Dark Stores: A sustainable approach to enhancing our way of life

Josiah Murray
19018590

UXCFXK-30-3
Digital Systems Project

UWE Bristol | University of the West of England

Abstract

A dark store application focused on improving consumer's lives in a sustainable fashion. The sustainable aspect will revolve around offering sustainable products, delivering customers groceries in both sustainable packaging and in a sustainable fashion. By developing an application in this manner, I hope to encourage more customers to both eat and live healthily.

The link to the source code on GitHub can be found in the appendix

Acknowledgements

Firstly, I would like to thank my supervisor, Haixia Liu for her consistent encouragement, support, and guidance throughout the project and my final year.

I would also like to thank my family and friends for supporting me through the many ups and downs presented by this project.

Table of Contents

Table of Figures

Table of Tables

Introduction

Grocery shopping is an integral social construct that according to Blázquez, (2021), 87% of UK consumers regularly perform. With over 30% being purchased online via online grocery platforms in 2020 (Tighe, 2021). Within this report I will discuss how dark grocery stores could influence more consumers to purchase sustainable alternatives and live more sustainably. As consuming non sustainable products produces vast amounts of emissions and non-recyclable waste. I plan to approach this problem by developing a sustainable grocery delivery application that will deliver groceries via bicycle or electric bike to reduce the estimated carbon emissions produced by standard methods of delivery.

1.1 Aims & Objectives

This project aims to provide a convenient solution to buying locally sourced & sustainable groceries through a mobile medium and have them delivered within the hour.

The project objectives are:

- Research into various topics to support my project aim.
- Compare and explore similar applications of on demand grocery services.
- Design, develop, and test an application using various technologies
- Utilise react native and Firebase to further domain knowledge.
- Design and develop a database to house the app data

1.2 Research Questions

These research questions were obtained through researching the various implementations of online grocery delivery services and how they attempt to tackle sustainability:

1. What is a dark store and how have they been implemented in the past?
2. What is the need for an alternative to standard online grocery shopping?
3. How has sustainability been tackled within the online grocery shopping sector?
4. What are the current flaws within standard online grocery shopping and how can this project solve them?
5. How can food delivery be optimised to improve delivery times?

1.3 Outline of research

After conducting my literature review, I have learnt a fair amount about what dark stores are and how they have captured a portion of the online grocery market. To then discover how important sustainability is within grocery retail and how e-grocers have implemented measures to maximise their sustainability. Next, I learnt about how online grocery shopping boomed during the pandemic and how consumer shopping behaviour can be influenced by grocery retailers, which lead to discussing how the concept of delivery slots are an inconvenience. Additionally, I learnt about how food delivery can be optimised to maximise the number of deliveries that can be made. This then led to me to researching how different

factors could affect grocery delivery. Finally, I researched the various technologies I would need to use to solve the proposed solution.

1.4 Report Outline

Chapter 2 consist of a critical review of the relevant literature related to the research questions found in the introductory chapter.

Chapter 3 provides an explanation of my chosen methodology and why other popular methodologies were not chosen.

Chapter 4 contains the requirements specification for the proposed solution.

Chapter 5 contains the project designs, which consists of use cases, user stories, an activity diagram, user interface designs and a JSON database diagram.

Chapter 6 consists of a description of the project implementation and provides detailed accounts of what happens in each component.

Chapter 7 consists of a detailed account of all the test cases and whether they passed or failed.

Chapter 8 consists of the project evaluation which provides my reflection on the project as a whole.

Chapter 9 contains the project conclusion that sums up the project as a whole.

Chapter 10 contains the project's references

1.5 Chapter Summary

This chapter gave a brief description of the proposed project along with its aims & objectives and the real world problem that it tries to tackle. Furthermore, it provided several research questions that would be used to support my research. Finally, it provided an outline of the research conducted and a report outline. Within the next chapter, I will critically discuss the relevant research for this project.

Literature Review

2.1 Chapter introduction

This chapter will discuss the current state of online grocery shopping and how this project will approach the idea of using dark stores to introduce the general populace to buying healthier and sustainably sourced produce. It will then explore how sustainability has been implemented in the grocery sector. Next, it will discuss how various food delivery apps approach the concept of delivery. Finally, it will explore the various algorithms and technologies that will be used to implement the project solution.

2.2 Dark Stores

Dark stores or online fulfilment centres (OFC) are a type of click and mortar business model that utilises fulfilment centres rather than physical supermarkets to distribute produce (Eriksson et al., 2019). Which allows them to operate in a way that benefits the customer first by providing them with convenience and a wide array of products (Reef, 2021). However, in comparison with brick and mortar stores such as Sainsburys, most "dark" delivery services offer a somewhat limited selection of products (Buchanan, 2021). Consequently, delivery services offer a wider array of locally sourced produce at a reduced price, which in 2020, gave rise to a high demand for "dark" delivery services such as Gorillas and Getir (Tugberk Ariker, 2021).

As a result of their popularity, Gorillas and Getir have strategically placed their fulfilment centres in highly populated areas to fulfil their promise of delivery within 10 - 20 minutes (Figure 1).

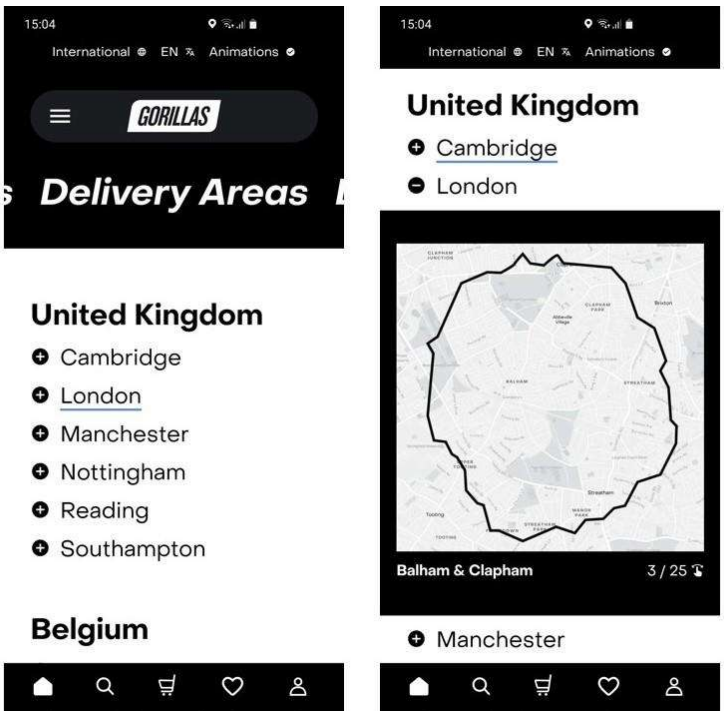


Figure 1: Gorillas Fulfilment Centre Placements (Gorillas Technologies GmbH, 2021)

However, due to the placement of their fulfilment centres, a large quantity of customers become unable to order. Therefore, the solution this project is proposing is to have a mobile delivery point, that will act as a collection hub for customers outside the boundary.

2.3 Sustainability within the grocery sector

Sustainability is the idea of conserving the environment to increase society's chances of long term survival (Cambridge Dictionary, n.d.). Therefore, within the grocery retail sector, sustainability is an integral component of their architecture, rather than a component that should be implemented later. However, this is easier said than done as in the UK, food waste is a significant issue due to the growing population and the excess produce left on shelves (Filimonau and Gherbin, 2017). This could also be due to the way grocery retailers market their deals such as "buy one, get one free", as customers may end up buying more food than necessary (Koivupuro et al, 2012). Despite facing these issues, UK grocery retailers have adopted various strategies to become more sustainable. According to Feedback (2018), these strategies include:

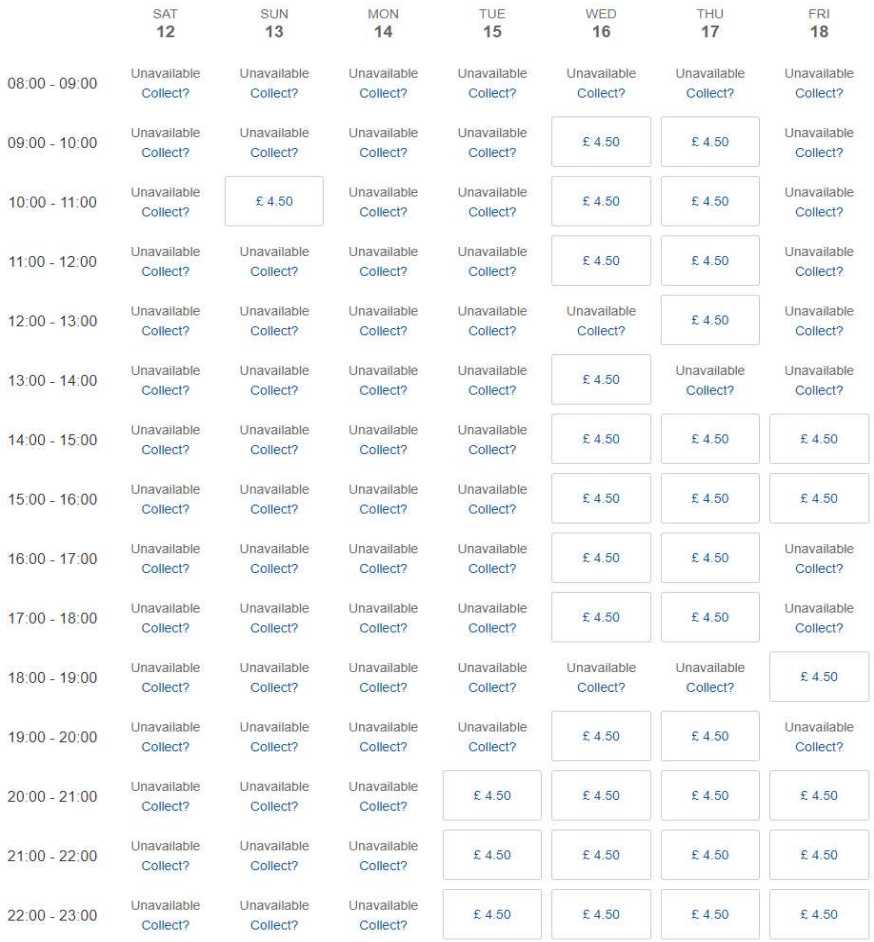
- Food donation
- Price reductions of produce towards its use by date
- Sending surplus food to animal feed
- Zero waste.

Within the e-grocery sector, sustainability is taken seriously as delivery services such as Getir and Gorrilas operate online and make an abundance of deliveries a day. Which suggests that each delivery service generates vast amounts of carbon emissions and food waste. However, according to Getir (2021), these emissions can be eliminated by delivering produce via e-bike or bicycle, instead of by car. Getir also estimates that by using the service, over 2 million miles of fossil fuel emissions can be avoided. This is a common strategy used throughout the e-grocery sector as it is an effective way of reducing emissions. Gorillas addresses food waste concerns by taking an on demand approach. Which offers customers an easy way to buy the essential produce they need, rather than buying perishable produce in bulk. Which according to Gorillas (n.d.), has been deemed as a common cause of food waste.

2.4 Online Grocery Shopping

Over the past two decades, online grocery shopping has been the only alternative customers have had to brick and mortar grocers, and during the pandemic, it saw a rise in popularity due to popular supermarkets such as Tesco and Sainsbury's being closed. In fact, according to Tighe (2021), in May of 2020 online grocery activity increased by over 30% in the UK. Furthermore, in a study performed by Tighe in 2021, it was discovered that over 42% of survey participants would continue shopping for food online post pandemic. Which suggests that the convenience and access to a large assortment of products and services encourage customers to reorder their groceries online.

However, due to consumer shopping behaviour, the most likely products to be purchased according to Brand, Schwanen and Anable (2020) are big brand products that are not necessarily sustainably produced nor the healthiest option, which suggests that online grocery shoppers are less likely to purchase the more sustainable options provided by online grocers than the products they are familiar with. Furthermore, the element of convenience is enhanced by the ability to choose when customers can have their groceries delivered through delivery slots (Figure 2).



	SAT 12	SUN 13	MON 14	TUE 15	WED 16	THU 17	FRI 18
08:00 - 09:00	Unavailable Collect?	Unavailable Collect?	Unavailable Collect?	Unavailable Collect?	Unavailable Collect?	Unavailable Collect?	Unavailable Collect?
09:00 - 10:00	Unavailable Collect?	Unavailable Collect?	Unavailable Collect?	Unavailable Collect?	£ 4.50	£ 4.50	Unavailable Collect?
10:00 - 11:00	Unavailable Collect?	£ 4.50	Unavailable Collect?	Unavailable Collect?	£ 4.50	£ 4.50	Unavailable Collect?
11:00 - 12:00	Unavailable Collect?	Unavailable Collect?	Unavailable Collect?	Unavailable Collect?	£ 4.50	£ 4.50	Unavailable Collect?
12:00 - 13:00	Unavailable Collect?	Unavailable Collect?	Unavailable Collect?	Unavailable Collect?	Unavailable Collect?	£ 4.50	Unavailable Collect?
13:00 - 14:00	Unavailable Collect?	Unavailable Collect?	Unavailable Collect?	Unavailable Collect?	£ 4.50	Unavailable Collect?	Unavailable Collect?
14:00 - 15:00	Unavailable Collect?	Unavailable Collect?	Unavailable Collect?	Unavailable Collect?	£ 4.50	£ 4.50	£ 4.50
15:00 - 16:00	Unavailable Collect?	Unavailable Collect?	Unavailable Collect?	Unavailable Collect?	£ 4.50	£ 4.50	£ 4.50
16:00 - 17:00	Unavailable Collect?	Unavailable Collect?	Unavailable Collect?	Unavailable Collect?	£ 4.50	£ 4.50	Unavailable Collect?
17:00 - 18:00	Unavailable Collect?	Unavailable Collect?	Unavailable Collect?	Unavailable Collect?	£ 4.50	£ 4.50	Unavailable Collect?
18:00 - 19:00	Unavailable Collect?	Unavailable Collect?	Unavailable Collect?	Unavailable Collect?	Unavailable Collect?	Unavailable Collect?	£ 4.50
19:00 - 20:00	Unavailable Collect?	Unavailable Collect?	Unavailable Collect?	Unavailable Collect?	£ 4.50	£ 4.50	Unavailable Collect?
20:00 - 21:00	Unavailable Collect?	Unavailable Collect?	Unavailable Collect?	£ 4.50	£ 4.50	£ 4.50	£ 4.50
21:00 - 22:00	Unavailable Collect?	Unavailable Collect?	Unavailable Collect?	£ 4.50	£ 4.50	£ 4.50	£ 4.50
22:00 - 23:00	Unavailable Collect?	Unavailable Collect?	Unavailable Collect?	£ 4.50	£ 4.50	£ 4.50	£ 4.50

Figure 2: Tesco Delivery Booking Slots (Tesco, 2022)

However, there is a fundamental flaw with this model that may discourage customers from ordering their groceries online. This issue is evident in the figure above, where delivery slots can become fully booked quickly. Which makes it inconvenient when customers cannot order their groceries whenever they want. Which became even more evident during the pandemic as popular online brands saw an influx in customers, which lead to multiple problems regarding food supply and delivery availability (Bauerová, 2021). Which according to Pantano et al (2020), lead to grocery retailers limiting the number of products each customer could buy and implementing measures to provide delivery slots to the most vulnerable before the general populace.

2.5 Delivery Optimisation

Delivery optimisation refers to the process of maximising the total amount of deliveries that can be made, whilst minimising the carbon emissions produced (Bortolini et al, 2016). Delivery optimisation techniques have been implemented in the past to solve multiple routing problems such as

- The travelling salesperson problem
    - This is an optimisation problem where a travelling sales person has to identify the shortest route between each city on a map without visiting a single city more than once (Jiang et al, 2019).
- The vehicle routing problem
    - This is an optimisation problem that consists of optimising multiple routes for a series of delivery driver fleets between the depot to a series of customers. With the goal of minimising the cost of delivery and maximising the total deliveries made (Ai and Kachitvichyanukul, 2009).

Within the on demand food delivery sector, various research has been conducted to optimise delivery routing and the various factors that affect its implementation. These factors mainly revolve around food perishability and delivery location, which according to Fikar and Braekers (2022) can also affected by the type of food being delivered, food temperature regulation and the cost of delay. As the possibility of spoiled food can lead to substantial monetary losses and unsatisfied customers. Therefore, according to Tu et al (2020), on demand food delivery services use a combination of crowd sourced delivery driver networks and genetic algorithms to optimise food delivery.

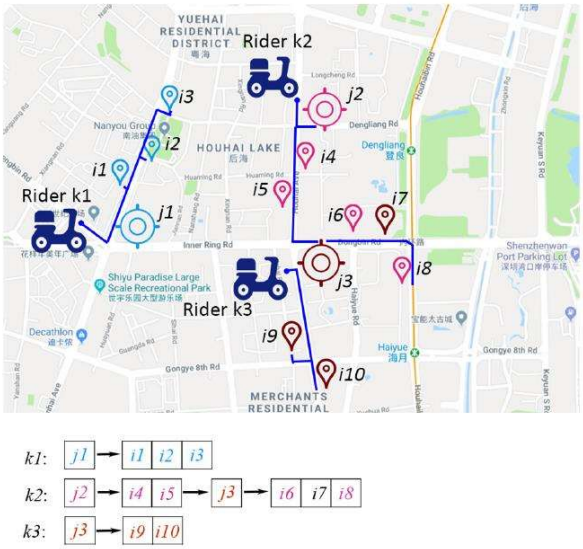


Figure 3: Crowd sourced food delivery optimisation (Tu et al, 2020)

An example implementation of this combination can be seen above and it suggests that delivery drivers can make multiple deliveries in a single trip rather than multiple deliveries over a series of short trips. Which both maximises the quality of food and allows drivers to return to each food provider at a faster rate.

2.6 Technologies I Used During Development

React Native

React Native is a JavaScript based cross platform app development framework developed by Meta, that operates by converting JavaScript code into platform specific code for either Android (Kotlin) or IOS (Swift UI)(React Native, 2022). React native offers a variety of features that make it a brilliant language for building the proposed project such as:

- A fast development time due to its innate ability to run the same codebase on both IOS and Android devices.
- Wide open source API and third party library support through node packages and JavaScript libraries.
- A detailed documentation outlining its easy to use (Ferreira, 2021).

However, due to react native not being the native language for either platform, it fails to provide platform specific features such as access to new device features and the native UI/UX experience. Despite these issues, we chose to use react native for this project due to its cross platform capabilities and the fact that we wanted to challenge ourselves by using a language we had never used before.

Firebase

Firebase is a cloud database suite developed by Google that offers various services such as a real time database, authentication, file storage and more (Firebase, 2022a). Firebase prioritises scalability and security above all as it intends to be used by developers who want a suite of services that grow with their application. These services are offered at several price points with the freemium version offering enough capabilities for smaller apps (Firebase, 2022b).

Firebase was chosen for this project due to its capability to store grocery and user data, as well as its capability to authenticate users either through email/phone numbers or their google accounts. Firebase was chosen over other cloud database providers such as MongoDB and AWS due to it providing better react native integration and the fact that it was specifically designed for app development purposes with the ability to work cross platform (MongoDB, 2022).

Google Maps API

The Google Maps API is a cloud based location suite that provides developers access to Google's maps services such as its Maps API, Geocoding API, Directions API and its Distance matrix API (Google Maps Platform, 2022). Google offers these services for a premium depending on how many requests the API receives and offers integration for various platform development such as web, Android and IOS development (Google Developers, 2022). Which are some of the reasons why the Google Maps API was chosen over other maps service providers such as the TomTom API and the Bing Maps API for the proposed project. Other reasons include its detailed and easy to understand documentation and the fact that the proposed project's target demographic is familiar with Google's services.

2.7 Chapter Summary

In summary, this chapter explored what dark stores are and why they are a suitable alternative to online grocery shopping. Then I discussed how the concept of sustainability is a core component of modern grocery shopping and how sustainability must be an integral component of the proposed project rather than a feature that should be part of the application. Next, I discussed the affects of consumer shopping behaviour on the products that are purchased by consumers. Furthermore, we discussed how delivery can be optimised to improve customer satisfaction and reduce the amount of unnecessary travel undertaken. Finally, we explored the various technologies that will be used to develop the proposed project. Within the next chapter, we will discuss the project requirements.

Requirements

3.1 Chapter Introduction

This chapter will discuss the various functional and non-functional requirements for the proposed project. It will explore the various degrees of priority under the MoSCoW method and assign each requirement a priority according to how integral it is to the completion of the project.

3.2 Functional Requirements

ID	Summary	Rationale	Priority	Success Criteria
FR-1	The app must allow the user to enter their postcode	To check if the user is within the delivery zone	Must	Upon entering a postcode, the user should be able to know if the service delivers to their address
FR-2	The app must allow the user to sign up for an account	To allow a user to access the application	Must	Upon entering their name, email, password and a phone number, user will be sent to the address entry screen
FR-3	The app should allow the user to sign back into the application	To allow a user to sign back into the application	Must	Upon entering their login details, user should be sent to the home screen.
FR-4	The app must auto sign in to the app if user has previously signed in and closed the app	To allow a user to open the app and access the home screen straight away	Could	Upon opening the app, user should be sent to the home screen
FR-5	The app should allow the user to endlessly scroll through the products and categories on the home page	To allow a user to view all the products the app has to offer	Won't have this time	User should be able to vertically scroll down the home screen and see all the products on offer.
FR-6	The app should allow the user to scroll through the various categories that are on offer.	To allow a user to view a select few products from each category	Must	User should be able to scroll down the home screen and see all the categories.
FR-7	The app should allow the user to view a specific category	To allow a user to view the items under a specific category	Should	User should be able to tap on any category in the list and be directed to that category's catalogue
FR-8	The app should allow users to navigate between the home and basket screens	To allow the user to navigate through the app quickly.	Should	User should be able to switch between the current page they are on to either the home or basket screen.

FR-9	The app should allow users to search for specific products	To allow the user to find the specific item they are looking for	Could	User should be able to enter the name of a product and the app should return suggestions
FR-10	The app should allow users to add items to their basket	To allow the user to add items they would like to purchase into their basket	Must	User should be able to select any item then select the amount they desire and add it to the basket.
FR-11	The app should allow users to view the items in their basket	To allow the user to see what exactly is in their basket	Must	User should be able to open the basket and view the items within it.
FR-12	The app should inform the user that their order has been accepted	To provide the user with reassurance that their order has been processed	Should	User should be able to see their order progressing through the delivery phases.

Table 1:Functional Requirements

3.3 Non-Functional Requirements

ID	Summary	Rationale	Priority	Success Criteria
NFR-1	The app must have an easy to navigate user interface	To allow the user to navigate through the app with ease	Must	User should be able to navigate from the home page to a specific product and back again.
NFR-2	The address details on the address entry screen should load within 5 seconds of transition	To allow the user to enter their address without having to fill the whole form.	Should	User should be able to see their address details except their house or flat number, within the text input fields.
NFR-3	The app must be useable on both IOS and Android	To make sure that any customer can use the app	Should	User should be able to access the app on either an IOS device or an Android device
NFR-4	The app should only support one customer at a single time.	To allow a single user to stay in a session for however long they desire without having to sign out	Must	User should only have to sign in once and access the app whenever they want to. The user should also be able to sign out if they want to.
NFR-5	The app database should be scalable.	To allow for multiple users and new products.	Could	In the event of an influx of users the database should be able to accept them. The database should be able to accept new products.
NFR-6	The app should be responsive on a varied number of devices	To allow for any device to use the application no matter what its screen size is.	Must	User should be able to use the application on different sized phones.

NFR-7	The app should be able to load products within 5 seconds	To enable users to view items without any issues	Should	User should be able to open the app then be sent to the home page and all of the visible items should have loaded.
NFR-8	The app should be reliable	To allow users to tap on a specific item and view the item's details. Instead of have another item' details being displayed.	Should	User taps on an item and the details for that specific item are displayed.
NFR-9	The application should be accessible for users with colour blindness	To enable users with colour blindness to utilise the application with ease	Won't have this time	App should provide an overlay that covers the screen.
NFR-10	The app must have a clear flow that is easy to understand	To enable users to go from viewing an item to adding it to the basket and finally purchasing the product.	Must	User selects an item and adds it to their basket and then checks out.
NFR-11	The app should not become non-responsive due to too many background processes	To enable users to have a consistent ordering experience.	Must	User is able to have multiple items in their basket at once without any unresponsiveness.
NFR-12	The app should confirm an order has been processed within 5 seconds of order completion	To ensure that the user knows that their order has been processed.	Should	User should be sent to an order confirmation screen alerting them, that their order has been accepted.
NFR-13	The app should be able to load images within 5 seconds	To ensure that users can see what items they are about to view	Should	User should not only be able to see the product name but the image associated to the product.

Table 2: Non-Functional Requirements

Methodology

This chapter will discuss the chosen methodology for the proposed project and will elaborate on why other software development methodologies were not chosen.

Within software development, projects are mainly managed using one of two methodologies, waterfall, and agile scrum. The waterfall methodology is a sequential set of phases starting from the initial planning phase to the testing phase, and according to Despa (2014) is suitable for small scale projects. Then the agile scrum methodology is a team-based framework that builds upon the waterfall model's principles and uses them to develop tasks through an iterative approach within short sprints rather than sequentially over a specified timescale (Despa, 2014).

On account of this project being planned, designed, implemented and tested by a single person, the agile scrum methodology was not chosen. Instead, this project will take a V-model approach as the project's requirements have been clearly defined and the approach allows for continuous testing (Kumar and Rashid, 2018). Which is arguably vital when developing mobile applications as missing a single bug or a series of bugs can ruin a user's experience.

According to Durmuş et al (2018), the V-model methodology is a guideline for software development that focuses on improving the efficiency of the software development process and improving the reliability of the software produced (see figure 4). Furthermore, it was inspired by the waterfall methodology, which suggests that the V-Model could be considered for small to medium sized projects such as the project proposed.

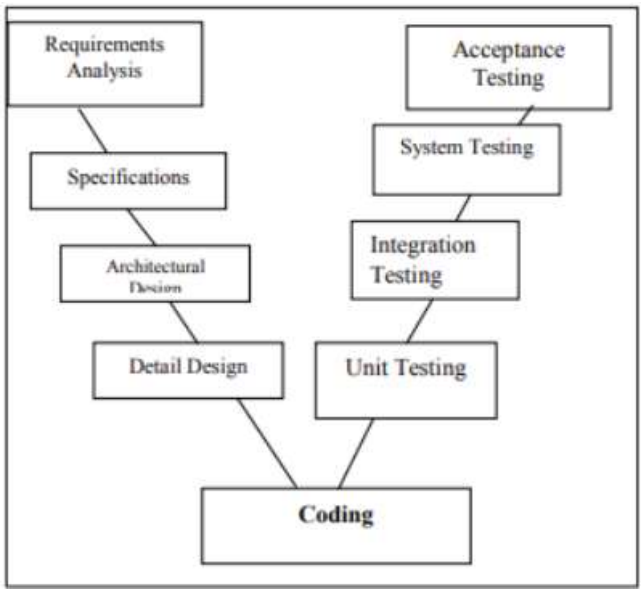


Figure 4: V Model Diagram (Mathur and Malik, 2010)

Additionally, the agile scrum methodology was not chosen due to it requiring continuous working prototypes after every sprint and required regular user feedback. Which would become challenging in such a short timescale.

Design

5.1 Introduction

This chapter will dive into the various high level and low level designs of the dark grocery delivery application. Within this chapter, I will discuss the applications' user stories, use cases, the activity diagram, then the applications' wireframe and composites and finally the document database.

5.2 User Stories

User stories are a collection of user specific tasks that consist of features that a certain type of user would want and the reasoning behind it (Rehkopf, n.a). The following table expresses specific tasks that I think app users would want.

Story Number	Story	I want/need to perform a task	So that I can achieve a goal	Acceptance Criteria
1	As a new customer	I want to check if the delivery service delivers to my postcode	so that I can have my groceries delivered to my address.	Ensure that the user can: - Enter their postcode - Click continue button
2	As a new customer	I want to create a new account by signing up with my email	so that I can use the features of the dark store app.	Ensure that the user can: - Input their personal details. - Click the sign up button. - Access the complete application after sign up button is pressed.
3	As a returning customer	I want to log back into the dark store app using my email and password	so that I can access the app and order groceries.	Ensure that the user can: - Input their email and password. - Click the log in button. - Reset their password if they click the forget password button. - Access the complete application after login button is pressed.
4	As a customer	I want to reset my password because I forgot my previous one	so that I can log back into the application to buy groceries.	Ensure that the user can: - Input their email address - Click on the reset password button

5	As a returning customer	I want to access the home page of the app on start up	so that I can order groceries without having to log in every time.	Ensure that the user can: - Open the app and instantly see the home page.
6	As a new customer	I want to enter my address details	so that I can have my groceries delivered to my address.	Ensure that the user can: - Input their building or flat number - Input their address line one - Input their city or town. - Input their postcode - Click on the continue button
7	As a customer	I want to view the products under a specific category	so that I can see what items are available.	Ensure that the user can: - Scroll sideways though the various categories - Click on a specific category - View the items under the category banner
8	As a customer	I want to scroll through the app home page	so that I can see what each category has to offer.	Ensure that the user can: - Scroll vertically through the various sub sections on the home page - Click on anything on the home page
9	As a customer	I want to search for specific products	so that I can find the exact item I want to purchase.	Ensure that the user can: - Click on the search bar - Enter an item name or category name - Click on the product and view it

10	As a customer	I want to view a specific product	so that I can add it to my basket.	Ensure that the user can: - Click on the product - Click on the plus/minus to select the product quantity - Click the add to basket button
11	As a customer	I want to view the items in my basket	so that I can edit my basket.	Ensure that the user can: - Change quantities on specific items - Remove items from basket - Clear the whole basket - Click checkout button
12	As a customer	I want to check on my order delivery	so that I know when my groceries will be delivered.	Ensure that the user can: - View the various stages of the delivery process - View the estimated delivery time - Click on the cancel order button
13	As a customer	I want to alter my details	so that I can update my existing information.	Ensure that the user can: - View their current name, phone number and email address - Change their name and phone number

Figure 5: User Stories

5.3 Use case diagram

A use case diagram is a visual representation of how the applications' users interact with different parts of the system (Lucidchart, n.d.). The figure below exhibits the use case diagram for the proposed dark grocery delivery application.

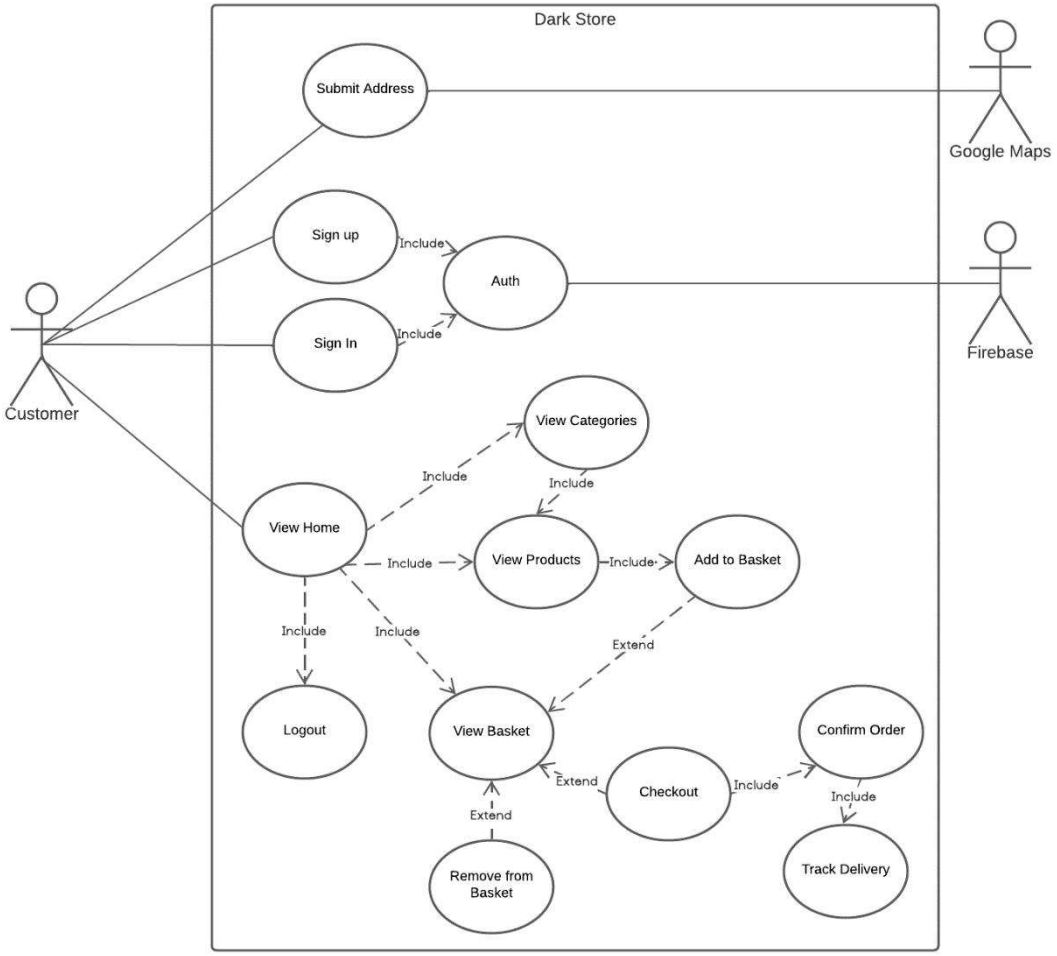


Figure 6: Use case diagram

5.4 Activity Diagram

An activity diagram is a type of UML diagram used for modelling how the proposed system will behave (Lucidchart, n.d.). It contains all the possible states the app could be in and the possible routes that could manifest if a user chooses a specific path.

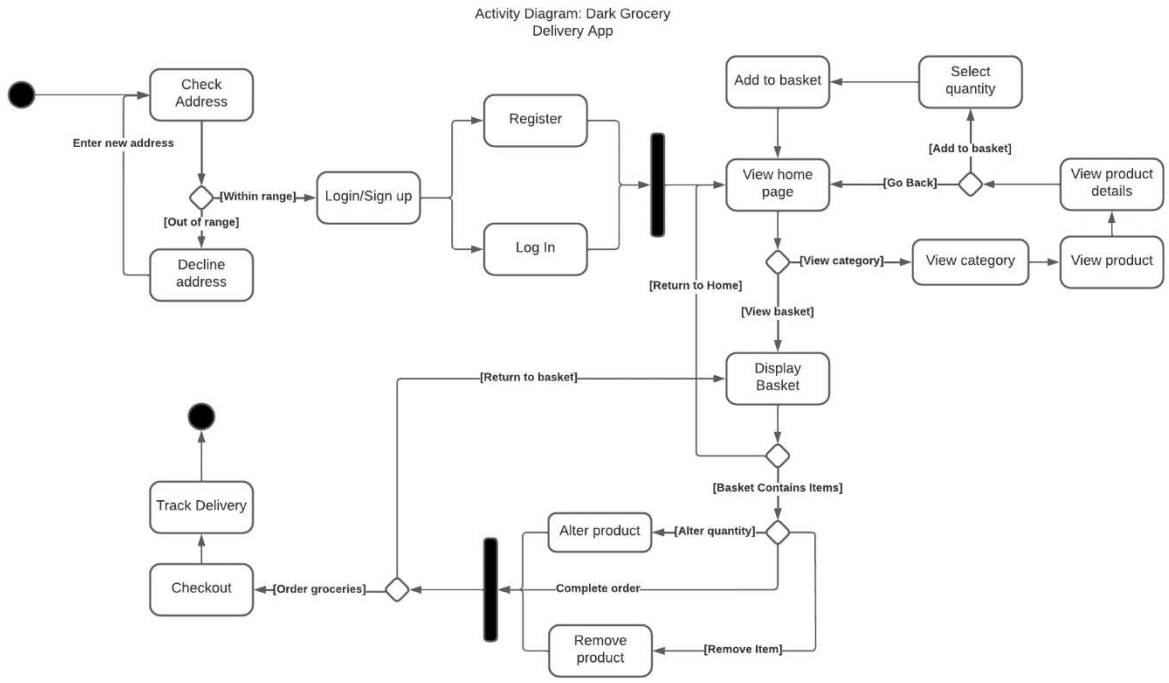


Figure 7: Dark Grocery delivery app activity diagram

The figure above can be segmented into 4 sections:

1. Address verification & Authentication
2. Product Viewing & Selection
3. Basket
4. Ordering

Address verification & Authentication

Within this section, the user would enter their address and the system would either inform them that they are either within range or outside the range of the fulfilment centre. In the case of the user being out of range, the user would be asked to enter another address. Once the user's address has been confirmed the system will request the user to either sign up or sign in. Once the user has authenticated themselves, the system will lead the user to the homepage.

Product Viewing & Selection

Within this section, the system displays the homepage and provide the user the option to either view categories or view the basket. If the user were to view the various categories, the

system would display an array of categories and a small selection of products from each category. At this point the user can select a product and view its details. The system will then ask the user if they want to either add the product to the basket or return to the homepage to view categories. If the user chooses to add a product to their basket, the system would ask the user to select a quantity. Then finally, the system would append the product and its quantity to a locally stored array. Once the basket has been updated the system then takes the user back to the homepage.

Basket

Within this section, the system displays the basket and depending on if the user selected view basket instead of view categories, the system would display all the products the user has added to their basket. However, if the basket is empty the system offers the user the option to return to the homepage. If he basket contains items, then the system offers the user the option to either alter, remove or order items. Once the user has completed their order, the system will display tracking information.

5.5 User interface designs

Throughout the design process, the apps' user interface design went through multiple re-designs due to inconsistencies in both the navigational flow and the overall ease of use. I started off designing how the basic grocery app would look like through roughly sketched wireframes, which can be seen in the figures below. As you can see, I originally designed the app in multiple different design styles, but I inevitably settled with a minimalistic design style because I did not want to distract the user from the core project concept with unnecessary assets and features. Furthermore, after re-designing the app several times, I realised that the core functionality could be streamlined and integrated into existing screens rather than having them on separate screens. For instance, the category list was originally designed to be displayed on both the home screen and on a separate category screen. Which, through testing, I found to be both a waste of resources and an unnecessary extra tap that an app user would have to make. By making this change, I was able to improve the overall ease of finding a product and adding it to the basket.

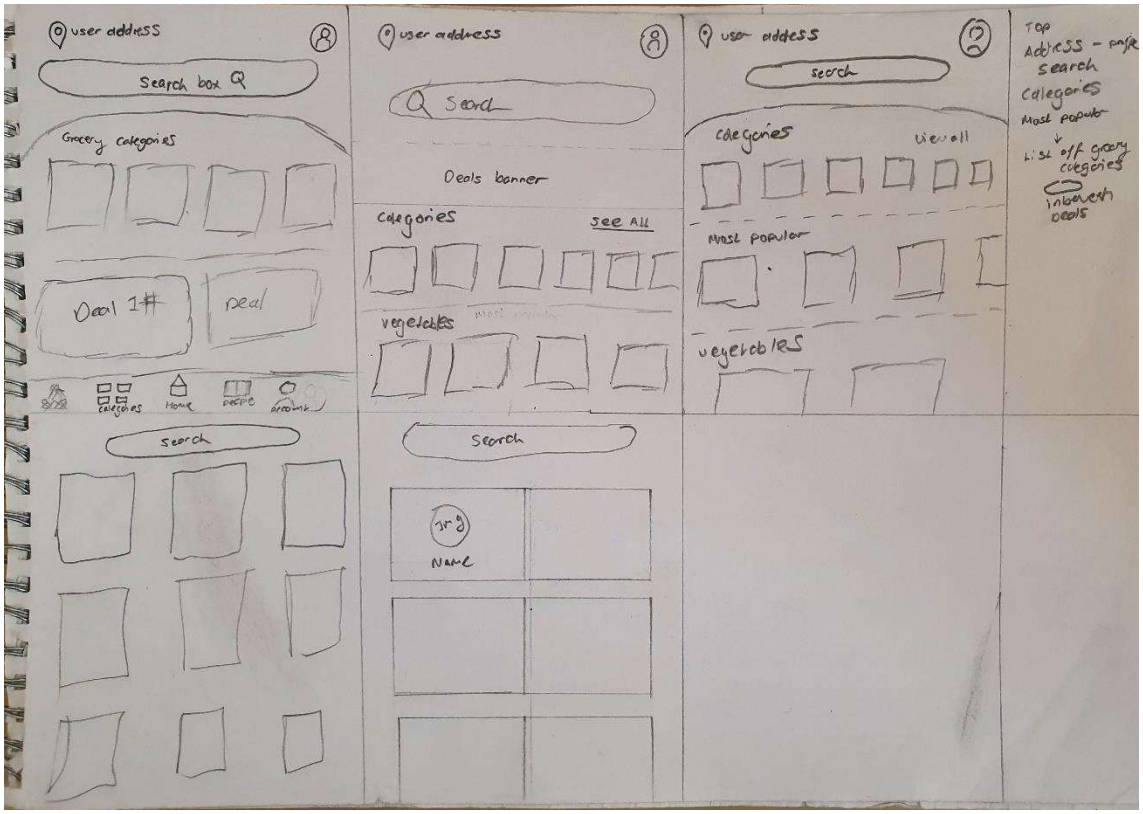


Figure 8: Address & Category Screen Wireframes

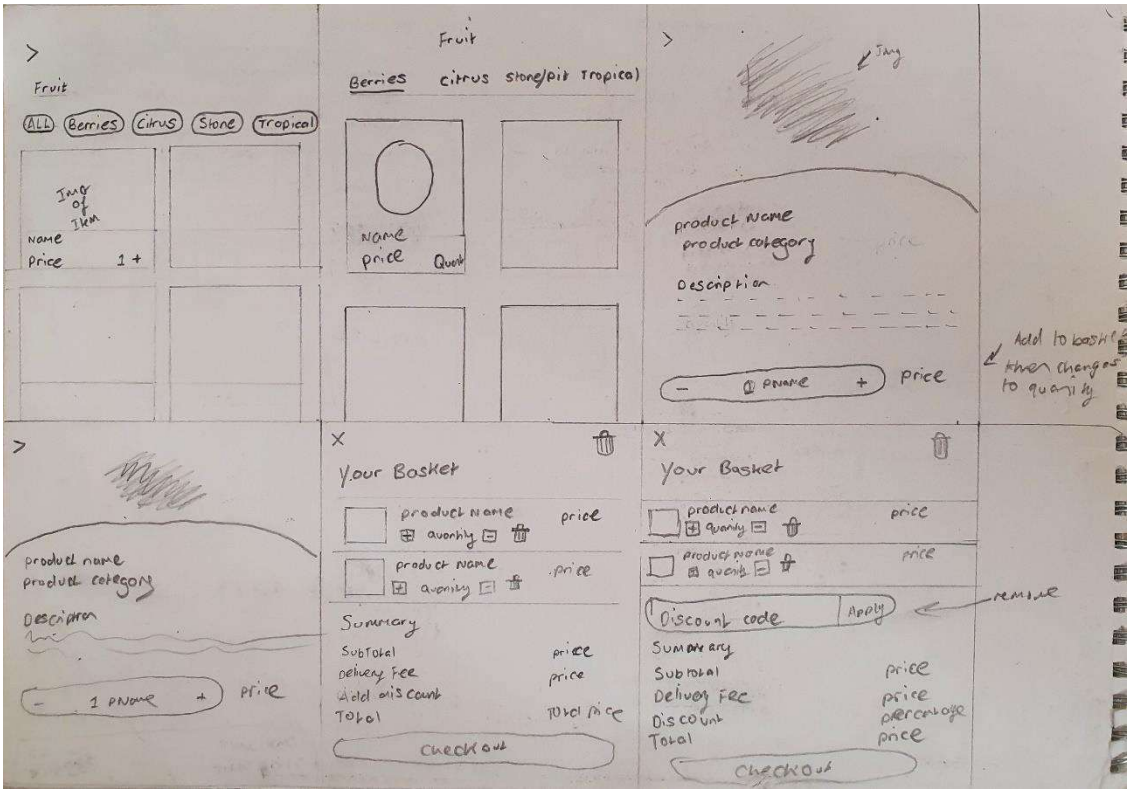


Figure 9: Wireframes for product category, product details, and basket screens

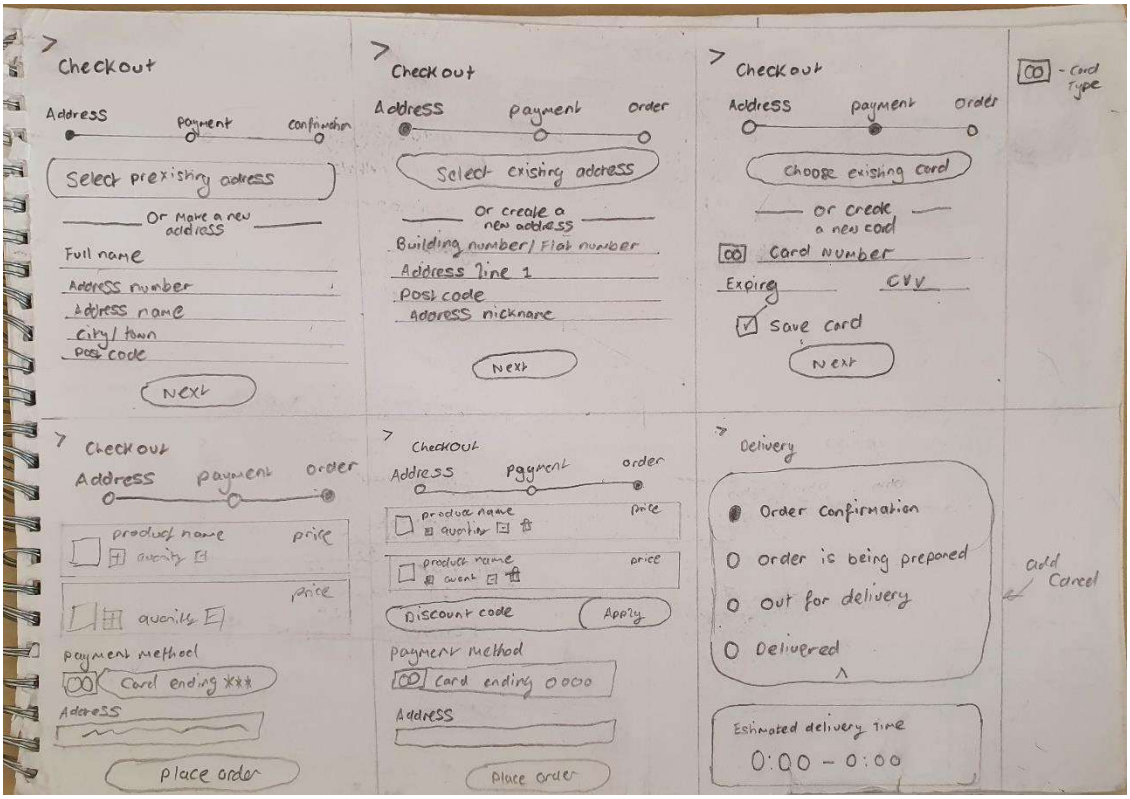


Figure 10: Wireframes for address selection, payment, order placement and delivery confirmation screens

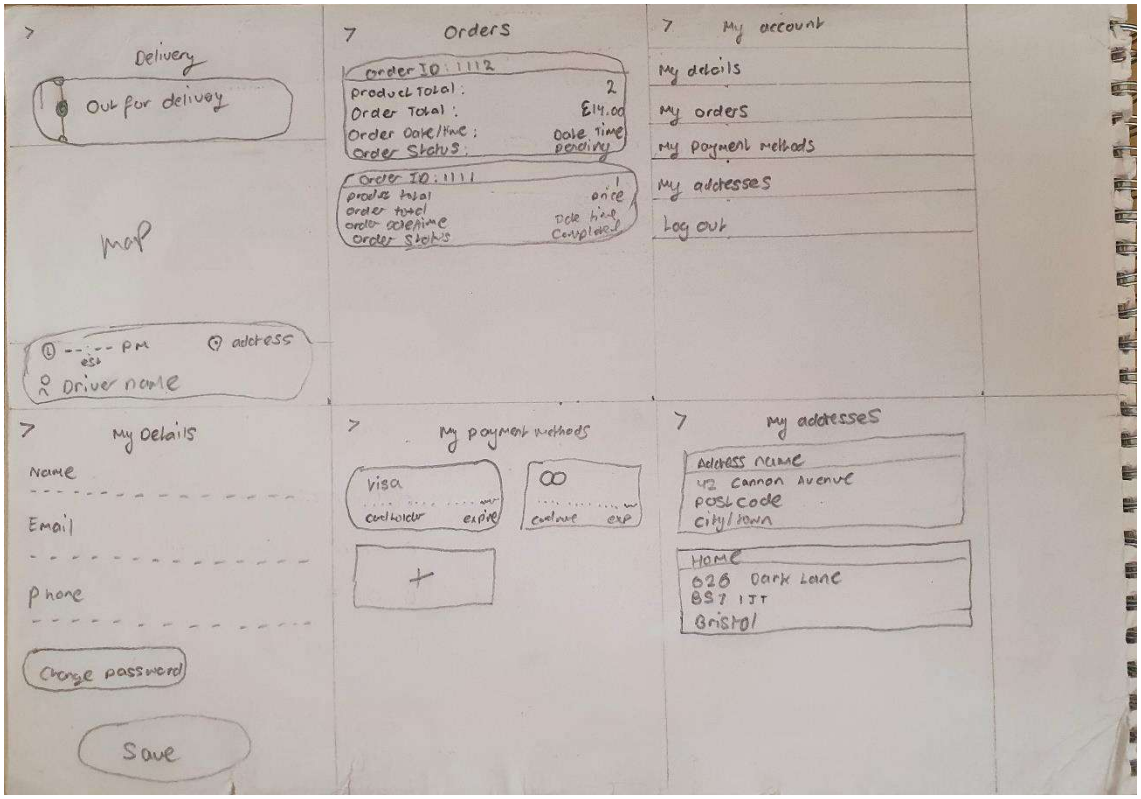


Figure 11: Wireframes for delivery tracking, order history, accounts, account details, payment details and address screens

Despite, these early designs having a strong influence on the final apps' design, I ended up taking a different design direction due to feeling that the app was still too cluttered and could still be streamlined. Therefore, I decided to return to researching app design and discovered that apps should be engaging from the start and users should be able to get what they want from the app in as few clicks as possible. Furthermore, according to Grove (2016), at least 25% of users install apps once and never use them again. Which for a food delivery service would not be ideal and suggests that I should remove certain screens and features to improve the apps' ease of use. Therefore, the following wireframes reflect these changes and due to the re-design, I was able to theoretically reduce the number of clicks to order a single item to less than 6 clicks. Additionally, I decided to make the app's colour scheme green to fit the sustainable theme.

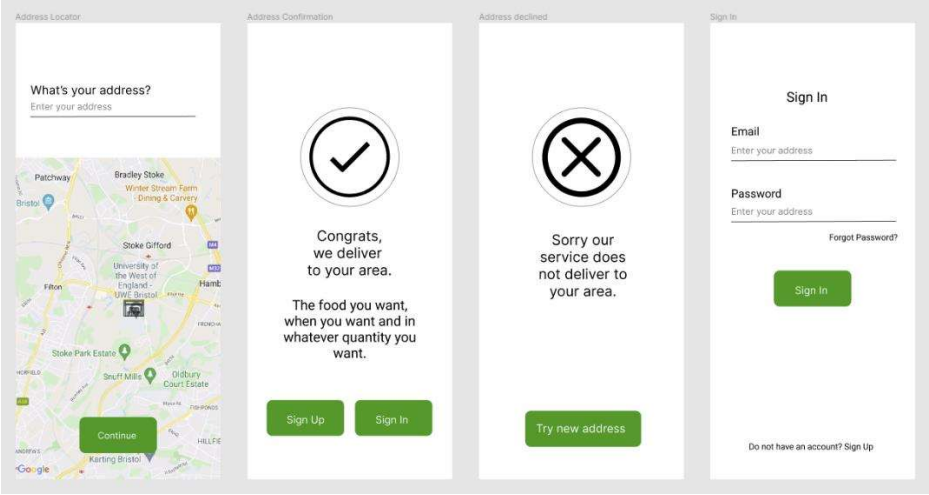


Figure 12: Wireframes for address locator, address confirmation, address declined and sign in screens

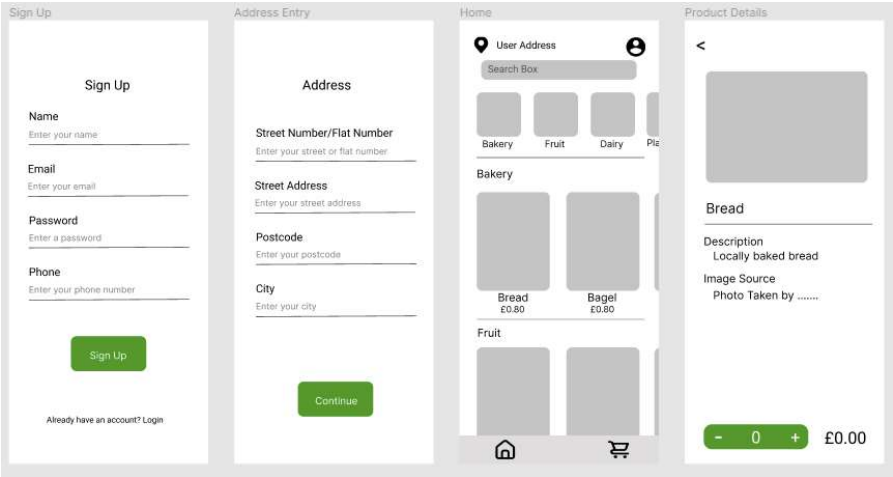


Figure 13: Wireframes for sign up, address entry, home, and product details

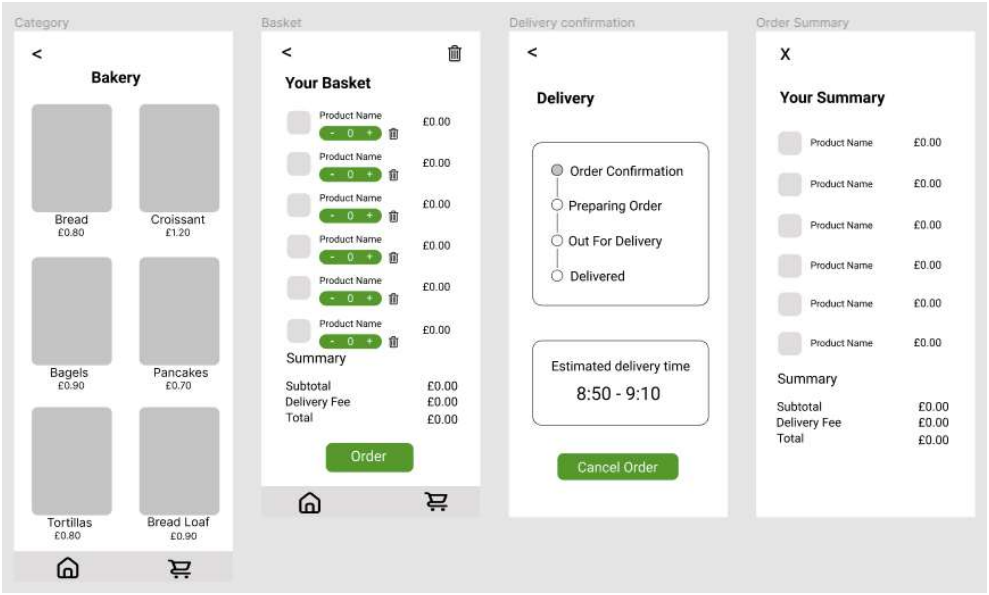


Figure 14: Wireframes for category, basket, delivery confirmation and order summary

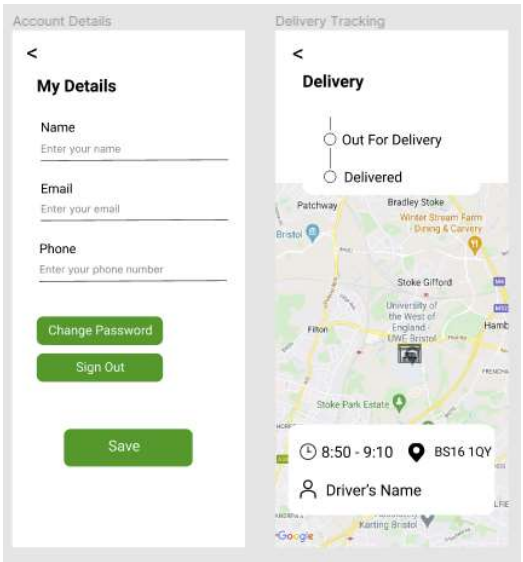


Figure 15: Wireframes for account details and delivery tracking

5.6 Document database model

As stated above, I decided to use firebase as my cloud based NoSQL document database and the figure below models the collections stored in the database.

A document database model is a diagrammatical model of a JSON document database. A document database according to Suma and Alqurashi (2019), is a NoSQL collection based storage method that uses key value pairs to store and reference data. The figure below showcases the proposed NoSQL database.

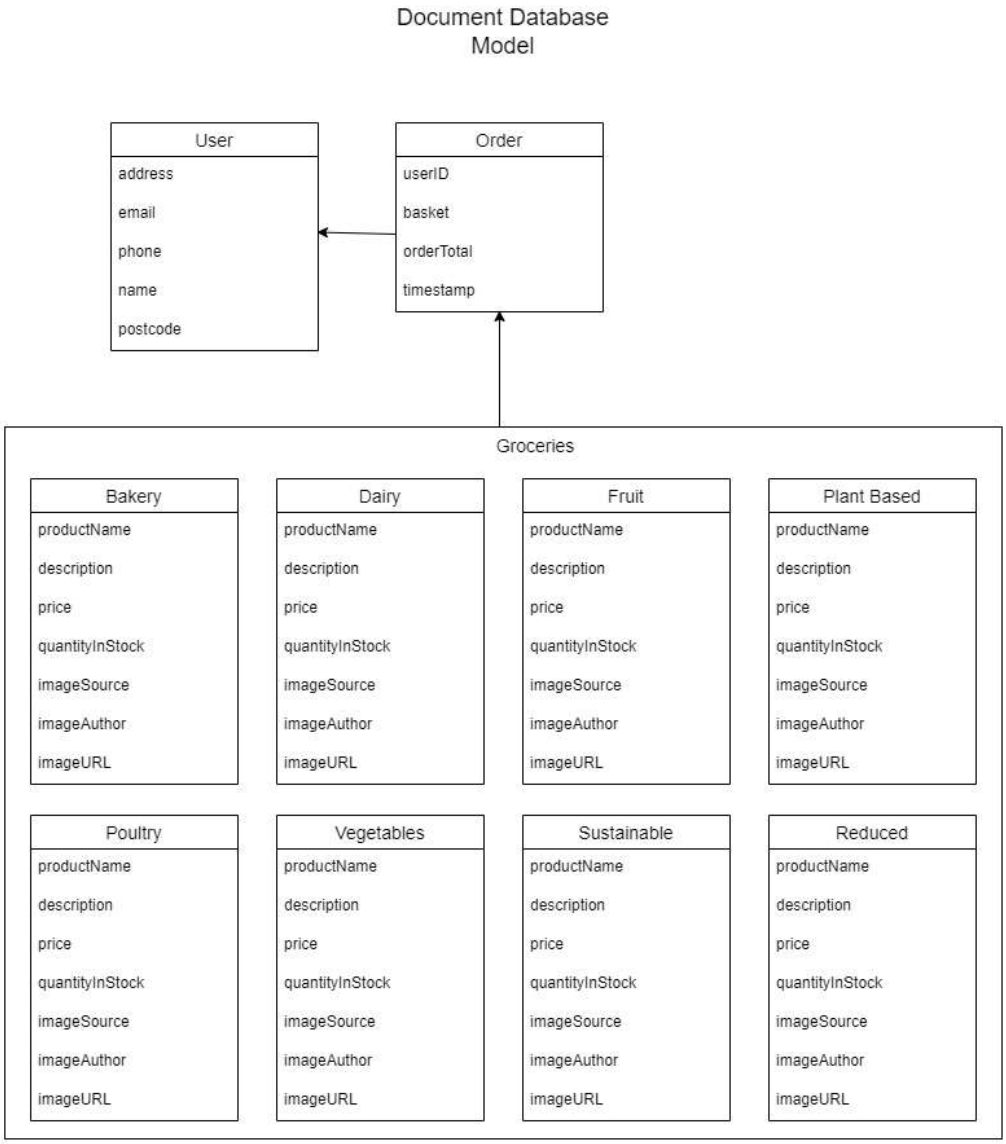


Figure 16: Document Database diagram

User

This collection refers to each user that creates an account for the service, and each document is associated with an authentication user ID, that contains the user's email and password. Every time a user creates an account their information is encrypted using HTTPS as per the firebase privacy and security guidelines (Firebase, 2022c).

Order

This collection refers to each order that each customer makes, and each entry is linked to an individual user and their basket.

The grocery collections

These collections store each of the different types of groceries within the system. However, due to this service being somewhat limited, the number of categories has been reduced.

5.7 Summary

Within this chapter, I identified how the application will look like and how users will interact with it. Then I discussed how the document database will operate and communicate with the application. Within the next chapter, I will delve into the implementation chapter.

Implementation

6.1 Introduction

Within this chapter, I will discuss how the implementation of the dark grocery delivery application went. In particular, I will explain how each of the individual screens operate and I will reflect on how certain aspects of their development went. Furthermore, I will explore the challenges I faced when implementing certain features and the communication between APIs.

As I discussed earlier in the methodology chapter, I will be taking a V-Model approach to the project's implementation. Meaning that within both this chapter and the following testing chapter, I will be discussing how its principles have affected the way in which I have developed the application.

6.2 Address Screens

This has been designed to be the first screen that the customer will view, and it displays a map of the delivery zone and it requests them to input their postcode. From there, the system will convert the customer's postcode into a longitudinal and a latitudinal value, in order to calculate the precise distance between the customer and the fulfilment centre (see figure 17). Then the system will either inform the customer if they are within or out of range of the fulfilment centre. This will be indicated in the format of either a confirmation or denial screen. For transparency reasons, I also implemented a location permissions request as throughout the app the customer's location will be used in various ways.

```
function checkAddressFromLocation() {
    // Provides API with a Google Maps Key
    Geocoder.init(googleMapsKey);
    // Provides function with the customers address
    // Then returns its longitude and latitude
    Geocoder.from(getLocate)
    .then(json => {
        // Stores the result of the geocoder function
        var location = json.results[0].geometry.location;
        // Returns the distance between the fulfilment centre and the customer in metres
        var distance = geolib.getPreciseDistance(
            {latitude: darkStoreRegion.latitude, longitude: darkStoreRegion.longitude},
            {latitude: location.lat, longitude: location.lng}
        );
        // Stores the customer's latitude and longitude values for google maps services
        dispatchHook(setCoordinates({longitude: location.lng, latitude: location.lat}))
        // Converts the distance to miles
        var miles = geolib.convertDistance(distance, 'mi');
        // Checks if the address is within 5 miles of the fulfilment centre
        if (Math.round(miles* 100)/100 <= 5.00) {
            console.log("Customer is in range");
            // Navigates to the sign in/sign up screen
            screenNavigate.navigate("Selector");
        }
        else {
            console.log("Customer is out of range");
            // Navigates to the address denial screen
            screenNavigate.navigate("Decline");
        }
    })
    .catch(error => alert(error.message));
}
```

Figure 17: Address checker function

Originally, this screen was designed to be the third view that the user would see, however, through testing I discovered that it would be more beneficial to the customer if it was the first view. Due to the fact that having a customer sign up to use the app and then being declined due to them being outside of the fulfilment centre's range would ultimately lead to a poor customer experience.

The checkAddressFromLocation function

The figure above contains the core functionality of the address screen and it consists of two node packages that work in tandem in order to identify if the customer is in range. This was arguably one of the most challenging parts of development not because the implementation was complex, but because of the fact that at the time of writing it, I did not have enough experience with using JavaScript frameworks or node packages. As a result, the process of developing a solution was a lot of trial and error. For instance, I initially tried to use react state hooks to store the user's longitude and latitude. However, this solution did not work due to the state never automatically updating to reflect the user's coordinates. Meaning that whenever a user would enter their address, it would take the state two button clicks to change. This made the address checking process rather infuriating as the system would never take your address.

Then I decided to experiment with redux, another JavaScript framework that specialises in state management. After a series of experiments, I had found a rudimentary solution that worked 90% of the time and did not 10% of the time. So, I returned to researching alternative ways to store both values. Through this I identified that storing the values in state was unnecessary, as I could have just called the react native geocoding package and stored the values in a local variable. Once I figured this out, the whole process of identifying if the customer was within range of the fulfilment centre became rather straightforward.

The displayMap function

This function handles displaying the delivery area to the user and it is built using a component called MapView from the Expo JavaScript framework (see figure 18). The MapView component uses the Google Maps platform to display the map region denoted by the initial region attribute.

```
function displayMap(){
    return(
        <View style={styles.bottomScreenLayout}>
            <MapView
                style={styles.displayMapStyle}
                // Tells the JavaScript package which maps service to use
                provider={PROVIDER_GOOGLE}
                // States the inital coordinates
                initialRegion={darkStoreRegion}
            >
                <Marker
                    // Sets the marker to the center of the map
                    coordinate={darkStoreRegion}
                >
                    {/* Sets a customer marker on map */}
                    <Image
                        source={require('../assets/icons8-warehouse.png')}
                        style={styles.marker}
                    />
                </Marker>

            </MapView>
        </View>
    )
}
```

Figure 18: Display Map Function

The implementation of this function was fairly straightforward, however, I initially had trouble displaying the map due to the Google Maps Platform requiring a billing subscription to use its services. However, once the subscription was purchased the issue never occurred again.

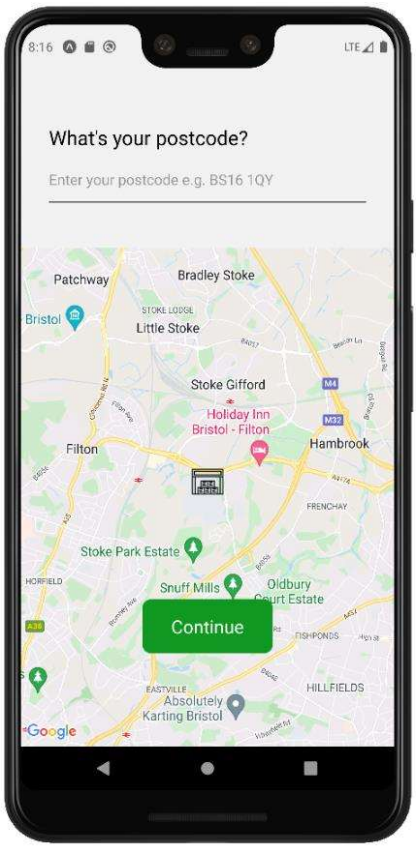


Figure 19: Address Checker Screen

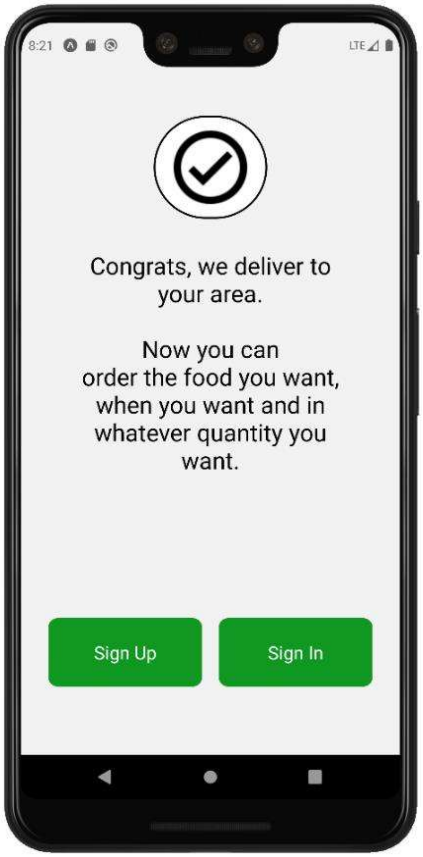


Figure 20: Address Confirmation Screen

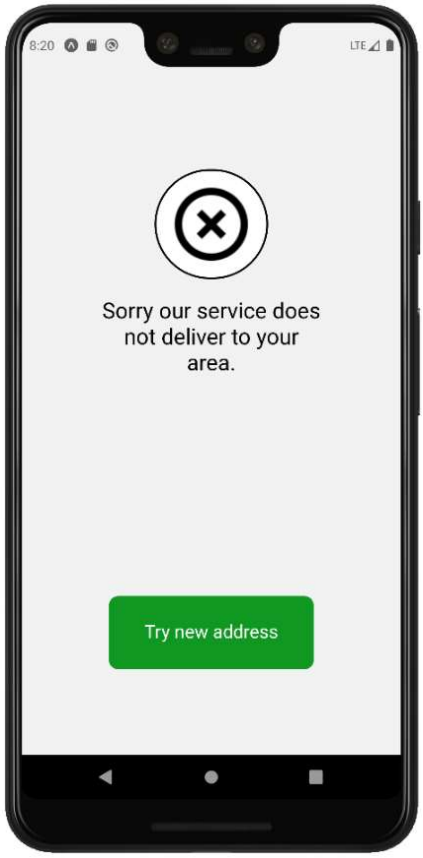


Figure 21: Address Denial Screen

6.3 Authentication Screens

Firebase

As stated in the literature review, this project uses Firebase for authentication purposes and through this, I was able to provide app users a secure method of authentication that did not require a lengthy set up. Which allowed me to register new users and log them into the app through various API calls.

Sign Up

Within this view, the system requests customers to enter their name, email, phone number and a password in to their respective text input fields. Then the system will request that the customer click on the sign up button, so that the system can use the inputted email and password to register a new account through Firebase Auth, as shown in figure 22. Then the system will redirect the customer to the home screen and submit the customer's details to Firebase, so that a new document can be registered within the users collection within the real time database (see figure 23).

```
const userSignUp = () => {
    // Clears the user data redux state if it contains any data
    dispatchHook(clearUserData())
    auth
        // This uses the email and password provided by the user to create a new account
        .createUserWithEmailAndPassword(getEmail, getPassword)
        .then((userCredential) => {
            // Stores the new user's details
            const user = userCredential.user;
            console.log("Signed up with", user.email);
            // Navigates to the home screen
            screenNavigate.navigate("Home")
            // Creates a new document in the users collection on Firebase using the new user's details
            return firestore.collection('users').doc(user.uid).set({
                name: getName,
                phone: getPhone,
                email: user.email
            })
        } )
        .catch(error => alert(error.message))
}
```

Figure 22: User sign up function

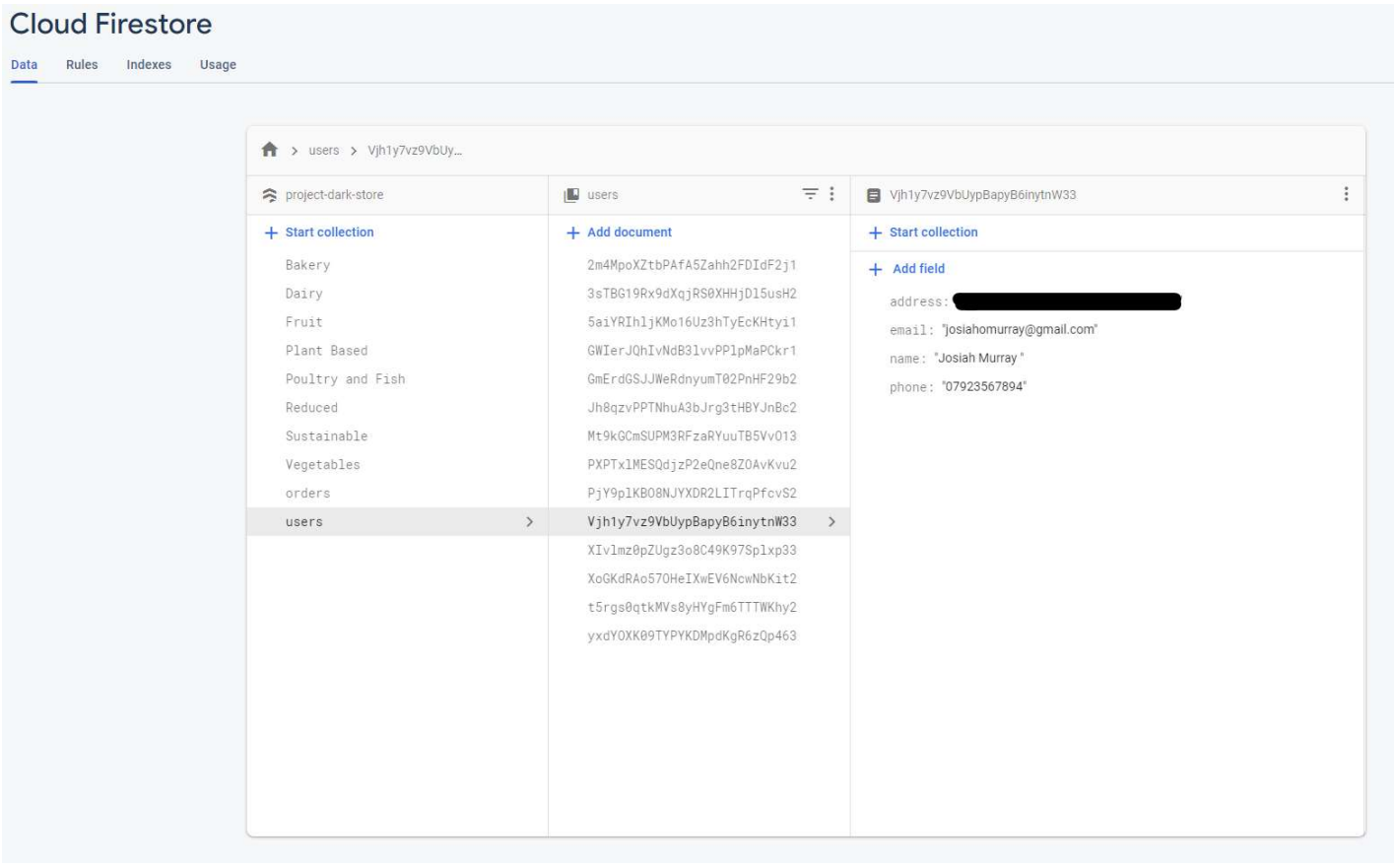


Figure 23: Firebase Users collection

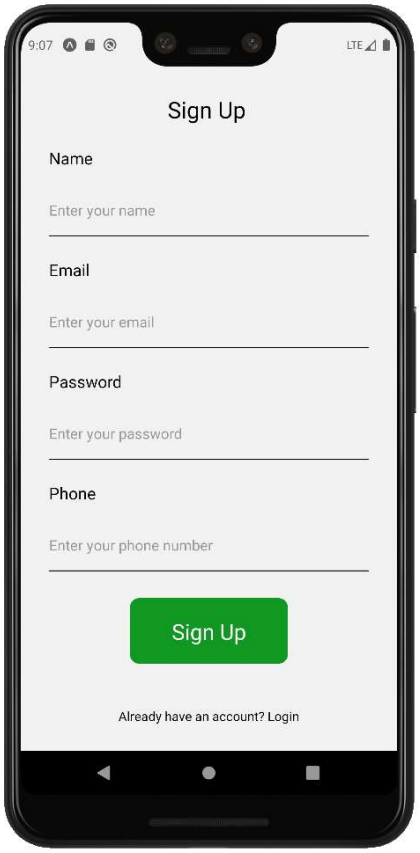


Figure 24: Sign Up Screen

Customer Address entry

Within this view, the system will request the customer to enter their address details into the various text input fields. Which will be partially filled due to the reverse geocoding function in figure 25. This function will pass the customer's latitudinal and longitudinal values to the Google Maps Geocoding API. Which will return the customer's street address, postcode and city, so that I can capture the response and store them in redux states. Furthermore, once the customer has entered their details, the system will make a request to update the customer's account details to the firebase API (See figure 26).

```
function reverseGeocodingGMaps() {
    const googleMapsKey = "                                    ";
    // Makes an API request to pull the customer's address information
    fetch('https://maps.googleapis.com/maps/api/geocode/json?address=' + lat + ',' + lng + '&key=' + googleMapsKey)
    .then((response) => response.json())
    .then((responseJson) => {
        // Stores the API fetch request's response in redux states.
        dispatchHook(setStreetAddress({streetAddress: JSON.parse(JSON.stringify(responseJson.results[0].address_components[1].long_name))}))
        dispatchHook(setPostcode({postcode: JSON.parse(JSON.stringify(responseJson.results[0].address_components[6].long_name))}))
        dispatchHook(setCity({city: JSON.parse(JSON.stringify(responseJson.results[0].address_components[2].long_name))}))
    })
    .catch(error => alert(error.message))
}
```

Figure 25: Reverse Geocoding Function

```
const addressSubmission = () => {
    const customerAddress = getStreetNumber + ' ' + getStreet + ', ' + getUserCity + ' ' + getPostal

    const userID = auth.currentUser.uid;

    firestore.collection('users').doc(userID).update({
        address: customerAddress
    })
    .then(() => {
        console.log("New Address added")
    } )
    .catch(error => alert(error.message))
}
```

Figure 26: Address Submission Function

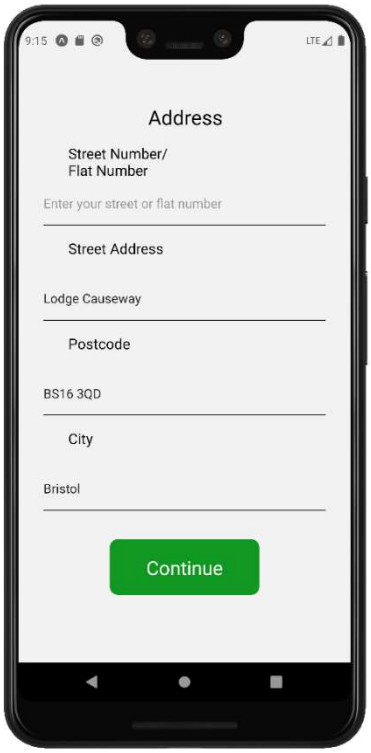


Figure 27: Address Submission Screen

Sign In

Within this view, the system will request the customer to input their registered email and password into their respective text input fields. Then the system will use the inputted email and password to make a new login attempt through Firebase Auth, as shown in figure 28. If the login credentials that the customer entered were correct and linked to an existing user, then the system will authenticate the customer and redirect them to the home screen. This will then reset the navigational flow of the app and force the navigation to start from the home screen rather than the address screen. However, if the login credentials do not match any authenticated users, then the system will request the customer to re-enter their details.

```
const userSignIn = () => {
    // Clears the user data redux state if it contains any data
    dispatchHook(clearUserData())
    auth
        // This uses the email and password provided by the user to login into the app
        .signInWithEmailAndPassword(getEmail, getPassword)
        .then((userCredential) => {
            // Stores the new user's details
            const user = userCredential.user;
            console.log("logged in with", user.email);
            // Resets the navigational stack and makes it start
            // from Home instead of the address screen
            screenNavigate.reset({
                index: 0,
                routes: [{name: 'Home'}]
            })
        })
        .catch(error => alert(error.message))
}
```

Figure 28: User Sign In Function

The customer can also choose to reset their password if they have forgotten their password as shown in figure 30. Then the customer will be sent an email if the email they entered is associated to an account (see figure 31). If the email is not, the system will inform the user that there is not an account associated to the email entered.

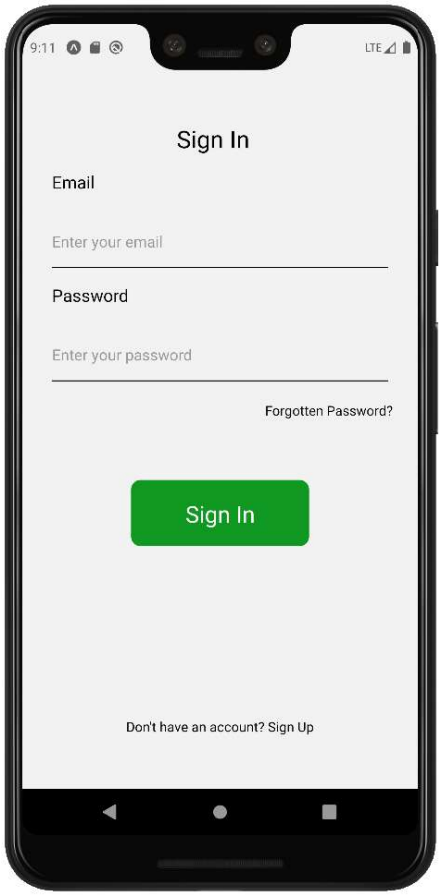


Figure 29: Sign In Screen

```
const resetPassword = () => {
    auth
        // This uses the entered email to send a password reset email to the user
        .sendPasswordResetEmail(getEmail)
        .then(() => {
            alert("Password reset email has been sent")
        })
        .catch((error) => alert(error.message))
}
```

Figure 30: Password Reset Function

Reset your password for project-782131384356

N noreply@project-dark-store.firebaseapp.com <noreply@project-dark-store.firebaseapp.com>
01:12

To: [redacted]@gmail.com

Hello,

Follow this link to reset your project-782131384356 password for your [redacted]@gmail.com account.

https://project-dark-store.firebaseapp.com/ /auth/action?
mode=resetPassword&oobCode=ThmU2F3r_jDF2D0iB4BTjoQbKg1vs_tjYMXz2SWDEysAAAGAIENteg&apiKey=[redacted]
[redacted]&lang=en

If you didn't ask to reset your password, you can ignore this email.

Thanks,

Your project-782131384356 team

Figure 31: Password Reset Email

6.4 Home, Category & Product Screens

Home

Within this view, the system displays the various categories that the grocery service has to offer and a select few products from each. Within each category section, the customer can scroll horizontally to view the individual products under the category banner (See Figure 32).

Figure 32: Display Product Function

```
const productDisplay = (categoryList) => {
    return (
        <View style={{paddingTop: "4%"}}>
            {/* Displays each item in an array and applies the same attributes to each */}
            <FlatList
                // Forces the flatlist to be displayed horizontally rather than the vertical default
                horizontal
                // Removes the scroll bar from the bottom of each view
                showsHorizontalScrollIndicator={false}
                data={categoryList.slice(0,4)}
                // For each item do the following
                renderItem={({item}) => (
                    <View>
                        <TouchableOpacity onPress={() => {
                            dispatchHook(setSelectedProduct({selectedProduct: item.productName}))
                            screenNavigate.navigate('Product')
                            //selectedItem = item.productName
                        }}>
                            <Image style={styles.productImage} source={{uri: item.imageUrl}}/>
                            <Text style={styles.productTitles}> {item.productName}</Text>
                            <Text style={{paddingLeft: "40%"}}>£{item.price}0</Text>
                        </TouchableOpacity>
                    </View>
                )}
            />
            {/* Displays a line on the screen */}
            <View style={{borderBottomColor: 'rgba(0, 0, 0, 0.2)', borderBottomWidth: 1, marginRight: "5%", marginLeft: "5%"}}/>
        </View>
    )
}
```

The customer can then view each of the individual product details. Which are stored in the firebase database (see figure 34) and are stored in redux state arrays. Furthermore, the customer can switch between the home screen and the basket screen through the tab navigation bar displayed at the bottom of the screen. If either are clicked, the system will redirect the customer to the respective screen. In addition, customers can also select the silhouette icon in the top right corner to view their account details. By doing so, the system will redirect the user to the accounts page.

During development of the home screen, I did not encounter too many issues except one regarding the vertical scroll feature. The issue was a semantic error, where I had only implemented a FlatList component, which at the time I did not know would only control the items present in the array rather than the whole screen. As a result, to solve this problem, I had to undertake some research to identify a suitable fix and discovered that I would need to wrap the FlatList component in a ScrollView component, as that will allow for the whole screen to be viewed.

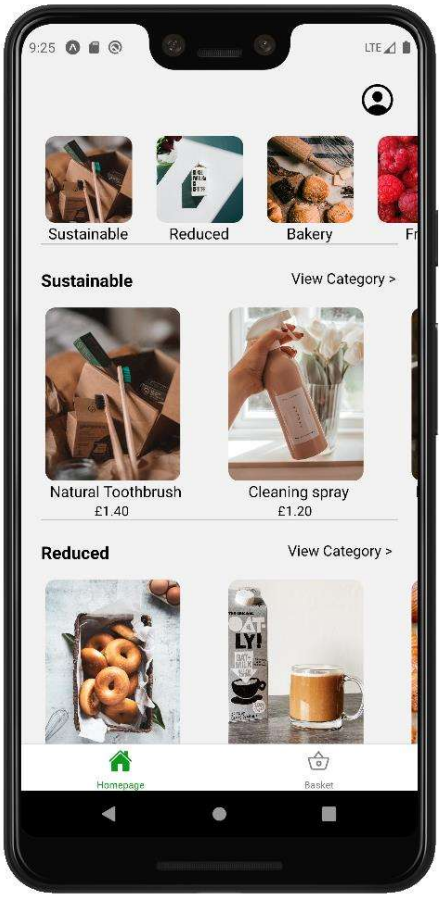


Figure 33: Home Screen

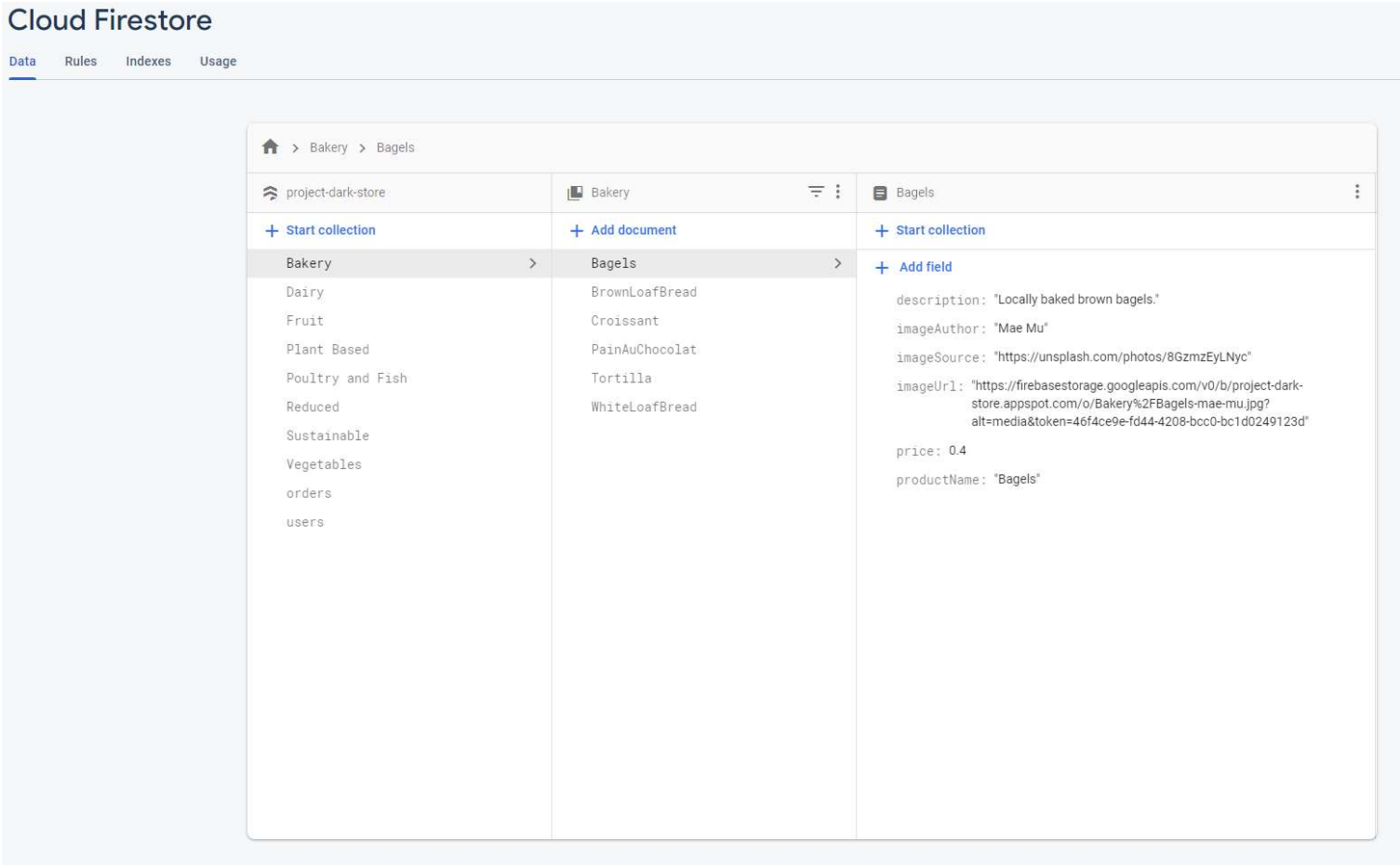


Figure 34: Firebase Grocery data

Category

Within this view, the customer can view each of the products under the category chosen by the customer on the home screen. If a customer clicks on a product, then the system will redirect the customer to the associated product details screen. In addition, the customer can also use the tab navigation bar to either return to the home screen or the basket screen.

During development of the category screen, I only encountered a single issue regarding the tab navigation bar. The issue entailed navigating to the wrong screen if a certain combination of tab navigation buttons are pressed. The bug would make the home button return to the category screen rather than the home screen, if the customer clicked on the basket button and then clicked on the home button. Therefore, to solve this problem, I had to change the stack navigation hierarchy as the category was immediately after the home screen in the stack.

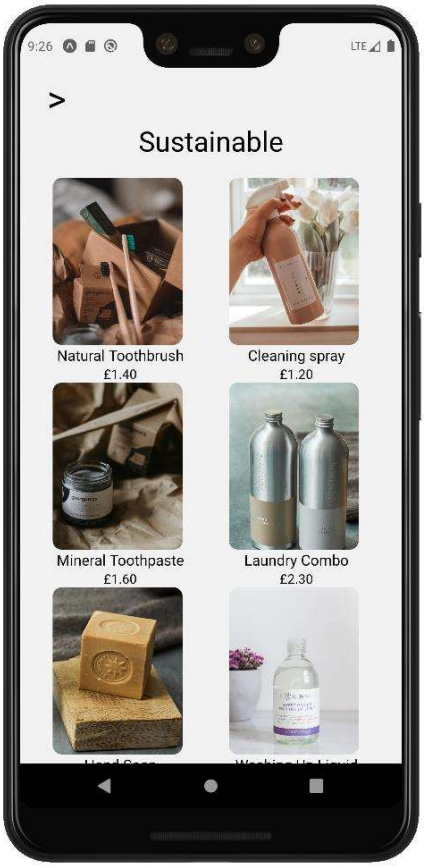


Figure 35: Category Screen

Product

Within this view, customers can view product details, select a quantity they want, and add products to their basket. If a customer wanted to increase or decrease the quantity of a product they want, then they can click on the two buttons at the bottom of the screen. Then if a customer clicked the add to basket button, then the system would append the product details and the selected quantity to a redux state array, which can be seen in figure 36 below.

```
export const basketSlice = createSlice({
    name: 'basket',
    initialState: {
        groceryBasket: [],
        finalPrice: 0,
        totalPrice: 0,
        deliveryTotal: 0,
    },
    reducers: {
        addToBasket: (state, action) => {
            // Check if the the passed product is already in the basket
            const product = state.groceryBasket.find(product => product.productName == action.payload.searchResult.productName)
            // if product is already in the basket just increase quantity
            if (product) {
                product.quantity += action.payload.quantity
            }
            else
            {
                state.groceryBasket.push({
                    basketID: Math.floor(Math.random() * 1000),
                    productName: action.payload.searchResult.productName,
                    description: action.payload.searchResult.description,
                    quantity: action.payload.quantity,
                    originalPrice: parseFloat((action.payload.searchResult.price).toFixed(1)),
                    price: parseFloat((action.payload.searchResult.price * action.payload.quantity).toFixed(1)),
                    imageAuthor: action.payload.searchResult.imageAuthor,
                    imageSource: action.payload.searchResult.imageSource,
                    imageUrl: action.payload.searchResult.imageUrl
                });
            }

            console.log(state.groceryBasket);
            // Displays an alert at the bottom of the screen
            // THIS IS ONLY VIEWABLE ON ANDROID DEVICES
            ToastAndroid.show('Added to basket', ToastAndroid.SHORT);
        },
```

Figure 36: Basket Slice

The redux state array in question is called groceryBasket and belongs to a slice of a store. The slice contains a host of manipulative functions called reducers that can affect its various states. These reducers can be viewed as event handlers that accept both a state (groceryBasket) and an action. Actions are records of current events occurring in the application, and in this application's case are customer interactions with the products in the basket. The add to basket reducer within figure 36 takes the product stored within the action and checks if there is an existing instance of the product in the basket. If there is, the existing product's quantity will be increased rather than having redundant products being appended to the basket. If there is no instance of the product, then the product stored in the action will be appended to the state array.

However, in order to observe if an event has been triggered, I had to implement an event listener, which in redux is called a store. Then once an event has occurred, the store will receive a notification that it has happened through a state hook called useDispatch and call the associated reducer function (see figures 37 & 38).

```
<View style={{flexDirection: "row"}}>
    {/* Add to basket button */}
    <TouchableHighlight
        onPress={() => {
            // Tells the store that a button has occurred
            dispatchHook(addToBasket({searchResult, quantity: count}))
            screenNavigate.goBack()
        }}
        style={styles.button}
        underlayColor="#DDDDDD"
        backgroundColor="#99D98C">
        <Text style={styles.buttonText}>Add to Basket</Text>
    </TouchableHighlight>
</View>
```

Figure 37: Add to basket Button

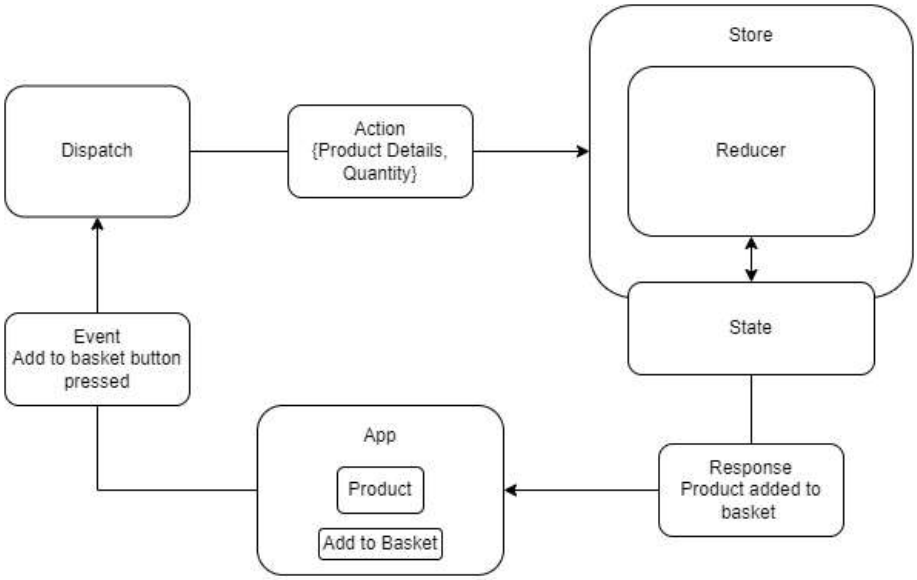


Figure 38: Redux Flow Diagram

Figure 39: Product Screen

6.5 Basket Screen

Within this view, customers can view the products within their basket, update a product's quantity, clear their basket, remove an item from the basket, view the order total and make an order. The individual products within the basket is displayed by outputting the groceryBasket state array in a FlatList component. However, originally I had an issue outputting the state array as I was trying to directly call the state. But, due to the way redux stores operate, the only way to output a state array is by using a useSelector hook. Which allows for a specific state to be called outside of a store slice.

Alter Products

In the event that a customer wants to make changes to a product in their basket, then they can either click on the increase or decrease quantity buttons or click on the delete product button.

```
updateProductQuantityAdd: (state, action) => {
    // Locates the selected product
    const product = state.groceryBasket.find(product => product.productName == action.payload.name)
    if (product)
    {
        // Increase both quantity and price
        product.quantity += 1
        product.price = parseFloat((product.quantity * product.originalPrice).toFixed(1))
    }
},
updateProductQuantitySubtract: (state, action) => {
    // Locates the selected product
    const product = state.groceryBasket.find(product => product.productName == action.payload.name)
    if (product)
    {
        if (product.quantity !== 1)
        {
            // Reduces both quantity and price
            product.quantity -= 1
            product.price = parseFloat((product.quantity * product.originalPrice).toFixed(1))

        }
    }
},
```

Figure 40: Increase and decrease product quantity function

If the customer chooses to change the quantity of a product, then that will commence an event informing the store to call one of the update product quantity reducers. Both of the reducers will search for the product that the customer has selected, and it will either reduce or increase both the product's quantity and the product total price (see figure 40).

If the customer chooses to remove a product from the basket, then that will commence an event informing the store to call the remove from basket reducer. This will search the state array to find the product that the customer wants to delete and create a new version of the array without the selected product (see figure 41).

```
removeFromBasket: (state, action) => {
    state.groceryBasket = state.groceryBasket.filter(
        // Create a new array without the selected item
        product => product.basketID !== action.payload.selectedID
    );

    console.log(state.groceryBasket);

    ToastAndroid.show('Item removed from basket', ToastAndroid.SHORT);

},
```

Figure 41: Remove Item from basket reducer

Clear Basket

If the customer wants to clear the entire basket, then they can press on the empty basket button. By doing so, this will commence an event and inform the store to call the clear basket reducer. This reducer will either set the state array to an empty array and alert the customer that the basket has been cleared, or it will inform the customer that they cannot clear an empty basket.

```
clearBasket: (state, action) => {
    if (state.groceryBasket.length == 0){
        ToastAndroid.show('Basket is empty', ToastAndroid.SHORT);
    }
    else{
        state.groceryBasket = []
    }
    ToastAndroid.show('Basket Emptied', ToastAndroid.SHORT);
},
```

Figure 42: Clear basket function

Order

If the customer decides to make an order via pressing the order button, then the function in figure 43 will be called. The submit order function will check if the basket is empty and in the event it is not, the customer's order will be pushed to the database. Then the customer will be redirected to the order confirmation screen, which will reset the navigational stack so that once the order has been made, the customer cannot return back to editing their basket. In the event that the basket is empty then the customer will be informed that they must add items to the basket before attempting to an order.

```
function submitOrder(){
    // Sends a request to obtain the user's unique identifier
    const userID = auth.currentUser.uid;
    if (basket.length !== 0) {
        // Pushes the new order to the firestore database
        firestore.collection('orders').add({
            userID: userID,
            basket: basket,
            orderTotal: parseFloat(orderTotal).toFixed(2),
            timestamp: firebase.firestore.FieldValue.serverTimestamp()
        })
        // Stores the order totals
        dispatchHook(setTotalPrice({totalPrice: price}))
        dispatchHook(setDeliveryTotal({deliveryTotal: deliveryPrice}))
        dispatchHook(setFinalPrice({finalPrice: orderTotal}))
        // This will reset the navigational stack so that it starts from the order screen
        screenNavigate.reset({
            index: 0,
            routes: [{name: 'Order'}]
        })
    }
    else {
        // Checks which platform the customer is running the app on
        if (Platform.OS === 'android') {
            // This will only show up on android
            ToastAndroid.show('Basket is empty', ToastAndroid.SHORT)
        }
        else {
            alert("Basket is empty")
        }
    }
}
```

Figure 43: Submit order function

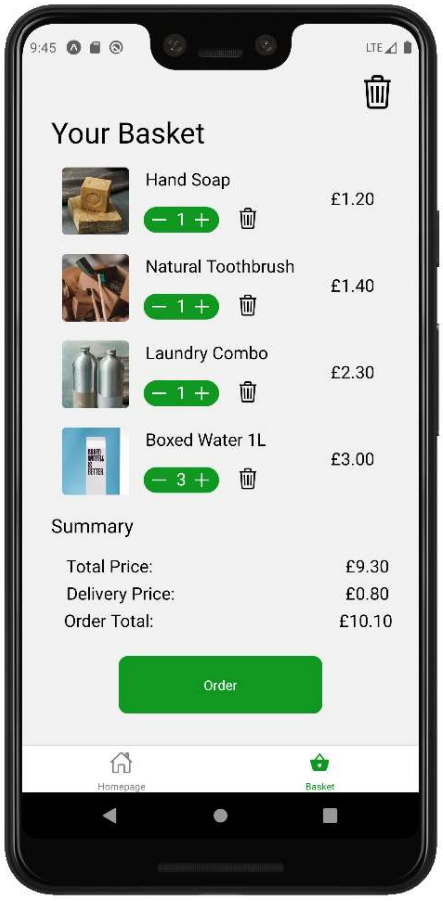


Figure 44: Basket Screen

6.6 Order Confirmation & Order Summary Screens

Order Confirmation

Within this view, the customer can view their order progress and when it is estimated to be delivered to them. The order progress bar was built using a node package called react-native-indicator, this package provided a visual progress indicator component that would display each order milestone (see figures 45 and 46). The package's documentation instructed that each milestone could be reached through pressing a series of buttons that corresponded to react states. However, due to the order having to progress automatically, I had to reconfigure how the milestones would be reached. I implemented this by using a useEffect hook and incrementing a react state after a delay, which allowed me to emulate multiple button presses.

```
<View style={{flex:1, marginLeft: "10%"}}>
    <StepIndicator
        customStyles={customStyles}
        currentPosition={phase}
        stepCount={4}
        labels={labels}
        direction={'vertical'}
    />
</View>
```

Figure 45: Display order progress

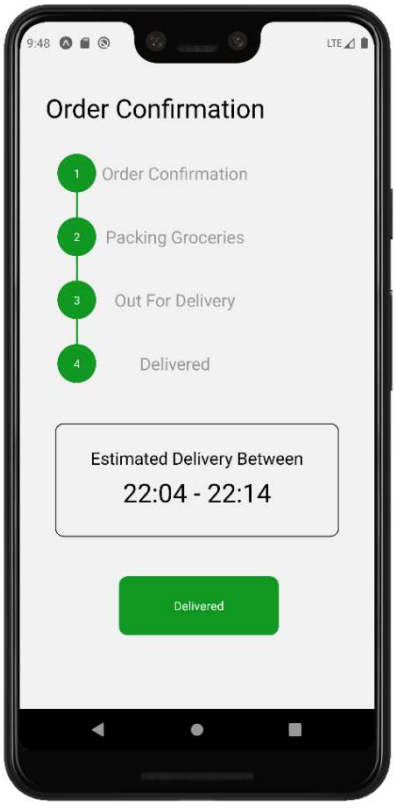


Figure 46: Order Confirmation Screen

The delivery estimate function was built using the Google Maps Distance Matrix API, which accepted various arguments such as the mode of transport that will be used and the location of both the fulfilment centre and the customer's delivery address. Then the API will calculate the travel distance between the two locations via bicycle and return the both the distance in kilometres and the estimated time it would take to travel as a JSON object (See figure 47). Then to display the estimated delivery time, I calculated the difference between the current time and the estimated delivery time (See figure 48).

```
async function getEstimatedDeliveryTime(){
    const googleMapsKey = "AIzaSyDDRYyy-kCd1dNrRH-eeQ4YHhQ4FoNRYIo";

    // Fulfilment centre address
    const fulfilmentCentreAddress = "Coldharbour Ln, Stoke Gifford, Bristol BS16 1QY"

    // Customer Delivery Address
    const customerAddress = accountDetails[0].address

    // Google Maps Distance Matrix API Call
    // This identifies the distance between
    // the fulfilment centre and the customers address via Bike
    await fetch('https://maps.googleapis.com/maps/api/distancematrix/json?origins=' + fulfilmentCentreAddress +
    '&mode=bicycling&destinations=side_of_road%' + customerAddress + '&key=' + googleMapsKey)
    .then((response) => response.json())
    .then((responseJson) => {
        dispatchHook(setEstimatedDeliveryTime({estimatedTime: JSON.parse(JSON.stringify(responseJson.rows[0].elements[0].duration.text))}))
    })
    .catch(error => alert(error.message))
}
```

Figure 47: Get estimated delivery time function

```
const getDeliveryTimeFrame = () => {
    // Split the estimated delivery time string
    var separateEstimate = estimatedDeliveryTime.split(" ")
    var date = new Date();
    // Calculating a delivery time from the current time and the estimated time
    var estimatedTimeFrame = new Date(date.getTime() + parseInt(separateEstimate[0])*60000);
    // Calculating 10 minutes extra delivery time
    var extraTime = new Date(estimatedTimeFrame.getTime() + 10*60000);
    // Returning the estimated delivery time and the delivery net time
    return estimatedTimeFrame.getHours() + ":" + (("0" + estimatedTimeFrame.getMinutes()).slice(-2))
        +  " - " + extraTime.getHours() + ":" + (("0" + extraTime.getMinutes()).slice(-2))
}
```

Figure 48: Get estimated delivery time 24hr function

Order Summary

This view will appear once an order has been delivered and will allow customers to review their order. On this view customers can see each of the products that they ordered and the total price of the order.

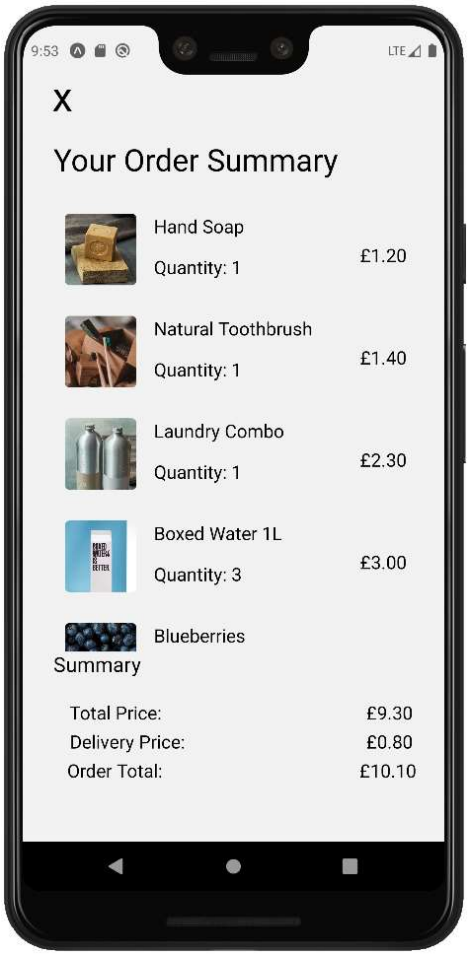


Figure 49: Order Summary Screen

6.7 Account Details Screen

Within this view, customers can alter their account details, change their password and sign out of the app. If a customer decides to change their account details present in the text input field, they can replace the existing value and press the save button. By doing so, this will call the updateUserData function in figure 50, which will make a request to the firebase API to update the document values associated with the customer. However, if the customer pressed the save button without making any changes, the function will return an alert requesting the customer to alter their details.

```
const updateUserData = () => {
    // Make a request to Firebase Auth to obtain the user's account ID
    const userID = auth.currentUser.uid;
    // Checks if any changes to the customer details have been made
    if(updateName !== userDetails[0].name || updatePhone !== userDetails[0].phone)
    {
        // Updates the name and phone attributes of the customer's account details
        firestore.collection('users').doc(userID).update({
            name: updateName,
            phone: updatePhone
        })
        .then(() => {
            alert("Your details have been updated.")
        })
        .catch((error) => {
            alert(error.message)
        })
    }
    // If no changes have been made
    else{
        alert("Please make changes")
    }
}
```

Figure 50: Update user details function

If the customer presses the change password button, this will call the changePassword function that will make a request to the Firebase Auth API to send the customer a password reset email similar to the one in figure 31. Then they will be signed out of their account through a Firebase API request and returned to the sign in screen.

Displaying the user's data

The user account data is handled by an asynchronous function called on the home screen if the account button is pressed. This function will obtain the current customer's account ID and use it to pull their account details from the users collection within the firebase database (see figure 51). Once the details have been pulled, they will be stored in a redux state array through a reducer function shown in figure 52. Then finally, when the customer presses the account details screen, the customers data is displayed using a useSelector function.

```
const userData = async () => {
    // Make a request to Firebase Auth to obtain the user's account ID
    const userID = auth.currentUser.uid;
    const details = []
    // An asynchronous request for pulling customer account data
    await firestore.collection('users').doc(userID).get()
    .then((doc) => doc.data())
    .then((docData) => {
        console.log(docData)
        details.push({
            id: userID,
            name: docData.name.toString(),
            phone: docData.phone.toString(),
            email: docData.email.toString(),
            address: docData.address.toString(),
        })
    })
    .catch((error) => console.log(error.message))
    dispatchHook(getUsersData({details: details[details.length - 1]}))
}
```

Figure 51: Pull user data function

```
getUsersData: (state, action) => {
    // Checks if the user data has already been pulled
    const checkUser = state.user.find(users => users.name === action.payload.details.name)

    // if the user data has not been pulled or has been updated
    if(!checkUser){
        state.user.push({
            id: action.payload.details.id,
            name: action.payload.details.name,
            phone: action.payload.details.phone,
            email: action.payload.details.email,
            address: action.payload.details.address
        })
    }

},
```

Figure 52: Redux Reducer for storing user data function

During the development of this view, I encountered multiple issues with displaying the customer details due to how API requests are made. The first solution I thought of was to make the function call asynchronous as that would give the API enough time to be called. However, this did method failed to work due to redux state not having enough time to update because of the navigation to the next screen. Therefore, the solution I found was to set a 5 second delay on when the function would navigate to the accounts screen (see figure 53).

```
<View style={{paddingLeft: "85%", paddingTop: "14%",}}>
    <Ionicons name="person-circle-outline" size={40}
        onPress={() => {
            userData()
            setTimeout(() => screenNavigate.navigate('Account'), 500)
        }}
    />
</View>
```

Figure 53: Delay navigation

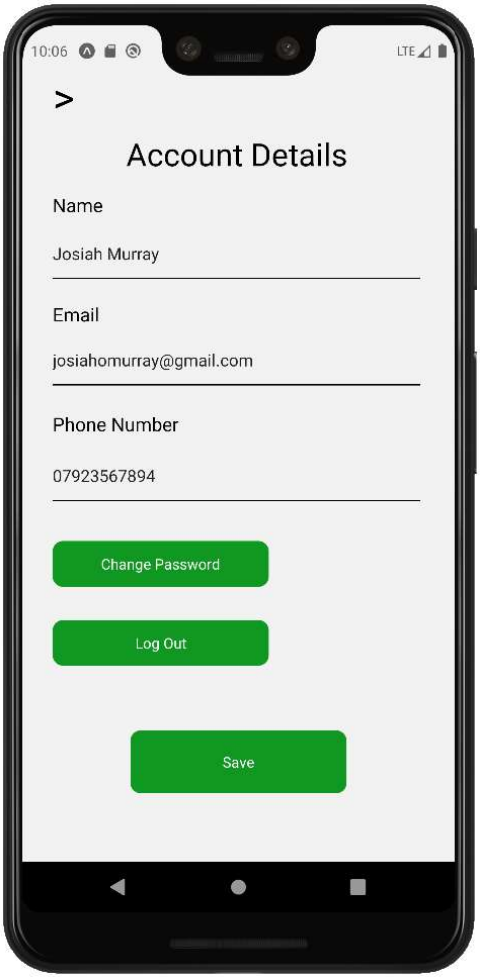


Figure 54: Account Details Screen

Testing

7.1 Introduction

In this chapter, I will test the dark grocery delivery application against the functional and non-functional requirements defined in the requirements chapter. Then I will discuss why certain tests failed and my approach to solving them.

7.2 Functional Requirement Testing

Test ID	FR/NFR ID	Pass/Fail	Expected Outcome	Actual Outcome
TF-01	FR-1	Pass	If user's postcode is within the 5-mile range, then they should be redirected to sign up or sign in.	User was redirected to the sign in/sign up screens.
TF-02	FR-1	Pass	If user's postcode is outside the 5-mile range, then they should be redirected to a decline screen.	User was redirected to the delivery decline screen and asked to try another address.
TF-03	FR-2	Fail	If the user's details are valid, then they should be redirected to the address entry screen.	User's details were valid, but the phone number validation flagged the entered phone number as invalid.
TF-04	FR-3	Pass	If the user's email and password are associated to an existing account, they should be redirected to the home screen.	User's email and password were valid, and they were redirected to the home screen.
TF-05	FR-3	Pass	If the user's email and password are not associated to an existing account, they should be informed that their email or password is invalid.	User's details were rejected, and an alert message was displayed, informing the user that their login credentials are incorrect.
TF-06	FR-4	Pass	If the user's address details are associated with a valid UK address, the user should be redirected to the home screen	User's address was valid, and they were redirected to the home screen.

TF-07	FR-5	Pass	If the user closes the app and reopens it, they should automatically be directed to the home screen.	App was reopened and started from the home screen.
TF-08	FR-7	Pass	If the user scrolls down the home screen, they should be able to see each of the categories and a select range of products from each.	The various categories and their products were rendered when the user scrolled down the home screen.
TF-9	FR-7	Pass	If the user scrolled horizontally on a category, they should be able to view a select few products from the category.	Four products could be seen when the user horizontally scrolled through the bakery category.
TF-10	FR-8	Fail	If the user pressed on a category, they should be redirected to the associated category screen.	User was redirected to the bakery category instead of the fruit category.
TF-11	FR-9	Pass	If the user pressed either the home or basket button on the tab navigation bar, then they will be redirected to their screens.	When user pressed the basket button, they were directed to the basket screen and vice versa with the home button.
TF-12	FR-11, FR-12	Pass	If the user added a product to the basket from a product details screen, then the basket should update to reflect the recent change.	The basket updates every time a new product is added.
TF-13	FR-13	Pass	If the user makes an order, then the app should redirect them to an order confirmation screen.	When order button is pressed, user is sent to order confirmation screen and can see when their order is supposed to be delivered.

Table 3: Functional Requirements Testing

7.3 Failed Functional Tests

TF-03

This test failed due to how the phone validation function was formatted, as shown in figure 55, the regex was only made to accept numbers that contained 11 characters but did not accept numbers that started with the UK country code (+44). Therefore, I needed to alter the function to prevent an error occurring if a customer typed the country code rather than zero. This new function can be viewed in figure 37 and it accepts both +44 or 0 numbers followed by 9 or 10 digits.

```
const phoneValidation = (phone) => {
    // Checks if value is 11 digits
    if (/^\d{11}$/.test(phone))
    {
        return true;
    }
    else{
        alert("Invalid Phone Number")
    }

}
```

Figure 55: Wrong phone validation

```
const phoneValidation = (phone) => {
    // Allows for numbers that start with UK country code or 0
    // Then it should be followed by 9 or 10 digits
    if (/^(?:0|\+?44)(?:\d\s?){9,10}$/.test(phone))
    {
        return true;
    }
    else{
        alert("Invalid Phone Number")
    }

}
```

Figure 56: Revised phone validation

TF-11

This test failed due to the selected category being stored in a global variable that would be imported and used to display the correct category screen (See Figure 57). When testing, this global variable would work for the first 4-5 category selections but would keep repeating the last category rather than the new one. Therefore, I thought of two solutions, one that would use react state hooks and one that would use a redux state with a reducer. Ultimately, I chose to use a redux state with a reducer as this proved to produce the most effective result when updating the current state value (See figures 58 & 59).

```
<FlatList
    horizontal
    showsHorizontalScrollIndicator={false}
    data={categories}
    renderItem={({item}) => (
        <TouchableOpacity
        onPress={() =>
            {
                screenNavigate.navigate("Category")
                selectedCategory = item.categoryName
            }
        }>
            <Image style={styles.categoryImage} source={{uri: item.imageUrl}}/>
            <Text style={styles.categoryTitles}> {item.categoryName}</Text>
        </TouchableOpacity>
    )}
/>
```

Figure 57: Broken Category selection

```
setSelectedCategory: (state, action) => {
    state.selectedCategory = action.payload.selectedCategory
},
```

Figure 58: Category selection reducer

```
<FlatList
    horizontal
    showsHorizontalScrollIndicator={false}
    data={categories}
    renderItem={({item}) => (
        <TouchableOpacity
        onPress={() =>
            {
                dispatchHook(setSelectedCategory({selectedCategory: item.categoryName}))
                screenNavigate.navigate("Category")
            }
        }>
            <Image style={styles.categoryImage} source={{uri: item.imageUrl}}/>
            <Text style={styles.categoryTitles}> {item.categoryName}</Text>
        </TouchableOpacity>
    )}
/>
```

Figure 59: Revised Category selection

7.4 Non-Functional Requirement Testing

Test ID	FR/NFR ID	Pass/Fail	Expected Outcome	Actual Outcome
TNF-01	NFR-1	Pass	If the user selects a product from the home screen, they should be directed to the product's details page and should be able to go back.	User selected blueberries and was taken to the blueberry product page. From there, the user was able to navigate back home via the back button.
TNF-02	NFR-2	Pass	If the user submits their sign up request form, the address details should automatically appear on the address entry screen.	User pressed sign up button and their street address, postcode and city were displayed in their respective input fields.
TNF-03	NFR-3	Pass	User should be able to use the app on both an IPhone and an android phone.	User was able to navigate and order groceries on both platforms.
TNF-04	NFR-4	Pass	If the user signs into the app they should not be forced to sign out at any point. Also, if the user closes the app, they should not have to sign back in.	User was able to continue shopping with the app without having to sign back in at any point. Even if they closed the app.
TNF-05	NFR-5	Pass	Multiple user accounts should be able to be made and new products should be able to be added to the database and pulled.	5 new users were created, and 5 new products were added to the database. This caused no issues.
TNF-06	NFR-6	Fail	User should be able to use the app without any inconstancies over 2 devices.	App had no inconsistencies as intended on large phones but had some CSS formatting inconsistencies on smaller phones.
TNF-07	NFR-7, NFR-13	Pass	Products should render within 5 seconds of window load.	User selected Fruit category and all products rendered within 4 seconds.
TNF-08	NFR-8	Fail	Selecting a product should redirect users to the correct product page.	Picking an apple from the home page, displayed the correct product details page. However, after selecting yoghurt from the dairy category page, the apple product details page was displayed instead.
TNF-09	NFR-10	Pass	A user should be able to select a product from the home screen, then add that product to the basket and finally finalise an order.	User was able to select a croissant from the home screen, then add 4 to the basket. Once the croissants were in the basket, they were able to make an order.

TNF-10	NFR-11	Pass	A user should be able to add multiple groceries to the basket without having any performance issues.	User was able to 20 individual products in their basket, and this caused no performance issues.
TNF-11	NFR-12	Pass	A user should be redirected to the order confirmation page as soon as the order button is pressed.	User was redirected to the order screen when the order button is pressed on the basket screen.

Table 4: Non-Functional Requirements Testing

7.5 Failed Non-Functional Tests

TNF-06

This test failed due to how the CSS of the application was formatted as originally, the CSS was designed with larger phones in mind. Which can be seen in figure 60, where the quantity and add to basket buttons cannot be seen. Furthermore, during development the application was tested on a Pixel 3 XL and a Samsung Galaxy S10 Plus. Therefore, to attempt to solve this issue I had to reformat the entire app to accommodate for smaller screen sizes. For this test, I tested the app using an emulated smaller sized phone offered by Android Studio. For the result see figure 61.

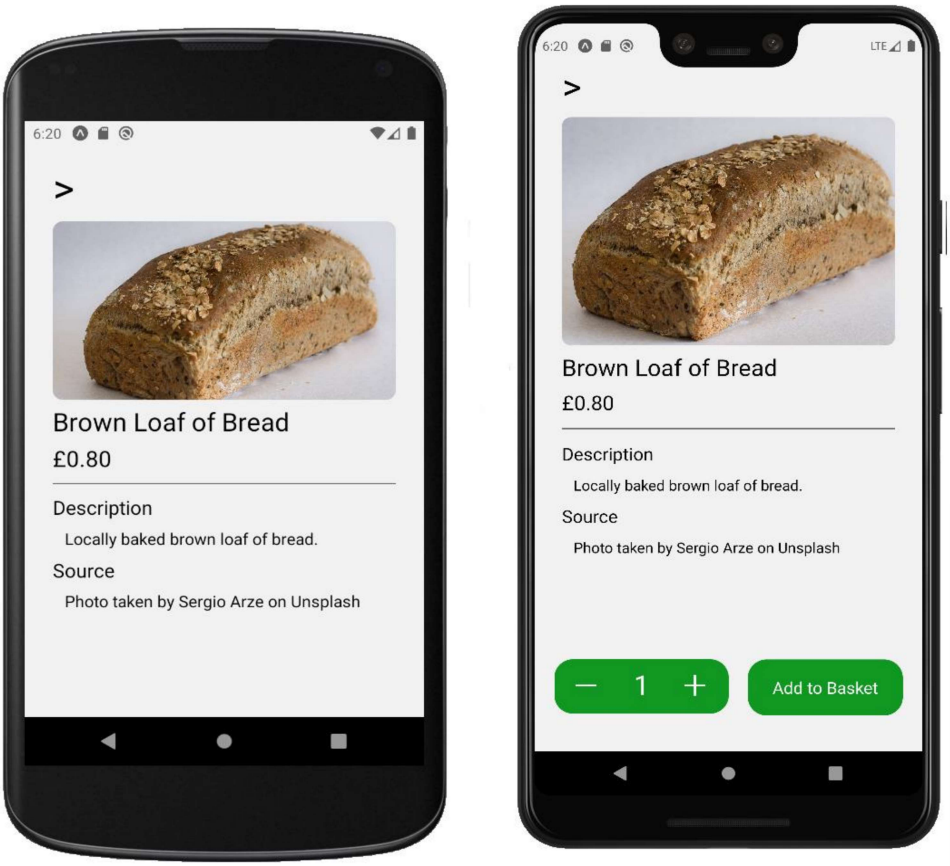


Figure 60: Original CSS for product screen

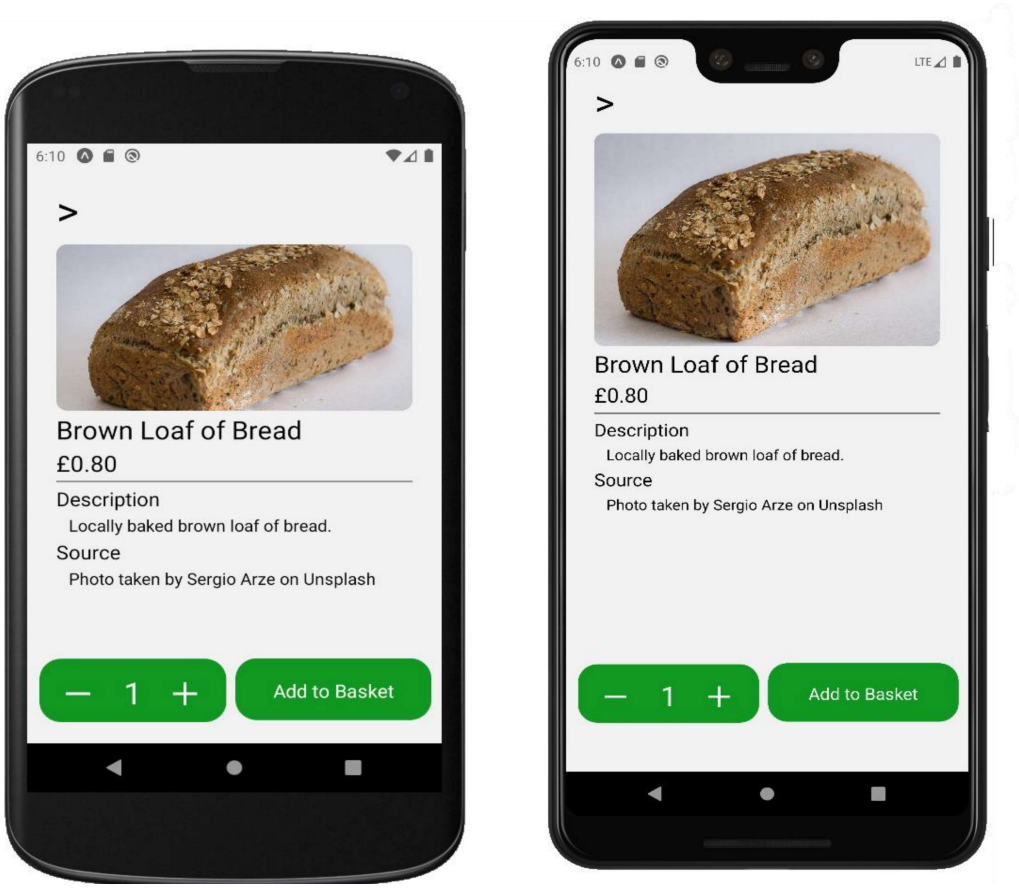


Figure 61: Revised CSS for product screen

TNF-08

This test failed due to a similar reason to the bug in TF-11, where the selected product being updated and stored in a global variable that would be used to display the correct product details screen (See Figure 62). Additionally, I was using two separate global variables for the products on both the home and category screens. This led to confusion when both variables were not reset. Therefore, I had to solutions, one being clearing each variable after a product had been set or one where a redux state and reducer were used. In the end I decided to implement the redux state to prevent the possibility of the clearing variable function failing in the future (See figures 63 & 64).

```
{/* Displays each item in an array and applies the same attributes to each */}
<FlatList
    // Forces the flatlist to be displayed horizontally rather than the vertical default
    horizontal
    // Removes the scroll bar from the bottom of each view
    showsHorizontalScrollIndicator={false}
    data={categoryList.slice(0,4)}
    // For each item do the following
    renderItem={({item}) => (
        <View>
            <TouchableOpacity onPress={() => {
                screenNavigate.navigate('Product')
                selectedItem = item.productName
            }}>
                <Image style={styles.productImage} source={{uri: item.imageUrl}}/>
                <Text style={styles.productTitles}> {item.productName}</Text>
                <Text style={{paddingLeft: "40%"}}>£{item.price}0</Text>
            </TouchableOpacity>
        </View>
    )}
/>
```

Figure 62: Broken product selection

```
setSelectedProduct: (state, action) => {
    state.selectedProduct = action.payload.selectedProduct
},
```

Figure 63: Product selection reducer

```
{/* Displays each item in an array and applies the same attributes to each */}
<FlatList
    // Forces the flatlist to be displayed horizontally rather than the vertical default
    horizontal
    // Removes the scroll bar from the bottom of each view
    showsHorizontalScrollIndicator={false}
    data={categoryList.slice(0,4)}
    // For each item do the following
    renderItem={({item}) => (
        <View>
            <TouchableOpacity onPress={() => {
                dispatchHook(setSelectedProduct({selectedProduct: item.productName}))
                screenNavigate.navigate('Product')
            }}>
                <Image style={styles.productImage} source={{uri: item.imageUrl}}/>
                <Text style={styles.productTitles}> {item.productName}</Text>
                <Text style={{paddingLeft: "40%"}}>£{item.price}0</Text>
            </TouchableOpacity>
        </View>
    )}
/>
```

Figure 64: Revised product selection

Project Evaluation

8.1 Introduction

Within this chapter, I will be evaluating the project and the report as a whole, focusing on what went well and what did not. Finally, I will be discussing how the project could be improved in the future.

8.2 Literature Review

This chapter overall went well but I encountered multiple challenges during its conception as there was not a lot of research into the topic of dark stores. Which forced me to pivot the focus more on how online grocery stores implement sustainability and how they influence consumer shopping behaviour to understand how I could implement an application that encourages customers to shop sustainably. Which originally I thought was not ideal, but overtime provided me with a deeper understanding of how a dark grocery delivery service would fair in practice. As from the research into online grocery stores, I was able to develop an application that built upon the flaws of standard online grocery shopping. However, regarding delivery optimisation the research was vast and it was rather challenging to extract relevant material as the research was mainly theoretical than applied.

This chapter could have been improved by researching into UI and UX for mobile applications as well as the most appropriate cloud based database solution for mobile app development. As I feel that by researching into these topics I could have enhanced my understanding of how to both effectively build an engaging mobile application and build a real-time database, as this project was my first experience in building both.

8.3 Requirements

Altogether, I think that my project's requirements are quite well devised as they are well defined and specific. However, I do feel that the non-functional requirements could have been formatted better as when I was conceptualising them; I struggled to capture suitable requirements that could be testable. As I found that most of the non functional requirements I was identifying were quite niche and repetitive. For instance, NFR-7 and NFR-13 are essentially the same requirement, but both require different outcomes from the system. However, despite the challenges posed by the non-functional requirements, I found the functionals to be straightforward to identify as they represent the requirements specified by the application's design.

This chapter could have been improved by identifying more of the underlying components of the app's design and the application as a whole. For instance, I could have added more functionals focusing on the order side of operations.

8.4 Methodology

For this project I took a V-model approach of development, which allowed me to segment my workload into individual screens rather than multiple screens at once. By doing so, I was able

to spot errors more frequently and find generalised solutions to them. Which overtime taught me how to prioritise certain issues over others. However, due to the V-model following the waterfall approach of only progressing once the phase has been completed and tested, I felt quite restricted by not being able to make alterations. Which meant that I had to be certain about the project's requirements specification and its design. Despite my previous remarks, I currently feel that the V-model allowed me to be more experimental with my designing and taught me to be more confident with my work. However, the future improvement I would make would be to choose a more iterative methodology such as the incremental or iterative methodologies as they are less restrictive.

8.5 Design

Due to my limited experience with both designing an app and developing script in react native, the project design was somewhat challenging. In particular, defining the application's architecture and identifying how users will interact with it. However, after redesigning the application's architecture multiple times, I was able to capture the system as a whole and identify the target user's functional requirements. Then I was able to design the app's user interface, which as stated in the design chapter took multiple iterations to complete. However, after I had identified how the app would operate, the overall experience of designing the application went quite well.

The app's design could be improved by receiving continuous feedback rather than every so often. Furthermore, in the future, I would like to add some quality of life features such as

- Grocery recommendations based on the customer's previous orders
- Displaying the customers usual grocery order
- A seasonal grocery selection
- A click and collect system if customers would prefer collection rather than delivery

8.6 Implementation

The implementation phase did not go well as there were multiple issues during development, that overtime were solved by proved to be a hindrance to the project's progress. However, despite how the development process progressed, the overall project can be considered a success as the final product did fulfil all the project requirements, aims, and objectives.

During development, I kept in line with the designs I made, even though there were multiple occasions where I wanted to experiment with new approaches I had discovered to features I had already implemented. However, by doing so, I was able to prioritise the core aspects of application rather than experimenting with new tools and features. Despite saying this, there were multiple occurrences where I was unable to progress due to my limited experience with React Native and JavaScript where I needed to deviate from the original plan. For instance, I had multiple difficulties surrounding React state management, only to find out that there was a standardised framework called Redux that solved the specific issue I was having.

Additionally, due to the project's progress being frequently halted, I had to remove certain quality of life features such as real time delivery tracking and the search bar. Which in the future, I would like to implement along with the features outlined above in the design

reflections. I would also like to implement more sign in/sign up options for customers to speed up the authentication process.

8.7 Testing

Testing was undergone throughout the project and progressed rather smoothly as the structure of the V-model methodology allowed me to refine the semantic behaviour of the application before it was even developed. Which meant that there were less problems with testing once requirements had been developed. However, there were some exceptions that were easily resolved as 4 out of the 24 tests failed. Which suggested to me that my previous planning was not enough, and suggests that in the future, more rigorous testing should be completed before the implementation phase is started.

However, due to the time constraints of this project, I was not able to test every fine detail of the application. Which in the future must be undertaken if this application was to be used by actual customers.

8.8 Limitations

Due to the time constraints of the project, there are a multitude of limitations that held the project back. With the first being the fact that this project had to be completed solo rather than in a group, where tasks could have been divided across the development team. This also meant that I missed having someone to bounce ideas off during the planning phase. Despite not having this, I did have help from my supervisor to refine my ideas.

The second limitation would be my inexperience with coding in React Native. I would say that not having any experience with it did prolong the development time of my application, even though I did take multiple courses to learn its fundamental syntax. However, overtime my prowess with the language did develop, allowing me to meet all the functional and non-functional requirements of this project. Finally, I would say that the biggest limitation of this project would be the fact that it had to be completed in tandem with other module assignments. Which did limit the amount of time I had to focus on the development of the project.

8.9 Feedback from my supervisor and second marker

During my first few meetings with my supervisor Haixia, we discussed how I would approach the idea of sustainability in this project, and we identified that the grocery service should only sell sustainably sourced products. At this point, Haixia introduced me to a multitude of sustainable online stores to gather products from. Additionally, during these sessions, Haixia provided me feedback on my chosen methodology, along with feedback on my requirements. From this I decided against using the agile methodology, in favour of the V-model.

During the later meetings, we discussed how I could solve multiple development issues that had encountered. One issue in particular was the address checking function, which we solved by researching the redux framework and alternative geocoding packages.

During the project in progress day meeting with my second marker, Frazer Barnes, we discussed ideal delivery methods and using the google maps platform suite to provide customers with both an estimated delivery time and real time delivery tracking. However, due to the time constraints, I was only able to complete the estimated delivery time portion. Finally, overall the feedback I received was extremely beneficial to the completion of my project.

Conclusion

To conclude, this project has been successful as I have developed an effective solution to encouraging people to live and eat more sustainably to help reduce the carbon emissions and non-recyclable waste that is produced by the conventional means of online grocery shopping. Furthermore, the application I have developed has met all the aims and objectives that I outlined at the beginning of this report. Additionally, relevant research has been conducted to support my solution's argument. This research has also provided me with relevant project requirements that were used to test the application's functionalities. Overall, this project has proven that dark stores could be an effective sustainable alternative to online grocery shopping.

References / Bibliography

Ai, T. J., and Kachitvichyanukul, V. (2009) A particle swarm optimization for the vehicle routing problem with simultaneous pickup and delivery. *Computers & Operations Research* [online]. 36, pp.1693-1702 [Accessed 02 February 2022]

Bauerová, R. (2021). Online grocery shopping is a privilege of millennial customers. Still truth in COVID-19 pandemic. *Acta Academica Karviniensia* [online]. *21*, pp.15-28. [Accessed 12 January 2022]

Blázquez, A. (2021) *Statista*. Available from: https://www.statista.com/topics/1983/supermarkets-in-the-united-kingdom-uk/#dossierKeyfigures [Accessed 25 October 2021].

Bortolini, M., Faccio, M., Ferrari, E., Gamberi, M., and Pilati, F. (2016) Fresh food sustainable distribution: cost, delivery time and carbon footprint three-objective optimization. *Journal of Food Engineering* [online]. 174, pp.56-67 [Accessed 27 January 2022]

Brand, C., Schwanen, T., and Anablem, J (2020) 'Online Omnivores' or 'Willing but struggling' Identifying online grocery shopping behavior segments using attitude theory. *Journal of Retailing and Consumer Services* [online]. 57, pp.102195. [Accessed 12 January 2022]

Buchanan, A. (2021) *The best online grocery shopping sites for your Christmas shopping slot – and beyond.* Available from: https://www.telegraph.co.uk/food-and-drink/features/best-online-grocery-shopping-sites-one-delivers-goods/ [Accessed 11 November 2021]

Cambridge Dictionary (n.d.) *Sustainability | meaning in the Cambridge English Dictionary*. Available from: https://dictionary.cambridge.org/dictionary/english/sustainability [Accessed 30 November 2021]

Despa, M. L., (2014) Comparative study on software development methodologies. *Database Systems Journal* [online]. *5*(3), pp.37-56. [Accessed 14 April 2022]

Durmuş, M., S., Üstoğlu, I., Tsarev, R., Y., and Börcsök, J., (2018) Enhanced V-Model. *Informatica* [Online]. 42, pp.577-585 [Accessed 02 November 2021]

Eriksson, E., Norrman, A., and Kembro, J. (2019) Contextual adaptation of omni-channel grocery retailers' online fulfilment centres. *International Journal of Retail & Distribution Management* [online]. 47(12), pp.1232-1250. [Accessed 22 October 2021]

Feedback (2018) *The Food Waste Scorecard: An assessment of supermarket action to address food waste* [online]. London: Feedback. Available from: https://feedbackglobal.org/wp-content/uploads/2018/06/Supermarket-scorecard_136_fv-1.pdf [Accessed 01 December 2021]

Ferreira, F., (2021) *React Native vs Native for Mobile App Development.* Available from: https://www.scalablepath.com/react-native/react-native-vs-native [Accessed 29 February 2022]

Fikar, C., and Braekers, K. (2022) Bi-objective optimization of e-grocery deliveries considering food quality losses. *Computers & Industrial Engineering* [online]. 163, pp.107848 [Accessed 27 January 2022]

Filimonau, V., and Gherbin, A. (2017) An exploratory study of food waste management practices in the UK grocery retail sector. *Journal of Cleaner Production* [online]. 167, pp.1184-1194 [Accessed 30 November 2021]

Firebase (2022a) *Firebase*. Available from: https://firebase.google.com/ [Accessed 29 February 2022]

Firebase (2022b) *Firebase Pricing*. Available from: https://firebase.google.com/pricing [Accessed 29 February 2022]

Firebase (2022c) *Privacy and Security in Firebase*. Available from https://firebase.google.com/support/privacy. [Accessed 29 February 2022]

Getir (2021) *Getir – pioneers of the sustainable and superfast grocery sector, to invest £100 million in the UK*. Available from: https://getir.uk/press-releases/getir-covers-3-4-million-miles-one-bikes-and-e-mopeds/ [Accessed 02 December 2021]

Google Maps Platform (2022) *The benefits of Scalable Mapping Tools - Google Maps Platform*. Available from: https://mapsplatform.google.com/why-google/ [Accessed 29 February 2022]

Google Developers (2022) *Google Maps Platform - Google Developers*. Available from: https://developers.google.com/maps [Accessed 29 February 2022]

Gorillas Technologies GmbH (2021) *Gorillas: Grocery Delivery* (Version 15.22) [Mobile App]. Available from: Google Play and App Store [Accessed 18 November 2021]

Gorillas (n.d.) *Manifesto* [online]. Available from: https://gorillas.io/en/manifesto [Accessed 02 December 2021]

Grove, G., (2016) *Principles of mobile app design: Introduction*. Available from https://www.thinkwithgoogle.com/intl/en-gb/marketing-strategies/app-and-mobile/principles-of-mobile-app-design-introduction/ [Accessed 20 February 2022]

Jiang, L., Chang, H., Zhao, S., Dong, J., and Lu, W. (2019) A Travelling Salesman Problem With Carbon Emission Reduction in the Last Mile Delivery. *IEEE Access* [online]. 7, pp.61620-61627 [Accessed 02 February 2022]

Koivupuro, H., Hartikainen, H., Silvennoinen, K., Katajajuuri, J., Heikintalo, N., Reinikainen, A., and Jalkanen, L. (2012) Influence of socio-demographical, behavioural and attitudinal factors on the amount of avoidable food waste generated in Finnish households. *International Journal of Consumer Studies* [online]. 36, pp.183-191. [Accessed 30 November 2021]

Kumar, M., and Rashid, E., (2018) An Efficient Software Development Life cycle Model for Developing Software Project. *I.J. Education and Management Engineering* [online]. 6, pp.59-68 [Accessed 02 November 2021]

Lucidchart. (n.d.) *UML Activity Diagram Tutorial.* Available from https://www.lucidchart.com/pages/uml-activity-diagram [Accessed 04 February 2022]

Lucidchart. (n.d.) *UML Use Case Diagram Tutorial*. Available from https://www.lucidchart.com/pages/uml-use-case-diagram [Accessed 04 February 2022]

Mathur, S., and Malik, S., (2010) Advancements in the V-Model. *International Journal of Computer Applications* [online]. 1(12), pp.0975.8887 [Accessed 02 November 2021]

MongoDB (2022) *Comparing Firebase vs MongoDB*. Available from: https://www.mongodb.com/firebase-vs-mongodb#:~:text=MongoDB [Accessed 29 February 2022]

Pantano, E., Pizzi, G., Scarpi, D., and Dennis, C. (2020) Competing during a pandemic? Retailers' ups and downs during the COVID19 outbreak. *Journal of Business Research* [online]. 116, pp.209-213. [Accessed 12 January 2022]

React Native (2022) *React Native - Learn once, write everywhere*. Available from: https://reactnative.dev/ [Accessed 29 February 2022]

Reef (2021) *What Is a Dark Store?*. Available from: https://reeftechnology.com/retail/what-is-a-dark-store [Accessed 08 November 2021]

Rehkopf, M. (n.d.) *User Stories with examples and a template*. Available from https://www.atlassian.com/agile/project-management/user-stories#:~:text=Summary%3A [Accessed 04 March 2022]

Suma, S. and Alqurashi, F. (2019) A comparison study of NoSQL document-oriented database system. *International Journal of Applied Mathematical Research* [online]. 8(1), pp.27-31 [Accessed 29 February 2022]

Tesco (2022) *Book a slot - Tesco Groceries*. Available from: https://www.tesco.com/groceries/en-GB/slots/delivery [Accessed 12 January 2022]

Tighe, D. (2021) *Statista*. Available from: https://www.statista.com/statistics/448040/influence-of-covid-19-on-online-grocery-shopping-uk/ [Accessed 25 October 2021]

Tugberk Ariker, C. (2021) Do consumers punish retailers with poor working conditions during COVID-19 crisis? An experimental study of Q-commerce grocery retailers. *Journal of Management, Marketing and Logistics* [online]. 8(3), pp.140-153 [Accessed 11 October 2021]

Tu, W., Zhao, T., Zhou, B., Jiang, J., Xia, J., and Li, Q. (2020) OCD: Online Crowdsourced Delivery for On-Demand Food. *IEEE Internet Of Things Journal* [online]. 7(8), pp.6842-6854 [Accessed 29 January 2022]

Appendix A: First Appendix

GitHub Link to code

https://github.com/JosiahM42/Dark-Store-App