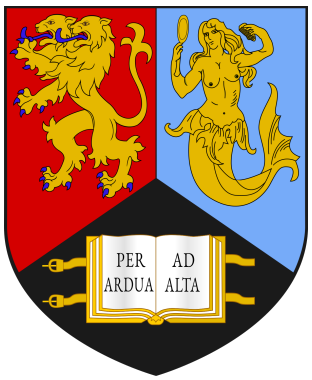



University of Birmingham

Department of Computer Science

# Developing a cross-platform application and implementing Object Detection to help reduces household food wastage

Jian Choong Liew

2053492

*Supervisor:* Leandro Minku

Bachelor of Science in *Computer Science*

**Report's total word count: 8311 words**

# Abstract

Around 33% of all food produced globally is wasted every year, causing resource depletion and severe environmental impact. Food waste are produced in all stages of the food supply chain, but private household has been identified as the key contributor for food waste generation.To successfully tackle household food wastage, a brief literature review is done to identify the factors and gather the functional requirements. With this, the project aims to design and develop a cross-platform application that provides alternative ways to handle excess food, in order to investigate the impact of technology on reducing household food wastage,. Besides that, it also trains a detection model for ingredients recognition, and discuss how it could be utilized to speed up the recipe search procedure. This work explores the possibilities of reducing food waste through a technological approach and exhibits the potential of machine learning in reducing household food waste generation.

# Contents

# Chapter 1

# Introduction

Food waste are edible food that was being discarded at any point of the supply chain, this includes food wasted during manufacturing, processing, packaging, by retails and households around the country. According to an analysis done by Gustavsson et al. (2011), household food waste contributes to 70% of the total UK's food waste. The amount of edible food thrown away was enough to create 10.5 billion meals.

The main factors that leads to household food wastage are overbuying, poor management of ingredients and the lack of guidance to handle excess ingredients. As humans, we are bad at memorizing the exact amount of remaining ingredients in the kitchen, and tend to buy more than what we could finish before the food itself goes bad. Moreover, we are often limited by the dishes that we know to fully utilized the excess ingredients. Additionally, disposing them seems to be the first option for us whenever we can't finish food before its expiry date. If people are provided with alternative ways to manage and handle these ingredients, it would help reduces the amount of food wastage.

With the advancement of technology, mobile phones are no longer just machines that enable us to make calls or texts. Services that were once only accessed through PC are now being provided to mobile users through smart phone applications. As technology becomes an inseparable part of our daily life, accessing online services from different platforms have never been easier. Thus, we look into the potential of building a cross platform application to deal with household food waste.

There are multiple applications that aim to reduce food waste out there. An example of that is Olio, which allows users to share their ingredients or full-blown meal with people around. Besides that, TooGoodToGo works with food service providers to sell their food surplus that would otherwise be discarded at the end of the day. The distinctive feature of our application is that we venture to apply an object detection model on this issue.

Object detection is a computer vision technology that deals with identifying certain objects from the image inputted and determines their respective class and position. Utilizing this, we implement a model that can quickly recognize the ingredients an image contains. This speeds up the process of inputting ingredients while searching for recipes with excess ingredients.

This project aims to reduce household food wastage via providing accessible alternative ways for user to deal with excess food. Specifically, by assisting user on managing their ingredients, sharing excess food through food listing and searching for recipes with a cross-platform

application, *FoodXcess*. It also explores the possibilities of integrating object detection model to tackle this issue, by experimenting with a model trained for detecting fruits, to ease the recipe search process.

**Git repository for the project:** `https://git-teaching.cs.bham.ac.uk/mod-ug-proj-2021/jcl992`

# Chapter 2

# Literature Review and Related Work

### Household Food Wastage

Given the amount of resources invested into food production and supply chain, food wastage indirectly causes negative impact on the economic and environment, with greenhouse gas emitted during food manufacturing, storage and transportation being the most obvious example (Mourad, 2016). Along the food production-consumption chain, private household has been identified as the main actor for generating food waste(Parfitt et al., 2010).

Hence, it is important for us to identify the possible solution to tackle this issue. A study by Parizeau et al. (2015) shows that planning before grocery shopping is effective to prevent overbuying and consequently, food waste. Farr-Wharton et al. (2014) further states that keeping track of food items at home when shopping is crucial to avoid purchasing unnecessary items. Besides that, the idea of "food-waste cooking", which is cooking based on what is found at home helps prevent food waste (Watson and Meah, 2012). However, this is often hindered by the knowledge of recipe to utilize those ingredients (Evans, 2012). Finally, despite not being a common practice, redistribution of excess food or ingredients is considered one of the most effective strategy to avoid food waste.

This shows that there are possible ways that we could help reduces food wastage, especially with the intervention of technology. This project focus on building an application that allows user to manage their food inventory, searches for recipes thru text or image input and commonizing food sharing via the food listing function.

### Object Detection

Object detection(OD) is a computer vision task that deals with locating and classifying certain classes of objects in digital image or videos. With integration of deep learning techniques, OD models are able to learn high-level and deeper features, and now being commonly applied for autonomous driving (Chen et al., 2015) and face recognition (Yang and Nevatia, 2016). OD can be grouped into two genre: "two-stage detection" and "one-stage detection", former generates region proposals and classifying them into different classes afterward, while latter directly achieves the locations and classes via regression/classification-based methods. Some examples of two-staged detectors are Regional Convolutional Neural Network(R-CNN) (Girshick et al., 2014) and Faster R-CNN(FRCNN) (Ren et al., 2015). Meanwhile, one stage detectors include YOLO(Redmon et al., 2016) and Single Shot MultiBox Detector(SSD)(Liu et al., 2016).

The key difference between these two is the trade-off between speed and accuracy. As one-stage detectors apply a single neural network to divide image into regions and predict bounding boxes and probabilities for each region simultaneously, their detection speed are much faster. However, this also reduces their localization accuracy.(Zhao et al., 2019).

Generally, the effectiveness of OD is evaluated using "Average Precision"(AP) and mean-AP(mAP) metrics. AP is defined as the average precision under different recall values, and it's usually class specific. AP for each class is then used to calculate the mAP, to evaluate the performance over all classes. For localization accuracy, "Intersection over union"(IoU) measures the overlap between the prediction and ground truth, and compare with predefined threshold to classify the prediction.

As user gets discouraged easily by poor usability, we employ OD to ease the ingredient input process. Due to the advantages of different models, we train FRCNN and SSD using a synthetic fruit dataset, and evaluate them using dataset that consists of real-world fruits images to select the model that best suits our objective.

**Related Work**

OLIO is a mobile application that aims to reduce food waste through food-sharing. It provides a platform for people with surplus food to connect with others who need or want to consume such food. The food shared can be in any form, as long as it is edible. Overall, Olio is a great food sharing platform, but its functionality is unitary and heavily community driven. For instance, a listed ingredient that got no response is highly likely to get discarded, hence food waste is still generated.

TooGoodToGo is an application that helps food service providers to sell their unsold product which would originally be discarded at the end of the day. In return, users will be able to get food that may worth three times the amount they paid for. It is a win-win solution, as users are able to get cheaper food and retailers avoid food lost. Despite that, it is a business-facing application which does not really correlates with household food waste.

In conclusion, services that both applications provide has successfully assisted users on reducing food wastage. However, our project aims to develop a one-stop cross platform application that offers multiple functionalities to avoid household food waste generation. Besides that, we also attempt on tackling this issue with a unique machine learning approach, as we believe that it has immense potential in food-related field.

# Chapter 3

# Requirement and Software Design

## 3.1 Requirement Specification

*FoodXcess* is based on the functional design approach, which allows the design to be easily explained through its workflow, use cases and modular implementation of system components. After reviewing food waste from a scholarly perspective, we are able to identify the main functionalities needed for the application to reduce food waste practically. Furthermore, gaining a deeper insight from other applications with similar aim helps further polish the requirements of *FoodXcess*. This section lists out all the functional and non-functional requirements of the application. It gives a detailed clear description of all system components and their functionalities.

### 3.1.1 Functional Requirements

#### Authentication

R.1.1 User must be able to create an account.
R.1.2 System must validate the username and password during sign up.
R.1.3 User must be able to sign in with correct credentials.
R.1.4 System must alert user when credentials are incorrect.
R.1.5 User must be able to sign out.

#### AI Search

R.2.1 System must acquire user's permission to access gallery.
R.2.2 System must acquire user's permission to access camera.
R.2.3 User must be able to upload image from gallery.
R.2.4 User must be able to upload image taken on camera.
R.2.5 System must be able to run inference on image uploaded.
R.2.6 System must return response with ingredients that the image contains.
R.2.7 User should be able to add or remove ingredients on the confirmation screen.
R.2.8 User should be able to search for recipes using listed ingredients.

#### Recipe Search

R.3.1 User should be able search by inputting the ingredients.
R.3.2 System must fetch the request to obtain relevant recipes.
R.3.3 System must display the search results.

R.3.4 System should display the name, calories and the ingredients used for each recipe.
R.3.5 System should redirect user to the full recipe that they are interested in.

#### Food Listing

R.4.1 System must display food listings created by other users.
R.4.2 System must display title, description and expiry date(if any) of each food listing.
R.4.3 User must be able to view the details of interested food listing.
R.4.4 User must be able to create new food listing.
R.4.5 User must be able to delete their own food listing.
R.4.6 User must be able to edit their own food listing.
R.4.7 User must be able to save interested food listing.
R.4.8 User must be able to view the list of saved food listings.

#### Inventory

R.5.1 User must be able to view the list of ingredients added.
R.5.2 System should display the details of the ingredient.
R.5.3 User must be able to add new ingredient.
R.5.4 User must be able to remove ingredient.
R.5.5 User must be able to edit the details of ingredient.
R.5.6 System should notify user when ingredient is about to expire.

#### Server

R.6.1 All data must be accessed through the API defined.
R.6.2 Server must respond to API request from user.
R.6.3 Server must return data queried by the user.

### 3.1.2 Non-Functional Requirements

#### Performance

R.7.1 System should have page load time of less than 1s.
R.7.2 Server must be able to handle multiple requests per second.
R.7.3 Database should be able to handle multiple queries per second.
R.7.4 Server should have response time lower than 2s.
R.7.5 Database should scale with increased number of users.
R.7.6 Server response time should not exceed 2s with increased number of users.

#### Usability

R.8.1 User should be able to navigate to desired page in less than 5s.
R.8.2 System must have a consistent user interface.

#### Security

R.9.1 System must not collect personal information about user.
R.9.2 System must not store credentials of user in plain text.
R.9.3 System must not allow unauthorised request.
R.9.4 Server must authenticate the CSRF token before handling POST request.

**Availability**

R.10.1 Server must be available to handle requests all the time.
R.10.2 Database must be available to perform queries all the time.

## 3.2 Use Case Diagram

Figure 3.1 shows the class diagram for *FoodXcess* which aims to summarize functional requirements of the application by deriving all the use cases. The sole actor of the application is the user, which are free to access all the functionalities of the application once they have sign up for an account, as user's identity are needed for functionalities such as adding ingredients to inventory and creating food listing.

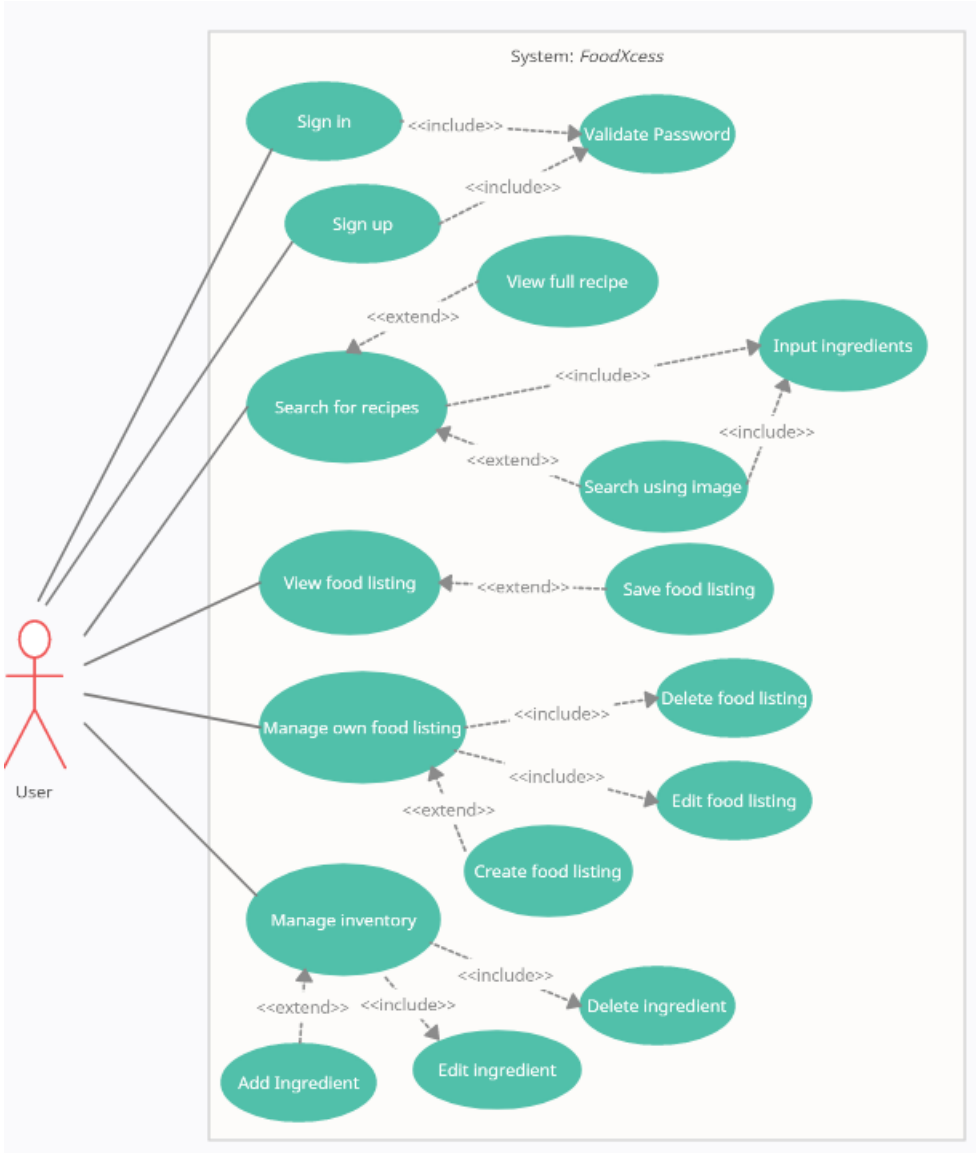



Figure 3.1: The use case diagram for the User in *FoodXcess*

## 3.3 Class Diagram

Figures 3.2 shows the class diagram of *FoodXcess* after going through the requirements and use case of each system component. It presents the static view of *FoodXcess* by clearly describing the attributes and operations for each class. Furthermore, the relationships between different classes and the constraints imposed on the system is shown.

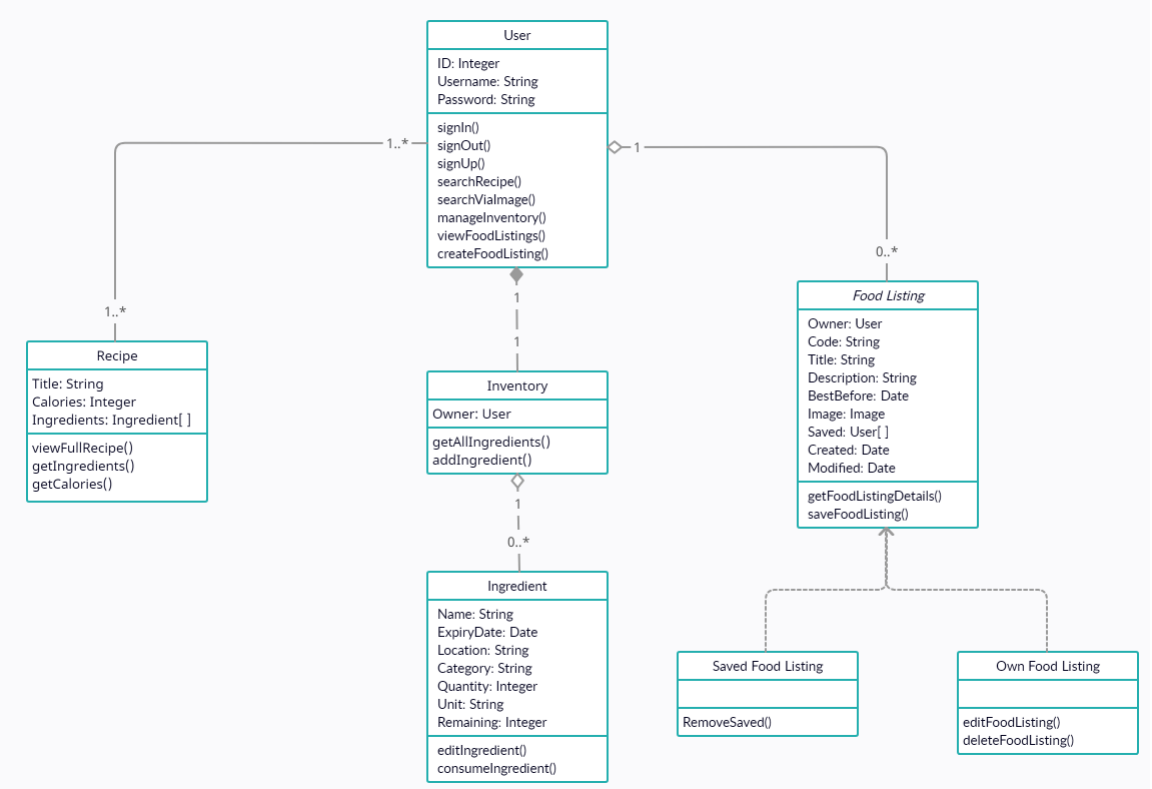



Figure 3.2: The class diagram for *FoodXcess*

## 3.4 System Design

*FoodXcess* is designed as a cross-platform application, hence it's sensible that we adopt the client-server architecture that allows linear interaction between database and different frontends. Figure 3.3 shows the diagram of the system's architecture. In our case, client is the user that access the service by sending request to server via the frontend, whereas the server listen for requests from clients and access the database to respond accordingly. By implementing REST application programming interface(API), users are able to send distinct HTTP request to access the desired resource. When server returns a response, the client application will process the data received and displays it via user interface(UI). Due to the decoupling relationship between client and server, the server is highly scalable and can increase its size dynamically. Since data is centrally managed by server, users are able to access the data from any platform, as long as valid request is sent. However, frontend must be adapted in order to communicate with different device. Therefore, we develop the frontend for desktop and mobile application individually to maximize the number of devices that can access the service.

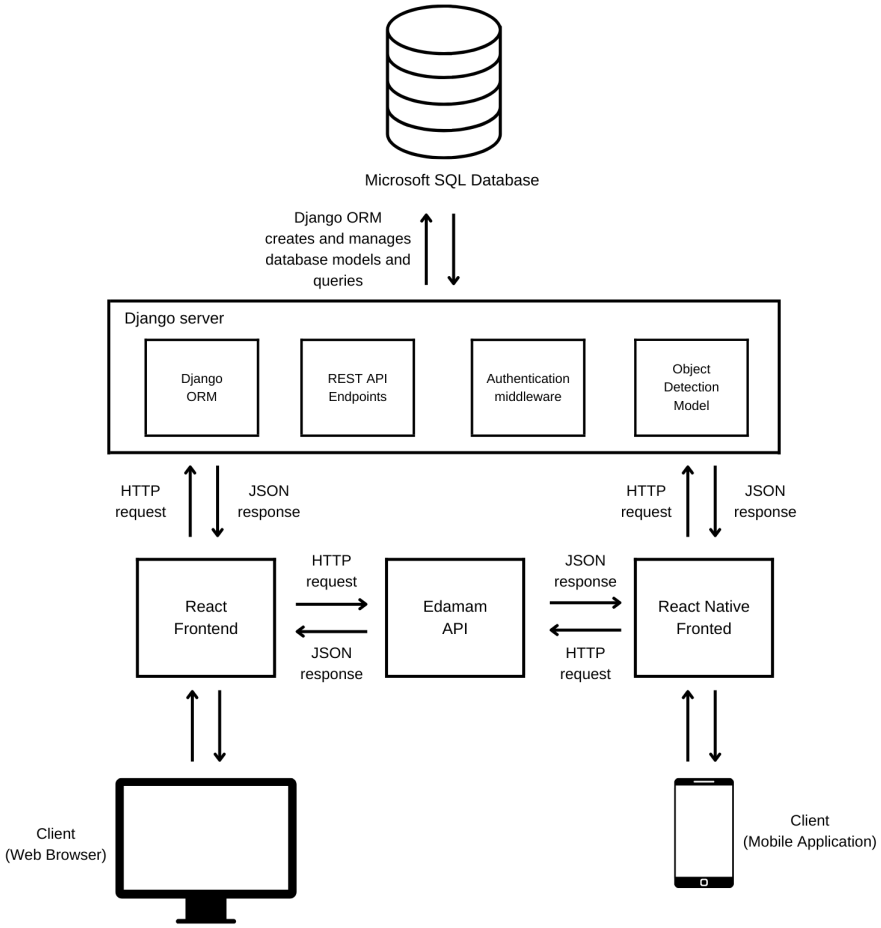



Figure 3.3: The client-server architecture adopted for *FoodXcess*

### 3.4.1 Backend

#### Microsoft SQL Database

For *FoodXcess*, we implement Microsoft SQL Server as the database, which allows the database to be hosted on the same computer or other computer across the network(including the internet). As Microsoft provides different editions of the database server, it can be easily scaled according to our usage. The database consists of 3 tables: User, FoodListing and Ingredient, which stores user's credentials, details of food listings and details of ingredients added by user respectively. The database is connected to the server via the mssql driver.

#### Django Server

The application is centrally hosted by a Django web server, which incorporates several modules that is responsible for different part of the functionality each. The Django server includes a default object-relational mapping layer (ORM), which can be used to interact with the data in database. Incoming data received from user's request are first validated to ensure that data

match the application's logic, and then serialized before ORM performs manipulation on the database. This not only strengthen the security of the application, but also ensure the data integrity. There are 3 types of database models: User, FoodListing and Ingredient, which have their own table in the database.

The Rest API Endpoints are accessed through HTTP requests from user, which performs different operations depending on the endpoint called, the request type and content of the request. Generally, GET request is used to retrieve data such as details of food listings and ingredients, while POST request is for creating or updating certain item with content embedded in the body of the request. A 400 Bad Request status will be returned if the content does not pass the validation check. The response return by server will usually contains the data queried in JSON format, along with a HTTP status code. The data returned would then be processed by the frontend to display the result to the user.

Authentication middleware is the key security feature of the application, as it adds an 'user' attribute to every incoming request. This allows the server to identify the user and restrict their access to certain functionality. For instance, user that isn't logged-in will not be able to access the inventory function, as ingredient in inventory is directly bind to the user. The middleware will save the identity in user's session when the correct credentials are submitted and clean out the session data when user signs out. Furthermore, the middleware also protects the server against cross site request forgery(CSRF) attack by examining the CSRF token that was sent along with the POST request.

The last module of the server is the object detection model that was trained to identify the ingredient that an image contains. The model is integrated into the server instead of residing in the client, so that it can act as a web-service. We are doing this as object detection is a resource-intensive task, and its performance may be affected by the hardware limitation of client, causing more latency. Whenever the user uploads an image from the frontend of the application, the image gets translated into bytes and sent along with the POST request, the server then runs the prediction function of the model on the image and sends back a JSON response with the result.

### 3.4.2 Frontend

*FoodXcess* consists of five main components: authentication, ai search, recipe search, food listing and inventory. For both the web and mobile application of *FoodXcess*, a simple user interface is implemented in order to increase the engagement of users and guide them to their desired action.

#### Web application

For this, we have decided to use the ReactJS framework developed by Meta Inc, which is an efficient and flexible open-source JavaScript library for building fast and scalable frontend. The web application is made up of multiple components, with each component having its own logic and control. These components are then rendered to fill data in the HTML DOM. It makes use of JavaScript XML(JSX) to accept HTML quoting and makes rendering components easier, instead of needing to learn a "domain-specific language". Besides that, ReactJS uses a virtual DOM that only changes individual DOM elements instead of reloading the whole DOM, hence improving performance of the web app.

The user interface for web application mainly consists of two components, which are the navigation bar and the main screen below it. The navigation bar allows the user to reach different pages easily and switch between different functionalities provided by the application. Meanwhile, the main screen will render different pages depending on the link clicked by the user. However, when the browser's width reaches the breaking point, a sidebar will replace the navigation bar as the navigation menu. Figure 3.4 shows the homepage in different window size and the sidebar component.

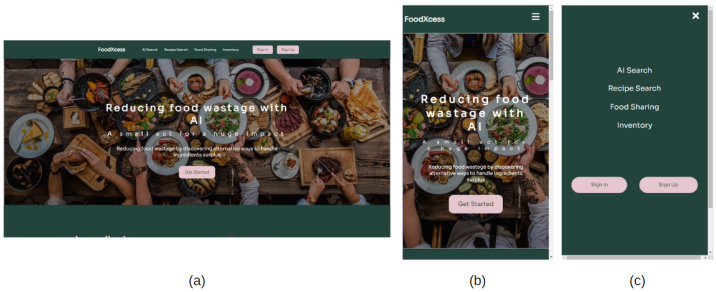



Figure 3.4: (a)The homepage of the web application (b)The homepage when the browser's window size reaches the breaking point (c)The sidebar navigation menu

By utilizing the BrowserRouter, Switch and Route components, different pages could be displayed on the main screen when user navigates. The main screen is wrapped inside the BrowserRouter, so that it only re-renders the main screen when there is an URL change. Switch and Route are used as route matcher, Switch would search through its children Route elements to find the path which matches the current URL and renders that Route. Figure 3.5 shows the different Route rendered.

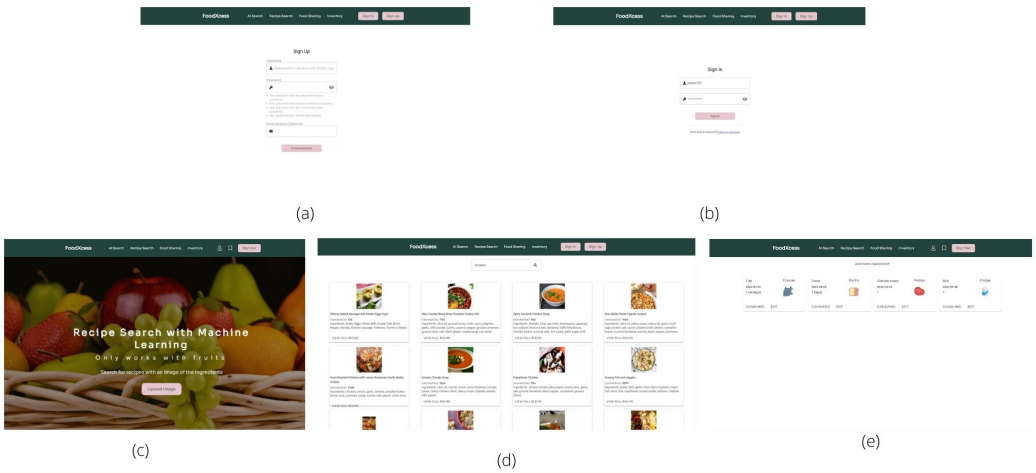
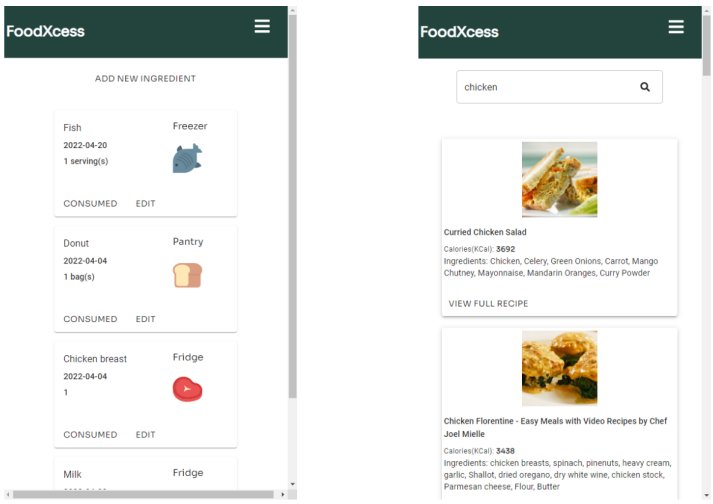



Figure 3.5: (a)Sign Up page (b)Sign In page (c)AI Search page (d)Recipe page (e)Inventory page

We have also imported some essential components from Material-UI library, as they are highly

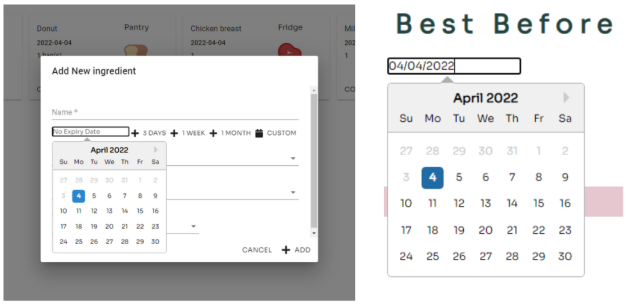
customizable with the help of some predefined props. The Grid component was widely used to help implements the responsive layout for the application, as it can be easily modified to deal with different breaking points. Moreover, figure 3.6 shows the simplistic design by using Card and Paper components to better display the recipes and ingredients. The Dialog component also allows the user to add or edit ingredients using a popup dialog box.




(a) (b)

Figure 3.6: (a)The Inventory page in small screen (b)The Recipe page in small screen

DatePicker and Momentjs is used to help user choose the expiry date for food listings and ingredients. The DatePicker component provides a small calendar that eases the date selection process, as shown in figure 3.7. The selected date is then wrapped by Momentjs to format the Date object into an acceptable date input for the backend.




(a) (b)

Figure 3.7: The DatePicker component used in (a)Selecting expiry date for ingredient (b)Selecting best before date for food listing

We have also utilized the StyledComponent library to enrich the aesthetics of the web appli-

cation. It allows CSS styling to be applied onto component, including features like pseudo-selectors, nesting, etc. With this, we have created different wrapper for layout purpose and customized components such as the navigation bar, home page and buttons.

Although the web application adopts a responsive design to provide greater accessibility for devices with smaller screen, it is much sensible to develop a mobile application for *FoodXcess* to improve the overall user experience.

### Mobile application

For the frontend of the mobile application, the React Native framework was selected because it allows the usage of React framework along with native platform capabilities and most of the code can be shared between both iOS and Android platforms. This not only reduces the development time, but also improves the user experience as React Native translate the JavaScript code written to render native UI elements of the host platform, instead of web-views like other framework does. React Native has provides many default components that are useful for developing the layout(i.e View, ScrollView and ImageBackground) and other key UI components such as TouchableOpacity, Text and TextInput.

Two main types of navigation were used throughout the design of *FoodXcess*: stacked navigation and tab navigation. Stack navigation provides a way for the app to transition into a new screen by stacking it on top of the current screen. In order to reach the final screen of the stack, user will need to go through all the screens in between. This allows the app to guide user on accessing different screen sequentially to complete certain task. In Figure 3.8, we show an example of using stack navigation to guide the process of creating a food listing.

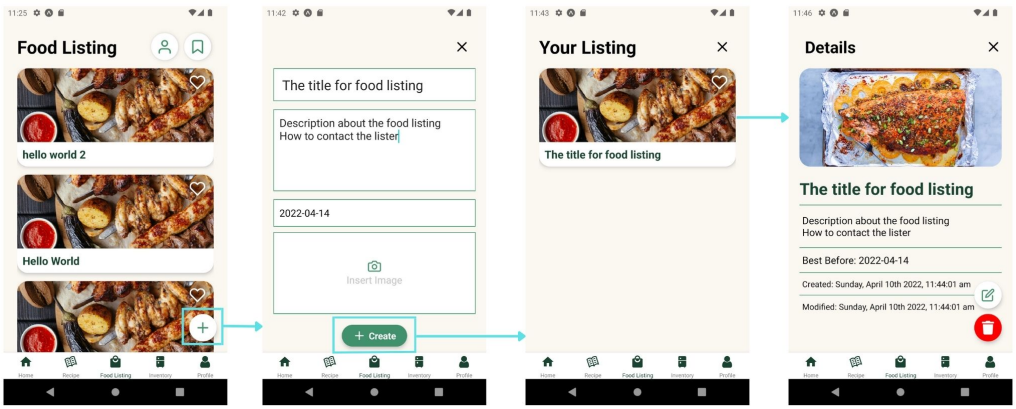
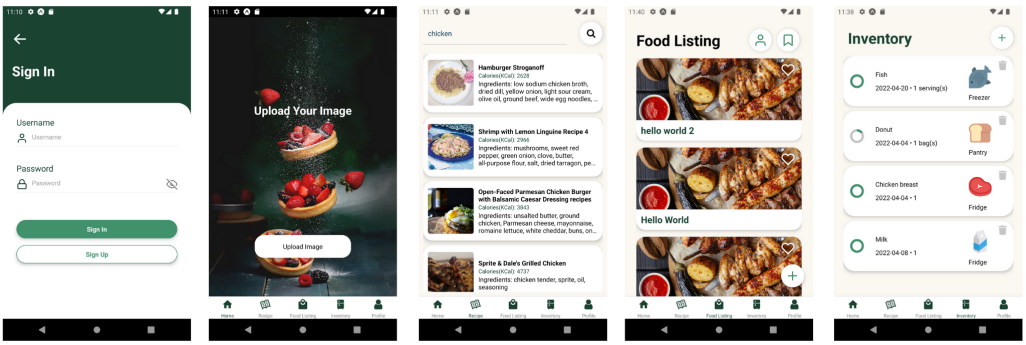



Figure 3.8: Using tab navigator to guide user on creating food listing

In this example, the user wants to create a food listing to share his excess food. Firstly, the user needs to be on the food listing main screen and clicks on '+' button. The app then navigates to a second screen which allows user to input the details of the food listing, and then a final screen which displays all the food listing the user created after clicking the 'create' button. Similar design is also applied on adding new ingredients and logging user into the application.




(a) (b) (c) (d) (e)

Figure 3.9: (a)The Sign In screen (b)AI Search screen (c)Recipe screen (d)Food Listing screen (e)Inventory screen

After user signed in, it will be guided to the main screen which consist of the tab navigator at the bottom of the screen. Tab navigation allows user to switch between all the key functionality of the app from the main screen. Figure 3.9 shows the 5 key components of the mobile application: Authentication, which deals with sign in and sign up; AISearch, which allows user to perform recipe search by uploading image of ingredients; Recipe, which enables user to search for recipes; Food Listing, which allows user to share excess food; and Inventory, for keeping track of ingredients.

View and ScrollView are the two most fundamental components for building the UI of *FoodXcess*. View is the container that supports the layout of the app. By using its flexbox and style props, nested components can be arranged according to our need. Meanwhile, ScrollView allows the display of long list of contents in the scrollable content region, underlying content would be displayed when the user scrolls it. Examples of ScrollView for *FoodXcess* can be seen on the Recipe, Food Listing and Inventory screens.

Instead of the Button component provided, we have decided to implement our own button design using the TouchableOpacity, Text and Icon components. By adding our desired control to the onPress prop of TouchableOpacity, they can act as buttons for form submission and navigation. Different icons are added to better express the intention of buttons.

# Chapter 4

# Object Detection and Model Selection

After going through most of the technological approach on addressing consumer food waste, we realised that there are not much machine-learning related attempts on this issue, hence we have decided to train an object detection model that can recognize all the ingredients from a single image. For the purpose of this project, which is only recognizing the type of ingredients, object detection might be an overkill, as the object localization done by the model is not utilized. However, the main reason for using object detection over multilabel classification is that the prediction made by the model can be easily interpreted for other purposes in the future, such as quantity estimation which could ease the process of adding ingredients into the inventory and searching recipes with exact amount of ingredients. By integrating the model to the server, users will be able to upload their image through the frontend and get the prediction result by the model. Faster RCNN and SSD are the two models that were considered, with each specializing in accuracy and speed aspect respectively.

## 4.1 Models considered

### 4.1.1 Faster RCNN

Faster RCNN (Ren et al., 2015) introduced the concept of Region Proposal Network(RPN) to generate the region proposals, this greatly improves the region proposal time from Fast RCNN that was using selective search algorithm which takes 2 seconds per image, and allows region proposal stage to share layers with the following detection stages. The input image is first being fed into the convolutional neural network(CNN) to get an output feature map, RPN will then places "anchors" boxes of different sizes and aspect ratio at possible location that contains an object. It then checks whether these anchors actually contain objects and refines them to give bounding boxes as object proposals. With these region proposals, the ROI pooling layer divides them into sub-windows from features maps and perform max pooling over them to get a fixed size output, which then pass through two fully connected layers before being fed into the classification and regression layer. The classification layer has a softmax layer to get the probability of a proposal belonging to each class(including a background class), while the regression layer improves the coordinates, width and height of the bounding box.

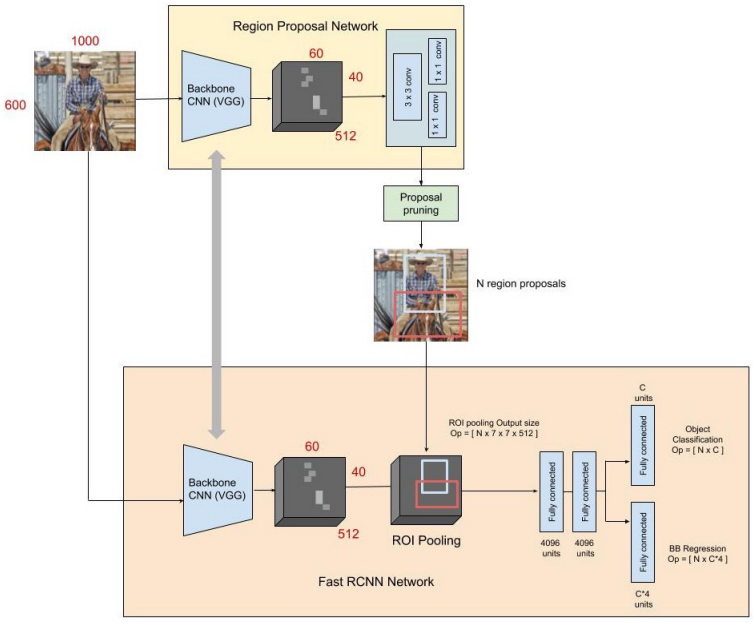



Figure 4.1: The two-stage architecture of Faster RCNN that uses VGG as its backbone network.

### 4.1.2 SSD

SSD speeds up the detection process by implementing multi-scale features and default boxes, instead of region proposal network. The input image is first passed into a CNN to extract feature maps, and applying a smaller convolutional layer on them afterward. For each location, a number of bounding boxes with predefined sizes and aspect ratios are generated(default bounding box). For each of the bounding box, prediction score for each class(including a background class) and 4 offsets relative to the default bounding box is computed. The value of default bounding box are usually chosen manually, as it differs based on the classes of object we are detecting. The same process is then repeated for different layer of the feature maps, to detect objects with different scales.

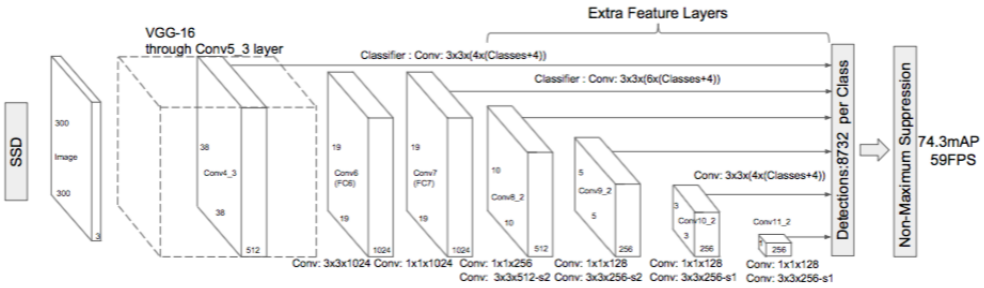



Figure 4.2: The architecture of SSD that uses VGG16 as its backbone network.

## 4.2 Training

Faster RCNN model with ResNet101(He et al., 2016) is selected as the first candidate model, as ResNet allows the training of deep CNN using the concept of "skip connection", thus increasing the representational ability and enhance the performance of the model. Secondly, SSD MobileNet(Howard et al., 2017) is also selected, with MobileNet being a small and low-latency CNN that further speeds up the inference. As the computational power required increases exponentially with the resolution of the image, the input size is unified to 640x640 for both models, so that their performance can be evaluated under a fair condition. Using the Tensorflow Object Detection API, we are able to initialize our model with weights pre-trained on MS-COCO 2017 dataset which contains 164k images and 897k annotated objects from 80 common object classes. Due to the resource constraint and the availability of food object detection dataset, we have decided to focus on detecting fruits classes only and train these models with a synthetic fruits dataset from Roboflow. The dataset contains 6000 images and 16830 annotations across 63 fruit classes. The entire training process took place on Google Colab which has GPU capabilities for faster training.


Figure 4.3: The class balance of the dataset and 4 sample images from the dataset

We separate the training process into two stages. Firstly, we train each model with 5k and 10k epochs using the default pipeline configurations, and generate the performance metric based on the evaluation dataset. Based on the evaluation, we eliminate the model that is not performing well. In the second stage, we focus on refining the configurations of the remaining model to further improve its performance. Obviously that the model eliminated during first stage might achieve better result if we get to further refine them, however the time restriction is the main reason that lead to this decision, as it is extremely time consuming to train each model, so we could only focus on refining the model that perform relatively well in the first stage, to maximize the performance of the final model on real-world images of different fruits.

For the second stage of training, we focus on finding the optimal number of epochs for Faster RCNN, fine tuning the weights of the RPN, weights of the regression layer and classification

layer. Most of the optimization are done through trial and error manually, as we assess the model by doing inference on real-life fruits images and examining the end result ourselves. Hence, there won't be much technical statistic to back the modification, instead reasoning along with examples of the inference result are provided.

## 4.3 Evaluation and Performance Analysis

For the first stage of the training, we are judging both models using their mean average precision (mAP) and average recall(AR) over a range of IoU thresholds on the evaluation dataset. Table 4.1 shows the evaluation metrics of each model after 5k and 10k epochs of training.

Table 4.1: Performance of both models after 5k and 10k epochs

| Model | Epochs | mAP@[.5,.95] | mAP@[.75] | mAP@[.5] | AR@1 | AR@10 |
|---|---|---|---|---|---|---|
| Faster RCNN with ResNet 101 | 5k | 0.608 | 0.755 | 0.816 | 0.726 | 0.766 |
| | 10k | 0.763 | 0.900 | 0.934 | 0.801 | 0.831 |
| SSD MobileNet | 5k | 0.603 | 0.747 | 0.788 | 0.716 | 0.757 |
| | 10k | 0.697 | 0.825 | 0.862 | 0.778 | 0.811 |

As we can see from above, Faster RCNN achieve a higher AR with max detection of 1 and 10 than SSD on both the 5k and 10k run. This indicates that the model is able to correctly detects most of the objects in the images, regardless that the image containing 1 or 10 objects. Furthermore, it also has a higher mAP[.5] and mAP[.75], which means that object will only be identified as positive if the IoU between prediction and ground truth is greater than 50% and 75% respectively. This shows the bounding box predicted by the model is able to locate the objects precisely., with high overlap between the prediction and ground truth. Lastly, its mAP averaged over the different IoU thresholds[0.5,0.95] is also better than what SSD achieved, further pointing out that Faster RCNN has better localization. Hence, we have decided to continue with the Faster RCNN model for second stage. As we want a model that is able to detect the ingredients from image correctly, the accuracy of the model plays a crucial role, which is also shown in the higher AR score.

For the second stage of training, we focus on finding the optimal number of epochs, fine tuning the weights of the RPN, regression layer and classification layer. The same training algorithm and dataset are used to train the model. However, we have changed the way of evaluating the model to doing inference on 50 real-world fruits images, instead of judging it by the training loss and mAP. We realised that the evaluation dataset doesn't really reflect the realistic fruits images halfway through refining the model, and it was already too late for us to annotate our own evaluation dataset. After running inference on the real-world images with the 10k Faster RCNN model, we discovered that the RPN is overfitting and the prediction layer is underfitting. These issues are better explained using the example provided in figure 4.4.

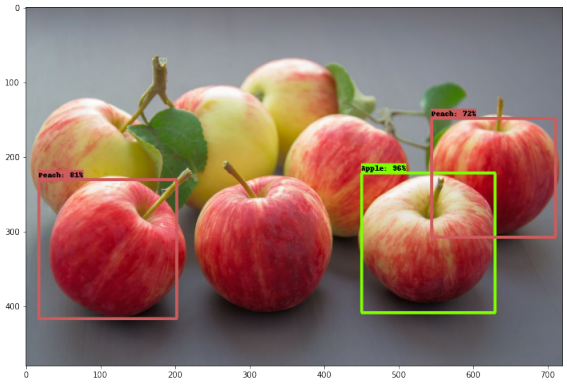



Figure 4.4: The model can't find all the apples in the images and misclassifying peach as apple

To counter this, the first solution that we thought of is reducing the number of epochs. As expected, better region proposals were generated by the RPN, but this also worsen the localization of the predicted box and classification. Therefore, we increases the weights for classification and the regression layer, so that the model can identify the fruits correctly and make precise bounding box prediction. The exact increments for these weights have been fine-tuned through each time of retraining the model and testing. Figure 4.5 shows that the performance of the final model has improved significantly after the second stage training.

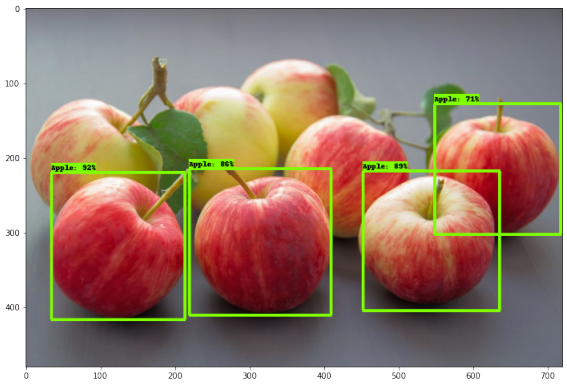



Figure 4.5: The model can correctly identify most of the apples in the image, besides those that have merged with background information

The final model is able to perform well on most of the real-world images, where fruits can be accurately found and classified. Despite that, we have also found some issue regarding the model. Firstly, the model does not perform that well when the objects are small and near to each other. Secondly, the model sometime misclassifies fruits with similar texture and colour(i.e lemon and canary melon). The discussion on these issues and the possible solution for them is in Chapter 6.

# Chapter 5

# Implementation and Testing

## 5.1 Implementation

We have decided to develop the application using the AGILE methodology, as both of our backend and frontend has been designed using a modular approach. Hence, we could easily break down the scope of work into different sprints. For each sprint, we would focus on developing a specific module of the application. With the help of Jira, a project management software, we ensure that our progress is always on track and clear about the tasks that we need to focus on. At the end of each sprint, we would run test on the specific module to ensure that it meets the requirements before integrating it.

### 5.1.1 Server

Once we have decided to develop a full stack cross-platform application, the first choice that we made was to implement it using the Django Web Framework, as it allows quick development for web application and has its own REST framework that ease the implementation of REST API. Besides that, it provides several built-in tools that help manages the database and routing. We have taken a modular approach when building the backend of *FoodXcess*, by creating three different Django modules: accounts, api and frontend. The accounts module is used to handle all the request related to user's authentication, while api module handles all functions related to the Food Listing and Inventory when their respective API endpoints are called. The frontend module is where all the JavaScript/React source codes reside. When a URL of the website is entered in the web browser, the frontend module would match the URL pattern and render the web page accordingly.

The database consists of three models: User, FoodListing and Ingredient. For the user model, we are utilizing the default user model provided by Django, which stores the username and password. Both remaining models are defined inside the "models.py" file of api module, figure 5.1 shows the Python code that define the structure of these models. With this, ORM creates the table for each model in the database. Each food listing has an associated owner, 6 letters unique code, title, description, image, best before date and the list of users that saved it. Meanwhile, each ingredient has its associated owner, name, expiry date, location where it's stored, category, quantity, unit and remaining amount. Whenever the ORM query the database, the returned object will need to be serialized to JSON format first before sending it back to user. Furthermore, we have also utilized the Microsoft SQL Server Management Studio to manage the database manually throughout the development and testing process.

```python
class FoodListing(models.Model):
    user = models.ForeignKey(User, on_delete=models.CASCADE)
    code = models.CharField(max_length=6, default=generate_unique_code, unique=True)
    title = models.CharField(max_length=50)
    description = models.TextField(max_length=250, blank=True, validators=[MaxLengthValidator(250)])
    best_before = models.DateField(blank=True, null=True)
    image = models.ImageField(upload_to='uploads/', null=True)
    saved = models.ManyToManyField(User, related_name='user', blank=True)

    created = models.DateTimeField(auto_now_add=True)
    modified = models.DateTimeField(auto_now=True)
```

(a)

```python
class Ingredient(models.Model):
    user = models.ForeignKey(User, on_delete=models.CASCADE)
    name = models.CharField(max_length=50)
    expiry_date = models.DateField(blank=True, null=True)
    location = models.CharField(max_length=30, choices=LOC_CHOICE, blank=True, null=True)
    category = models.CharField(max_length=30, choices=CAT_CHOICE, blank=True, null=True)

    quantity = models.IntegerField(default=1)
    unit = models.CharField(max_length=30, choices=UNIT_CHOICE, blank=True, null=True)
    remaining = models.IntegerField(default=100, validators=[MaxValueValidator(100), MinValueValidator(0)]
```

(b)

Figure 5.1: (a) Structure of the FoodListing model (b) Structure of the Ingredient model

With Django REST framework, we have created a list of API endpoints inside the api module as shown in figure 5.2. Whenever the server receives an API call, it will first match the list of endpoints and run the function accordingly. For instance, when GET "/api/inventory/" request is received, the server would query the database for all the ingredients added by the user, and return the serialized data in the response. For security purpose, all POST requests are required to have the CSRF token in the header.

```python
path('foodlist/', FoodListingListView.as_view()),
path('create-foodlist/', CreateFoodListingView.as_view()),
path('foodlist/<str:code>/', FoodListingView.as_view()),
path('inventory/', IngredientListView.as_view()),
path('ingredient/<str:id>/', IngredientView.as_view()),
path('add-ingredient/', AddIngredientView.as_view()),
path('saved-foodlist/', SavedFoodListingView.as_view()),
path('own-foodlist/', OwnFoodListingView.as_view()),
path('save-foodlist/<str:code>/' , SaveFoodListingView.as_view()),
path('get-token/', getCSRFTokenView.as_view()),
path('upload/', imageUploadView.as_view()),
```

Figure 5.2: The list of REST API endpoints available in the api module

For the frontend module, the server will first render a plain html file whenever the user enters the URL, which will then be "taken over" by the "app.js" file. The BroswerRouter component will then match the URL with the predefined path of its children Route component, and render the corresponding page. Since ReactJS utilized the concept of Virtual DOM to fill in the HTML, it allows fast rendering and guarantees a better user experience.

### 5.1.2 Authentication

Most of the functionalities of *FoodXcess* requires users to sign in to their account, as these functionalities are associated with the account, and will return different results based on that.

The mobile app begins with prompting user to sign in or create an account, while the website provides these buttons on the navigation bar. In order to create an account, the user will first need to be on the sign up page. User will then be prompted to enter a username and password, which needs to meet all the security requirements. If the password is not valid, the unmet conditions will be shown(figure 5.3(b)). When the server receives the request to create an account, it will store the username and the hashed password in the database. The application will then redirect the user to sign in.

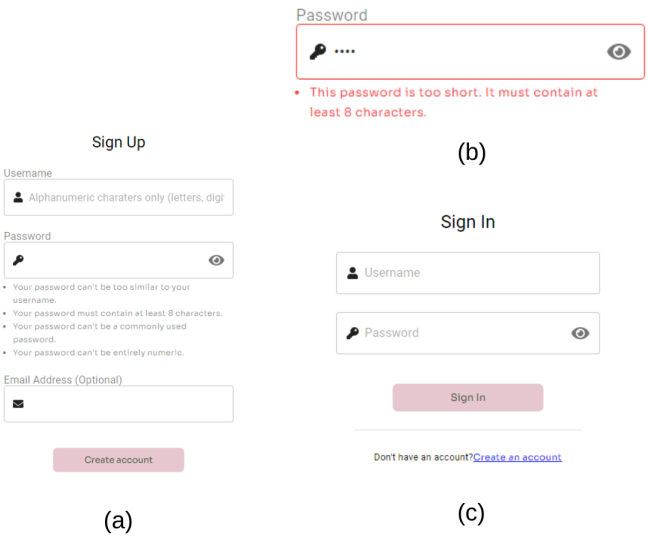



Figure 5.3: (a)The Sign Up page that prompts user to sign up with valid username and password (b)Error displayed when password is invalid (c)Redirect to Sign In page after user complete with sign up

A POST request is sent to the server when the user click the 'Sign In' button, along with the username and password entered. The server will then authenticate the input with the credentials stored in the database. With the right credentials, the authentication middleware will adds the "user" attribute to the user's session, and uses it in future request to identify the user. A user token will also be stored locally, for the frontend to render certain components differently (i.e the login required will not be rendered). When user signs out, the user token stored and the "user" attribute in session's data are cleaned.

### 5.1.3 Food Listing

The food listing was designed to be a function that enables user to share excess food. For instance, if a user has some ingredients or food that he can't finish on time, he can share it on *FoodXcess* by adding an image of the item along with a title, description and its best before date. Other user who is interested in the food listing, can find the contact details in the description and discuss on how they can hand over the item. In order to create a food listing, user will need to enter the "Create Food Listing" page by clicking on the 'Create Food Listing' or '+' button. The page has two text input field for title and description, an image upload button and date input for best before. Upon submission, the server validates the data to make sure all the required fields are filled, and a HTTP 200 or HTTP 400 response will be

returned depending on the validation. An alert with error message will be displayed if there is anything wrong with the creation. Figure 5.4 shows the different pages of the Food Listing function.

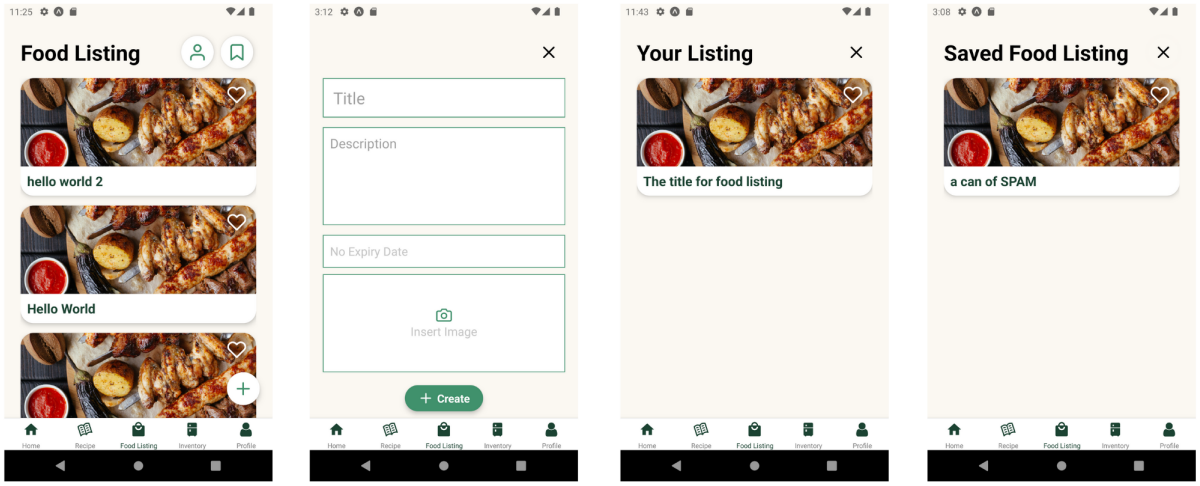
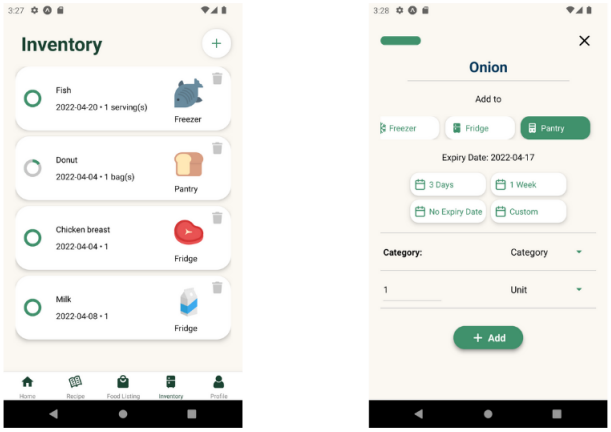



Figure 5.4: (a)Food Listing main page (b)Create Food Listing page (c)Own Food Listing page (d)Saved Food Listing page

The "Food Listing" page in the application displays all the food listings created by other users, as the GET request is called to retrieve them from the database. Besides the details of the food listing, a 'Save' button is also rendered on the frontend component. This button allows users to save their interested food listings. The 'person' and 'bookmark' icon buttons on the top right of the screen, enables users to view their own food listings and saved food listings respectively. From the "Own Food Listing" page, the user is able to edit the food listing by clicking the 'Edit' button or the pencil icon on the specific food listing.

### 5.1.4 Inventory

Once authenticated, users are granted access to the inventory screen, which helps user to manage their ingredients at home. The frontend renders the Inventory page by sending a GET request to the API endpoint, which will return all the ingredients added by the user. For each ingredient, the frontend will render a component that displays the details of the ingredient and an illustration about its category, so that they can be easily classified. Figure 5.5 shows the different pages of the Inventory function.




(a) (b)

Figure 5.5: (a)Inventory page (b)Add/Edit Ingredient page

When the "Add Ingredient" or '+' button is clicked, the user will be redirected to a form which allows user to input the details of the ingredient. Once user have finished inputting those details, they will be attached to the body of the POST request sent to the server. The server then validates the data and create the ingredient in the database. The user can edit the ingredient by clicking the 'Edit' button or the ingredient component. The same form will be displayed for user to edit the details. If the 'consume' button or the trash icon is clicked, a DELETE request is sent to the server and the ingredient will be removed. When the expiry date for an ingredient is near, the mobile application will push a notification to the user, to remind the user about it.

### 5.1.5 Recipe Search

In order perform recipe search using various ingredients input, we have decided to utilize the Edamam API, a recipe search API which contains 2.3 million recipes. A search box is provided on the "Recipe" page so that users can enter their ingredients or name of dishes. Upon submitting, the frontend sends a GET request to the URL which includes the search query. The response returned contains 20 different recipes, which frontend will render a component that display the image, title, calories and ingredients for each recipe. If user is interested in any of the recipe, clicking it redirects user to the full recipe.

Besides that, *FoodXcess* also allows user to search for recipes using images of fruits on the "AI Search" page. After selecting the desired image, a POST request with the image in bytes format will be sent to the server. The server will convert the bytes file into a numpy array, then let the model runs the inference on the image. The model will output the list of fruits that the image contains and attaches it on the response. Once the frontend receives the output, it redirects user to a confirmation page which allows the user to make corrections regarding the result. After that, a recipe search with those ingredients is performed, and the recipes returned are displayed.

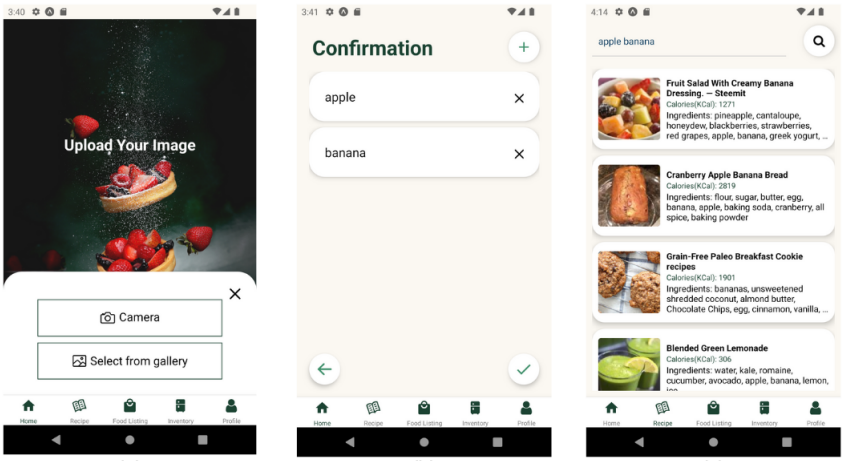



Figure 5.6: (a)User can select image from gallery or camera (b)Ingredients Confirmation page (c)The search result page

## 5.2 Testing

### Postman API

In order to ensure the REST API endpoints that we have created for the server fulfil the requirements for functionality and data connectivity, we conducted manual testing on them using Postman API at the end of the sprint. It provides templates for test suites which ease the creation process for collection of test cases for different functionality of the application. This feature also come in handy when we make some changes to some endpoints from previous sprints, as we can easily run the test again with previous test cases. The response time for the server is also tested in order to meet the performance requirements.

### Unit Testing and Integration Testing

Unit testing are conducted throughout the development phase to ensure that a specific component or section of code is performing what we expected. Using the ingredient component on the mobile application as an example, we apply custom JSX styling to its children components in order to better display the details of the ingredient, and this requires constant test and adjustment to meet the expected design. Besides that, we also test the behaviour of certain components under different conditions(i.e authenticated user and guest user). After that, we integrate the component into the main application, and run another test on the functionality to make sure that the integration does not affect the performance of the code.

### Automated testing

As more functionalities are added into the application, manual testing becomes more costly and hard, as the interactions between different components become more complex and a small change in one can impact another. Hence, we have implemented automated testing using Django's TestCase class. However, there are some functionalities such as simulating image upload from user was difficult to reproduce in the test file. Despite that, the automated test

was able to cover most of the main functionalities of the application, such as ensuring the user identity is added to the session's data by authentication middleware thru mocking user log in. Besides that, the create, read, update and delete(CRUD) functions for ingredients and food listings are also tested via mocking GET and POST request to the API endpoints.

# Chapter 6

# Discussion

## 6.1 Achievements

**Development of cross platform application**

The decision to use a server-client architecture has greatly benefits the successful implementation of the application. With this, clients are able to access the functionalities provided via the website on their browser or the mobile application on their iOS or Android smartphones. Due to the decoupling relationship between backend and frontends, we are able to develop each of them individually and establish a connection using the REST API endpoints afterward.

The combination of Django and ReactJS has simplified the development and enhance the performance of the application. Due to my prior experience with these frameworks during my internship, I only need to further explore on functions that I am not familiar with instead of learning everything from scratch, which allows me to spend more time on refining the functionalities of the application. Developing the backend server with Django was successful with the help of libraries that supports common developments need and the REST framework that made building REST API endpoints straightforward. Furthermore, the usage of React and React Native for frontend enhanced the performance of the application and facilitate the cross-platform development, with the concept of virtual DOM and rendering of native component using JavaScript code.

Besides that, the AGILE methodology applied throughout the development process also plays a huge role. During the development phase, we follow a weekly sprint schedule, which allows us to implement changes in a short amount of time. Besides that, the iteration nature of AGILE also helps with continuous improvement of the application, as we avoid mistakes from previous sprint and do better on the next one. We also use Jira to help manage the sprints, by categorizing tasks into different priorities and giving a realistic view on the project's progress all the time. As testing is integrated in project execution phase, this also increases the quality of the code and reduces the technical debt.

**Machine learning approach on tackling food waste**

We have successfully integrated the model to the server and run inference on fruits images that user uploads. Generally, the object detection is a great first approach for using object detection in the food related computer vision task, as there is relatively less involvement of machine learning in this field. This leads to an issue where we are not able to find food dataset dedicated for food object detection. Eventually, we revise our scope to fruits only

after considering factors such as hardware limitations and dataset available. The model that we trained is able to identify up to 63 types of fruits and alleviate the ingredients input process for searching recipe.

## 6.2 Deficiencies

**Missing key functions**

Even though the application has met all the functional requirements and most of the non-functional requirements, we found out that there might be some key functions that are missing from the application after reviewing the end product. Firstly, the application lacks the function of allowing users to interact with each other regarding their food listings. Leaving the contact details in description exposes user to greater risk of identity theft, stalking and harassment. Besides that, it is more sensible for us to add a location attribute to each food listing, so that user can filter out those that are unreachable. For instance, it is impossible that a user travel beyond 10km just to get a bag of carrots. Lastly, we need to encrypt the communication between the frontend and server to protect the app from cyber security attack, as requests or responses might contain sensitive data of user. An obvious example is implementing HTTPS that uses Transport Layer Security(TLS).

**Lacks of user testing**

One of the biggest mistakes made by us during the testing phase is the lack of user testing, as we can't find suitable users that are realistic end user for the application. Without the feedback from end users, we face difficulty of ensuring some of the non-functional requirements are met. Consequently, we lack the evidence to evaluate the actual impact of the application and how effective the application can help user on reducing food wastage. Furthermore, we also can't measure how well the user interface is able to facilitates the interaction between user and the application.

**Model's performance**

From the inference done by the model on real-world fruits images, we have identified a few issues with the final model trained. Firstly, it performs badly when the objects are small and near to each other. The is mainly caused by RPN not being able to generate the region proposal as the high-resolution layers are not selected, because their semantic value are low and slows down the speed significantly. One possible solution is to implement Feature Pyramid Network(FPN) for feature extraction, that generates high resolution multi-scale feature maps from semantic-rich layer by applying an additional top-down approach. With the multi-scale feature maps, RPN will be able to make more precise region proposals on small objects.

Secondly, the model sometime encounters difficulty while classifying fruits that have similar texture and colour. After running inferences on around 30 images of peach and apple, the two fruits which the model performs relatively bad on, we realised that the illumination of the image plays a huge part in the misclassification. The possible solution for this is using image augmentation to duplicate the training data by modifying the brightness of the images or turning the image to greyscale. This would help model reduces its weightage on illumination and hence focusing on other details of the fruits.

# Chapter 7

# Conclusions and Future Work

## 7.1 Conclusions

The main objective of this project is to create a cross-platform application to help reduces household food waste by providing alternative ways for user to handle excess food or ingredients. Furthermore, we also tried implementing a machine learning approach to address this issue by training our own object detection model that can identifies up to 63 types of fruits. The system produced was able to meet all the functional and most of the non-functional requirements generated from reviewing scholarly articles regarding food wastage and applications with similar aims. *FoodXcess* allows users to search for recipe using words or image input, create food listings and monitor the amount ingredients at home. The trained model was able to produce accurate prediction most of the time on clear picture. Despite the achievements, there are also various deficiencies that could have improved on. A few ideas to advance this project in the future are listed below.

## 7.2 Future work

#### Increase usage of the object detection model

The trained model is only used to ease the input for searching recipes, by returning the list of ingredients that an image contains. This is an undermining of the capabilities of the model, as it is able accurately estimate the quantity of each object by locating all the fruits from the image. This ability might be useful for facilitating user to input the ingredients into the inventory. However, this information is not fully utilized since we lack the ability to search for recipes using the exact amount of the ingredients.

#### Extending the scope of the object detection model in food field

For this project, the model is only trained to locate and identify fruits from images. Hence, going forward it would be valuable if the model is able to recognize different types of food such as meats, seafood and etc. As food itself is a relatively large and complex field, overcoming challenges such as food packaging and different parts of meat is going to require further research. Also, an interesting project going forward would be introducing food dataset for object detection, because there are not much related work in this field.

# References


Chen, C., Seff, A., Kornhauser, A. and Xiao, J. (2015), Deepdriving: Learning affordance for direct perception in autonomous driving, *in* 'Proceedings of the IEEE international conference on computer vision', pp. 2722–2730.

Evans, D. (2012), 'Binning, gifting and recovery: the conduits of disposal in household food consumption', *Environment and Planning D: Society and Space* **30**(6), 1123–1137.

Farr-Wharton, G., Foth, M. and Cho, J. (2014), 'ıdent fy ng factors that promote consumer behav ours caus ng exp red domest c food waste', *Journal of Consumer Behav our, 13 (6): 393* **402**.

Girshick, R., Donahue, J., Darrell, T. and Malik, J. (2014), Rich feature hierarchies for accurate object detection and semantic segmentation, *in* 'Proceedings of the IEEE conference on computer vision and pattern recognition', pp. 580–587.

Gustavsson, J., Cederberg, C., Sonesson, U., Van Otterdijk, R. and Meybeck, A. (2011), 'Global food losses and food waste'.

He, K., Zhang, X., Ren, S. and Sun, J. (2016), Deep residual learning for image recognition, *in* 'Proceedings of the IEEE conference on computer vision and pattern recognition', pp. 770–778.

Howard, A. G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., Andreetto, M. and Adam, H. (2017), 'Mobilenets: Efficient convolutional neural networks for mobile vision applications', *arXiv preprint arXiv:1704.04861* .

Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S., Fu, C.-Y. and Berg, A. C. (2016), Ssd: Single shot multibox detector, *in* 'European conference on computer vision', Springer, pp. 21–37.

Mourad, M. (2016), 'Recycling, recovering and preventing "food waste": Competing solutions for food systems sustainability in the united states and france', *Journal of Cleaner Production* **126**, 461–477.

Parfitt, J., Barthel, M. and Macnaughton, S. (2010), 'Food waste within food supply chains: quantification and potential for change to 2050', *Philosophical transactions of the royal society B: biological sciences* **365**(1554), 3065–3081.

Parizeau, K., von Massow, M. and Martin, R. (2015), 'Household-level dynamics of food waste production and related beliefs, attitudes, and behaviours in guelph, ontario', *Waste management* **35**, 207–217.

Redmon, J., Divvala, S., Girshick, R. and Farhadi, A. (2016), You only look once: Unified, real-time object detection, *in* 'Proceedings of the IEEE conference on computer vision and pattern recognition', pp. 779–788.

Ren, S., He, K., Girshick, R. and Sun, J. (2015), 'Faster r-cnn: Towards real-time object detection with region proposal networks', *Advances in neural information processing systems* **28**.

Watson, M. and Meah, A. (2012), 'Food, waste and safety: Negotiating conflicting social anxieties into the practices of domestic provisioning', *The Sociological Review* **60**, 102–120.

Yang, Z. and Nevatia, R. (2016), A multi-scale cascade fully convolutional network face detector, *in* '2016 23rd International Conference on Pattern Recognition (ICPR)', IEEE, pp. 633–638.

Zhao, Z.-Q., Zheng, P., Xu, S.-t. and Wu, X. (2019), 'Object detection with deep learning: A review', *IEEE transactions on neural networks and learning systems* **30**(11), 3212–3232.