Software Requirements Specification

for

*SuperPrice*

Version 2.0

Prepared by P04-02

Software Engineering: Processes & Tools

06/08/2023

Table of Contents

Table of Contents ii

Version History ii

1. Introduction 1

1.1 Overview 1

1.2 Purpose 1

1.3 Stakeholders 1

2. Features & Requirements 1

2.1 Functional Requirements 1

2.2 Non-Functional requirements 3

3. Use Cases 3

4. System Architecture 7

5. Data Model 8

6. UI Design 9

7. Assumptions & Constraints 14

8. Dependencies 14

9. Testing & Acceptance Criteria 14

Appendix A: Glossary 17

Appendix B: M1 Additional Documentation 17

Sprint 1 Planning Notes 17

Sprint 2 Planning Notes 17

Sprint 0 Retro 18

Appendix C: M2 Additional Documentation 19

Continuous Integration Docs 19

Testing 19

Revision Summary 19

Sprint 2 Planning Notes (Sprint 1 Version) 19

Sprint 1 Retro 20

Version History

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Date** | **Description/Changes** | **Version** |
| Milestone 1 | 20.08.2023 | Initial Version | 1.0 |
| Milestone 2 | 17.09.2023 | Sprint 1 Version (for changes see m2 docs) | 2.0 |

# Introduction

## Overview

SuperPrice is a website focused on comparing prices on a product offered at several stores and finding the best price possible for a customer. It also allows users to manage deliveries of products they purchase through the website. Retailers will be able to affiliate themselves with the application by paying a commission fee.

## Purpose

The purpose of the product is to find the best deals for customers, and make managing the deliveries of those purchases easier. Retailers will also have the opportunity to have their products promoted to customers, increasing business.

## Stakeholders

The stakeholders of this project include the product owner, who will benefit from the successful launch of the product. The customers, who will save money on their purchases. The developers who are responsible for the production of the application. And participating retailers, who will have the opportunity to promote their business through the application.

# Features & Requirements

## Functional Requirements

### Product Search & Categorisation

Users will be able to search the catalogue of products. They can either search for a specific product, or browse by category.

To search for a specific product, the user can type the name of the product into the search bar, click the search button, and any matching results from the database will be listed. To browse by category, the user will be presented with a selection of categories to choose from. After clicking on a category, they will be presented with a list of products under that category.

After being presented with the search results, they can also click on a specific product to see a focused view of the product which will include a more detailed overview of the product.

**REQ-1:** The website’s homepage will display a search bar, in which a user can type the name of a product they are searching for into. Open submitting their query, products matching the search will be listed.

**REQ-2:** The website will present a list of available product categories, which the user can select one of at a time. Once they have selected one, only products under that category will be presented to the user.

**REQ-3:** In the case of a search yielding no results. The user will be notified that the search produced no results.

### Price Comparison

To help the user find the best deal, the user will always be presented with the best price offering first. When presented with search results, the best price of that product and the associated retailer will be shown. To view more prices, a user can click on the product search result, taking them to a focused view of the product. Here, all prices and associated retailers for that product will be shown in ascending order.

**REQ-1:** In the search results listings. The lowest price for each product listing will be shown along with associated retailer.

**REQ-2:** In the focused view of a product. All the available prices for that product will be displayed in ascending order. The retailers offering each price will also be shown alongside.

**REQ-3:** In the case that multiple prices are equal, the listings will be prioritized by rating, then alphabetically by retailer names.

### Delivery Organization

The user will have the ability to add products to their cart and checkout through a standard online shopping experience. During the delivery selection phase, the user will be presented with available delivery options offered, and the available time slots to be delivered in.

**REQ-1:** During checkout, the user will be presented with the available delivery options to choose from. The user can select either same day delivery, or pick any future date to deliver on.

**REQ-2:** After selecting a delivery day, the user can also choose a particular time-frame for their delivery. This includes morning, afternoon, and evening options.

### Notifications & Alerts

On the website, users can opt-in to a mailing list by providing their email address. They will then be notified of any offers through an email to the provided address.

**REQ-1:** On the footer of the website, there will be an option to sign up to the mailing list. The user is prompted to input their email address, and clicking the submit button will add their email to the mailing list.

### User Reviews & Ratings

The user will have the ability to place reviews and ratings on products they have previously purchased through the application. Any reviews and ratings stored in the database will also be shown in the focused product view pages.

**REQ-1:** From their account page, the user will be presented with an option to review products. On clicking this, they will be presented with a list of previously purchased products. From here, they can navigate focused view page for that product.

**REQ-2:** On the focused view of a product, there will be a prompt to make a review for that product from that page. If they have previously purchased the product, they will be able to leave a review. Otherwise, they will be notified that they cannot leave a review if they have not purchased it before.

**REQ-3:** On the focused view of a product, the average rating and a list of all reviews, with their individual ratings will be shown.

## Non-Functional requirements

Usability

The product will have a user-friendly and intuitive interface. This will make it accessible to a wide range of users. The website will follow conventional web design patterns that will be familiar to the user. *See UI Designs for more details on the design of the application.*

Performance

The search function is expected to produce results in an acceptable time frame.

Security

The products search feature should prevent SQL injection.

# Use Cases

**1 Product Search (Search Bar)**

**As a** frequent shopper, **I want** to be able to look up specific products, **so that** I can quickly find the items I need.

Tasks

* Design the layout for homepage to include a prominent search bar **(2 hours)**
* Implement the searchbar and search results using HTML and CSS **(3 hours)**
* Set up Javascript functions to handle user input and trigger search queries from backend **(3 hours)**
* Write SQL queries for search function **(3 hours)**
* Design the database to store product information **(3 hours)**
* Implement Tests for search result listings **(1 hours)**

*User Story 2 has been removed*

**3 Delivery Options**

**As a** busy individual, **I want** to have multiple delivery options, including to be able to set delivery times, **so that** I can easily organise deliveries to my house and be efficient to save time.

Tasks

* Design the layout for the ordering page with delivery options. **(2 hours)**
* Implement the ordering page and delivery option layout using HTML and CSS **(3 hours)**
* Set up a form for users to select delivery option. **(1 hour)**
* Write SQL queries to get details for different delivery options such as price, time estimate etc. **(3 hours)**
* Implement Tests for delivery option retrieval **(1 hours)**

**4 Product Price Comparison**

**As a** time constrained shopper, **I want** to see the see the prices offered by other shops, **so that** I can compare products and choose the offer that’s right for me.

Tasks

* Design the layout for comparing prices for a product between different brands. **(2 hours)**
* Implement the comparison page layout using HTML and CSS **(3 hours)**
* Set up Javascript functions to sort items that are being compared to. **(2 hours)**
* Write SQL queries to get items from different stores that are similar to a given item. **(2 hours)**
* Implement Tests for similar item retrieval **(1 hours)**

**5 Display Reviews**

**As a** mother of 4 children, **I want** to buy reliable and best quality products for my children by looking at reviews for an item, **so that** I can decide if the item will be safe to use on my children.

Tasks

* Design a review section for the product page. **(1 hour)**
* Set up Javascript functions to display reviews. **(2 hours)**
* Review Write SQL queries to retrieve reviews. **(2 hours)**
* Implement Tests for review retrieval **(1 hours)**

**6 Receive Deal Notifications**

**As a** savvy shopper always on the lookout for special offers, **I want** to receive timely notifications and alerts about price drops and exclusive promotions, **so that** I never miss out on great deals and discounts.

Tasks

* Design UI to include newsletter prompt **(1 hour)**
* Implement the newsletter prompt in front end **(2 hours)**
* Add JS functions to save an input email to the database **(3 hours)**
* Implement test for saving input email **(1 hours)**

**7 Leave Reviews**

**As** an avid shopper, **I want** to leave reviews on products I have purchased, **so that** other shoppers can make informed decisions about these products.

Tasks

* Design a leave review form on product page. **(1 hour)**
* Implement the review form in front end **(2 hours)**
* Set up Javascript functions to save user reviews. **(2 hours)**
* Write SQL queries to save reviews. **(2 hours)**
* Implement Test for successful review creation in database **(1 hours)**

**8 Sign in Page**

**As a** frequent shopper, **I want** to be able to sign in to my account, **so that** I can return to my account at a later date.

Tasks

* Design the Sign in Page **(1 hour)**
* Implement the sign in page on front end **(3 hours)**
* Connect front end with back end api to confirm/allow login **(3 hours)**
* Create API methods to allow account login **(4 hours)**
* Implement tests for sign in validation **(1 hours)**

**9 Account Registration**

**As** **a** frequent shopper, **I want** to create an account, **so that** I can manage my purchases.

Tasks

* Design the Account Registration Page **(1 hour)**
* Implement the registration page on front end **(3 hours)**
* Create API methods to allow for account creation **(3 hours)**
* Connect front end with back end api to create account **(3 hours)**
* Implement tests for account creation **(1 hours)**

**10 Remember Login**

**As a** frequent shopper, **I want** the application to remember my login details, **so that** I don’t have to login every time.

Tasks

* Implement the remember login checkbox on front end **(1 hour)**
* Create method in in backend to create authorisation token **(3 hour)**
* Connect front end with backend to allow for login remembering **(2 hours)**
* Implement Tests for login token validation **(1 hours)**

**11 Product Search (By Category)**

**As a** frequent shopper, **I want** to browse products by categories, **so that** I can find products that suit my needs.

Tasks

* Design Homepage to display categories to select **(1 hour)**
* Implement category display on homepage **(3 hour)**
* Connect front end with backend api **(3 hours)**
* Write sql queries for retrieving categories **(2 hours)**
* Implement Tests for each product category **(1 hours)**

**12 Search Result Sorting**

**As a** user, **I want** to sort any search results, **so that** I find the product I am look through search results methodically.

Tasks

* Implement Search Result Sorting Selection on Front End **(1 hour)**
* Write sql queries to sort results by selected sort function. **(1 hours)**
* Connect front end with backend api **(3 hours)**
* Implement Tests for each sorting type **(1 hours)**

**13** **Search Result Filtering**

**As a** user, **I want** to filter any search results, **so that** I only see results relevant to me.

Tasks

* Implement Search Result Filtering Selection on Front End **(1 hour)**
* Write sql queries to sort results by selected Filter function. **(1 hours)**
* Connect front end with backend api **(3 hours)**
* Implement Tests for each filter option **(2 hours)**

**14 Products on Special**

**As a** user, **I want** to browse products that are on special, **so that** I save money on my shopping.

Tasks

* Implement “Special” Tab and Row on Front end Homepage **(1 hour)**
* Write sql queries to retrieve products on special. **(1 hours)**
* Connect front end with backend api **(3 hours)**
* Implement Tests retrieving products on special **(1 hours)**

**15 Trending Products**

**As a** user, **I want** to browse products that a bought most frequently, **so that** I find the most common items on offer.

Tasks

* Implement “Trending” Tab and Row on Front End Homepage **(1 hour)**
* Write sql queries to retrieve products which have been purchased the most. **(3 hours)**
* Connect front end with backend api **(3 hours)**
* Implement Tests for trending product calculation **(2 hours)**

# A diagram of a company Description automatically generatedSystem Architecture

**Presentation Layer:**

* The presentation layer is responsible for displaying a user interface for the end user to interact with the website.
* The data presented by the presentation layer will be retrieved through the application layer.
* The technology chosen for this layer is NextJS and is used over ReactJS as NextJS is used to create web applications through server-side rendering, overall improving the performance of the website on the end-user side.

**Application Layer:**

* The application layer is responsible for providing logic and interactions between the presentation layer and the data layer.
* The layer consists of an orchestrator microservice which interacts with the other microservices attached. The individual microservices will handle its own requests from the orchestrator service, and requests to the server from presentation layer are handled by the orchestrator service. Each individual microservice attached to the orchestrator service will interact with the data layer, requesting and receiving products. Furthermore, in the event that there are large loads of requests, the load balancer can create more instances and balance the requests.
* The technology chosen for this layer is Spring Boot. The choice of Spring over other Java APIs, for example Javalin, is that Spring Boot makes it easy to create stand-alone, production-grade APIs that can be easily run.

**Data Layer:**

* The data layer is responsible for providing a gateway for the application layer to request and retrieve product data.
* Each microservice will use its own database through the data layer and will interact with the corresponding database to retrieve the requested data. This data is sent back to the respective microservice in the application layer
* The technology used for this layer is SQLite, as it allows for a lighter application without requiring an external service to run. SQLite directly stores data into a singular file, making it easy to copy and backup. Furthermore, considering that the data stored is very basic, it does not require a more complicated database like MySQL, and does not require any complex configurations.

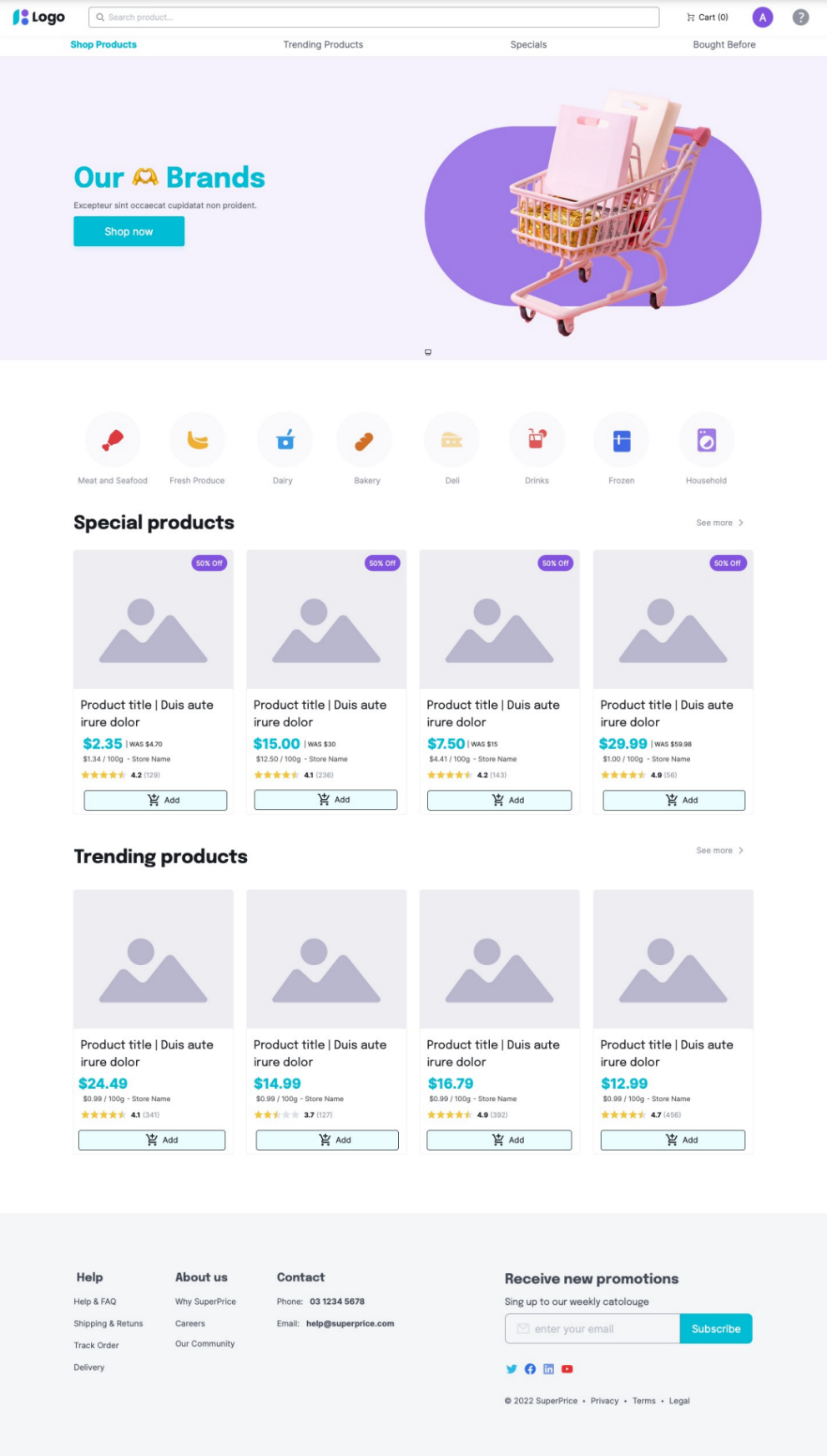
# Data Model

A computer screen shot of a diagram

Description automatically generated

# UI Design

**Home/Landing**

****

**Search Results**

**A screenshot of a website

Description automatically generated**

**Individual Product**

**A screenshot of a website

Description automatically generated**

**Checkout**

**A screenshot of a website

Description automatically generated**

**A screenshot of a login page

Description automatically generatedSign-in**

**Review**

**A screenshot of a review

Description automatically generated**

# Assumptions & Constraints

We are creating the website in English only.

We are creating our own products and prices for demonstration purposes because we don’t have access to supermarket databases.

We are using Typescript, Javascript for the front-end programming, Java and SQL for back-end programming.

We are not using HTTPS as we do not own a server for hosting the website and thus cannot get a SSL certificate.

# Dependencies

The product depends on the following external systems, libraries, and/or API’s.

* SpringBoot
* React App
* SQLite

# Testing & Acceptance Criteria

1 Product Search (Search Bar)

**Given:** The user is in the applications main page.

**When:** The user enters a product name, brand, or keyword in the search bar and clicks the search button.

**Then:** The application should display relevant search results.

*User Story 2 has been removed*

3 Delivery Options

**Given:** The user is in the ordering page of a product.

**When:** The user enters a specific date and time for the order to be delivered.

**Then:** The application should display an order placed confirmation

4 Product Price Comparison

**Given:** A product has multiple price listings.

**When:** The user clicks on to the specific page of a product

**Then:** The application should display the similar products from supermarkets with their prices.

5 Display Reviews

**Given:** A product has reviews **and** the user is on the specific page of a product.

**When:** The user clicks on the reviews tab of the page.

**Then:** Then reviews for the item with a rating summary will be displayed.

6 Receive Deal Notifications

**Given**: The user is not already subscribed to the mailing list

**When**: The user provides their email address to the email prompt

**Then**: They will be subscribed to the promotional newsletter **and** an email will be sent out when new promotions are available.

7 Leave Reviews

**Given**: The user has purchased an item previously **and** is logged in

**When**: There are on the page for an item

**Then**: They can click on the review button to leave a star and written review.

**Given**: The user has not purchased an item previously **or** is not logged in

**When**: There click on the review button

**Then**: The page will show a pop up explaining that they cannot leave a review.

8 Sign in Page

**Given**: The user has an account in the database

**When**: When the user inputs their correct details

**Then**: They will be logged in to their account **and** they will have access to their profile page

9 Account Registration

**Given**: The user does not have an account in the database

**When**: When the user inputs their details

**Then**: an account will be created with their details.

10 Remember Login

**Given**: The user checks the “remember me” box in the login page

**When**: The user logs in

**Then**: They will be logged in for an extended period of time.

11 Product Search (By Category)

**Given**: The user is on the homepage

**When**: The user selects a particular category

**Then**: The application will load results for products in that category only

12 Search Result Sorting

**Given:** The user is on a search result page

**When:** The user selects a sorting type

**Then:** The application will display results in the sorting order they have selected.

13 Search Result Filtering

**Given:** The user is on a search result page

**When:** The user selects a filtering type

**Then:** The application will only display items which are applicable to that filter.

14 Products on Special

**Given:** The user is on the Homepage

**When:** The user clicks on the “specials” tab

**Then:** The application will display search results with items that are on special.

**Given:** The user is on the Homepage

**When:** The user clicks on “see more” beside the specials row

**Then:** The application will display search results with items that are on special.

15 Trending Products

**Given:** The user is on the Homepage

**When:** The user clicks on the “trending” tab

**Then:** The application will display search results with items that have been purchased most.

**Given:** The user is on the Homepage

**When:** The user clicks on “see more” beside the trending row

**Then:** The application will display search results with items that have been purchased most.

Appendix A: Glossary

|  |  |
| --- | --- |
| **Term** | **Definition** |
| MVP | Minimum Viable Product |
| ERD | Entity-Relationship Diagram |

Appendix B: M1 Additional Documentation

## Sprint 1 Planning Notes

Duration: 3 Weeks

Goal: The aim of Sprint 1 is to create a minimum viable product.

What is the team’s vision for the sprint?

We have mainly committed the items from the product backlog that focus on producing a functional product by the end of the sprint. With an MVP, we will be able to get feedback on the functionality of the product. In its working form, the MVP will have all the main features listed in section 4.

Estimation

We estimate that the tasks for sprint 1, as we are only creating an MVP will require less effort than sprint 2.

## Sprint 2 Planning Notes

Duration: 3 Weeks

Goal: The aim of Sprint 2 is to turn the MVP into a finished product.

What is the team’s vision for the sprint?

We want to create a viable product that has the full functionality to successfully operate and satisfy the requirements of the product owner. If full functionality cannot be achieved, then we will try to complete the required requirements so that the program can be operational, possibly sacrificing the implementations of some non-functional requirements.

Estimation

We estimate that sprint 2 will be where most of our effort goes into. Polishing the website to a final product will require more effort than the MVP. We may also run into problems in sprint 1 that will need to be addressed in sprint 2, which will require more effort.

## Sprint 0 Retro

Things That Went Well

* + The team collaboration, everyone did their tasks on time and was able to join weekly meetings to keep everyone updated.
  + Everyone was happy with what we achieved, and we had no arguments about who did more. Work was fairly distributed.

Things That Could Have Gone Better

* The team could improve on keeping up to date with requirements.

Things That Surprised Us

* + The amount of documentation that is needed.
  + How we jumped straight into the sprint. Had to hit the ground running.

Lessons Learned

* + Scrum Standups require every individual to speak individually rather than as groups.

Final Thoughts

Things to keep

* + - Collaboration and fair work distribution, as well as keeping each other up to date.

Things to change

* + - Keeping more up to date with the requirements
    - Making sure that everyone is individually contributing to each SCRUM meeting

# Appendix C: M2 Additional Documentation

## Continuous Integration Docs

The GitHub Actions Continuous Integration (CI) configuration will trigger on push to the main branch, it will execute the three defined jobs on the latest version of ubuntu. The defined jobs include:

- Builds and tests orchestrater service

- Builds and tests customer-service

A screen shot of a computer

Description automatically generated- Builds and tests product-service

A screenshot of a computer

Description automatically generated

In the screenshot above: This build failed because the customer service folder did not have the maven wrapper files.

A screenshot of a computer

Description automatically generated

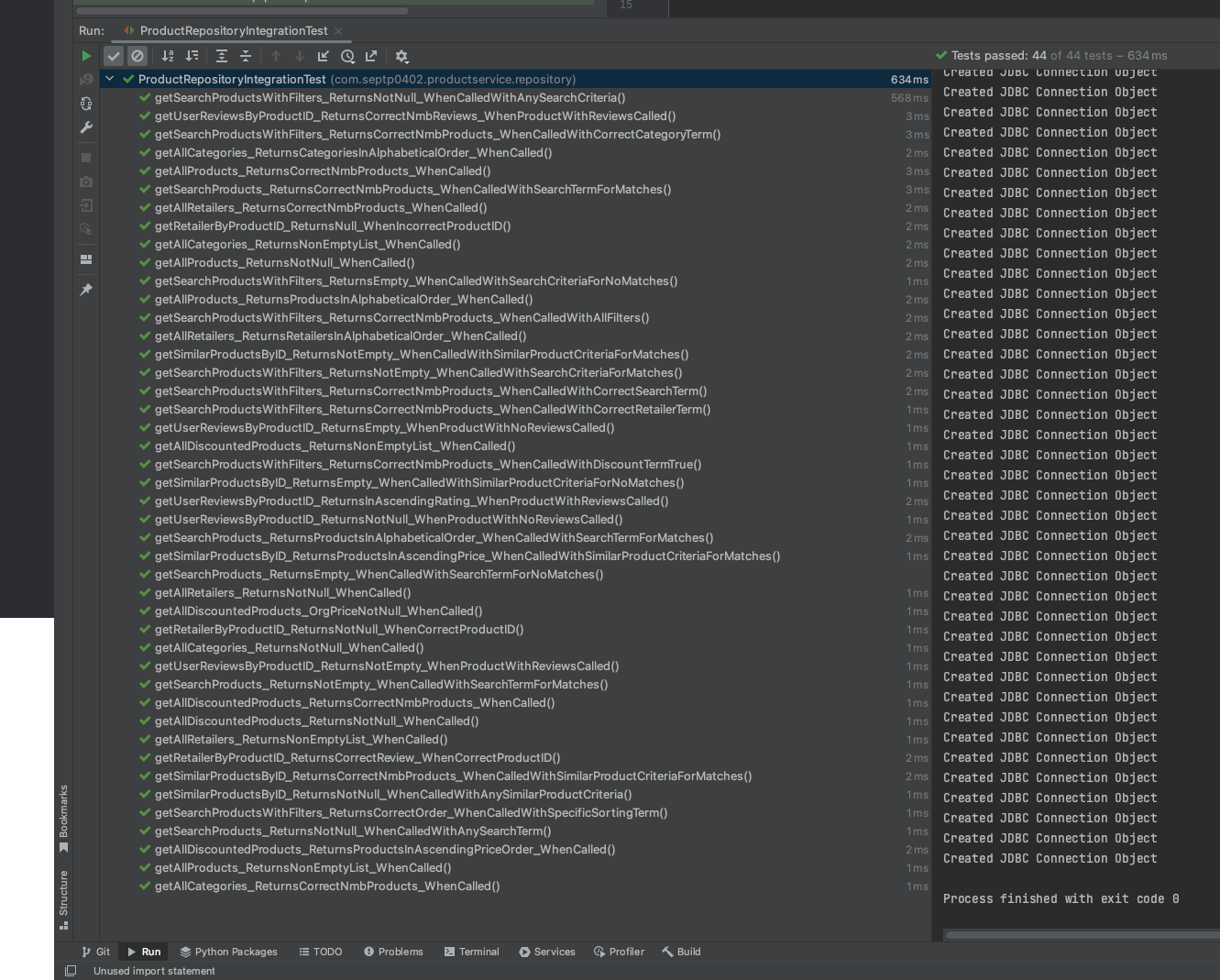
In the screenshot above: Once the wrapper files were added, then the build passed

## Testing

For testing, we created integration and unit tests for our product microservice. We created several tests for each method to ensure we covered basic and edge cases. These tests are separated into three files. ProductRepositoryIntegrationTest, tests the integration between the layer that’s closest to the database and the database. ProductControllerUnitTest conducts unit tests on the controller layer of the product service. It ensures the functionality of the controller layer is behaving appropriately. ProductServiceApplicationIntegrationTests performs full product microservice. It makes sure everything is working together from the controller layer to the repository layer, to the database.

**Test Results**

ProductRepositoryIntegrationTest Results



ProductServiceApplicationIntegrationTest Results

A screenshot of a computer

Description automatically generated

ProductControllerUnitTest Results

A screenshot of a computer

Description automatically generated

## Revisions Summary

Product Backlog

We moved the newsletter notifications task to sprint 2 as they aren’t a priority for the MVP. Instead, we chose to focus on implementing the search, search results, and product pages.

User Stories

Based on feedback, we broke down our user stories into smaller user stories. One example is User Story 5, which contained both displaying and leaving reviews. So, we split that into 2 separate user stories that focused on one function each. We also revised the prioritization of our user stories in the product backlog to reflect their priority more accurately. We also updated our GH project to match up with the rest of our documentation.

We removed User Story 2 -- Displaying only the lowest priced product in search results – because it limited the options a customer had to find the product they were looking for. And because the ability to compare prices of products was already covered in User Story 4, there was no need for US2. This removal also addresses feedback about user stories overlapping.

Finally, we added several new user stories to address any functionality that was present in our wireframes but hadn’t been covered by other forms of documentation.

This includes User Stories 8-15

Software/Architecture Design (Design Justifications)

Based on feedback, we revised our architecture diagram to show scalability, through the addition of microservices, and performance, with the inclusion of a load balancer.

We have also since designed and included an ERD that describes the design of our database.

Other revisions to our software design include:

* + Changing our front end from Nextjs to Reactjs. This is because our team is more familiar with implementing front end features in React than Next.

New designs explored within our implementation:

* Following our architecture design, in an implementation we follow a Microservices approach.
  + Functionalities are separated into services. Hence tasks associated with products will be in the product microservice, tasks associated with customers are in the customer microservice, tasks associated with orders are in the order microservice and so on. Like this we will have corresponding microservices for the required tasks.
  + There will be another microservice, which is the orchestrator service, also known as the back-end for front-end (bff) service. This will act as a bridge between the front end and the various microservices.
  + As of now, only the orchestrator service, product service and a portion of the customer service is implemented.
  + Rationale and Benefits:
    - **Decoupling -** By ensuring that the components will have clear separation of concerns, this allows each service to evolve independently and aids in faster development and deployment cycles.
    - **Scalability -** Each microservice can be scaled independently based on its demand and requirements.
    - **Resilience -** Failures in one service do not necessarily cause system-wide outages. Each service can have its own fault tolerance mechanisms. Meaning if the order service is down, users can not order a product, but they can still view and compare them.
    - **Unified Entry Point -** The front end will have a single endpoint for communication, simplifying the front–end development process.
    - **Optimized Data Transfer -** Due to the bff service containing business logic, it can collect and aggregate data from multiple services. This allows for less complex implementations for front end where only one call needs to be made for a specific requirement, and the bff service will collate all necessary data for various microservices and return it as one response.
    - **Enhanced Performance -** As a result of the optimised data transfer, we get a reduced number of trips from the frontend to various microservices by handling the orchestration at the backend.
* Each of our micro-services exposes RESTful APIs which are consumed by the orchestrator service. The orchestrator service then extends those RESTful APIs for the front-end to access.
  + Product service will have endpoints like ‘/product’ and ‘/product/search’.
  + Customer service will have endpoints like ‘/customer/registration’ and ‘/customer/login’.
  + Like this we will expose endpoints to accomplish different tasks
  + Rationale and Benefits:
    - **Standardized Communication –** By utilising REST conventions for CRUD operations, it allows for easier understanding and implementation.
    - **Statelessness -** Every request from the client to the server contains all the information needed to understand and process the request.
    - **Scalability and Performance -** As REST is stateless, this allows for easier scalability as there is no session-related dependency.

Wireframes

We also made the following revisions to remove unnecessary features that were not in line with the requirements of this project:

* + Removed Recipes section from homepage.
  + Removed “heart” saving feature.
  + Changed Product Images to a single Image.
  + Removed Share Product feature.
  + Removed size option from checkout.
  + Removed voucher feature from checkout.
  + Removed nickname and email input on review form.

Testing

Based on Feedback, we added more details of our testing strategy. This can be found in the Testing section above. We also revised our acceptance criteria to be less vague and align with our wireframes.

## Burndown Chart

For input values. See Product Backlog Spreadsheet: Burndown Chart Page

## Sprint 2 Planning Notes (Revised)

Duration: 3 Weeks

Goal: The aim of Sprint 2 is to turn the MVP into a finished product.

What is the team’s vision for the sprint?

In sprint 1, we were able to implement the basic functionalities that allowed a user to find and compare product prices, which allowed them to save money on their shopping. We were also able to get started on some additional functions initially planned for sprint 2.

In sprint 2, we want to extend our application’s functionality to include the rest of the features outlined in the project specifications. This includes the shopping checkout, deal notifications, and reviews. By the end of the sprint, we want to have a polished product that satisfies the requirements of the product owner.

Estimation

We estimate that sprint 2 will require similar effort to sprint 1. We have laid a lot of the groundwork for our application in Sprint 1, which helps greatly for development in Sprint 2. However, we do also have several new features to implement in sprint 2, which might have their own unique problems that might require more effort than expected. Additionally, polishing the website to a final product may also come with its own challenges.

## Sprint 1 Retro

Things That Went Well

* + Our team was able to communicate and collaborate effectively. Everyone was able to complete tasks collaboratively. As well as attending weekly meetings to keep everyone updated.
  + We were able to work ahead of our plan for sprint 1, getting a start on some features in sprint 2.

Things That Could Have Gone Better

* Something that held us up was the upkeep of documentation. Because our documentation was incomplete from sprint 0, sprint 1 required us to play catch-up with updating to match our current progress.

Things That Surprised Us

* + The amount of documentation

Lessons Learned

* + Lack of proper documentation can lead to extra work in the future.

Final Thoughts

Things to keep.

* + - Communication and Collaboration of the team

Things to change.

* + - Make better use of the project board and documentation.