# Project Report [Archived]

Product Ingredient Comparison Application

App Name: Scannera

## Developers:

Adam Poulton
Yaoyang Ding
Yinghua Zhou

# Table of contents

# Introduction

Comparing food products manually by reading the packaging labels and the nutritional information can be tedious and time consuming, especially in the context of shopping where going through all the candidate products is a huge task, and interpretation errors can be easily made. What if there's a solution to such a problem by just having a mobile phone?

Scannera is a food product comparison mobile application developed by our team for the purpose of enabling consumers to easily compare items in the supermarket based on their nutritional characteristics. Our team is comprised of three members: Yinghua - who is the Front-end developer responsible for the Android Application, Adam - who is the backend developer responsible for the web server and Optical Character Recognition pipeline, and Yaoyang - who is the backend developer responsible for the web server API endpoints as well as database design and configuration.

The aim of this project is to help consumers to make more informed purchases at the supermarket by providing product nutritional information and comparisons through an easy-to-use mobile application.

Scannera will achieve this aim by enabling users to:

1. Scan product barcodes with their phone to…
    I. View the nutritional information.
    II. Compare the nutritional information with similar products.

2. Contribute product information that is missing by…
    I. Submitting the product name, brand, category, price and photo.
    II. Capturing the nutritional table using their phone camera.

3. Access supportive features that help with their food buying decisions.

# Project Background

All food products in Australia are required to display a label showing the nutritional information of the product as well as a barcode that uniquely identifies the product. The nutritional label must follow a standard tabular format and display the information in a consistent order. The information displayed in the table includes energy, protein, fat, sugar and sodium content with the exact list of nutritional categories varying based on the type of product. Consumers use this information to make decisions about which products to purchase depending on their nutritional needs and desires. The problem faced by consumers is that evaluating products by manually comparing them label by label can be both difficult and time consuming.

Our application aims to enable consumers to compare and contrast products quickly and effortlessly by using their phones. Users can scan product barcodes with their phone camera

and our application will be able to look the product up in a database to provide a nutritional breakdown of the product including a comparison of the nutritional attributes against products of a similar category. Users can also contribute to the product database by providing missing information themselves. We have it easy for users to contribute to the system by leveraging Optical Character Recognition (OCR) technology to automatically extract the nutritional information from user supplied images of the product nutrition tables. By providing a quick and easy mechanism for users to compare and contrast products as well as a convenient method of contributing product information using OCR we will ensure that users will be able to make more informed decisions around their food product purchases.

# Literature Review

## Project Management Methodology

The predictive approach as the traditional project management methodology has been around for decades that is commonly adopted in industries, it aims to plan the project holistically at the start with the initial detailed plan being followed throughout the project (Špundak, 2014; Thesing et al., 2021). Such a plan gives a structured vision of how the project will be conducted thereby providing stakeholders with strong predictability of the project (Thesing et al., 2021). However since the predictive approach strictly follows the initial plan and delivers all at once, it can potentially contribute to the project failure as no proper responses are designed for the inevitability of changes and deviations in the plan (Špundak, 2014; Serrador & Pinto, 2015). Thus the resulting objections to the predictive approach have brought the advent of approaches like agile.

The agile approach contrasts with the predictive approach by emphasising flexibility during the development, as well as the interaction and collaboration among the team and the stakeholders (Serrador & Pinto, 2015). It focuses on managing and delivering the solutions in an incremental and iterative manner rather than rigidly following the initial plan (Serrador & Pinto, 2015), therefore providing opportunities to respond to changes and uncertainties, as well as resolving the deviations, in such a way to proactively mitigate risks that can potentially threaten the project success (Imani et al., 2017). Although the agile approach has gained increasing popularity due to its advantages, it is not possible to assert that it works better than the others for any projects (Špundak, 2014).

Scrum as a symbolic framework of the agile approach, has become the face of agile and the terms are often used interchangeably (Srivastava et al., 2017). It treats major portions of software development as a controlled black box (Schwaber, 1997), and provides steps to manage and control the development process in a gradual manner (Srivastava et al., 2017). A scrum project typically consists of small phases called sprints, which usually lasts 1 to 3 weeks, that the team would work together on the assigned tasks and complete within the timeframe (Srivastava et al., 2017). In such a way, it embraces flexibility and team interaction to guide us on achieving the project goals progressively, therefore reducing the risks by eliminating problems proactively as a whole and providing viable solutions

(Schwaber, 1997; Srivastava et al., 2017). The Scrum approach however, significantly reduces the amount of documentation that it could be difficult for developers to work on unfamiliar parts if no such guidance is provided, therefore resulting in unnecessary time consumption or potential failures (Cho, 2008). Just like other approaches, although Scrum is a powerful framework, it does not mean it's suitable for any kind of project, many factors like working environment could affect the performance of the approach (Cho, 2008).

Kanban as another framework that falls under the agile approach, is also widely used in software development to drive project teams to visualise the workflow, limit work in progress at each workflow stage, and measure cycle time (Ahmad et al., 2013). On top of agile, it focuses on the transparency of each team member's progress towards their assigned tasks, and therefore potentially facilitates team collaboration, while also mitigating the relevant risks (Ikonen et al., 2011). However, besides clarifying team members' awareness of the current circumstances including upcoming tasks, Kanban itself does not guide us on shaping any particular tasks or milestones of the project, that it is up to the project team to manage and make decisions (Ikonen et al., 2011).

It is quite common that elements from the aforementioned approaches may be considered to manage a particular project, in which the project is expected either because of the environments or the stakeholders to be predictable in its scope and flexible in its requirements (Imani et al., 2017), and that's when the hybrid approaches come into play. The hybrid approaches typically enrich the plan-driven process but also make use of the agile principles (Thesing et al., 2021), thereby combining the essences of the approaches in order to increase the success rate. Imani et al. (2017) proposed an empirical study based overview of the hybrid approaches that helped prove the improvements of using hybrid approaches. However, just like some other research, the proposed paper makes no attempt to give sufficient consideration to the downsides of the approaches and the limitations of the study may have impacted the outcomes of the empirical research.

## Usability

As mobile applications are having growing popularity due to their significant advantages in terms of portability and accessibility, the number of mobile applications on the market has increased significantly, requiring developers to consider more aspects of their applications to attract users' interest, and an important one is the usability (Nayebi et al., 2012). Usability as defined by ISO 9241, "the extent to which a system, product or service can be used by specified users to achieve specified goals with effectiveness, efficiency and satisfaction in a specified context of use" (The International Organization for Standardization [ISO], 2018), such a definition has been widely adopted, which guides the developers to think in terms of the users during the development as to fulfil their satisfactions therefore improving the project success. Users tend to prefer applications that are efficient to use, easier to learn and meet their expectations, which are the crucial aspects of usability for all types of software applications (Nayebi et al., 2012), in order to enhance the competitiveness of the application in the marketplace. On the other hand, a poorly designed application may not have the expected value even though it is well implemented, as users may be frustrated with using the application and never want to use it anymore (Ayob et al., 2009).

Ayob et al. (2009) summarised the existing guidelines with additional elements for mobile applications since no similar guidelines were developed at that time, these include “Design for speed and recovery”, “Design for multiple and dynamic contexts” and so on. However the paper fails to offer an adequate explanation for the proposed strategies and lack of assessment on the effectiveness of the proposed framework is concerned, otherwise the paper does provide a meaningful observation on the usability guidelines for mobile application development. Shitkova et al. (2015) after a few years developed a structured and evaluated version of guidelines that contributed to the usability field in terms of mobile application development. Such guidelines are classified into “Layout”, “Navigation”, “Design”, “Content” and “Performance” with each containing very detailed expressions that make them easy to understand and follow. Years later in 2020, Weichbroth proposed the most crucial usability attributes specifically for mobile applications based on his empirical research, that are the efficiency, satisfaction, effectiveness, learnability, memorability, cognitive load and errors, in the sequence of their importance from high to low. Such study has brought the usability concept in a broad term in terms of mobile applications into a more implementable state, that facilitates mobile App developers integrating the user perspectives into practice. However guidelines are not fixed rules and various factors need to be considered when adapting to different contexts (Shitkova et al., 2015), therefore it’s always important to conduct usability testing and evaluate the results to enhance the usability of the application (Nayebi et al., 2012).

Most studies in the field of usability have only focused on testing, evaluations and measurements, however, although many have identified usability has been the one of the main concerns during development, since the users of an application, and their judgement, ultimately decide on its success or failure (Weichbroth, 2020), usability as a broad term that can be categorised into different images, may be subjectively interpreted in some specific contexts, by various users based on their different perspectives, experiences or cultures. Therefore research on usability in terms of guidelines for a particular software domain may be beneficial, and the targeted context of usage and users may be specifically evaluated before integrating the design into actual practice (Hertzum, 2010).

## OCR

Optical character recognition (OCR) is a process of text recognition in digital images and it can save a lot of time and work when it is implemented on digital images and documents. Document scanning using OCR can be done by mobile phone camera (Ch et al., 2015). The performance of OCR can be judged based on the quality of the documents and the camera being used to capture the raw image(Ch et al., 2015). Since we won’t change the external factor printed quality as it is not a part of our goals, we would focus on improving OCR by modifying the raw image quality. The author issued factors that would affect image quality, which are the focus of the camera, the resolution of the picture and the amount of noise present. A method called pre-processing was proposed to address noise produced in the scanned image, examples can be to use normalisation and smoothing tools. For normalisation, it is to handle the uniform size, slant and skew correction and for smoothing, is using filling and thinning technique(Ch et al., 2015). This will give us great inspiration for improving the readability of OCR by reducing the noise of images.
Brisinello et al. (2018) performed OCR with Tesseract on images with colourful backgrounds and concluded that it usually ends up with wrong results. The author proposed a

preprocessing method to improve this issue. Another step was mentioned to improve the accuracy is to manually crop the raw image from larger images, which has inspired us on another way of image pre-processing. Ch et al. (2015) constructed an OCR system with Tesseract as the engine to recognize characters from scanned images by taking steps of feature extraction and recognition. The accuracy of OCR can be improved by recognizing the words in two passes using Tesseract. It tries to recognize the words and improve the accuracy in the first pass, and if the match is found, the found word will be passed on to the Adaptive Classifier, which takes a step further to enhance the text accuracy. Words that were not well recognised would be processed again in the second pass, along with the fuzzy spaces would be resolved (Ch et al., 2015). A complete Tesseract architecture with its flow chart for the OCR system has been presented by Ch et al. in 2019, which provides us an entire overview of how the Tesseract framework works. However, the paper failed to provide more detailed explanations and the viable solutions toward the issues when dealing with noisy or low resolution images that could be frequently encountered while using OCR on handheld devices, therefore more practical discoveries had to be made in practice.

## Database Model

Data storage methods are becoming increasingly diversified due to the pervasive use of data in modern applications across different industries. We then hence need to find a suitable database model that works best on our application with respect to storing data and querying data. Our selection criteria of the database is mainly based on the performance and data consistency.

B et al. (2016) argued that MongoDB is commonly adopted in all organisations because it enables them to build applications which are faster, handle highly diverse types of data, and manage applications more efficiently at scale. Given the application scenarios of the hypermarket, the experiment conducted by B et al. (2016) was to test performing two operations, insertion and searching. It was concluded that when the number of records inserted or searched is smaller, there is no difference in the execution time. However, when the number of records is higher, MongoDB takes less time compared to MySQL. MongoDB can be preferred for better performance. Given the opinion, the amount of our processed data can be considered to be small and not variable, so there is no need to pursue fastness.

Therefore, other characteristics (such as reliability and consistency) become a priority factor. The transaction function of MongoDB is weak compared with the relational database system, and if a failure occurs during the update, the data may be lost, and the memory space is large.(Seo et al., 2017). More generally, SQL databases are outweighing the benefits of NoSQL regarding data consistency for an application since NoSQL does not offer any security features and does not support transactional processing (Čerešňák & Kvet, 2019).

Bhalerao et al.(2016) discussed the difference between SQL and NoSQL. In terms of system maturity, NoSQL might generate more issues or security breaches because of the lack of maturity of the system. A study comparing the performance of Select and Insert queries between MySQL and MongoDB systems by examining the time taken was conducted by the author. It was concluded that MySQL required less response time on Select query but more on Insert query. Moreover, MySQL responses were more stable as compared to MongoDB,

which is the stability factor we are concerned with. These factors have led us to consider MySQL a more attractive option compared with MongoDB based on the amount, complexity and structure of our data.

# Outcome

## What has been Implemented

1. A frontend Android Application that serves as the product that our end users will interact with.

2. A backend Web Server that the frontend application will interface with in order to execute its functionality.

3. An OCR Pipeline module that the Web Server calls upon to extract nutritional table data from images.

4. A backend Database server that the Web Server will communicate with to retrieve and modify information.

## Result achieved/Product delivered

We achieved the delivery of the Scannera Android application that helps users make informed decisions about food products. We also delivered a Web Server, Database Server and OCR Pipeline that facilitate the communication, functionality and data requirements of the Scannera application.

## How are requirements met

We have identified the requirements for our project based on the provided project brief, with the requirements traceability matrix established in the previous semester that we have been strictly adhering to. The following outlines our solution to each of the requirements.

***Front-end Android Application***

[*Requirement 1*]

| **Function / Activity** | Login & Sign Up system |
|---|---|
| **Description** | The App is capable of handling user login, sign up to allow access to the functionalities of the App. |
| **Solution** | We have delivered a sign up page for user account registration, a login page for registered accounts to enter the App home page, a forgotten password page for users to reset their password if lost. The App additionally also supports Facebook & Google logins to facilitate the experience. |

[*Requirement 2*]

| | |
|---|---|
| **Function / Activity** | Retrieving product data via barcode scanning. |
| **Description** | Users should be able to retrieve product data by scanning its barcode. |
| **Solution** | A scanning interface for capturing the numbers from a barcode has been delivered. Users can easily and immediately get the information of the associated product by pointing the camera to its barcode if the product exists in our database. |

[*Requirement 3*]

| | |
|---|---|
| **Function / Activity** | Product breakdown page. |
| **Description** | Which should display the previously stored nutrition information for a product once retrieved, and those of similar products, and a way to compare them. |
| **Solution** | If a product exists in the database and its barcode has been scanned, the App offers a page that displays the relevant information of the product, including its brand, name, category, price and the nutrition attributes in a list format. The similar products are also contained in their card views on the same page, users can sort them by selecting a factor, say Sugar, with a specified order, for comparison. |

[*Requirement 4*]

| | |
|---|---|
| **Function / Activity** | Capture the data of a product. |
| **Description** | The App should allow users to take a photo of the nutrition table of a product, and input the other necessary attributes. |
| **Solution** | Whenever the barcode of a product which doesn't exist in the database is scanned, the App prompts the user a way to enter the information of the product. The nutrition table and the product display are required to be photo taken, with the others manually entered as different products have different styles on the texts that could be difficult for OCR processing. The submit button only shows up when all the required data is filled in to maintain data integrity. |

[*Requirement 5*]

| | |
|---|---|
| **Function / Activity** | Retrieve the scanned products. |
| **Description** | Users should be able to access their recently scanned products. |
| **Solution** | The App provides the "Scan History" page for this, which can be accessed under the "Account" tab, with each item under the corresponding timeline it was scanned by the user. |

[*Requirement 6*]

| | |
|---|---|
| **Function / Activity** | Edit user account details. |
| **Description** | The App should provide a way for users to edit their account details. |
| **Solution** | “Account Information” button under the “Account” tab has been provided for users to view and edit their account details, editable fields including the profile image, the username, password, firstname and lastname. The Delete account button is also provided. |

[*Requirement 7*]

| | |
|---|---|
| **Function / Activity** | Send data to the back-end server. |
| **Description** | The App should be able to communicate with the back-end server for data. |
| **Solution** | The front-end Android App uses Retrofit library which provides Android API framework to make API requests and receive the responses. With all the necessary API calls being defined and the data encapsulation pre-processed to fulfil the required formats, the App can connect with the server whenever necessary. |

## ***Back-end Server***

[*Requirement 8*]

| | |
|---|---|
| **Function / Activity** | Receive and process data accordingly from the front-end. |
| **Description** | The server should be able to distinguish each request received, and process it accordingly without failures. |
| **Solution** | We delivered the RESTful APIs using the Flask framework in Python as discussed in the methodology section. The server distinguishes each API request by identifying its HTTP method, endpoints and the request body data, and will process accordingly based on the contexts. |

[*Requirement 9*]

| | |
|---|---|
| **Function / Activity** | Read and process nutrition table images using Optical Character Recognition (OCR). |
| **Description** | The OCR algorithm should identify the nutrition table amongst other irrelevant information within the image, and retrieve the text accordingly from the table. |
| **Solution** | We implemented the OCR Pipeline as discussed in the methodology section. Given an input image it is able to extract the nutritional table information. |

[*Requirement 10*]

| | |
|---|---|
| **Function / Activity** | Store or update all the product or user data sent from the front-end. |
| **Description** | The server should be able to retrieve, store, update or delete the corresponding data if required. |
| **Solution** | We connected our Flask server and the remote MySQL database server by utilising the SQLAlchemy Python library, which provides a handy way to perform the database CRUD operations on our Flask server. Each API function on the server has been assigned the corresponding logic to manipulate with the data (retrieve, store, update or delete) based on the requirement. |

For some of the requirements specified in the project brief, such as rejecting product that has already been stored, is not explicitly implemented, however the same goal is achieved by our delivered software, as the front-end Android App does not show up the entry for collecting a new product upon successful retrieval of it from the database using its barcode, which is implicitly handled by our mechanism overall. The classification of products is also potentially carried out as the category of a product is required and will be stored for later queries. It turns out that our deliverables have satisfied all the given requirements as a whole.

## Justification of decisions made

1. ***Decision made*:** We introduced Kanban at the beginning of our project execution on top of the hybrid approach of predictive and agile as originally planned.
   ***Justification*:** Although we had an original plan from our previous semester as a roadmap to guide us, there were a noticeable amount of deviations in practice that we weren't able to visualise by following our Scrum strategy itself. Therefore Kanban is the way to make up for the lack of documentation while also visualising each team member's work for a better collaboration.

2. ***Decision made*:** We chose SQL over NoSQL.
   ***Justification*:** It is easier to find resources on relational database management systems that could support our implementation throughout, as it comes earlier and has been extremely popular and mature. SQL provides better reliability and stability based on the research we have carried out, and since our data model highly involves relationships between each other (e.g., USER as an entity scans a PRODUCT as another entity), and we are relatively familiar with SQL over NoSQL based on our experience, therefore SQL has been chosen for our project.

3. ***Decision made*:** The front-end UI design was eventually embedded in the corresponding XML implementation.
   ***Justification*:** Due to the limitation of time resources provided as students, we realised at the beginning of the project execution, that it would be impossible to fully come up with the UI design separately as planned within the given time frame, that

would take approximately 4 weeks to complete, which could pose significant risk of delays of delivering software pieces as planned. Therefore we eventually decided to embed the UI design in our XML implementation, since the XML implementation basically is for designing the UI components, but in a technical specific style, which is acceptable.

4. ***Decision made*:** Deployment of our server on Heroku platform.
   ***Justification*:** We required our server to be accessible by the front-end mobile application as well as by each team member at all times during development. It would be inefficient and ineffective to have our server run locally only, as this would be an impediment to our continuous collaborative testing and debugging. We were initially open to using the virtual machine that was provided by the university for hosting, however after discussions with the teaching team we were not confident in resolving the communication challenges posed by the university VPN. Due to the uncertainty and lack of online documentation on dealing with this niche issue we instead opted to utilise the well documented, free hosting service Heroku for our web server.

5. ***Decision made*:** Cancel the feature which allows authorised users to manually edit product data through the front-end Android App, replaced by other features.
   ***Justification*:** Initially we were looking for such a feature to make data manipulation easier for authorised users (typically just the developer team at the beginning) by providing the entries through the front-end. However upon discussion and our highly usage of Postman, which is a third-party platform that provides a handy way to make requests to the server, therefore we found that having the aforementioned feature would bring very little value but cost a bunch of time resources which is unnecessary, and such a feature would not be relevant to most of our potential users. So we decided to not implement that feature, however introduced some other new features like the chatbot as a replacement.

6. ***Decision made*:** Acceptance of imperfection of the OCR capabilities in our existing software.
   ***Justification*:** We accept that our OCR algorithm will not work for every possible picture of every possible product. We aimed to have our OCR detect a wide variety of product nutritional table images, however it was not possible for us to achieve perfection. Our solution works well for the majority of products where the background is light, the text is dark, the surface is flat and light reflections are minimal. We identified some modifications that could improve the performance of the algorithm on the images that did not conform to the visual characteristics of the majority of product labels, such as inverting images that contained white text of a dark background. Unfortunately we were unable to incorporate these improvements to handle dark-background images without reducing the detection accuracy for light-background images. There is further work that could be done on applying different operations to images based on detected heuristics, or running multiple concurrent pipelines on the same image and using the best result - However this would come at the cost of slowing down processing time or increasing the computing capacity of the server.

7. ***Decision made*:** Publish the front-end App to Google Play.

***Justification*:** Having the App working on Android Studio alone without unexpected behaviours doesn't mean it's bug free or will function properly when it's on the users' phones, as the ultimate goal of our project. Therefore we decided to publish the App to Google Play to see whether everything can work expectedly and resolve any problems if time allows.

## Discussion of all results

Our project addressed the majority of the key requirements as defined in the Project Scope section of our project proposal document. Each of the product user acceptance criteria conditions were met. The notable points of difference between the delivered product and these requirements and acceptance criteria is that we did not create an interface screen for manually editing the product data. Manual edits can instead be made by interfacing with the API directly (we used Postman). Another point of difference is that our API does not require any form of authentication. Security is something we consciously de-prioritised for convenience of development, testing purposes and to prioritise features we considered more interesting. We reasoned that it would be relatively easy to tighten security by leveraging features of the framework as part of future work. Overall we consider the project a success as the deliverables are aligned with the metrics of success that were stated in the project proposal.

## Limitations of project outcomes

1. For the scanner functionality, the barcode information may not be properly retrieved, that a part of the barcode may be interpreted instead of the entire one, if the camera is not close enough with the barcode or the barcode gets scanned from the side angles. The reason is mainly due to the library used not being capable of handling such special cases.

2. Users' Email Address is not editable once registered that could potentially affect the user experience.

3. Our current product information page on the Android App has relatively low usability based on the testing results received from participants.

4. Users can click on each nutrition attribute of a product to display its degree (e.g., whether it's high or low) based on the data of similar products. However the results of such are not accurate, and are highly dependent on the similar products that exist in our database, therefore could mislead users sometimes.

5. The product information that is submitted by users is not independently checked for errors or inaccuracies.

6. The OCR pipeline may produce incorrect results from certain types of images (e.g., noisy images, skewed angles, dark colours).

7. There is a margin of error with the accuracy of OCR results even for good quality images.

8. The front-end Android App hasn't been tested on all device sizes and configurations, therefore potentially the App may not present properly on some devices.

9. No authentication is required to make API requests for our existing system, therefore could cause significant data compromisation if the App gets widely used, as a part of our security limitations.

10. Relatively long response time as free tier of Heroku platform would go into sleep mode after a period of inactivity. This results in long waiting time on the front-end Android App if it's not being used actively for a certain amount of time.

11. The App was published to Google Play, which revealed the problem that the App can't be properly executed on an actual Android device if it's downloaded from Google Play, due to some issues regarding the release and debug (development) mode.

12. Our current data model for storing the "Category" of a product has potential data anomalies in terms of database design:
    I. *Insert Anomaly*: When adding "Category" to the "PRODUCT" table, it is required to add other product data like brand, etc…
    II. *Update Anomaly*: Changing the name for a "Category" requires multiple product instances to be changed.
    III. *Delete Anomaly*: When a product instance is deleted, such that the product is the only one of its category, and that category would also be removed if the product gets deleted. However in practice, we would usually prefer to store the category separately for future use and search.

    Therefore, it may result in data inefficiency or inconsistency when the App gets widely used.

13. Our data model for storing the "Nutrition" of a product is currently in a JSON format, which has potential similar data anomalies as outlined above in terms of database design. If in some cases we want to retrieve a particular nutrition attribute of a product, we will have to parse the entire product data, and its entire nutrition data to get such value, which is super inefficient. A lot of duplicate data such as nutrition names could also be avoided that could cause ineffectiveness and waste of storage when the App gets used by many users.

## Possible Improvements & Future Works

1. Our Android App barcode scanning can be potentially improved by customising the scanner based on another basic scanner library, and handling the special cases properly to avoid false retrieval of the barcode information.

2. Some of the UI screens still require optimisations in terms of usability, especially the product information page, in order to make it more aesthetically pleasing and intuitive to use to fulfil user satisfaction. Research on the relevant elements along with potential continuous testing may be carried out to support this.

3. The current formula to come up with the degree of a particular nutrition attribute of a product is very fundamental, and highly dependent on the similar products that are in the same category, lack of data to support the accuracy is also concerning. Therefore more study needs to be done in the relevant areas, especially how the nutritional degree of a product may be assessed, what factors need to be considered when making the arithmetic evaluation, etc… In order to provide better accuracy to avoid misleading users.

4. It's potentially needed to re-design the flow of manually putting the information of a new product on the front-end Android App in order to have a robust system, especially in terms of verifying user inputs on whether they are meaningful or consistent, and ideally provide a proper way to prompt them for optional verification on the nutrition information retrieved by our OCR, in order to make up the lack of its perfection that could result in false data in our database. The design also needs to take usability and user experience into account, therefore a lot of research, data collection and practices are expected to be carried out to support this.

5. Improve image preprocessing of OCR pipeline to detect and correct skewed images, to enhance OCR accuracy.

6. Improve row detection of OCR pipeline to handle labels that are located on the row above their values, to enhance OCR accuracy.

7. Our current XML implementation of UI screens need to be further tested in order to enhance device compatibility therefore making sure the App can be used by as many users as possible.

8. Implementing an administrative panel on the front-end Android App for authorised users may still be beneficial, as if the App gets used widely, by having such a page will make viewing the feedback provided by the users, editing false data and so on easier, therefore the system can be better managed especially without the unnecessary time consumption to manually make each request on Postman.

9. Authenticating each API request is of crucial importance in terms of enhancing the software security, as currently the API functions can be directly accessed from the public without any authentication, therefore could pose threats to the data. One way we have evaluated is to integrate session tokens into each API request, which will be assigned to each user instance whenever logged in, so that to make sure only the front-end Android App can make such requests as a user instance is required. However some back-end work needs to be done to verify user authentication regarding the third party logins as proposed by themselves in order to collaborate this.

10. Increasing the tier of Heroku server so that it does not go into “sleep mode” would improve response times. This would require additional monetary investment to pay for the added server resources and uptime.

11. Investigate differences between the debug and release mode of Android Apps by conducting relevant research, aiming to solve issues that prevent the application being properly executed on an actual Android smart device after being published to Google play.

12. Improvement on the database design, especially the PRODUCT table, in order to eliminate the data anomalies mentioned above in the limitations, and therefore deliver a better structure to avoid unnecessary storage, the extra time needed to query and the risks of losing data in some specific contexts.

# Methodology

## Design

The architecture of our project is graphically represented in Figure 1. The frontend is the Scannera Android application, developed using Java and XML. The Backend includes the web server, developed using the python Flask framework and hosted on Heroku. The web server used the SQLAlchemy python library to interact with the MySQL Database server which is hosted on a separate Heroku instance. The OCR Pipeline is a module contained in the web server.

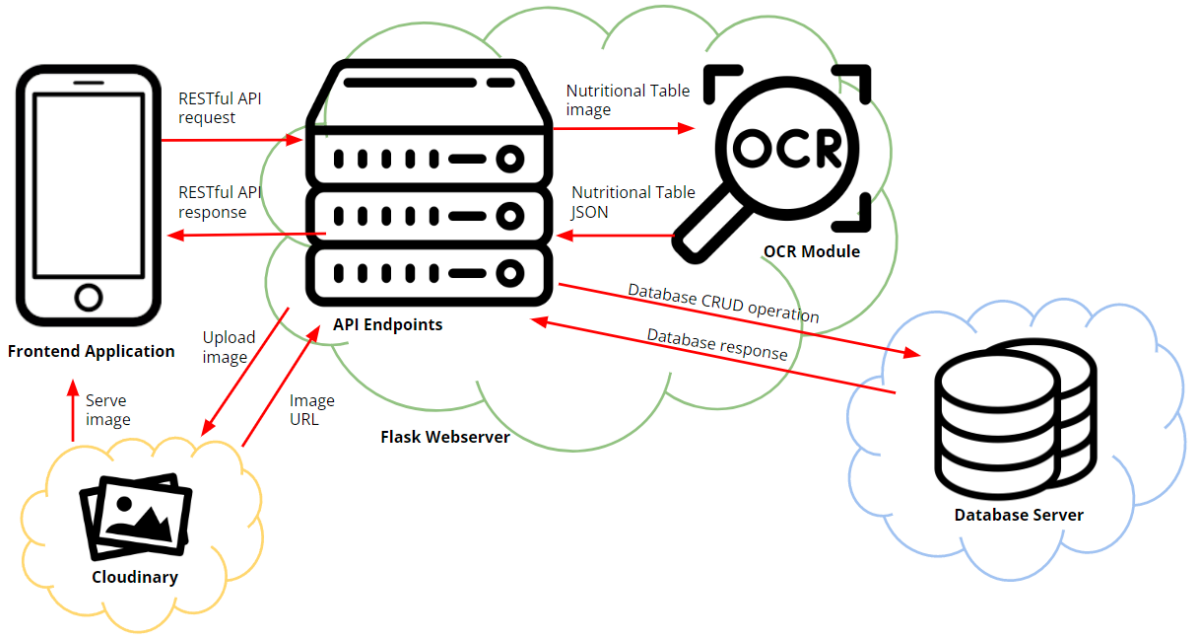


*Figure 1: Final Design Overview*

The mobile application uses RESTful HTTPS requests to communicate with the server. The server calls upon the OCR Pipeline module to process nutritional table images as required.

Create, Read, Update and Delete operations are constructed using SQLAlchemy and executed by the database server.

## Design Implementation

### ***Front-end***

The front-end Android App is fully implemented in Android Studio IDE. The implementation mainly consists of the XML code for each of the UI screens and its components, regarding their looks, styles, layouts, etc…, with the Java code that follows the Android framework, for linking up everything together by referencing the UI components from their respective XML files, and logically assign them functionalities to make the presented features possible, including the API requests to communicate with the back-end and other processings internally.

The following ***software tools or libraries*** have been used to support the implementation:

1. Retrofit which is a widely used library for Android API frameworks that can turn the HTTP API into a Java interface, for communicating with our back-end server.

2. A barcode scanner github library developed by a private group, based on ZXing library which is a widely used open source barcode image processing library, for our barcode scanning matters.

3. Android material components library which is an official library for Android Material design that is used to make the buttons, text fields, displayed errors etc… more aesthetically pleasing.

4. Facebook login for Android library to make the Facebook login possible.

5. Google Play Services to make the Google login possible.

6. Some other libraries for specific components have also been used, that are:
    I. Jakarta Android Mail API library for automatic email sending.
    II. Google Cloud Dialog Flow library as a back-end for our chatbot activity.
    III. Picasso, an Android library for image downloading and caching using URL, typically used for displaying our user or product images by having their URLs.
    IV. Lottie, for animations.

The Android App is implemented screen by screen, with the Java logical code of a particular screen built up once the corresponding XML code has been finalised, although potential changes would also be constantly made as to satisfy the design requirements. When making an API request to the back-end server, the required data would be encapsulated in the specified form with the API type defined, that will be instantiated by Retrofit, which provides us the response within its framework. The unsuccessful and failure responses will be handled respectively, and if it's a successful one, the data will be parsed if there's any, using either the JsonParser API or Retrofit itself, and the App will react accordingly. All the UI navigation, temporary data storage, data populating matters and so on are handled by following Android framework guidelines.

### ***Back-end (Server)***

The web server was developed in python using the Flask framework. We hosted the web server on Heroku and deployed using a separate remote branch. We created API endpoints by defining routes for each functional requirement (see Appendix K). These API endpoints were documented on the server landing page as they were developed, which assisted team members in using them correctly. We used the SQLAlchemy python library to interact with our remote MySQL database. We used Cloudinary as a third party host for all images and stored their urls instead of serving the files directly. Various environmental variables including our Cloudinary API key and Tesseract data path were loaded directly into our Heroku server.

### ***Back-end (OCR Pipeline)***

The OCR pipeline was built as a self-contained module and deployed on the web server. We built the OCR pipeline using the following libraries:

1. OpenCV
2. Tensorflow
3. Tesseract
4. Regular Expressions

We used OpenCV for image processing and manipulation, Tensorflow models for identifying the nutritional table with an image as well as the individual cells within the detected table, Tesseract for optical character recognition to extract the text from the identified cells and Regular Expressions for cleaning and parsing the extracted text. A custom python dictionary was implemented in order to facilitate many-to-one fuzzy mapping of nutrition table categories to consistent output keys. For example “Fat - Total”, “Fat” and “Total Fat” all map to the output key “fat-total”.

The steps for the OCR Pipeline are as follows when supplied an input image:

1. Apply thresholding to the image using binary & Otsu’s threshold algorithms in OpenCV.
2. Detect the bounds of the nutritional table within the image using a Tensorflow table detection model and crop the image to those bounds.
3. Detect each cell within the table using a Tensorflow text detection model.
4. Crop each cell and use Tesseract OCR to extract the text and store with its location.
5. Parse every cell text using regular expressions and fuzzy matching to find the nutritional table categories and their output keys.
6. Match each category with its corresponding value cell by locating the right-most cell on the same row and use regular expressions to split the numerical value from the unit.
7. Add the value and unit to the output dictionary under the corresponding output key for that fuzzy-matched category.

### ***Back-end (Database)***

We used a remote MySQL database that was hosted on Heroku. We used the SQLAlchemy python framework for all interactions with the remote database from our web server. We

configured the database by connecting to the remote database server using an engine and binding the engine to a session factory that would handle the creation and destruction of database sessions (see Appendix E). We mapped python classes to tables on the relational database and defined the classes based on our previously designed conceptual model (see Appendix F). Create, Read, Update and Delete operations were performed inside transactions encapsulated in a session (see Appendix G, H, I & J respectively). The software DBeaver was used to validate database structure and data after constructing the database and making changes.

# Software Deliverables

## Front-end (Android App)

The front-end source code mainly consists of the XML resource files and the Java files, that together deliver the UI pages with the associated features that are presented in each of the screens. Below is an overview of the delivered Android App.

1. **A login page:** Which allows users to login with his/her registered Scannera account, or Facebook / Google just like normal Apps do. Entries to sign up a new account or reset the password if forgotten are also provided.

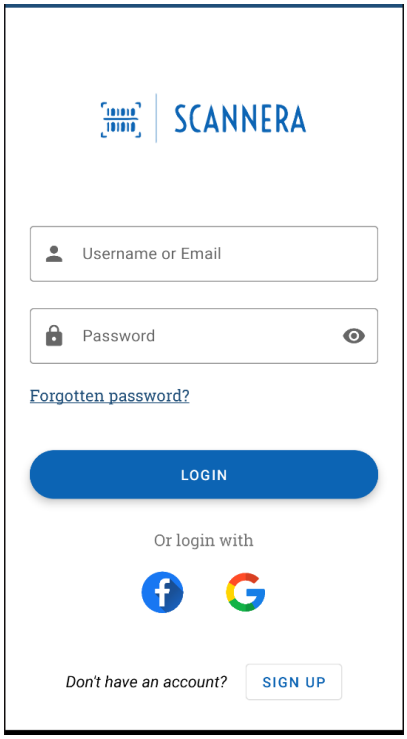


*Figure 2: Scannera Login Page*

2. **A Sign Up page:** Which allows users to register a new Scannera account. This activity also handles false user inputs for all fields and will respond correspondingly.

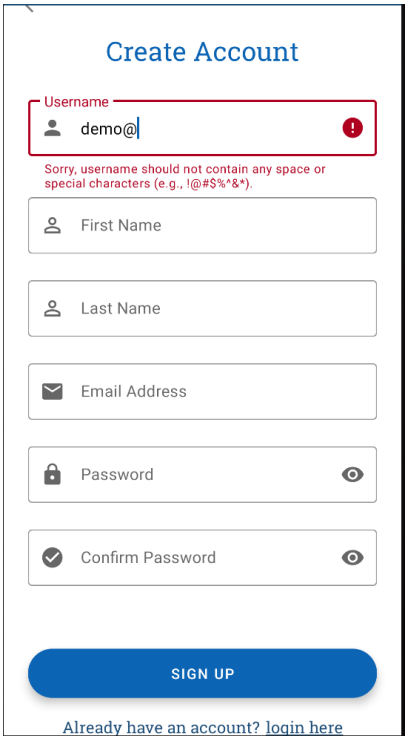


*Figure 3: Scannera Sign Up Page*

3. **A Forgotten Password page:** Which allows users to reset their passwords if forgotten, by entering the associated Email Address, an email with the temporary password and the reset instructions will be automatically sent to the provided Email Address. The page for handling successful “Email Sent” is also implemented and will be displayed on email sent if a valid and registered Email Address is entered.

4. **A Home page:** Which recommends products to users based on their starred products or scanned products if there are any, otherwise will display random products.

5. **An Account page:** Which displays a list of buttons (options) the user can do that are related to his/her account or the App itself. It also displays the user’s username, profile image and contribution score (Like rewards based on their activities within the App).

6. **An scanner page:** Which is for scanning a product barcode, that will automatically navigate to the relevant product information page if the product associated with the scanned barcode exists in the database, otherwise will display a dialog like in the figure below to prompt the user on whether he/she wants to contribute by entering the product information manually.

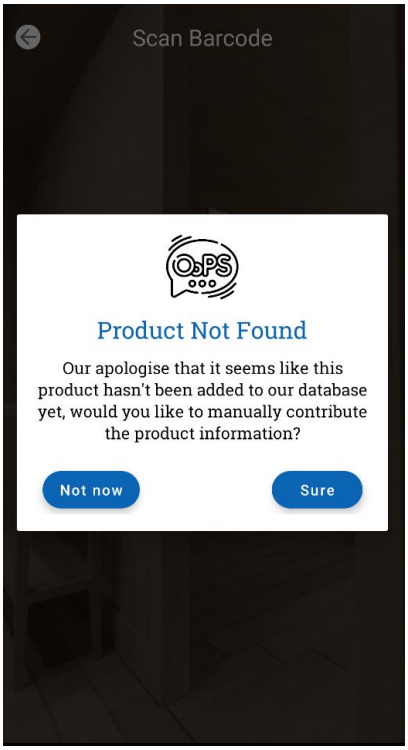


*Figure 4: PNF dialog*

7. **A Product Information page:** Which displays the relevant product information and contains a set of interactive features:
    I. “Star” (or called favourite) the product.
    II. Horizontally scroll the nutrition information section.
    III. Click any of the nutrition attributes to check its degree based on the similar products.
    IV. Click on the product image to view its enlarged version.
    V. Sort the similar products by selecting a factor.
    VI. Specify the order the similar products to be sorted.
    VII. View the product reviews posted by other users.
    VIII. Write a review for the particular product.
    IX. Navigate directly back to the home page.


*Figure 5: Product information page*

8. **An Account Information page:** Which allows users to view their account details, or edit any except the Email Address if they wish to, including the profile image. The profile image can be updated by either selecting from the album or taking an immediate photo. The page also includes a “DELETE MY ACCOUNT” button for users to delete their accounts.

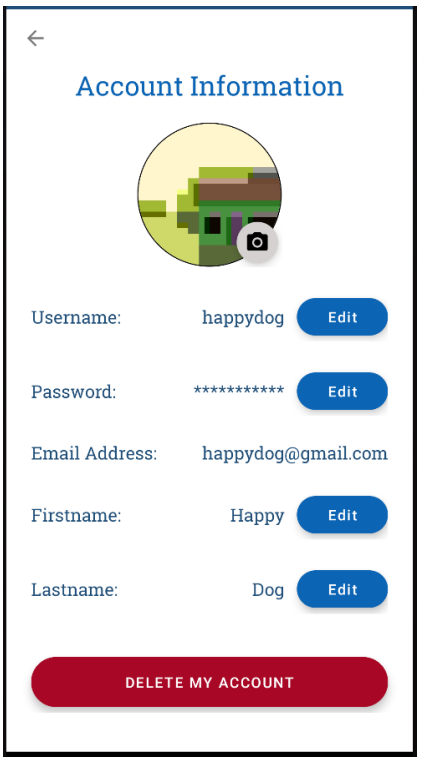


*Figure 6: Account Information page*

9. **A Starred Products page:** Displays the products starred by the user.

10. **A Scan History page:** Displays the products (max of 20) recently scanned by the user, with the timeline of the scan date indicated.

11. **A Chat Bot page:** Which allows users to ask any app specific questions, like “how to scan” or “How to delete my account”, etc… Some small talks are also supported, and will respond with a corresponding predefined message by parsing the user messages using machine learning techniques provided by a third party called “DialogFlow”.

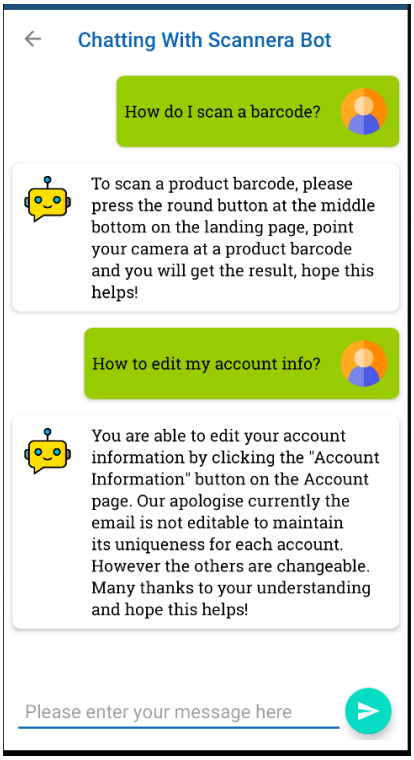


*Figure 7: Chatbot page*

12. **An About Us page:** Which includes a brief description of our project and a button that takes users to provide feedback to us. It also includes the buttons for “Privacy Policy” and “Terms And Conditions”.

On top of the aforementioned deliverables, the App also contains other fundamental features like handling failure requests, unexpected code behaviours, user permissions, internal processings, etc… to enhance robustness and user experience. Additionally it also contains a XML file of Chinese translations of each string to support Chinese language. A more detailed description of the Android App in terms of the user perspectives is included in the user guide.

## Back-end (Server)

The web server delivers the API endpoints that support the requirements of the front-end application. The front end application interacts with the API by making HTTPS RESTful requests and providing any required data in the body of the request or as a URL parameter. The url structure is defined using Flask Blueprints (see Appendix L) that iteratively build up the prefix structure. The endpoint routes are configured using python decorators to set the url and allowed methods with the subsequent function defining the behaviour of that endpoint (see Appendix K).

The OCR Pipeline has been implemented as a module within the web server. It serves the purpose of extracting the JSON representation of nutritional table data from a supplied image.

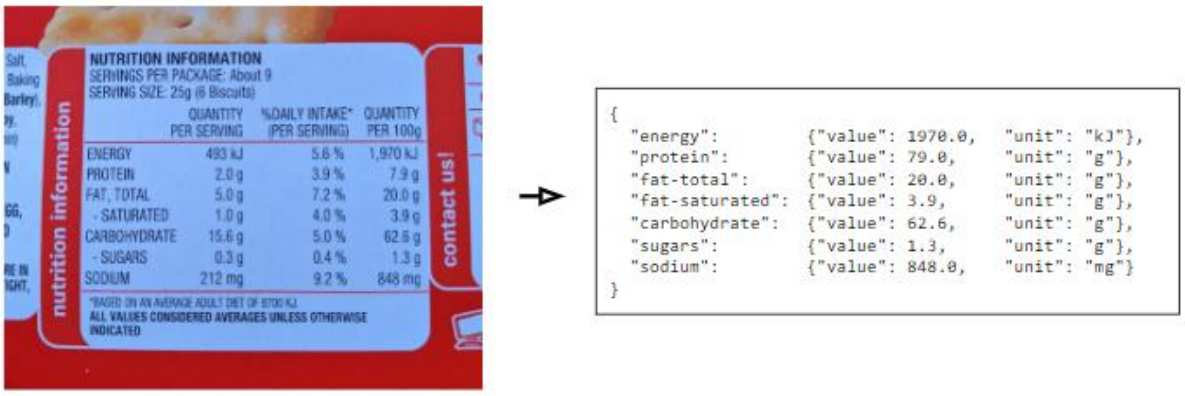


*Figure 8: OCR Pipeline Overview*

## Back-end (Database)

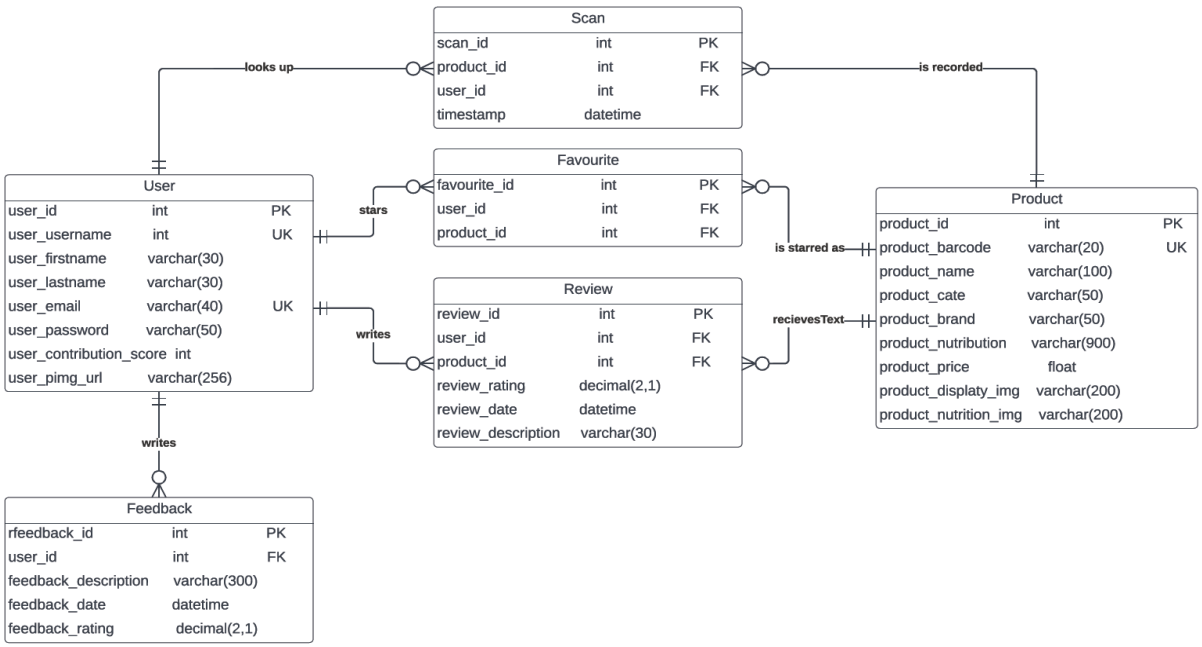


*Figure 9: Database Conceptual Model*

For our back-end database, we have designed and come up with the corresponding conceptual model for our project as our back-end data structure for our data storage. The database server will be interacted with our back-end Flask server by calling the functions that match the CRUD operations. Our database has been constructed based on the conceptual model above and the database server has been connected with the Flask server using the SQLAlchemy framework in Python. The database server has been deployed to Heroku and hence it is constantly listening to CRUD operations being declared from the Flask server.

## Sample Code Explanation

### Front end

**Appendix A:** The Android App will automatically save the user details locally on successful login to avoid future logins, unless the user has logged out, deleted his/her account or cleared up the App storage externally. To achieve this it stores the user details as a JSON string inside an Android Shared Preferences with a pre-defined key, so that later on by calling the function shown in Appendix A called “getLoggedUser”, it will return an user instance with the relevant details stored inside, by converting the string into the user instance using the Gson library.

**Appendix B:** In order for the front-end to communicate with the back end for data, we are using **Retrofit** which is a widely used library for Android API frameworks that can turn the HTTP API into a Java interface. The function shown in Appendix B is to build a Retrofit instance with our back-end base URL, some customised settings, and the Java interface which contains all the necessary API request definitions, that will be called by all the API requests later on, to achieve the data transmission.

**Appendix C:** The function shown is an instance of the API definition in the Java interface mentioned previously, which is to get the product of the barcode parameter in its JSON format pre-defined by us. It also takes the user id in as we want to record the scan history for the user.

**Appendix D:** The code snippet presented in Appendix D is like the client side that calls the API function mentioned above. The necessary parameters are provided in the context and the response will be captured in the specified format (in this case, ResponseBody) using the Retrofit framework. Successful response and failure response are handled correspondingly.

### Back end

**Appendix E:** The code snippet presented creates an engine and a session factory for connecting with our database server. The engine connects and interacts with the database server directly as a middleware and the session establishes a stateless transaction based on the connection created for some query. The base class is created for the extension of model classes definitions shown in Appendix E.

**Appendix F:** The code snippet presented shows an example of model classes definitions that extends the base class as superclass. Each model class need to be created and defined by specifying according table name and defining each property with some constraints being set up to avoid problematic queries regarding the specific cases that can cause failures to the server or the front-end, such as “nullable=False” and “delete='CASCADE'”. Relationships are also defined by setting foreign keys in the class definition.

**Appendix G:** The code snippet shows how the CREATE operation in the database executed using SQLAlchemy.

**Appendix H:** The code snippet shows how the RETRIEVE operation in the database executed using SQLAlchemy.

**Appendix I:** The code snippet shows how the UPDATE operation in the database is executed using SQLAlchemy.

**Appendix J:** The code snippet shows how the DELETE operation in the database is executed using SQLAlchemy.

**Appendix K:** The code snippet is an example of an API endpoint on the web server using the Flask framework. The python decorator on the first line defines the url and allowed methods. This endpoint returns a user that matches to the given user_id supplied as part of the URL. If no matching user is found the response is None.

**Appendix L:** The code snippet showcases using blueprints to iteratively build up the url structure by registering first a blueprint to the app context then subsequent blueprints are registered to blueprints up the chain. This lets us reference the `user` blueprint in a decorator such as `@user.route('/get')` which would register for the the url: `<server_address>/api/user/get`

## Software qualities

### ***Robustness***

The front-end App follows Android framework guidelines which provides a high degree of inherent robustness. The inputs coming from the UI are validated therefore false or unwanted data will be rejected with an error displayed to the users. The API requests are handled using the Retrofit framework which provides a way to capture and deal with unsuccessful requests. The data received from the back-end is parsed safely to ensure stability in the case of malformed data. The code blocks that may throw exceptions are handled by try and catch blocks to deal with failures.

We have also conducted various tests including unit testing, black-box testing, integration testing and so on to consolidate the robustness of our software as a whole. Detected problems were cleared up soon after.

Requests made to the web server are checked for expected failure cases and respond to them with an error and debug message (see Appendix M). This ensures that when errors are

encountered the frontend can respond to the user accordingly and advise them how to proceed.

However, the front-end Android App itself alone isn't capable of handling respond-less requests, in which case it will wait endlessly, as a potential point of lack of robustness to be resolved in the future.

### ***Security***

The front-end App is secure now as the source code is private. Although it has been published to Google Play, the credentials like the App's release keystore that are required to modify any of the App settings are managed directly by Google itself, and the uploaded source code bundle has also been minified by ProGuard, which shrinks and obfuscates the source code for protection. Therefore nobody can access and modify the Android App unless authorised, or the credentials are lost and the data is compromised.

The Cloudinary service uses a private API key, which is stored as an environmental variable on the heroku server. In terms of our database security, we employed ClearDB service for storing and managing our MySQL database remotely, which ensures that access to our data requires authorised credentials.

However, there is no authentication required to make API requests for our back-end server, meaning the public can access and manipulate with our data if the API endpoints are acquired. Our strategy to address this problem is by synchronising API requests with session tokens, which will be assigned to each front-end user once logged in, as a part of each API request, as mentioned in our future work above.

A security consideration is that our database URL is exposed through our Git repository. This issue could be resolved by removing the URL from the repository, deleting the exposed login credential, creating a new connection URL with fresh credentials and storing the URL as an environmental variable on the Heroku server.

### ***Usability***

Prior to implementing a UI page on the front-end App, a big picture of the design outcome would have been come up with, that will be evaluated through the research conducted, to ensure the consistency with the similar pages thereby avoiding potential usability confusions. The design of each UI component is aimed to be as simple as possible over fancy looks, to avoid potential violation of the proposed usability rules. We have also attempted to integrate the usability attributes we absorbed from our literature review into the design, to maximise the efficiency and effectiveness of performing a task through the App therefore enhancing user satisfaction. We have also conducted usability testing at a later stage to consolidate the front-end UI and fixed a number of relevant issues throughout the App.

However, usability issues still exist in our current software due to various reasons. It could be our lack of knowledge to implement a particular UI screen in an aesthetically pleasing manner due to the required components and features to be contained, or potentially compromised due to the attributes of the requirements.

An example would be the page where users manually enter the information of a new product, that no dialog is provided to yield closure, thereby may impact user experience when performing the job as they don't know when the process finishes, especially when they are in a shopping context with a hurry mind.

Another example is the page that displays the information of a product. Many participants from usability testing have implicitly complained about the layout of the page, especially the way the nutrition list is designed, and it takes a noticeable amount of delays for them to perform a particular task on that page.

There are certainly various improvements that can be carried out, and as the App gets used widely, some potential usability issues will also arise and have to be addressed to enhance user satisfaction, otherwise the project success would have been compromised.

### ***Scalability***

We have built our software in a way that allows for future scalability. By implementing our web server database as separate Heroku instances, we can increase the capacity of the web server and database server independently of each other, to address different scalability issues. If more concurrent connections to the API are required, then the capacity of the web server Heroku instance can be scaled up to a higher tier with more memory. If additional concurrent read/write operations are required for the database, then the database server Heroku instance can be scaled up and duplicated or sharded as required.

### ***Documentation and Maintainability***

Each of the back-end API endpoints has been documented with descriptions on the server landing page for clarification. Each of the functions or files throughout the entire system (front-end & back-end) are named in a meaningful and clear manner for easy lookup and reference. We have also attempted to manage duplicate code by assigning a function for it, and so only the function needs to be called instead of a whole block of code repetition. Any string, dimension or other resources on the Android side, that would be used multiple times are stored in their corresponding XML files, like strings.xml, that will be referenced for usage, therefore whenever there needs to be a change for a particular resource instance, we only need to change once in its definition. The OCR algorithm has been fully documented with inline comments wherever appropriate.

However, there's a significant lack of inline comments or documentation regarding the front-end Android App and the back-end server functions, since we highly relied on the respective developer(s) to illustrate whenever needed instead of looking up by ourselves, the importance of such documentation has been underestimated, and have caused a noticeable amount of confusion that prevented the other developers from proceeding some further work during the development. Making up for the lack of documentation for those parts is also of crucial importance as the software grows.

# Overall Critical Discussion

As stated in the Discussion of all results subsection of the Outcome section of this report, we verified that we have delivered the majority of the key project requirements and have met the project user acceptance criteria. We recognise that we have not implemented complete secure authentication and have justified this decision by reasoning that we could improve this later as part of future work. The editing of product information by authorised users was implemented as part of the API but we did not create an interface in the application. We initially planned to create an interface yet decided against it during execution. We justified this decision by reasoning that the only authorised users were the developers at this stage and we were all comfortable making requests directly.

Initially we theorised that we could use OCR to capture the product name, description and brand attributes in addition to the nutritional table. After some further thought and a discussion with the lecturer we realised that it would be better to limit the OCR to the nutritional table information. We reasoned that the OCR would be most beneficial to capture the lengthy nutrition table information and that the other attributes could be entered manually by users without much difficulty.

We identified during project execution that we did not plan some parts of our design precisely enough. This was evident early on by the lack of exact definitions for our variables and naming conventions between the separate code bases of our project. For example when we began communicating between the application and server we found inconsistencies between the attribute names of a product class on the server and a product class on the application. This caused some confusion and was the source of some programming mistakes as developers had to remember which namespace to use across multiple contexts. If we had our time again we would invest more time at the beginning to precisely and consistently document the structure and signatures of all our classes and methods.

# Conclusion

The primary goal of this project was to help supermarket customers make quick and easy informed nutritional choices at the supermarket. We understood that whilst nutritional labels work well to inform the customer of what's in a particular product, they are not well suited to comparing many different products quickly and simultaneously.

Scannera serves as a way for customers to quickly and easily identify products that meet their nutritional needs. It was delivered on time, addresses the key project requirements and meets the user acceptance criteria. There were areas of the project execution that we identified that we could have improved upon and we will apply those lessons in the future. Overall we consider the project a success based on our personal experiences completing it and upon reflecting against what we set out to accomplish.

# Appendix

## Appendix A

```
249 @       public static User getLoggedUser(Context context)
250         {
251             Gson gson = new Gson();
252             Type type = new TypeToken<User>() {}.getType();
253             User user =  gson.fromJson(context.getSharedPreferences(Utils.APP_LOCAL_SP,  mode: 0)
254                     .getString(Utils.LOGGED_USER,  defValue: null), type);
255
256             return user != null ? user : new User();
257         }
```

## Appendix B

```
279 @   public static Retrofit getRetrofit(Context context)
280     {
281         OkHttpClient okHttpClient = new OkHttpClient().newBuilder().connectTimeout( timeout: 60, TimeUnit.SECONDS)
282                 .readTimeout( timeout: 60, TimeUnit.SECONDS).writeTimeout( timeout: 60, TimeUnit.SECONDS).build();
283
284         return new Retrofit.Builder().baseUrl("https://fit3162-app-group7.herokuapp.com/").addConverterFactory(GsonConverterFactory.create())
285                 .client(okHttpClient).build();
286     }
287
288 @   public static ServerRetrofitAPI getServerAPI(Context context)
289     {
290         return getRetrofit(context).create(ServerRetrofitAPI.class);
291     }
```

## Appendix C

```
136         @GET(GET_PRODUCT_SERVER + "{product_barcode}")
137         Call<ResponseBody> getASingleProduct(
138                 @Path("product_barcode") String product_barcode,
139                 @Query(USER_ID_SERVER) String user_id
140         );
```

## Appendix D

```
136         Call<ResponseBody> call = Utils.getServerAPI( context: this).getASingleProduct(product_barcode, user.getId());
137         call.enqueue(new Callback<ResponseBody>() {
138             @Override
139 o↑          public void onResponse(@NonNull Call<ResponseBody> call, @NonNull Response<ResponseBody> response) {
140                 if (loading_dialog.isShowing())
141                     loading_dialog.dismiss();
142
143                 if(response.isSuccessful() && response.body() != null)
144                 {
```

## Appendix E

```
engine = create_engine(REMOTE_DB_URL)
db_session = scoped_session(sessionmaker(bind=engine))
Base = declarative_base()
```

## Appendix F

```
@dataclass
class Review(Base):
    review_id: int
    user_id: int
    product_id: int
    review_rating: float
    review_date: str
    review_description: str

    __tablename__ = 'Review'
    review_id = Column(Integer, primary_key=True)
    user_id = Column(Integer, ForeignKey("User.user_id", ondelete='CASCADE'), nullable=False)
    product_id = Column(Integer, ForeignKey("Product.product_id", ondelete='CASCADE'), nullable=False)
    review_rating = Column(DECIMAL(2, 1), nullable=False)
    review_date = Column(DateTime(timezone=True), default=func.now(), nullable=False)
    review_description = Column(String(300), nullable=False)
```

## Appendix G

```
with db_session() as session:
    new_prod = Product(
        product_name=name,
        product_brand=brand,
        product_cate=category,
        product_barcode=barcode,
        product_price=price,
        product_nutrition=nutrition,
        product_display_img=display_img_url,
        product_nutrition_img=nutrition_img_url)
    session.add(new_prod)
    session.commit()
```

## Appendix H

```
products = Product.query.all()   # Query all product from the database
```

## Appendix I

```
updated_product = session.query(Product).filter_by(product_barcode=barcode).first()  # Query the product
updated_product.product_name = name  # update its name
session.commit()
```

## Appendix J

```
_product = session.query(Product).get(product_id)   # Query the product
session.delete(_product)    # Delete the product
session.commit()
```

## Appendix K

```
@user.route('/get/<user_id>', methods=['GET'])
def get_user_by_id(user_id):
    """
    Returns a User by the given user_id in the URL or None
    """
    user_match = User.query.get(user_id)
    return jsonify(user_match)
```

## Appendix L

```
user = Blueprint('user', __name__)
api = Blueprint('api', __name__)
api.register_blueprint(user, url_prefix='/user')
app.register_blueprint(api, url_prefix='/api')
```

## Appendix M

```
# check to ensure record for barcode does not exist in database
match = Product.query.filter_by(product_barcode=barcode).first()
if match is None:
# barcode does not exist, add the product
...
else:
    return jsonify({"status": "error", "message": "barcode already
exists"})
```

# References


Serrador, P., & Pinto, J. K. (2015). Does Agile work? — A quantitative analysis of agile project success. *International Journal of Project Management*, *33*(5), 1040-1051. https://doi.org/10.1016/j.ijproman.2015.01.006

Thesing, T., Feldmann, C., & Burchardt, M. (2021). Agile versus Waterfall Project Management: Decision model for selecting the appropriate approach to a project. *Procedia Computer Science*, *181*, 746-756. https://doi.org/10.1016/j.procs.2021.01.227

Špundak, M. (2014). Mixed Agile/Traditional Project Management Methodology - Reality or Illusion?. *Procedia - Social and Behavioral Sciences*, *119*, 939-948. https://doi.org/10.1016/j.sbspro.2014.03.105

Imani, T., Nakano, M., & Anantatmula, V. (2017). Does a Hybrid Approach of Agile and Plan-Driven Methods Work Better for IT System Development Projects?. *International Journal of Engineering Research and Applications*, *07*(03), 39-46. https://doi.org/10.9790/9622-0703043946

Nayebi, F., Desharnais, J., & Abran, A. (2012). The state of the art of mobile application usability evaluation. *2012 25th IEEE Canadian Conference on Electrical and Computer Engineering (CCECE)*, 1-4. https://doi.org/10.1109/CCECE.2012.6334930

Ayob, N. Z. B., Hussin, A. R. C., & Dahlan, H. M. (2009). Three Layers Design Guidelines for Mobile Application. *2009 International Conference on Information Management and Engineering*, 427-431. https://doi.org/10.1109/ICIME.2009.99

The International Organization for Standardization. (2018). *Ergonomics of human-system interaction — Part 11: Usability: Definitions and concepts*. https://www.iso.org/obp/ui/#iso:std:iso:9241:-11:ed-2:v1:en

Shitkova, M., Holler, J., Heide, T., Clever, N., & Becker, J. (2015). Towards Usability Guidelines for Mobile Websites and Applications. *Wirtschaftsinformatik Proc. 2015*. 1603-1617. http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.1040.7687&rep=rep1&type=pdf

B, D. D., Salim, S., & Vargese, S. M. (2016). Mongodb Vs Mysql: A Comparative Study Of Performance In Super Market Management System. *International Journal of Computational Science and Information Technology, 4(2),* 31–38. https://doi.org/10.5121/IJCSITY.2016.4204

Čerešňák, R., & Kvet, M. (2019). Comparison of query performance in relational a non-relation databases. *Transportation Research Procedia, 40,* 170-177. https://doi.org/10.1016/j.trpro.2019.07.027

Seo, J.Y., Lee, D.W., & Lee, H.M. (2017). Performance Comparison of CRUD Operations in IoT based Big Data Computing. *International Journal on Advanced Science, Engineering and Information Technology, 7(*5*)*, 1765-1770. https://doi.org/10.18517/ijaseit.7.5.2674

Schwaber, K. (1997). SCRUM Development Process. *Business Object Design and Implementation,* 117-134. https://doi.org/10.1007/978-1-4471-0947-1_11

Srivastava, A., Bhardwaj, S., & Saraswat, S. (2017). SCRUM model for agile methodology. *2017 International Conference on Computing, Communication and Automation (ICCCA),* 864-869. https://doi.org/10.1109/CCAA.2017.8229928

Cho, J. (2008). ISSUES AND CHALLENGES OF AGILE SOFTWARE DEVELOPMENT WITH SCRUM. *Issues in Information Systems, 9*(2), 188-195. https://doi.org/10.48009/2_iis_2008_188-195

Ahmad, M. O., Markkula, J., & Oivo, M. (2013). Kanban in software development: A systematic literature review. *2013 39th Euromicro Conference on Software Engineering and Advanced Applications,* 9-16. https://doi.org/10.1109/SEAA.2013.28

Ikonen, M., Pirinen, E., Fagerholm, F., Kettunen, P., & Abrahamsson, P. (2011). On the Impact of Kanban on Software Project Work: An Empirical Case Study Investigation. *2011 16th IEEE International Conference on Engineering of Complex Computer Systems,* 305-314. https://doi.org/10.1109/ICECCS.2011.37

Weichbroth, P. (2020). Usability of Mobile Applications: A Systematic Literature Study. *IEEE Access, 8*, 55563-55577. https://doi.org/10.1109/ACCESS.2020.2981892

Hertzum, M. (2010). Images of Usability. *International Journal of Human-Computer Interaction, 26*(6), 567-600. https://doi.org/10.1080/10447311003781300

Rautmare, S. & Bhalerao, D.M. (2016). MySQL and NoSQL database comparison for IoT application. *2016 IEEE International Conference on Advances in Computer Applications (ICACA)*, 235-238. https://doi.org/10.1109/ICACA.2016.7887957

Ch, S., Mahna, S., & Kashyap, N. (2015). Optical Character Recognition on Handheld Devices. *International Journal of Computer Applications*, *115*(22), 10-13. https://doi.org/10.5120/20281-2833

Brisinello, M., Grbic, R., Stefanovic, D., & Peckai-Kovac, R. (2018). Optical Character Recognition on images with colorful background. *2018 IEEE 8th International Conference on Consumer Electronics - Berlin (ICCE-Berlin)*. https://doi.org/10.1109/ICCE-Berlin.2018.8576202