Electronics and Computer Science
Faculty of Physical Sciences and Engineering
University of Southampton

Fedor Selenskiy
May 2020

Use of Machine Learning to Improve Eating Habits

Project supervisor: Dr Heather Packer
Second examiner: Dr Paolo Rapisarda
A progress report submitted for the award of BSc Computer Science

1 Abstract

In this project, I created a cloud based Android application that estimates a user's Healthy Eating Index score and uses machine learning techniques to recommend healthy food items maximising their HEI score, that the user is likely to enjoy.

2 Statement of Originality

- I have read and understood the ECS Academic Integrity information and the University's Academic Integrity Guidance for Students.
- I am aware that failure to act in accordance with the Regulations Governing Academic Integrity may lead to the imposition of penalties which, for the most serious cases, may include termination of programme.
- I consent to the University copying and distributing any or all of my work in any form and using third parties (who may be based outside the EU/EEA) to verify whether my work contains plagiarised material, and for quality assurance purposes.

1. I have acknowledged all sources, and identified any content taken from elsewhere.
2. I have not used any resources produced by anyone else
3. I did all the work myself, or with my allocated group, and have not helped anyone else.
4. The material in the report is genuine, and I have included all my data/code/designs.
5. I have not submitted any part of this work for another assessment.
6. My work did not involve human participants, their cells or data, or animals.

3 Acknowledgements

I would like to thank my supervisor, Dr Heather Packer, for guiding me through the project. I would also like to thank my father, Dr Evgeny Selensky, for insightful comments on the technical aspect of the project, and my older brother, Nick Selensky, for helpful comments regarding the user interface of the application.

Contents

4 Acronyms

- **EM**: Expectation Maximisation
- **UI**: User Interface
- **API**: Application Programming Interface
- **DAO**: Data Access Object
- **DGA**: Dietary Guidelines for Americans
- **GMM**: Gaussian Mixture Model
- **HEI**: Healthy Eating Index
- **URI**: Universal Resource Identifier
- **URL**: Universal Resources Locator
- **FPED**: Food Patterns Equivalents Database
- **REST**: Representational State Transfer
- **USDA**: United States Department of Agriculture
- **MVVM**: Model-View-ViewModel design architecture
- **JSON**: JavaScript Object Notation
- **HTTP**: Hypertext Transfer Protocol
- **HTTPS**: Hypertext Transfer Protocol Secure
- **WWEIA**: What We Eat In America
- **FNDDS**: Food and Nutrition Database for Dietary Studies

5 Introduction

5.1 Problem

There exists a wide variety of health and fitness applications. A lot of the food tracking applications, such as MyFitnessPal, serve as food diaries and progress trackers. Such applications typically show the user the total amount of calories they consumed, as well as providing a nutritional breakdown of logged food items. This leaves the interpretation of the collected data to the user.

A user with a lack of experience may struggle to effectively utilise such data. The user may begin to question whether they are eating too much or too little, whether they are eating the right foods, and whether their diet is actually healthy. I aim to create an Android application that will aid an individual in gaining a deeper understanding of the quality of their diet by estimating the Healthy Eating Index (HEI) of the set of foods consumed in a day, as well as make personalised recommendations on food items that will increase their HEI score.

5.2 Goals

This application should:

1. Enable the user to keep a food diary, where the user inputs the food items they consumed.
2. Estimate HEI based on the food input.
3. Use a machine learning algorithm to build a recommender system that would suggest food items the user might like that will increase their HEI.
4. Present the HEI score in a radar graph in order to help the user gain a better understanding of their diet.

5.3 Scope

The datasets available to me are relatively small and not detailed. For example, portion sizes of the food items are required to calculate HEI. Such information was missing, thus assumptions and approximates had to be made. Similar techniques and methods could be applied to larger and more detailed datasets, which would eliminate the need for such approximations, and result in more accurate HEI estimates. Using a bigger dataset would result in more personalised recommendations.

6 Literature Review

6.1 Background Research

6.1.1 Healthy Eating Index

Many people may not even be aware that they are consuming too much or too little of a specific food category, as they may not be aware of the dietary guidelines. This may lead to undesirable effects, for example not eating enough fruits and vegetables may lead to vitamin deficiency, or excessive consumption of added sugars can lead to weight gain. Easy access to the HEI score of an individual may help raise awareness of their consumption of certain food groups, and hopefully encourage them to become healthier.

HEI [10] is a density-based measure for assessing the dietary quality by measuring the degree to which a set of foods aligns with the Dietary Guidelines for Americans (DGA). Food items are scored based on the pertinent densities of its constituents, which fall into one of 13 categories, referred to here as HEI components.

The average score of the 13 components represents the overall diet quality, while individual component scores can help reveal strengths weaknesses of the diet. HEI does not account for the amount of food consumed, as all components represent amounts per 1000 kcal (with the exception of Fatty Acid Ration component, which is a ratio). See Table 1 for HEI 2015 scoring standard.

Component	Standard for Max Points	Standard for Min Points
Total Fruit	$\geq$ 0.8 c equivalents (5 points)	No fruit
Whole Fruit	$\geq$ 0.4 c equivalents (5 points)	No whole fruit
Total Vegetables	$\geq$ 1.1 c equivalents (5 points)	No vegetables
Greens & Beans	$\geq$ 0.2 c equivalents (5 points)	No dark green vegetables or legumes
Whole Grains	$\geq$ 1.5 c equivalents (10 points)	No whole grains
Dairy	$\geq$ 1.3 c equivalents (10 points)	No dairy
Total Protein Foods	$\geq$ 2.5 c equivalents (5 points)	No protein foods
Seafood & plant proteins	$\geq$ 0.8 c equivalents (5 points)	No seafood or plant proteins
Fatty Acid Ratio †	(PUFAs + MUFAs)/SFAs $\geq$ 2.5 (10 points)	(PUFAs + MUFAs)/SFAs $\leq$ 1.2
Refined Grains	$\leq$ 1.8 oz equivalents (10 points)	$\geq$ 4.3 oz equivalents
Sodium	$\leq$ 1.1 gram (10 points)	$\geq$ 2.0 grams
Added Sugars	$\leq$ 6.5% of energy (10 points)	$\geq$ 26% of energy
Saturated Fats	$\leq$ 8% of energy (10 points)	$\geq$ 16% of energy

Table 1: Scoring Standards for HEI 2015 (adapted from source [10])

† Polyunsaturated Fatty Acids (PUFA), Monounsaturated Fatty Acids (MUFA), Saturated Fatty Acids (SFA).

The basics steps in calculating HEI are:

1. Acquiring a set of foods to consider.
2. Determining the total amount of each dietary constituent across the set.
3. Deriving pertinent densities of each dietary constituent and scoring against the HEI standard.

6.1.2 Side Effect of Diets

Diets focusing on macronutrient intake may affect the micronutrient intake of an individual. A study on the micronutrient quality of weight-loss diets [11] has shown that a significant proportion of the participants following the Ornish diet have shifted to intakes associated with risk of inadequacy of vitamin E and B-12 and zinc. On the other hand, the risk of inadequacy of vitamins A, E, K and C was found to decrease in participants following the Zone diet.

6.1.3 Satiety and Satiation

A study on the relationship between regular eating adherence and binge eating frequency [9] shows that a higher number of regular eating adherent days is has a high positive correlation with lower binge eating frequency. This provides evidence to suggest that regular satiety and satiation can reduce food craving.

Many factors can influence satiation, which ends a meal, and satiety, which determines the time between meals. Macronutrient intake plays an important role in both satiation and satiety. Protein is the most effective macronutrient in prolonging the feeling of fullness [7]. Additionally, an increase in protein consumption has been shown to increase diet-induced thermogenesis [7], which contributes to 10% of daily energy expenditure [8].

Food volume contributes to satiation more than energy content. Therefore reducing the energy density of the food, or consumption of food with a lower energy density, is a way to promote satiation [7].

6.1.4 Food Craving

Modern temptations to eat are stronger than in the past [3]. Despite many efforts to increase health awareness, from social media influence to regulations on food labelling [14], we still fall victim to craving junk food.

Different environmental factors can affect food craving. A study on the effects of the time of day on momentary hunger and food craving [4] has shown that there is a high coherence between hunger and craving, and that desirability of a food group is affected by the time of day. An individual's mood or exposure to food-related cues are potentially the biggest triggers of craving [13], which means that it may prove difficult for an individual to be rid of cravings completely.

In extreme cases, food craving could be the result of brain reward circuit dysfunction. That may be so if the motivation for food consumption begins to occur without hedonic liking of the food, which are usually connected processes [3]. Conversely, reward circuit dysfunction could be the result of excessive food consumption. More of the pleasurable food has to be consumed to trigger the same level of reward circuit activation, which has built up a tolerance due to overeating [3].

6.1.5 Summary of Background Research

Understanding the cause of food craving is important in overcoming it. Suggesting to reduce the time between meals to an individual to reduce hunger could also help them reduce cravings. Conversely, suggesting a planned cheat meal to an individual with cravings could activate the brain reward circuits, thus satisfying the cravings. Estimating the HEI scores of the individual categories can clarify what foods categories the user is scoring less in. Foods from those categories maximising satiation and satiety can then be suggested.

6.2 Technical Research

6.2.1 Android Application Overview

Android applications are built from the following components which are entry points through which the system or the user can enter the application [15]:

- **Activity**: entry point for the user which contains the user interface.
- **Service**: entry point for running the application in the background, which does not require a user interface.
- **Broadcast receiver**: component enabling the event delivery to the application outside of use, and reception of broadcast announcements.
- **Content provider**: manages application data and its accessibility by other applications.

The main building blocks of an Android application include the following:

- **View**: represents a visual component in the application [17]. All visual components extend this class. In order for the user to be able to interact with a View, it must implement event handling and drawing.
- **Fragment**: represents a collection of related Views [18]. Combination of Fragments can be used to construct a dynamic UI inside an Activity.
- **ViewModel**: serves as an abstraction layer between the Activities or Fragments and the data [19], also storing UI data. In order to ensure high cohesion and low coupling, communication amongst Fragments and between Fragments and Activities is achieved via the ViewModel [16]
- **LiveData**: life-cycle aware, observable data holder class [20]. A LiveData object can be observed for changes, and those changes can be acted upon accordingly. Life-cycle awareness ensures that when the data changes, only active observers are notified.
- **Room Persistence Library**: architectural components serving as an abstraction layer over SQLite [21]. An Entity class describes the structure of a table in a RoomDatabase class. A DAO interface defines the SQLite queries that Room uses to automatically generate Entity objects, which represent specific entries in the table. Values can be accessed with defined getter methods from the Entity object.

6.2.2 Android Application Architecture

Android applications are not required to adhere to a specific architecture. However, a good design architecture could boost development time, improve the maintainability, extensibility, and performance of the application [6].

Android developers at Google advise to decouple the UI from the data by using a Model-View-Viewmodel (MVVM) [22] architecture when writing an Android application [16]. MVVM architectural components serve the following purpose:

- **Model**: representation of the data. Room persistence library [21] is an example of Model representation in Android.
- **View**: all the visual components, what the user actually sees. This is represented by classes such as View, Activity and Fragment.
- **Viewmodel**: intermediary between the raw data and the view, acting as a layer of abstraction. This allows the view to be developed independently of the model. ViewModel class in Android is an implementation of this structure.

Separation of the view from the model helps maintain good user experience, by minimising the workload of the view. This is especially important for low powered devices running Android.

6.2.3 Machine Learning Algorithms

Machine learning is a subset of artificial intelligence that can learn patterns from data without having to define them a priori. Categories of machine learning algorithms include [5]:

- **Supervised machine learning** is used to discover mapping function for input and corresponding output dataset, which can be used for decision making, such as prediction and classification (putting data in predefined classes).
- **Unsupervised machine learning** is used to capture the relationship, or the correlation, among input data, which can be used for clustering (the process of grouping similar data with no predefined classes).
- **Semisupervised machine learning** is a multistage process in which one learning type is used as preprocessing for the other.
- **Reinforcement learning** algorithms learn behaviour based on reward mechanisms.

The data available to me limits me to the use of unsupervised machine learning algorithms, as the MyFitnessPal dataset [23] I am going to be working with contains practically raw data with minimal processing.

K-Means clustering is an unsupervised machine learning algorithm which aims to group data points into 'k' clusters, where the value of 'k' is defined *a priori* [38]. This iterative process is initialised by randomly generating 'k' data points, which are called centroids. At every consecutive iteration, every data point is assigned to the closest centroid. The mean value of all the data points assigned to a centroid is calculated, and the centroid is shifted to that mean position. This process is repeated until the centroids stop shifting between iterations. K-Means is fast as it has linear time complexity, but fails when clusters are not circular or their means are close together, which is a result of using the mean as a cluster centre.

Expectation Maximisation (EM) is a more general approach [38]. EM algorithms aim to find parameters to statistical models of clusters in a way that maximises the likelihood of data points belonging to those clusters. EM algorithms allow different assumptions about the shapes of the clusters to be made. Similarly to K-Means, the number of clusters has to be chosen *a priori*, and the parameters of each cluster are chosen randomly. Each iteration consists of two steps. The first 'E' step generates the expectation function for each cluster. The second 'M' step finds parameters that maximise the expectation [40]. Gaussian Mixture Model (GMM) is a statistical model where the data is assumed to be derived from multiple normal distributions [39]. Modelling data with GMM and using an EM algorithm can be used to cluster data. GMM offers greater flexibility in the shape of the clusters, and supports mixed membership.

6.2.4 Cloud Computing

'Cloud computing is a model for enabling convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction.' [12]

Important components of cloud computing include [12]:

- On-demand self-service: consumers can gain access to resources automatically.
- Network access: computing resources are accessed through the internet.
- Resource pooling: computing resources are assigned to users based on demand

6.2.5 HTTP

HTTP is a stateless application layer protocol used for communication over the World Wide Web. Stateless means that every HTTP request must be complete and not use any state information. An HTTP URL consists of three main parts. For example *"http://example.com:80/dir1/dir2/file?a=1b=2"*:

- **Host**: example.com:80 points to a host located at the IP address associated with example.com on port 80.
- **Path**: the requested resource is located in the path "/dir1/dir2/file".
- **Query**: the query, separated from the path by a question mark, contains any required parameters separated by an ampersand.

The two fundamental HTTP requests are GET and POST, both of which are used to retrieve data. The key difference between the two is that GET puts the query parameters in the URL, whereas POST places the query parameters in the message body. However, it is important to mention that neither GET nor POST HTTP requests are secure. HTTPS should be used for secure communication as it uses encryption.

Offloading computationally intensive tasks, such as HEI calculations and generation of recommendations, to a server would make the Android application as fast and as lightweight as possible. The server would act as the cloud, by providing computing resources and remote storage capabilities. HTTP protocol would mediate the communication between the cloud and the Android application, with GET and POST requests being used

6.2.6 String Comparison Metrics

String matching algorithms are designed to find a target string in a source document. Identifying food items in the user input would require an implementation of a string matching algorithm. A string comparison metric is required to evaluate the similarity of two strings in order to do string matching. String similarity metrics I am going to be using in this project are:

- **Levenshtein distance**: otherwise known as edit distance, this metric describes the smallest amount of edit operations required to transform one string into another [32]. An edit operation is the insertion, deletion or substitution of a single character.
- **Jaccard index**: otherwise known as intersection over union, this metric describes the similarity of two sets. It is calculated by computing the number of elements in the intersection of two sets divided by the size of the union of those two sets. This could be very useful for comparing the similarity of sentences rather than just individual words, however this would still require a way of comparing individual words [33] [34]. Given sets of words A and B, the Jaccard index of A and B is the following:

$$Jaccard(A, B) = \frac{A \cap B}{A \cup B}$$

6.2.7 String Matching

Wagner-Fischer algorithm computes the Levenshtein distance between two strings. It has time and space complexity of O(nm), where 'n' and 'm' are the lengths of the strings being compared [36]. This algorithm is sensitive to word order, however it is straightforward to implement and to use, as strings with a lower edit distance are more likely to be a match than those with a higher edit distance.

Jaccard index of two strings can be used in string matching. The similarity of words could be measured using an edit distance algorithm, such as Wagner-Fischer, and words could be considered similar if their edit distance is below a certain threshold. The number of similar words can then be divided by the size of the set of words in both strings.

Mongue-Elkan algorithm is used to match sentences by computing the average similarity of closest words from each string, giving it time complexity of O(nm), where 'n' and 'm' are the number of words in each string [37]. This algorithm also requires the use of a similarity metric for determining closest word pairs.

By definition, this algorithm is less sensitive to word order, thus making it more desirable for real-world applications. Given a similarity metric 'sim' and sets of words A,B, Mongue-Elkan measure is the following [37]:

$$Mongue - Elkan(A, B) = \frac{1}{|A|} \sum_{i=1}^{|A|} max\{sim(a_i, b_j)\}_{j=1}^{|B|}$$

Sergio Jimenez et. al. [37] propose a modification to the Mongue-Elkan algorithm that will promote words with higher similarity, as they are assumed to be more informative, using generalised mean:

$$GeneralisedMongue - Elkan(A, B) = (\frac{1}{|A|} \sum_{i=1}^{|A|} (max\{sim(a_i, b_j)\}_{j=1}^{|B|})^m)^{\frac{1}{m}}$$

The value of 'm' represents the degree to which higher similarity words are prioritised.

6.2.8 Recommender Systems

A recommender system is a personalised information filtering system for providing useful item suggestions [35]. Types of recommender systems include:

- **Content based**: this type of recommender system aims to suggest items with similar features to items that the user previously liked.
- **Knowledge-based**: this type of recommender system utilises domain knowledge to suggest items with features that would meet the user's needs.
- **Collaborative filtering**: this type of recommender system suggests items that similar users liked in the past.
- **Hybrid**: this type of recommender system uses a combination of techniques.

With the currently available to me resources, it is incredibly difficult to build a content or knowledge-based recommender system. Luckily, the nature of the MyFitnessPal dataset [23] would make it possible to build user models for 9'800 real-world users. As a result, a collaborative filtering recommender system would be the most suitable.

My approach is to generate feature vectors that represent users, and group users by clustering their corresponding feature vectors using a clustering algorithm. The clusters will contain groups of users with similar diets, who are therefore likely have similar food tastes. Based on this implicit collaborative filtering, a set of foods that any particular user is likely to find enjoyable can be generated by looking at what other users from the same cluster have consumed in the past. Foods with the highest HEI score can then used as recommendations for the user of interest.

Unsupervised machine learning algorithms are the most suitable for the available datasets, which contain practically raw data. In order to make the recommender system fast, an algorithm with a fast time complexity is desirable. However, scalability may be an issue.

I talked about K-Means clustering algorithm in section 5.2.3 'Machine Learning Algorithms', mentioning the fact that it has linear time complexity. The problem with K-Means is that it has to be rerun with every new user. A user with a drastically different diet may upset the existing clusters, which could lead to worse recommendations being produced. I believe that K-Means suffices for demonstrative purposes of this project, however the recommender system would have to be reevaluated if it were to be built with scalability in mind.

6.2.9 Summary of Technical Research

Although there are no restrictions on the architecture of Android applications, a good design architecture can be very beneficial. Although it is advised to use MVVM architecture when building an Android application, some of the features of the Android Passive MVC [6] can be implemented to improve the application. For example, MVVM doesn't restrict Activities being made very bloated. Applying the general approach of the Android Passive MVC design by making the Activities and Fragments lightweight would definitely add to the user experience and fluidity of the UI. The application could be made even more lightweight by offloading the computationally intensive tasks to the cloud, providing a smooth and responsive experience to the end-user.

7 Proposed Final Design

7.1 Overview

This project is the implementation of a cloud-based Android application that estimates a user's HEI. The application will serve as a food diary for gathering the necessary information to calculate the HEI estimate.

A collaborative filtering recommender system will be used to suggest food items that the user may like in order to increase their HEI. MyFitnessPal dataset [23], which contains food diaries of over 9000 users from 2014 to 2015, will be used to populate the recommender system. A user profile is going to be built for every user in the dataset, and a clustering algorithm will be used to group users with similar diets.

The application is going to be written for Android in Java, following the advised MVVM architecture. The cloud-based component is going to be used for the storage of the user information and running the recommender system. Additionally, this approach provides privacy, as the end-user does not have access to other user's information.

7.2 Justification

7.2.1 Cloud

The reasons for implementing this project as a cloud application are as follows:

- Running machine learning algorithms may be computationally intensive, offloading the task to the cloud would speed up the application.
- The model can be dynamically updated with every new user that joins and every new food diary entry.
- The user would have the option to retrieve their data in case they lose it locally.

7.2.2 Android

The reasons for using MVVM architecture in the development of the application are as follows:

- MVVM is UI driven.
- The architecture ensures the extensibility and maintainability of the application, which simplifies updating the application in the future.
- Updating visual components can be done independently of the data, thus making it easier to update the application with the newest HEI standards.

7.2.3 Estimating HEI

Estimating the user's HEI can give very beneficial insight into their eating patterns. The user can be informed which food categories they are scoring low in, and foods belonging to those categories can be suggested to the user in order to increase their HEI score.

7.2.4 Recommender System

One of the biggest problems that people face when it comes to dieting or attempting to improve their health, is replacing foods that they like with foods that they do not.

While it is incredibly difficult to quantify something as subjective as taste, data science can reveal underlying patterns in what foods people may or may not like. Recommender systems are constantly used to offer users products based on the subjective liking of other products, such as movies and music.

A recommender system can help tackle this issue, by advising the user to eat foods that are healthier, yet the user is still likely to enjoy. The assumption is made that people with similar diets are likely to have similar tastes. There are many factors that play a role in a person's diet, which could be biological or environmental. Thus the assumption is made that the user's diet is representative of the food they like.

7.3 Requirements

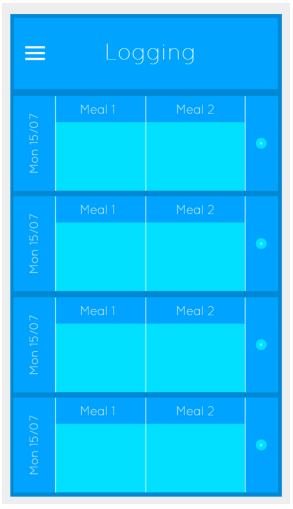
Table 2 shows the non-functional and functional requirements.

Number	Type	Name	Description
1	Non-functional	Simple design	The application should have a simple, minimal design that is easy to follow.
2	Non-functional	Fast	The application should be fast and smooth.
3	Functional	Platform	The application should be available on Android.
4	Functional	Logging	The application should allow the user to log the food they consume.
5	Functional	HEI score	The application should estimate the HEI score of the user based on the logged foods.
6	Functional	Presentation	The application should present HEI estimate in a radar graph.

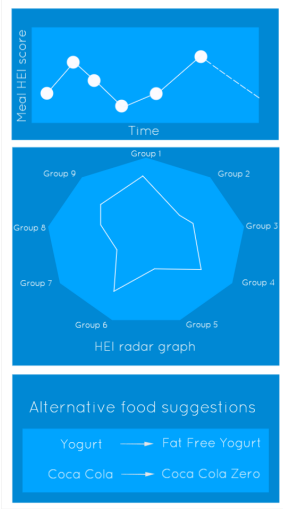
Table 2: Requirements

7.4 Layout and User Interface

Figure 1a shows a reference design of the food diary section of the application, and Figure 1b shows a reference design of the summary section, where the HEI score is presented in a radar graph.



(a) Page 1: Food Diary



(b) Page 2: Summary

Figure 1: Layout and User Interface Design of the Android Application

8 Project Management

8.1 Project Planning

Appendix D contains a Gantt chart with a breakdown of the work I planned to do at the beginning of the year 2020. Unfortunately, a few obstacles I ran into during the project really set me back. I successfully implemented the core functionality of the application as planned, but I did not manage to implement all of the additional features that I would have liked.

I was originally planning on implementing sign up an log in functionality, where the users would sign up with an email, and log in upon email verification, as described in appendices A and C.

There was a very steep learning curve in learning about Android application development. I ran into a number of obstacles during the implementation of the Android application, including saving user input to internal memory and working with LiveData objects.

8.2 Risk Assessment

Number	Risk	P	S	RI	Mitigation
1	Loss of work	1	5	5	Keep work online and keep backups in different locations.
2	Small dataset	3	3	9	Look into identifying different eating patterns†.
3	Erroneous application development	3	3	9	Regular code testing. Modularity of code could help ensure the development of one component will not affect the other components.
4	Running out of time	3	4	12	Keep a structured working pattern. Make and adhere to a plan. Focus on producing a minimum viable product. Plan for slow progress by dedicating enough time for the project.
5	Infeasibility of approach	3	5	15	Look into identifying different eating patterns†.
6	Data not suitable	3	5	15	Search for datasets conveying different data. However, it could be possible that a suitable dataset is not publicly available.

Table 3: Risk assessment

Table 3 shows the risk assessment I carried out at the start of the year. P is the probability of the risk occurring on a scale of 1 to 5. S is the severity of the risk on a scale of 1 to 5. RI is the risk index of the risk, which is P*S.

† Analysis of eating times, nutrient distributions throughout the day, or comparison of diets of users struggling to achieve a dietary goal with users who accomplished those goals could be done if calculations of HEI with the available resources are infeasible.

8.3 Encountered Problems

During this project, I encountered risks 1,2,4 and 6 described in Table 3. One global string matching technique I wanted to implement in addition to the others, was term frequency-inverse document frequency (TF-IDF) weighting for terms in the FNDDS database [1]. I generated a table with the weights but erroneously deleted it, thinking it was an outdated table I no longer needed. The MyFitnessPal dataset [23] is relatively small, containing information on

just under 10'000 users. One of the steps in building the recommender system, was to generate feature vectors for every user. I only had time to generate feature vectors for about 300 users, but that was sufficient to demonstrate the recommender system working.

The MyFitnessPal dataset was missing portion size, which is crucial in calculating HEI. I used the ratio of the number of calories in the user input to the number of calories in the matched food item from the FNDDS database to estimate portion size.

Calculating the HEI of a set of foods automatically is a very complex process. Matching food input to a food item in the FNDDS database allows making an estimate of the HEI score relatively quickly, using required precalculated information. The FNDDS database contains under 9000 food items, which is far too small to make accurate estimates. However, this was the most reasonable approach given the time frame.

9 Application Development

9.1 Requirement Generation

I started the design process by creating a list of use cases (see Appendix A) and putting them in a use case diagram (see Appendix B) which helped guide the process of requirement elicitation (see Appendix C). I approached the software development side of the project with an Agile methodology, following a similar approach to Scrum. I used the use cases, the use case diagram and the requirements to generate user stories for the product backlog, see Figure 2.

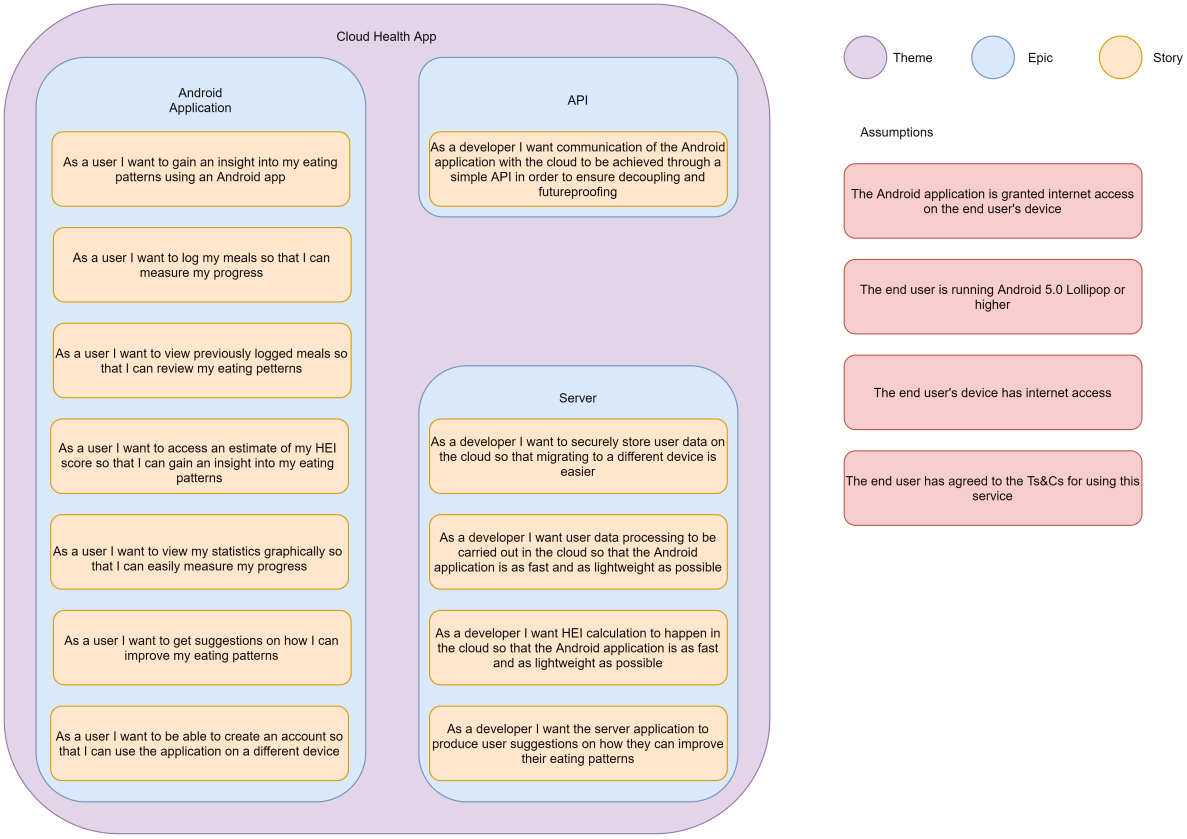


Figure 2: Product Backlog

Splitting the user stories in such a way gave room for modularity and low coupling, which allowed me to work on three separate parts of the project simultaneously. Below is a brief documentation of the process of creating the application.

9.2 Sprint 1

I chose 3 stories and broke them down into tasks to begin Sprint 1, see Table 4.

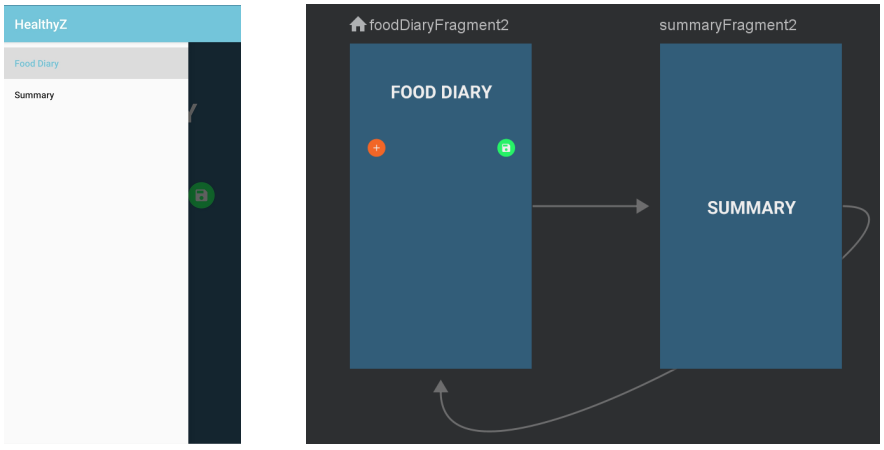
Story	Tasks
As a user I want to gain an insight into my eating patterns using an Android app	Define Fragment layouts in XML
	Define button layout in XML
	Define navigation graph
As a user I want to log my meals so that I can measure my progress	Define Fragment classes
	Define event handling for button clicks
As a user I want to view previously logged meals so that I can review my eating patterns	Define Room Database
	Define the ViewModel
	Save current meals
	Load previously saved meals

Table 4: Stories and Corresponding Tasks for Sprint 1

9.2.1 Defining Fragment Layouts and the Navigation Graph

After some consideration, I decided to modify the initial design of the application layout. I chose to display all food items belonging to a meal in a single box that would expand vertically downwards as more food items would be added to it, see Figure 4. Unnatural horizontal scrolling is thus eliminated.

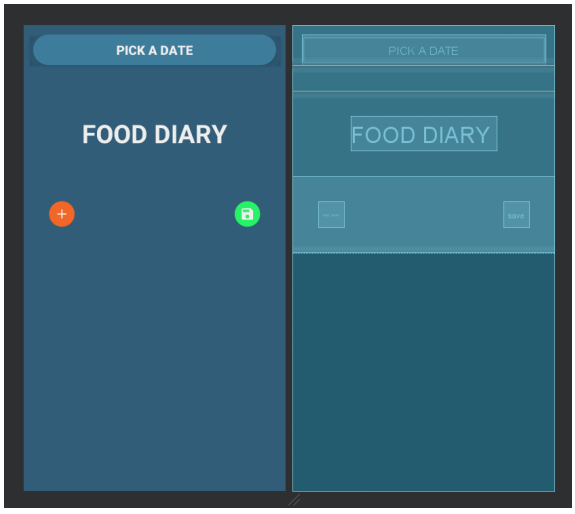
The navigation around the application remained unchanged from the initial design. A navigation drawer accessible by swiping from the left seemed to be the most reasonable and intuitive approach, see Figure 3.



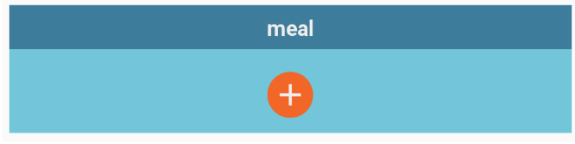
(a) Navigation Drawer (b) Navigation Graph

Figure 3: Screenshots of the Different Screens in the Application

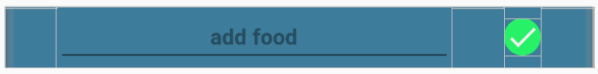
(a) Summary Fragment

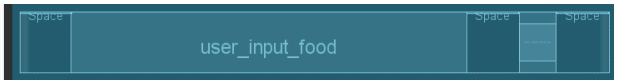


(b) Food Diary Fragment



(c) Meal Fragment





(d) Food Fragment

Figure 4: Fragment Layouts

9.2.2 Defining Fragment Classes

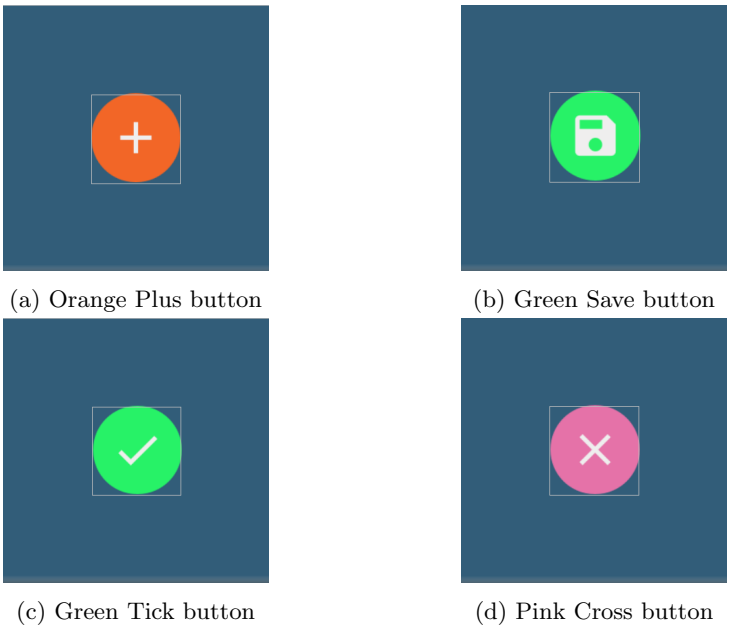
The fragment classes themselves follow the design exactly. The navigation drawer allows the user to select one of two destinations: Food Diary, or Summary page.

- **FoodDiaryFragment**: has buttons for creating a new meal (labelled with a plus symbol) and for saving the current meals to persistent storage (labelled with a tick). The FoodDiaryFragment also serves as a container for MealFragments that appear upon user request.
- **MealFragment**: has a button for adding a new food item (labelled with a plus symbol). The MealFragment, similarly to FoodDiaryFragment, serves as a container for FoodFragments.
- **FoodFragment**: contains a text field where the user can input a specific food item.

9.2.3 Event Handling

There are 4 button layouts defined in the application, which are:

- **Orange Plus**: this button creates child fragment instances. A button with this layout positioned in the FoodDiaryFragment will create a MealFragment instance, and similarly a button with this layout in a MealFragment will create a FoodFragment instance, see Figure 5a.
- **Green Save**: this button saves the user inputted meals to permanent storage, see Figure 5b.
- **Green Tick**: this button saves the food item that the user inputted into the FoodFragment ,where it is located, to the ViewModel, see Figure 5c.
- **Pink Cross**: this button deletes the fragment it is in. This button can be accessed in both MealFragment and FoodFragment instances by long clicking the corresponding Orange Plus button, see Figure 5d.



(a) Orange Plus button

(b) Green Save button

(c) Green Tick button

(d) Pink Cross button

Figure 5: Button Layouts

9.2.4 Defining a Room Database

Figure 6 shows a high level overview of the application database.

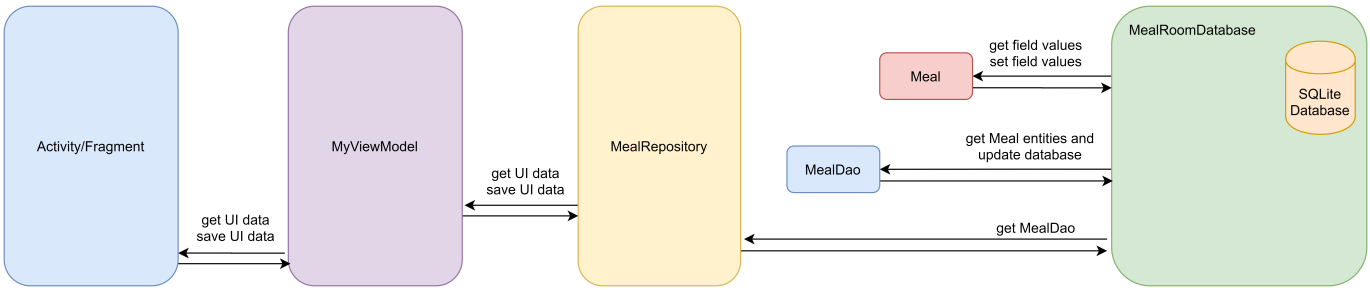


Figure 6: Application Overview

In Figure 6, there are 6 distinct levels, which are:

- **Activity/Fragment**: this includes all the previously mentioned fragments.
- **MyViewModel**: the purpose of the ViewModel is to store the UI data. The ViewModel survives configuration changes, such as device rotation, during which the Activities and Fragments are destroyed and recreated, which makes it perfect for temporarily storing the user inputted meals and food items before committing to persistent storage.
- **MealRepository**: the repository acts as a mediator between the data source and the ViewModel. The benefit of using a repository is flexibility in future development. Adding another data source would thus only require changes to the repository, the ViewModel is unaffected. I plan to add cloud storage as a second data source later on.
- **MealDao interface**: annotated with '@Dao', this interface is used to interact with the database. Methods defined in the interface are attached to SQLite queries that are executed when their corresponding methods are called.
- **Meal entity**: annotated with '@Entity', this class represents a row in the database, with getters and setters for the columns in that row.
- **MealRoomDatabase**: annotated with '@Database', this implementation of a Room database follows a singleton design pattern in order to ensure that it is accessed correctly. Singleton design pattern is allowing only one instance of a class to be made. This is sufficient for such a small application as the database will not be accessed from multiple different parts of the application simultaneously.

9.2.5 Defining a ViewModel

MealFragments are assigned IDs when they are first created so that they can be saved and recreated later. The ID counter is kept in the MyViewModel class, ensuring unique IDs for MealFragments. FoodFragments receive the ID from their parent MealFragment upon creation. MyViewModel class contains a map of integers to lists of strings. User inputted food item is added as a string to the list mapped to by the value of the meal ID in the map stored in the MyViewModel class.

9.2.6 Saving and Loading User Data

I implemented a simple date picker so that the user could change previously entered meals. I implemented the DatePickerDialog interface in the FoodDiaryFragment class in a way that causes the date picker to pop up when the user presses the 'Pick a Date' button at the top of the FoodDiaryFragment. I chose a colour scheme that was consistent with the rest of the application. See Figure 7 for a screenshot of the date picker dialogue.

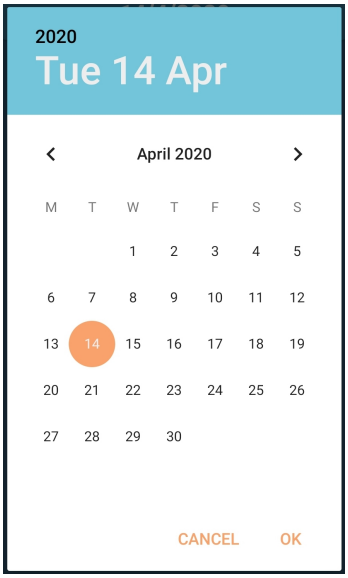


Figure 7: Date Picker

MyViewModel class has a date variable, the default value of which is the current date. The date variable is used to retrieve data from the SQLite database. When a user picks a different date in the date picker, the date variable in MyViewModel is updated and meals from the new date are loaded. Clicking the green save button (see Figure 5b) causes data stored in MyViewModel class to be saved to the SQLite database.

9.3 Sprint 2

At this point, I needed to develop the back-end of the application. The second sprint consisted of stories from the 'Server' epic in the product backlog, see Table 5.

Story	Tasks
As a developer I want to securely store user data on the cloud so that migrating to a different device is easier	Create SQLite database for storing user data
	Create SQLite table for storing user input
	Create SQLite table for storing user food set
	Create SQLite table for storing user food set by date
As a developer I want user data processing to be carried out in the cloud so that the Android application is as fast and as lightweight as possible	Generate food set from user input for every user in the dataset
As a developer I want HEI calculation to happen in the cloud so that the Android application is as fast and as lightweight as possible	Create a fuzzy string matching program
	Use fuzzy string matching program to match user inputted food items in food set to a food item in the FNDDS database
	Calculate an estimate of the scores of the HEI components by using the matched foods from the FNDDS, using nutritional data from FNDDS and FPED databases

Table 5: Stories and Corresponding Tasks for Sprint 2

9.3.1 Creating an SQLite Database for Storing User Data

Figure 8 shows the tables in the MyFitnessPal database that was created, as well as the FNDDS and FPED databases that were required for this project:

- **FoodDiary**: this table contains a list of foods and its nutritional value stored in a JSON format. The primary key is the entry ID. The reason for this is the fact that neither one of user ID, date, food list, daily goal, nor any combination of those fields could serve as a key as they're not unique.
- **DateFoodSetNut**: this table contains every user's set of foods and their nutritional values for every recorded day.
- **FoodSetNut**: this table contains every user's set of foods and their nutritional values for the entirety of the time the user kept the food diary.
- **FNDDS**: this table is the official FNDDS database as supplied by USDA.
- **FPED**: this table is the official FPED database as supplied by USDA.

Both FPED and FNDDS tables were contained in Microsoft Access Database files with the same names. These databases contain detailed nutritional information of nearly 9000 food items.

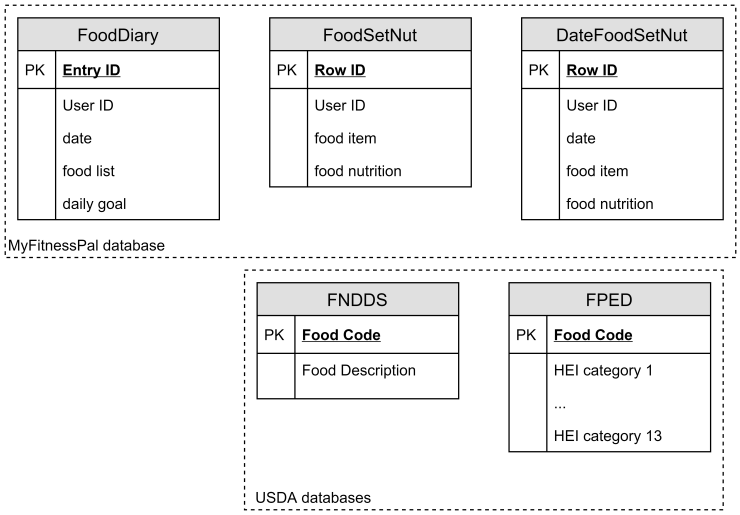


Figure 8: User Database Model

9.3.2 Generating Food Sets for Every User

The first table I populated was the FoodDiary table. I created a parser for the MyFitnessPal data set [23] and put every single line from the tab-separated file into a row in the FoodDiary table.

I then populated the FoodSetNut table by retrieving every single food item of the user input from the FoodDiary table, which added it to a map of the food item name (as a string) to that food item's nutritional value (as a string). Using a map to store the user data eliminated any duplicates, resulting in the food set of interest, which was then saved to the FoodSetNut table. Similarly, the DateFoodSetNut table was populated, but for every recorded day.

9.4 Fuzzy String Matching

The food items in MyFitnessPal data set [23] were formatted consistently, thus making it possible to efficiently clean the input. This was achieved by deleting unnecessary information about portion size, removing any brackets, numbers and non-alphabetical characters, and changing the input to lower case. Lastly, Porter Stemmer implementation from Apache OpenNLP library [26] was used to stem every word in the description.

My initial attempt was to use Levenshtein distance (see section 5.2.6 'String Comparison Metrics') as a metric to match user input to the closest food item in the FNDDS database. The problem with this approach was word order. Changing the word order would result in a low similarity score, which is undesirable.

Instead, a slightly modified version of the Mongue-Elkan algorithm was implemented (see section 5.2.7 'String Matching'). In my implementation of this algorithm, the string similarity was computed in the following way:

- For every word in the shorter sentence, assign a word from the longer sentence with the lowest Levenshtein distance.
- Return the sum of these distances.

The difference here was that my implementation was a one-to-one mapping. A word in the longer sentence could map to at most one word in the shorter sentence. Additionally, my implementation is symmetric, whereas the original algorithm is not [37].

9.4.1 Matching User Input to a Food Item in the FNDDS

In order to calculate the HEI score, I chose to take advantage of the available databases. FNDDS contains nearly 9000 food items with an in-depth description and nutritional information. I simplified the process of estimating the HEI score by finding the closest match of the user input to a food item in the FNDDS database using the fuzzy string matching program to calculate string similarity, and returning the most similar FNDDS food item. The food code associated with the match in the FNDDS was used to find the required information to calculate the HEI score in the FNDDS and FPED databases.

9.4.2 Estimating HEI Score

According to the 2015 HEI Update [10], the three basic steps in calculating the HEI score are :

1. Identifying a set of foods
2. Determining amounts of dietary constituents
3. Deriving ratios and scoring against the 2015 HEI standard

For every user input, I identified the closest food item in the FNDDS database using the fuzzy string matching program. From the FNDDS, the food code, amount of saturated fats, energy content, sodium content, monounsaturated fats and polyunsaturated fats were extracted for every match. The food code was then used to retrieve all the values of the necessary components in calculating the HEI score from the FPED database. In order to determine the amounts of the dietary constituents, I multiplied the values from the FPED by the ratio of calories the user input to the calories obtained from the FNDDS for a particular food item. In this way, the portion sizes were estimated.

Summing the relevant values up and comparing to the 2015 HEI standards resulted in an estimate of that user's HEI score, based on all the food that they entered.

9.5 Sprint 3

At this point, I needed to implement a visual representation of the HEI score. This is commonly achieved with a radar graph. The stories and their corresponding tasks are displayed in Table 6.

Story	Tasks
As a user I want to access an estimate of my HEI score so that I can gain an insight into my eating patterns	Display HEI score numerically in the application
As a user I want to view my statistics graphically so that I can easily measure my progress	Implement a radar graph for the HEI score in the application

Table 6: Stories and Corresponding Tasks for Sprint 3

9.5.1 Implementing a Radar Graph

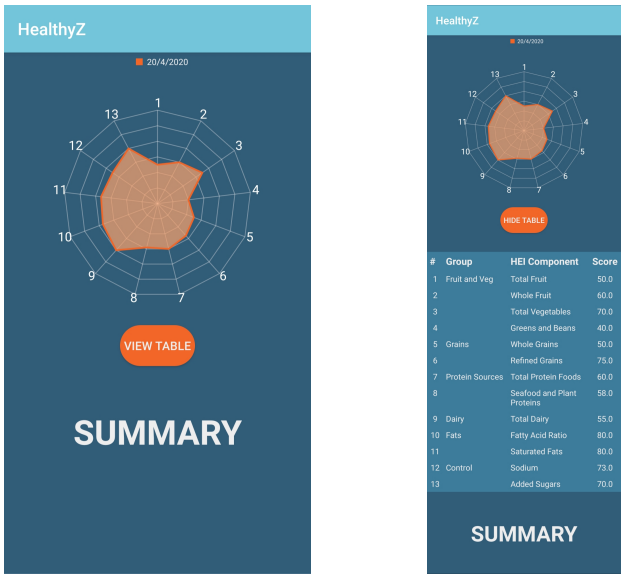
A great visual way of displaying the HEI score is in a radar graph with 13 branches, where every branch corresponds to one of the 13 HEI components.

MPAndroidChart library [24] allowed me to efficiently implement an aesthetically pleasing radar graph. Using the libraries documentation, I added a radar graph following the general colour theme of the application to the summary page.

At this point, the server-side has not yet been implemented. For testing purposes, a random HEI score, stored as an array of doubles, was hard-coded in MyViewModel. The HEI score could be retrieved from the cloud by the MealRepository instance associated with MyViewModel once the server-side was implemented. A getter method returning the HEI score array was implemented in MyViewModel, which is called when the user navigates to the summary page of the application, where the radar graph displays the HEI score. See Figure 9 for screenshots of the application with the numerical scores visible and hidden.

9.5.2 Displaying HEI Score Numerically

Due to the size constraints of the application running on a smartphone, I chose to present the individual scores of the HEI components in a table. Furthermore, as the HEI scores are only estimations and thus not precise, I would like to draw the user's attention to the radar graph displaying the overall trend and away from the numerical values. The HEI score table is hidden by default, its visibility is toggled by pressing the corresponding orange button.



(a) HEI Score Table is Hidden (b) HEI Score Table

Figure 9: Screenshots of the Summary Page in the Application
(Each Branch in the Radar Graph Corresponds to a HEI Component)

9.6 Sprint 4

At this point, there was a slight miss alignment in the project. The back-end code calculating an estimate of a user's HEI utilised the additional nutritional information of the recorded food items in the MyFitnessPal dataset [23]. The number of calories was necessary to estimate the portion size, and the salt content was to be used later for evaluation. This caused a reevaluation of the product backlog and redesign of feature implementations, see Table 7.

Story	Tasks
As a developer I want to gather additional nutritional information in order to calculate the HEI estimate	Add calorie and sodium input boxes in the FoodFragment
	Update MyViewModel and MealRepository classes to store the user input as JSON objects
As a developer I want the user to store the HEI score locally, minimising server usage and optimising loading speeds	Update project structure
	Create a new Entity class for the HEI score
	Create a DAO class for accessing the HEI score
	Update the MealRoomDatabase class to contain two tables, for storing user input and HEI score

Table 7: Stories and Corresponding Tasks for Sprint 4

9.6.1 Updating Project Structure

I decided to give the android application project some structure. As I was developing the application using the MVVM architecture, I decided to split the classes into packages based on the purpose they were serving.

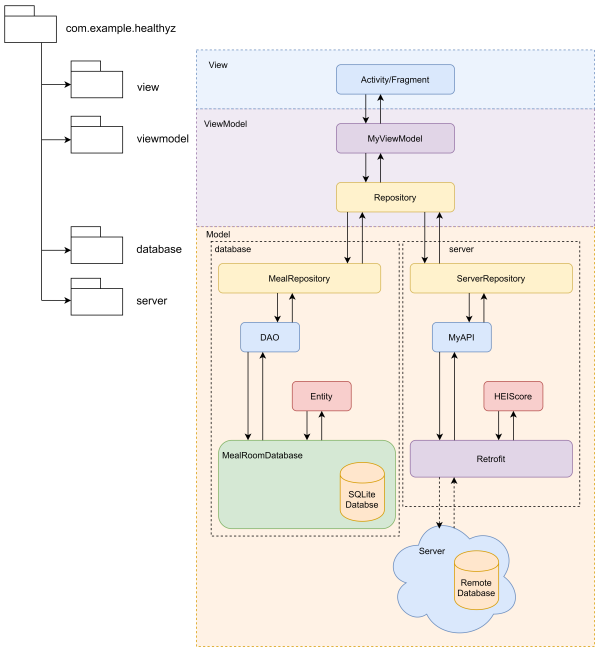


Figure 10: Application Structure

Figure 10 shows the updated design of the application architecture, with all the classes being split into sub-packages based on their purpose. All the classes responsible for the UI were placed in the 'view' package and similarly, the classes responsible for handling data were placed in the 'viewmodel' package. I split the classes responsible for data storage and retrieval into two packages, the 'database' package contained classes for local data storage, and the 'server' package contained classes for connecting to the server.

Splitting the classes into sub-packages provided an additional layer of abstraction. The MealRoomDatabase class follows the singleton design pattern to ensure correct usage. The instance of the database is package protected, and thus directly inaccessible from the other packages. It can only be accessed with the public methods declared in the MealRepository class, thus providing extra security on its correct usage.

9.6.2 Altering the Application Layout

I redesigned the application layout to allow the user to input calorie and sodium intake, as this information is required for HEI calculations, and later for evaluation. Alternatively, it would be possible to allow the user to input very detailed nutritional information, such as including protein, fat and carbohydrate intake also. This would make the application UI too messy, however. The minimum information required to perform the calculations is collected.

I added two input boxes in the FoodFragment. I then changed the green tick button event handling slightly, to ensure that the user inputted a number in both the calorie and sodium input boxes, see Figure 11.

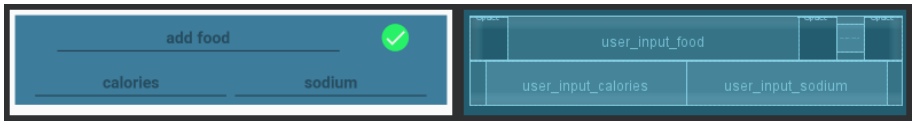


Figure 11: Updated FoodFragment

9.6.3 Updating ViewModel

Previously, user input was stored in MyViewModel class in a map of integers to lists of strings, where the integer was the meal ID and the list of strings was the list of user inputted food items. With the addition of calorie and sodium intake information, this could no longer work. I chose to store the user input in JSON format, with the attributes 'food_item', 'calories' and 'sodium' being assigned the values that the user inputs. I proceeded to update MyViewModel class to map integers to lists of JSON objects. An alternative to lists would be to simply map integers to JSON arrays. I made a compromise by keeping lists, which reduced the amount of code I had to rewrite.

JSON format was optimal, as it is serializable. This meant that I could leave the Meal entity class and MealDao interface unchanged, whilst retaining the useful functionality of the JSON Java objects in the rest of the project. Additionally, JSON is often used to transfer data, thus storing the user input in a JSON format would reduce the amount of work required to upload the user's food diary to the server for HEI calculations.

9.6.4 Storing HEI Scores locally

In order to minimise server usage and optimise loading speeds of the application, I chose to store the calculated HEI scores locally. This was achieved in the same way as with the user input information, described in section 8.2.4 'Defining a Room Database'. I took the following steps:

- **Create HEIRecord class**: annotated with '@Entity', this class is used to set and retrieve field values in the corresponding table contained in the MealRoomDatabase.
- **Create HEIRecordDao interface**: annotated with '@Dao', this interface defines the API for interacting with the table.
- **Update MealRoomDatabase class**: I added the HEIRecord to the entity declaration so that RoomDatabase superclass creates two tables for storing user input and HEI scores.
- **Update MealRepository class**: I implemented the methods defined in the HEIRecordDao interface and made them publicly accessible.

9.7 Sprint 5

Having coded the foundations of the back-end and front-end, it was then time to begin bringing it together by establishing communication between the android application and the server. Stories and their corresponding tasks are displayed in Table 8

Story	Tasks
As a developer I want communication of the Android application with the cloud to be achieved through a simple API in order to ensure decoupling and future-proofing	Set up a Docker container with TomCat and SQLite
	Implement a Java Servlet for running the HEI calculations and returning the estimated HEI score as a JSON object
	Implement a Java Servlet for allocating a user ID to an individual user
	Implement a Java Servlet for saving the user input on the server
	Implement an API for accessing and executing relevant calculations, independent of the code for these calculations

Table 8: Stories and Corresponding Tasks for Sprint 5

9.7.1 Setting Up a Docker Container

Having researched multiple different server software, I decided that Apache Tomcat [29] would be the most suitable for this project, as it provides a Java environment for web applications. In order to ensure future-proofing and to simplify the development process, I decided to deploy the application inside a Docker container [27], as this would allow the application to be deployed with ease from any machine. IntelliJ ULTIMATE integrated development environment [28] contains built-in support for using Docker containers. The latest TomCat docker container was pulled with 'docker pull tomcat' command. SQLite was then installed in the container, which made it ready for deployment.

9.7.2 Implementing a Servlet for Allocating a User ID

The purpose of a servlet is to handle HTTP requests. My servlet classes extended the HttpServlet from Java Platform Standard Edition 7.

Allocating the user ID based on the next available ID from the database would ensure unique identification of users. Due to the large size of the tables in the database, finding the maximum value of the user ID column in any table was a time-consuming process. To speed up the process, I inserted an entry at the start of the FoodDiary table with values 'entry_id'=-1, 'user_id'=MAX(user_id) and NULL for the other columns. This entry would never be used in any calculations, and it contains the highest 'user_id' in the records. When a new user is created, the 'user_id' value in this entry is retrieved, incremented, and assigned to the new user. This entry is then updated with the new, incremented value of the highest 'user_id' in the records. A more practical approach would be to use a user-chosen login, nickname, or email address, however, as I do not have such information for all the users in the MyFitnessPal dataset [23], I chose to use numbers.

9.7.3 Implementing a Servlet for Uploading User Input

Changing the Android application to save the user input in JSON format had the additional benefit of simplifying the process of uploading the user's food diary to the server; JSON is serializable and thus a popular choice of data format for data transfer.

I proceeded to implement a servlet that receives a JSON array containing user input information. This JSON array was then automatically saved to the 'DateFoodSetNut' table. In order to guarantee data freshness, any existing data for the 'userID' and 'date' parameters specified in the HTTP request is deleted from 'DateFoodSetNut' prior to insertion of the fresh data.

9.7.4 Implementing a Servlet for HEI Calculations

I implemented a Java servlet that calculated the HEI score, based on the parameters specified in the HTTP request. Supplying 'userID' and 'date' parameters in the query would run the previously implemented Java code (see section 7.5.2 Estimating HEI Score) and return a JSON array of the 13 HEI components and the actual sodium intake of the user from the MyFitnessPal dataset [23]. For this calculation, the food items and their nutritional values are retrieved from 'DateFoodSetNut' table.

9.7.5 Implementing a Server API

Tomcat servers have a 'web.xml' configuration file, which maps URIs to servlets [30]. Giving the aforementioned servlets clear and descriptive mappings makes it easy to execute them. For example, 'UploadServlet' has a servlet mapping of '/upload'.

9.8 Sprint 6

At this point, I had enough data and resources to begin implementing the recommender system. See Table 9

Story	Tasks
As a developer I want the server application to produce user suggestions on how they can improve their eating patterns	Implement K-Means clustering algorithm
	Generate feature vectors for every user in the MFP dataset
	Implement collaborative filtering by generating a list of foods that other users in the cluster have eaten in the past
	Filter the list of foods to produce food items maximising the user's HEI score
As a user I want to get suggestions on how I can improve my eating patterns	Change the layout of the SummaryFragment in the Android application to display a list of suggested food items

Table 9: Stories and Corresponding Tasks for Sprint 6

9.8.1 Implementing K-Means Clustering Algorithm

The K-Means implementation takes a list of 'User' objects containing the user ID and feature vector, the number of clusters 'k', and the number of iterations the algorithm should be executed.

Clustering is performed by randomly generating 'k' many centroid vectors, assigning the feature vector of every 'User' object to the closest centroid and then updating the centroid to be the mean vector of all the vectors assigned to it. If a cluster is empty, its centroid is assigned a randomised value.

This process is repeated for a specified number of iterations and a list of 'User' objects from the same cluster as the 'User' of interest is returned. Theoretically, this list of users would contain users with similar food preferences to the user of interest. The list of foods that they eat can then be extracted and used as a basis for the recommendations.

9.8.2 Generating Feature Vectors for Users in Dataset

'FoodSetNut' table in the database contains the set of foods a user consumed to date. I believe that such a set of foods is likely to be representative of the user's food preferences. I chose to disregard the frequency of consumption of a specific food item, as I do not think it is reflective of the user's liking of that food.

A user may really like a dish, but consume it infrequently as it is expensive or energy-dense. On the contrary, a user may force themselves to regularly consume a food item for its health benefits, despite disliking its taste. Using a set of foods eliminates that problem, however the direct consequence of this is

that foods that the user ate just once and really disliked will also, unfortunately, affect the user profile.

The data is likely to be biased. People who are willing to track their food consumption are likely to be more health-conscious than the average person, and as a result, may alter their diet. It is also important to note that the MyFitnessPal dataset [23] has been collected through the years 2014 and 2015. Social media influence has been found to affect health consciousness of young adults [42]. The changes in social media popularity since 2014-2015 could have also had an effect on the relevance of the dataset.

I recognise that biological and environmental factors are going to affect a persons diet. Allergies, financial status and even geographical location may play a role in the availability and consumption of certain food items. Taking such factors into consideration is outside of the scope of this project. For the above reasons, the following assumptions have been made:

ID	Assumption
1	Set of foods a user consumed throughout the duration of the data collection is representative of their food preferences.
2	Every user had access to foods they like.
3	Social media influence on the health consciousness of the average person has not significantly changed since the collection of the dataset.
4	Geographical location of a user did not have a significant impact on their diet due to globalisation.

Table 10: Assumptions Made in Recommender System Design

'FoodSetNut' table in the database contains columns 'food_item', which contains a description of the food item in the food set, and 'WWEIA_code', which is the corresponding WWEIA category code of the matched food item from the FNDDS database. There are 155 unique WWEIA food categories.

A feature vector for a user was generated by constructing a bit vector of size 150, where the index of a bit would correspond to one of the WWEIA food categories. The set of WWEIA codes for a user is retrieved from the database. For every WWEIA code in the set, the bit at the index corresponding to that WWEIA code was set to 1, and all other bits set to 0. If the set of WWEIA food categories for a user is A, then:

$$vector = [x_1, x_2..x_{155}]$$

$$x_{i \in \{1..155\}} = \begin{cases} 1, & \text{if } WWEIA_i \in A \\ 0, & \text{otherwise} \end{cases}$$

Given the limited resources at hand, and having made the aforementioned assumptions, I believe that this feature vector suffices. The amount of possible feature vectors of this kind is 2^{155} , which is 4.57e+46 to three significant figures. The dataset contains 9897 users, which means it is incredibly unlikely that any two users will have the same feature vector.

9.8.3 Vector Similarity Measure

Euclidean distance and cosine similarity are two metrics I considered. The formulae for Euclidean distance and cosine similarity of two vectors are:

$$Euclidean(\vec{u}, \vec{v}) = \sqrt{\sum_{i=1}^{n}(\vec{u_i} - \vec{v_i})^2} = \sqrt{|\vec{u}|^2 + |\vec{v}|^2 - 2\vec{u} \cdot \vec{v}}$$

$$CosineSimilarity(\vec{u}, \vec{v}) = \frac{\sum_i^n \vec{u_i}\vec{v_i}}{\sqrt{\sum_i^n \vec{u_i^2}}\sqrt{\sum_i^n \vec{v_i^2}}} = \frac{\vec{u} \cdot \vec{v}}{|\vec{u}||\vec{v}|}$$

I noticed that the suggestions produced by the recommender system using Euclidean distance in the initial test runs were based on the size of the input, rather than the types of food in the input.

I used bit vectors to represent users in the dataset. The value of such a feature vector at a specific index is 0 or 1, depending on whether or not the user consumed food items belonging to the WWEIA category corresponding to that index. This means that the length of the feature vector is the square root of the number of 1s in that vector. This results in the length of the vector being the square root of the number of unique WWEIA categories of foods the user consumed.

Similarly, the dot product of two feature vectors is equal to the number of WWEIA categories that both users consumed. Given feature vectors u and v, with their corresponding sets of WWEIA codes A and B, Euclidean distance between the two feature vectors is given by:

$$Euclidean(\vec{u}, \vec{v}) = \sqrt{(\sqrt{|A|})^2 + (\sqrt{|B|})^2 - 2(|A \cap B|)}$$

$$= \sqrt{|A| + |B| - 2(|A \cap B|)}$$

$$= \sqrt{|A \cup B - A \cap B|}$$

Cosine similarity of those feature vectors is given by:

$$CosineSimilarity(\vec{u}, \vec{v}) = \frac{|A \cap B|}{\sqrt{|A|}\sqrt{|B|}}$$

This shows that Euclidean distance between two feature vectors is the total number of WWEIA categories that are unique to each user. Cosine similarity of two feature vectors is proportional to the number of WWEIA categories that are common to both users, which is far more useful. Cosine similarity can thus be used to assess the similarity of two diets, rather than just comparing the number of their constituent food categories. However, Euclidean distance could be a useful measure of the difference in the variety of two diets. This could be implemented to measure a user's progress by assessing how the variety of their diet has changed over time.

9.8.4 Implementing Collaborative Filtering

Running the K-Means clustering implementation returns a list of users with similar food preferences, and a list of foods that the user of interest may like is built from the food sets of each user in that list. Implicit collaborative filtering is applied here, as there are no explicit ratings from each user. If, however, the list of users with similar food preferences is empty, food items maximising the HEI score of the user of interest are picked straight from the FNDDS database. Figure 12 shows the steps taken to produce the recommendations.

9.8.5 Filtering Set of Foods User is Likely to Enjoy

A problem I ran into during the implementation process, was the sheer size of the generated food set, which slowed down the response time significantly. A straightforward solution was to limit the number of similar users considered in the program. K-Means implementation was modified to return at most 3 users from the same cluster as the user of interest. 3 closest users were picked. This greatly sped up the process.

Two approaches came to mind about how to filter out unhealthy foods from the set of foods the user of interest is likely to enjoy. The approach that I implemented was to simply return 5 food items that increased the users average HEI score the most. This is determined by the average HEI score of the individual food item.

Potentially a better way to approach this would be to look to balance out the diet of the user of interest. This could be achieved by weighting the components. The degree to which the food item helps balance the HEI score can be determined using the following formula, where foodHEI_i represents the value of an HEI

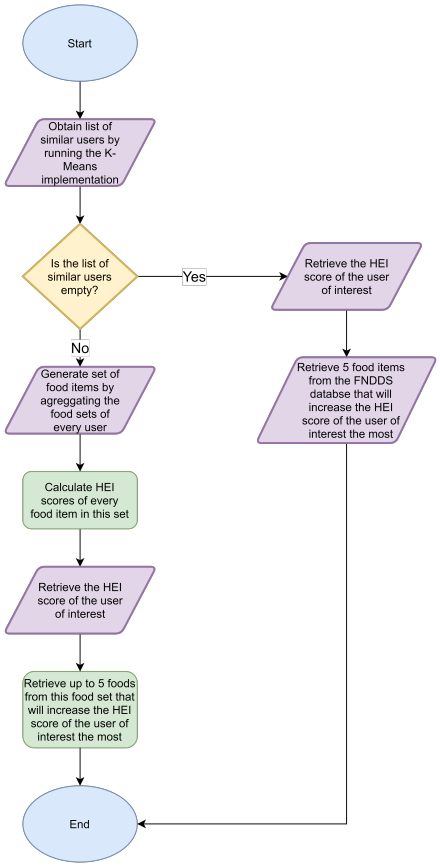


Figure 12: Steps in the Recommender System

component for the food item, and similarly userHEI_j represents the value of an HEI component for the user:

$$usefulness = \frac{1}{13}\sum_{i=1}^{13} foodHEI_i * w(i)$$

$$w(i) = \frac{\sum_{j=1}^{13} userHEI_j}{userHEI_j}$$

The proposed weighting system may not be optimal. Finding the optimal weighting function would require further analysis, which is why I chose the simpler of the two proposed methods.

10 Evaluation

10.1 Evaluation of the HEI Score Estimates

Majority of the food diary entries in the MyFitnessPal dataset [23] include information on the sodium content of the food items, which happens to be on the HEI components. Due to time constraints, it was infeasible to evaluate the outcome of the project by manually computing the HEI score of a set of foods and comparing that to the output of the HEI estimating program. A more practical approach would be to compare the estimated sodium content of a set of foods produced by the program, to the actual sodium content, derived directly from the dataset.

I am assuming that the accuracy of the sodium content estimate is the same, or similar to the accuracy of the estimates of the other HEI components.

10.1.1 Implementing Evaluation Program

A modified version of the HEI scoring application was made. In order to speed up the process, any unnecessary steps involving any calculations other than those relevant to sodium were removed. The end result was a program that returns the total estimated and actual values for the sodium content of a set of foods.

Comparing actual and estimated HEI scores for sodium would be incorrect. A set of foods with less than 1.1g of sodium per 1000 kcal would be given the maximum 10 points. Similarly, a set of foods exceeding 2.0g of sodium per 1000 kcal would be given the minimum score of 0. Therefore different food sets with sodium contents above or below the 1.1-2.0g of sodium per 1000kcal would have identical scores, which would not give any insight on the accuracy of the estimate.

10.1.2 HEI Score Evaluation Results

Due to the size of the dataset, I deemed random sampling to be the most suitable approach. The 'DateFoodSetNut' table contains the set of foods consumed, sorted by user and date. 100 such sets were randomly picked and used as input for the evaluation program. The average percentage error turned out to be 2340%.

I believe that there are multiple causes for such a high average percentage error. One such cause being that the HEI score was estimated rather than calculated. The FNDDS and FPED databases were of great use, but unfortunately, they were very small, containing only 9000 food items. The MyFitnessPal dataset [23] contains food items that are not in the databases, so the food item with the highest string similarity in the databases was used instead. HEI could be calculated directly by looking up the ingredients of the food items and their

amounts. This could be a standalone project. Additionally, the portion size was estimated using calorie content, as it was not available in the dataset.

Another such cause is my string matching implementation. I would have liked to experiment with EM clustering with GMM, as well as with text preprocessing techniques, but unfortunately, I was unable to do so in the given time frame.

I believe that another cause for such inaccuracy to be the limiting resources. Both the databases and the dataset are relatively small. Additionally, the dataset was missing important information, such as portion size. My recommender system used an implicit collaborative filtering technique, had the dataset included food ratings, explicit collaborative filtering techniques could have been used instead. This would have allowed me to make fewer assumptions and result in more accurate user models. In turn, that would have resulted in better predictions.

10.2 Evaluation of the Recommender System

10.2.1 Using My Personal Input

A simple approach to the evaluation of the recommender system would be to supply the food I eat as input and comment on whether I like the recommended food items or not. Having retrieved nutritional information from the packaging, I supplied my typical daily diet as input, see Table 11.

Calories (kcal)	Sodium (mg)	Food Description
750	2180	BBQ Chicken Pizza
244	100	Magnum Chocolate Ice Cream
840	1150	Beef Mince
230	3195	Chilli con carne sauce
894	2460	Doritos
2	200	Monster Energy Drink

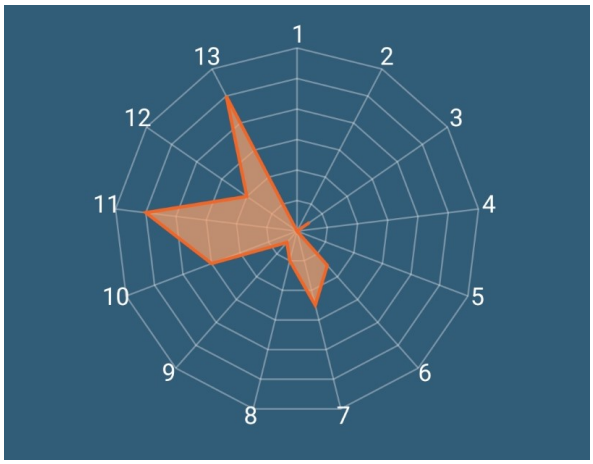
Table 11: My Typical Daily Diet

Figure 13 show the result that the application displays. I believe that the HEI score estimate does capture the overall trend, and evaluating the estimate gave a percentage error of just 47%. The recommender system suggested apple chips, eggs, sesame seeds, falafel, and chicken or turkey salad. I really like chicken and eggs, these two suggestions are quite good, eggs being a staple food in my diet. I do enjoy apples, but I do not eat them frequently, hence this is the best suggestion, as I clearly do not eat enough fruits and vegetables. On the other hand, neither sesame seeds nor falafel are the kinds of foods I think I would enjoy. 3 out of the 5 suggestions are acceptable and I would consider to be successful recommendations.

#	Group	HEI Component	Score
1	Fruit and Veg	Total Fruit	0.0
2		Whole Fruit	0.0
3		Total Vegetables	1.0
4		Greens and Beans	0.0
5	Grains	Whole Grains	0.0
6		Refined Grains	6.0
7	Protein Sources	Total Protein Foods	5.0
8		Seafood and Plant Proteins	2.0
9	Dairy	Total Dairy	3.0
10	Fats	Fatty Acid Ratio	1.0
11		Saturated Fats	10.0
12	Control	Sodium	10.0
13		Added Sugars	4.0

(a) HEI Score in a Radar Graph

(b) Recommendations Received



(c) HEI Score in a Table

Figure 13: Screenshots of the Application Upon Providing My Daily Food Intake

10.2.2 Using an Input from Dataset

I decided to run the recommender system using the input from a user in the dataset. I used user with user_id=1, and date='2014-09-21'. The recommendations produced were:

1. Egg omelet or scrambled egg, with dark-green vegetables, NS as to fat added in cooking.
2. Falafel.
3. Pesto sauce.
4. Fish curry.
5. White beans, dry, cooked, NS as to fat added in cooking.

Although it would be impossible to verify whether or not that user would like the recommended food items, an inspection of that user's food set will give a vague idea. Records show that user consumed food items with the following descriptions:

- *'Eggs - Poached (whole egg), 2 large'.*
- *'Eurest - Curry Chicken, 2 serving(s)'*
- *'Morrisons - Chicken Basil Pesto Pasta, 200 g Pack'*
- *'Mao - Seafood Laksa, 1 bowl'*
- *'Beans Green - Boiled, 50 g'*
- *'Heinz - Baked Beans - 201, 220 g'*
- *'Dim Sum - Shrimp Dumplings, 6 pc'*

I think that recommendation 1 is reasonable because the user consumed poached eggs and green beans in the past. I also believe recommendation 3 to be reasonable, as the user consumed chicken and basil pesto pasta. Recommendation 5 is a healthier alternative to baked beans, and thus an improvement. I think that recommendations 1, 3, and 5 are thus very logical.

It is difficult to say whether or not the user would like recommendation 4. That user has previously consumed seafood products, such as shrimp dumplings and seafood laksa, and chicken curry, but it is impossible to comment on another person's subjective liking. However, I do believe it is reasonable to assume that fish curry is a sensible suggestion, given the record.

Overall, 4 out of the 5 suggestions produced by the recommender system are very reasonable.

11 Conclusion

In conclusion, I successfully implemented a cloud-based application that estimates the HEI score of a user based on the food input they provided. The application also uses a machine-learning algorithm to produce useful suggestions on what enjoyable foods the user could eat to maximise their HEI score.

References

[1] Food and Nutrient Database for Dietary Studies [online] Available at: *https://www.ars.usda.gov/northeast-area/beltsville-md-bhnrc/beltsville-human-nutrition-research-center/food-surveys-research-group/docs/fndds-download-databases/*

[2] Food Patterns Equivalents Database [online] Available at: *https://www.ars.usda.gov/northeast-area/beltsville-md-bhnrc/beltsville-human-nutrition-research-center/food-surveys-research-group/docs/fped-data-tables/*

[3] Berridge, K.C., Ho, C.-Y., Richard, J.M. and DiFeliceantonio, A.G. (2010). *The tempted brain eats: Pleasure and desire circuits in obesity and eating disorders. Brain Research, 1350, pp.43–64..*

[4] Reichenberger, J., Richard, A., Smyth, J.M., Fischer, D., Pollatos, O. and Blechert, J. (2018). *It's craving time: time of day effects on momentary hunger and food craving in daily life. Nutrition, 55–56, pp.15–20.*

[5] Lee, I. and Shin, Y.J. (2019). *Machine learning for enterprises: Applications, algorithm selection, and challenges. Business Horizons.*

[6] Sokolova, K., Lemercier, M. and Garcia, L. (2013). *Android Passive MVC: a Novel Architecture Model for Android Application Development.PATTERNS 2013 : The Fifth International Conferences on Pervasive Patterns and Applications.*

[7] Campbell, C.L., Wagoner, T.B. and Foegeding, E.A. (2017). *Designing foods for satiety: The roles of food structure and oral processing in satiation and satiety. Food Structure, 13, pp.1–12.*

[8] Fonseca, D.C., Sala, P., de Azevedo Muner Ferreira, B., Reis, J., Torrinhas, R.S., Bendavid, I. and Linetzky Waitzberg, D. (2018). *Body weight control and energy expenditure. Clinical Nutrition Experimental, 20, pp.55–59.*

[9] Zendegui, E.A., West, J.A. and Zandberg, L.J. (2014). *Binge eating frequency and regular eating adherence: The role of eating pattern in cognitive behavioral guided self-help. Eating Behaviors, 15(2), pp.241–243.*

[10] Krebs-Smith, S.M., Pannucci, T.E., Subar, A.F., Kirkpatrick, S.I., Lerman, J.L., Tooze, J.A., Wilson, M.M. and Reedy, J. (2018). *Update of the Healthy Eating Index: HEI-2015. Journal of the Academy of Nutrition and Dietetics, 118(9), pp.1591–1602.*

[11] Gardner, C.D., Kim, S., Bersamin, A., Dopler-Nelson, M., Otten, J., Oelrich, B. and Cherin, R. (2010). *Micronutrient quality of weight-loss diets that focus on macronutrients: results from the A TO Z study. The American Journal of Clinical Nutrition, 92(2), pp.304–312.*

[12] Dillon, T., Wu, C. and Chang, E. (2010). *Cloud Computing: Issues and Challenges.*

[13] Christensen, L. (2007). *Craving for sweet carbohydrate and fat-rich foods - possible triggers and impact on nutritional intake. Nutrition Bulletin, 32(s1), pp.43–51.*

[14] Europa.eu. (2011). EUR-Lex - 32011R1169 - EN - EUR-Lex. [online] Available at: *https://eur-lex.europa.eu/legal-content/EN/ALL/?uri=CELEX%3A32011R1169* [Accessed 27 Nov. 2019].

[15] Android Developers. (2019). Application Fundamentals — Android Developers. [online] Available at: *https://developer.android.com/guide/components/fundamentals* [Accessed 1 Dec. 2019].

[16] Android Developers. (2020). Guide to App Architecture — Android Developers. [online] Available at: *https://developer.android.com/jetpack/docs/guideoverview* [Accessed 5 May. 2020]

[17] Android Developers. (2020). View — Android Developers. [online] Available at: *https://developer.android.com/reference/android/view/View* [Accessed 5 May. 2020]

[18] Android Developers. (2020). Fragments — Android Developers. [online] Available at: *https://developer.android.com/guide/components/fragments* [Accessed 5 May. 2020]

[19] Android Developers. (2020). ViewModel — Android Developers. [online] Available at: *https://developer.android.com/reference/androidx/lifecycle/ViewModel* [Accessed 5 May. 2020]

[20] Android Developers. (2020). LiveData — Android Developers. [online] Available at: *https://developer.android.com/topic/libraries/architecture/livedata* [Accessed 5 May. 2020]

[21] Android Developers. (2020). Room — Android Developers. [online] Available at: *https://developer.android.com/topic/libraries/architecture/room* [Acceesed 5 May. 2020]

[22] Gossman, J. (2005) Introduction to Model/View/ViewModel pattern for building WPF apps. [online] Available at: *https://blogs.msdn.microsoft.com/johngossman/2005/10/08/introduction-to-modelviewviewmodel-pattern-for-building-wpf-apps/* [Accessed 5 May. 2020]

[23] Ingmar Weber and Palakorn Achananuparp. (2016). *Insights from Machine-Learned Diet Success Prediction. In Proceedings of Pacific Symposium on Biocomputing (PSB).*

[24] Jahoda, P. (2020) MPAndroidChart *(API version 3.1.0) https://github.com/PhilJay/MPAndroidChart*

[25] Square Open Source (2020) Retrofit *(API version 2.7.1) https://square.github.io/retrofit/*

[26] Apache Software Foundation (2020) Apache OpenNLP *(API version 1.7.2) https://opennlp.apache.org/*

[27] Docker, Inc. (2020) Docker *(Version 19.03) https://www.docker.com/*

[28] JetBrains s.r.o. (2020) IntelliJ IDEA Ultimate *(Version 2020.1) https://www.jetbrains.com/idea/*

[29] Apache Software Foundation (2020) Apache Tomcat *(Version 9.0.34) http://tomcat.apache.org/*

[30] Apache Software Foundation (2020) Apache Tomcat [online] Available at: *http://tomcat.apache.org/tomcat-7.0-doc/appdev/web.xml.txt* [Accessed 8 May. 2020]

[31] Cohen, A. et al. (2020) FuzzyWuzzy Java Implementation *(Version 1.3.0) https://github.com/xdrop/fuzzywuzzy*

[32] Levenshtein, V. I, (1966) *Binary Codes Capable of Correcting Deletions, Insertions and Reversals, Soviet Physics Doklady, Vol. 10, p.707*

[33] Jaccard, P. (1901) *Étude comparative de la distribution florale dans une portion des Alpes et des Jura, Bulletin de la Société vaudoise des sciences naturelles, 37: 547–579*

[34] Cheatham, M. et. al. (2013) *String Similarity Metrics for Ontology Alignment, The Semantic Web – ISWC 2013. ISWC 2013. Lecture Notes in Computer Science, vol 8219. Springer, Berlin, Heidelberg*

[35] Ricci, F. et. al. (2011) *Introduction to recommender systems handbook. In Recommender systems handbook*

[36] R. A. Wagner and M. J. Fischer (1974) *The string-to-string correction problem, Journal of the ACM, Vol. 21, No. 1, pp. 168-173* Ricci, F. et. al. (2011) *Introduction to Recommender Systems Handbook. Recommender Systems Handbook, 1, pp.1-35.*

[37] Jimenez, S. et. al. (2009) *Generalized Mongue-Elkan Method for Approximate Text String Comparison, Computational Linguistics and Intelligent Text Processing, 10th edition, pp. 560-570*

[38] Seif, G. (2018) The 5 Clustering Algorithms Data Scientists Need to Know. [online] Available at: *https://towardsdatascience.com/the-5-clustering-algorithms-data-scientists-need-to-know-a36d136ef68* [Accessed 7 May. 2020]

[39] Lingling, N. et. al. (2020) *Streamflow forecasting using extreme gradient boosting model coupled with Gaussian mixture model, Journal of Hydrology, Volume 586*

[40] Dempster, A. P. et. al. (1977) *Maximum Likelihood from Incomplete Data via the EM Algorithm, Journal of the Royal Statistical Society. Series B (Methodological) Vol. 39, No. 1, pp. 1-38*

[41] Fielding, R. et. al. (1999) Hypertext Transfer Protocol. [online] Available at: *https://tools.ietf.org/html/rfc2616* [Accessed 7 May. 2020]

[42] Mitchell, J. et. al. (2015) *#Gettinghealthy: The perceived influence of social media on young adult health behaviors, Computers in Human Behavior, Volume 45 pp.151-157*

12 Appendix A

UC1	Sign In
Description	Signing in to the application upon application launch
Actors	User, Server
	Starts when: user launches the application. Ends when: user successfully logs into the application or the application shows an error message.
Basic Flow	1.) User launches the application 2.) User enters correct credentials for an existing account 3.) User clicks the login button 4.) Authentication successful - user is taken to meal logger screen
Exception Flow	Failure to connect to server to authenticate user, incorrect credentials and nonexistent credentials could cause failure at step 4, at which point the user is notified by a dialog box with an error message

Table 12: Use Case 1

UC2	Sign Up
Description	Creating an account to use the application
Actors	User, Server
	Starts when: user clicks the "sign up" button at the login screen Ends when: user account successfully created
Basic Flow	1.) User clicks the "sign up" button at the login screen 2.) User is taken to sign up page 3.) User enters their email, password and unique username into correct text fields 4.) User is sent authentication code to the provided email address 5.) The user is taken to authentication code input screen 6.) User verifies their email by entering the authentication code 7.) The user is taken to the home screen
Alternate Flow	The user may click "cancel" button at any point during steps 2-6
Exception Flow	The user may not receive authentication email, the authentication code inputted at step 6 could be incorrect or email verification could fail due to connection loss to server. The user is notified by a dialog box with error message

Table 13: Use Case 2

UC3	Log a Meal
Description	User logs a meal
Actors	User
	Starts when: user is at the meal logger screen Ends when: meal is successfully logged
Basic Flow	1.) User navigates to meal logger screen in the application 2.) User clicks “add meal” button 3.) User enters foods consumed 4.) User enters total calories, proteins, carbs and fats consumed 5.) User clicks “save” button 6.) Meal is logged as a JSON object
Alternate Flow	The user may click “cancel” button at any point during steps 3 and 4
Exception Flow	The application may not save the logged meal, for example if there isn’t enough local storage on the mobile device. The user is notified by a dialog box with an error message

Table 14: Use Case 3

UC4	Delete Logged Meal
Description	Delete an erroneous meal
Actors	User
	Starts when: the user is at the meal logger screen and has identified a logged meal they wish to delete Ends when: the erroneous meal is deleted
Basic Flow	1.) User navigates to meal logger screen 2.) User selects the desired meal 3.) User clicks "delete" button 4.) User is presented with dialog box asking for confirmation to delete the meal 5.) User clicks "OK" button 6.) Meal is deleted from meal logger
Alternate Flow	The user may click "cancel" at step 4
Exception Flow	The application may not delete the meal. This could arise if information is stored in the cloud and across multiple servers

Table 15: Use Case 4

UC5	Navigate to Desired Screen
Description	Navigating between the meal logger screen and the summary screen
Actors	User
	Starts when: user is at the meal logger screen Ends when: user is at the summary screen
Basic Flow	1.) User is at the meal logger screen 2.) User clicks the side toolbar button 3.) Side toolbar appears on the edge of the screen 4.) User clicks “Summary” button 5.) User is taken to summary screen
Alternate Flow	The user may click any other option available in the toolbar or go from summary screen to meal logger screen
Exception Flow	The summary screen may not load due to a bug in the code or a glitch in the operating system of the mobile device the app is run on

Table 16: Use Case 5

UC6	Log Out
Description	Logging out of the application
Actors	User
	Starts when: a user is logged into the application Ends when: the application is at the login page
Basic Flow	1.) User is logged into the application 2.) User clicks the side toolbar button 3.) Side toolbar appears on the edge of the screen 4.) User clicks the "logout" button at the bottom of the toolbar 5.) User is presented with dialog box asking for confirmation to logout 6.) User clicks "OK" button 7.) Application displays the login page
Alternate Flow	The user may click "cancel" button during step 5
Exception Flow	The application may not logout the user due to a bug in the code or a glitch in the operating system of the mobile device the app is run on

Table 17: Use Case 6

UC7	Upload Logged Meals to Server
Description	Logged meals are uploaded to server for analysis
Actors	Server
	Starts when: server requests logged meals or user requests HEI calculation Ends when: meals are saved in server storage
Basic Flow	1.) Server requests logged meals 2.) The logged meals are parsed as a single JSON object 3.) The JSON object is sent to the server 4.) The server saves the JSON object in its storage
Exception Flow	The server may not receive the logged meal or receive corrupted log, for example if the internet connection has been disrupted during the process. For this process, acknowledgements must be used

Table 18: Use Case 7

UC8	Calculate HEI Score
Description	Calculating HEI score based on the meals logged by the user
Actors	Server
	Starts when: all food items logged by a user have been mapped to food items in the FNDDS Ends when: HEI score is calculated
Basic Flow	1.) Once all food items logged by a user have been mapped to food items in the FNDDS, the ratio multiplier is calculated by dividing user inputted calories by calories in the FNDSS database. 2.) The amounts of all necessary food groups are looked up for the food item in the Food Patterns Equivalents Database (FPED) and multiplied by the ratio multiplier 3.) The gathered information about the food groups is inputted into the HEI scoring algorithm, which is provided by the United States Department of Agriculture (USDA), outputting the HEI score for that particular user
Exception Flow	The server may not receive the logged meal or receive corrupted log, for example if the internet connection has been disrupted during the process. For this process, acknowledgements must be used

Table 19: Use Case 8

UC9	View Previous Logs
Description	View logged meals for previous days
Actors	User
	Starts when: the user is at the meal logger screen and wants to view a meal that they previously logged Ends when: the previously logged meal is displayed
Basic Flow	1.) User is at the meal logger screen 2.) User clicks "select day" button 3.) User is presented with calendar widget 4.) User navigates to the desired date 5.) User clicks the desired date 6.) The meal log for that day is loaded
Exception Flow	If the user hasn't logged any meals for the selected day, no information is going to be presented. The user is notified by a dialog box with an error message

Table 20: Use Case 9

UC10	Starting the Application
Description	Loading the application on a mobile device running Android operating system
Actors	User
	Starts when: application is downloaded and located on the user's device Ends when: application is loaded
Basic Flow	1.) User locates the application on their home screen 2.) User clicks the application icon 3.) The application loads up
Alternate Flow	The application may not load due to a bug in the code or an error in the operating system of the device

Table 21: Use Case 10

13 Appendix B

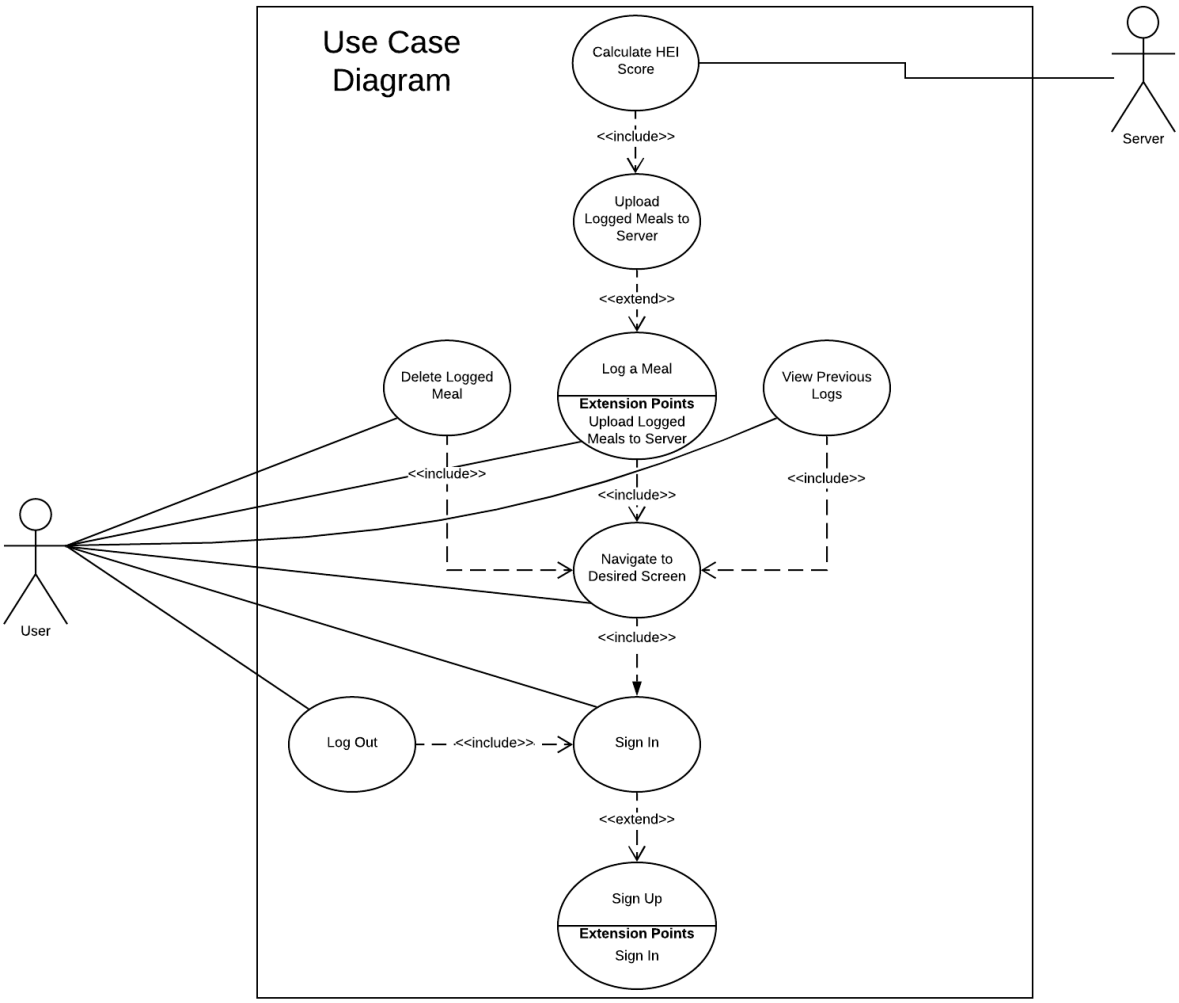


Figure 14: Use Case Diagram

14 Appendix C

Req. ID	Description	Associated Use Case	Req. Number
1	The application should be written for Android	10	10.1
2	The application should have internet connection	7	7.1
3	The application should be a cloud application	7	7.2
4	The application should securely store user information on the server	7	7.3
5	The application should have a navigation drawer so that the user can navigate round the application easier	5	5.1
6	The application should have a "Meal Logger" screen where the user can input meals they consumed	5	5.2
7	The application should have a "Summary" screen where the user can view their estimated HEI score	5	5.3
8	The application should allow the user to create a new account upon launch so that they can use the application	2	2.1
9	The application should allow the user to sign in upon launch so that the user can use the application on multiple different devices	1	1.1
10	The application should only permit successful login with correct credentials	1	1.2
11	The user should be able to log out of the application so that an another user may log in	6	6.1
12	The application should allow the user to log a meal they consumed by providing the names of the food items they consumed and their nutritional values (calories, proteins, carbs and fats)	3	3.1
13	The application should save the logged meals in JSON format	3	3.2
14	The application should allow the user to view previously logged meals	9	9.1
15	The application should allow the user to delete previous meal entries so that the data input for calculating the HEI score is more representative	4	4.1
16	The application should calculate an estimate of the user's HEI score (calculations are carried out on the server)	8	8.1
17	The application should recalculate the estimate of the user's HEI score if the user deleted old entries or added new entries	8	8.2
18	The application should display the estimated HEI score in a radar graph on the "Summary" screen	8	8.3

Table 22: Requirements

15 Appendix D

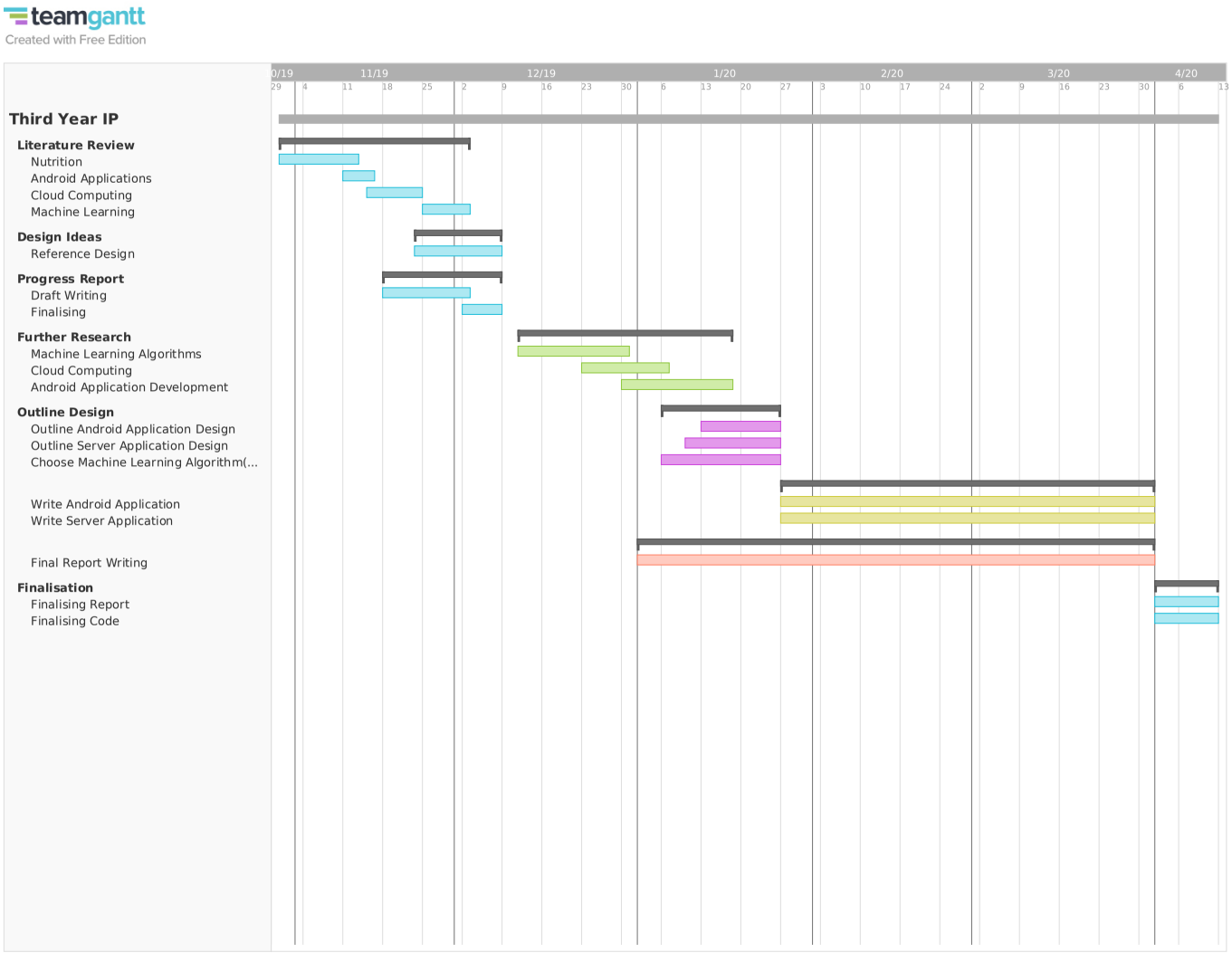


Figure 15: Gantt Chart

16 Appendix E

COVID-19

Self isolation during lock down has had an impact on my mental health, which resulted in decreased productivity and lack of motivation. Nevertheless, setting targets and milestones helped me implement the project with the core functionality.