

National College of Ireland

BSHC

Software Development

2020/2021

Paula Olariu

x17370023

x17370023@student.ncirl.ie

Shopping Wishlist

Technical Report

Table of Contents

Executive Summary	5
1.0 Introduction	6
1.1. Background	6
1.2. Aims.....	7
1.3. Technology.....	8
1.4. Structure	10
2.0 System.....	11
2.1. Requirements.....	11
2.1.1. Functional Requirements	11
2.1.1.1. Use Case Diagram	11
2.1.1.2. Requirement 1 - User Registration	12
2.1.1.3. Requirement 2 - User Login	14
2.1.1.4. Requirement 3 - Download Extension	16
2.1.1.5. Requirement 4 - Extension Functionality.....	18
2.1.1.6. Requirement 5 - Access Wish Boards	20
2.1.1.7. Requirement 6 - Manage Wish Boards	22
2.1.1.8. Requirement 7 - Manage Items	24
2.1.1.9. Requirement 8 - View Items	26
2.1.2. Data Requirements	28
2.1.3. User Requirements	28
2.1.4. Environmental Requirements	28
2.1.5. Usability Requirements.....	29
2.2. Design & Architecture.....	30
2.3. Implementation	31
2.4. Graphical User Interface (GUI).....	48
2.5. Testing.....	53
2.6. Evaluation	55
3.0 Conclusions	58
4.0 Further Development or Research	58
5.0 References	59
6.0 Appendices.....	62
6.1. Project Proposal.....	62
6.1.1. Objectives.....	62
6.1.2. Background	62
6.1.3. Technical Approach.....	63

6.1.4.	Project Plan	64
6.1.5.	Technical Details	65
6.1.6.	Evaluation	65
6.2.	Reflective Journals	66
6.2.1.	October Reflective Journal.....	66
6.2.2.	November Reflective Journal.....	67
6.2.3.	December Reflective Journal	67
6.2.4.	January Reflective Journal	68
6.2.5.	February Reflective Journal	68
6.2.6.	March Reflective Journal	69
6.2.7.	April Reflective Journal	70

Table of Figures

Figure 1 System Use Case Diagram.....	11
Figure 2 User Registration Use Case	12
Figure 3 User Login Use Case	14
Figure 4 Download Extension Use Case	16
Figure 5 Extension Functionality Use Case	18
Figure 6 Access Wish Boards Use Case	20
Figure 7 Manage Wish Boards Use Case.....	22
Figure 8 Manage Items Use Case	24
Figure 9 View Items Use Case	26
Figure 10 Architecture Diagram.....	30
Figure 11 Firebase Connection	31
Figure 12 Initialising Authentication Actions Part 1.....	32
Figure 13 Initialising Authentication Actions Part 2.....	33
Figure 14 Creating Login Form Part 1	34
Figure 15 Creating Login Form Part 2	35
Figure 16 Creating Signup Form Part 1	36
Figure 17 Creating Signup Form Part 2	37
Figure 18 Social Authentication Provider Checker	38
Figure 19 Social Authentication Interface.....	38
Figure 20 Reset Account Password Part 1	39
Figure 21 Reset Account Password Part 2	39
Figure 22 Update User Account Details	40
Figure 23 Restricted Content Visibility.....	41
Figure 24 Sidebar Navigation	42
Figure 25 Sidebar Navigation Components	42
Figure 26 Create New Wish Board Part 1	43
Figure 27 Create New Wish Board Part 2	43
Figure 28 Delete Wish Board Part 1.....	44
Figure 29 Delete Wish Board Part 2.....	44
Figure 30 Display Items Part 1	44
Figure 31 Display Items Part 2	45
Figure 32 Search Saved Items	45
Figure 33 Manifest Setup.....	46
Figure 34 Retrieving Wish Boards from Firestore.....	46
Figure 35 Getting the Current Tab URL.....	47
Figure 36 Saving Item Details.....	47
Figure 37 Unregistered User Landing Page.....	48
Figure 38 Signup and Login Forms	49
Figure 39 Home Page for Registered Users	49
Figure 40 Home Page with Released Sidebar Navigation	50
Figure 41 Wish Board Page	50
Figure 42 Create New Wish Board.....	51
Figure 43 Save Item with Chrome Extension	51
Figure 44 View Item on Website.....	52
Figure 45 Performance Load Time Test Results.....	55
Figure 46 Performance Response Time Test Results.....	55

Figure 47 Language Code Summary Table	56
Figure 48 Directory Code Summary Table	56
Figure 49 Puppeteer Generated Screenshot of Curry's Webpage.....	57

Executive Summary

This document presents the goals achieved by the developed project. It outlines the motivation for the project and the variety of features offered. As part of this project a website and a browser extension were developed using mainly JavaScript and ReactJS. The Shopping Wishlist browser extension was developed and optimised to solely operate on the Google Chrome browser. To create the project, packages such as the react-router-dom, react-scroll, react-icons and more were used to improve the functionality and request response times.

The application allows users to create accounts or register with one of their existing social accounts such as Facebook or Google. Registered users can use the website to create or delete wish boards from the account. Any additions or alterations performed on the website are instantly reflected on the Firebase Cloud Firestore database. Users can also use the website to view items saved to specific wish boards as well as remove any of the items at any point. The browser extension is used for gathering the information required for displaying the item details in the specific wish board. The main purpose of this project is to improve the online shopping experience for users, especially for the users that do not shop online frequently or are not extremely familiar with the latest technologies.

This report further describes the drive behind this project and provides insights into the development process as well as the various technologies used. The necessary system requirements are identified and explained below. Diagrams and descriptions presenting the relations between the project components, actions, and hooks are inserted in the following sections.

1.0 Introduction

1.1. Background

The goal of the application is to make online shopping easier and less stressful for users. The idea for this project emerged once the lockdown initially began, with non-essential shops being closed and travel restrictions put in place everyone became more dependent on online shopping. A large variety of stores can be found on the web, with very large numbers of products available for users to choose from, making the right decision when shopping online becomes increasingly difficult. A web tool that can categorise items into specific sections can be a very useful tool for many people who cannot afford certain items or are not necessarily sure if the item is essential. Due to the changes that occurred in the past year, many people lack the funds required for purchasing things that are not considered a necessity.

The web application is aiming to transform online shopping into a more enjoyable experience while enabling the user to make educated choices when purchasing items. Such an app can be a useful tool for people especially currently when people are more cautious when placing orders online. The website could save users a lot of time, money and reduce buyer's remorse resulting in more informed and planned purchases. The application should be easy to navigate to get people from different age groups and backgrounds to be interested in using the extension as people with less technical knowledge tend to be scared by such web resources.

1.2. Aims

The website and browser extension that is to be developed and implemented for the purpose of this project will strive to achieve the following goals:

- ❖ To access all of the other features that the web application has to offer, the user will first create an account on the website. The options of logging in using social accounts such as Google and Facebook is available for users to make the process faster and more efficient.
- ❖ Using the extension icon located on the Chrome browser, the user is able to save items to the account. The process of adding items is easy to follow to allow users from various age groups to readily access the system. When the extension icon is clicked, it displays a clean and simple design is shows to the user.
- ❖ The user can select any of the three discount options present on the plugin pop out form before saving the item.
- ❖ The browser extension lets the user enter a name for the item that is to be saved.
- ❖ The user can choose the wish board on which the item should be saved using an easily accessible dropdown list. If the user does not select a wish board from the list.
- ❖ The web application side of the project gives users access to saved items, wish boards and other relevant information.
- ❖ The website is used by users to create new wish boards. Users are free to create an unlimited number of wish boards.
- ❖ Wish boards can be managed by the user using the web application, once items saved are of no further need to the user, the user can remove them. Similarly, the user can remove wish boards at any chosen moment.
- ❖ Users can click on the image icon displayed inside of the card view of each item to be redirected to the item's store page.
- ❖ The website displays card views with direct links to online stores that might be of interest to the user.
- ❖ Users can add an unlimited number of items to their shopping wish boards.
- ❖ The items saved by the user can filtered using a search bar. The user can enter the name of any wish board to only view the items present in that wish board.

1.3. Technology

The editor used for the development of this application is Visual Studio Code, using the code editor, the project is developed, evaluated, and tested. The main coding languages used for developing this project is ReactJS and JavaScript. [1] [2] [3]

ReactJS was used for creating the project structure and initiated both the website and browser extension. To create the project the *"npx create-react-app shopping-wishlist"* command was executed. This initialised the standard build, node_modules, public and src packages along with some other base scripts.

JavaScript, HTML, CSS and Json were mainly used for the development of the browser extension. These languages were best suited as they suited the Manifest action requirements best. The Browser extension was initially developed using Manifest version 2; however, version 3 became available, and the extension was upgraded to the latest version to maintain availability as version 2 will be deprecated this year. [4] [5] [6]

In addition to JavaScript, the jQuery library was also used for event handling and DOM manipulation. The library was integrated for creating various functionalities required for the browser extension. [21]

The react-router-dom package was used for handling the website's navigation. The package provides dynamic routing for rendering requests and suited the needs of the project well. The react-scroll package was also integrated into the project. This package improves the user interface of the website as it provides animated vertical scrolling. React-scroll provides smooth scrolling which operates by showing the user an animated scroll effect when navigating to different parts on the same page. [7] [8]

A variety of packages were used for designing the interface of the project. The two packages that were principally used are the react-bootstrap and styled-components packages. React-bootstrap was largely used for creating card views and for passing predefined bootstrap component styling to elements. The styled-components package was extensively used for creating custom components for displaying information on the website. The react-icons package was used for importing opensource icons from the react library, and the semantic-ui package was used for improving the website's user interface. [12] [13] [14] [15]

Redux was used for testing the project, ensuring that the website and extension behave consistently, and debug and track the application's state changes. The Loadster tool was also used to test the performance of the system. The tool enable a variety of request, and stress tests to be performed on the system. [17] [28]

The yarn environment manager was used for installing, updating, and maintaining the version of the dependencies needed for developing the project. The Babel JavaScript compiler was needed during the development stage for monitoring the ReactJS JSX syntax. Along with the babel-sublime package it helped debug syntax errors and compiled code. [9] [10] [11]

The Google Firebase is the database chosen for storing all of the project's data. Using the Cloud Firestore real-time NoSQL database, wish boards and items collections and database relations were created for retrieving data and displaying it to the user on the website and browser extension. [18]

The Puppeteer Node library was used for carrying out tests on the Chrome extension. It is a powerful API that creates a headless browser environment in the command line and enables the Chromium web features. The tool was used for generating screenshots of the webpages retrieved by the browser extension. [22]

GitHub and Git were used for storing the project in a secure location. These were useful for version control and for ensuring that a stable version of the project was available on the main branch. [20]

1.4. Structure

This section describes the flow of the content presented in this report and presents the objectives, requirements, implementation, and overall outcome of the project. A list of the report sections along with a brief overview for each section was inserted below.

- ❖ **Section 1 - Introduction:** This section of this report provides a general overview of the project objectives, motivations and describes the use of specific technologies over other available options
- ❖ **Section 2.1 – Requirements:** This section provides a detailed breakdown of the project requirement specifications, including both the functional requirements and non-functional requirement.
- ❖ **Section 2.2 – Design and Architecture:** This section of the report provides an explanation of the system and a descriptive architecture diagram containing the architecture and structure of the system.
- ❖ **Section 2.3 – Implementation:** This section provides a detailed explanation of the development stage for the project. Several figures were inserted to provide visualization into the code structure and the practises used for developing the key project functionalities.
- ❖ **Section 2.4 – GUI:** This section present the user interface of the project and explains the design decisions made.
- ❖ **Section 2.5 – Testing:** In this section an account of the various testing performed for the project is given along with a summary of the test results.
- ❖ **Section 2.6 – Evaluation:** This section presents the evaluation tools, methods, and results gathered as part of this project.
- ❖ **Section 3 – Conclusion:** This section describes the outcome of the project and a summary of the overall obstacles and achievements of this project.
- ❖ **Section 4 – Further development and Research:** This section describes the steps that could be taken in the future to enhance and expand the functionality of the system.
- ❖ **Section 5 – References:** In this section a list outlining the learning resources used for this project can be found.
- ❖ **Section 6 – Appendices:** This section contains the project proposal, reflective journals and other materials used.

2.0 System

2.1. Requirements

2.1.1. Functional Requirements

2.1.1.1. Use Case Diagram

Figure 1 System Use Case Diagram

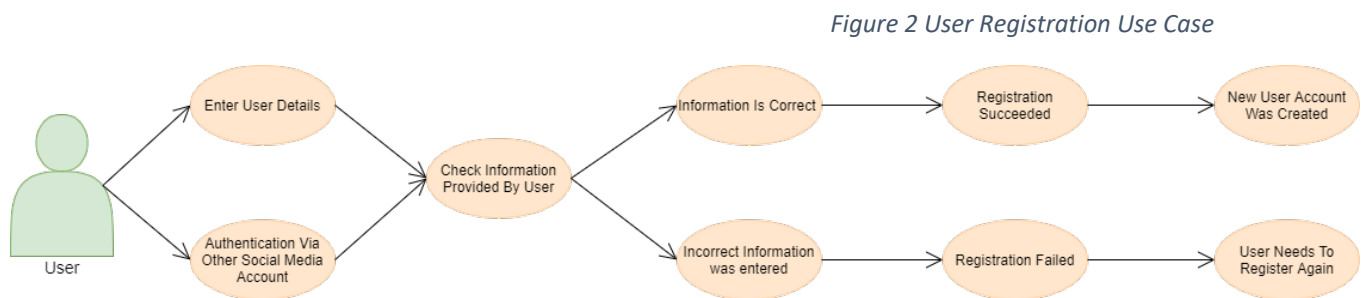


2.1.1.2. Requirement 1 - User Registration

a. Description & Priority

When the user registers an account, the option of registering using other social accounts such as Google or Facebook is provided to make the registration process easier for the user.

b. Use Case



i. Scope

The scope of this use case is to set up accounts for users to enable the use of the system

ii. Description

This use case describes to setup of the initial user account

iii. Flow Description

Precondition

The system is waiting to be accessed by the user

Activation

The use case starts when the user visits the website

Main flow

1. The system identifies the arrival of the user
2. The user is presented with the option of registering or logging in
3. The user selects the register option
4. The system displays the registration form
5. The user fills in the required details on the form
6. The system saves the information added by the user on the database

Alternate flow

A1

1. The user chooses to register using a different social account
2. The user enters the required information to register that account with the system

3. Registration is successful

Exceptional flow

1. The user enters incorrect information to the registration form
2. The user clicks the register button
3. Registration fails

iv. Termination

The system stores the information entered by the user on the database.

v. Post Condition

User can login using the details entered during registration.

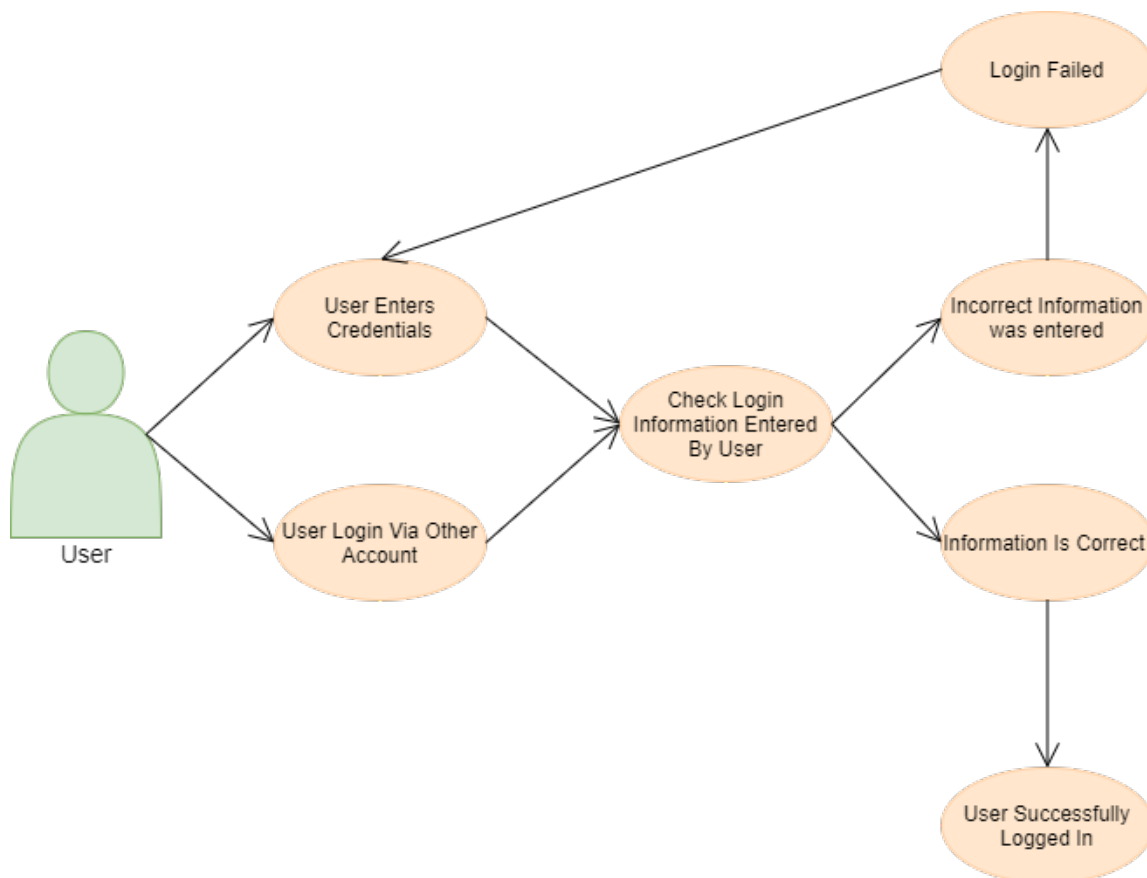
2.1.1.3. Requirement 2 - User Login

a. Description & Priority

The login use case is required as without a login the user would not be able to access other features that the system has to offer.

b. Use Case

Figure 3 User Login Use Case



i. Scope

The scope of this use case is for the user to log in the application.

ii. Description

This use case describes how the user logs into their account on the web application.

iii. Flow Description

Precondition

The user is required to have a registered account

Activation

The use case starts when the user visits the website or after registering an account.

Main flow

1. The system shows the login page
2. The user enters the correct username
3. The user enters the correct password
4. The user clicks on the login button
5. The login has been successful

Alternate flow

A1

1. The user chooses to login using a different social account
2. The user enters the required information for the login authentication

Exceptional flow

1. The user enters incorrect information
2. The user clicks on the login button
3. The login has been unsuccessful

iv. Termination

The system identifies the account information entered and the user is brought to the home page of the website.

v. Post Condition

The user can use all the functionality that the system has to offer.

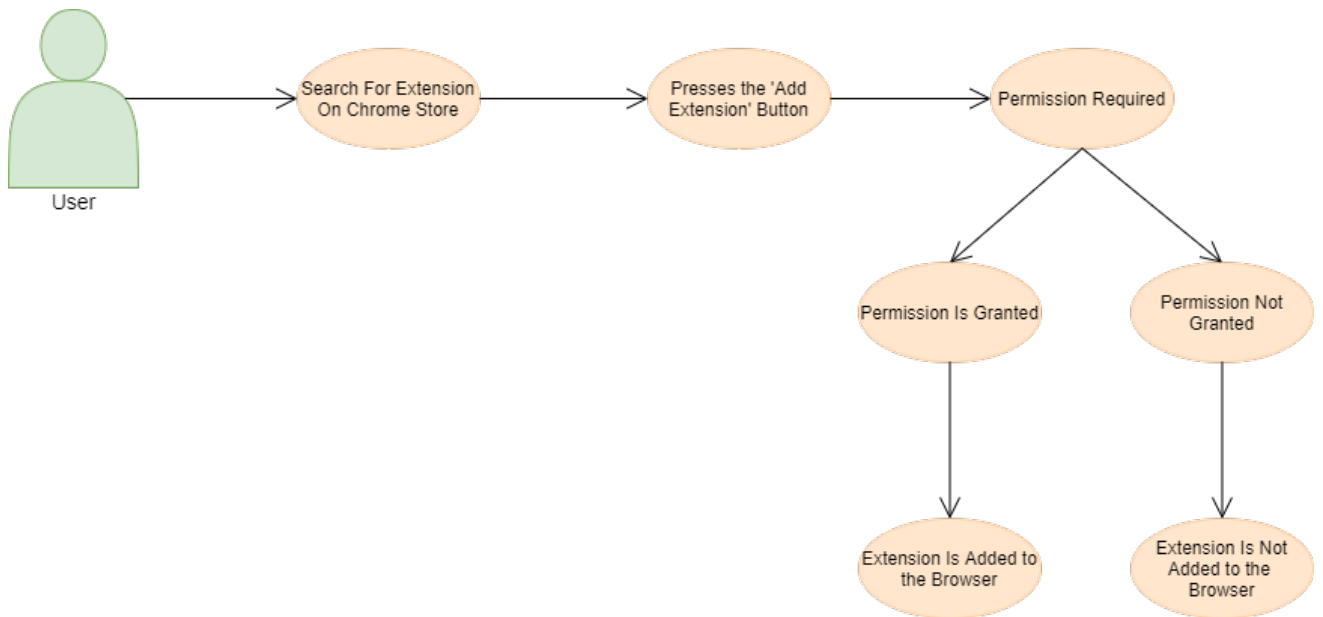
2.1.1.4. Requirement 3 - Download Extension

a. Description & Priority

The download extension use case is required as without adding the browser extension on the Chrome engine the user cannot use all of the features available.

b. Use Case

Figure 4 Download Extension Use Case



i. Scope

The scope of this use case is to enable the extension the user's browser.

ii. Description

This use case describes how the user can download and enable the Chrome extension on their device.

iii. Flow Description

Precondition

The user needs to be logged into their account.

Activation

The use case starts after user logs in.

Main flow

1. The user searches for the web application on the Google Chrome store
2. The user presses on the add extension button in the store
3. The user grants the extension permission to operate on the browser

4. The extension button is added to the user's library

Alternate flow

Not applicable.

Exceptional flow

1. The user searches for the web application on the Google Chrome store
2. The user presses on the add extension button in the store
3. The user does not grant permission to the extension
4. The process fails

iv. Termination

The user successfully downloads and installs the extension on the browser.

v. Post Condition

The user can use the extension button is added on the user's browser.

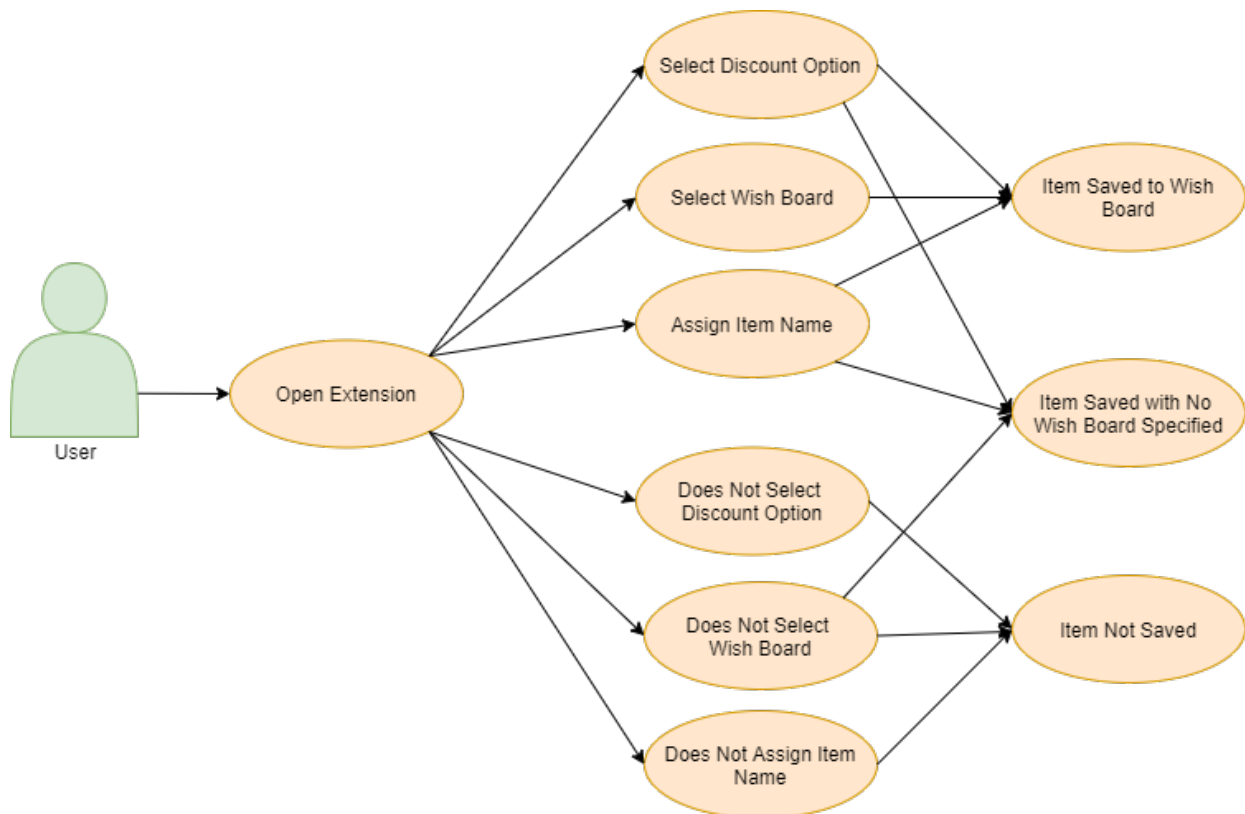
2.1.1.5. Requirement 4 - Extension Functionality

a. Description & Priority

The extension functionality use case is required as it enables the user to access the features of the browser extension.

b. Use Case

Figure 5 Extension Functionality Use Case



i. Scope

The scope of this use case is for the user to save items using the browser extension.

ii. Description

This use case describes how the user can use the extension and the functionality that the extension provides.

iii. Flow Description

Precondition

The user is required to have the browser extension installed on the device.

Activation

The use case starts when the user clicks on the extension button in the browser.

Main flow

1. The extension button is clicked
2. The user selects when to get notified about the item
3. The user selects the wish board in which the item should be saved
4. The user clicks on the save button to add the item to the wish board
5. Item is added to the specific board

Alternate flow

A1

1. The extension button is clicked
2. The user selects when to get notified about the item
3. The user does not select the wish board in which the item should be saved
4. The user clicks on the save button to add the item to the wish board
5. Item is saved

A2

1. The extension button is clicked
2. The user does not select when to get notified about the item
3. The user selects the wish board in which the item should be saved
4. The user clicks on the save button to add the item to the wish board
5. Item is saved to the selected board

Exceptional flow

1. The extension button is clicked
2. The user does not select when to get notified about the item
3. The user does not select the wish board in which the item should be saved
4. The user clicks on the save button to add the item to the wish board
5. Item does not get saved

iv. Termination

The extension saved the item, and the extension closes.

v. Post Condition

The user stores the selected item.

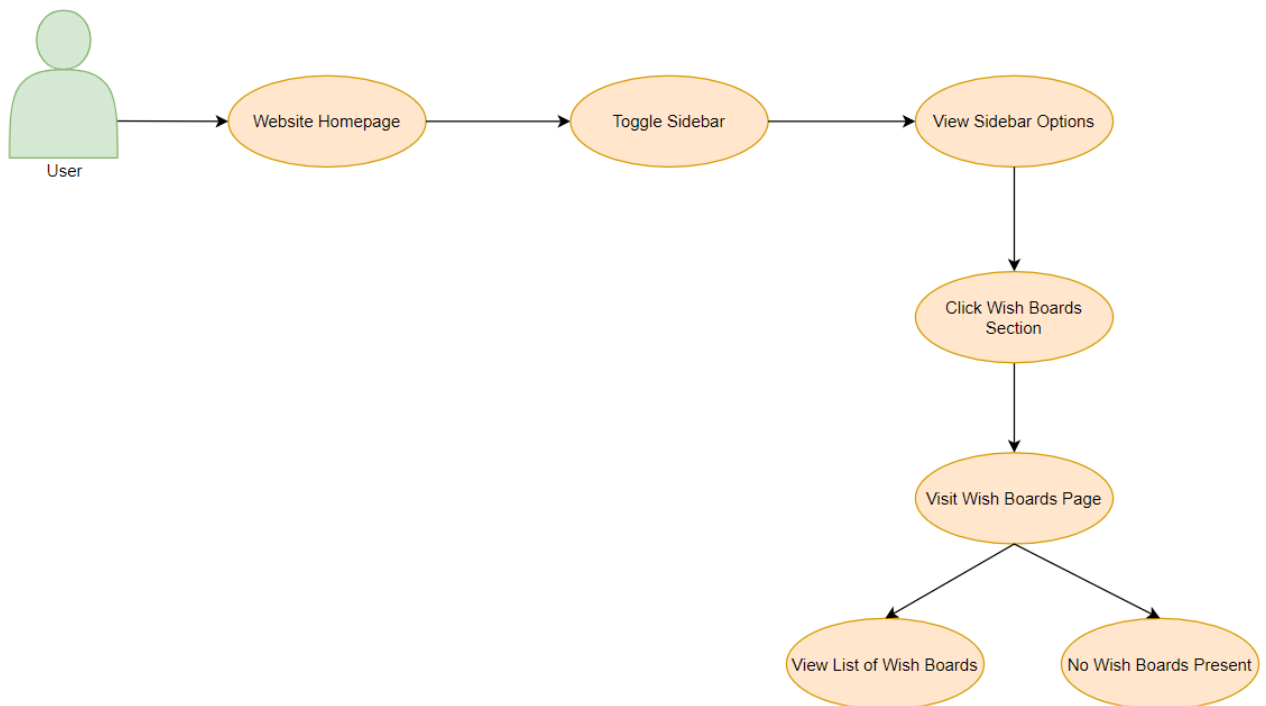
2.1.1.6. Requirement 5 - Access Wish Boards

a. Description & Priority

The Access Wish Board use case is required for displaying the existent wish boards.

b. Use Case

Figure 6 Access Wish Boards Use Case



i. Scope

The scope of this use case is for the user to access the wish boards.

ii. Description

This use case describes how the user can view the boards that have been previously created.

iii. Flow Description

Precondition

The user is expected to have an account.

Activation

The use case starts when the user visits the homepage of the website.

Main flow

1. The user accesses the homepage of the website
2. The user navigates clicks on the menu icon located on the navigation bar to toggle the sidebar

3. The 'Wish Boards' button is clicked by the user
4. The system redirects the user to the 'Wish Boards' page
5. The user find a list of previously created wish boards

Alternate flow

Not applicable.

Exceptional flow

1. The user accesses the homepage of the website
2. The user navigates clicks on the menu icon located on the navigation bar to toggle the sidebar
3. The 'Wish Boards' button is clicked by the user
4. The system redirects the user to the 'Wish Boards' page
5. The user does not have any wish boards to view

iv. Termination

The user selects the board that is to be viewed.

v. Post Condition

The user analyses the saved items in the selected board.

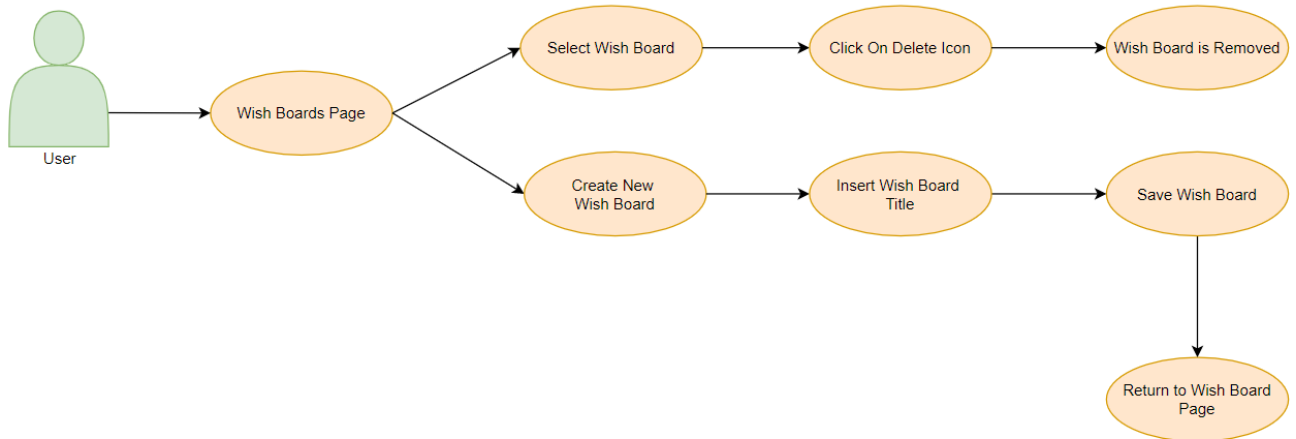
2.1.1.7. Requirement 6 - Manage Wish Boards

a. Description & Priority

The Manage Wish Boards use case is required for customising, removing, or adding wish boards to the user's account.

b. Use Case

Figure 7 Manage Wish Boards Use Case



i. Scope

The scope of this use case is for the user to control wish boards.

ii. Description

This use case describes how the user can customise, edit, or remove wish boards present in the user's account as well as add or create new boards.

iii. Flow Description

Precondition

The user is expected to have navigated to the wish board navbar.

Activation

The use case starts when the user accesses the wish board section of the website.

Main flow

1. The user clicks on the wish boards button
2. The system presents the wish board page
3. The user selects a wish board from the list
4. The user clicks on the trash can icon to delete the wish board
5. The selected wish board is removed

Alternate flow

A1

1. The user clicks on the wish boards button
2. The system presents the wish board page
3. The user click on the 'Create New Board' button
4. The system shows the form for creating a new wish board
5. The user enter a title for the new board
6. The user click on the save button
7. The user is brought back to the wish board page

Exceptional flow

Not applicable.

iv. Termination

The user concludes the process of managing the wish boards.

v. Post Condition

The user can review the changes made in the wish board menu.

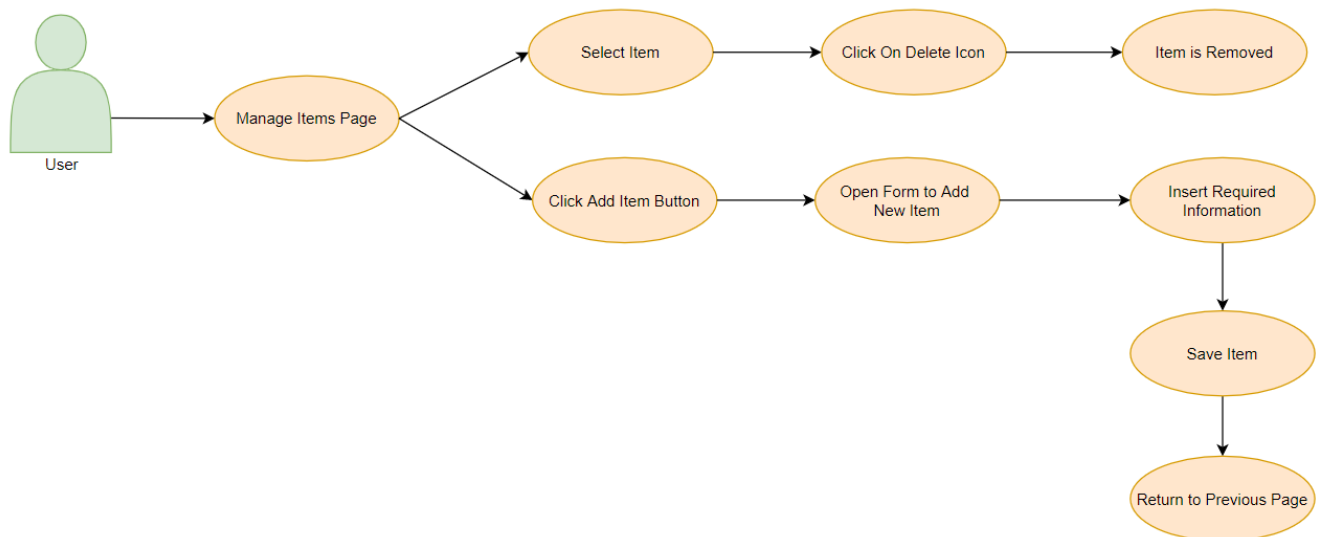
2.1.1.8. Requirement 7 - Manage Items

a. Description & Priority

The Manage Items use case is required for adding and removing items from wish boards.

b. Use Case

Figure 8 Manage Items Use Case



i. Scope

The scope of this use case is for the user to edit items saved with a wish board.

ii. Description

This use case describes how the user can add and remove items from the wish boards present in the user's account.

iii. Flow Description

Precondition

The user is expected to be logged into an account.

Activation

The use case starts when the user navigated to the 'Manage Items' page.

Main flow

1. The user presses on the 'Manage Items' button on the sidebar navigation
2. The system redirects the user to the page
3. The user clicks the 'Add Item' button

4. The system brings the user to the create new item page
5. The user fills in the required fields
6. The user saves the new item
7. The system returns the user to the previous page

Alternate flow

A1

1. The user presses on the 'Manage Items' button on the sidebar navigation
2. The system redirects the user to the page
6. The user clicks on the trash can icon to delete the item
3. The system displays all the items present in the board
4. The user selects the item to be removed
5. The selected item is removed

Exceptional flow

Not applicable.

iv. Termination

The user the user leaves the page.

v. Post Condition

The user can view the items added or check for the removed items on the items page.

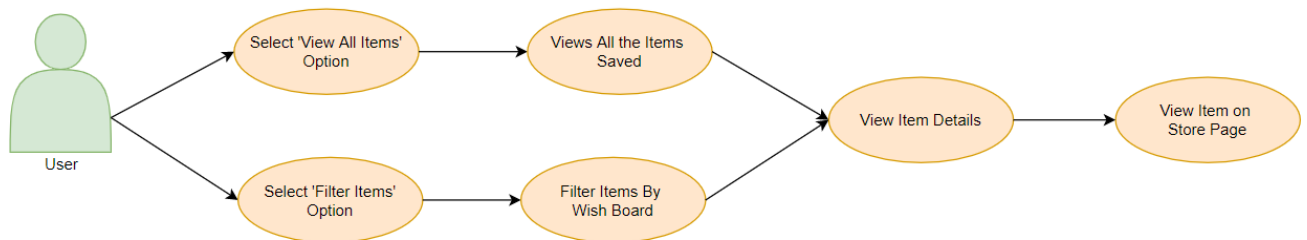
2.1.1.9. Requirement 8 - View Items

a. Description & Priority

The View Items use case is required to allow the user to manage and filter the way items are displayed.

b. Use Case

Figure 9 View Items Use Case



i. Scope

The scope of this use case is for the user to customise viewing preferences.

ii. Description

This use case describes how the user can view items within a board. The user can select to view all items or filter items by wish board.

iii. Flow Description

Precondition

The user is expected to have the sidebar open.

Activation

The use case starts when the user accesses the items section of the website.

Main flow

1. The user selects the 'View All Items' option
2. The system displays all items saved by the user
3. The user checks the item details
4. The user clicks on the image icon of an item
5. The system redirects the user to the official store item page

Alternate flow

1. The user selects the 'Filter Items' option
2. The system displays all items saved by the user
3. The user searched for a specific wish board title
4. The system shows the search results

5. The user views saved items
6. The user clicks on the image icon of an item
7. The system redirects the user to the official store item page

Exceptional flow

Not applicable.

iv. Termination

The user views the item.

v. Post Condition

The user accessed the item successfully.

2.1.2. Data Requirements

The system required the ability to transfer, save, and alter data inside the database. Some of the data requirements needed for the development of this project are:

- ❖ The browser extension has permissions to alter the database by adding or editing items located in the database.
- ❖ The changes that occur within the database as a result of the interactions carried by the browser extension are also shared with the website and displayed in the appropriate location.
- ❖ The website itself has the ability to change the data found in the database as it can insert, remove, or replace objects.
- ❖ Both the website and the browser extension share the same Firebase database which enables both features of the project can easily interact with the user data.
- ❖ The Cloud Firestore database was configured to allow both the browser extension and the website to communicate, and shares the objects located in the database with both projects.

2.1.3. User Requirements

When attempting to access the system, the user must have a stable internet connection. The user is required to have a verified account and have the Chrome browser extension installed on the device. The user is also required to sign into the registered account by providing the correct credentials.

Once the authentication stage is successfully completed, the user can access the existent wish boards as well as creating new wish boards or deleting existing ones. The user can also remove or insert new items to a wish board if chosen. The user also has the option of changing the account's password or email address through the update profile section found on the website.

2.1.4. Environmental Requirements

A major environmental factor that can influence the operational capabilities of the project is internet access. The web application requires an internet connection to easily access the system.

Another factor that can have an effect on the application is the web environment on which it is ran. The Shopping Wishlist website and browser extension are developed to solely operate on the modern Google Chrome web browser.

2.1.5. Usability Requirements

Each area of the system is required to have an easy to follow layout and a descriptive user interface. The website and Chrome extension should contain meaningful and relevant naming, to ensure that users from all backgrounds and age groups can effortlessly navigate through the system to avoid confusion and create an inclusive user environment. The unnecessary content that can be often found on websites will not be added to the project to ensure that the user does not need to follow meaningless steps to achieve the wished goal.

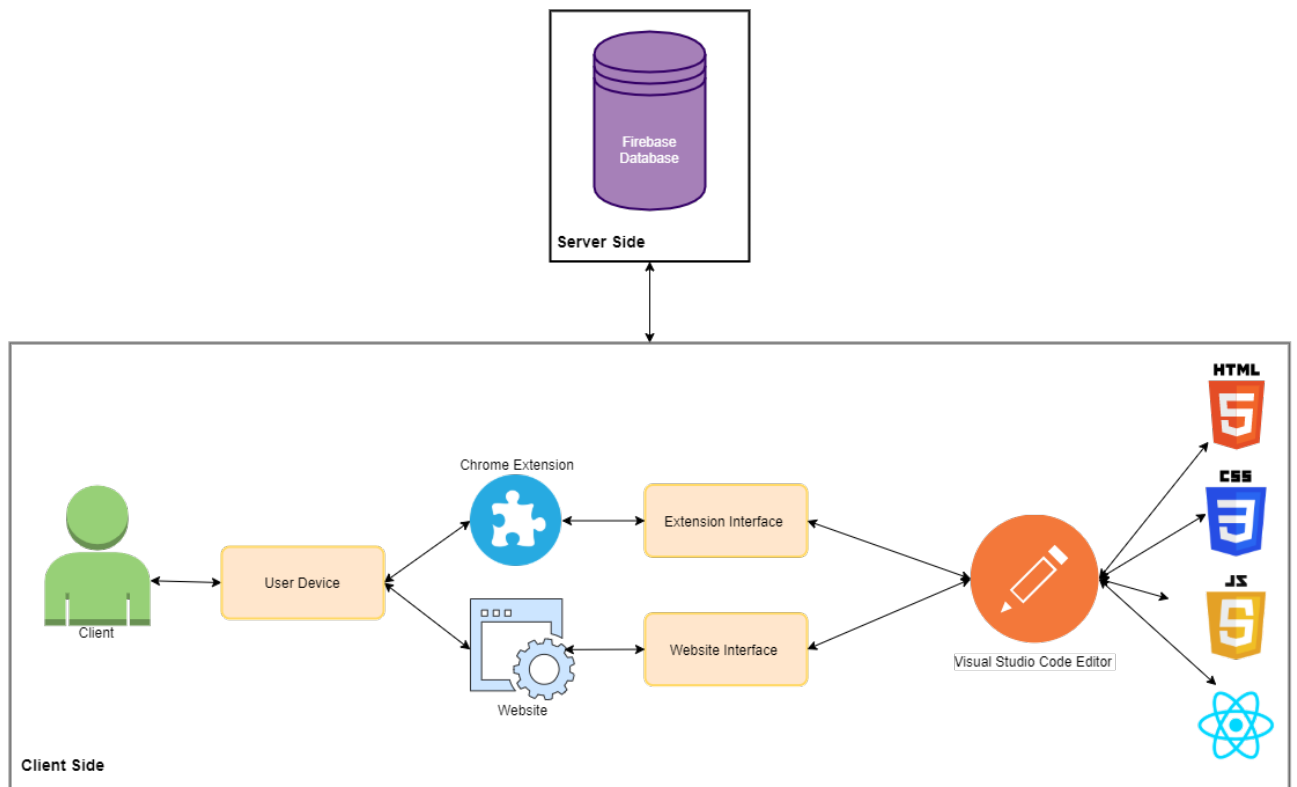
The application and browser extension should follow a consistent layout and colour story across all sections and pages to maintain consistency and eliminate redundancy. A defined set of components that can be reused throughout the project should be established to ensure that a consistent UI theme is maintained.

2.2. Design & Architecture

The design and architecture of the system consists of the client side and the server side components. The client side of the system architecture contains the client who interacts with a device. The set device then calls either the website system or the browser system, depending on the action requested by the user. In response to the client request made by the device, the system contacts the HTML, CSS, JavaScript, and the React JS components to carry out the client request.

The client side components of the system are also linked with the server side components. On the server side of the system, the database can be found. For storing the user account, user data and other required information the Google Firebase database is used. The database section of the system architecture communicates with the client side component and with Node.js when carrying out requests made by client.

Figure 10 Architecture Diagram



2.3. Implementation

The Shopping Wishlist website and browser extension are developed in JavaScript, HTML, CSS, ReactJS, and JSON. To initialize the ReactJS project, the Yarn, and Babel package managers were installed. These tools are responsible for installing other packages that are needed for the project and for keeping the versions of the installed dependencies found in the *package.json*, and *yarn.lock* scripts up to date. These package managers are also needed for building and launching the project on the localhost and on Heroku during the development stage.

The Shopping Wishlist website was developed using several React components that communicate closely with each other. These components are located inside subdirectories of the *src* folder. Subdirectories were created to group together the files that were created for a specific project feature. The other subdirectories created inside of the *src* directory contain reusable hooks, database service scripts, Redux related actions, and scripts for data mocking while testing features.

CONNECTING THE PROJECT WITH FIREBASE:

The firebase database integration was carried out by installing the *firebase* package dependency using the *yarn* package manager. The firebase project credentials were inserted into a *.env.local* script to keep the database keys private and secure during development. However, the secrets were inserted into the *firebase.js* file once the project was deployed to Heroku to enable the cloud provider access to the database. These keys were imported into the *firebase.js* script as seen in Figure 11. The firebase application was initialised in the Figure 11 code snippet, this was done to retrieve the Cloud Firestore and Authentication dependencies from Firebase. An export statement was created to make the database connection accessible to other project files.

Figure 11 Firebase Connection

```
import firebase from 'firebase/app';
import 'firebase/firestore';
import "firebase/auth";

const app = {
  apiKey: process.env.REACT_APP_FIREBASE_API_KEY,
  authDomain: process.env.REACT_APP_FIREBASE_FIREBASE_AUTH_DOMAIN,
  databaseURL: process.env.REACT_APP_FIREBASE_FIREBASE_URL,
  projectId: process.env.REACT_APP_FIREBASE_FIREBASE_PROJECT_ID,
  storageBucket: process.env.REACT_APP_FIREBASE_FIREBASE_STORAGE_BUCKET,
  messagingSenderId: process.env.REACT_APP_FIREBASE_FIREBASE_MESSAGE_SENDER_ID,
  appId: process.env.REACT_APP_FIREBASE_FIREBASE_APP_ID,
  measurementId: process.env.REACT_APP_FIREBASE_FIREBASE_MEASUREMENT_ID
};

firebase.initializeApp(app);

export default firebase;
```


AUTHENTICATION CONTEXT ACTIONS:

A context file named *AuthContext.js* was created for declaring the actions that need to occur when users are asked to register, login, logout, reset password or update profile details. Figures 12 and 13 show the functions created to allow the user to perform authentication related actions on the website. These functions were created in the same file to ensure that none of the functions are altered and eliminate potential errors. The functions in figures 12 and 13 are enabled in other react scripts through import statements. The signup function uses the firebase imported *createUserWithEmailAndPassword* function to store the details entered by the user and add the details to the database. The login function listens to the details entered by the user and checks for the details in the database using the *signInWithEmailAndPassword* firebase authentication function.

Figure 12 Initialising Authentication Actions Part 1

```
import React, { useContext, useEffect, useState } from 'react';
import firebase from '../firebase';

const AuthContext = React.createContext();

export function useAuth() {
  return useContext(AuthContext);
}

export function AuthProvider({ children }) {

  const [currentUser, setCurrentUser] = useState()
  const [loading, setLoading] = useState(true)

  function signup(email, password) {
    return firebase.auth().createUserWithEmailAndPassword(email, password);
  }

  function login(email, password){
    return firebase.auth().signInWithEmailAndPassword(email, password)
  }

  function logout() {
    return firebase.auth().signOut()
  }

  function resetPassword(email) {
    return firebase.auth().sendPasswordResetEmail(email)
  }

  function updateEmail(email) {
    return currentUser.updateEmail(email)
  }

  function updatePassword(password) {
    return currentUser.updatePassword(password)
  }
}
```

Figure 13 Initialising Authentication Actions Part 2

```
function updateEmail(email) {
  return currentUser.updateEmail(email)
}

function updatePassword(password) {
  return currentUser.updatePassword(password)
}

useEffect(() => {
  const unsubscribe = firebase.auth().onAuthStateChanged(user => {
    setCurrentUser(user)
    setLoading(false)
  })
  return unsubscribe;
}, []);

const value = {
  currentUser,
  login,
  signup,
  logout,
  resetPassword,
  updateEmail,
  updatePassword
}

return (
  <AuthContext.Provider value={value}>
    {!loading && children}
  </AuthContext.Provider>
)
```

LOGIN ACTIONS:

Figures 14 and 15 show how the login function imported through *useAuth* from the authentication context file the user login process takes place. Using the *emailRef* and *passwordRef* variables the system validates the details entered by the user during login. The Container *react-bootstrap* tag contains the login form that is displayed to the user. This file also manages errors that can occur when the user enters incorrect information, these errors are presented to the user using *react-bootstrap* alert tags. Once the user is validated, using the *useHistory* import from the *react-router-dom* package the user is redirected to the home page.

Figure 14 Creating Login Form Part 1

```
import React, { useRef, useState } from 'react';
import { Form, Button, Card, Alert, Container } from 'react-bootstrap';
import { useAuth } from '../contexts/AuthContext';
import { Link, useHistory } from 'react-router-dom';
import SocialAuth from './SocialAuth';

export default function Login(){

  const emailRef = useRef()
  const passwordRef = useRef()
  const { login } = useAuth()
  const [error, setError] = useState('')
  const [loading, setLoading] = useState(false)
  const history = useHistory()

  async function handleSubmit(e){
    e.preventDefault()

    try {
      setError('')
      setLoading(true)
      await login(emailRef.current.value, passwordRef.current.value)
      history.push('/')
    } catch {
      setError('Failed to log in')
    }
    setLoading(false)
  }

  return(
    <Container className="d-flex align-items-center justify-content-center" style={{ minHeight: "100vh" }}>
      <div className="w-100" style={{maxWidth: '400px'}}>
        <Card>
          <Card.Body>
            <h2 className="text-center mb-4">
              Log In
            </h2>

```

Figure 15 Creating Login Form Part 2

```
{error && <Alert variant="danger">{error}</Alert>}
<Form onSubmit={handleSubmit}>
  <Form.Group id="email">
    <Form.Label>Email</Form.Label>
    <Form.Control type="email" ref={emailRef} required />
  </Form.Group>
  <Form.Group id="password">
    <Form.Label>Password</Form.Label>
    <Form.Control type="password" ref={passwordRef} required />
  </Form.Group>
  <Button
    disabled={loading}
    className="w-100"
    style={{ background: '#588cfc' }}
    variant="outline-light"
    type="Submit"
  >
    Log In
  </Button>
  <div className="w-100 text-center mt-2 mb-2">Or</div>
  <SocialAuth />
</Form>
<div className="w-100 text-center mt-3">
  <Link to="/forgot-password">Forgot Password?</Link>
</div>
</Card.Body>
</Card>
<div className="w-100 text-center mt-2">
  Need an account? <Link to="/signup">Sign Up</Link>
</div>
</div>
</Container>
)
}
```

SIGNUP ACTIONS:

Similarly, to the login method, the sign up action is carried out through the `signup` method imported through `useAuth` from the authentication context file. In the `handleSubmit` asynchronous function the registration details entered by the user are checked. If both of the passwords entered match and the email address entered follows a proper email format the system will successfully register the user. Once registration is complete, the system redirects the user to the home page using the `react-router-dom` package. If any of the details provided by the user do not meet the set requirements, the user will be presented with an error message and will be asked to re-enter the details provided. The registration form provided to the user has a similar layout to the login form to maintain consistency. The code structure of the form and validation functions can be found in figures 16 and 17.

Figure 16 Creating Signup Form Part 1

```
import React, { useRef, useState } from 'react';
import { Form, Button, Card, Alert, Container } from 'react-bootstrap';
import { useAuth } from '../contexts/AuthContext';
import { Link, useHistory } from 'react-router-dom';
import SocialAuth from './SocialAuth';

export default function Signup(){

  const emailRef = useRef()
  const passwordRef = useRef()
  const passwordConfirmRef = useRef()
  const { signup } = useAuth()
  const [error, setError] = useState('')
  const [loading, setLoading] = useState(false)
  const history = useHistory()

  async function handleSubmit(e){
    e.preventDefault()

    if(passwordRef.current.value !== passwordConfirmRef.current.value){
      return setError('Passwords do not match')
    }

    try {
      setError('')
      setLoading(true)
      await signup(emailRef.current.value, passwordRef.current.value)
      history.push('/')
    } catch {
      setError('Failed to create an account')
    }
    setLoading(false)
  }

  return(
    <Container className="d-flex align-items-center justify-content-center" style={{ minHeight: "100vh" }}>
      <div className="w-100" style={{ maxWidth: '400px' }}>
```

Figure 17 Creating Signup Form Part 2

```
<Card>
  <Card.Body>
    <h2 className = "text-center mb-4">
      Sign up
    </h2>
    {error && <Alert variant="danger">{error}</Alert>}
    <Form onSubmit={handleSubmit}>
      <Form.Group id="email">
        <Form.Label>Email</Form.Label>
        <Form.Control type="email" ref={emailRef} required />
      </Form.Group>
      <Form.Group id="password">
        <Form.Label>Password</Form.Label>
        <Form.Control type="password" ref={passwordRef} required />
      </Form.Group>
      <Form.Group id="password">
        <Form.Label>Password Confirm</Form.Label>
        <Form.Control type="password" ref={passwordConfirmRef} required />
      </Form.Group>
      <Button
        disabled={loading}
        className="w-100"
        style={{ background: '#588cfc' }}
        variant="outline-light"
        type="submit"
      >
        Sign up
      </Button>
      <div className="w-100 text-center mt-2 mb-2">Or</div>
      <SocialAuth />
    </Form>
  </Card.Body>
</Card>
<div className="w-100 text-center mt-2">
  Already have an account? <Link to="/login">Log In</Link>
</div>
</div>
```

SOCIAL AUTHENTICATION WITH GOOGLE AND FACEBOOK:

Social authentication through Google and Facebook was implemented using the *selectedProvider* function available through firebase. When the user clicks on either of the two social authentication options, the system opens a popup page that redirect the user to the chosen provider. The implementation of this can be seen in figures 18 and 19. The *SocialAuth* function in figure 18 is rendered at the bottom of both the login and signup forms.

Figure 18 Social Authentication Provider Checker

```
import { toast } from 'react-toastify';
import firebase from '../firebase';
import { setUserProfileData } from './firebaseService';

export async function socialLogin(selectedProvider) {
  let provider;

  if(selectedProvider === 'facebook') {
    provider = new firebase.auth.FacebookAuthProvider();
  }

  if (selectedProvider === 'google') {
    provider = new firebase.auth.GoogleAuthProvider();
  }

  try {
    const result = await firebase.auth().signInWithPopup(provider);
    if (result.additionalUserInfo.isNewUser) {
      await setUserProfileData(result.user);
    }
  } catch (error) {
    toast.error(error.message);
  }
}
```

Figure 19 Social Authentication Interface

```
import React from 'react';
import { Button } from 'react-bootstrap';
import * as RiIcons from 'react-icons/ri';
import * as SiIcons from 'react-icons/si';
import { useHistory } from 'react-router';
import { socialLogin } from '../contexts/firebaseService';

export default function SocialAuth() {

  const history = useHistory()

  const handleSocialLogin = async (provider) => {
    socialLogin(provider);
    history.push('/');
  }

  return (
    <>
      <Button
        onClick={() => handleSocialLogin('facebook')}
        className="w-100"
        style={{ background: '#000033', color: '#fff', marginBottom: '10px' }}
        variant="outline-light"
      >
        <RiIcons.RiFacebookFill className="mr-2" style={{ marginTop: '-3px' }} />
        Login with Facebook
      </Button>
      <Button
        onClick={() => handleSocialLogin('google')}
        className="w-100"
        style={{ background: '#c40000', color: '#fff' }}
        variant="outline-light"
      >
        <SiIcons.SiGmail className="mr-3" style={{ marginTop: '-3px' }} />
        Login with Gmail
      </Button>
    </>
  )
}
```

RESET ACCOUNT PASSWORD:

The option to reset the account password is available for users that have an existing account but cannot remember the password. Figures 20 and 21 show the code responsible for resetting the user's password. The asynchronous function in figure 20 checks the email that the user entered and if the email address exists in the database, the user will receive an email with further instructions for resetting the account password. Figure 21 shows the structure of the webpage responsible for giving the user access to the reset password functionality.

Figure 20 Reset Account Password Part 1

```
export default function ForgotPassword(){
  const emailRef = useRef()
  const { resetPassword } = useAuth()
  const [error, setError] = useState('')
  const [message, setMessage] = useState('')
  const [loading, setLoading] = useState(false)

  async function handleSubmit(e){
    e.preventDefault()

    try {
      setMessage('')
      setError('')
      setLoading(true)
      await resetPassword(emailRef.current.value)
      setMessage('Check your inbox for further instructions')
    } catch {
      setError('Failed to reset password')
    }
    setLoading(false)
  }
}
```

Figure 21 Reset Account Password Part 2

```
return(
  <Container className="d-flex align-items-center justify-content-center" style={{ minHeight: "100vh" }}>
    <div className="w-100" style={{ maxWidth: '400px' }}>
      <Card>
        <Card.Body>
          <h2 className = "text-center mb-4">
            Password Reset
          </h2>
          {error && <Alert variant="danger">{error}</Alert>}
          {message && <Alert variant="success">{message}</Alert>}
          <Form onSubmit={handleSubmit}>
            <Form.Group id="email">
              <Form.Label>Email</Form.Label>
              <Form.Control type="email" ref={emailRef} required />
            </Form.Group>
            <Button
              disabled={loading}
              className="w-100"
              style={{ background: '#588cfc' }}
              type="Submit"
            >
              Reset Password
            </Button>
          </Form>
          <div className = "w-100 text-center mt-3">
            <Link to="/login">Log In</Link>
          </div>
        </Card.Body>
      </Card>
      <div className = "w-100 text-center mt-2">
        Need an account? <Link to="/signup">Sign Up</Link>
      </div>
    </div>
  </Container>
)
```


UPDATE USER DETAILS:

Figure 22 shows the function responsible for updating the user password and email on the update profile page. The user can reset the account password by entering a new one on the provided form. The Promise JavaScript constructor was used in the asynchronous *handleSubmit* function to let the function supply the system with the promises variable which let the system know that a value will be returned at some in the future.

Figure 22 Update User Account Details

```
async function handleSubmit(e){
  e.preventDefault()

  if (passwordRef.current.value !== passwordConfirmRef.current.value){
    |   return setError('Passwords do not match')
  }

  const promises = []
  setLoading(true)
  setError('')
  if (emailRef.current.value !== currentUser.email) {
    |   promises.push(updateEmail(emailRef.current.value))
  }

  if (passwordRef.current.value){
    |   promises.push(updatePassword(passwordRef.current.value))
  }

  Promise.all(promises).then(() => {
    |   history.push('/profile')
  }).catch(() => {
    |   setError('Failed to update account')
  }).finally(() => {
    |   setLoading(false)
  })
}
```

UNAUTHENTICATED USER VISIBILITY RESTRICTION:

Most of the features available on the website have restricted visibility and can be viewed only by successfully authenticated users. A private route function, figure 23, was created for this functionality. This function checks the user's status and redirects accordingly. If the current user is logged into an account, the restricted webpages become available. Users that are not authenticated are redirected to the login page.

Figure 23 Restricted Content Visibility

```
import React from 'react'
import { Route, Redirect } from 'react-router-dom'
import { useAuth } from '../contexts/AuthContext'

export default function PrivateRoute({component: Component, ...rest}) {
  const { currentUser } = useAuth()

  return (
    <Route
      {...rest}
      render={props => {
        return currentUser ? <Component {...props} /> : <Redirect to="/login" />
      }}
    >
    </Route>
  )
}
```

WEBSITE SIDEBAR NAVIGATION:

Registered users can navigate the website using a toggle sidebar that contains links to various sections of the website. The user can access the profile page, check the settings page, view wish boards and saved items using this menu, and navigate to any other available page. Figure 24 shows how the sub-sections located inside of the Items section on the menu become visible using a dropdown list. The *SubMenu* class in figure 24 receives the items children props from the *UserSidebar* class in figure 25. The item props are passed to the *SubMenu* custom components to render the sidebar sections. In the *UserSidebar* class the sidebar sections are mapped to the *SubMenu* component.

Figure 24 Sidebar Navigation

```
export default function SubMenu({ item }) {  
  
  const [subnav, setSubnav] = useState(false);  
  
  const showSubnav = () => setSubnav(!subnav);  
  
  return (  
    <>  
      <SidebarLink to={item.path} onClick={item.subNav && showSubnav}>  
        <div>  
          {item.icon}  
          <SidebarLabel>  
            {item.title}  
          </SidebarLabel>  
        </div>  
        <div>  
          {item.subNav && subnav  
            ? item.iconOpened  
            : item.subNav  
            ? item.iconClosed  
            : null}  
        </div>  
      </SidebarLink>  
      {subnav && item.subNav.map((item, index) => {  
        return (  
          <DropdownLink to={item.path} key={index}>  
            {item.icon}  
            <SidebarLabel>{item.title}</SidebarLabel>  
          </DropdownLink>  
        )  
      })}  
    </>  
  )  
}
```

Figure 25 Sidebar Navigation Components

```
export default function UserSidebar() {  
  
  const [sidebar, setSidebar] = useState(false);  
  
  const showSidebar = () => setSidebar(!sidebar);  
  
  return (  
    <>  
      <IconContext.Provider value={{ color: '#fff' }}>  
        <Nav>  
          <NavIcon to="#" style={{ marginLeft: '-185px' }}>  
            <FaIcons.FaBars onClick={showSidebar} />  
          </NavIcon>  
          <SidebarNav sidebar={sidebar}>  
            <SidebarWrap>  
              {SidebarData.map((item, index) => {  
                return <SubMenu item={item} key={index} />  
              })}  
            </SidebarWrap>  
          </SidebarNav>  
        </Nav>  
      </IconContext.Provider>  
    </>  
  )  
}
```

CREATE NEW WISH BOARD ACTIONS:

Figure 26 and 27 show how the user can create new wish boards using the website. The user is required to give the wish board a title, and the system will assign an ID to the document using uuidv4 which generates an encrypted 4-bits universally unique ID. Figure 27 presents the layout structure of the new wish board form available on the website.

Figure 26 Create New Wish Board Part 1

```
const ref = firebase.firestore().collection("wishboards");
const [title, setTitle] = useState("");

function addWishboard(newWishboard) {
  ref
    .doc(newWishboard.id)
    .set(newWishboard)
    .catch((err) => {
      console.log(err);
    });
}
```

Figure 27 Create New Wish Board Part 2

```
<Container className="d-flex align-items-center justify-content-center" style={{ minHeight: "100vh", marginTop: '-100px' }}>
  <div className="w-100" style={{ maxWidth: '400px' }}>
    <Card>
      <Card.Body>
        <h2 className="text-center mb-4">
          Create New Wish Board
        </h2>
        <Form>
          <Form.Group>
            <Form.Label>Wish Board Name</Form.Label>
            <Form.Control
              type="text"
              placeholder="Enter Wish Board Name"
              name="title"
              onChange={(e) => setTitle(e.target.value)}
            />
          </Form.Group>
          <Button
            onClick={() => addWishboard({ title, id: uuidv4() })}
            className="w-100"
            style={{ background: '#588cfc' }}
            variant="outline-light"
            type="submit"
          >
            Save
          </Button>
        </Form>
      </Card.Body>
    </Card>
    <div className="w-100 text-center mt-2">
      <Link to="/boards">Return to Wish Boards</Link>
    </div>
  </div>
</Container>
```

REMOVE WISH BOARD ACTION:

Users can delete wish boards using from their account by pressing on the trashcan icon displayed next to each wish board on the manage wish boards page. Figure 28 shows how this functionality was integrated with the Firestore database. Wish boards are removed based on the document ID that is mapped to a div tag as a key in figure 29. Wish boards created by the user are rendered from the database and displayed in containers on the screen. This, along with the code structure for the page can be seen in figure 29.

Figure 28 Delete Wish Board Part 1

```
function deletewishboard(wishboard) {
  ref
  .doc(wishboard.id)
  .delete()
  .catch((err) => {
    console.log(err);
  });
}
```

Figure 29 Delete Wish Board Part 2

```
<>
{wishboards.map((wishboard) => (
  <Button
    className="d-flex align-items-center justify-content-center mt-3 shadow"
    style={{ width: '34rem', height: '65px', borderRadius: '18px', fontSize: '18px', background: '#fff', color: '#000', borderColor: '#fff' }}
  >
    <div className="flex flex-row" key={wishboard.id}>
      <h3 style={{ fontSize: '22px' }}>{wishboard.title}</h3>
    </div>
    <div className="flex flex-row ml-4">
      <RiIcons.RiDeleteBin2Line
        style={{ width: '30px', height: '30px', float: 'right', justifySelf: 'end'}}
        onClick={() => deletewishboard(wishboard)}
      />
    </div>
  </Button>
)
)
}</>
```

SHOW SAVED ITEMS ON THE WEBSITE:

The items saved by the user can be viewed on the website using the navigation sidebar. The product details saved using the browser extension or inserted by the user using the 'Add New Item Form' option on the website is retrieved from Firestore and passed down to the page components to be displayed on the webpage, this is shown in figure 30. Figure 31 contains the structure in which data is printed on the website.

Figure 30 Display Items Part 1

```
const [savedItem, setSavedItem] = useState([]);
const [loading, setLoading] = useState(false);

const ref = firebase.firestore().collection("items");

function getItems() {
  setLoading(true);
  ref.onSnapshot((querySnapshot) => {
    const items = [];
    querySnapshot.forEach((doc) => {
      items.push(doc.data());
    });
    setSavedItem(items);
    console.log(items);
    setLoading(false);
  });
}

useEffect(() => {
  getItems();
  // eslint-disable-next-line react-hooks/exhaustive-deps
}, []);
```

Figure 31 Display Items Part 2

```
<>
{savedItem.map((item) => (
  <div className="item text-center shadow" style={{ width: '240px', height: '420px', marginRight: '8px' }} key={item.id}>
    <div className="cont cont-size">
      <a href={item.location}>
        <img src={item.photoURL} alt="item" className="item-img-top" />
      </a>
    </div>
    <div className="item-title text-dark">
      <p className="item-name" style={{ marginTop: '-40px' }}>{item.displayName}</p>
      <p className="item-text text-secondary">Size: {item.size}</p>
      <p className="item-text text-secondary">Price: {item.price}</p>
      <div style={{ marginTop: '-15px', marginLeft: '180px' }}>
        <AiIcon.AiOutlineClose style={{ width: '25px', height: '25px' }} />
      </div>
    </div>
  </div>
)}}
</>
```

SEARCH SAVED ITEMS ON THE WEBSITE:

The items saved by the user can be filtered by wish board title using the search bar on the “Filter Items” page. Figure 32 shows that filtering is performed on the database elements and only the objects that match the input are printed on the screen. The objects are set to lower case to reduce user inconvenience when searching for wish boards.

Figure 32 Search Saved Items

```
<InputGroup className="mb-3">
  <FormControl
    aria-label="Default"
    aria-describedby="inputGroup-sizing-default"
    placeholder="Search Wish Boards"
    onChange={event => {setSearchTerm(event.target.value)}}
  />
</InputGroup>
{savedItem.filter((item) => {
  if(searchTerm === ""){
    return item;
  } else if(item.wishboard.toLowerCase().includes(searchTerm.toLowerCase())){
    return item;
  }
}}}
```

BROWSER EXTENSION MANIFEST:

Figure 33 shows the *Manifest.json* file where the Chrome extension dependencies are stored. It contains the content security policy that enables Firebase, Bootstrap, and jQuery to be used within the project.

Figure 33 Manifest Setup

```
{
  "manifest_version": 2,
  "name": "Shopping Wishlist Extension",
  "version": "1.5.2",
  "content_scripts": [
    {
      "matches": ["<all_urls>"],
      "js": ["app.js"],
      "run_at": "document_end"
    }
  ],
  "background": {
    "page": "background.html",
    "persistent": false
  },
  "permissions": [
    "activeTab",
    "<all_urls>",
    "tabs"
  ],
  "content_security_policy": "script-src 'self' https://www.gstatic.com/ https://*.firebaseio.com https://www.googleapis.com https://code.jquery.com https://maxcdn.bootstrapcdn.com;
  "browser_action": {
    "default_icon": "shopping-bag.png",
    "default_popup": "popup.html",
    "default_title": "Shopping Wishlist"
  }
}
```

RETRIEVING WISH BOARDS FROM FIRESTORE:

A function was created to get the details stored inside of the wish board collection on Firebase, the function was after mapped in the snapshot statement, figure 34. The ID of a select HTML tag was called, and within that tag a new option HTML tag was created to render the wish board titles inside of the plugin popup page.

Figure 34 Retrieving Wish Boards from Firestore

```
const wishboardOptions = document.querySelector('#wishboard-options');

//creating element and rendering wishboards in dropdown list
function passWishboards(doc){
  let title = document.createElement('option');

  title.setAttribute = doc.id;
  title.textContent = doc.data().title.charAt(0) + doc.data().title.slice(1);

  wishboardOptions.appendChild(title);
}

db.collection('wishboards').get().then((snapshot) => {
  snapshot.docs.map(doc => {
    passWishboards(doc);
  })
})
```

GET THE URL OF THE CURRENT TAB:

The function that scrapes the URL of the current tab was integrated using jQuery. The *chrome.tabs* method provided by Google was enabled in the *Manifest.json* file under the permissions section. Enabling the *tabs* permission allowed the Chrome extension to send requests to the content script to view the URL of the currently open tab. The URL of the tab is passed and stored in an object with a *host* ID.

Figure 35 Getting the Current Tab URL

```
$(document).ready(function () {
  chrome.tabs.getSelected(null, function (tab) {
    var link = document.createElement("input");
    link.href = tab.url;
    $("#host").html(link.href);
  })
});
```

SAVING ITEM DETAILS TO THE ITEMS COLLECTION:

Figure 36 contains a snippet of the function responsible for saving the item details to Firestore. A form HTML tag with the *add-item* ID holds the content of the popup page. Using the ID's of the item name, reduction preference, wish board choice, and item URL the product details are inserted into the items collection on Firebase. To prevent the *start* function from rendering before the page content the *window.onload* method was inserted.

Figure 36 Saving Item Details

```
function start() {
  const form = document.querySelector('#add-item');

  form.addEventListener('submit', (e) => {
    e.preventDefault();

    db.collection('items').add({
      displayName: form.itemName.value,
      reduction: form.twentyPercent.value,
      wishboard: form.wishboardOptions.value,
      pathURL: form.host.value
    });
  });
}

window.onload=start;
```


2.4. Graphical User Interface (GUI)

The website and browser extension have a minimalistic layout with limited information displayed on pages to reduce user confusion and improve the user experience. A simple colour scheme was selected for printing information to users. The key colours used for styling the project are dark navy, white, and soft blue. These colours were consistently used when designing the project's interface to create a user friendly layout for the system.

LANDING PAGE:

The website contains a landing page for unregistered users which displays general information about the system and the services offered. Figure 37 shows the screen layout and the navigation options available to the user on this page. This section of the website uses react-scroll which adds a smooth scrolling effect to the page when scrolling to other sections on this page. Using the get started button, the user can navigate to the sign up page where registration takes place. The login button on the navigation bar redirects the user to the login form.

Figure 37 Unregistered User Landing Page



SIGNUP AND LOGIN PAGES:

Figure 38 show the layout of the signup and login forms. Both forms contains input fields with small labels. The two forms show social authentication methods for Facebook and Google. These forms have a similar simple and modern layout to maintain consistency.

Figure 38 Signup and Login Forms

The image displays two side-by-side form layouts. The left form is titled "Log In" and contains an "Email" input field, a "Password" input field, a blue "Log In" button, an "Or" separator, a dark blue "Login with Facebook" button, a red "Login with Gmail" button, and a blue "Forgot Password?" link. At the bottom, it says "Need an account? Sign Up". The right form is titled "Sign up" and contains an "Email" input field, a "Password" input field, a "Password Confirm" input field, a blue "Sign up" button, an "Or" separator, a dark blue "Login with Facebook" button, and a red "Login with Gmail" button. At the bottom, it says "Already have an account? Log In".

HOME PAGE AND WEBSITE SIDEBAR NAVIGATION:

Figure 39 shows the home page of the website. this page contains card views with direct links to online shops that users might be interested in visiting. From this page users can click on the menu icon located in the corner of the screen to access the sidebar which will enable other navigation options. These options can be seen in figure 40 where the toggle sidebar is visible.

Figure 39 Home Page for Registered Users

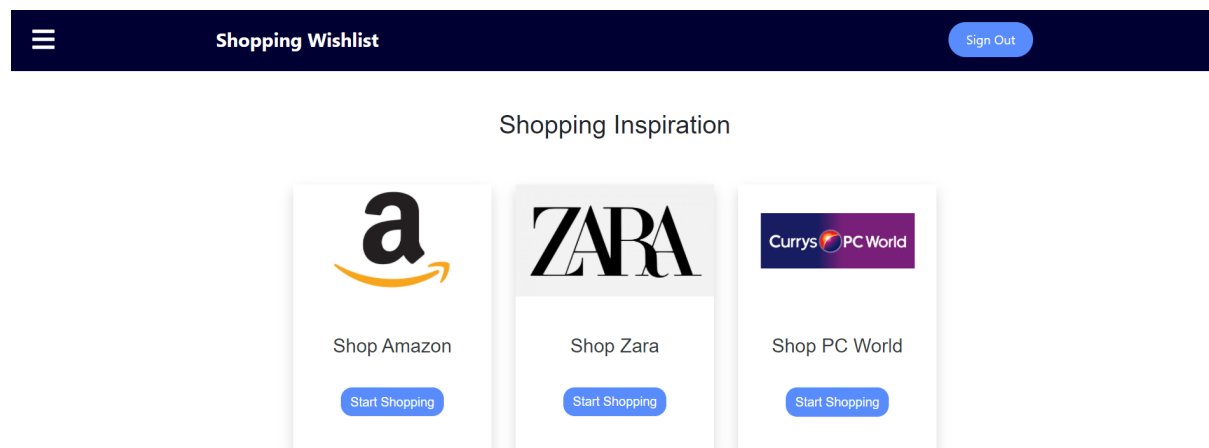
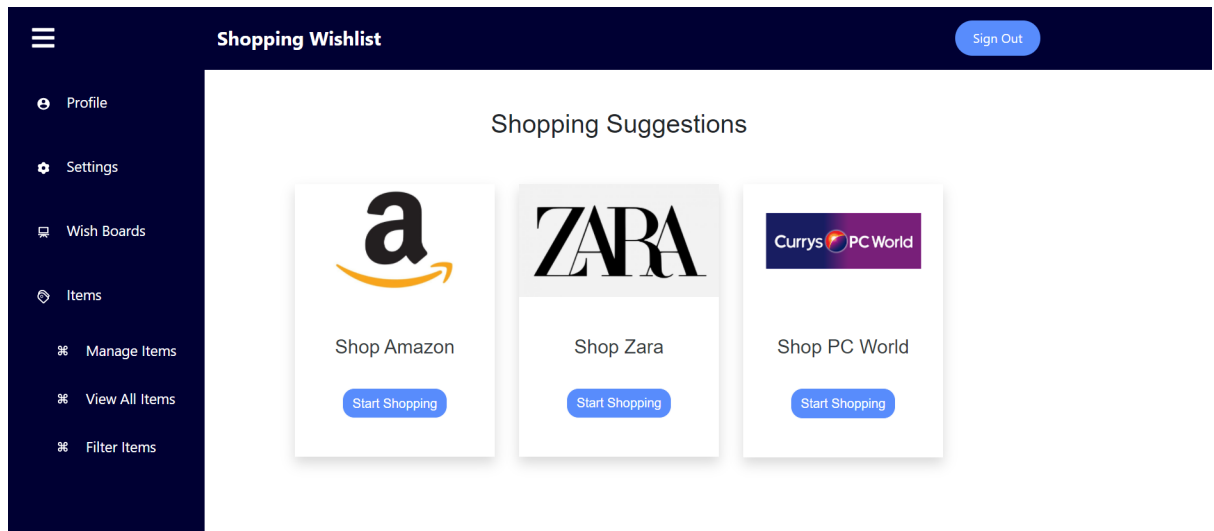


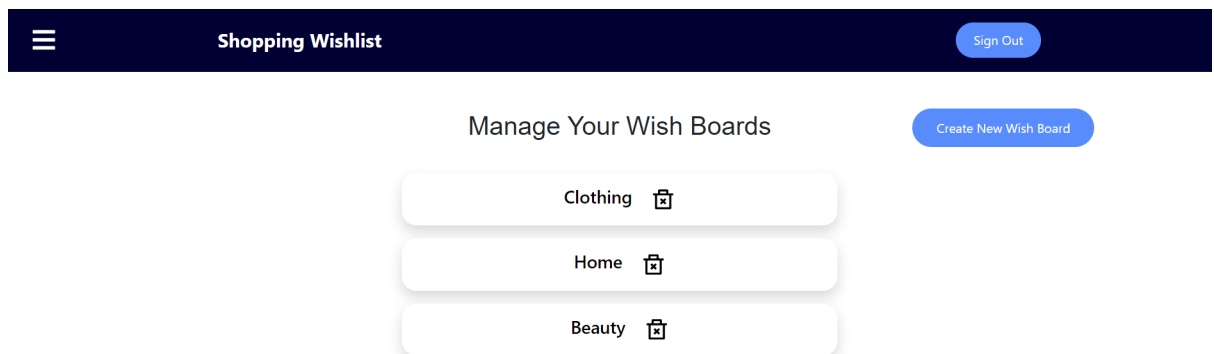
Figure 40 Home Page with Released Sidebar Navigation



MANAGE WISH BOARDS:

Figure 41 shows the wish board page where users can create wish boards using the large soft blue button or manage existing wish boards. Each board in the user's account is displayed in an individual container. Wish boards can be removed by the user using the trashcan icon located next to the board name.

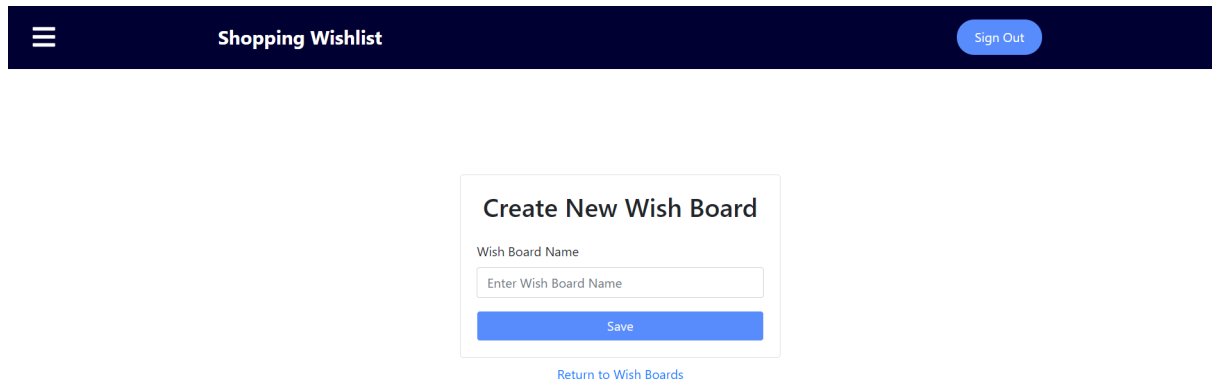
Figure 41 Wish Board Page



CREATE NEW WISH BOARD FORM:

Figure 42 shows the form displayed to the user when the “Create New Wish Board” button is clicked. The user is required to enter a name for the wish board before clicking on the save button to insert a new object into the database. Once the wish board is created, the “Return to Wish Board” link will redirect the user to the previous page.

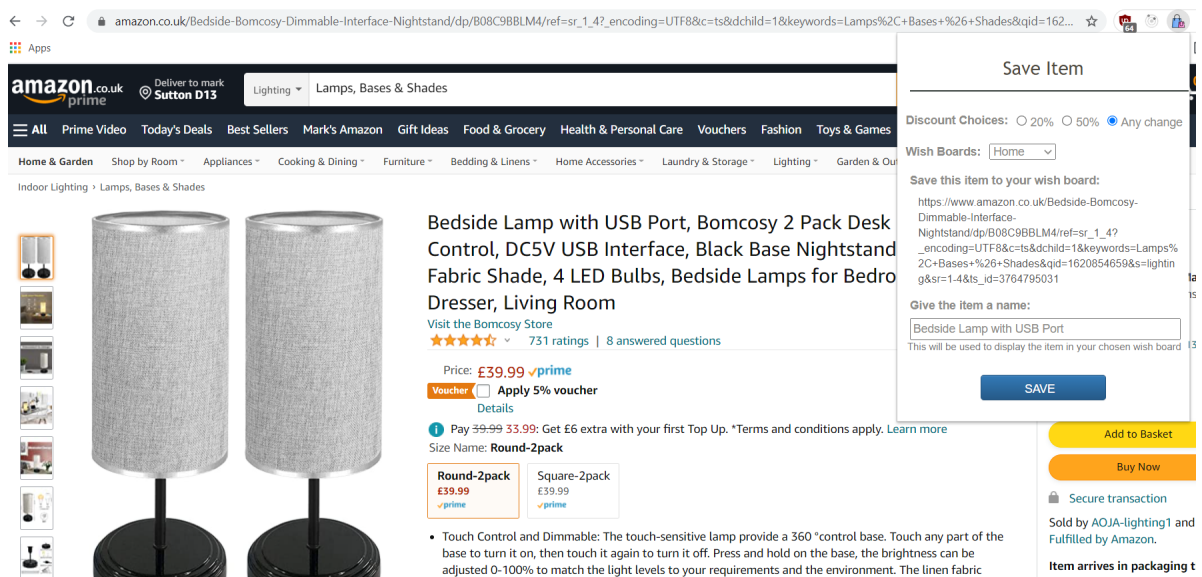
Figure 42 Create New Wish Board



SAVE ITEM WITH BROWSER EXTENSION:

In figure 43 the process the user follows when saving an item to a specific wish board can be seen. The browser extension consists of three interactive fields that require a little amount of effort from the user. The user can select any of the available discount radio options with a simple click. The list of existing wish boards can be viewed by clicking on the dropdown arrow present in the box. And lastly, an input field where the user can give the item a descriptive name, this name will be shown on the website inside of the item container.

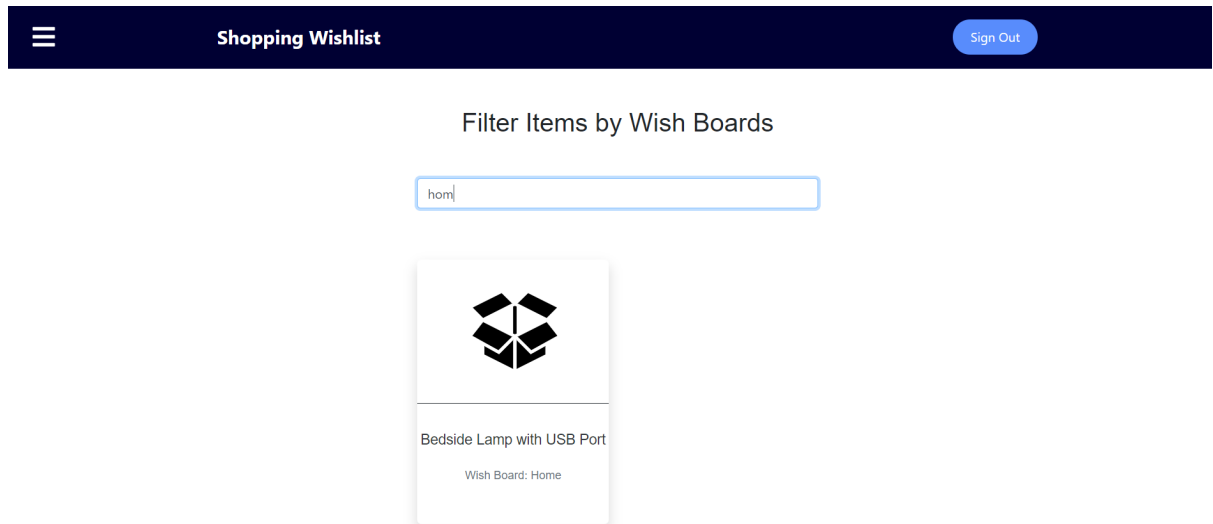
Figure 43 Save Item with Chrome Extension



VIEW SAVED ITEMS ON THE WEBSITE:

Figure 44 shows the item that was saved using the browser extension. Using the search bar at the top of the page, the items saved by the user can be filtered based on the wish board name. As soon as the first letter of input is typed into the search bar, the system starts filtering the database for possible matches.

Figure 44 View Item on Website



2.5. Testing

SYSTEM INTEGRATION TESTING:

The project was tested rigorously throughout the development cycle. The website and browser extension undertook testing when any new feature was added to the project. On the website side of integration, tests were performed and executed both on the localhost and on the Heroku deployed website. On the Chrome extension side, features implemented were tested using a background script. These system tests were included in the project to catch errors within the system and to scan the newly added features.

DATABASE TESTING:

The Cloud Firestore Firebase database used for storing the data types required for this project involved extensive testing when new features that interact with the database were added. To enable both the website and the browser plugin to access the same database secrets, data and communicate with each other, the Firebase project setting were altered to contain two web applications. This established communication between the otherwise independent projects as both web applications share the same Firebase keys and secrets.

BROWSER RESPONSIVE TESTING:

Browser responsive tests were run on the website to analyse the responsiveness of the project across different devices. These tests were carried out on each page of the website to ensure that the project follows the responsive design interface principles. The device compatibility tests were executed using the device toolbar available in the console of the Chrome browser.

CHROME DEVTOOLS:

The Chrome DevTools available on the Google Chrome browser was used for testing the features implemented in the browser extension. The DevTools were applied for diagnosing problems within the system. A background script was linked with the *Manifest.json* file and was use for debugging on the browser. The background script alerts when for potential vulnerabilities appear in the console to help identify errors faster. The Chrome DevTools enable the background script to inspect the unpacked extension bugs during the development stage.

REDUX TESTING:

The Redux dependency was included into the website created for this project to test the React states and behaviours. The Redux DevTools plugin was installed to test the state changes as actions are carried out on the website through localhost. This tool provides error trace functionality to help identify the exact location where bugs occurred. Using the time-travel debugging offered by Redux actions and states were traced to origin.

PUPPETEER TESTING:

The Chrome browser extension was tested in the command line using the Puppeteer library. This tool was selected as it contains a powerful API that creates a headless browser on the local environment. The Puppeteer tool was useful for testing the browser extensions actions as it can replicate any desired steps. A scraper script was created to provide the Puppeteer with the actions to be performed on a specified web page. The tool can retrieve specific data from the web page or generate a screenshot of the page's content. This tool was used to test the accuracy of the webpage URL retrieved by the browser extension.

PERFORMANCE TESTING:

A series of performance tests were completed using the Loadster tool and plugin. The performance of the website was tested to optimize request response times and prevent downtime. The Loadster tool conducted separate scenario tests for specific features and also full system test to analyse the entire project. Two distinct scripts were used for these tests, one of the scripts was a protocol script to analyse HTTP request and the second script was a browser script to test the system under a full headless browser.

2.6. Evaluation

PERFORMANCE TESTING RESULTS:

Figures 45 and 46 show the results generated by Loadster when the performance of the system was analysed. To evaluate the performance of the system, the project had to be deployed to a cloud provider and using the URL of the deployed project Loadster recorded the web actions performed and launched 5 minute tests with 25 protocol bots that replicated the previously recorded actions. In figure 45 the average load time of a series of web requests is displayed. The graph represents each request with a uniquely coloured line and shows the average load times for the 5 minutes predefined time limit. Figure 46 shows the results of the response time percentiles test performed using the same time limit and number of protocol bots. The tests were executed on a protocol script to analyse the response time of HTTP requests.

Figure 45 Performance Load Time Test Results

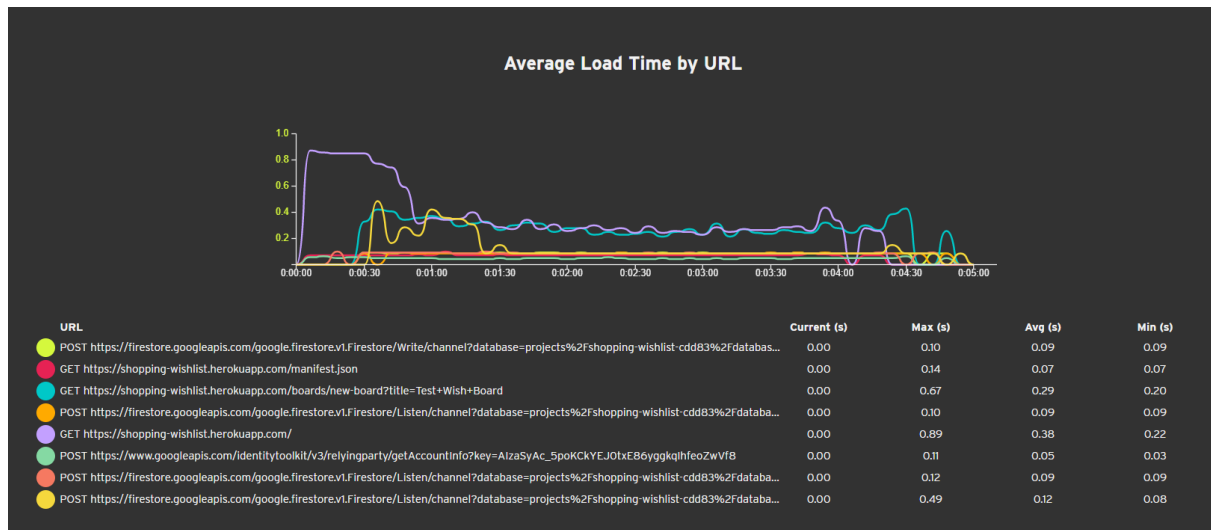
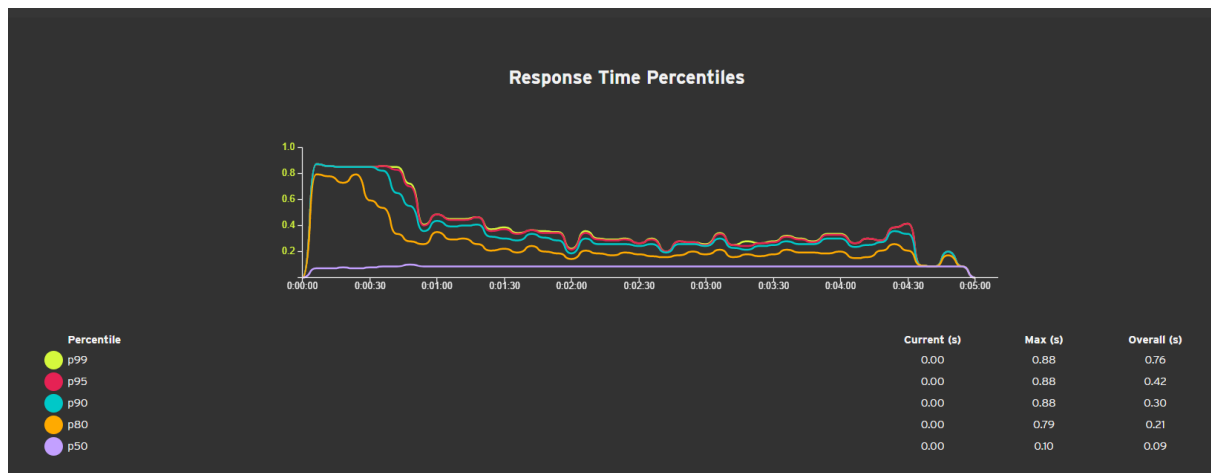


Figure 46 Performance Response Time Test Results



DEPLOYMENT AND VERSION CONTROL:

The browser extension and website have been stored on individual GitHub repositories from the earliest development stage to reduce the risk of misplacing the project. As features were implemented and changed, sub-branches of the working tree were created to prevent bugs from corrupting the main project branch. The project was deployed to the Heroku Cloud Service by creating a link between the GitHub repository and the Heroku project. Automated deployments were enabled on Heroku to update the deployed project with the most recent changes merged into the main GitHub branch.

CODE SUMMARY:

Figure 47 shows a code summary table of the languages used for the creation the website. It shows a breakdown of the number of files, code lines, and total lines of code created using each programming language. Similarly, in figure 48 a summary table of the directory is shown along with the specific path to each subdirectory of the project. Figure 48 also shows the number of files and the total number of code lines in each directory.

Figure 47 Language Code Summary Table

language	files	code	comment	blank	total
JavaScript React	35	1,512	6	151	1,669
JavaScript	39	1,402	12	207	1,621
CSS	2	81	0	20	101
XML	9	57	0	1	58
JSON	1	12	0	1	13

Figure 48 Directory Code Summary Table

path	files	code	comment	blank	total
.	86	3,064	18	380	3,462
actions	3	73	0	7	80
api	1	7	0	1	8
components	57	2,647	12	318	2,977
components\app-bar	2	102	0	18	120
components\dashboard-cards	3	84	0	15	99
components\footer	8	300	0	29	329
components\hero-section	2	139	0	19	158
components\info-section	3	200	0	19	219
components\items	8	380	4	49	433
components\navbar	2	208	0	27	235
components\profile	6	178	3	20	201
components\services	2	111	0	16	127
components\sidebar	2	119	0	14	133
components\user-sidebar	3	179	0	23	202
components\wish-boards	4	157	3	18	178
contexts	3	82	0	21	103
firebase	1	23	2	6	31
hooks	2	45	0	6	51
images	9	57	0	1	58
store	1	3	0	0	3

PUPPETEER TESTING RESULTS:

Figure 49 shows the results retrieved by the *scraper.js* script created for executing the Puppeteer headless browser. The URL retrieved by the browser extension was copied and rendered in the *scraper.js* script to evaluate the accuracy of the URL addresses gathered by the plugin. The URL retrieved from the Curry's website by the Chrome extension was inserted into the Puppeteer script. After the script concluded the execution of the command, the screenshot shown in figure 49 was generated as a PNG file. This determined that the browser extension retrieved the correct item URL.

Figure 49 Puppeteer Generated Screenshot of Curry's Webpage



The screenshot displays the Curry's PC World website interface. At the top, there is a dark blue navigation bar with the Curry's PC World logo, a search bar, and links for Menu, Stores, Sign in, and Basket. Below the navigation bar, a breadcrumb trail reads: Home > Computing > Laptops > Laptops > VivoBook K403JA 14" Laptop - Intel® Core™ i5, 512 GB SSD, Silver Blue. The main product title is "ASUS VivoBook K403JA 14" Laptop - Intel® Core™ i5, 512 GB SSD, Silver Blue" with a product code of 782512. A star rating of 4.5 is shown, along with the text "Read 270 customer reviews" and a link to "Ask an owner". A red "TOP OFFER" badge is positioned to the left of the laptop image. The laptop is shown in a silver finish with a blue screen displaying a mountain scene. To the right of the laptop, the price is listed as €719.00, with a red "Save €180.99" badge indicating a discount from a previous price of €899.99 (valid from 19/02/21 to 23/03/21). Below the price, there are two buttons: a green "Add to basket" button and a white "Save for later" button. At the bottom, there is a section titled "Check availability in your area" with a search input field for "Enter town or eircode" and a "Got it" button.

3.0 Conclusions

This project set out to create a web application and a browser extension for Google Chrome. The aim of the project was to develop a shopping assistant system that is easy to use and inclusive to people of any age group or technical skill. The main purpose of this project was to create a browser extension that users can easily access on any webpage and to allow the user to save any item to a secure location. The second aim of the project was to enable users to create wish boards for storing items, and to allow users to easily get access to the previously saved products. The project succeeded in creating the browser extension and website which enables the system's key functionality to users. The developed system has the potential to reduce the time that the user spends doing online shopping, and to enable users to make educated purchases. The Shopping Wishlist system allows users to shop online better and reduces the stress related with online shopping and impulse purchases as products can be stored in wish boards for an unlimited period of time, or until deleted by the user. The features provided by the project can benefit users in many ways and provide aid while shopping online.

4.0 Further Development or Research

The system can be expanded and enhanced in various ways in the future as online shopping becomes the preferred shopping method for many people. As the project was created within a strict time limit, potential features were overlooked, however, the system can be further extended to contain new enhancements and features.

A key enhancement that can be performed on the system is cross browser accessibility. The current browser extension was built and optimised with emphasis on running on the Google Chrome browser. This can limit the number of users that interact with the system and create user inconvenience. In the future the system could be expanded to operate on other popular browsers such as Firefox and Safari.

The item containers could be enhanced to include more details about the item. This could enable the user to recognise the item faster, making the system more efficient. Additional security measures could be implemented in the future to prevent data breaches and keep user data secure. The enhancements mentioned above can improve the user experience and efficiency of the overall system.

5.0 References

1. CODE, V. Visual Studio Code - Code Editing. Redefined In-text: (Code, 2021) Your Bibliography: Code, V., 2021. Visual Studio Code - Code Editing. Redefined. [online] Code.visualstudio.com. Available at: <<https://code.visualstudio.com/>> [Accessed 6 May 2021].
2. REACT – A JAVASCRIPT LIBRARY FOR BUILDING USER INTERFACES In-text: (React – A JavaScript library for building user interfaces, 2021) Your Bibliography: Reactjs.org. 2021. React – A JavaScript library for building user interfaces. [online] Available at: <<https://reactjs.org/>> [Accessed 6 May 2021].
3. JAVASCRIPT.COM In-text: (JavaScript.com, 2021) Your Bibliography: Javascript.com. 2021. JavaScript.com. [online] Available at: <<https://www.javascript.com/>> [Accessed 6 May 2021].
4. HTML STANDARD In-text: (HTML Standard, 2021) Your Bibliography: Html.spec.whatwg.org. 2021. HTML Standard. [online] Available at: <<https://html.spec.whatwg.org/>> [Accessed 6 May 2021].
5. CSS: CASCADING STYLE SHEETS | MDN In-text: (CSS: Cascading Style Sheets | MDN, 2021) Your Bibliography: Developer.mozilla.org. 2021. CSS: Cascading Style Sheets | MDN. [online] Available at: <<https://developer.mozilla.org/en-US/docs/Web/CSS>> [Accessed 6 May 2021].
6. MANIFESTS | APPS SCRIPT | GOOGLE DEVELOPERS In-text: (Manifests | Apps Script | Google Developers, 2021) Your Bibliography: Google Developers. 2021. Manifests | Apps Script | Google Developers. [online] Available at: <<https://developers.google.com/apps-script/concepts/manifests>> [Accessed 6 May 2021].
7. REACT ROUTER: DECLARATIVE ROUTING FOR REACT In-text: (React Router: Declarative Routing for React, 2021) Your Bibliography: ReactRouterWebsite. 2021. React Router: Declarative Routing for React. [online] Available at: <<https://reactrouter.com/web/guides/quick-start>> [Accessed 7 May 2021].
8. REACT-SCROLL In-text: (react-scroll, 2021) Your Bibliography: npm. 2021. react-scroll. [online] Available at: <<https://www.npmjs.com/package/react-scroll>> [Accessed 7 May 2021].
9. NPM In-text: (npm, 2021) Your Bibliography: Npmjs.com. 2021. npm. [online] Available at: <<https://www.npmjs.com/>> [Accessed 12 May 2021].
10. YARN In-text: (Yarn, 2021) Your Bibliography: Yarnpkg.com. 2021. Yarn. [online] Available at: <<https://yarnpkg.com/>> [Accessed 7 May 2021].
11. BABEL · THE COMPILER FOR NEXT GENERATION JAVASCRIPT In-text: (Babel · The compiler for next generation JavaScript, 2021) Your Bibliography: Babeljs.io. 2021. Babel · The compiler for next generation JavaScript. [online] Available at: <<https://babeljs.io/>> [Accessed 7 May 2021].
12. REACT-BOOTSTRAP In-text: (React-Bootstrap, 2021) Your Bibliography: React-bootstrap.github.io. 2021. React-Bootstrap. [online] Available at: <<https://react-bootstrap.github.io/>> [Accessed 7 May 2021].

13. STYLED-COMPONENTS: DOCUMENTATION In-text: (styled-components: Documentation, 2021) Your Bibliography: styled-components. 2021. styled-components: Documentation. [online] Available at: <<https://styled-components.com/docs>> [Accessed 7 May 2021].
14. SEMANTIC UI In-text: (Semantic UI, 2021) Your Bibliography: Semantic-ui.com. 2021. Semantic UI. [online] Available at: <<https://semantic-ui.com/>> [Accessed 7 May 2021].
15. REACT ICONS In-text: (React Icons, 2021) Your Bibliography: React-icons.github.io. 2021. React Icons. [online] Available at: <<https://react-icons.github.io/react-icons/>> [Accessed 7 May 2021].
16. JEST ·  DELIGHTFUL JAVASCRIPT TESTING In-text: (Jest ·  Delightful JavaScript Testing, 2021) Your Bibliography: Jestjs.io. 2021. Jest ·  Delightful JavaScript Testing. [online] Available at: <<https://jestjs.io/>> [Accessed 7 May 2021].
17. REDUX - A PREDICTABLE STATE CONTAINER FOR JAVASCRIPT APPS. | REDUX In-text: (Redux - A predictable state container for JavaScript apps. | Redux, 2021) Your Bibliography: Redux.js.org. 2021. Redux - A predictable state container for JavaScript apps. | Redux. [online] Available at: <<https://redux.js.org/>> [Accessed 7 May 2021].
18. FIREBASE In-text: (Firebase, 2021) Your Bibliography: Firebase. 2021. Firebase. [online] Available at: <<https://firebase.google.com/>> [Accessed 7 May 2021].
19. CHROME WEB STORE In-text: (Chrome Web Store, 2021) Your Bibliography: Chrome.google.com. 2021. Chrome Web Store. [online] Available at: <<https://chrome.google.com/webstore/category/extensions/extensions.html>> [Accessed 7 May 2021].
20. GITHUB: WHERE THE WORLD BUILDS SOFTWARE In-text: (GitHub: Where the world builds software, 2021) Your Bibliography: GitHub. 2021. GitHub: Where the world builds software. [online] Available at: <<https://github.com/>> [Accessed 9 May 2021].
21. JS.FOUNDATION, J. jQuery In-text: (js.foundation, 2021) Your Bibliography: js.foundation, J., 2021. jQuery. [online] Jquery.com. Available at: <<https://jquery.com/>> [Accessed 9 May 2021].
22. PUPPETEER | TOOLS FOR WEB DEVELOPERS | GOOGLE DEVELOPERS In-text: (Puppeteer | Tools for Web Developers | Google Developers, 2021) Your Bibliography: Google Developers. 2021. Puppeteer | Tools for Web Developers | Google Developers. [online] Available at: <<https://developers.google.com/web/tools/puppeteer>> [Accessed 9 May 2021].
23. CHROME DEVTOOLS - CHROME DEVELOPERS In-text: (Chrome DevTools - Chrome Developers, 2021) Your Bibliography: Chrome Developers. 2021. Chrome DevTools - Chrome Developers. [online] Available at: <<https://developer.chrome.com/docs/devtools/>> [Accessed 9 May 2021].
24. GETTING STARTED – REACT In-text: (Getting Started – React, 2020) Your Bibliography: Reactjs.org. 2020. Getting Started – React. [online] Available at: <<https://reactjs.org/docs/getting-started.html>> [Accessed 22 December 2020].
25. BUILD A CHROME EXTENSION IN REACT In-text: (Build a Chrome Extension in React, 2020) Your Bibliography: Build a Chrome Extension in React. 2020

26. LIMPITSOUNI, K. AND GESOULIS, A. unDraw - Open source illustrations for any idea
In-text: (Limpitsouni and Gesoulis, 2021) Your Bibliography: Limpitsouni, K. and Gesoulis, A., 2021. unDraw - Open source illustrations for any idea. [online] Undraw.co. Available at: <<https://undraw.co/>> [Accessed 7 May 2021].
27. PEXELS — BEAUTIFUL FREE PHOTOS CONTRIBUTED BY OUR TALENTED COMMUNITY.
In-text: (Pexels — Beautiful free photos contributed by our talented community., 2021) Your Bibliography: Pexels. 2021. Pexels — Beautiful free photos contributed by our talented community. [online] Available at: <<https://www.pexels.com/search/videos/online%20shopping/>> [Accessed 7 May 2021].
28. LOADSTER: LOAD & STRESS TESTING FOR HIGH-PERFORMANCE WEBSITES In-text: (Loadster: Load & Stress Testing for High-Performance Websites, 2021) Your Bibliography: Loadster.app. 2021. Loadster: Load & Stress Testing for High-Performance Websites. [online] Available at: <<https://loadster.app/>> [Accessed 12 May 2021].
29. HEROKU In-text: (Heroku, 2021) Your Bibliography: Dashboard.heroku.com. 2021. Heroku. [online] Available at: <<https://dashboard.heroku.com/>> [Accessed 16 May 2021].

6.0 Appendices

6.1. Project Proposal

6.1.1. Objectives

The goal of the application is to make online shopping easier and less stressful for users. The web application will store the products that the user saves and notify the user through email when a certain product's price has been reduced. The web application will allow the user to organise saved items into different boards. The boards created by the user will also be shareable so that the user can share the saved items with friends and family when looking for shopping advice.

Once the google extension button is added to the user's browser, the user will be able to save items to specific wish boards by opening the google extension. When the extension button is clicked, I want it to display saving option to the user such as when to receive email notifications about price reductions (any reductions, 20%, 50% off), the board in which the product should be saved and other. The user will be allowed to save items from various sectors of commerce and not just retail or cosmetics. In the user's account, the web application will display a list of the wish boards created by the user and the items saved in each list to make everything easily accessible and more organised.

Ultimately the web app will aim to transform online shopping into a more enjoyable experience while enabling the user to make educated choices when purchasing items by sending notifications when the item is on sale and so giving the user time to decide if the item is truly a necessity. Such an app can be a useful tool for people especially now when everyone is more cautious when placing orders online. The website could save users a lot of time, money and reduce buyer's remorse resulting in more informed and planned purchases.

6.1.2. Background

I first thought of the project idea at the start of this year, after the lockdown began and people became more dependent on online shopping. There are so many different online stores, and each store has such a large number of products available for users to choose from so deciding on an item over the other available options can be tricky and time consuming. I thought that a web app that can categorise items into separate sections and track price changes can be a very useful tool for many people who want to save money or cannot afford certain items or are not completely sure if the item is a necessity. This app can give the user more time to decide on whether or not the purchase is required. With all the changes that we went through this year, many people were left with less funding for items that are not essential, this web app can monitor price changes and notify the user when price changes occur and help the user save money when shopping online.

While I was researching my idea, I came across two similar apps, but I found that both were lacking essential features. The first web app I encountered is called +Wishlist and it allows users to save items into their specific account for later. The user can save items from various online stores, but the user cannot create wish boards for the saved items to organise shopping. Also, when an item is saved using the extension icon, the user does not get the option to select when to receive notifications about the item, nor track price changes. The second web app I found is called Shoptagr, it allows users to monitor price changes and to save items into different categories, but it only accepts products from retail and cosmetics sectors to be saved into the user's account. The user is also not given the choice to share the created boards with other users or via social media platforms. The web app I intend on creating will contain the features that are most useful to the user when planning online shopping. Each of the other two web extensions lack some of the essential features that could improve the user's experience while shopping.

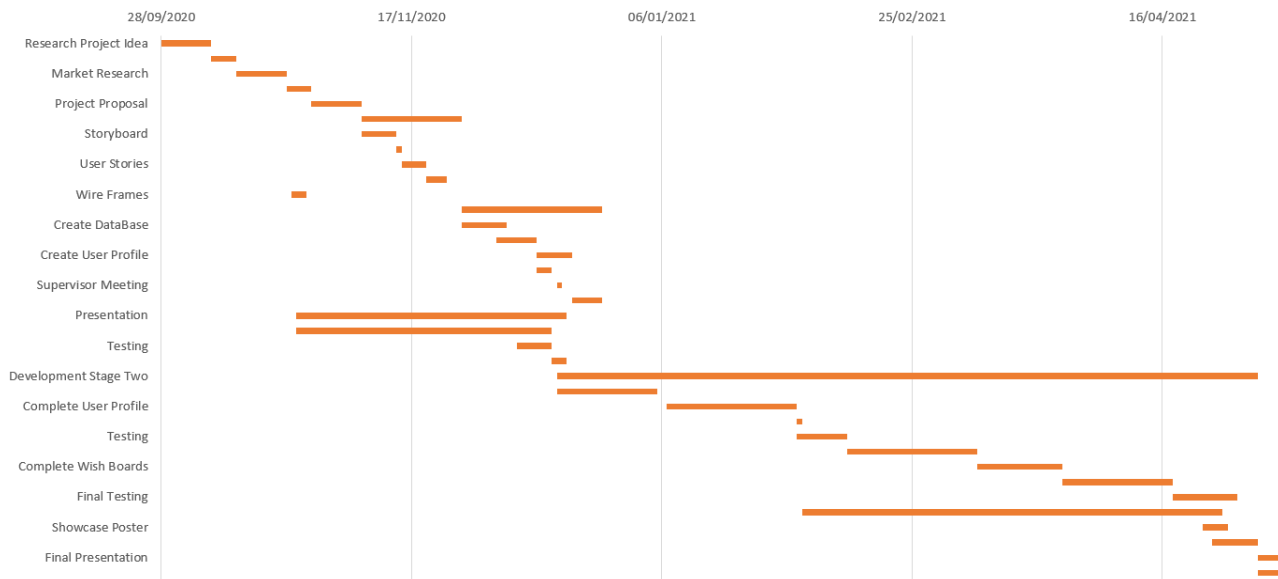
6.1.3. Technical Approach

When a user first creates an account, the option of creating an account using Gmail or a different email will be available to the user to increase user accessibility. Once the user's account is created and the extension is added to the browser, the user can start saving items and create wish boards. I want the user interface to appear clean and easy to interact with, and to only display essential and relevant information to not intimidate users with little technology or online shopping knowledge.

The more complex features of the web application should be saving items to the profile using the extension icon and viewing the created wish boards, by focusing on the main functionality of the application and making it easy to interact with, I aim to get people from different age groups and backgrounds to be interested in using the extension as people with less technical knowledge tend to be scared by such web resources.

6.1.4. Project Plan

Research Project Idea	10	28/09/2020	08/10/2020
Project Pitch Video	5	08/10/2020	13/10/2020
Market Research	10	13/10/2020	23/10/2020
Reflective Journal October	5	23/10/2020	28/10/2020
Project Proposal	10	28/10/2020	07/11/2020
Planning Stage	20	07/11/2020	27/11/2020
Storyboard	7	07/11/2020	14/11/2020
Supervisor Meeting	1	14/11/2020	15/11/2020
User Stories	5	15/11/2020	20/11/2020
Use Cases	4	20/11/2020	24/11/2020
Wire Frames	3	24/10/2020	27/11/2020
Development Stage	28	27/11/2020	25/12/2020
Create DataBase	9	27/11/2020	05/12/2020
Create User Login	8	04/12/2020	12/12/2020
Create User Profile	7	12/12/2020	19/12/2020
Research Google Extension	3	12/12/2020	15/12/2020
Supervisor Meeting	1	16/12/2020	17/12/2020
Requirements Specification	6	19/12/2020	25/12/2020
Presentation	54	25/10/2020	18/12/2020
Prototype	51	25/10/2020	15/12/2020
Testing	7	08/12/2020	15/12/2020
Midpoint Presentation	3	15/12/2020	18/12/2020
Development Stage Two	140	16/12/2020	05/05/2021
Integrate Extension	20	16/12/2020	07/01/2021
Complete User Profile	26	07/01/2021	02/02/2021
Supervisor Meeting	1	02/02/2021	03/02/2021
Testing	10	02/02/2021	12/02/2021
Complete User Interface	26	12/02/2021	10/03/2021
Complete Wish Boards	17	10/03/2021	27/03/2021
Debugging	22	27/03/2021	18/04/2021
Final Testing	13	18/04/2021	01/05/2021
Documentation	84	03/02/2021	28/04/2021
Showcase Poster	5	24/04/2021	29/04/2021
Showcase	9	26/04/2021	05/05/2021
Final Presentation	5	05/05/2021	10/05/2021
Presentation	5	05/05/2021	10/05/2021



6.1.5. Technical Details

The web application will be developed using Visual Studio Code and I will use JavaScript with emphasis on the React.js library, JSX, JSON for the creation of the google extension, HTML and CSS for styling.

Google Firebase will handle all the data storage required by the web application. For development, monitoring purposes, version control and bug tracking I will use GIT and GitHub.

6.1.6. Evaluation

Unit Testing: During the development cycle of the web application, I plan on conducting unit testing to ensure that every new feature added performs as expected and no bugs emerged from the changes made.

Quality Assurance: Once all of the backlog tasks are implemented or the last 3 weeks of the project plan are reached, I will stop adding or improving features and I will start working towards fixing any of the remaining bugs to optimise any outstanding errors before the final submission.

6.2. Reflective Journals

6.2.1. October Reflective Journal

This month I mainly worked on conducting research for the software project. For the first part of the month, I worked on researching a suitable and feasible idea to present in the pitch video. I aimed to find an idea that will solve a problem or provide a useful service for people in these changing times. After examining all the ideas, I had I decided to go forward with a Shopping Wishlist web application/google extension. I conducted further research on what the features that the application should contain and some additional tools and languages that could be useful for developing the project in order to present a good explanation of the idea in the pitch video.

I also started planning for the project proposal and conducted some market research to identify competitors and the usefulness of the application to users by communicating with friends and relatives to determine if they would be interested in using such a tool over other available resource.

The research conducted for the pitch video and project proposal helped me identify new features for my project and areas that need more planning before I start development.

Before I start creating the project proposal, I will need to meet with my supervisor to discuss the project idea present in the pitch video and consider suggestions that could improve the project's potential. I will conduct further research to identify all the possible competitors and ensure that the application provides the most demanded features.

I am also planning on further investigating the tools and technologies required for creating a high-level product that will be useful for users and make shopping easier.

6.2.2. November Reflective Journal

During the first part of this month, I mainly worked on writing the Project Proposal report for the final project. For the Project Proposal I had to identify the main features of the project, state the technical details and the tools to be used for the development of the project, and find other web resources that provide similar services and describe the differences between my project idea and other possible competitors. As part of the Proposal, the project features were categorised based on the features that will provide the main functionality of the web app and the ones that will be secondary features, each being described in the report. A Gantt Chart displaying the plan for the development of the project was also created, mapping out the tasks to be performed and the predicted time duration for each section.

This month I mostly worked on getting a good understanding of how React JS works and the basics of it. I started some React courses and followed online tutorials which helped me get a good understanding of the basic functionality of the library and main features. I started a beginner React course on Egg Head which covers various elements and components of React. I also found a good two-part course on Code Academy which explains React components and elements in more depth. The React tutorials followed this month served to provide additional insights into how to correctly write code using the library.

This month I also started to create wireframes for the user interface of the project. Having wireframes ready when starting development can be very useful as they provide a guide for what the product should look like, eliminating confusion and last-minute decisions. I also further researched google extensions to understand how it should be implemented as I have never worked with such technologies before.

6.2.3. December Reflective Journal

In the month of December, I had two meetings with my supervisor to discuss the deliverables for the mid-point submission. During the first meeting we talked about what needs to be done for each section of the Technical Report and prototype. In the second meeting, on the 18th we discussed the progress made on the report and the last minute changes that had to be done before the submission for the Technical Report and prototype.

This month I also worked on getting the prototype ready for the mid-point presentation. The browser extension, the login page of the website and the Google Firebase database were implemented to show how the final product will work. The browser extension was added on my personal Chrome store for testing purposes and is yet to be published. The login page of the website was built using React JS and is connected with the Firebase. Throughout this month, a significant amount of work went into completing the required sections of the Technical Report document.

6.2.4. January Reflective Journal

During the month of January, I worked on completing the Showcase Profile template. I added in the relevant information required for each section of the template so that it can be further reviewed. This month I also worked on the browser extension for my project. I added more functionality and improved the layout to make it more user friendly and interactive. A colour story for the browser extension and website was selected to ensure consistency across the project. A group of three main colours were selected consisting of a deep shade, a pale shade, and a medium vibrancy colour. Additional colours might be used in the future to suit further needs.

A logo and icon for the website and browser extension was designed and will be used in the creation of the project. The logo follows the same colour story selected for the extension and website to maintain consistency throughout the project. In addition to the existent signup form, the login page of the website was created and linked with the Firebase database.

6.2.5. February Reflective Journal

In the month of February, I conducted research on various methods of saving URL's of a webpage using a browser extension. I found some good leads and papers that can prove useful when that part of the project is implemented. Based on the current progress, I will probably start working on incorporating this part of the project's functionality in the next month.

During this month, I was able to make some progress with the React website that I am building as part of the project. Since authentication through Firebase was completed in the previous month, this month I decided to work on creating the homepage and wish board pages of the website. I also created some base pages for most of the features of the project.

This month I also completed some additional React and JavaScript tutorials to understand the more complex aspects of the languages better. Having a better understanding of these components will make future development easier as I am diving into the higher complexity traits of the project.

6.2.6. March Reflective Journal

In the month of March, I mainly focused on adding more functionality to the project. I improved the performance and layout of the extension by starting to work on implementing the URL scraping of other websites and I will continue working on this in the coming weeks. This month I also worked on further testing the project components that have been integrated recently. I carried out a number of unit tests and integration tests to validate the project and catch bugs within the system.

To enhance the appearance of the website, decided to create a landing page for unsubscribed users to provide some insight into the services and features provided by the project. This section of the website was time consuming as I wanted use React packages such as react-scroll for a smooth scroll effect on the page, the styled-components package for customising components, and also react-icons and react-route-dom for better visualisation and navigation. This month I also worked on creating the various navigation bars required for the project such as the navbar for unsubscribed users and the navbar for current users, toggle bars to make the website responsive on any screen size, and side bars for wish boards and settings.

I also did some work on the final report based on the suggestions and feedback received during the last supervisor meeting. I expanded on the Testing and Implementation sections, but I will add further details to both sections once more functionality is added in, and more tests are ran.

6.2.7. April Reflective Journal

In the past month I worked on integrating some of the features that were left to be added to the project. For the first two to three weeks of this month I mainly worked on finishing assignments for other modules, and I stopped making improvements to my final project until those assessments were complete, this way I was able to complete work faster.

During the rest of the month, I worked on improving the functionality of the user profile section and made some modifications to the form used when updating the user profile. A settings page was also created to allow users to easily find the support service and answers for other inquires. Layout updates were done on the home page which now displays suggested shopping websites with direct links to those websites. Wish boards are now being stored in the firestore database and using reusable clickable containers the wish boards are mapped on the page, this way the user can view all the boards and select the desired one to view. Users can also edit and delete boards from firestore using the buttons inserted on each wish board page. The side menu also shows the wish boards stored by the user on firestore and the desired wish board can also be selected from there now. Reusable components were also created for displaying the items that are saved to the wish board. Similarly, these components will be used for mapping the data stored by the user on firestore and display the data in a clean layout.

Options for social authentication through Facebook and Google were added to the sign up and login forms for users that do not wish to create an account with for the system. This is done by displaying popup pages for users to enter their social account details.

Unit tests and integration tests were created for the new updates made to the system to identify issues within the system. Some additional work was also done on the final report to include the recent updates and new features.