etc.

Constraints

Time constraints will play a big factor due to both project deadlines and other assignments. There is a very short period of time to dedicate and therefore I have to be careful in getting my sprints done on time. Appropriate guidance required in the right time will also play a major role as this should help me keep a track of my progress. Finding appropriate resources of the concepts I will be implementating could be a time consuming and

challenging element. To ensure the quality of this project, I have to be strict with my project management.

Key Problems

As the project complexity increases over time, it would be difficult to manage the code properly. From the start while implementing I have to be careful in making component systems which are independent and flexible (to promote loose coupling) which could be modified even at a later time without breaking the program.

I also have to be observant enough to find and implement correct architectural and design patterns. To do this, I can create UML diagrams before setting all the classes in the program.

In terms of the program itself I have to make sure I am able to identify the user needs. The best way to go ahead is to make a basic prototype of the software first and get an overall feedback from my supervisor, family and friends. I would have to make sure at the later stage to create appropriate GUI designs in Figma for comfortable user experience while using this program. As well as this, I can test against usability guidelines to make sure that the app is more accessible e.g., for color blind people. (Oikonomou et al., 2010)

One problem would be to understand the relationship between the components in the mock store database I would be creating. Since the databases are kept confidential it is a bit difficult to find a consistent database examples used in big companies. For this I will using Enhanced Entity-Relationship (EER) diagrams for modelling interfaces in MySQL Workbench to understand the relationships among the tables and entities. (*MySQL :: MySQL Workbench Manual :: 9.1.1.3 EER Diagrams*, n.d.)

Another problem could be connecting to the Google API. There aren't many resources other than the website documentation they have provided. So I reckon this is going to be challenging.

Since the project has a lot of scope it would not be possible to implement all the extra advanced features and therefore limiting certain functionalities would be required.

System and Work Outline

The program will be developed in Eclipse using Java and Swing as its user interface with the help of the window builder.

Methods of analysis, design and implementation

Several types of analysis would be carried out, including analysing user needs and the priorities of the project.

I am going to design my components first using CRC cards and make a UML diagram to understand the relationships between the classes created. This modelling will also help me to decide which Swing components to use in the project to promote a more model-driven development (MDD) (Almendros-Jimenez & Iribarne, 2005). To analysis the user needs I would have to create user stories and conduct noun verb analysis.

This will be helped by also conducting user testing, to check if the product is equivalent with the users. This analysis will be done to ensure that the users are satisfied with the product and also that it gets done within the time frame.

Development Environment – Languages and Tools

I have created desktop applications before using Eclipse IDE framework with Windows Builder components with Swing as its Graphical Interface and therefore, I am comfortable using the Java Language in Maven as my project management tool. I am planning to use a simple mock database in MySqlWorkbench allowing me to access and manage the database using the GUI tools provided by that program. I would also have to understand json formatting for the basic API setup needed for my project. Further towards the deployment stage, I would have to use Pascal to write the installer Inno Setup Script.

High level design breakdown

There will be four main high level components to this project which will include desktop application, store database, local database, API server. The desktop application and the two databases will also include sub components such as classes or tables.

The desktop application sub components will include:

- LogFileManager – responsible for creating and updating .txt logs for warnings and errors.
- SQLManager - storing and executing sql queries.
- UserManager – manages the user login details
- DesignManager – handles visual changes applied to the GUI components
- ServerManager – sents queries and receives responses from the external API
- Interface – Initialises components and their event handlers/listeners.

The Store database sub components will include:

- Customer table
- Item table

The local database sub components will include:

- Recipes saved and favourited
- Ingredients in the cupboard

API Server

The API server will not be managed by me. It would be an external server so the sub components will not be known. However, it would be interfaced with using the Java Application. The SerpApi API would be an intermediary between the java application and the Google Recipe API to scrape, extract and make sense of the data. (SerpApi, n.d.)

Personal development

There haven't been much research on the SerpApi and limited resources are available. I would have to first put time into understanding the fundamentals of API setups with Java and make test programs first to get more comfortable to implement the generation of the meal plan feature.

MoSCoW Analysis

The following is the MoSCoW Analysis which has been created to outline the different plan features of this project in order of the priorities.

Must have:

1) A basic prototype of the program should be ready
2) UML diagram to get the table relationships should be created
3) System has a client mock store database with simple examples using create, read, update, delete (CRUD) features.
4) Able to connect the database to the program and a SQLManager helper library to run queries of that database
5) A basic GUI layout using Swing is implemented.
6) Simple recipes displayed in the program
7) Recipes retrieved from the API based on ingredients displayed (from the database),
8) Generate a simple meal plan with breakfast, lunch, and dinner, displaying the recipes received from the API
9) Design patterns implemented where appropriate
10) Libraries created for ease of expansion later
11) The project is deployed in a Runnable JAR.

Should have:

1) Program retrieves ingredients based on the best before dates to display the recipes which will have to use up the ingredients before they go off.
2) Change option implemented in the meal plan to allow users to see list of recipes
3) Feature to modify the recommended recipes and providing an option of adding their own recipes
4) Features for writing recipes and save in their web localhost either as a json file or in their local folder where the application is downloaded creating json files which could be shared on social media.
5) Use graphics libraries (Graphics2D etc.) to make more professional looking Swing components
6) Appropriate testing methods used in the program.
7) The project is deployed in an installer

Could have:

1) Application allows multiple users such as friends, families, couples, housemates, etc., can share and make a meal plan together based on the number of servings
2) Allow multiple grocery database connections so that no matter which grocery store the user buys the items from, they all can be updated in their database.
3) Allow the user to add their allergy info, religious info, etc., for better meal generation as per the user customization
4) Further develop the desktop application into a mobile application and a web application.

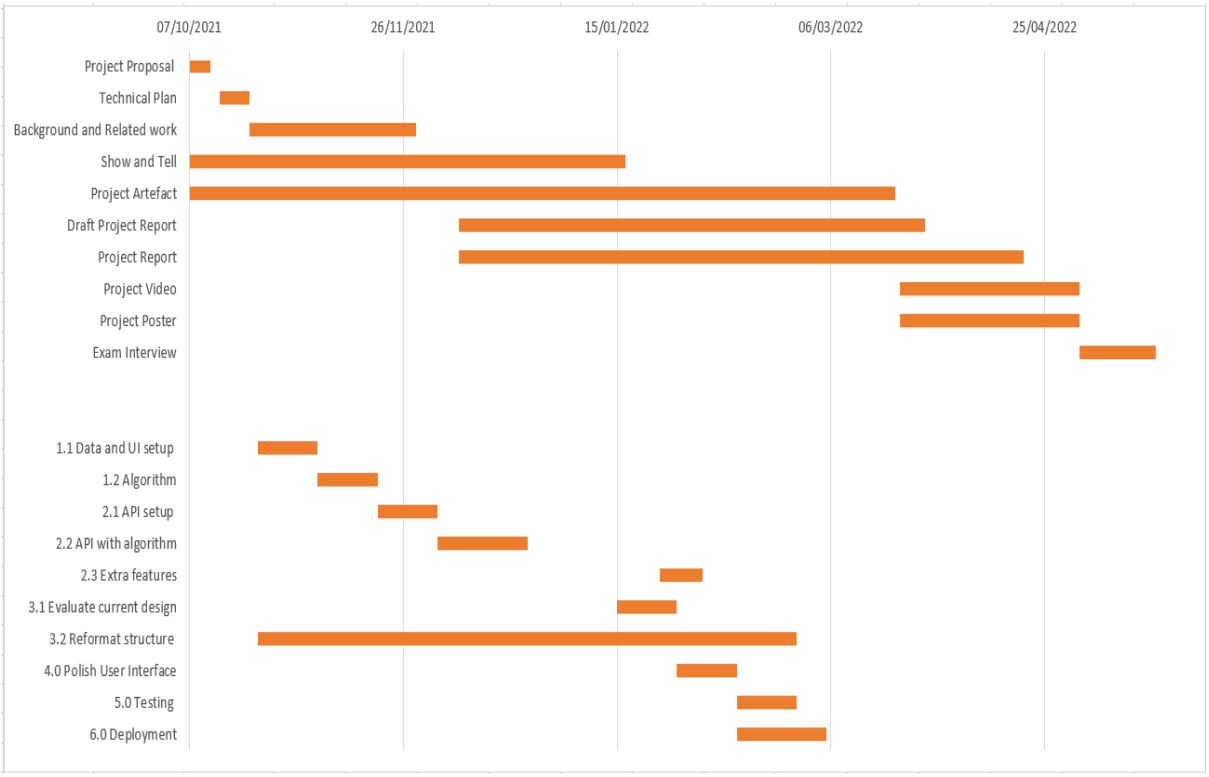
Would have:

1) The meal generation could be customised into a machine learning recommender system
2) Each users information with their modified and saved recipes can further be store in cloud services. For example, this can be stored in buckets in the AWS S3.
3) The stored AWS S3 data can further be converted into graphs using the AWS Quicksight to help the recommender system to learn from its users. This will also allow the developers to understand how many hours users spend on the program, is the program recommending the right meal plans and how many users prefer intercontinental meal plans.

Project Activities

In this section, two diagrams are provided. A Gantt Chart has been created for this project to show the order of which things would be worked on throughout the duration of the project. The first picture, is a table of the different deadlines and activities with their start and end dates. The second is the Gantt Chart.

	Deadlines	Start Date	End Date	Time (Days)
2	Project Proposal	07/10/2021	11/10/2021	5
3	Technical Plan	14/10/2021	20/10/2021	7
4	Background and Related work	21/10/2021	29/11/2021	39
5	Show and Tell	04/10/2021	17/01/2022	105
6	Project Artefact	04/10/2021	21/03/2022	168
7	Draft Project Report	09/12/2021	28/03/2022	109
8	Project Report	09/12/2021	20/04/2022	132
9	Project Video	22/03/2022	03/05/2022	42
10	Project Poster	22/03/2022	03/05/2022	42
11	Exam Interview	03/05/2022	21/05/2022	18
12				
13				
14	**Project**			
15	**1.0. Make a basic prototype**			
16	1.1 Data and UI setup	23/10/2021	06/11/2021	14
17	1.2 Algorithm	06/11/2021	20/11/2021	14
18	**2.0 API functionality**			
19	2.1 API setup	20/11/2021	04/12/2021	14
20	2.2 API with algorithm	04/12/2021	25/12/2022	21
21	2.3 Extra features	25/01/2022	15/01/2022	10
22	**3.0 Reformatting and OOP**			
23	3.1 Evaluate current design	15/01/2022	29/01/2022	14
24	3.2 Reformat structure	23/10/2021	26/02/2022	126
25	**4.0 Polish User Interface**	29/01/2022	12/02/2022	14
26	**5.0 Testing**	12/02/2022	26/02/2022	14
27	**6.0 Deployment**	12/02/2022	05/03/2022	21
28				



Risk Analysis

Below is the table listing different risks and information about them such as their severity and likelihood.

Risk	Severity	Likelihood	Action
Losing database info	High	Medium	Store backup database in google drive and external hard drive
Problems with PC (Harddrive fail, etc)	High	Low	Can solve with backups using version control, google drive, external hard drives, etc.,
Investing time into Google API and not been suitable for what it was needed	High	Medium	Find open source related recipes API instead of focusing on only one
Unable to implement the Must Haves priorities needed in the project	High	Low	Early start of the project and following the sprints should able to reach the goal of the project
Making bad changes to the program and breaking things	Low	Medium	Solve with source control. Can go back to the older versions

Not maintaining the versions of the program in GitHub	Medium	Low	Make sure to commit after making changes in the code to keep consistenty progress

Options

There are multiple reasons I have chosen to use a Kanban board for this project. During my placement year, I was the only developer in the company and hence had to plan most of the projects before the feature deadline. I usually had daily tasks assigned before starting to work on my project. I would come up with simple tasks which I was confident that I could do that day. These tasks would be noted down on the online Kanban board. This enabled me to keep track of my work for both myself and my supervisor in the company. I am going to continue using this approach since it has been beneficial to me and I have become comfortable following this methodology. I could also use other methodologies such as Extreme Programming since it is known for small teams and small companies.

I have chosen Eclipse as my IDE, but have also considered alternatives. I could use other modern IDEs like IntelliJ. Even though IntelliJ is free to use as a student, there are monthly pricings later while using the software professionally after university. Eclipse allows free commercial projects. This would be an advantage for me if I want to make this project commercial later. Another benefit of the Eclipse IDE is that it is popular and therefore, has an abundance of tutorials. (Murphy et al., 2006)

I could have used other relational database software such as Microsoft SQL Server Management and other GUI SQL tools. SSMS 18 does not support EER diagrams which helped me choose MySQL Workbench.

I could have used other external APIs for my recipes such as Yummly API. However, I have to ask their permission personally before using external APIs. The SerpAPI provides 100 searches free for a month, allowing me to have more financial flexibility.

In terms of the deployment system, I could have used Launch4J to convert Runnable JARs into .exes. The Launch4J gives a GUI tool rather than used in a command line making it easier for the user to use. It allows to package small simple projects into executables. However, one big reason for not using this alternative is that can run into many technical problems while packaging large projects with multiple libraries and folders. Thus this can make it challenging to problem solve and this can be time consuming. JPackage allows you to look into metafiles, and the success rate of the conversion is much better and more efficient. It also lets you know any issues it encounters and hence allows me to debug and deploy my projects quicker.

Potential Ethical or Legal Issues

Legal Issues:

It is important for me to make sure the use of the Google API recipes is free to use. Some external APIs might have pricings and I have to be aware of not using API servers without paying when needed. The use of the user data like the user name, user address, etc., must obey GDPR laws. ("Chapter 3 – Rights of the data subject," 2018)

Ethical Issues:

There is a probability that the program might not generate the recipes from the API as wanted since the meals could be too complex or unfamiliar to the users.

I would also have to be responsible for the recipe's recommended are not being hazardous. This includes not generating recipes with excessive fats, calories, salt, etc. This could be tackled by creating healthiness ratings for different recipes and ingredients. (Freyne & Berkovsky, 2010)

In the case of multiple users, I also have to take dietary needs (glucose free, lactose intolerance, etc.,), allergies, religious diets, etc., in consideration.

Not only could use of data be considered a legal concern (GDPR), but it may also be considered an ethical concern as people may not want their data being sent to construct a meal plan from the Google API.

Commercial Analysis

This project will be a free open source project. Since this project has a huge scope in developing further, it is better to keep the software commercially free to use at this stage. Depending on the features added in the software, the value of this program may increase. There is a probability the project could be led into A.I and machine learning which would allow selling commercially as this would be better than the competing products.

In the perspective of a commercial software at a later stage, it can further take revenues from the companies providing the data of their customers. This will allow for more development time and features.

The below table is a rough estimate of the money needed to fund this project if it were a commercial.

Factor name	Description	Is this a cost or a benefit	Estimated Amount	Estimate of when paid
Salary	Payment for the developer	cost	£25,000 per year	Monthly
API	Payment for API usage	cost	Estimated about £50 a month / 100 searches a month free plan	Monthly

Employability Contribution

My final year project will illustrate my database handling and management skills. It will also allow me to demonstrate project and time management skills including the ability to make models, diagrams, relationships, etc. I also think I will gain experience in using external

APIs. API usage in complex application programs show my readiness to learn new concepts and implement them effectively.

This will develop my knowledge of software design patterns as well as helping me on my test driven approach to development. My ability to use project management life cycles and deployment practices in a java project successfully will have hopefully improved during this project. This would also showcase my code management and maintainable skills.

I believe my final year project would also help my problem solving skills. This is because, I have got this idea for this project and so would be responsible for it. It will let me encounter new problems no body would have had before.

This project would also showcase my UI design skills and user requirement needs allowing me to qualify as a full stack engineer.

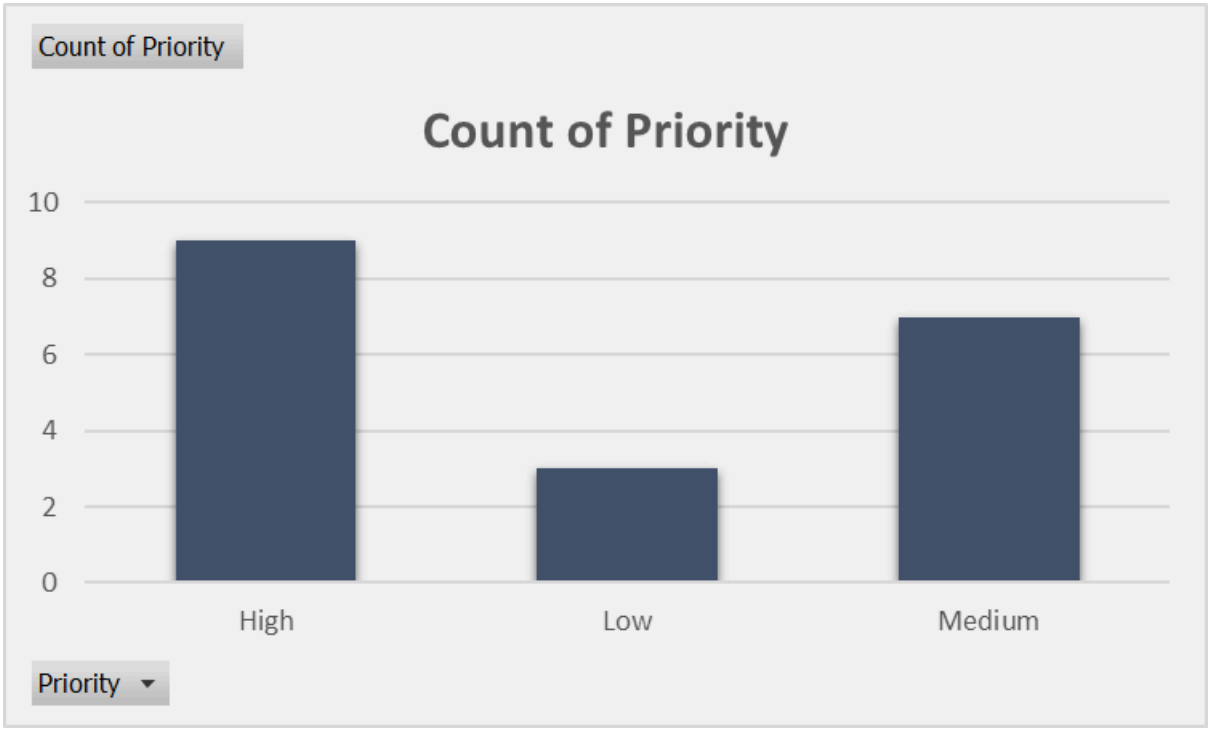
References

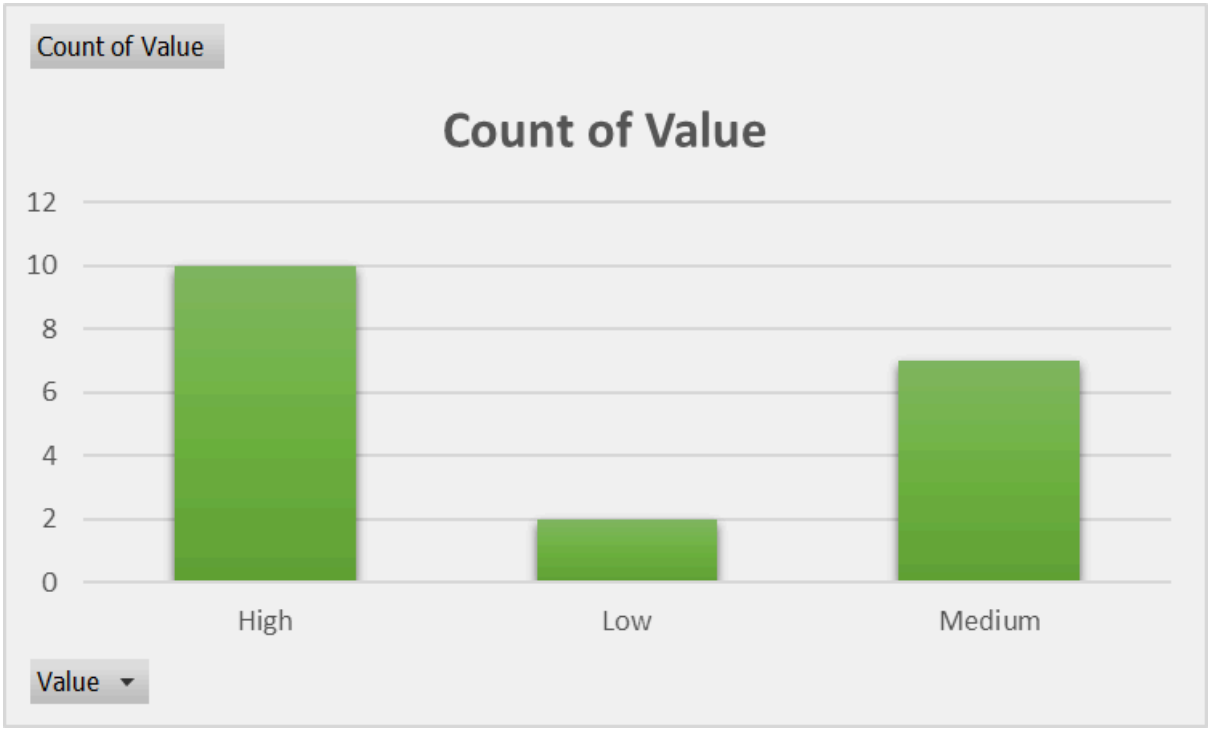
1) *MySQL :: MySQL Workbench Manual :: 9.1.1.3 EER Diagrams*. (n.d.). Dev.mysql.com. Retrieved October 25, 2021, from https://dev.mysql.com/doc/workbench/en/wb-eer-diagrams-section.html
2) *Chapter 3 – Rights of the data subject*. (2018, October 5). General Data Protection Regulation (GDPR). https://gdpr-info.eu/chapter-3/
3) Murphy, G. C., Kersten, M., & Findlater, L. (2006). How are Java software developers using the Elipse IDE? *IEEE Software*, *23*(4), 76–83. https://doi.org/10.1109/ms.2006.105
4) Kuchana, P. (2004). *Software Architecture Design Patterns in Java (1st ed.)*; Auerbach Publications. https://www.taylorfrancis.com/books/mono/10.1201/9780203496213/software-architecture-design-patterns-java-partha-kuchana
5) SerpApi. (n.d.). *Google Recipes Results API*. SerpApi. Retrieved October 25, 2021, from https://serpapi.com/recipes-results
6) Freyne, J., & Berkovsky, S. (2010). Recommending Food: Reasoning on Recipes and Ingredients. *User Modeling, Adaptation, and Personalization*, 381–386. https://doi.org/10.1007/978-3-642-13470-8_36
7) Oikonomou, T., Votis, K., Tzovaras, D., & Korn, P. (2010). Designing and Developing Accessible Java Swing Applications. *Lecture Notes in Computer Science*, 186–188. https://doi.org/10.1007/978-3-642-14097-6_30
8) Almendros-Jimenez, J. M., & Iribarne, L. (2005, April 1). *Designing GUI components from UML use cases*. IEEE Xplore. https://doi.org/10.1109/ECBS.2005.31

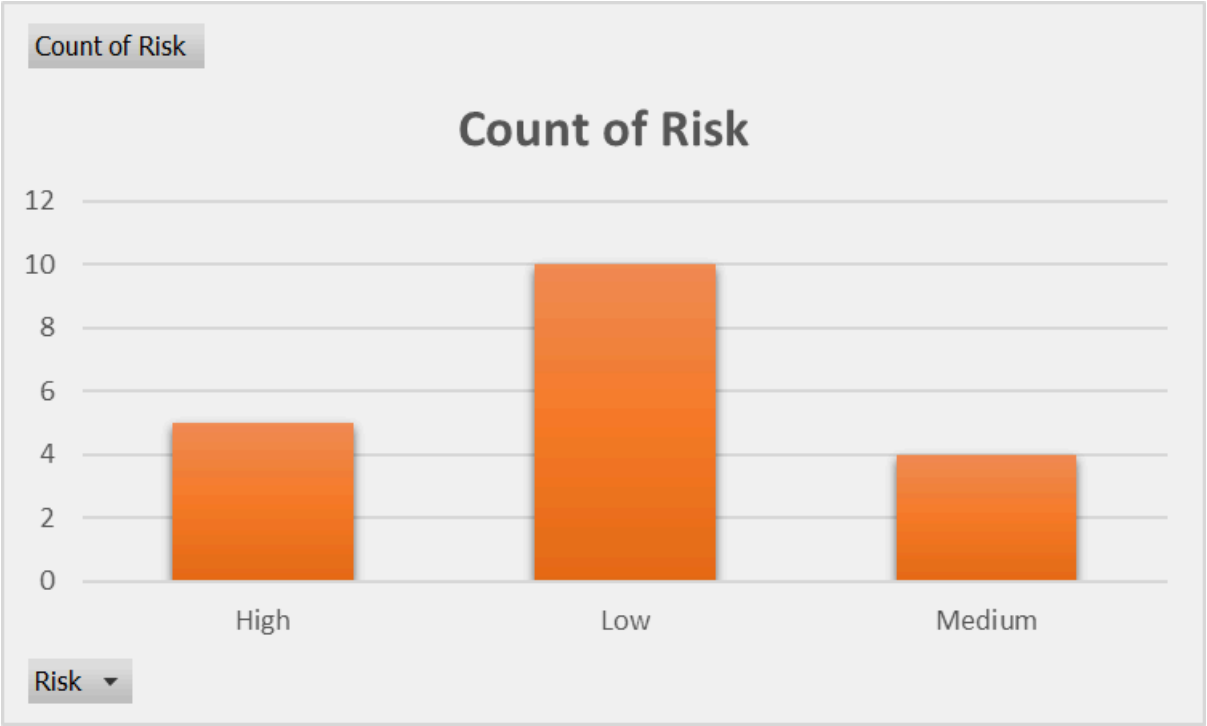
Appendix 3 – Release Plan

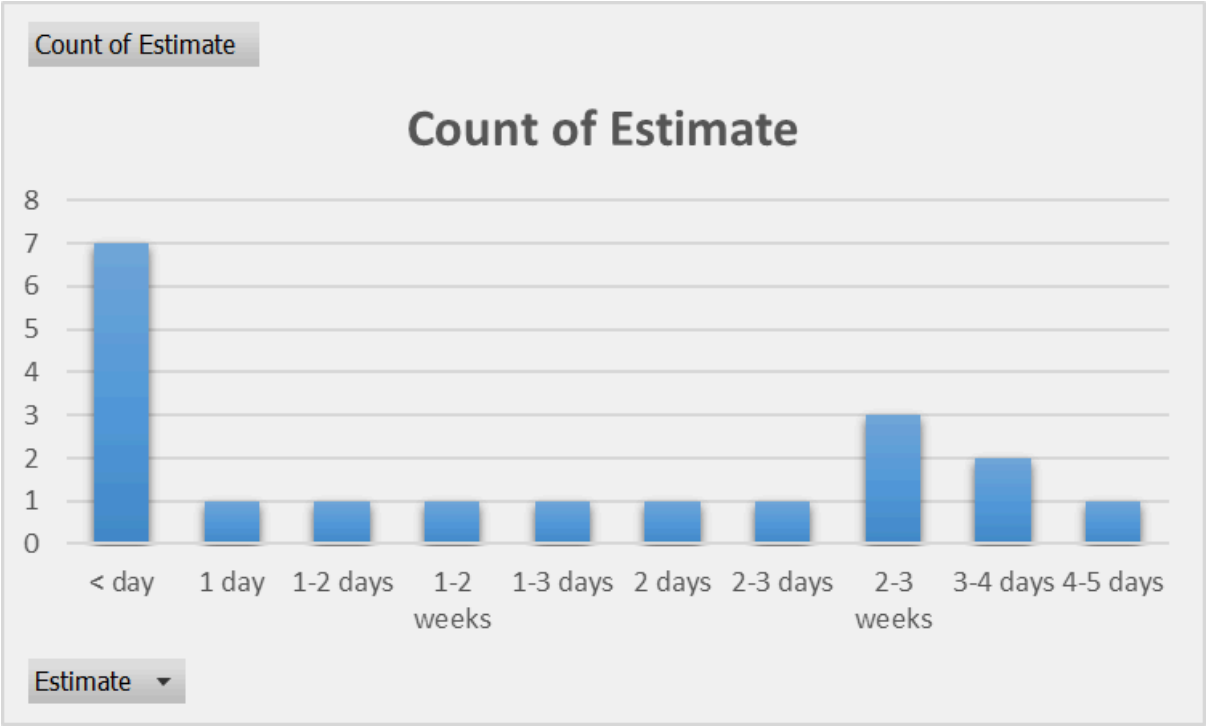
Release Plan

Feature	User Story	Priority	Value	Risk	Estimate
Login account	As a user, I want to be able to put login details so that I can log in the application.	High	High	Low	< day
Login details	As a user, I want the username and password the same as the one used to login for the grocery app so that I can see my recent bought grocery items	High	High	Low	< day
Create a fridge tab	As a user, I want to view all the items with calories, quantity and best before date of each item bought from the grocery store, and items they have currently in their kitchen so that I can keep track of my items	High	High	Low	1 day
Modification in fridge tab	As a user, I want to add and delete the items from the fridge table so that I can update my app when I throw out food or add new ingredient in my fridge.	Medium	Medium	Low	< day
Sort fridge tab	As a user, I want to sort my ingredients in the fridge table with the soonest expiry date so that I know which ingredients are getting expired soon.	Low	Medium	Low	< day
Sort fridge tab	As a user, I want to sort the items from the table in the Fridge tab alphabetically so that I can search for things easily	Low	Low	Low	< day
Generate meal plan	As a user, I want to generate a meal plan from the items I have so that I do not have to worry about creating a meal plan myself.	High	High	High	2-3 weeks
Generate meal plan	As a user, I want to know if the items I have are not enough to create a weekly plan and the application should tell me to buy more ingredients so that I am settled for a week's food.	Medium	High	High	4-5 days
Recipe details	As a user, I want to get the recipe details when I double click on any recipe in the meal plan so that I do not have to search for the recipe and find the descriptions manually.	Medium	Medium	Medium	2 days
Generate new meal plan	As a user, I want to save the current meal plan and generate a new meal plan for the next week so that I can roughly estimate how many food items I must buy.	High	High	Medium	3-4 days
Create a recipe tab	As a user, I want to view a list of general recipes and be able to favourite any recipe I liked so that I can read other recipes which takes my interest	High	High	Low	1-3 days
Create personal recipe	As a user, I want to create my own recipe and save or favourite them for later so that I have my own personal recipes to use.	Medium	Medium	Low	1-2 days
Meal plan	As a user, I want to see my favourited recipes more in my meal plan generation so that the meal plan has the recipes I like.	Medium	Medium	Medium	2-3 days
Create a favourite tab	As a user, I want to view all favourited recipes in a favourite tab and for it to suggest new recipes I might like so that I find more recipes of my taste.	High	High	High	2-3 weeks
Create a suggestion table	As a user, I want to see my suggested recipes as per my cuisine so that the recipes shown are suited to my taste.	Medium	Medium	High	1-2 weeks
Create a recipe finder tab	As a user, I want to search for any recipe or ingredient which should be displayed and remembered so that I can later read those recipes.	High	High	High	2-3 weeks
Account update	As a user, I want to able to update my password in my settings tab so that I do not have to worry in case I forgotten or need to change my password details.	Low	Low	Low	< day
Create preference tab	As a user, I want to able to change any preferences so that my meal plan is customised based on how I like it.	Medium	Medium	Medium	3-4 days
Log out Account	Atlas, as a user, I want to able to log out of the application so that nobody else use my application.	High	High	Low	< day

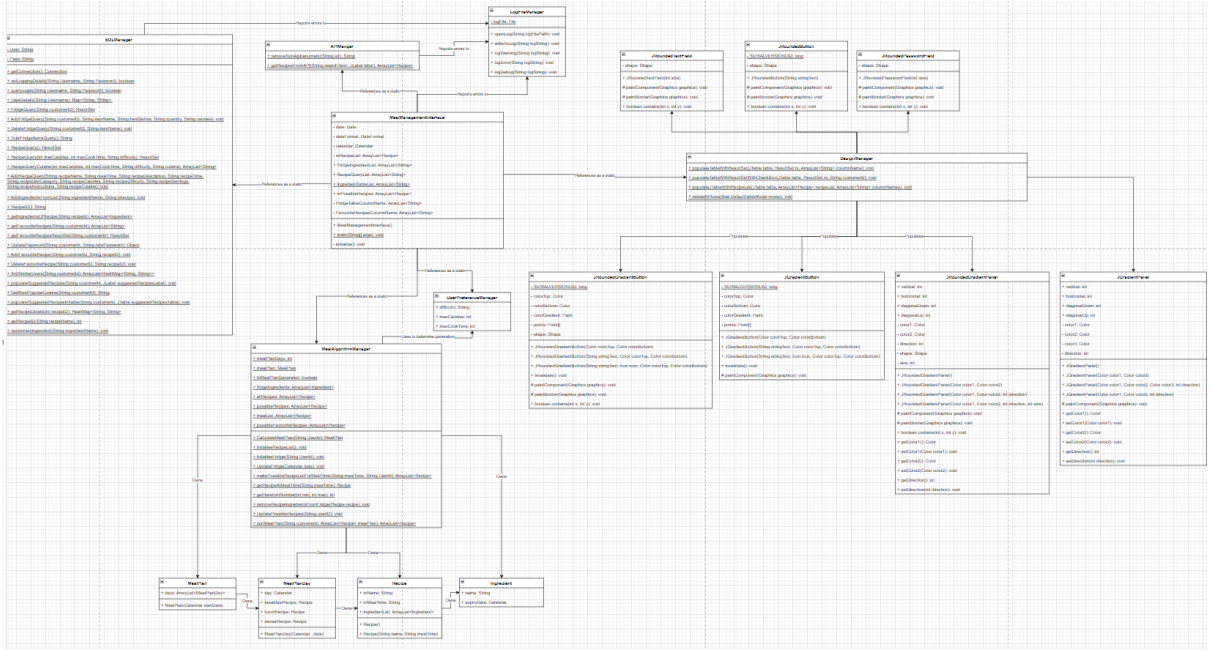

Count of Priority
Count of Priority
10
8
6
4
2
0
High
Low
Medium
Priority


Count of Value
Count of Value
12
10
8
6
4
2
0
High
Low
Medium
Value

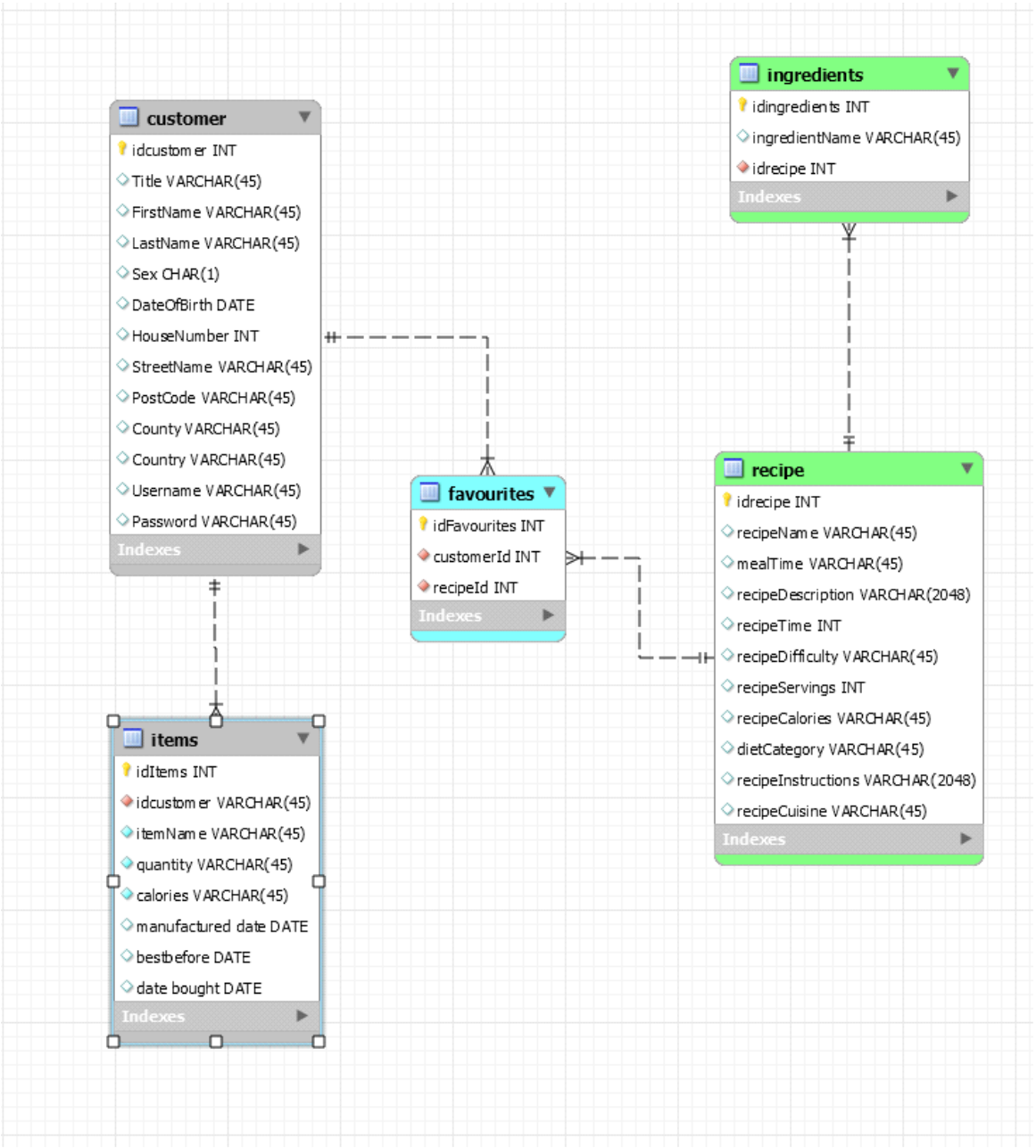

Count of Risk
Count of Risk
12
10
8
6
4
2
0
High
Low
Medium
Risk


Count of Estimate
Count of Estimate
8
7
6
5
4
3
2
1
0
< day
1 day
1-2 days
1-2 weeks
1-3 days
2 days
2-3 days
2-3 weeks
3-4 days
4-5 days
Estimate

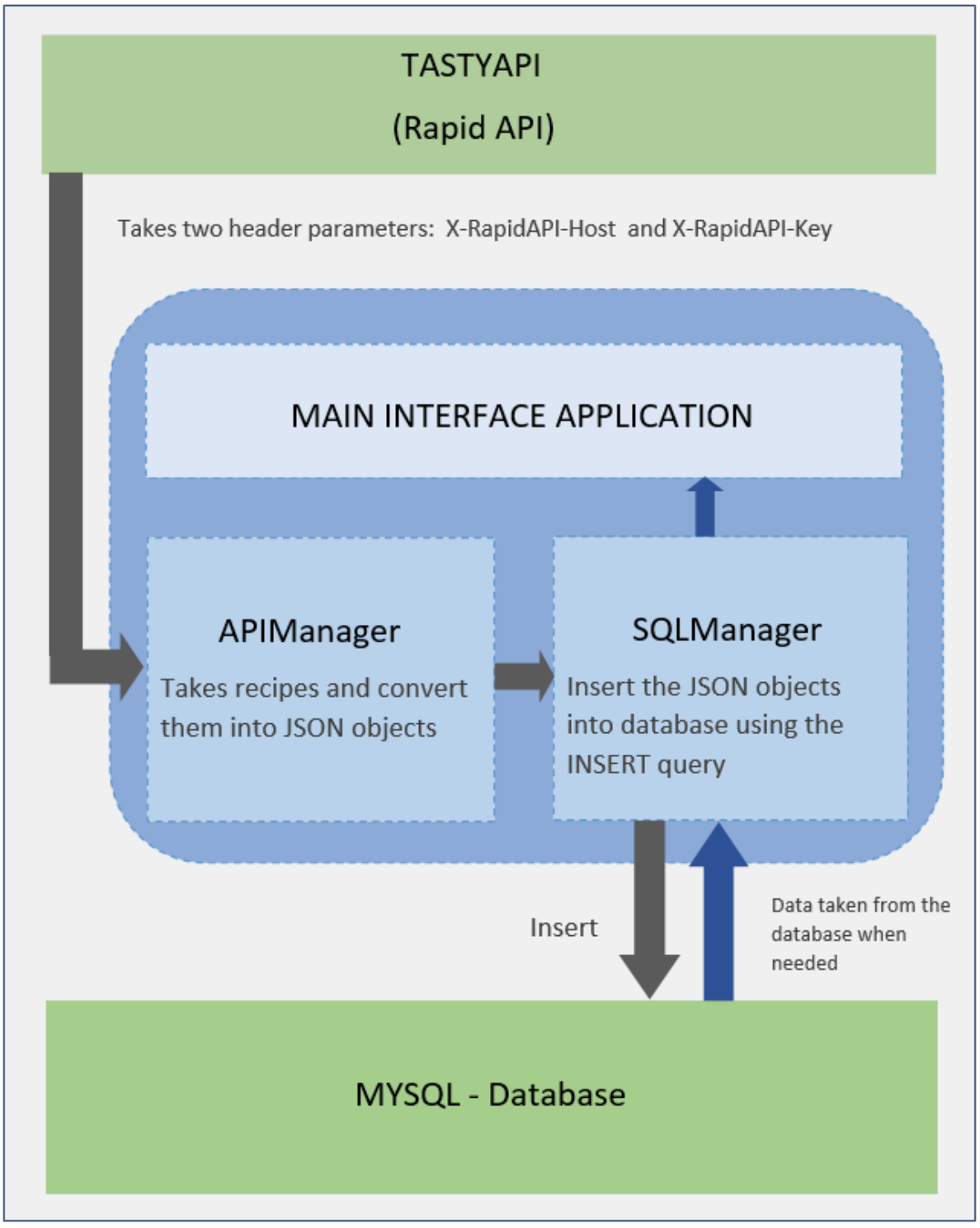
Appendix 4 – UML Diagram



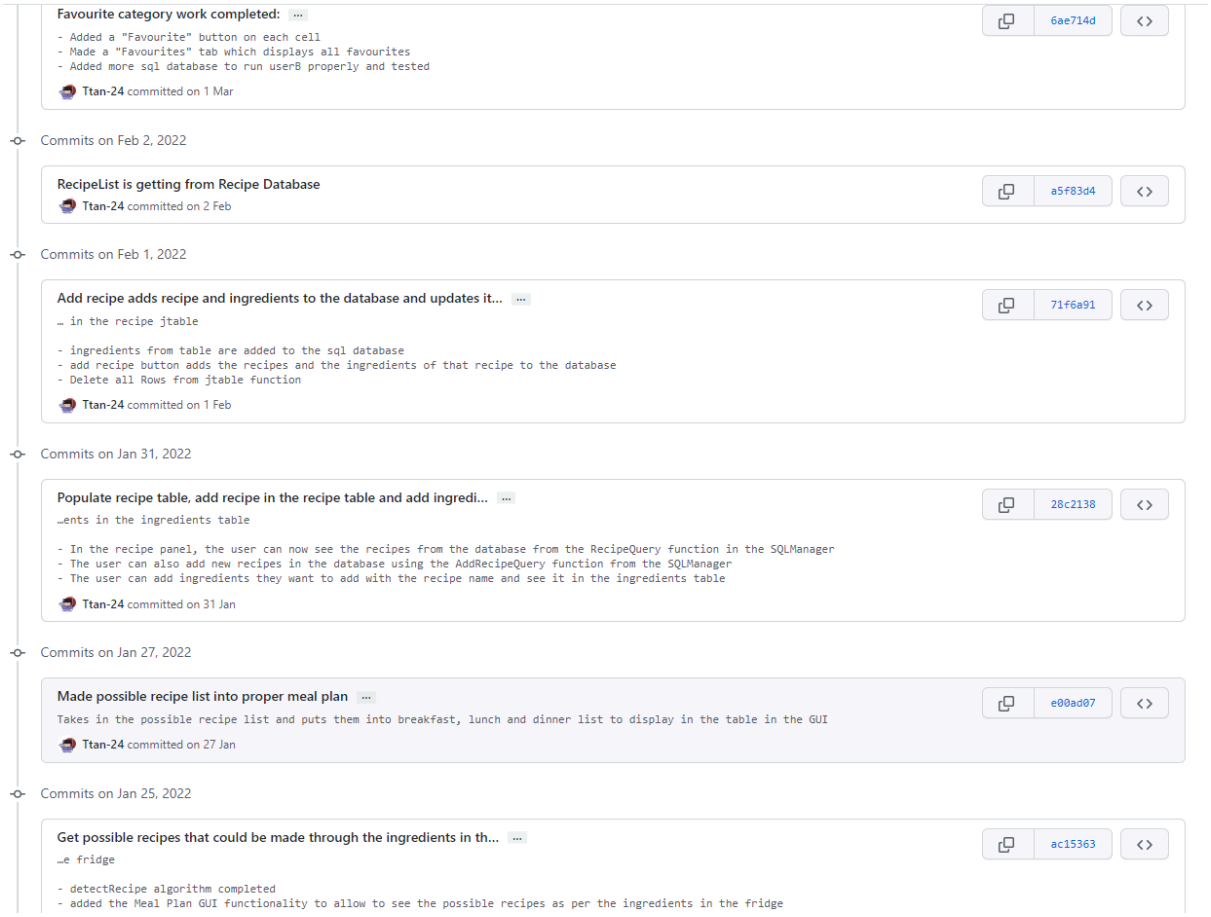
Appendix 5 – Database Design


customer
idcustomer INT
Title VARCHAR(45)
FirstName VARCHAR(45)
LastName VARCHAR(45)
Sex CHAR(1)
DateOfBirth DATE
HouseNumber INT
StreetName VARCHAR(45)
PostCode VARCHAR(45)
County VARCHAR(45)
Country VARCHAR(45)
Username VARCHAR(45)
Password VARCHAR(45)
Indexes
ingredients
idingredients INT
ingredientName VARCHAR(45)
idrecipe INT
Indexes
favourites
idFavourites INT
customerId INT
recipeId INT
Indexes
recipe
idrecipe INT
recipeName VARCHAR(45)
mealTime VARCHAR(45)
recipeDescription VARCHAR(2048)
recipeTime INT
recipeDifficulty VARCHAR(45)
recipeServings INT
recipeCalories VARCHAR(45)
dietCategory VARCHAR(45)
recipeInstructions VARCHAR(2048)
recipeCuisine VARCHAR(45)
Indexes
items
idItems INT
idcustomer VARCHAR(45)
itemName VARCHAR(45)
quantity VARCHAR(45)
calories VARCHAR(45)
manufactured date DATE
bestbefore DATE
date bought DATE
Indexes

Appendix 6 – API Working



Appendix 7 – Using GitHub to keep track everyday work



Commits on Jan 25, 2022

Get possible recipes that could be made through the ingredients in th... ...

...e fridge

- detectRecipe algorithm completed
- added the Meal Plan GUI functionality to allow to see the possible recipes as per the ingredients in the fridge
- the fridge ingredients are populated at the start of the login program to allow the user to see the meal plan before clicking the fridge menu button.

Ttan-24 committed on 25 Jan

ac15363

Commits on Jan 2, 2022

Matching ingredients with the recipe

Ttan-24 committed on 2 Jan

bab394a

Commits on Nov 2, 2021

items displayed on table + recipelist match with customeritems ...

- Implemented and modified userdetails and populate table with result set function where the customer items are displayed from the database as per the customerid
- Made an algorithm where it matches the ingredients from the recipe list and the customer items list

Ttan-24 committed on 2 Nov 2021

a226569

Design, Login, sql queries, functions ...

- Fixed the ViewCardLayout and added the functionality where menpanel card and the fridge panel card are interconnected
- Added the Fridge table with scrollpane
- Fixed the Username and password issue by making the queryLogin into a boolean function rather than an arraylist map, modifying the query where the username is equal to the username inputted in the textfield. And checked if the login details inputted are correct
- perform simple test to check if the login panel recognises the logindetails correctly with valid invalid fucntionality
- Created sql queries which would display individula grocery items from the items table in the mysql
- Created userdetails, fridgequery and populateTableWithResultSet. Still has to work on it.

Ttan-24 committed on 2 Nov 2021

a147922

Commits on Oct 15, 2021

Got the basic swing UI set up

Ttan-24 committed on 15 Oct 2021

ed998c2

Commits on Oct 12, 2021

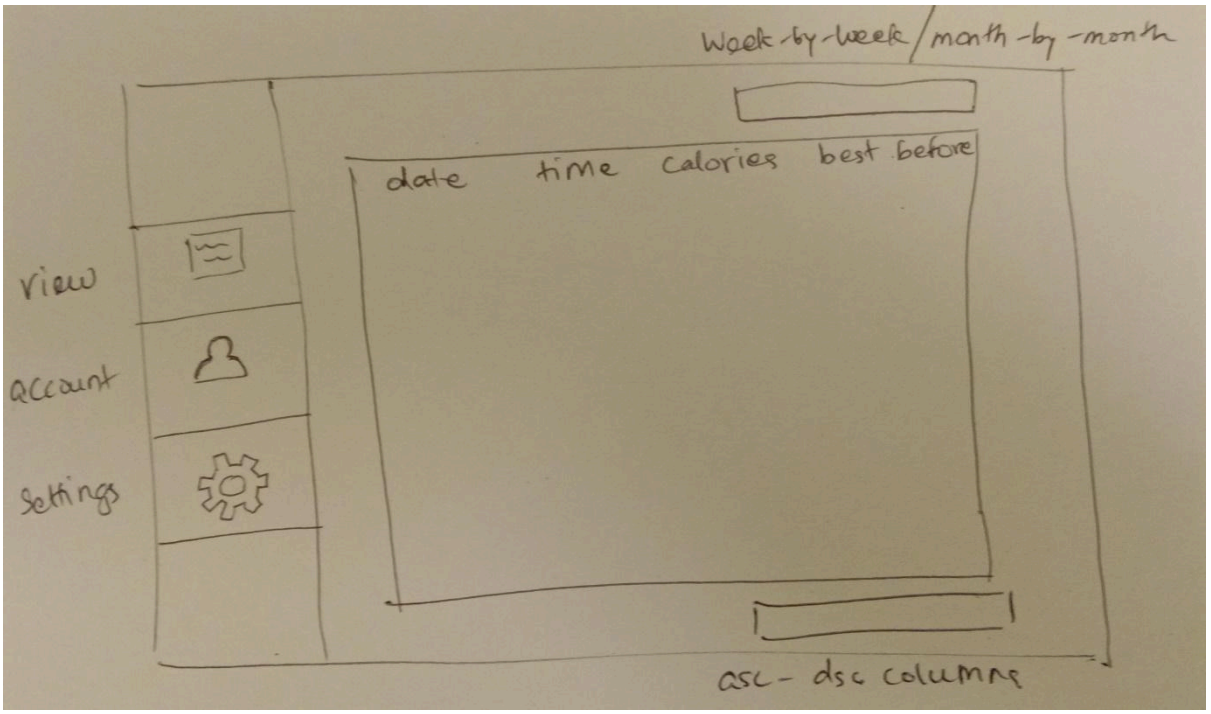
login functionality added + sqlmanager ...

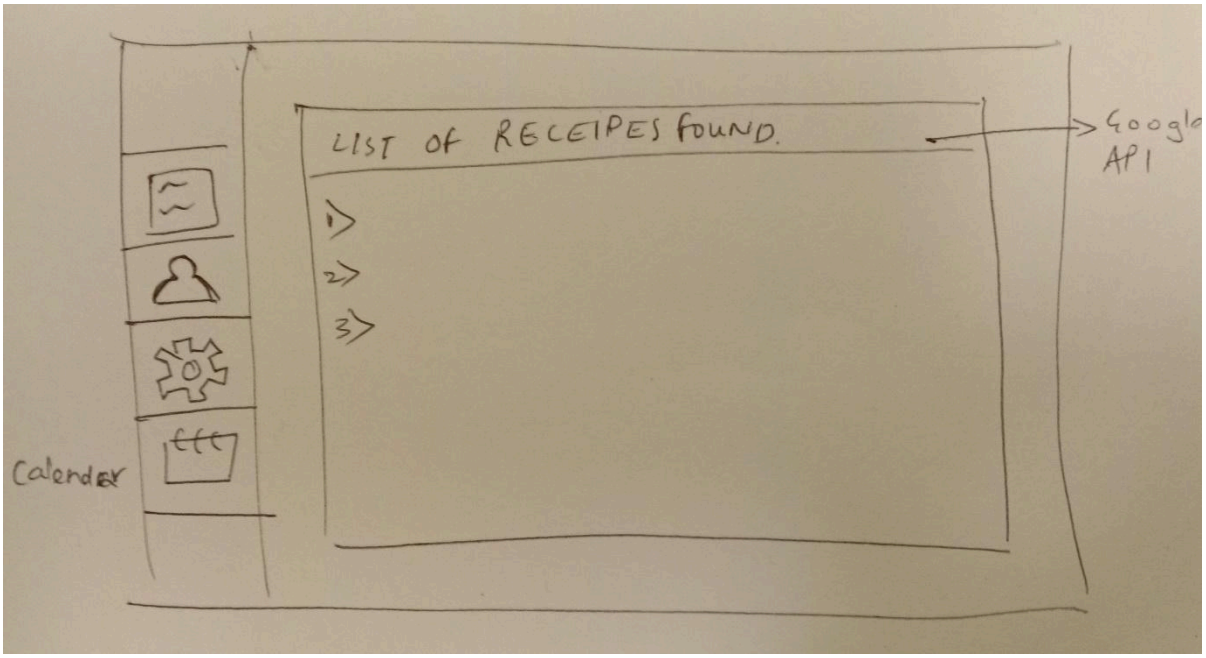
The program connects to the database when run. It allows the user to go to the main panel if the username and password matches with the ones in the database

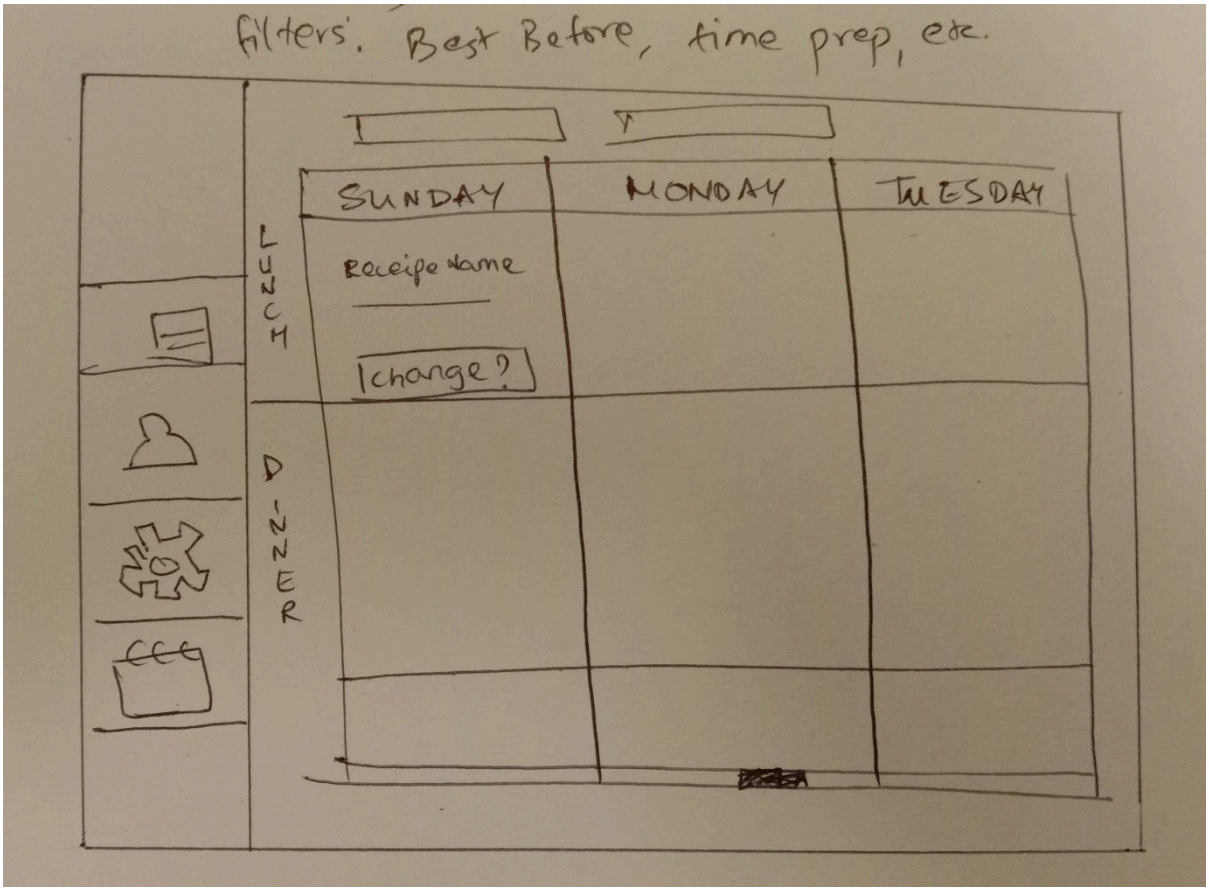
Ttan-24 committed on 12 Oct 2021

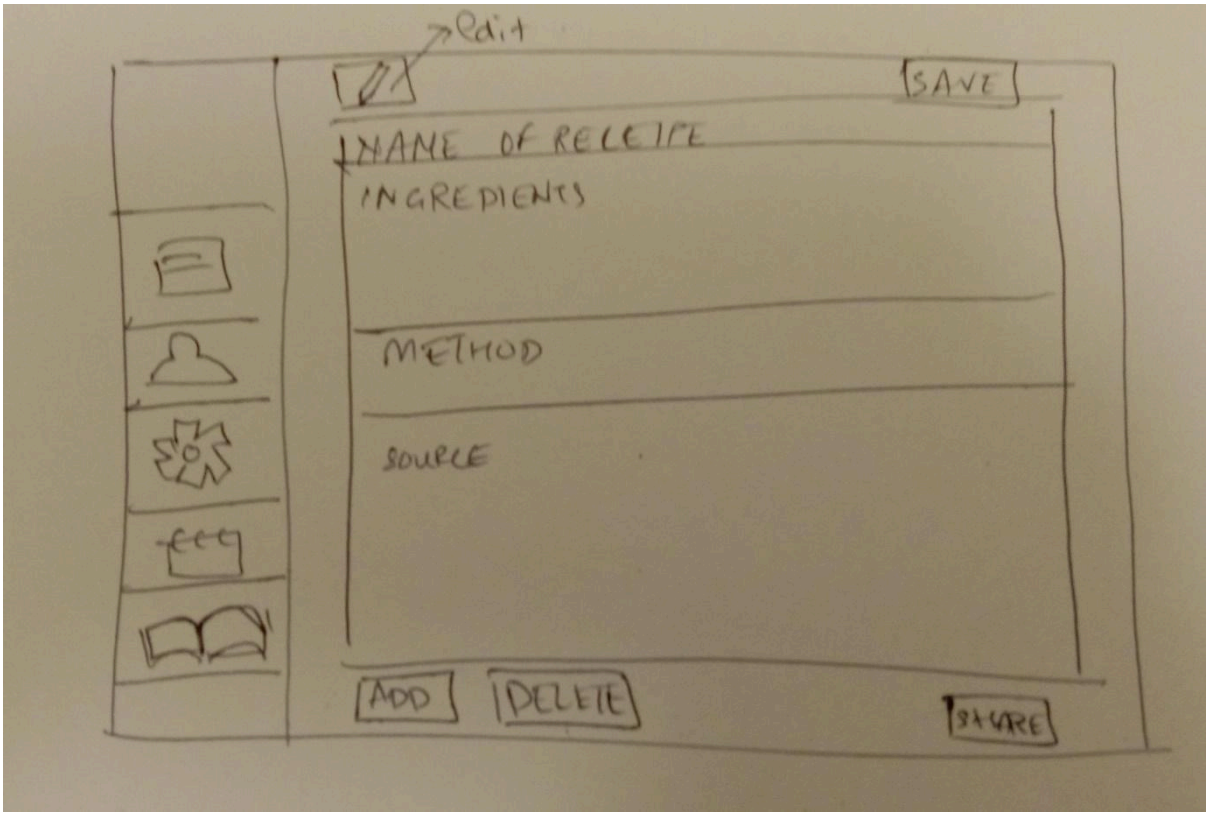
4311406

Appendix 8 – Paper Designs

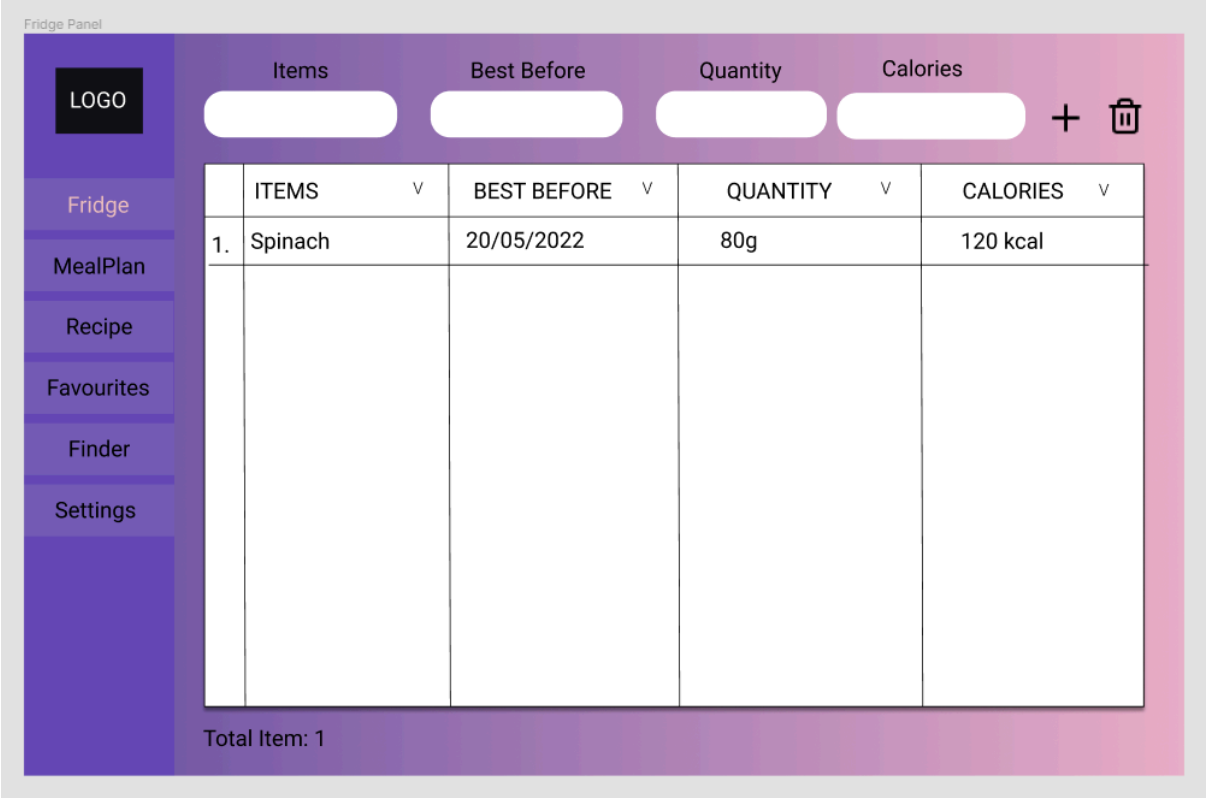


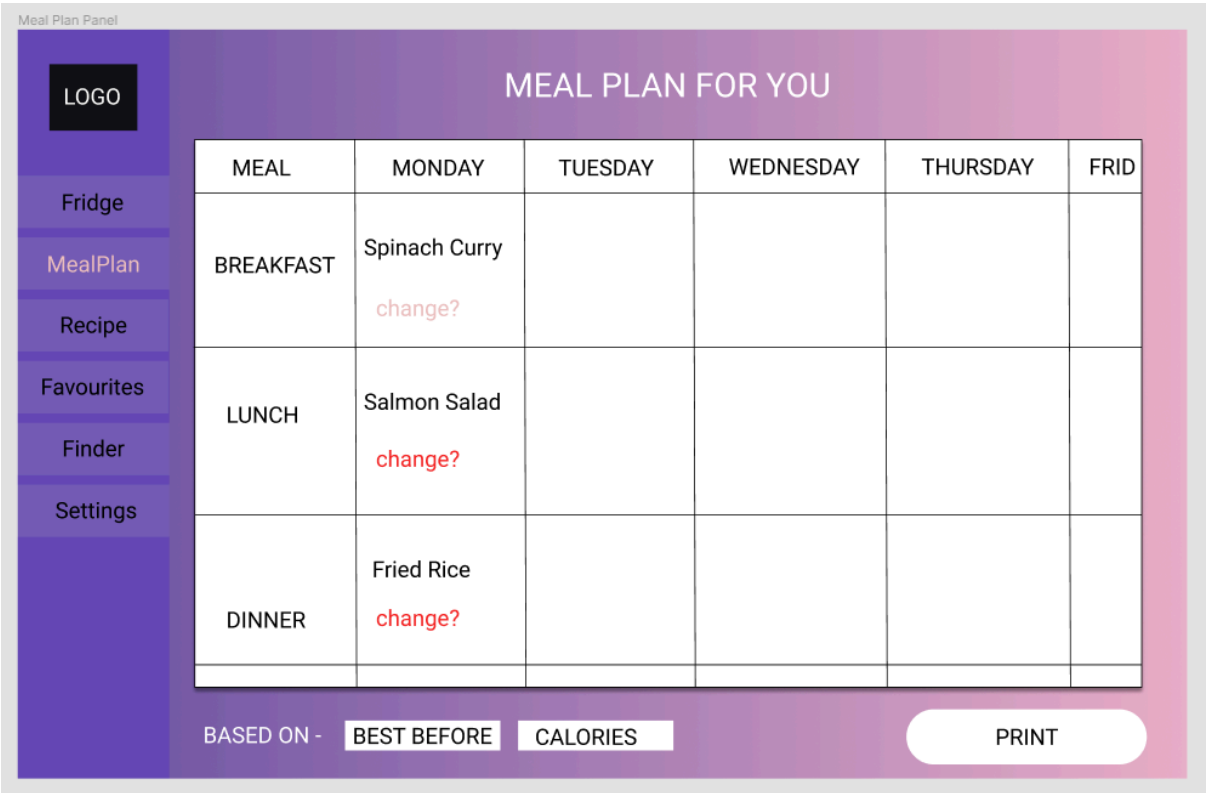



filters. Best Before, time prep, etc.
SUNDAY
MONDAY
TUESDAY
LUNCH
Receipe name
change?
DINNER

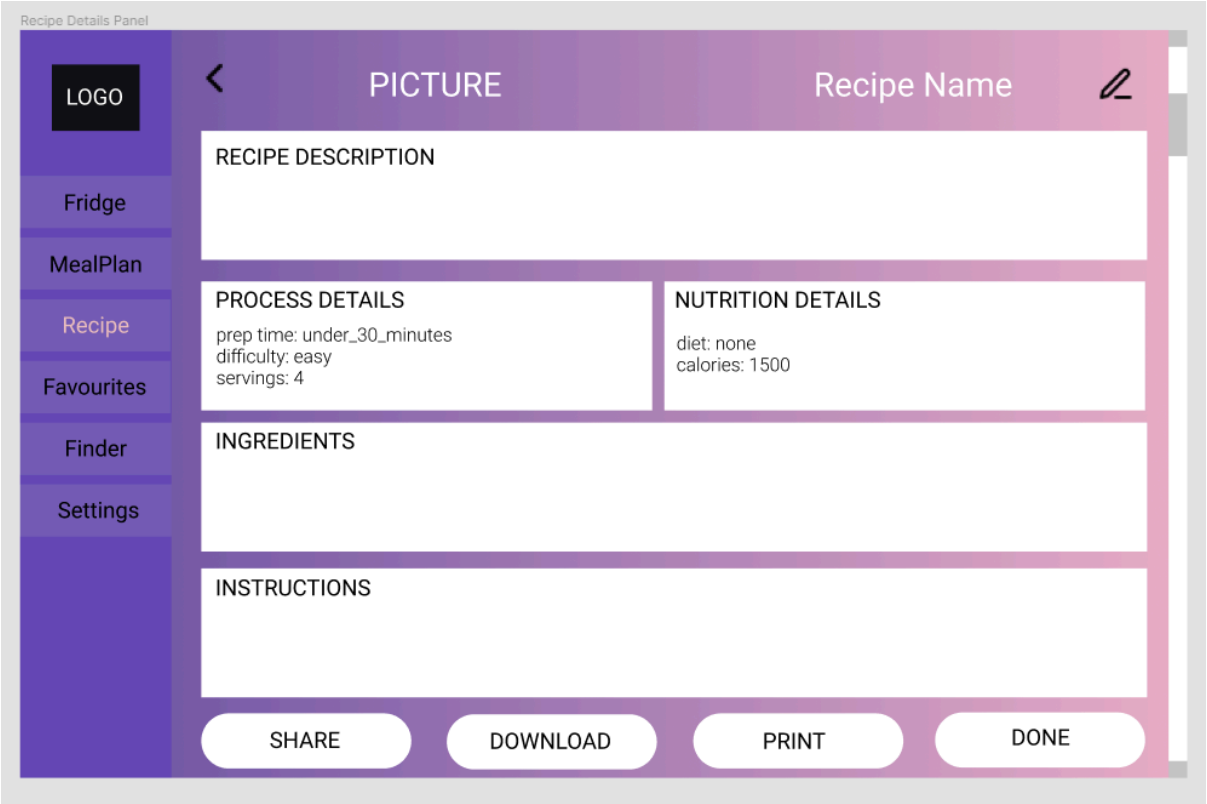

edit
SAVE
NAME OF RECEIPE
INGREDIENTS
METHOD
SOURCE
ADD
DELETE
SHARE

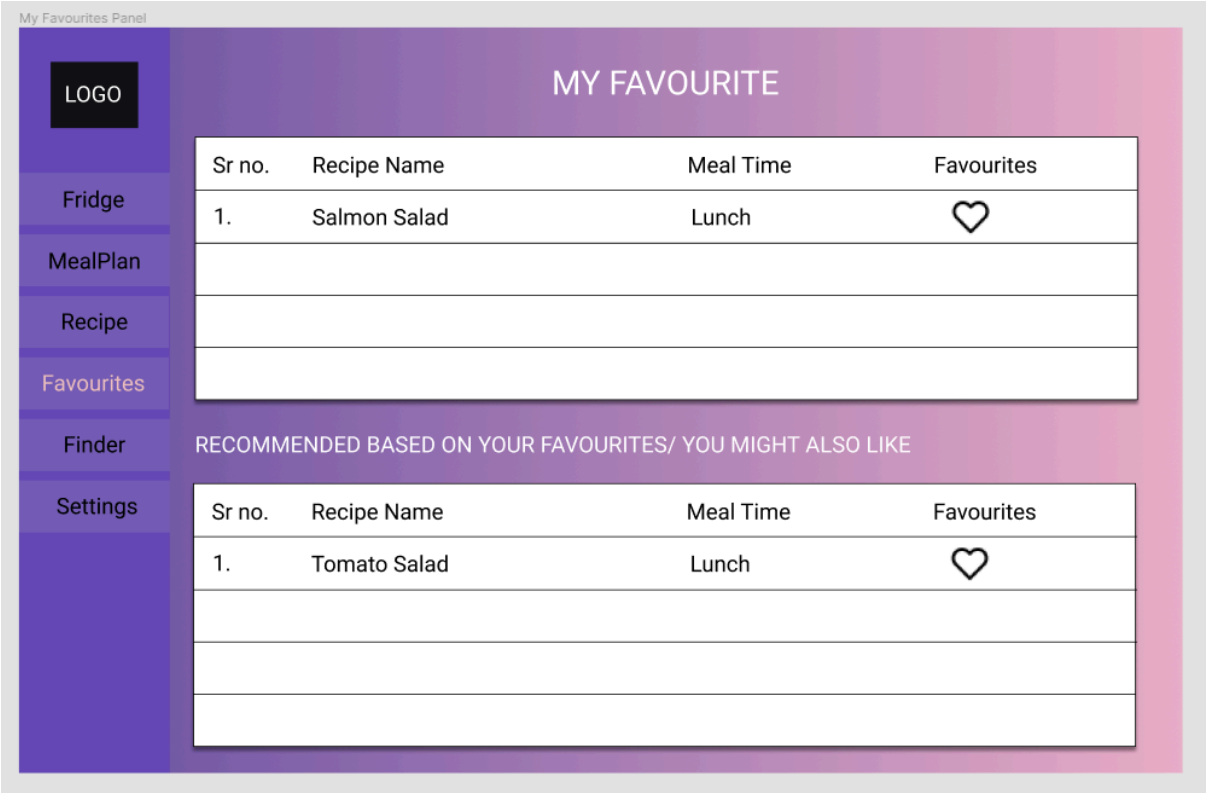
Appendix 9 – Figma Designs

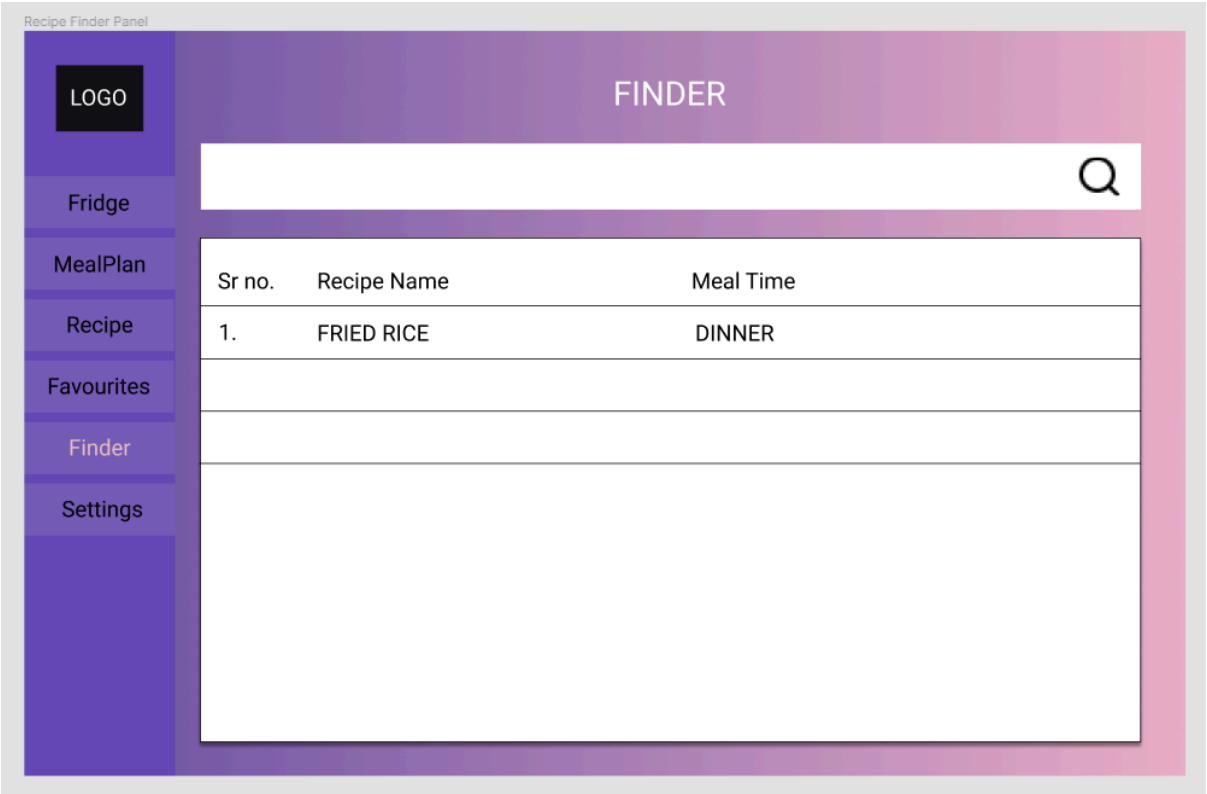


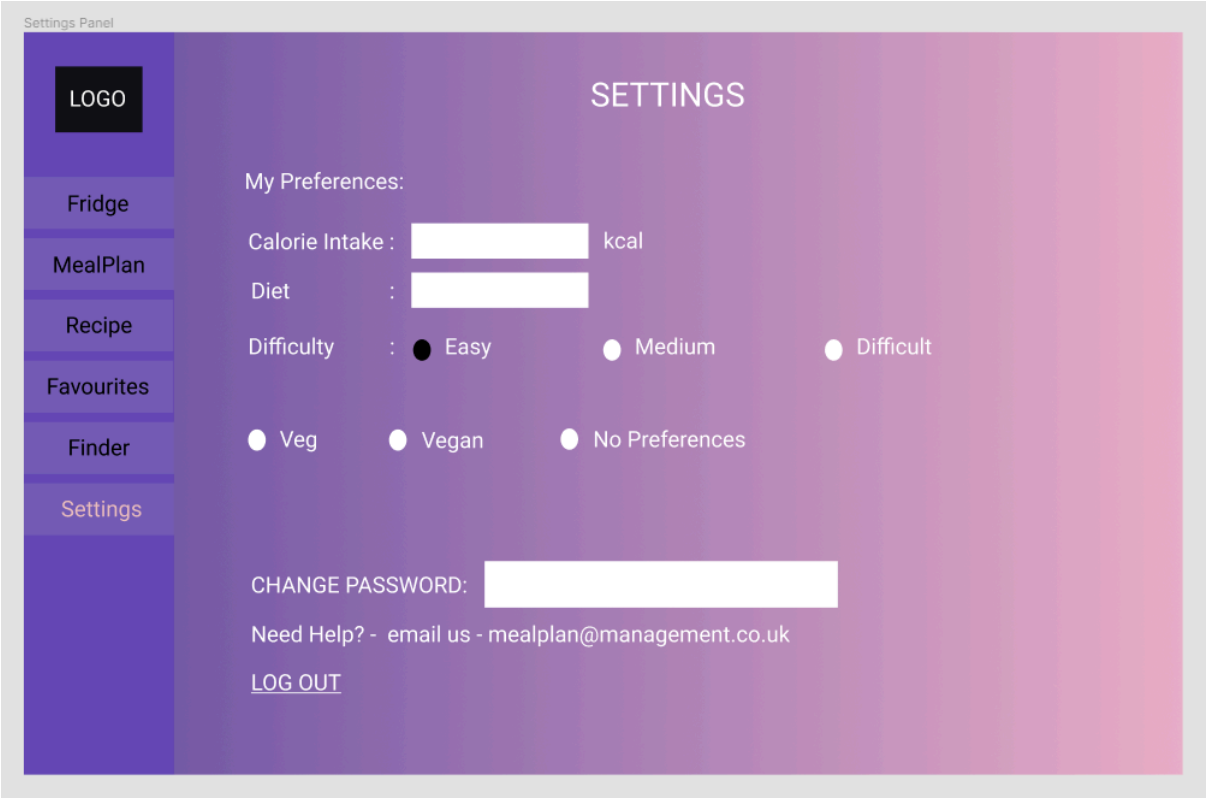



Recipes Panel
LOGO
Fridge
MealPlan
Recipe
Favourites
Finder
Settings
RECIPES
Sr no.
Recipe Name
Meal Time
Favourites
1.
Salmon Salad
Lunch
ADD RECIPE


Recipe Details Panel
LOGO
Fridge
MealPlan
Recipe
Favourites
Finder
Settings
PICTURE
Recipe Name
RECIPE DESCRIPTION
PROCESS DETAILS
prep time: under_30_minutes
difficulty: easy
servings: 4
NUTRITION DETAILS
diet: none
calories: 1500
INGREDIENTS
INSTRUCTIONS
SHARE
DOWNLOAD
PRINT
DONE


My Favourites Panel
LOGO
Fridge
MealPlan
Recipe
Favourites
Finder
Settings
MY FAVOURITE
Sr no.
Recipe Name
Meal Time
Favourites
1.
Salmon Salad
Lunch
RECOMMENDED BASED ON YOUR FAVOURITES/ YOU MIGHT ALSO LIKE
Sr no.
Recipe Name
Meal Time
Favourites
1.
Tomato Salad
Lunch


Recipe Finder Panel
LOGO
Fridge
MealPlan
Recipe
Favourites
Finder
Settings
FINDER
Sr no.
Recipe Name
Meal Time
1.
FRIED RICE
DINNER



My figma link for this project - https://www.figma.com/file/frVJ1tjbhXigNwT5GfBcnj/Prototyping-in-Figma?node-id=641%3A382

Appendix 10 – Inno Setup Script

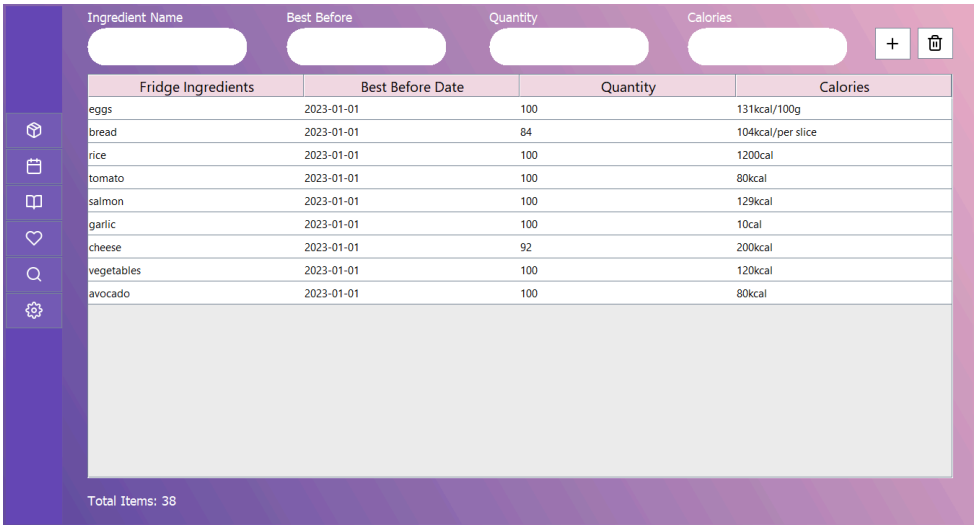
```
; Script generated by the Inno Setup Script Wizard.

#define MyAppName "Meal Management"
#define MyAppVersion "1.0"
#define MyAppPublisher "Tanvi Gavali"
#define MyAppURL "com.uclan"
#define MyAppExeName "MealManagement.exe"

[Setup]
AppId={{8EF3C741-52AD-4E71-A15A-3786D5B91592}
AppName={#MyAppName}
AppVersion={#MyAppVersion}
;AppVerName={#MyAppName} {#MyAppVersion}
AppPublisher={#MyAppPublisher}
AppPublisherURL={#MyAppURL}
AppSupportURL={#MyAppURL}
AppUpdatesURL={#MyAppURL}
DefaultDirName={autopf}\{#MyAppName}
DisableProgramGroupPage=yes
UsedUserAreasWarning=no
PrivilegesRequiredOverridesAllowed=dialog
OutputDir=C:\Users\User\Documents\MealManagement
OutputBaseFilename=Meal_Management_V1.0_Installer
Compression=lzma
SolidCompression=yes
WizardStyle=modern

[Languages]
Name: "english"; MessagesFile: "compiler:Default.isl"

[Tasks]
Name: "desktopicon"; Description: "{cm:CreateDesktopIcon}"; GroupDescription: "{cm:AdditionalIcons}"; Flags: unchecked
Name: "quicklaunchicon"; Description: "{cm:CreateQuickLaunchIcon}"; GroupDescription: "{cm:AdditionalIcons}"; Flags: unchecked; OnlyBelowVersion: 6.1; Check: not IsAdminInstallMode

[Files]
Source: "C:\Program Files\MealManagement\MealManagement.exe"; DestDir: "{app}"; Flags: ignoreversion
Source: "C:\Program Files\MealManagement\*"; DestDir: "{app}"; Flags: ignoreversion recursesubdirs createallsubdirs

[Icons]
Name: "{autoprograms}\{#MyAppName}"; Filename: "{app}\{#MyAppExeName}"
Name: "{autodesktop}\{#MyAppName}"; Filename: "{app}\{#MyAppExeName}"; Tasks: desktopicon
Name: "{userappdata}\Microsoft\Internet Explorer\Quick Launch\{#MyAppName}"; Filename: "{app}\{#MyAppExeName}"; Tasks: quicklaunchicon
```

```
[Run]
Filename: "{app}\{#MyAppExeName}"; Description: "{cm:LaunchProgram,{#StringChange(MyAppName, '&', '&&')}}"; Flags: nowait postinstall skipifsilent
```

Appendix 11 – Disability Testing

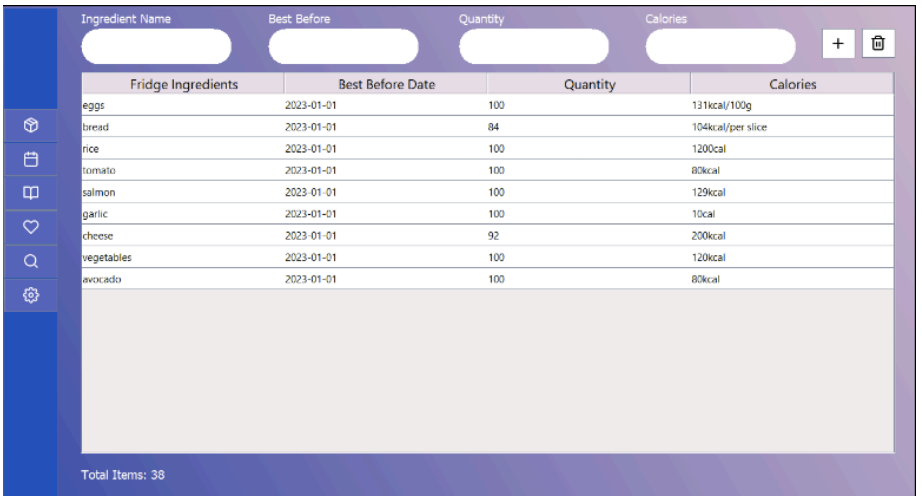
Normal

Colour blind test – Normal

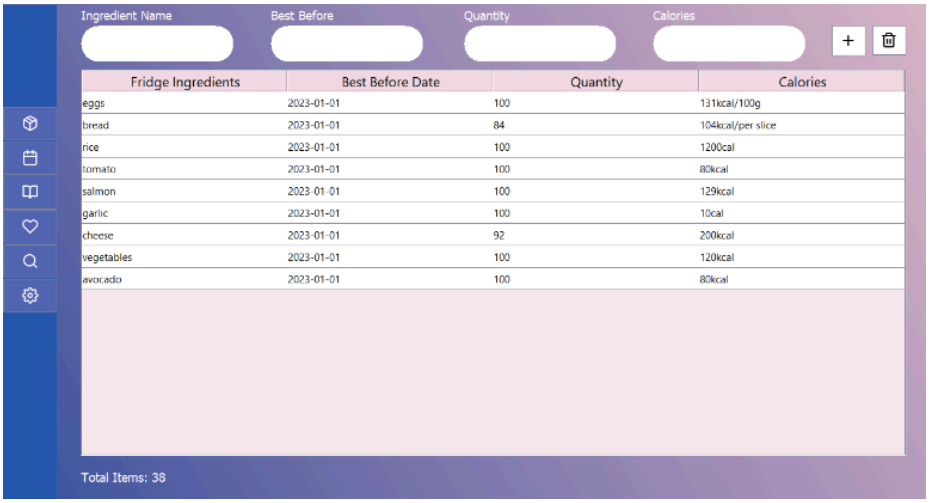


Anomalous Trichromacy

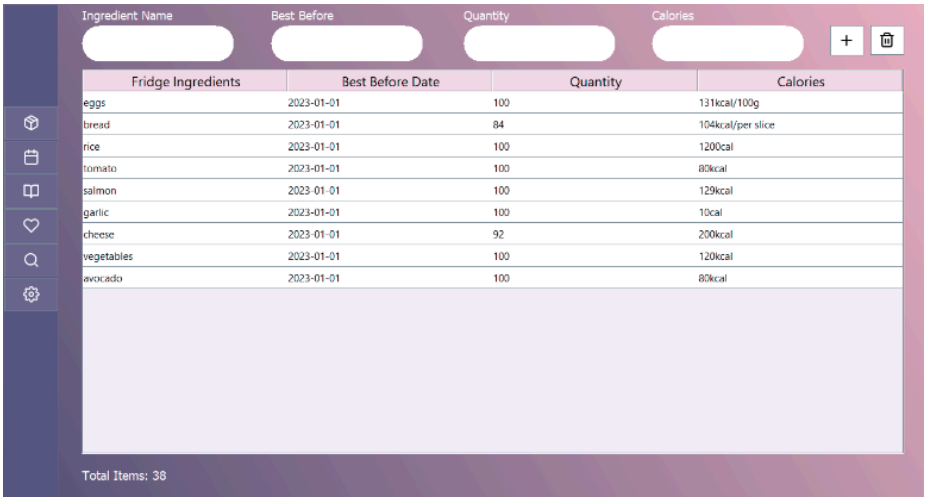
Red-Weak Protanomaly



Green-Weak Deuteranomaly

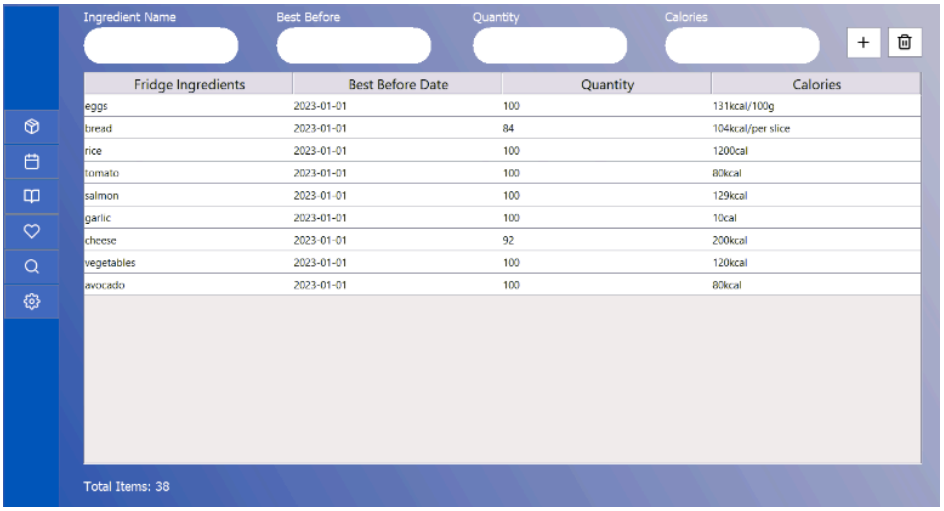


Blue-Weak Tritanomaly

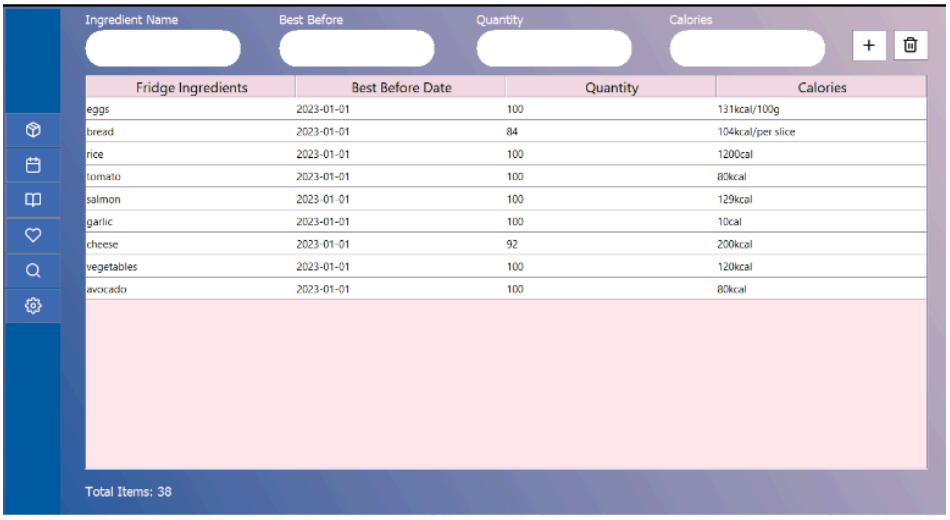


Dichromacy

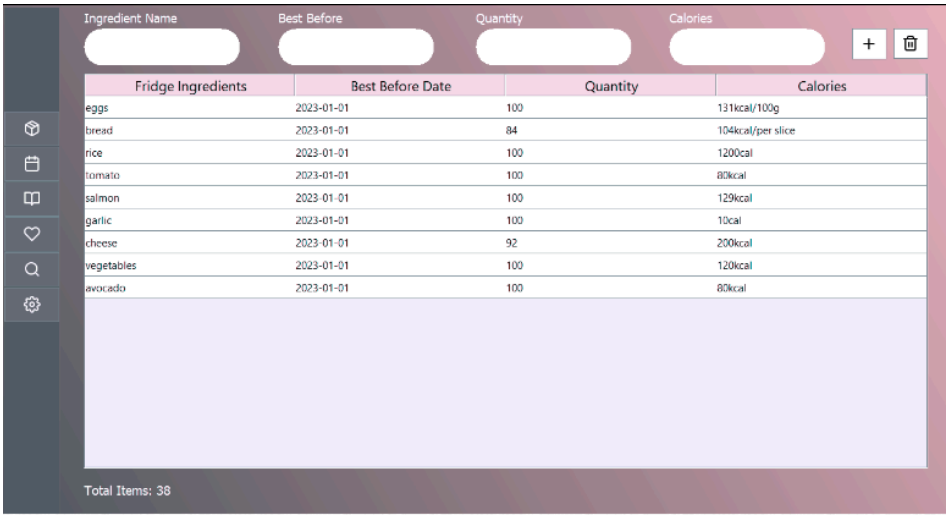
Red-Blind Protanopia



Green-Blind Deuteranopia

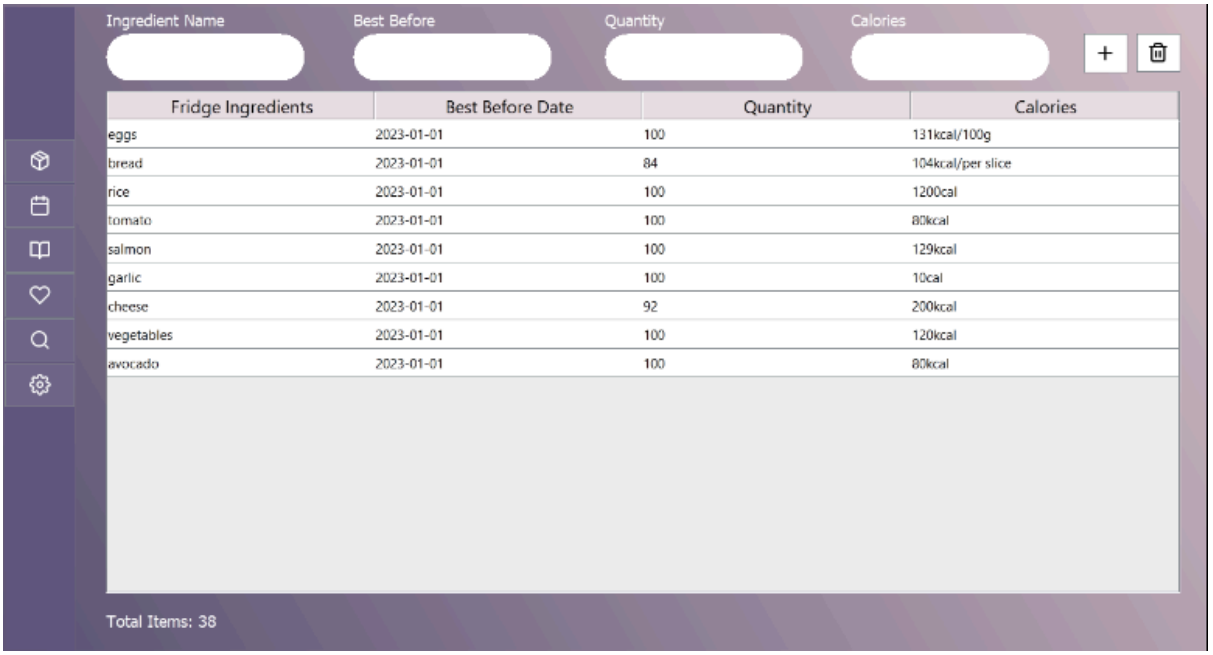


Blue-Blind Tritanopia

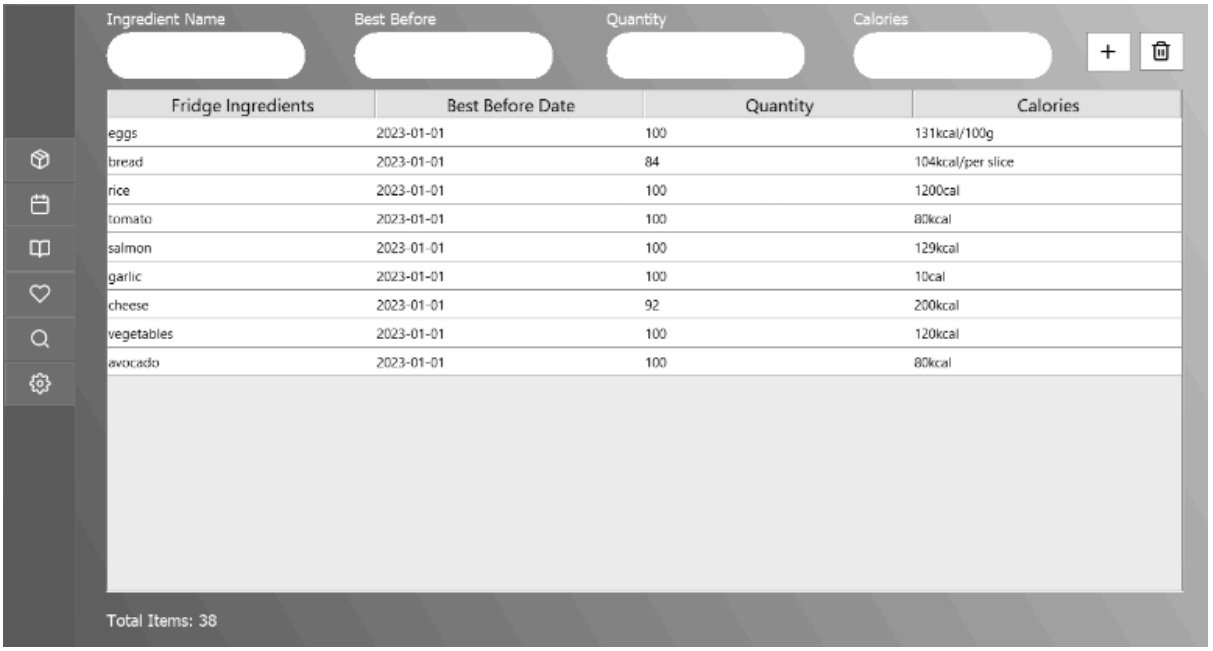


Monochromacy

Blue-Cone Monochromacy



Monochromacy Achromatopsia



Appendix 12 – org.json.JSONObject nested decoding

```
// Get root JSON
String jsonString = response.body();
JSONObject rootJSON = new JSONObject(jsonString);

// Get recipe array
JSONArray recipeArray = rootJSON.getJSONArray("results");

// Iterate through recipe array
if (recipeArray != null) {
	for (int i = 0; i < recipeArray.length(); i++) {
		// Get recipe JSON
		JSONObject recipeJson = recipeArray.getJSONObject(i);

		// Declare details
		String recipeName = "";
		String recipeDescription = "";
		String recipeTime = "0";
		String recipeDietCategory = "";
		String recipeCalories = "0";
		String recipeDifficulty = "";
		String recipeServings = "0";
		String recipeCuisine = "";
		String mealTime = "";

		// Get recipe nutrition
		if (!recipeJson.isNull("nutrition")) {
			JSONObject recipeNutrition = recipeJson.getJSONObject("nutrition");
			if (!recipeNutrition.isNull("calories")) {
				recipeCalories =
String.valueOf(recipeNutrition.getInt("calories"));
			}
		}

		// Get details
		recipeName = recipeJson.getString("name");
		recipeDietCategory = "none";
		if (!recipeJson.isNull("description")) {
			recipeDescription = recipeJson.getString("description");
		}
		if (!recipeJson.isNull("total_time_minutes")) {
			recipeTime =
String.valueOf(recipeJson.getInt("total_time_minutes"));
		}
		recipeDifficulty = "";
		if (!recipeJson.isNull("num_servings")) {
			recipeServings = String.valueOf(recipeJson.getInt("num_servings"));
		}
```

```
                    // Get instructions
                    String recipeInstructions = "";
                    if (!recipeJson.isNull("instructions")) {
                            JSONArray recipeInstructionsJSON =
recipeJson.getJSONArray("instructions");
                            for (int j = 0; j < recipeInstructionsJSON.length(); j++) {
                                    recipeInstructions +=
recipeInstructionsJSON.getJSONObject(j).getString("display_text") + "\n";
                            }
                    }

                    // Format description
                    recipeDescription = removeNonAlphanumeric(recipeDescription);
                    recipeInstructions = removeNonAlphanumeric(recipeInstructions);

                    // Sections - for tags and ingredients
                    if (!recipeJson.isNull("sections")) {
                            // Get tags
                            JSONArray sections = recipeJson.getJSONArray("sections");
                            JSONObject section = sections.getJSONObject(0);
                            JSONArray components = section.getJSONArray("components");
                            JSONArray tags = recipeJson.getJSONArray("tags");

                            // Iterate through tags
                            for (int j = 0; j < tags.length(); j++) {
                                    JSONObject tag = tags.getJSONObject(j);
                                    String tagType = tag.getString("type");
                                    String tagName = tag.getString("name");

                                    // What mealtime? (lunch/breakfast/etc.)
                                    if (tagType.equals("meal")) {
                                            mealTime = tagName;
                                    }

                                    // Difficulty
                                    if (tagType.equals("difficulty")) {
                                            recipeDifficulty = tagName;
                                    }

                                    // Difficulty
                                    if (tagType.equals("cuisine")) {
                                            recipeCuisine = tagName;
                                    }
                            }

                            // Look at each ingredient/component
                            for (int j = 0; j < components.length(); j++) {
                                    // Get ingredient name
                                    JSONObject component = components.getJSONObject(j);
                                    JSONObject ingredient =
component.getJSONObject("ingredient");
```

```
                                        String ingredientName = ingredient.getString("name");

                                        // Make into general ingredient
                                        for (int k = 0; k < generalIngredients.size(); k++) {
                                                String generalIngredient =
generalIngredients.get(k);
                                                if (ingredientName.contains(generalIngredient))
{
                                                        ingredientName = generalIngredient;
                                                }
                                        }

                                        // Add ingredient to SQL database
                                        SQLManager.AddIngredientsFromList(ingredientName,
SQLManager.RecipeID());
                                }
                        }

                        // Add recipe to SQL database
                        SQLManager.AddRecipeQuery(recipeName, mealTime, recipeDescription,
recipeTime, recipeDietCategory,
                                        recipeCalories, recipeDifficulty, recipeServings,
recipeInstructions, recipeCuisine);

                        // Add recipe to arraylist for return
                        Recipe newRecipe = new Recipe();
                        newRecipe.mName = recipeName;
                        newRecipe.mMealTime = mealTime;
                        recipeList.add(newRecipe);
                }
}
```

Appendix 13 – RapidAPI Tasty API response

Java example code:

```
HttpRequest request = HttpRequest.newBuilder()
                    .uri(URI.create("https://tasty.p.rapidapi.com/tags/list"))
                    .header("X-RapidAPI-Host", "tasty.p.rapidapi.com")
                    .header("X-RapidAPI-Key",
"1fe57c4133msh7efab93e804ef92p1e2bb3jsneafe38f2b6dd")
                    .method("GET", HttpRequest.BodyPublishers.noBody())
                    .build();
HttpResponse<String> response = HttpClient.newHttpClient().send(request,
HttpResponse.BodyHandlers.ofString());
System.out.println(response.body());
```

Partial example response:

```
{
        4 items
        "count": 302
        "results": 302 items
        [
        100 items
        0:{
                4 items
                "id": 64447
                "type": "cuisine"
                "name": "british"
                "display_name": "British"
        }
        1:{
        4 items
                "id": 64453
                "type": "cuisine"
                "name": "italian"
                "display_name": "Italian"
        }
        2:{
        4 items
                "name": "mexican"
                "display_name": "Mexican"
                "id": 64457
                "type": "cuisine"
        …
        …
}
```

Appendix 14 – Test Cases

Project Name	Meal Management	**Test Case Set Description**	Functional Testing For Meal Management Components	**Version**	1.1		
Created By	Tanvi Gavali	**Reviewed By**	Tanvi Gavali				
Feature	**Test Case**	**Step**	**Description**	**Status**	**Expected Result**	**Actual Result**	**Comment**
API Calls	Connecting to API	1	Run app in Debug Mode from Eclipse	PASS			
		2	Put breakpoint in APIManager connection	PASS			
		3	Navigate to API page	PASS			
		4	Click connect	PASS	Connection successful		
	Connecting to API with custom query	1	Start app from .exe	PASS			
		2	Navigate to API page	PASS			
		3	Insert custom query for "korean"	PASS			
		4	Run query	PASS	Returned data has filtered results for korean food		
	Saving the API recipes in the database	1	Run app in Debug Mode from Eclipse	PASS			
		2	Put a breakpoint in the response.body string	PASS			
		3	Step over and check the data is parsed in JSON	PASS			
		4	Put breakpoint where the JSON object is put in database	PASS			

		5	Open MySQL store_db database	PASS			
		6	Check the recent data from API in the database	PASS			
		7	Check the data is populated in the app	PASS	Returned data is displayed in the recipe table		
Database	Connecting to database	1	Run app in Perspective Mode from Eclipse	PASS			
		2	Check the console output in Eclipse	PASS	Connection successful		
		3	Check for any exceptions in the log .txt file	PASS	No exceptions output		
	Getting any data from the database	1	Run raw sql query in MySQL Workbench	PASS			
		2	Check the output for that query	PASS			
		3	Run app in Debug Mode from Eclipse	PASS			
		4	Put a break point on the query string and step over	PASS			
		5	Check correct parameter inputted in the query	PASS			
		6	Step over until the return line and check the query string	PASS	Returned data has information from the database		
	Populating the query in a JTable	1	Start app from .exe	PASS			
		2	Navigate to the fridge page	PASS			
		3	Check the fridge table	PASS	Populated the query in a table		
		4	Check for any exceptions in the log .txt file	PASS	No exceptions output		

	Insert the recipe in favourite table	1	Start app from .exe	PASS			
		2	Navigate to the recipe page	PASS			
		3	Check the recipe table	PASS	Populated the query in a table		
		4	Tick the checkbox of one recipe in the table	PASS	Sucessful check		
		5	Navigate to the favourite page	PASS			
		6	Check the favourite table in app with the recent add	PASS	Populated the query in a table and the recent add		
		7	Check in the favourite table in MySQL Workbench	PASS	Table has the checked recipe in the database		
		8	Check for any exceptions in the log .txt file	PASS	No exceptions output		
	Create new recipe	1	Start app from .exe	PASS			
		2	Navigate to the recipe page	PASS			
		3	Click create recipe	PASS			
		4	Add recipe details in the create recipe panel	PASS			
		5	Click add recipe	PASS	Returned text 'Recipe added'		
		6	Click back	PASS			
		7	Check added recipe in the recipe page	PASS	Successfully displays new recipe in the table		
		8	Open MySQL store_db database	PASS			
		9	Check the recent recipe added in the recipe table	PASS	Added data is found in the database		

Login	Check login	1	Run app in Perspective Mode from Eclipse	PASS			
		2	Enter username and password	PASS			
		3	Check the console output in Eclipse	PASS	Successfully logged in the application		
		4	Enter incorrect username/password	PASS	Not Successful with Invalid details text		
Favourite	Similarity score in database	1	Create main user with individual favourite set	PASS			
		2	Create users 1 and 2 with similar items	PASS			
		3	Check favourite recipes of main user from the database	PASS			
		4	Check favourite recipes of user 2 from the database	PASS			
		5	Check favourite recipes of user 3 from the database	PASS			
		6	Run the similar user count raw query in the database	PASS			
		7	Check the similarity score	PASS	Sucessful similarity score of 2 given		
	Similar user code	1	Start app from .exe	PASS			
		2	Navigate to the favourite page	PASS			
		3	Check the 'you might also like' table	PASS			
		4	Compare the recipes taken from the highest score user	PASS	Successful highest score user's recipe		
	Most popular cuisine code	1	Start app from .exe	PASS			

		2	Navigate to the favourite page	PASS			
		3	Check the 'you might also like' table	PASS			
		4	Navigate to the recipe page	PASS			
		5	Check box similar cuisine recipes	PASS			
		6	Navigate back to the favourite page	PASS			
		7	Check box 'you might also like' table	PASS	Returned similar cuisine recipes		
Meal Plan	Generate meal plan	1	Start app from .exe	PASS			
		2	Navigate to the meal plan page	PASS			
		3	Click generate meal plan	PASS	Successfully dislpayd the meal plan table		
		4	Click save meal plan	PASS	Successfully saved the meal plan table		
		5	Click finish week	PASS	Successfully removed the meal plan table created previously		
	Check quantity	1	Start app from .exe	PASS			
		2	Navigate to the fridge page	PASS			
		3	Check item quantity	PASS			
		4	Navigate to the meal plan page	PASS			
		5	Click generate meal plan	PASS			
		6	Click save meal plan	PASS			
		7	Navigate to the fridge page again	PASS			
		8	Check item quantity	PASS	Sucessfully removed the quantity of the item		

		9	Navigate to the meal plan page again	PASS			
		10	Click finish week	PASS			
		11	Click generate meal plan	PASS	Returned text 'not enough meal ingredients'		
Settings	User preference	1	Start app from .exe	PASS			
		2	Navigate to the settings page	PASS			
		3	Change the calorie intake	PASS			
		4	Change the cook time max	PASS			
		5	Change the difficulty option	PASS			
		6	Save preferences	PASS	Returned text 'saved preferences '		
		7	Navigate to the meal plan page	PASS			
		8	Click generate meal plan	PASS			
		9	Double click the recipe in the meal plan	PASS			
		10	Check recipe detail panel displays	PASS			
		11	Check preferences matches with the recipe details	PASS	Sucessful generation of meal plan with preferences		
	Update Password	1	Start app from .exe	PASS			
		2	Navigate to the settings page	PASS			
		3	Insert updated password	PASS			
		4	Log out	PASS			
		5	Start app from .exe again	PASS			
		6	Insert old password	PASS	Returned text 'Invalid details'		

		7	Insert updated password	PASS	Successfully logged in the application		