Software Requirements Specification

for

<SuperPrice - Price Matching and Delivery Application >

Version 3.0 approved

Prepared by <Abhijeet Kumar, Ibrahim Al-Ashhab, Rashik Raj, Udit Pradeep Malshe, Vidyut Venkatesan>

<RMIT University>

<08/10/2023>

Copyright © 1999 by Karl E. Wiegers. Permission is granted to use, modify, and distribute this document.

Table of Contents

1. Introduction

1.1 Purpose

SuperPrice is a price matching and delivery application, which will help users to compare the prices of products in local stores. In this SRS document, the following will be discussed: functional and non-functional requirements, use cases, system architecture, user interface design, assumptions and constraints, dependencies, testing and acceptance criteria.

1.2 Document Conventions

Bold and underline - headings

1.3 Intended Audience and Reading Suggestions

This document is designed to guide developers, project managers, and testers while developing, managing, and testing the application.

1.4 Product Scope

The focus of this SuperPrice project is the development of an easy-to-use price matching and delivery application. The Super Price app will show you the best prices of items from different stores and supermarkets in your area. The app will allow you to shop easily, save time and money while offering a variety of delivery options. Realtime price drop and special offer alerts will also be provided by the app. To make it more reliable, the app will be based on user reviews and ratings. At the end of the day, SuperPrice's goal is to make shopping easier for consumers and local businesses alike.

1.5 References

<List any other documents or Web addresses to which this SRS refers. These may include user interface style guides, contracts, standards, system requirements specifications, use case documents, or a vision and scope document. Provide enough information so that the reader could access a copy of each reference, including title, author, version number, date, and source or location.>

2. Overall Description

2.1 Product Perspective

SuperPrice is a novel, self-contained product designed to revolutionize the shopping experience by offering a comprehensive platform for price comparison, product details exploration, and delivery organization. While there are existing systems in the market that offer some of these features,

SuperPrice aims to provide a more integrated, user-friendly, and efficient solution, making it a standout product in the e-commerce domain.

2.2 Product Functions / Functional Requirements

- ***Product Search and Categorization:***
  - *Allow users to search for specific products or browse through different categories.*
  - *Provide a smooth and hassle-free shopping experience.*
- ***Price Comparison:***
  - *Enable users to search for products and compare prices across various local supermarkets.*
  - *Help users identify the store offering the lowest price for their desired items.*
- ***Delivery Organization:***
  - *Facilitate user-organized deliveries, offering multiple flexible delivery options.*
  - *Ensure convenient and efficient delivery arrangements.*
- ***Notifications and Alerts:***
  - *Provide timely notifications and alerts about price drops and special offers.*
  - *Ensure users are informed of opportunities to save money.*
- ***User Reviews and Ratings:***
  - *Allow users to leave reviews and ratings for supermarkets and products.*
  - *Enhance reliability and assist users in making informed purchasing decisions.*
- ***User-Friendly Interface:***
  - *Prioritize simplicity and ease of use for both novice and experienced shoppers.*
  - *Ensure an intuitive and accessible application interface.*
- ***Price Comparison****:*
  - *Allows users to compare prices of products across different stores.*
- ***Product Details****:*
  - *Provides detailed information about products, including images, descriptions, and reviews.*
- ***Offer Exploration****:*
  - *Showcases various offers and discounts available in different stores.*
- ***User Reviews****:*
  - *Users can read reviews for products.*
- ***Search Functionality****:*
  - *Enables users to search for specific products or categories.*
- ***User Accounts****:*
  - *Allows users to create and manage their personal accounts, including profile settings and order history.*
- ***Cart Management****:*
  - *Users can add products to their cart, view cart contents, and proceed to checkout.*
- ***Notification System****:*
  - *Notifies users about price drops, new offers, and stock availability.*
- ***Multiple Payment Options****:*
  - *Supports various payment methods including credit cards, digital wallets, and cash on delivery.*
- ***Secure Checkout****:*
  - *Implements advanced security measures to ensure safe and secure transactions.*

2.3 User Classes and Characteristics

1. Casual Shoppers:

Characteristics*:*

- *Use the product occasionally, perhaps during sales or special occasions.*
- *Primarily interested in basic features like searching for products and comparing prices.*
- *May not have accounts or use advanced features.*
- *Prefer an intuitive and straightforward interface.*

Importance*: Moderate. While they don't use the product frequently, their overall satisfaction can lead to word-of-mouth recommendations.*

2. Regular Shoppers:

Characteristics*:*

- *Use the product frequently, making them familiar with most of its features.*
- *Likely to have user accounts and use features like cart management, user reviews, and notifications.*
- *Value the accuracy and timeliness of price comparisons and notifications.*

Importance*: High. Their consistent use of the product makes them a primary target audience.*

3. Tech-Savvy Users:

Characteristics*:*

- *Comfortable with advanced features and settings.*
- *Likely to explore and use all functionalities of the product.*
- *May provide feedback on technical issues or suggest improvements.*

Importance*: Moderate. While not the primary target audience, their feedback can be invaluable for product improvement.*

4. Retailers and Vendors:

Characteristics*:*

- *Use the product to monitor the pricing of their products in comparison to competitors.*
- *Interested in analytics and data-driven insights.*
- *May have different privilege levels, allowing them to update product details or offers.*

Importance*: Moderate. Their engagement is crucial for accurate product listings and offers.*

5. Administrators:

Characteristics*:*

- *Have the highest level of privileges and access to all product features.*
- *Responsible for managing user accounts, resolving disputes, and maintaining the integrity of the platform.*
- *Require advanced tools for monitoring, analytics, and user management.*

***Importance**: High. They ensure the smooth operation of the platform and address any issues that arise.*

6. Guest Users:

***Characteristics**:*

- *Use the product without creating an account.*
- *Limited to basic functionalities like product search and price comparison.*
- *Cannot access features like cart management or user reviews.*

***Importance**: Low. While they contribute to the product's traffic, they don't engage deeply with its features.*

In summary, while all user classes have their unique importance and role in the ecosystem of the product, Regular Shoppers and Administrators are the most crucial. Their satisfaction and smooth experience are paramount for the product's success.

2.4 Operating Environment

Operating Environment

SuperPrice is designed to be a versatile and adaptable application, capable of functioning efficiently across a variety of environments. Below are the details of the operating environment in which SuperPrice is optimized to run:

Hardware Platform:

- ***Server (AWS t3.micro instance)**:*
  - ***vCPU**: 2 virtual cores.*
  - ***RAM**: 1 GB memory.*
  - ***Network**: Up to 5 Gigabit bandwidth for seamless data transfer and communication.*
  - ***Storage**: EBS (Elastic Block Store) volumes, with the option to scale based on the application's needs.*
- ***Client**:*
  - *SuperPrice is designed to be responsive and can be accessed from various devices including desktops, laptops, tablets, and smartphones.*
  - *Minimum 2 GB RAM for smooth operation on client devices.*
  - *Modern graphics card for rendering product images and UI components efficiently.*

Operating System:

- ***Server**:*
  - *Linux (Ubuntu 20.04 LTS or later versions) is the primary OS for backend operations on the AWS t3.micro instance.*
  - *Other distributions like CentOS or Debian are also supported.*

- ***Client***:
    - *SuperPrice is web-based and can be accessed via web browsers. Thus, it's compatible with all major operating systems including Windows, macOS, Linux, iOS, and Android.*

Other Software Components:

- ***Web Browsers****: SuperPrice is optimized for modern web browsers including Google Chrome (version 90 and above), Mozilla Firefox (version 88 and above), Safari (version 14 and above), and Microsoft Edge (version 90 and above).*
- ***Database****: MySQL (version 8.0 or higher) for data storage.*
- ***Backend****: Java (Version 17.0 or higher) for running the backend services. Apache Maven for dependency management.*
- ***Frontend****: Node.js and npm for managing and running the frontend components.*
- ***Containerization****: Docker for creating and managing containers for both frontend and backend components.*
- ***Deployment****: AWS Elastic Beanstalk for deploying and managing the application on t3.micro instances.*
- ***Version Control****: Git for source code management and versioning.*
- ***Cloud Services****: AWS services including Elastic Beanstalk for deployment, Elastic Container Registry (ECR) for storing Docker images, and Elastic IP for static IP configurations.*

Coexistence:

SuperPrice is designed to coexist peacefully with other software components. Its modular architecture ensures that each component of the application runs in isolation, reducing the chances of conflicts with other software. Additionally, by leveraging containerization through Docker and deploying on AWS Elastic Beanstalk, SuperPrice ensures that its environment remains consistent and isolated from external interferences.

2.5 Design and Implementation Constraints

SuperPrice, while aiming to be a comprehensive and user-friendly application, operates within certain constraints that shape its design and implementation. These constraints are as follows:

1. Corporate or Regulatory Policies:

- ***Data Privacy****: SuperPrice must adhere to data protection regulations such as GDPR (General Data Protection Regulation) and CCPA (California Consumer Privacy Act). This means user data must be handled with utmost care, ensuring encryption, anonymization, and the right to be forgotten.*
- ***E-commerce Regulations****: As a price comparison and shopping facilitator, SuperPrice must comply with local e-commerce regulations, ensuring transparent pricing, clear return policies, and accurate product descriptions.*

2. Hardware Limitations:

- ***Server Capacity****: Being hosted on AWS t3.micro instances, there are limitations in terms of CPU, memory, and storage. This necessitates efficient code and database optimizations to ensure smooth performance.*

3. Interfaces to Other Applications:

- *SuperPrice might need to interface with various local supermarket databases or APIs for real-time price updates. These interfaces might have their own rate limits, data structures, and availability constraints.*

4. Specific Technologies, Tools, and Databases:

- *The application is bound to use specific technologies like Java for the backend, React for the frontend, and MySQL for the database. This might limit the use of certain features or libraries exclusive to other technologies.*

5. Parallel Operations:

- *Given the multi-user nature of the platform, SuperPrice must support concurrent operations. This introduces constraints related to data consistency, race conditions, and real-time updates.*

6. Language Requirements:

- *Initially, SuperPrice might be available in English. Expanding to other languages would require localization efforts, which might not be immediately feasible.*

7. Communications Protocols:

- *SuperPrice will primarily use HTTP/HTTPS for communication. This might limit real-time features that could benefit from protocols like WebSockets.*

8. Security Considerations:

- *SuperPrice must implement robust security measures, including encryption for data at rest and in transit, secure authentication mechanisms, and protection against common web vulnerabilities like SQL injection and XSS.*

9. Design Conventions or Programming Standards:

- *Developers must adhere to specific coding standards and best practices to ensure maintainability. This might include conventions related to naming, commenting, and code organization.*

10. Cloud Services Limitations:

- *Being hosted on AWS, there are specific limitations and costs associated with data transfer, storage, and compute resources. This might influence decisions related to data retention, backups, and scaling.*

11. Budget Constraints:

- *As with any project, there might be budgetary constraints that limit the use of certain premium tools, services, or resources.*

In conclusion, while SuperPrice aims to provide a seamless and comprehensive shopping experience, it operates within a set of constraints that shape its design, features, and performance. These constraints, however, also guide the development process, ensuring that the application is robust, efficient, and compliant with all necessary standards and regulations.

2.6 User Documentation

User Documentation

The primary source of user documentation for the SuperPrice application is available through the project's GitHub repository. The documentation is structured in the form of README files, which provide comprehensive details about the setup, deployment, and usage of both the frontend and backend components of the application.

1. ***Main Repository README**:*
    - *Location: SuperPrice Main Repository README*
    - *Description: This README provides an overview of the SuperPrice application, including its features, architecture, and deployment instructions. It also contains links to other relevant documentation and resources.*
2. ***Backend README**:*
    - *Location: SuperPrice Backend README*
    - *Description: This README is dedicated to the backend component of the SuperPrice application. It provides detailed instructions on setting up the backend environment, running the backend services, and any other backend-specific considerations.*
3. ***Frontend README**:*
    - *Location: SuperPrice Frontend README*
    - *Description: This README focuses on the frontend component of the SuperPrice application. It offers guidance on setting up the frontend environment, starting the frontend application, and other frontend-specific details.*

Users are encouraged to refer to these README files for a thorough understanding of the SuperPrice application. The documentation is designed to be user-friendly, catering to both technical and non-technical users. It provides step-by-step instructions, code snippets, and relevant links to ensure that users can easily navigate, set up, and use the SuperPrice application.

2.7 Assumptions and Dependencies

Assumptions:

1. ***Stable Third-Party Services**: It is assumed that all third-party services and APIs integrated into the SuperPrice application will remain available and stable throughout the application's lifecycle.*
2. ***Consistent User Base**: The system is designed with the assumption of a certain range of concurrent users. A sudden and significant increase in users might require additional resources or optimizations.*
3. ***Operating System Compatibility**: The application assumes that most users will access the platform using modern operating systems that support the latest web browsers.*
4. ***Internet Connectivity**: The SuperPrice application assumes that users have a stable internet connection for seamless access to all features.*

5. ***Cloud Infrastructure****: It is assumed that the cloud infrastructure, specifically AWS services, will be available with a high uptime, ensuring the application's availability.*
6. ***Data Integrity****: The application assumes that the data fed into it, either by users or third-party integrations, is accurate and reliable.*

Dependencies:

1. ***AWS Services****: The SuperPrice application is heavily dependent on various AWS services, such as AWS Elastic Beanstalk and AWS ECR, for deployment, scaling, and container management.*
2. ***Docker****: The application's deployment and consistency across different environments are dependent on Docker for containerization.*
3. ***Database****: The application relies on a MySQL database for storing user data, product information, and other essential data. Any changes or issues with the database version or compatibility could impact the application.*
4. ***Frontend and Backend Frameworks****: The application is built using React for the frontend and Spring Boot for the backend. Any significant changes or discontinuation of support for these frameworks could necessitate modifications to the application.*
5. ***Third-Party APIs****: The application might integrate with third-party APIs for various functionalities, such as payment gateways or product information. The availability and functionality of these APIs are crucial for the corresponding features in the SuperPrice application.*
6. ***Security Protocols****: The application depends on modern security protocols and standards, such as JWT for authentication. Changes or vulnerabilities in these protocols could impact the application's security.*

3. External Interface Requirements

3.1 User Interfaces

The SuperPrice application is designed with a user-centric approach, ensuring that the interface is intuitive, responsive, and easy to navigate. The application's user interface is built using React, a popular JavaScript library, and follows modern design principles to provide a seamless user experience.

Key Characteristics:

1. ***Responsive Design****: The application is designed to be responsive, ensuring that it looks and functions well on devices of all sizes, from desktops to mobile phones.*
2. ***Consistent Layout****: The application maintains a consistent layout across different pages, with a standard navigation bar at the top and a footer at the bottom.*
3. ***Interactive Elements****: Interactive elements like buttons, dropdowns, and sliders are designed to provide visual feedback when interacted with, enhancing user engagement.*
4. ***Error Handling****: In case of errors or invalid inputs, the application provides clear and concise error messages to guide the user.*
5. ***Loading Indicators****: During data fetching or processing, loading indicators are displayed to inform users that the application is working in the background.*
6. ***Accessibility****: The application follows accessibility guidelines, ensuring that it is usable by individuals with disabilities.*

Sample Screens:

1. ***Homepage****: Displays featured products, top offers, and allows users to search for specific products or categories.*
2. ***Product Detail Page****: Provides detailed information about a selected product, including its price, description, reviews, and related products.*
3. ***Search Results Page****: Displays products based on the user's search query, with options to filter and sort the results.*
4. ***User Profile****: Allows users to view and edit their profile information, view order history, and manage their preferences.*

UI Components:

The SuperPrice application is modular, with reusable UI components that ensure consistency and reduce redundancy. Some of the primary components include:

- ***Navbar****: Contains links to the main sections of the application and provides access to user account settings and notifications.*
- ***Footer****: Displays essential links, contact information, and copyright details.*
- ***Product Card****: A reusable component that displays product information in a concise format, used in search results and recommendations.*
- ***Review Section****: Allows users to read and write reviews for products.*
- ***Price Comparison Tool****: Enables users to compare prices of a product across different stores.*

UI Standards:

The application follows a set of UI standards to maintain consistency:

- ***Color Scheme****: A consistent color scheme is used throughout the application, with primary, secondary, and accent colors defined.*
- ***Typography****: A standard font family, size, and style are used across all pages to ensure readability and consistency.*
- ***Spacing and Alignment****: Consistent spacing and alignment rules are followed to ensure a clean and organized layout.*
- ***Icons****: Standardized icons are used for common actions like search, cart, and user profile.*

Future Enhancements:

While the current user interface meets the primary requirements, future enhancements could include:

- ***Dark Mode****: An option for users to switch to a dark-themed interface.*
- ***Personalized Recommendations****: Displaying product recommendations based on user preferences and browsing history.*
- ***Interactive Tutorials****: Guided tutorials for new users to familiarize themselves with the application's features.*

In conclusion, the SuperPrice application's user interface is designed to provide a smooth and intuitive shopping experience. With a focus on user-centric design principles and consistent UI

standards, the application ensures that users can easily navigate, search, and make informed purchasing decisions.

3.2 Hardware Interfaces

User Devices:

1. ***Desktops/Laptops**:*
    - *Accessible via standard web browsers.*
    - *Supports keyboards, mice, and touchpads.*
2. ***Mobile Devices**:*
    - *Optimized for touch interactions.*
    - *Adapts to various screen sizes.*

Server Infrastructure:

1. ***Cloud Servers (e.g., AWS EC2 t3.micro)**:*
    - *Hosts backend services and databases.*
    - *Communicates via APIs.*
2. ***Databases (e.g., AWS RDS)**:*
    - *Stores product data and user info.*
    - *Backend interfaces using SQL.*
3. ***Storage Systems (e.g., AWS S3)**:*
    - *Stores static assets like images.*
    - *Accessed via URLs.*
4. ***Networking Equipment**:*
    - *Ensures communication between devices and servers.*
    - *Implements security measures.*

Communication Protocols:

1. ***HTTP/HTTPS**: For web pages and API calls.*
2. ***WebSockets**: For real-time notifications.*
3. ***SQL**: For database queries.*
4. ***TCP/IP**: Base transport protocol.*

In essence, SuperPrice interfaces with user devices and servers, ensuring efficient communication and performance.

3.3 Software Interfaces

Software Interfaces

1. Backend API (Version 1.0.0):

- ***Purpose**: Handles data transactions, user authentication, and product queries.*
- ***Incoming Data**: User credentials, search queries, product updates.*
- ***Outgoing Data**: Product details, authentication tokens, search results.*
- ***Communication**: RESTful API using JSON payloads.*

2. Database (MySQL 8.0-oracle):

- ***Purpose****: Stores user data, product information, and transaction records.*
- ***Incoming Data****: New user registrations, product additions, transaction logs.*
- ***Outgoing Data****: User profiles, product details.*
- ***Services Needed****: SQL queries, backup, and restore services.*

3. Operating System (linux/amd64):

- ***Purpose****: Hosts the application and provides system resources.*
- ***Services Needed****: System monitoring, task scheduling, security updates.*

4. Frontend Framework (React 18.2 with TypeScript 4.9):

- ***Purpose****: Renders the user interface and interacts with the backend.*
- ***Incoming Data****: User interactions, form submissions.*
- ***Outgoing Data****: Rendered web pages, user notifications.*
- ***Services Needed****: Component rendering, state management, event handling.*

5. External Libraries:

- ***axios (1.5)****: For making HTTP requests.*
- ***react-router (6.15.0)****: For frontend routing.*
- ***bootstrap (5.3.1)****: For UI styling.*

6. Cloud Services (AWS Beanstalk & S3):

- ***Purpose****: Deployment, scaling, and storage of static assets.*
- ***Incoming Data****: Deployment packages, images, static files.*
- ***Outgoing Data****: Deployed application instances, served static files.*
- ***Services Needed****: Deployment automation, scaling, storage management.*

Shared Data*:*

- ***User Profiles****: Shared between the backend and frontend for authentication and personalization.*
- ***Product Details****: Shared between the database, backend, and frontend for display and transactions.*

3.4 Communications Interfaces

Communications Interfaces

1. Web Browser Communications:

- ***Purpose****: To render the application's user interface and facilitate user interactions.*
- ***Protocols****: HTTP/HTTPS.*
- ***Message Formatting****: HTML, CSS, and JavaScript.*
- ***Security****: HTTPS with SSL/TLS encryption for secure data transmission.*

2. Network Server Communications:

- ***Purpose****: To handle requests and responses between the frontend and backend.*
- ***Protocols****: TCP/IP for general network communications, HTTP/HTTPS for web-specific communications.*
- ***Message Formatting****: JSON for data interchange.*
- ***Security****: Secure API endpoints using authentication tokens (e.g., JWT).*

3. Electronic Forms:

- ***Purpose****: To gather user input for tasks like registration, product search, and feedback.*
- ***Protocols****: HTTP POST method for form submission.*
- ***Message Formatting****: Form data encoded as* ***application/x-www-form-urlencoded*** *or* ***multipart/form-data*** *for file uploads.*
- ***Security****: Input validation and sanitization to prevent SQL injection and XSS attacks.*

4. File Transfers:

- ***Purpose****: To upload/download files such as product images, user avatars, and digital receipts.*
- ***Protocols****: FTP (File Transfer Protocol) for direct file transfers, HTTP/HTTPS for web-based transfers.*
- ***Message Formatting****: Binary for direct file data, Base64 encoded for embedded files in JSON or XML.*
- ***Security****: SFTP (Secure File Transfer Protocol) for encrypted file transfers, HTTPS for secure web-based transfers.*

5. Real-time Notifications:

- ***Purpose****: To inform users of real-time events like price drops, new offers, or chat messages.*
- ***Protocols****: Notification API is used.*
- ***Message Formatting****: JSON for structured data.*

6. Third-party API Integrations:

- ***Purpose****: To fetch external images, or utilize other services.*
- ***Protocols****: HTTP/HTTPS.*
- ***Message Formatting****: JSON or XML, depending on the third-party service.*
- ***Security****: API keys, OAuth tokens, or other authentication mechanisms.*

In summary, SuperPrice utilizes various communication interfaces to ensure efficient data exchange, real-time updates, and secure interactions. Proper encryption, authentication, and validation mechanisms are in place to safeguard user data and enhance the overall user experience.

4. Other Nonfunctional Requirements

4.1 Performance Requirements

- ***Response Time:***
  - *The application should respond to user actions, such as product searches and price comparisons, within 2 seconds to ensure a smooth and seamless user experience.*
  - *Rapid response times will prevent user frustration and maintain engagement.*
- ***Search and Comparison Speed:***
  - *The application must be capable of handling simultaneous product searches and price comparisons for multiple users without significant delays.*
  - *Searches and comparisons should complete within 1 second, even during peak usage.*

4.2 Security Requirements

- ***User Data Protection:***
  - *User data, including personal details and payment information, must be encrypted both during transmission and storage to prevent unauthorized access.*
- ***Authorization Levels:***
  - *Different user roles (users, administrators) should have appropriate levels of access and permissions to ensure data integrity and prevent unauthorized actions.*
- ***Secure API Integration:***
  - *APIs connecting with supermarket systems and payment gateways should use secure communication protocols (e.g., HTTPS) and require API keys or tokens for access.*

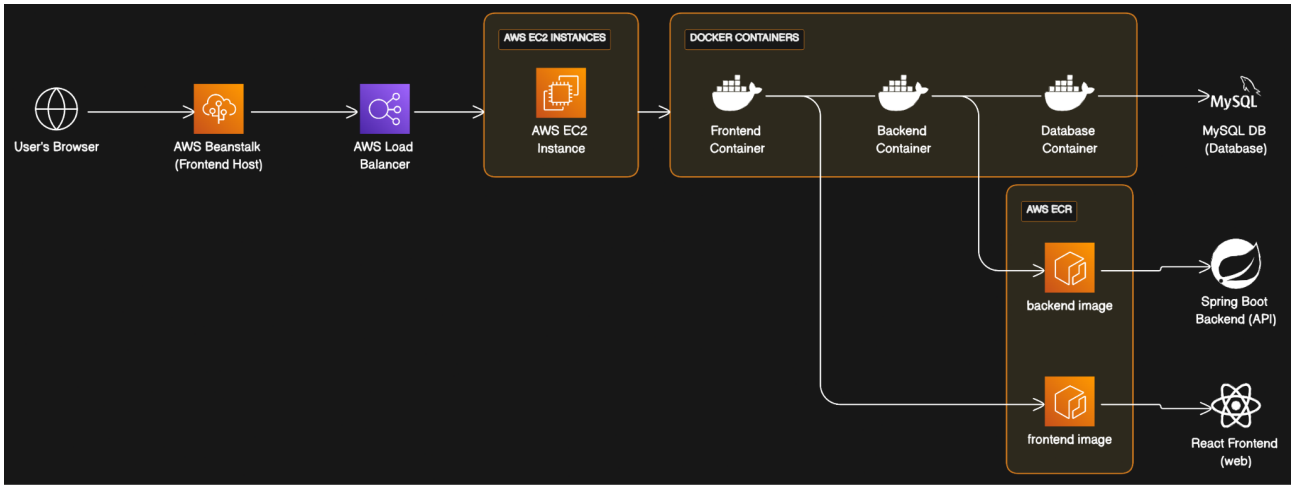
4.3 Software Quality Attributes

- ***Usability:***
  - *The application's user interface should be intuitive and easy to navigate*
- ***Availability:***
  - *The application's server infrastructure should provide high availability,*

5. Other Requirements

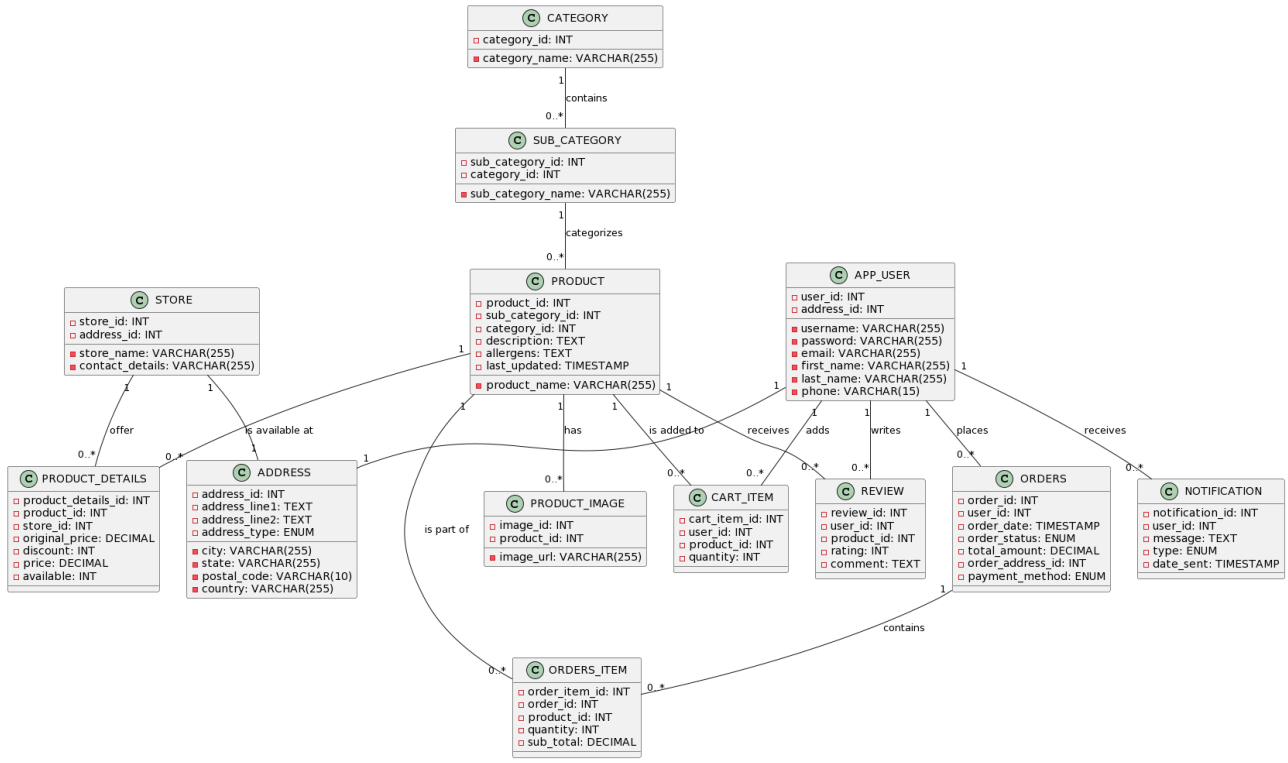
- **Database Requirements:**
  - *The application's database should support efficient storage and retrieval of product information, and reviews.*

6. System Architecture



- ***Client***: *Represents the user's device or web browser, which interacts with the application.*
- ***Internet***: *Acts as the medium for transmitting requests and responses.*
- ***Frontend (React)***: *Provides the user interface and handles user interactions.*
- ***Backend (Spring Boot)***: *Processes requests, executes business logic, and interacts with the database. It uses JWT for authentication and authorization, ensuring secure access to resources.*
- ***JPA (Java Persistence API)***: *Acts as the data access layer, providing an abstraction over database operations and ensuring smooth interaction with the MySQL database.*
- ***MySQL***: *The relational database management system where all application data is stored.*
- ***JWT (JSON Web Tokens)***: *Used for authentication and authorization. It ensures that users are who they claim to be and have the right permissions to access resources.*
- ***Docker***: *Provides containerization, ensuring that the application runs consistently across different environments by packaging it with all its dependencies.*
- ***AWS ECR (Elastic Container Registry)***: *A managed container image registry service where Dockerized application images are stored and retrieved.*
- ***AWS Beanstalk***: *A platform-as-a-service (PaaS) that allows developers to deploy, monitor, and scale applications. It abstracts away infrastructure details, making deployment easier and more efficient.*

7. Data Model



ADDRESS:

Represents the address details.
- Attributes: address_id, address_line1, address_line2, city, state, postal_code, country, address_type.
- Relationships:
- One-to-Many with STORE: A store can have one address.
- One-to-Many with APP_USER: A user can have one address.
- One-to-Many with ORDERS: An order can have one address.

CATEGORY:

Represents different categories of products.

- Attributes: category_id, category_name.

- Relationships:

- One-to-Many with SUB_CATEGORY: A category can have multiple sub-categories.

- One-to-Many with PRODUCT: A category can have multiple products.

SUB_CATEGORY:

Represents sub-categories of products.

- Attributes: sub_category_id, sub_category_name, category_id.

- Relationships:

- Many-to-One with CATEGORY: Each sub-category belongs to one category.

- One-to-Many with PRODUCT: A sub-category can have multiple products.

PRODUCT:

Represents individual grocery items.

- Attributes: product_id, product_name, sub_category_id, category_id, description, allergens, last_updated.

- Relationships:

- Many-to-One with SUB_CATEGORY: Each product belongs to one sub-category.

- Many-to-One with CATEGORY: Each product belongs to one category.

- One-to-Many with PRODUCT_IMAGE: A product can have multiple images.

- One-to-Many with REVIEW: A product can have multiple reviews.

- One-to-Many with ORDERS_ITEM: A product can be part of multiple orders.

- One-to-Many with CART_ITEM: A product can be added to multiple carts.

- One-to-Many with PRODUCT_DETAILS: A product can have multiple details.

PRODUCT_IMAGE:

Represents images of products.

- Attributes: image_id, product_id, image_url.

- Relationships:

- Many-to-One with PRODUCT: Each image belongs to one product.

STORE:

Represents supermarkets and stores.

- Attributes: store_id, store_name, address_id, contact_details.

- Relationships:

- One-to-Many with PRODUCT_DETAILS: A store can have details of multiple products.

- Many-to-One with ADDRESS: Each store is located at one address.

APP_USER:

Represents registered users.

- Attributes: user_id, username, password, email, first_name, last_name, phone, address_id.

- Relationships:

- One-to-Many with REVIEW: A user can write multiple reviews.

- One-to-Many with ORDERS: A user can make multiple orders.

- One-to-Many with NOTIFICATION: A user can receive multiple notifications.

- One-to-Many with CART_ITEM: A user can have multiple items in their cart.

- Many-to-One with ADDRESS: Each user can have one address.

REVIEW:

Represents user reviews and ratings for products.

- Attributes: review_id, user_id, product_id, rating, comment.

- Relationships:

- Many-to-One with APP_USER: Each review is written by one user.

- Many-to-One with PRODUCT: Each review is about one product.

ORDERS:

Represents a purchase made by a user.

- Attributes: order_id, user_id, order_date, order_status, total_amount, order_address_id, payment_method.

- Relationships:

- One-to-Many with ORDERS_ITEM: An order can consist of multiple items.

- Many-to-One with APP_USER: Each order is made by one user.

- Many-to-One with ADDRESS: Each order can have one address.

ORDERS_ITEM:

Represents individual items within an order.

- Attributes: order_item_id, order_id, product_id, quantity, sub_total.

- Relationships:

- Many-to-One with ORDERS: Each order item is part of one order.

- Many-to-One with PRODUCT: Each order item represents one product.

NOTIFICATION:

Represents notifications sent to users.

- Attributes: notification_id, user_id, message, type, date_sent.

- Relationships:

- Many-to-One with APP_USER: Each notification is received by one user.

***CART_ITEM**:*

Represents items added to a user's shopping cart.

- Attributes: cart_item_id, user_id, product_id, quantity.

- Relationships:

- Many-to-One with APP_USER: Each cart item belongs to one user.

- Many-to-One with PRODUCT: Each cart item represents one product.

***PRODUCT_DETAILS**:*

Represents details of products in stores.

- Attributes: product_details_id, product_id, store_id, original_price, discount, price, available.

- Relationships:

- Many-to-One with PRODUCT: Each product detail is about one product.

- Many-to-One with STORE: Each product detail is available in one store.

DataModel Changes

1. Additions:

*- **ADDRESS Table**: A new table to represent the address details.*

*- **CATEGORY Table**: While the old schema had a CATEGORY table, the new schema has added a unique constraint to the category_name.*

*- **SUB_CATEGORY Table**: A new table to represent sub-categories of products. This table also establishes a relationship with the CATEGORY table.*

*- **PRODUCT Table**:*

- Added sub_category_id to establish a relationship with the SUB_CATEGORY table.

- Added allergens attribute.

*- **PRODUCT_IMAGE Table**: A new table to represent images of products.*

*- **STORE Table**:*

- The new schema has added an address_id to establish a relationship with the ADDRESS table.

*- **APP_USER Table**:*

- Renamed from "USER" to "APP_USER".

- Added attributes: first_name, last_name, and phone.

- Address attribute from the old schema has been replaced with address_id to establish a relationship with the ADDRESS table.

*- **PRODUCT_DETAILS Table**: A new table to represent details of products in stores.*

2. Modifications:

*- **PRODUCT Table**:*

- Removed Price, Availability, and ImageURL attributes. These details are now in the PRODUCT_DETAILS and PRODUCT_IMAGE tables.

*- **STORE Table**:*

- Removed StoreLocation. The location details are now in the ADDRESS table.

*- **REVIEW Table**: No significant changes.*

*- **ORDERS Table**:*

- Renamed from "TRANSACTION" to "ORDERS".

- Added attributes: order_status, order_address_id, and payment_method.

*- **ORDERS_ITEM Table**:*

- Renamed from "TRANSACTION_ITEM" to "ORDERS_ITEM".

- Removed SubTotal attribute.

3. Removals:

*- **PRODUCT_CATEGORY (Association Table)**: This table is not present in the new schema. The many-to-many relationship between products and categories seems to be resolved using the SUB_CATEGORY table in the new schema.*

*- **PRODUCT_STORE (Association Table):** This table is not present in the new schema. The relationship between products and stores is now handled by the PRODUCT_DETAILS table.*

DataModel Changes (Milestone 3)

1. Additions:

No new tables were added in this milestone.

2. Modifications:

*- **CART_ITEM** Table:*

- *Added a new attribute **product_details_id** to establish a relationship with the PRODUCT_DETAILS table. This change allows the cart items to reference specific product details, such as price and availability in a particular store.*

*- **PRODUCT_DETAILS** Table:*

- *Added a **CHECK** constraint for the **discount** attribute to ensure that the discount value is between 0 and 100. This ensures that the discount percentage is valid.*

*- **SUB_CATEGORY** Table:*

- *Removed the commented-out **product_id** attribute and its foreign key constraint. This indicates a decision to not associate sub-categories directly with individual products.*

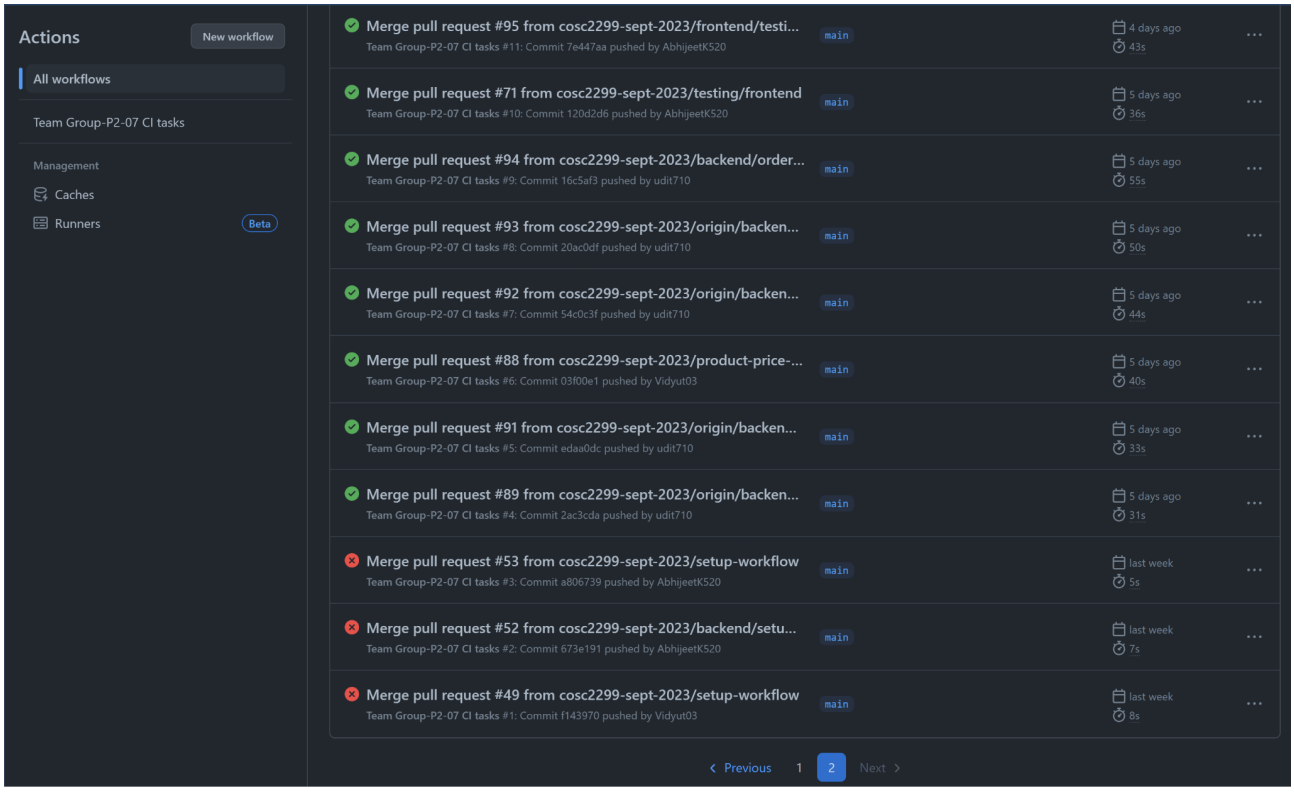
*- **NOTIFICATION** Table:*

- *Introduced a new **type** ENUM attribute to categorize notifications into 'OFFERS', 'ORDER', or 'ERROR'. This helps in classifying the type of notification sent to the user.*

3. Removals:

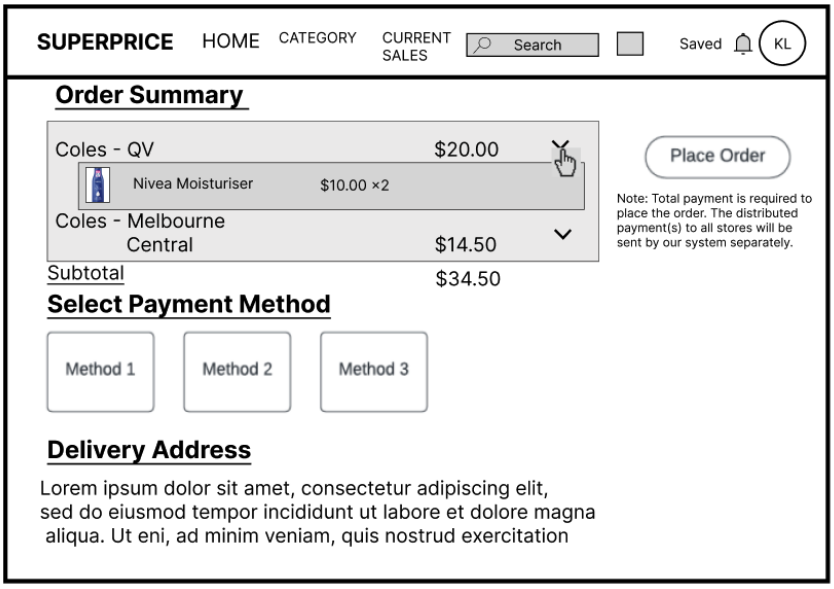
- *No tables were removed in this milestone.*

These modifications in milestone 3 further refine the data model to cater to specific requirements and ensure data integrity. The changes reflect a more detailed and structured approach to handling product details, user notifications, and cart items.

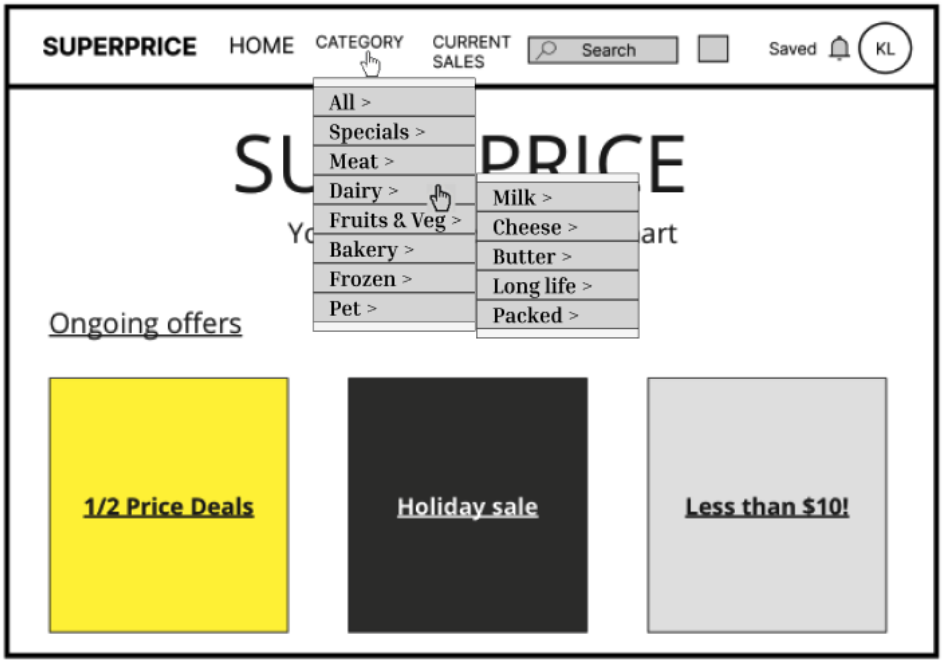


8. User Interface Design / Wireframes

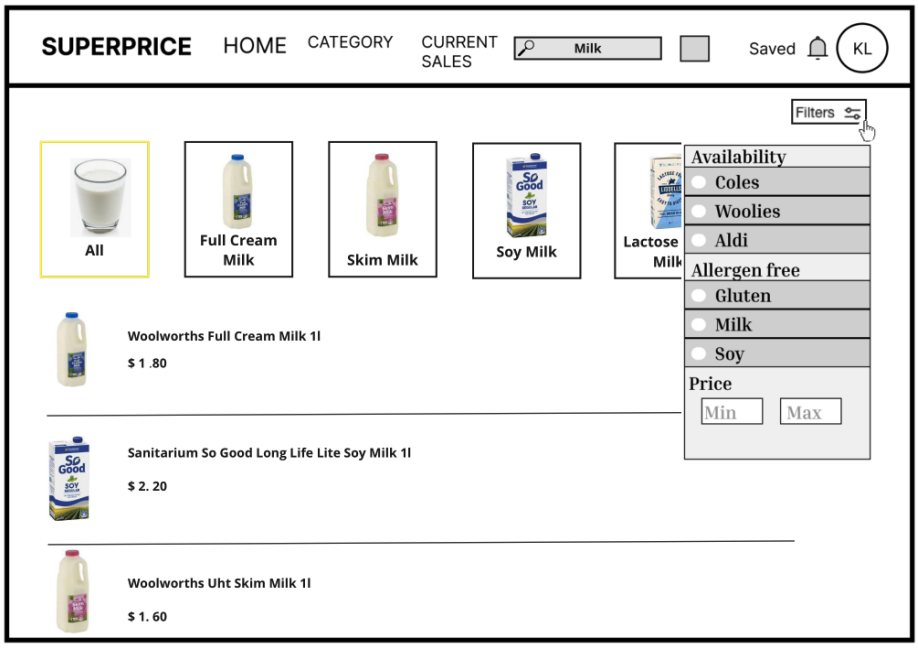
1. Checkout Page



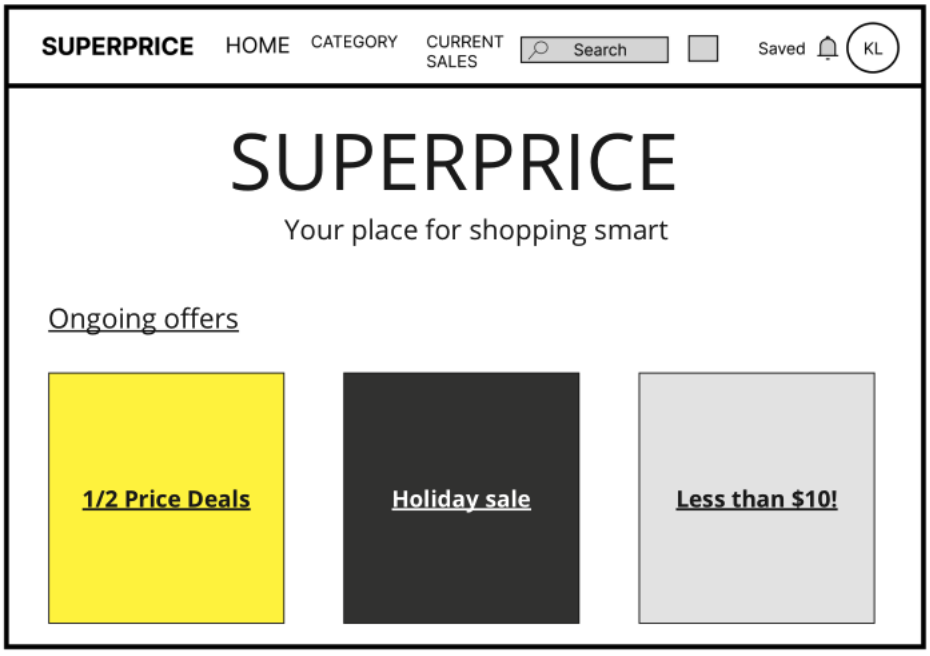
2. Category Hover view



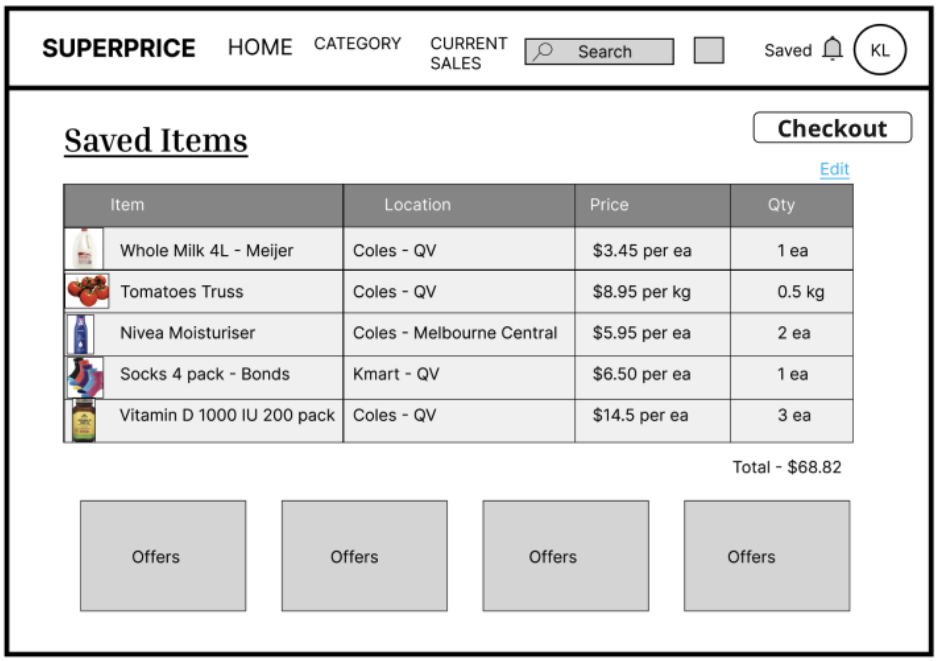
3. Filter View



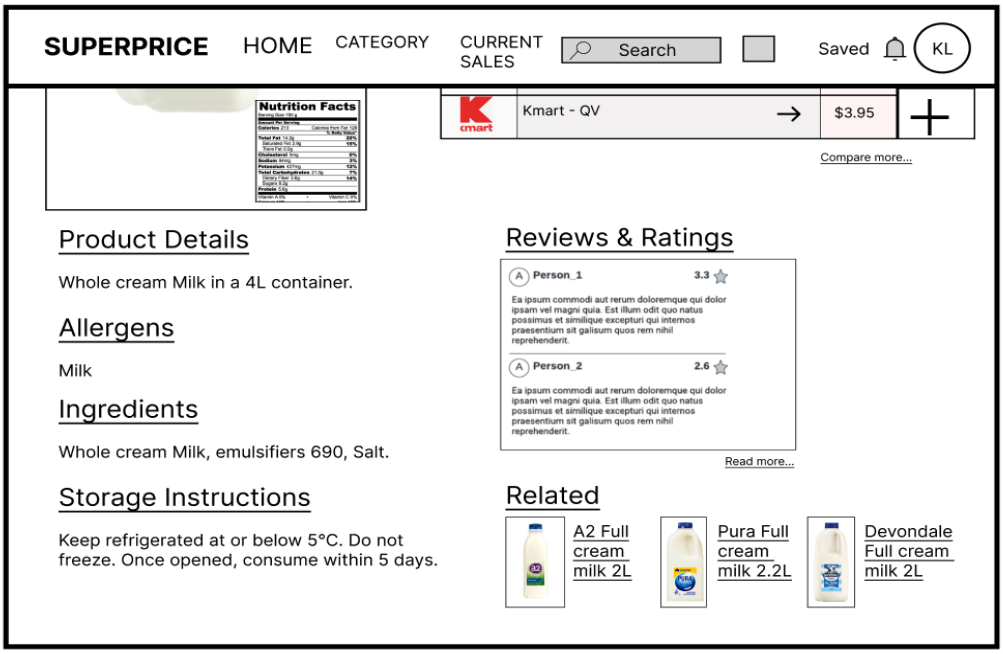
4. Home Page



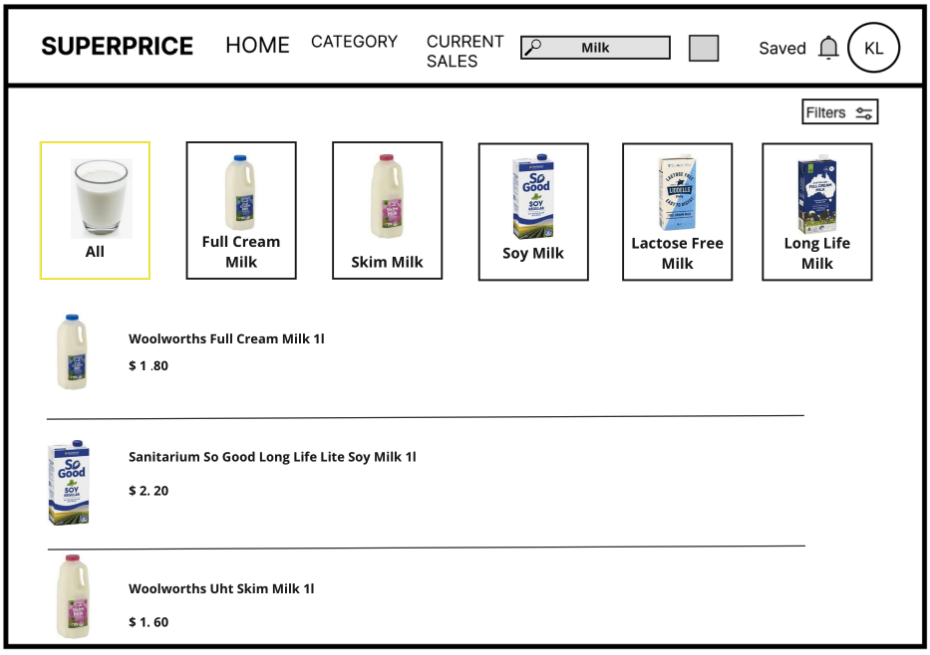
5. Saved Items Page



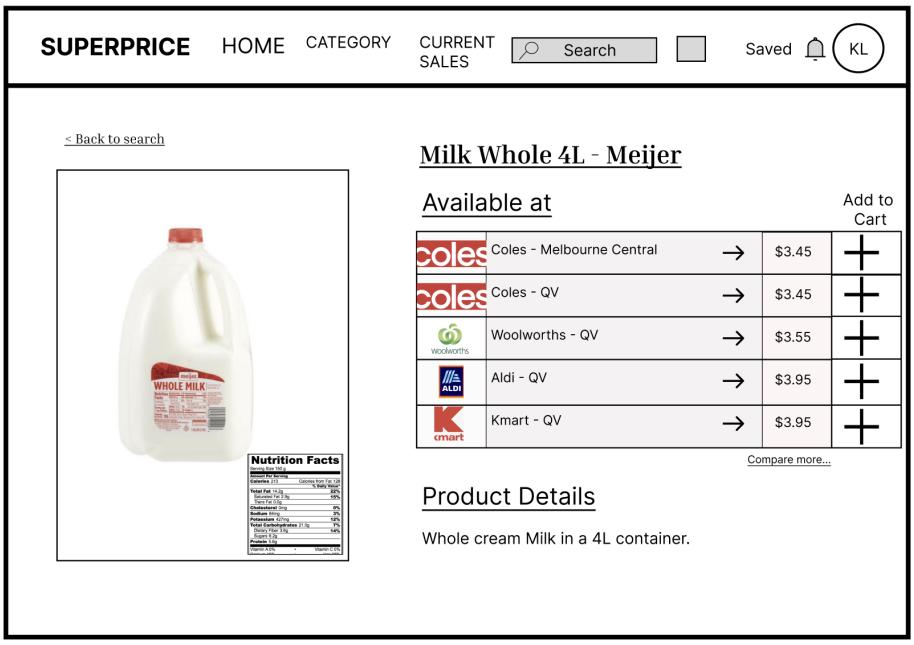
6. Product Details View



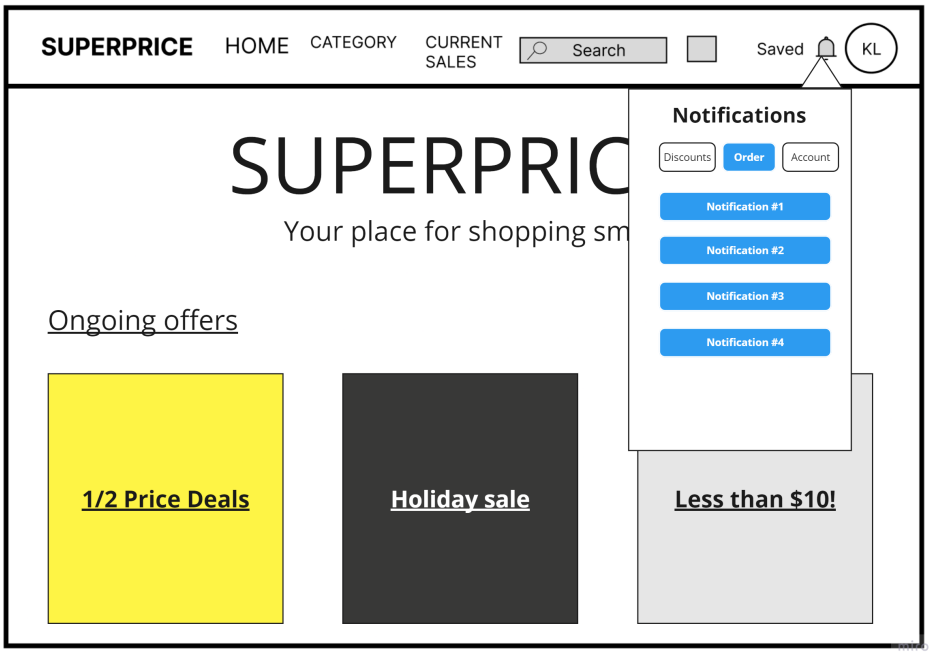
7. Search Results Page



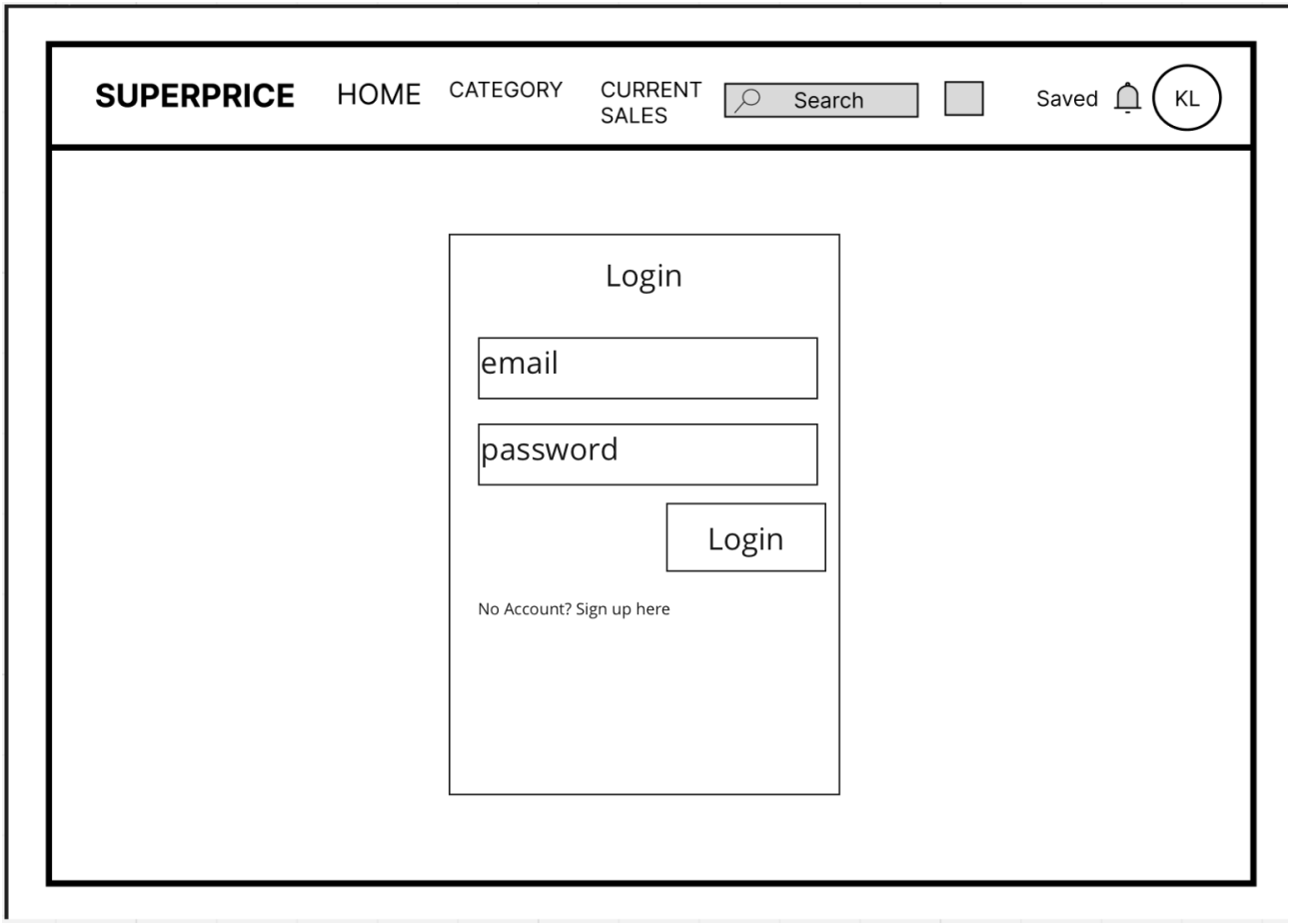
8. Product Availability View



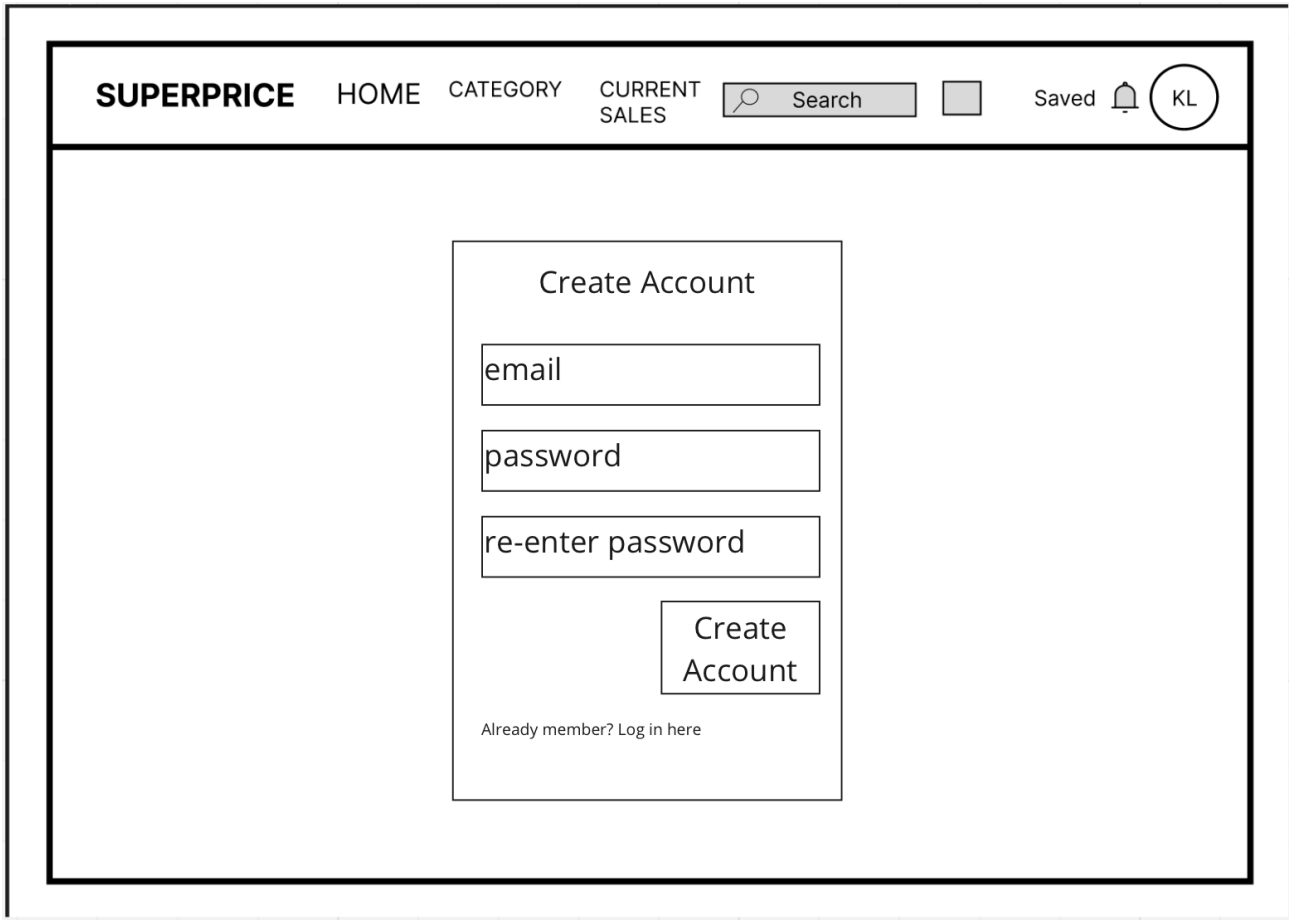
9. Notification Tab



10. User Login Page



11. User Sign up Page



9. Assumptions and Constraints

Assumptions:

1. ***Single Database: The architecture assumes a single MySQL database instance. In a real-world scenario, there might be multiple databases or replicas for scalability and redundancy.***
2. ***Stateless Backend: The backend is assumed to be stateless, meaning it doesn't maintain any user session information. Any state information required is either stored in the database or on the client side.***
3. ***Single Backend Instance: The diagram shows a single backend instance. In reality, there might be multiple instances for load balancing and failover.***
4. ***Monolithic Backend: The backend is assumed to be monolithic, even though there's a mention of services. In a microservices architecture, each service could be a separate entity with its own database.***
5. ***Static Pages: The pages (Homepage, ProductPage, CheckoutPage) are assumed to be static and served by Next.js. Dynamic content is fetched via API calls.***
6. ***Direct Database Access: The backend has direct access to the database. In some architectures, there might be a separate data access layer or service.***

Constraints:

1. ***Scalability: With a single database and backend instance, scalability might be a concern. Horizontal scaling strategies would need to be considered.***

2. ***Performance: Direct database access from the backend can introduce latency, especially if the database is not optimized or if complex queries are used.***

3. ***Security: Direct exposure of API to the frontend can be a security risk. Proper authentication and authorization mechanisms need to be in place.***

4. ***Maintenance: A monolithic backend can become complex over time, making it harder to maintain and deploy.***

5. ***Database Dependency: The system is heavily dependent on the MySQL database. Any downtime or performance issues with the database will directly impact the application.***

6. ***Limited Integration Points: The diagram shows a single "Services" integration point. In reality, there might be multiple third-party services (e.g., payment gateways, recommendation engines) that the backend interacts with.***

10. Dependencies

- *Spring Boot*
- *React*

11. Testing, User Stories, and Acceptance Criteria

Story #1:	**Item Checkout**	**Priority**	
		Effort	
As a	user		
I want	to be able to review my order summary,		
So that	I can select a payment method, and review your shipping address during checkout.		
Acceptance criteria	Given that I am checking out, I should be able to see my order summary and total, address and be able to select the payment method.		
Test Case	Test Case : Select Payment Method Description: Ensure the user can select a payment method during the checkout process. Preconditions: User is on the checkout page with an order summary displayed. Steps: • Review the order summary. • Select a payment method (e.g., credit card, PayPal). • Provide necessary payment information. • Continue to the next step. • Expected Result: The user successfully selects a payment method and proceeds to the next step of the checkout process		

Story #2:	**Notifications**	**Priority**	
		Effort	
As a	user		
I want	I would like to be notified of any new discounts		
So that	I can keep track of all the discounts.		
Acceptance criteria	1. I should get notified of any new discounts as soon as they are released 2. There is option to get notified via app notification, email or mobile 3. There is a view to filter the discounts		
Test Case	**Quick Access to Latest Deals** • Description: As a deal-seeking shopper, I want quick access to the latest deals, discounts, and special offers on the website's home page to save money and take advantage of limited-time promotions. **Test Case: Display Latest Deals on Home Page** • Scenario: The user visits the website's home page. • Steps to Reproduce: a. Open the web browser. b. Navigate to the website's home page. • Expected Outcome: The home page should prominently display the latest deals, discounts, and special offers.		

Story #3:	**Availability of an item**	**Priority**	
		Effort	
As a	budget-conscious shopper,		
I want	to see if a specific product is currently in stock at nearby supermarkets,		
So that	I can decide where to buy it without making an unnecessary trip.		
Acceptance criteria	Given I am a budget-conscious shopper, When I search for a specific product, Then I should see a list of supermarkets that have the product in stock near me.		
Test Case	**Description: To check availability of product** **Steps:** 1. Search for product 2. View item page Expected Outcome: Availability of product on different stores is seen		

Story #4:	**Delivery Tracking**	**Priority**	
		Effort	
As a	online shopper		
I want	to be able to track my order's delivery status in real-time,		
So that	I can be informed of its whereabouts and estimated time of arrival, ensuring I'm available to receive it.		
Acceptance criteria	Given I have placed an order and it's ready for delivery, When I go to the 'My Orders' section in the application, Then I should see a 'Track Delivery' option for my recent order. Given I click on the 'Track Delivery' option, When the tracking page loads, Then I should see the current status of my delivery (e.g., "Out for delivery", "Arriving soon", "Delayed"). Given I'm on the delivery tracking page, When I look at the delivery details, Then I should see an estimated time of arrival (ETA) for my order.		
Test Case	Test Case: Track delivery Preconditions: Logged into an account. Steps 1. Make a mock purchase to be dlivered 2. Track the delivery drive Expected result: Delivery driver is accurately tracked.		

Story #5:	**Product Search and Categorization**	**Priority**	
		Effort	
As a	online user		
I want	to easily search for products and view them categorized,		
So that	I can quickly find what I'm looking for without browsing through unrelated items.		
Acceptance criteria	Given I'm on the app homepage, When I use the search bar, Then I should see relevant product results based on my search query. Given I'm browsing products, When I look at the navigation or filter options, Then I should see products categorized (e.g., "Dairy", "Electronics").		
Test Case	Test Case: Search for items Steps: 1. Input an item into the search bar 2. Check if system returns related items Expected output: System only shows items related to the search term.		

Story #6:	**Price Comparison**	**Priority**	
		Effort	
As a	Online Shopper		
I want	to be able to compare the price of a product at different supermarkets		
So that	I can save on spending		
Acceptance criteria	Given I am an online shopper, when I click on a searched product, Then I get a list of all prices for that product in supermarkets around me.		
Test Case	Test Case: Compare prices for items from different stores Steps: 1. Search for an item 2. Click the item from the search page • Expected result: The prices for the item from different stores are displayed.		

Story #7:	**Saved Items Page**	**Priority**	
		Effort	
As a	logged in online shopper		
I want	to be able to view a list of my saved items		
So that	I can make changes to my selection		
Acceptance criteria	Given I am a logged in online shopper, when I click on the saved option on the navigation bar, Then I get a list of all products I have added to my list along with their respective loctions, prices and quantities.		
Test Case	Test Case: Verify a logged-in online shopper can access and view their list of saved items, including product details. Preconditions: User is logged in to an account Steps: 1. Visit the saved items page Expected result: System returns all the items saved by this account. 1. Item Checkout		

Story #8:	**Home Page**	**Priority**	
		Effort	
As a	deal-seeking shopper		
I want	to have quick access to the latest deals, discounts, and special offers on the website's home page		
So that	I can save money and take advantage of limited-time promotions		
Acceptance criteria	Given I open the website's home page, When I scroll or navigate to the "Ongoing offers" section, Then I should be presented with a curated list of current deals and offers.		
Test Case	Description: Home page has good design for it Steps: 1. Open the website Expected outcome: Good designed home page that clearly shows all the features.		

Story #9:	**Filter Search**	**Priority**	
		Effort	
As a	price-conscious shopper		
I want	to filter all the search results for an item to less than a certain amount		
So that	I can stay in my budget.		
Acceptance criteria	Given that I am on the Search results page, When I click on the the "Filters" option Then I can see the option to filter the item by price		
Test Case	**Filter Search Results by Price** • Description: As a price-conscious shopper, I want to filter search results for an item to less than a certain amount to stay in my budget. **Test Case: Filter Search Results by Price** • Scenario: The user searches for a specific item and applies a price filter. • Steps to Reproduce: a. Enter the item's name in the search bar. b. Apply a price filter of less than $50. • Expected Outcome: The search results should only display items with a price less than $50.		

Story # 10	Filter Search	Priority	
		Effort	
As a	online shopper who does not like the quality of one brand,		
I want	to filter all the search results for an item to show the item in all brands except that brand,		
So that	I can save time while searching.		
Acceptance criteria	Given that I am on the Search results page, When I click on the the "Filters" option Then I can see the option to filter the item by brand.		
Test Case	**Filter Search Results by Brand** • Description: As an online shopper who does not like the quality of one brand, I want to filter search results for an item to show the item in all brands except that brand to save time while searching. **Test Case: Filter Search Results by Excluding Brand** • Scenario: The user searches for a specific item and excludes a particular brand. • Steps to Reproduce: a. Enter the item's name in the search bar. b. Apply a filter to exclude a specific brand. • Expected Outcome: The search results should not display items from the excluded brand.		

Story # 11	Category search	Priority	
		Effort	
As a	user looking for specific products		
I want	to navigate through a category dropdown on the website's home page to easily filter and explore different product categories and subcategories,		
So that	I can quickly find the items I need.		
Acceptance criteria	Given I open the app's home page, When I interact with the category dropdown menu, Then I should see a list of main categories, including "All," "Specials," "Grocery," "Meat," "Dairy," and more.		
Test Case	Description: Verify that a user can view items organized by their respective categories on the category page and using filters. Preconditions: User is on the category page. Steps: 1. Navigate to a specific category page (e.g., "Electronics," "Clothing," "Home Decor"). 2. Observe the displayed items within the chosen category. Expected Result: The user can view a list of items that belong to the selected category, and only items that match the filters selected.		

Story # 12	**Category search**	**Priority**	
		Effort	
As a	health-conscious shopper		
I want	to be able to easily filter and explore organic products within specific categories using the app's category dropdown,		
So that	I can make healthier choices while shopping.		
Acceptance criteria	Given I open the app's home page, When I interact with the category dropdown menu, Then I should see an option to filter for "Organic" products within each main category.		
	Description: Check that the category section is working. Preconditions: User is on the category page. Steps: 1. Choose catogory from menu 2. Get all items in that catogory Expected Result: User get items in that category.		

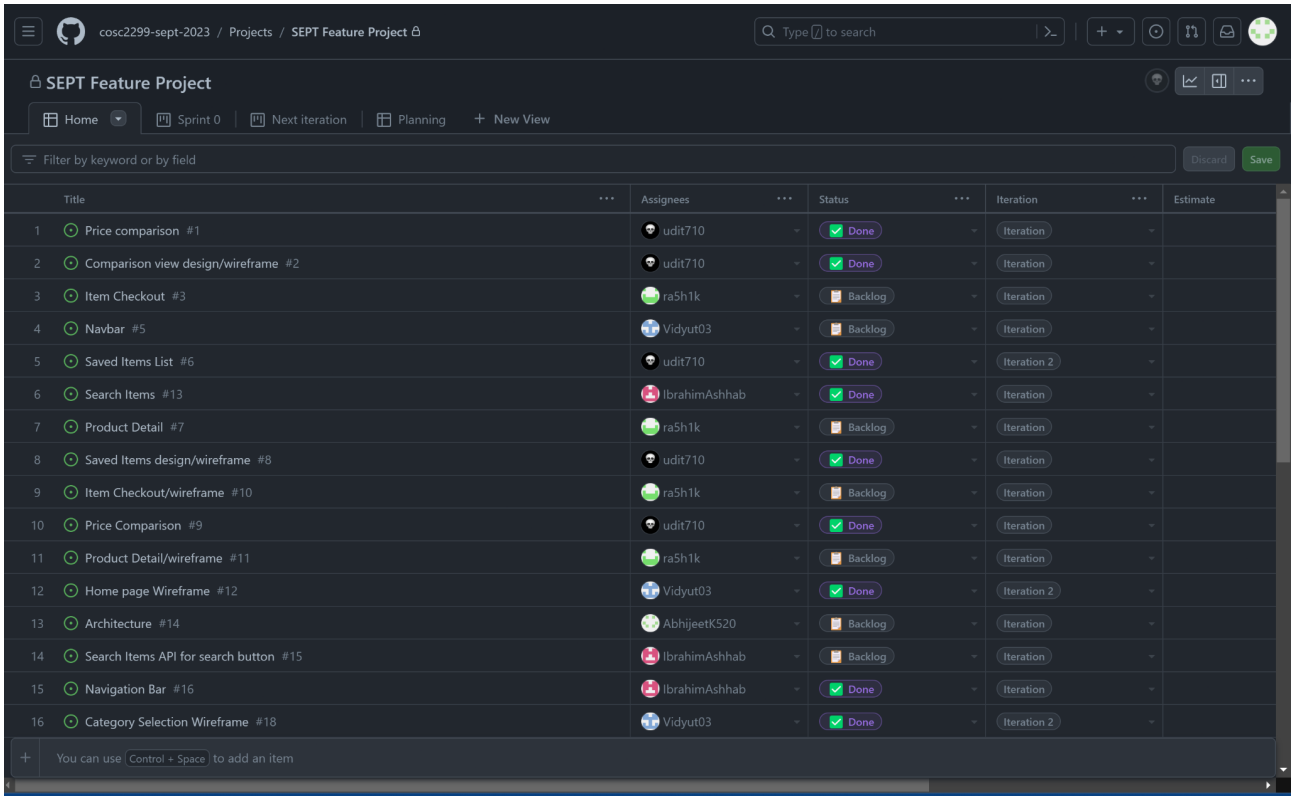
Story #13	**Product Details Section**	**Priority**	
		Effort	
As a	user allergic to certain allergens,		
I want	to check the ingredients list and get information on the allergens contained in a product		
So that	I can buy the safe product for myself.		
Acceptance criteria	Given that I can see a searched product, When I click on the product and scroll to the "Product Details" section, Then I can see the entire ingredients list and allergen warnings for the same.		
Test Case	Description: Show details of a product to the user Steps: 1. Click on product to visit its product page. 2. Observe the displayed product details, including: Expected Result: The user is able to view the Product name, Description, Images of the product, Price, Shipping information, Ingrediants/Contents and Customer reviews specified product information on the product detail page.		

Story #14	**Product Details Section**	**Priority**	
		Effort	
As a	shopper confused about the storage of various products,		
I want	to know the storage instructions for a product		
So that	I can store it in the correct manner.		
Acceptance criteria	Given that I am on the search results page, When I click on a product and scroll to the "Product Details" section, Then I can see the storage instructions provided by the seller for that product.		
Test Case	**Test Case: Display Storage Instructions for a Product** • Scenario: The user selects a specific product and seeks its storage instructions. • Steps to Reproduce: a. Browse the product catalog. b. Select a product of interest. c. Look for storage instructions on the product details page. • Expected Outcome: The product details page should clearly display the storage instructions for the selected product.		

Story #15	**Item Checkout Page**	**Priority**	
		Effort	
As a	online shopper,		
I want	to be able to view my selected items one last time during checkout,		
So that	I know which items are from which store(s).		
Acceptance criteria	Given that I am on the "Saved Items" page, When I click on the the "Checkout" option Then I am taken to the checkout page where I can see all the items and their price(s) listed store wise.		
Test Case	Description: To buy the item required Preconditions: User is on the checkout page with an item selected. Steps: 1. Select "Purchase" button. 2. Enter address 3. Choose payment method 4. Confirm payment details Expected Result: The user's order will go through and be delivered to them.		

Story #17	**User Signup**	**Priority**	
		Effort	
As a	user who wants to make purchases		
I want	to create a new account		
So that	I can save items to order		
Acceptance criteria	Given that I am on the sign up page When I input details and hit submit Then I my account is created and added to database.		
Test Case	**Check if adding properly in backend** • Check if the repository is adding to the database using a mock service **Check if form exist on fronntend** • Check if the form element can post data in frontend		

Story # 16	**User login**	**Priority**	
		Effort	
As a	user who wants to purchase items for cheap		
I want	to log into my account		
So that	i can make orders		
Acceptance criteria	Given that I am on the login page, When I enter my details and hit submit Then I can see the option to filter the item by brand.		
Test Case	**Get user account information in backed** • Use a mock user to see if the login page returns the same user as used for the mock. Check if form works on frontend • Check if the form element exists and posts data in frontend		


cosc2299-sept-2023 / Projects / SEPT Feature Project
Type / to search
SEPT Feature Project
Home
Sprint 0
Next iteration
Planning
New View
Filter by keyword or by field
Discard
Save
Title
Assignees
Status
Iteration
Estimate
1 Price comparison #1 udit710 Done Iteration
2 Comparison view design/wireframe #2 udit710 Done Iteration
3 Item Checkout #3 ra5h1k Backlog Iteration
4 Navbar #5 Vidyut03 Backlog Iteration
5 Saved Items List #6 udit710 Done Iteration 2
6 Search Items #13 IbrahimAshhab Done Iteration
7 Product Detail #7 ra5h1k Backlog Iteration
8 Saved Items design/wireframe #8 udit710 Done Iteration
9 Item Checkout/wireframe #10 ra5h1k Backlog Iteration
10 Price Comparison #9 udit710 Done Iteration
11 Product Detail/wireframe #11 ra5h1k Backlog Iteration
12 Home page Wireframe #12 Vidyut03 Done Iteration 2
13 Architecture #14 AbhijeetK520 Backlog Iteration
14 Search Items API for search button #15 IbrahimAshhab Backlog Iteration
15 Navigation Bar #16 IbrahimAshhab Done Iteration
16 Category Selection Wireframe #18 Vidyut03 Done Iteration 2
You can use Control + Space to add an item

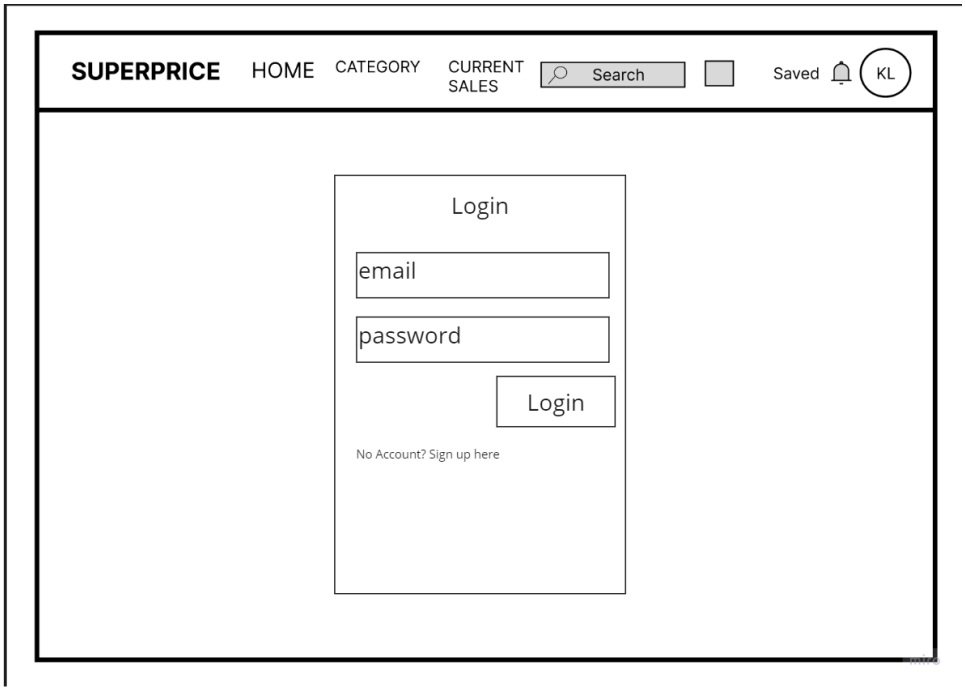
New Additions in Milestone 2

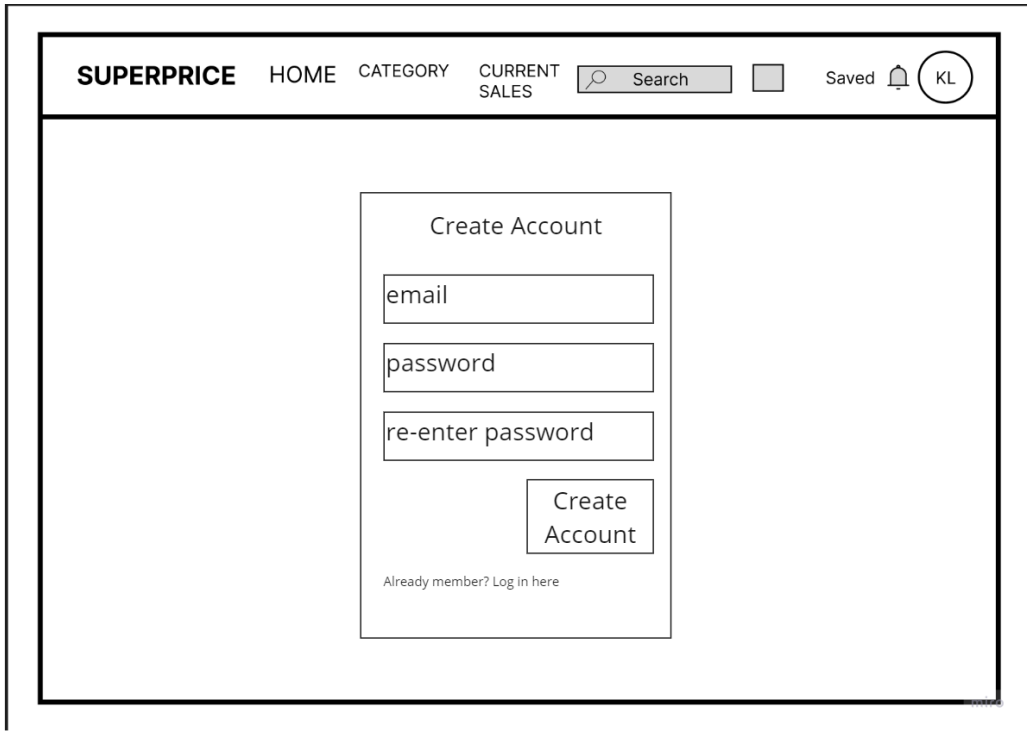
User Stories:

Story #16	**User login**	**Priority**	
		Effort	
As a	user who wants to purchase items for cheap		
I want	to log into my account		
So that	i can make orders		
Acceptance criteria	Given that I am on the login page, When I enter my details and hit submit Then I can see the option to filter the item by brand.		
Test Case	**Get user account information in backed** • Use a mock user to see if the login page returns the same user as used for the mock. Check if form works on frontend • Check if the form element exists and posts data in frontend		

Story #17	**User Signup**	**Priority**	
		Effort	
As a	user who wants to make purchases		
I want	to create a new account		
So that	I can save items to order		
Acceptance criteria	Given that I am on the sign up page When I input details and hit submit Then I my account is created and added to database.		
Test Case	**Check if adding properly in backend** • Check if the repository is adding to the database using a mock service **Check if form exist on fronntend** • Check if the form element can post data in frontend		

Wireframes:





Summary of new requirements/code/tests:

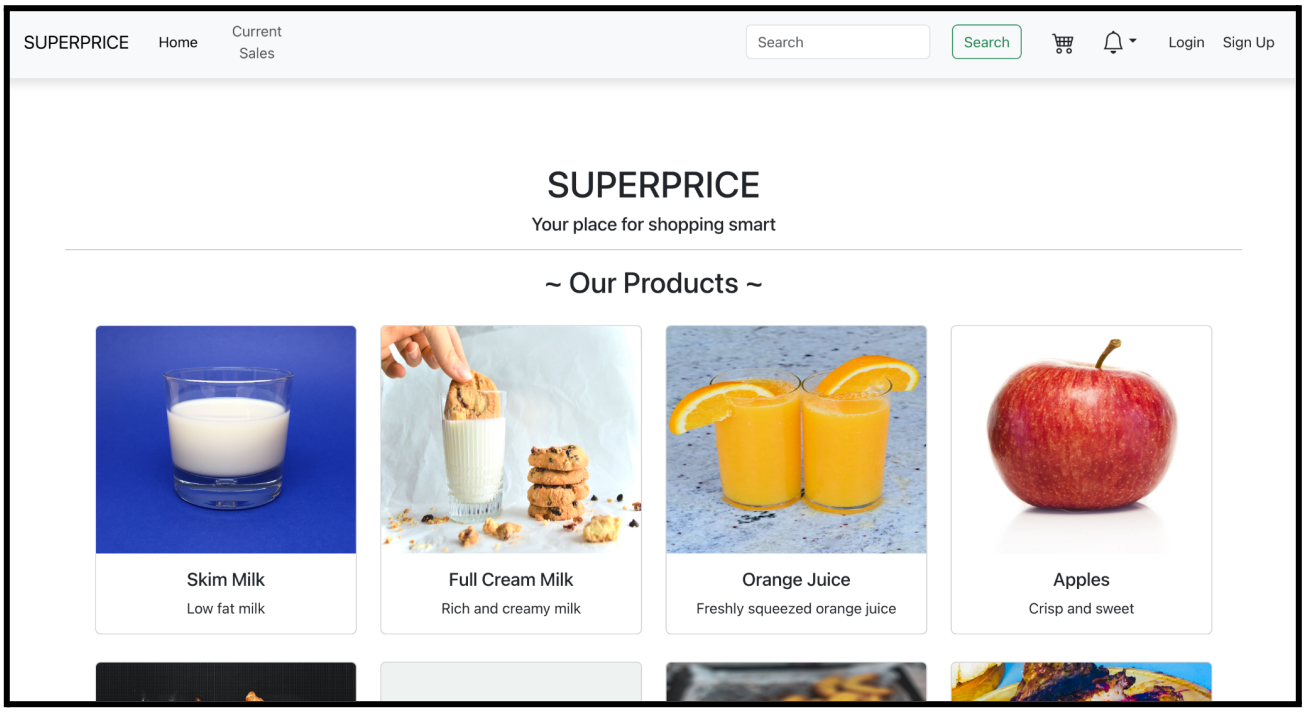
1. Frontend:
    - React init files (#39 PR)
    - Navigation Bar (#40 PR)
    - Search Results page (#43 PR)
    - Product details page components (#44 PR)
    - Price comparison components (#45 PR)
    - Routing for search page (#47 PR)
    - Footer (#50 PR)
    - Home Page (#51 PR)
    - Routing for Home Page (#54 PR)
    - Integrate product API data (#108 PR)
    - Integrate search product data (#129 PR)
    - Product review components (#124 PR)
    - Offers based on discounts in Home Page (#138 PR)
    - Product Details routing (#137 PR)
    - Updated the UI for product Details Page (#137 PR)
2. Backend:
    - Product API endpoints (#48 PR)
    - Image Data (#48 PR)
    - Review API endpoints (#86 PR)

- Orders API endpoints (#89, 91, 92 PR)
- Data loading for product details page from backend to frontend (#108 PR)
- Data loading for search product page from backend to frontend (#129 PR)
- User API endpoints (#111 PR)
- Category API (#122 PR)
- Product Search API endpoints (#118 PR)

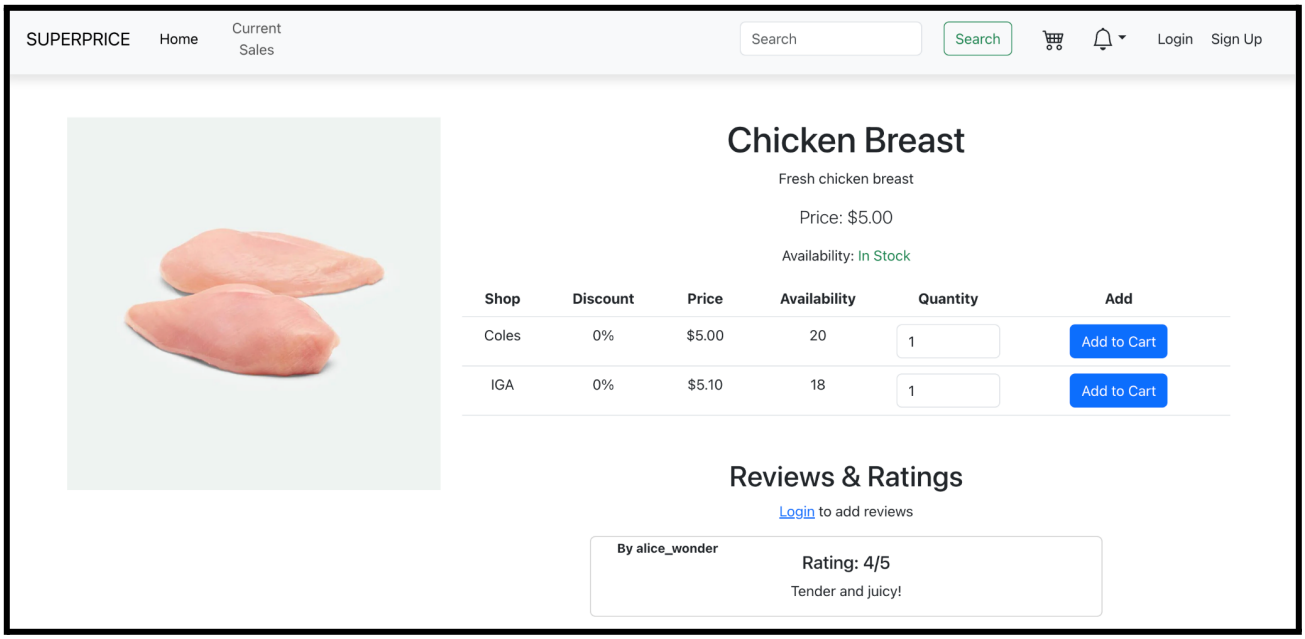
3. Model:
   - Updated database from MySQL to H2 (#52 PR)
   - Product price Database (#88 PR)
   - Created SubCategory Model (#123 PR)
   - Updated Product Model (#123 PR)
   - Created Category Model(#126 PR)

4. Testing:
   - Added frontend testing library (#71 PR)
   - Orders API testing (#94 PR)
   - Testing search results page (#95 PR)
   - Testing for Review API (#86 PR)
   - Testing for User API (#111 PR)

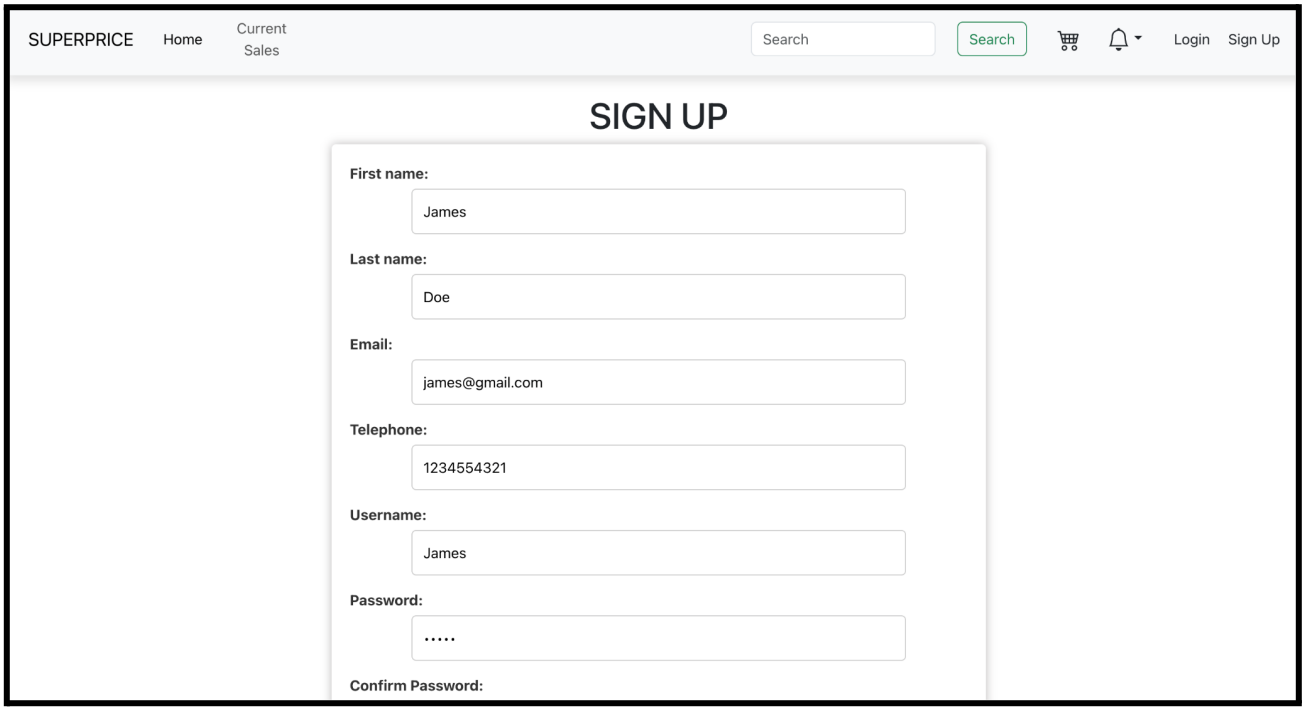
11. Web Pages from final Product:



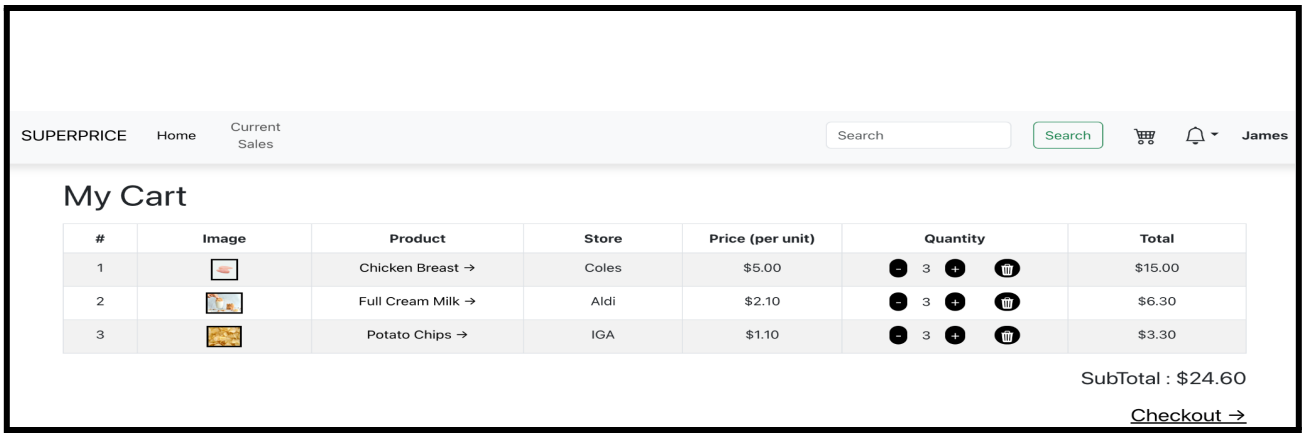
Home Page



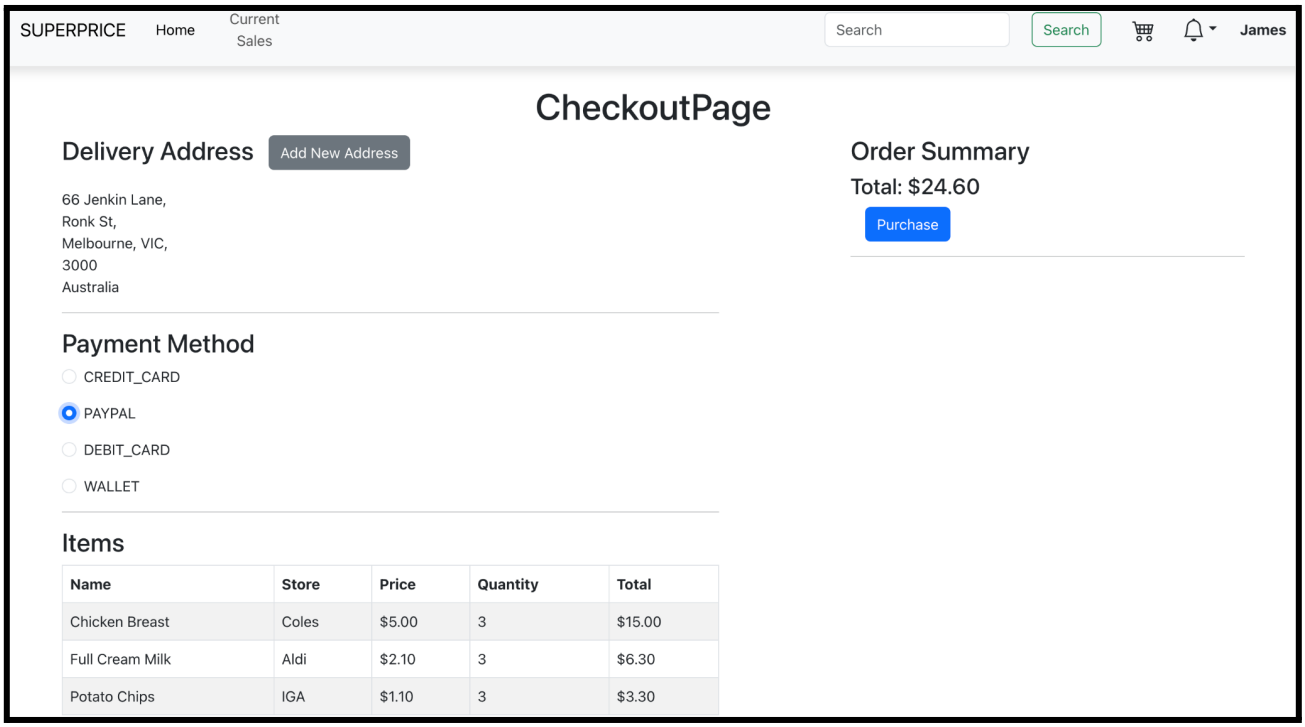
Product Details Page



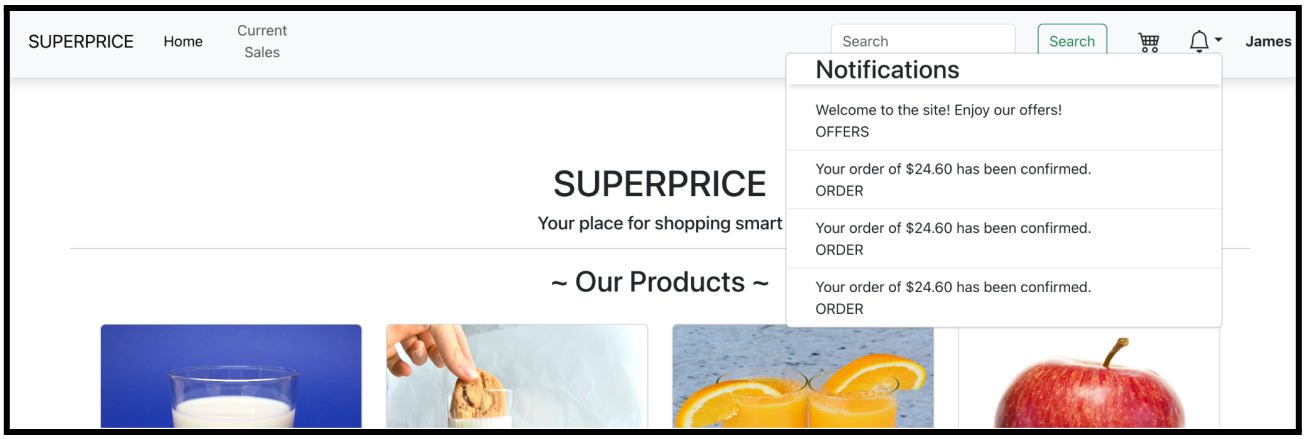
Sign Up Page



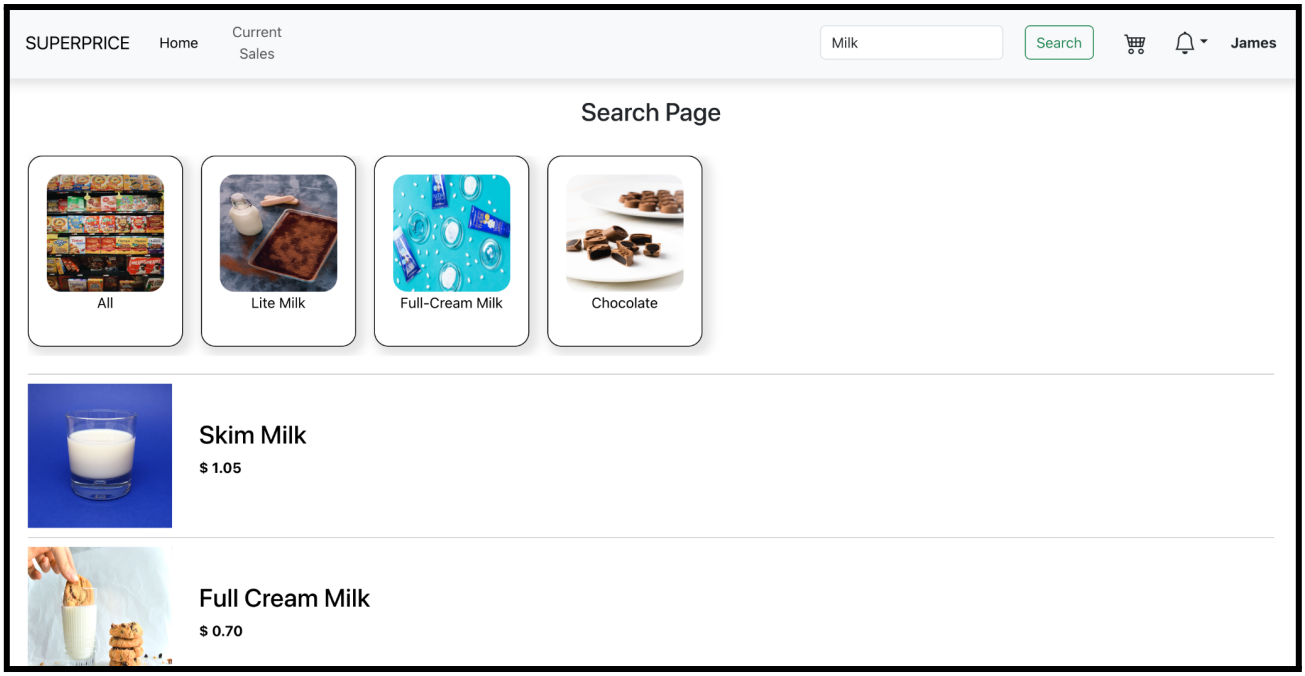
Cart Items Page



Checkout Page



Notifications



Search Results Page