SMART GROCERY

Mini Project Report

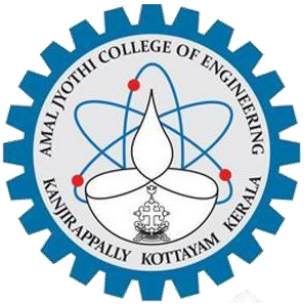
Submitted by

SACHIN SAM JACOB

Reg. No.: AJC23MCA-2054

In Partial fulfillment for the Award of the Degree of

MASTER OF COMPUTER APPLICATIONS (MCA)



AMAL JYOTHI COLLEGE OF ENGINEERING AUTONOMOUS

KANJIRAPPALLY

[Approved by AICTE, Accredited by NAAC, Accredited by NBA.
Koovappally, Kanjirappally, Kottayam, Kerala – 686518]

2024-2025

DEPARTMENT OF COMPUTER APPLICATIONS

AMAL JYOTHI COLLEGE OF ENGINEERING AUTONOMOUS

KANJIRAPPALLY

CERTIFICATE

This is to certify that the Project report, "**SMART GROCERY"** is the bonafide work of **SACHIN SAM JACOB (Regno: AJC23MCA-2054)** in partial fulfillment of the requirements for the award of the Degree of Master of Computer Applications under **Amal Jyothi College of Engineering Autonomous, Kanjirappally** during the year 2024-25.

Ms. Sona Maria Sebastian
Internal Guide

Mr. Binumon Joseph
Coordinator

Rev. Fr. Dr. Rubin Thottupurathu Jose
Head of the Department

DECLARATION

I hereby declare that the project report **"SMART GROCERY"** is a bonafide work done at Amal Jyothi College of Engineering, towards the partial fulfilment of the requirements for the award of the Master of Computer Applications (MCA) from Amal Jyothi College of Engineering Autonomous during the academic year 2024-2025.

Date: 06/11/2024
KANJIRAPPALLY

SACHIN SAM JACOB
Reg: AJC23MCA-2054

ACKNOWLEDGEMENT

First and foremost, I thank God almighty for his eternal love and protection throughout the project. I take this opportunity to express my gratitude to all who helped me in completing this project successfully. It has been said that gratitude is the memory of the heart. I wish to express my sincere gratitude to our Director (Administration) **Rev. Fr. Dr. Roy Abraham Pazhayaparampil** and Principal **Dr. Lillykutty Jacob** for providing good faculty for guidance.

I owe a great depth of gratitude towards our Head of the Department **Rev. Fr. Dr. Rubin Thottupurathu Jose** for helping us. I extend my whole hearted thanks to the project coordinator **Mr. Binumon Joseph** for his valuable suggestions and for overwhelming concern and guidance from the beginning to the end of the project. I would also express sincere gratitude to my guide **Ms. Sona Maria Sebastian** for her inspiration and helping hand.

I thank our beloved teachers for their cooperation and suggestions that helped me throughout the project. I express my thanks to all my friends and classmates for their interest, dedication, and encouragement shown towards the project. I convey my hearty thanks to my family for the moral support, suggestions, and encouragement to make this venture a success.

SACHIN SAM JACOB

ABSTRACT

The **Smart Grocery** project is an innovative platform designed to transform the traditional grocery shopping experience through modern technology and user-centric features. This platform offers a convenient and efficient online grocery shopping solution, addressing the growing demand for personalized, fast, and seamless services. By integrating key functionalities such as User Authentication, Profile Management, and a comprehensive Product Catalog, the platform allows users to securely browse and shop from a wide range of grocery items.

One of the core features is the Smart Shopping Cart, which enables real-time cart management and easy addition of products. The platform also ensures secure transactions through its Payment and Checkout module, allowing users to complete purchases with confidence. For administrators, the platform offers a dedicated Admin Panel that facilitates product updates, order management, and user account monitoring, ensuring smooth operations.

The Smart Grocery project focuses on providing a simple, intuitive, and user-friendly interface, making grocery shopping more convenient, personalized, and efficient. This platform is poised to set new standards in the online grocery industry by combining advanced features with a seamless shopping experience.

CONTENT

List of Abbreviations

- UML - Unified Modelling Language
- ORM - Object-Relational Mapping
- MVT - Model-View-Template
- MVC - Model-View-Controller
- RDBMS - Relational Database Management System
- 1NF - First Normal Form
- 2NF - Second Normal Form
- 3NF - Third Normal Form
- IDE - Integrated Development Environment
- HTML - HyperText Markup Language
- JS - JavaScript
- CSS - Cascading Style Sheets
- AJAX - Asynchronous JavaScript and XML
- JSON - JavaScript Object Notation
- API - Application Programming Interface
- UI - User Interface
- HTTP - Hypertext Transfer Protocol
- URL - Uniform Resource Locator
- PK - Primary Key
- FK - Foreign Key
- CRUD - Create, Read, Update, Delete

CHAPTER 1

INTRODUCTION

1.1 PROJECT OVERVIEW

The **Smart Grocery** project is an innovative and forward-thinking platform designed to revolutionize how consumers shop for groceries. By leveraging modern technologies such as artificial intelligence, advanced search algorithms, and real-time data processing, the platform aims to create a seamless, user-centric experience that transforms the traditional grocery shopping journey into a more efficient and personalized process.

In today's fast-paced world, the demand for online grocery shopping has seen unprecedented growth. Consumers are increasingly seeking convenience, quick delivery, and personalized shopping experiences without compromising on product quality or variety. Smart Grocery addresses this demand by offering a comprehensive online marketplace where users can browse, select, and purchase groceries from a wide range of categories, all from the comfort of their homes. The platform is designed to cater to different types of users—customers, administrators, and district operations managers—each with tailored functionalities to ensure smooth operations and superior service delivery.

The platform integrates key features such as User Authentication, which allows secure login and profile management, Product Catalog and Smart Shopping Cart, providing detailed product information and real-time cart management, and purchase history. Furthermore, Smart Grocery emphasizes efficiency through its Secure Payment and Checkout process, making it a comprehensive solution for modern grocery shopping needs.

From an operational perspective, Smart Grocery also empowers administrators and district operations managers to manage the platform effectively. Administrators oversee critical functions such as user management, product catalog updates, and overall system security, while District Operations Managers handle region-specific operations like product restocking, delivery logistics, and supplier collaborations.

The primary objective of the Smart Grocery project is to offer an all-in-one platform that enhances convenience, personalization, and reliability for consumers. By combining user-friendly features with advanced technology and integrating recipe-based meal planning, the project aims to set a new standard in the grocery shopping industry, creating a smarter, faster, and more enjoyable shopping experience.

1.2 PROJECT SPECIFICATION

The **Smart Grocery** project aims to revolutionize the grocery shopping experience by integrating cutting-edge technology to create a seamless, efficient, and personalized platform for users. The platform includes distinct roles and functionalities for various users, such as administrators, customers, and district operations managers, to ensure effective operations and a user-friendly experience.

Administrators:

- **User Management:** Administrators can manage user accounts, monitor activity, and enforce policies.
- **Product Management:** Admins have the authority to add, update, and remove products in the catalog.
- **Order Management:** Admins oversee and manage all orders placed on the platform.
- **Banner Management:** Ability to manage promotional banners displayed on the homepage.

Customers:

- **User Registration/Login:** Secure registration and login functionality with password encryption and validation.
- **Profile Management:** Users can manage their personal information, delivery addresses, and payment methods.
- **Product Browsing & Shopping:** Customers can browse categorized grocery items, apply search filters (brand, price, etc.), and access detailed product descriptions.
- **Smart Shopping Cart:** Customers can add items to their cart, manage quantity, and remove or save items for later.
- **Checkout Process:** Multiple secure payment options and an optimized checkout process, including the ability for guest checkout.
- **Provide Feedback:** Option to leave feedback and product reviews, helping inform future customers.

District Operations Managers:

- **Product Management:** District managers oversee product availability, restocking, and ensure sufficient supply for their region.
- **Order Management:** Handle order fulfillment, deliveries, and returns within their specific

regions.

- **Region Management:** Ability to add or remove delivery regions (by pincodes) to ensure efficient distribution and service coverage.

CHAPTER 2

SYSTEM STUDY

2.1 INTRODUCTION

The **Smart Grocery** project is a modern platform designed to transform the grocery shopping experience by leveraging technologies like advanced search, and real-time data. It aims to provide a convenient, personalized, and efficient shopping process.

As online grocery shopping grows in demand, Smart Grocery offers a comprehensive marketplace where users can browse, select, and purchase groceries from various categories. The platform caters to different users—customers, administrators, and district operations managers—each with tailored features to streamline operations and ensure top-notch service.

Key features include User Authentication for secure access, Product Catalog and Smart Shopping Cart for easy navigation and real-time management. The platform also offers secure payments for a seamless shopping experience.

Administrators manage user profiles and product catalogs, while District Operations Managers handle logistics. Overall, the Smart Grocery project aims to create a smarter, more convenient, and personalized shopping platform, setting a new standard in the industry.

2.2 EXISTING SYSTEM

The current grocery shopping experience, whether in physical stores or through standard e-commerce platforms, has several limitations that Smart Grocery aims to address. Traditional grocery shopping involves time-consuming tasks such as navigating aisles, comparing products manually, and managing lists, which can be particularly inconvenient for busy consumers. While many grocery chains and online retailers have introduced online shopping, they often lack advanced features like personalized recommendations, dynamic pricing, and integrated meal planning, leading to an incomplete user experience.

2.2.1 NATURAL SYSTEM STUDIED

In traditional grocery shopping, customers either visit physical stores or use basic online platforms, both of which have significant inefficiencies.

1. **Physical Grocery Shopping**: While shopping in person allows for immediate product

access, it is time-consuming, lacks personalized assistance, and often involves stock shortages. Customers also manually manage shopping lists, leading to potential oversights.

2. **Basic Online Systems**: Online grocery platforms offer limited personalization, no meal planning integration, static pricing, and minimal cart management features. They often fail to assist with dietary needs or provide smart recommendations.
3. **Customer-Supplier Interaction**: There is minimal engagement between customers and suppliers, limiting transparency, especially for organic or sustainable products.
4. **Lack of Smart Assistance**: These systems do not utilize AI or machine learning to personalize and enhance the shopping experience, resulting in a static, inefficient process.

Overall, both physical and basic online systems fall short of providing the convenience, efficiency, and personalization that consumers expect, laying the groundwork for the development of **Smart Grocery**—a platform designed to address these gaps using advanced technology.

2.2.2 DESIGNED SYSTEM STUDIED

The **Smart Grocery** platform is designed to overcome the limitations of traditional grocery shopping by offering a more personalized, efficient, and technologically advanced experience.

1. Smart Shopping Cart:

The platform provides real-time updates in the shopping cart, allowing users to manage items efficiently, save items for later, and receive alerts on price changes or low stock.

2. Secure Payment and Checkout:

The system offers multiple secure payment options, including digital wallets and credit cards, with support for guest checkout and saved payment details for convenience.

3. Admin and District Operations Manager Modules:

The platform includes tools for administrators to manage users, products, and orders, while district managers handle regional product restocking, delivery, and supplier collaborations.

2.3 DRAWBACKS OF EXISTING SYSTEM

When analyzing existing grocery shopping systems (both online and offline), a few common drawbacks can be identified, especially when compared to your innovative "Smart Grocery"

platform. Here are some of the key drawbacks of existing systems:

1. Limited Personalization

Most traditional grocery websites lack effective personalization. They do not use advanced AI algorithms to suggest products based on user preferences, past purchases, or dietary needs.

2. Inconvenient Cart Management

Many systems offer basic cart management without options like saving items for later, or quick editing. In some cases, adding, removing, or updating cart items is slow and unintuitive.

3. Lack of Advanced Search and Filtering

Search functionality in many grocery systems is often limited to simple keyword searches. Filters are not as detailed or customizable, making it difficult to find specific products based on user preferences like dietary restrictions, brands, or price range.

4. Limited Payment Options

Some systems only support a few payment options, often excluding modern payment methods like digital wallets (e.g., Google Pay, Apple Pay) or cryptocurrency.

This lack of flexibility in payment options can be a dealbreaker for some users.

5. Static Pricing

Many traditional systems offer fixed pricing, without incorporating dynamic pricing models that adapt based on stock levels, demand, or location.

Users miss out on potential deals, and retailers might lose opportunities to optimize pricing for profitability and user satisfaction.

6. Lack of Real-Time Inventory and Stock Information

Many platforms do not provide real-time stock information, meaning users may add items to their cart that are later found to be out of stock at checkout.

This creates a poor shopping experience, as users have to find alternatives or cancel part of their order.

7. Inadequate Supplier and Product Management

In systems where inventory management is suboptimal, there may be delays in restocking products or updating the availability of items.

Users may face frequent out-of-stock issues or receive outdated information about product availability.

8. Weak District-Level Operations

Existing systems may lack a district-level operations management structure, meaning

that regional demand, delivery zones, and logistics may not be optimized.Users in certain areas may experience slower delivery times, higher delivery fees, or limited product availability.

2.4 PROPOSED SYSTEM

The proposed **Smart Grocery** system aims to transform the grocery shopping experience by offering an advanced and user-friendly online platform. By incorporating modern technologies such as AI-driven personalized recommendations, a smart shopping cart, and a secure checkout process, the system ensures a seamless and efficient user journey. It is designed to cater to a wide range of consumers, providing them with the convenience of browsing, selecting, and purchasing groceries from their homes while maintaining a smooth and intuitive shopping experience.

At the heart of the system is its user-friendly interface, which allows customers to easily navigate through various product categories, access detailed product descriptions, and make informed purchasing decisions. The platform includes an enhanced search functionality with powerful filters that enable users to quickly find products based on their preferences, such as brand, price, or specific dietary needs. This ensures that the shopping process is not only straightforward but also highly customizable to each user's individual requirements.

One of the standout features of the **Smart Grocery** system is its smart shopping cart. This dynamic cart updates in real-time, allowing users to add, modify, or remove items seamlessly. It also supports saving products for future purchases, giving customers greater flexibility in managing their grocery lists. Additionally, the platform leverages AI to provide personalized recommendations based on user behaviour, purchase history, and preferences. This feature enhances the shopping experience by offering relevant product suggestions and ensuring users have easy access to items they may need.

The payment and checkout process is designed with security and simplicity in mind. The system supports multiple payment methods, ensuring that transactions are completed safely and efficiently. The streamlined checkout process reduces unnecessary steps, making it quick and easy for users to finalize their purchases.

In summary, the proposed **Smart Grocery** system addresses the limitations of traditional grocery shopping by offering an innovative, technology-driven solution that enhances convenience, personalization, and operational efficiency. The platform is designed to provide a more streamlined,

enjoyable, and reliable shopping experience for consumers while also offering robust tools for system management.

2.5 ADVANTAGES OF PROPOSED SYSTEM

The **Smart Grocery** system offers several significant advantages over traditional grocery shopping platforms, leveraging advanced technology and user-centric features to create a seamless and efficient shopping experience. Here are the key advantages:

1. Enhanced User Experience

- **Personalization**: The platform provides personalized product recommendations based on user behavior, preferences, and past purchases, making it easier for users to find relevant products.
- **Convenient Shopping**: Features like the Smart Shopping Cart and voice-activated assistance streamline the shopping process, allowing users to add products easily and manage their carts in real time.

2. Secure Payment and Seamless Checkout

- **Multiple Payment Options**: Offering a variety of secure payment methods, including credit cards, digital wallets, and bank transfers, increases convenience and caters to different user preferences.
- **Fast Checkout**: The streamlined checkout process, with options for saved payment details and guest checkout, makes transactions quick and easy.

3. Efficient District Operations Management

- **Localized Operations**: The District Operations Manager can manage regional stock levels, add/remove delivery regions, and collaborate with suppliers to ensure timely restocking and delivery in specific areas.
- **Improved Delivery Services**: By optimizing delivery zones and managing stock levels regionally, the platform ensures faster deliveries and better product availability.

4. Comprehensive Admin Panel

- **Control and Monitoring**: Admins have full control over user accounts, product listings, orders, and other operational aspects, providing a centralized way to manage the platform efficiently.
- **Security and Analytics**: Admins can monitor user activity and generate reports, which helps with decision-making and security enhancements.

5. Scalability and Flexibility

- **Adaptability**: The modular nature of the platform allows for easy scaling as the business grows. New features, regions, and products can be added with minimal disruption to the existing system.
- **Flexible Shopping Options**: With both desktop and mobile-friendly interfaces, users can access the platform from any device, enhancing accessibility.

6. Sustainability and Efficiency

- **Reduced Waste**: The ability to manage stock levels in real time helps reduce waste by ensuring products are sold or restocked based on demand.
- **Optimized Delivery**: By optimizing delivery routes and restocking strategies, the platform can reduce carbon footprints and ensure timely deliveries, promoting more sustainable operations.

7. Increased Customer Retention

- **Loyalty**: The combination of personalized recommendations, custom deals, efficient customer support, and tailored meal planning increases customer satisfaction, which leads to higher retention rates.
- **Engagement**: Features like recipe suggestions, smart recommendations, and dynamic pricing keep users engaged and returning to the platform for their grocery needs.

8. Feedback and Reviews

- **User Engagement**: Users can provide feedback and reviews on products and services, which not only helps improve the platform but also creates a sense of community and trust.
- **Improvement**: Admins can use customer feedback to make data-driven improvements, refining the platform to meet evolving user expectations.

CHAPTER 3
REQUIREMENT ANALYSIS

3.1 FEASIBILITY STUDY

The primary purpose of conducting a feasibility study is to comprehensively assess whether the proposed project can successfully meet the organization's objectives in terms of available resources, labor, and time. This crucial study allows the developers and decision-makers to gain valuable insights into the project's potential viability and prospects. By carefully examining various aspects of the proposed system, such as its impact on the organization, its ability to fulfill user requirements, and the optimal utilization of resources, a feasibility study helps in determining the project's feasibility and potential success.

The assessment of the proposed project's feasibility involves multiple dimensions, each playing a critical role in the decision-making process.

- Technical Feasibility
- Behavioral Feasibility
- Economic Feasibility

A well-conducted feasibility study provides valuable insights to decision-makers, allowing them to make informed judgments about the project's potential success. It assists in identifying potential risks, challenges, and opportunities associated with the proposed endeavor, enabling stakeholders to devise effective mitigation strategies.

3.1.1 Economic Feasibility

As a student project, the economic feasibility of the **Smart Grocery** platform focuses on utilizing available resources while ensuring a practical learning experience. The cost estimation is modest, considering that the project development will primarily rely on open-source technologies and free development tools, minimizing the need for significant financial investment. The primary costs associated with the project include minimal cloud storage or hosting services, software licenses (if required), and infrastructure for deployment and testing.

The market analysis considers the potential growth in the online grocery shopping industry, but the project's focus remains on delivering a functional and user-friendly platform within the academic context. While risks such as technical challenges or integration issues are acknowledged, the project emphasizes overcoming these obstacles as learning opportunities rather than focusing on high financial returns.

The project can be funded through self-financing or utilizing university-provided resources such as access to servers or development environments. Financial projections are based on the limited scope and duration of the student project, with the primary objective of gaining technical knowledge and developing a proof-of-concept that could be expanded in the future.

In conclusion, the economic feasibility of the **Smart Grocery** project is highly achievable within the confines of a student project, emphasizing educational outcomes, technical learning, and resource-efficient development.

3.1.2 Technical Feasibility

The technical feasibility of the **Smart Grocery** project assesses the practicality of developing the platform using the available technical resources, tools, and team expertise. As a student project, it examines whether the required technologies—such as the MERN stack (MongoDB, Express.js, React, Node.js), cloud hosting are accessible and implementable within the project's scope. The development team is proficient in these technologies, ensuring that the platform can be built effectively.

The study evaluates integration possibilities for third-party services like payment gateways, scalability to handle increasing users and product data, and security measures to protect sensitive information. It also considers the infrastructure required, such as cloud storage and server capabilities, to ensure smooth operation.

Additionally, the development timeline, availability of technical support, and thorough testing processes are factored in to ensure the platform is reliable and functional. Overall, the technical feasibility confirms that the **Smart Grocery** project can be developed within the given constraints, leveraging the team's skills and available resources effectively.

3.1.3 Behavioral Feasibility

The behavioral feasibility of the **Smart Grocery** project focuses on assessing how well the platform aligns with the behaviours and needs of its target users, primarily grocery shoppers and administrators. The platform is designed to enhance convenience, efficiency, and personalization in grocery shopping, addressing common consumer pain points like time consumption, product availability, and lack of tailored shopping experiences.

The system requirements have been structured around user inputs, such as product searches, shopping cart management, and secure payments, ensuring that these processes are simple and

intuitive. The platform's output, including real-time product recommendations and order tracking, is aimed at improving the shopping experience, making it faster and more personalized.

User-friendly interfaces and efficient procedures for user registration, profile management, and customer support are central to ensuring a positive user experience. The system's design encourages seamless interactions, minimizing the need for complex actions from users while offering clear guidance throughout their shopping journey.

By focusing on user behaviour and providing an intuitive, streamlined shopping process, **Smart Grocery** ensures a high level of user engagement and satisfaction, making it an accessible and efficient solution for modern grocery shopping.

3.1.4 Feasibility Study Questionnaire

1. **Project Overview?**

   The project aims to develop an online platform called Smart Grocery that revolutionizes the grocery shopping experience. It provides a seamless, user-friendly interface for consumers to browse products, manage their shopping carts, make secure payments, and track orders. The goal is to enhance convenience and efficiency in grocery shopping while fostering a strong community around healthy eating and local sourcing.

2. **To what extent is the system proposed for?**

   The proposed system, Smart Grocery, aims to serve as a comprehensive online grocery shopping platform for consumers and suppliers. It will initially focus on essential functionalities, such as user registration, product catalog management and secure checkout. As a student project, it is designed with scalability in mind, allowing for future enhancements like subscription services, community features, and advanced analytics.

3. **Specify the Viewers/Public which is to be involved in the System?**

- **Consumers:** Individuals seeking an efficient and personalized grocery shopping experience.
- **Suppliers:** Providers of grocery products who manage their inventory and fulfill orders.
- **District Operations Managers:** Responsible for monitoring inventory levels and making restock requests.
- **Administrators:** Staff members overseeing platform management, user support, and

content moderation.

4. **List the modules in your system:**

- **User Management Module:** Handles user registration, login, profile management, and preference customization.
- **Product Catalog Module:** Displays a comprehensive list of grocery items with search and filtering options.
- **Smart Shopping Cart Module:** Allows users to add, edit, and remove items in their cart with real-time updates.
- **Secure Payment Module:** Facilitates multiple secure payment options and a streamlined checkout process.
- **Admin Panel Module:** Enables administrators to manage users, products, orders, and site settings.
- **Feedback and Review Module:** Lets users provide feedback on products and services, enhancing community engagement.
- **District Operation Manager Module:** Enables to manage products, pincodes, orders and restock of products of a particular region.

5. **Identify the users in your project?**

The viewers/public involved in the Smart Grocery system include:

- **Consumers:** Users who shop for groceries, manage their accounts, and interact with product recommendations and reviews.
- **Suppliers:** Businesses that provide grocery products, manage inventory, and fulfill orders on the platform.
- **District Operations Managers:** Individuals responsible for monitoring product availability and initiating restock requests.
- **Administrators:** Staff who manage the platform, oversee user accounts, handle customer support, and monitor supplier activity.
- **General Public:** Individuals who may browse the platform for information without creating an account.

6. **Who owns the system?**

   The Smart Grocery platform is owned and managed by the project's administrators and the associated organization.

7. **System is related to which firm/industry/organization?**

   The system is related to the grocery retail industry, serving consumers, suppliers, and operational managers to enhance the grocery shopping experience.

8. **Details of person that you have contacted for data collection?** Kiranjith K S, Grocery Shop Staff.

9. **Questionnaire to collect details about the project? (Min 10 questions, include descriptive answers, attach additional docs (e.g., Bill receipts), if any?)**

   a. **Are there any significant costs associated with developing Smart Grocery as part of the project work?**

      No, the proposed system is being developed as part of a student project, so there are no significant manual costs incurred in its development.

   b. **What is the estimated cost of hardware and software required for Smart Grocery?**

      All necessary hardware and software resources, such as servers and development tools, are already available to the student team, making the project cost-effective.

   c. **Are there any additional costs for operational expenses, such as maintenance or server hosting?**

      No, operational expenses are kept minimal, as the project is designed to utilize existing resources and operate within the limits of the student project framework.

   d. **Is the project feasible within the limits of current technology?**

      Yes, the project is entirely feasible within current technological capabilities, focusing on creating a robust online grocery shopping platform that leverages available technologies.

e. What technical challenges have been identified during the investigation?

The investigation did not uncover any significant technical challenges that would impede the development of Smart Grocery.

f. Can the technology be easily applied to current problems in grocery shopping?

Yes, the technology can be effectively applied to contemporary issues in grocery shopping, providing a streamlined and user-friendly experience for consumers.

g. Does the technology have the capacity to handle the required functionalities of Smart Grocery?

Yes, the technology is capable of supporting the functionalities outlined in the project, taking into account the expected user interactions and data volume.

h. Is the required technology readily available and accessible to the student team for developing Smart Grocery?

Yes, the necessary technology and development resources are readily accessible to the student team, ensuring smooth project execution.

i. Does the student team possess the necessary technical skills and knowledge to design and develop the platform effectively?

Yes, the student team is equipped with the necessary technical skills and knowledge to effectively design and develop Smart Grocery, aligning with the project's objectives.

j. Are the infrastructure requirements for Smart Grocery, such as servers and hosting services, feasible and within the project's scope?

Yes, the infrastructure requirements are feasible within the project's scope, considering the planned scale of the platform.

k. Will users receive adequate support while using Smart Grocery?

Yes, Smart Grocery is committed to providing users with comprehensive support options, including FAQs, customer service, and live chat features to enhance user experience.

l. **Will users be exposed to any harmful elements or content while using Smart Grocery?**

No, Smart Grocery has been designed with user safety in mind, implementing measures to ensure a secure environment free from harmful content.

m. **Does Smart Grocery offer user-friendly features and an intuitive interface?**

Yes, Smart Grocery focuses on delivering a user-centric design with intuitive features, allowing users to navigate the platform effortlessly.

n. **Is there a mechanism in place for users to share their thoughts and suggestions?**

Absolutely, Smart Grocery encourages user feedback through various channels, ensuring that user suggestions are heard and incorporated into future improvements.

o. **What additional features or enhancements do users suggest for the Smart Grocery platform?**

User input is actively sought to identify potential enhancements or new features that could further improve their shopping experience.

3.1 SYSTEM SPECIFICATION

3.2.1 Hardware Specification

Processor - Ryzen 7 6800H

RAM - 1 6 G B

Hard disk - 5 1 2 G B

3.2.2 Software Specification

Front End - React Js

Back End - Node Js, Express Js

Database - MongoDB

Client on PC - Windows 7 and above.

Technologies used - JS, CSS, JSON

3.3 SOFTWARE DESCRIPTION

3.3.1 React Js

React.js (commonly referred to as React) is a popular JavaScript library used for building user interfaces, particularly single-page applications where efficiency and user experience are key. Developed and maintained by Facebook, React allows developers to create large web applications that can update and render efficiently in response to changing data, without requiring a page reload. It is component-based, meaning developers can break down complex UIs into smaller, reusable pieces called components, each handling its own state and logic.

One of the key features of React is the use of a virtual DOM (Document Object Model), which improves performance by only re-rendering the parts of the page that actually change. Instead of updating the entire DOM every time something changes, React creates a virtual representation of the DOM in memory. When data changes, React compares this virtual DOM with the actual DOM and only updates the specific elements that need to be changed, making the app faster and more efficient.

React also emphasizes declarative programming, which makes it easier for developers to understand how the app will behave. With a declarative syntax, developers describe what the UI should look like for a given state, and React handles the rendering. This leads to cleaner, more predictable code, especially in larger applications.

Additionally, React is highly versatile, working well with other libraries or frameworks like Redux for state management or Next.js for server-side rendering. Its ecosystem is vast, with a rich collection of tools, libraries, and community resources that enable developers to create dynamic and robust web applications. React's component-based architecture, along with its high performance and flexibility, makes it one of the most popular choices for front-end development today.

3.3.2 MongoDB

MongoDB is a popular NoSQL database known for its flexibility, scalability, and ease of use, particularly in applications that require handling large volumes of unstructured or semi-structured

data. Unlike traditional relational databases that store data in tables and rows, MongoDB stores data in flexible, JSON-like documents called BSON (Binary JSON), making it highly adaptable for a wide range of data types and formats. This document-oriented approach allows for more natural data modeling, especially in use cases where the structure of the data might change over time, such as in modern web applications.

One of the key strengths of MongoDB is its horizontal scalability, which makes it well-suited for large-scale applications with rapidly growing data. MongoDB's architecture allows for easy distribution of data across multiple servers (sharding), ensuring high availability and performance even as the database grows. This scalability is particularly useful in cloud-based and distributed systems, where data needs to be managed across different regions and instances.

MongoDB also offers robust querying capabilities. Developers can perform complex queries, filtering, and aggregations on the data, even with its non-relational structure. Its indexing features improve query performance, and it supports a wide range of operations, including text search, geospatial queries, and real-time analytics. Additionally, MongoDB provides built-in replication and high availability through its replica set feature, ensuring that data is continuously backed up and available even in case of hardware failures.

3.3.3 Express and Node Js

Node.js is a powerful, open-source JavaScript runtime environment that allows developers to run JavaScript code outside of the browser. Built on Google Chrome's V8 JavaScript engine, Node.js is designed for building fast, scalable, and efficient server-side applications. It operates using an event-driven, non-blocking I/O model, which makes it particularly well-suited for handling multiple simultaneous requests, real-time applications, and applications with heavy I/O operations, such as APIs or chat applications.

One of Node.js's key features is its asynchronous, single-threaded architecture. Unlike traditional server-side environments that use multi-threaded models to handle concurrent requests, Node.js handles all requests in a single thread using asynchronous callbacks. This enables Node.js to efficiently process thousands of requests at once, making it ideal for applications where scalability and speed are crucial. Node.js's package ecosystem, managed through npm (Node Package Manager), is another major advantage, offering access to a vast repository of open-source libraries and modules that developers can easily integrate into their projects.

Express.js is a minimalist web application framework built on top of Node.js, designed to simplify the process of building web servers and APIs. While Node.js provides the foundation for server-side JavaScript applications, Express.js offers a lightweight framework that abstracts away much of the boilerplate code involved in handling HTTP requests, routing, and middleware integration. Express is known for its simplicity, flexibility, and high performance, making it one of the most widely used frameworks for building RESTful APIs and web applications.

One of the core features of Express.js is its routing system, which allows developers to define URL routes and associate them with specific handler functions. This makes it easy to create clean, organized, and maintainable code for handling different HTTP methods (GET, POST, PUT, DELETE) and paths. Express also supports middleware, which are functions that run during the request-response cycle, enabling developers to handle tasks like authentication, logging, request parsing, and error handling in a modular way.

Together, **Node.js** and **Express.js** form a powerful combination for building full-stack JavaScript applications, with Node.js providing the server-side execution environment and Express.js simplifying the creation of web servers and APIs.

CHAPTER 4
SYSTEM DESIGN

4.1 INTRODUCTION

The initial stage of developing any engineered product or system is the design phase, which involves a creative approach. A well-crafted design plays a critical role in ensuring the successful functioning of a system. Design is defined as the process of employing various techniques and principles to define a process or system in enough detail to enable its physical realization. This involves using different methods to describe a machine or system, explaining how it operates, in sufficient detail for its creation. In software development, design is a crucial step that is always present, regardless of the development approach. System design involves creating a blueprint for building a machine or product. Careful software design is essential to ensure optimal performance and accuracy. During the design phase, the focus shifts from the user to the programmers or those working with the database. The process of creating a system typically involves two key steps: Logical Design and Physical Design.

4.2 UML DIAGRAM

UML, which stands for Unified Modeling Language, is a standardized language used for specifying, visualizing, constructing, and documenting the elements of software systems. The Object Management Group (OMG) is responsible for the creation of UML, with the initial draft of the UML 1.0 specification presented to OMG in January 1997. Unlike common programming languages such as C++, Java, or COBOL, UML is not a programming language itself. Instead, it is a graphical language that serves as a tool for creating software blueprints.

UML is a versatile and general-purpose visual modeling language that facilitates the visualization, specification, construction, and documentation of software systems. While its primary use is in modeling software systems, UML's applications are not limited to this domain. It can also be employed to represent and understand processes in various contexts, including non-software scenarios like manufacturing unit processes.

It's important to note that UML is not a programming language, but it can be utilized with tools that generate code in different programming languages based on UML diagrams. UML encompasses nine core diagrams that aid in representing various aspects of a system.

- Class diagram
- Object diagram

- Use case diagram
- Sequence diagram
- Activity diagram
- State chart diagram
- Deployment diagram
- Component diagram

4.2.1 USE CASE DIAGRAM

A use case is a tool for understanding a system's requirements and organizing them, especially in the context of creating or using something like a product delivery website. These tools are represented using "use case" diagrams within the Unified Modeling Language, a standardized way of creating models for real-world things and systems.

A use case diagram consists of these main elements:

- The boundary, which delineates the system and distinguishes it from its surroundings.
- Actors, representing individuals or entities playing specific roles within the system.
- The interactions between different people or elements in specific scenarios or problems.
- The primary purpose of use case diagrams is to document a system's functional specifications. To create an effective use case diagram, certain guidelines must be followed:
- Providing clear and meaningful names for use cases and actors.
- Ensuring that the relationships and dependencies are well-defined.
- Including only necessary relationships for the diagram's clarity.
- Utilizing explanatory notes when needed to clarify essential details.

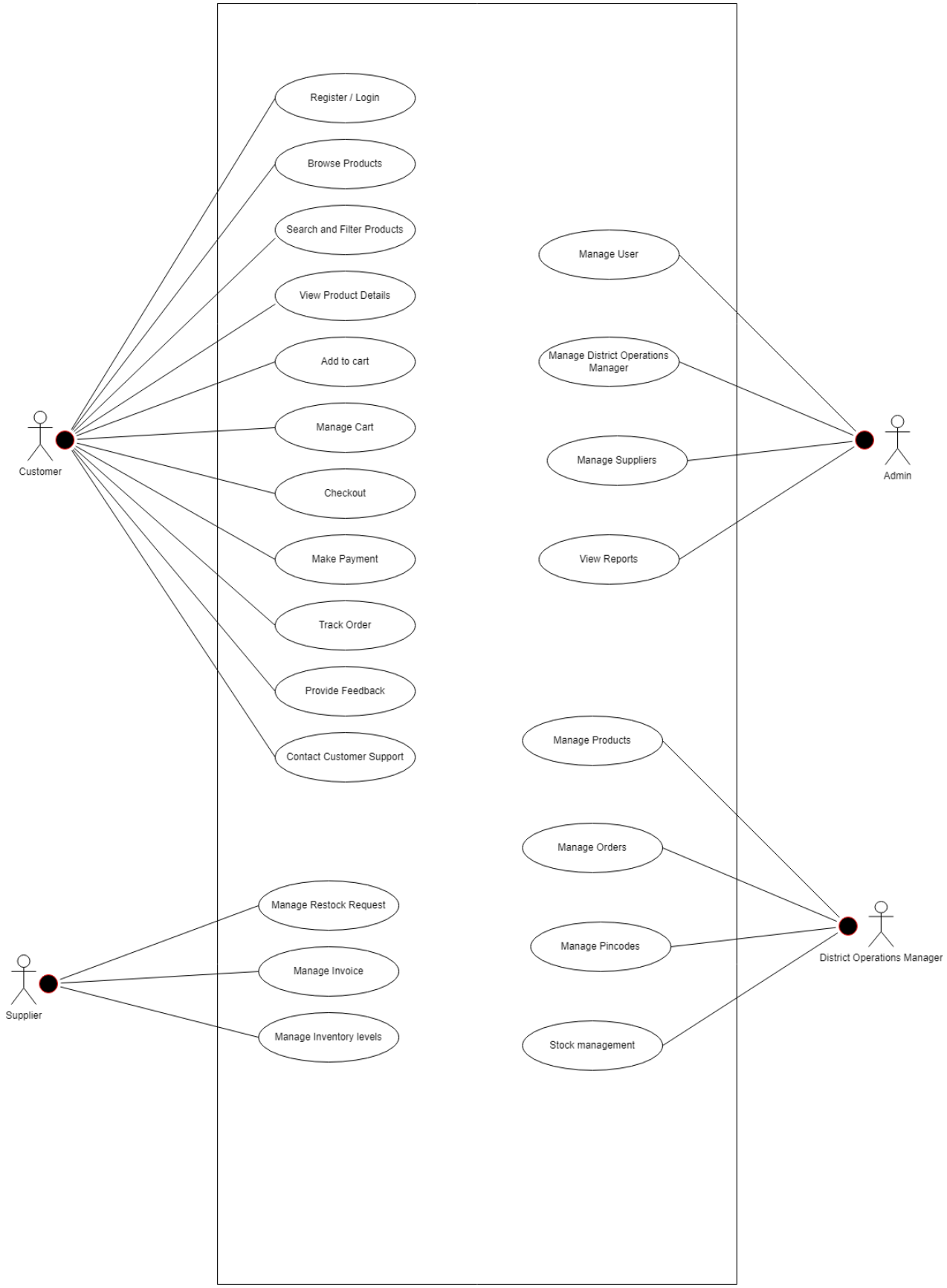


Fig 4.2.1: Use Case Diagram

4.2.2 SEQUENCE DIAGRAM

A sequence diagram illustrates the specific order in which objects interact with each other, showcasing the sequential flow of events. This type of diagram is also known as event diagrams or event scenarios. Sequence diagrams serve the purpose of elucidating how various components of a system collaborate and the precise sequence in which these actions occur. These diagrams find frequent application among business professionals and software developers, aiding in the understanding and depiction of requirements for both new and existing systems.

Sequence Diagram Notations:

i. Actors - Within a UML diagram, actors represent individuals who utilize the system and its components. Actors are not depicted within the UML diagram as they exist outside the system being modeled. They serve as role-players in a story, encompassing people and external entities. In a UML diagram, actors are represented by simple stick figures. It's possible to depict multiple individuals in a diagram that portrays the sequential progression of events.

ii. Lifelines - In a sequence diagram, each element is presented as a lifeline, with lifeline components positioned at the top of the diagram.

iii. Messages - Communication between objects is achieved through the exchange of messages, with messages being arranged sequentially on the lifeline. Arrows are employed to represent messages, forming the core structure of a sequence diagram.

iv. Guards - Within the UML, guards are employed to denote various conditions. They are used to restrict messages in the event that specific conditions are met, providing software developers with insights into the rules governing a system or process.

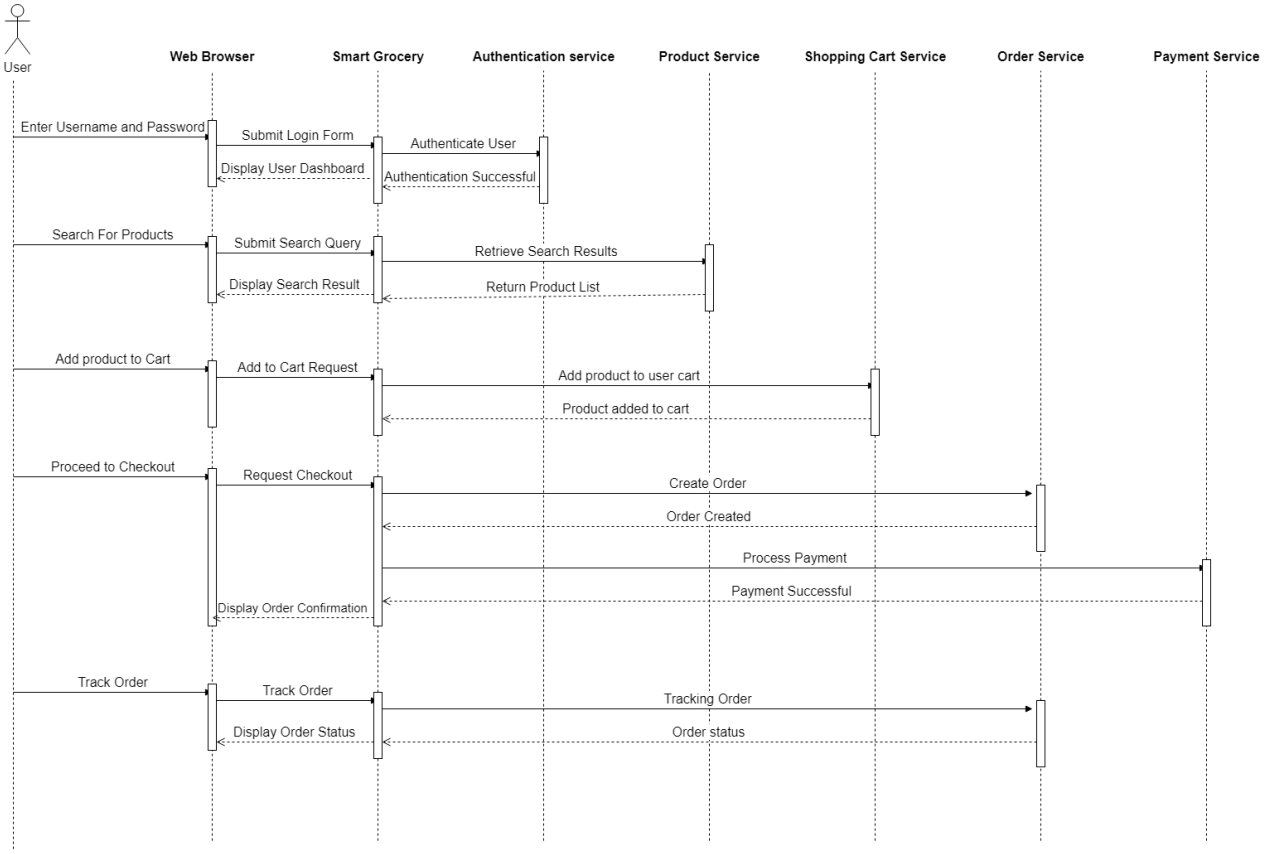


Fig 4.2.2: Sequence Diagram

4.2.3 State Chart Diagram

A state machine diagram, also known as a state chart, visually represents the various states an object undergoes within a system and the sequence in which these states are traversed. It serves as a record of the system's behavior, illustrating the collaborative functioning of a group of entities, whether it's a team, a collection of students, a large assembly, or an entire organization. State machine diagrams are a valuable method for depicting the interactions of diverse components within a system, outlining how objects evolve in response to events and elucidating the diverse conditions that each entity or component can inhabit.

Notations within a state machine diagram encompass:

- Initial state: Symbolized by a black circle, this signifies the commencement of a process.
- Final state: Representing the conclusion of a process, this is denoted by a filled circle within another circle.

- Decision box: Shaped like a diamond, it aids in decision-making by considering the evaluation of a guard.
- Transition: Whenever a change in authority or state occurs due to an event, it is termed a transition. Transitions are depicted as arrows with labels indicating the triggering event.
- State box: It portrays the condition or state of an element within a group at a specific moment. These are typically represented by rectangles with rounded corners.

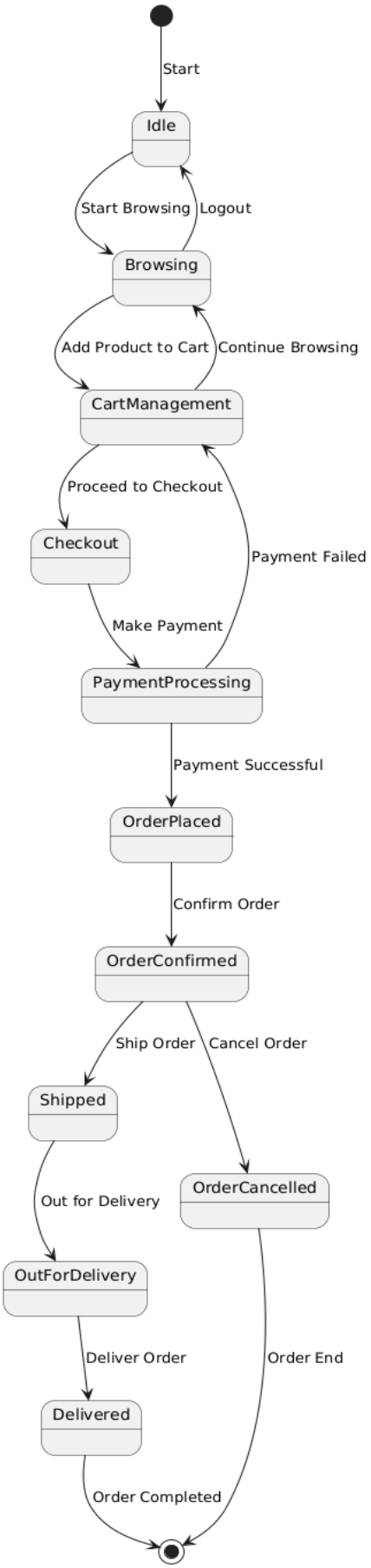


Fig 4.2.3: State Chart Diagram

4.2.4 Activity Diagram

An activity diagram is a visual representation of how events unfold simultaneously or sequentially. It aids in understanding the flow of activities, emphasizing the progression from one task to the next. Activity diagrams focus on the order in which tasks occur and can depict various types of flows, including sequential, parallel, and alternative paths. To facilitate these flows, activity diagrams incorporate elements such as forks and join nodes, aligning with the concept of illustrating the functioning of a system in a specific manner.

Key Components of an Activity Diagram include:

a) Activities: Activities group behaviors into one or more actions, forming a network of interconnected steps. Lines between these actions outline the step-by-step sequence of events. Actions encompass tasks, controlling elements, and resources utilized in the process.

b) Activity Partition/Swim Lane: Swim lanes are used to categorize similar tasks into rows or columns, enhancing modularity in the activity diagram. They can be arranged vertically or horizontally, though they are not mandatory for every activity diagram.

c) Forks: Fork nodes enable the simultaneous execution of different segments of a task. They represent a point where one input transforms into multiple outputs, resembling the diverse factors influencing a decision.

d) Join Nodes: Join nodes are distinct from fork nodes and employ a Logical AND operation to ensure that all incoming data flows converge to a single point, promoting synchronization.

Notations in an Activity Diagram include:

- Initial State: Depicting the beginning or first step in the process.
- Final State: Signifying the completion of all actions with no further progress.
- Decision Box: Ensuring that activities follow a specific path.
- Action Box: Representing the specific tasks or actions to be performed in the process.

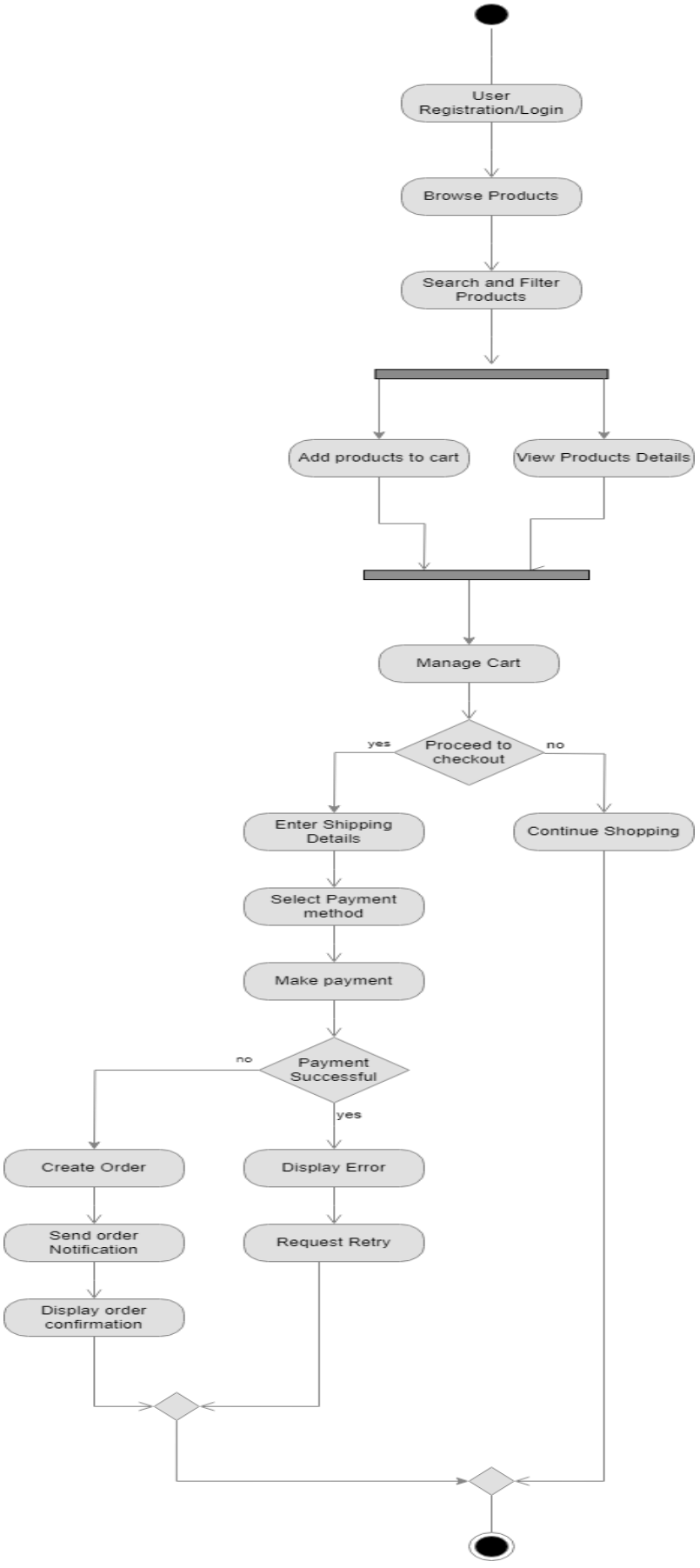


Fig 4.2.4: Activity Diagram

4.2.5 Class Diagram

A class diagram serves as a static blueprint for an application, illustrating its components and their relationships when the system is in a dormant state. It provides insights into the system's structure, showcasing the various elements it comprises and how they interact. In essence, a class diagram acts as a visual guide for software development, aiding in the creation of functional applications.

Key Aspects of a Class Diagram:

- System Overview: A class diagram offers a high-level representation of a software system, presenting its constituent parts, associations, and collaborative dynamics. It serves as an organizational framework for elements such as names, attributes, and methods, simplifying the software development process.
- Structural Visualization: The class diagram is a structural diagram that combines classes, associations, and constraints to define the system's architecture.

Components of a Class Diagram:

The class diagram comprises three primary sections:

- Upper Section: This top segment features the class name, representing a group of objects that share common attributes, behaviors, and roles. Guidelines for displaying groups of objects include capitalizing the initial letter of the class name, positioning it in the center, using bold lettering, and employing slanted writing style for abstract class titles.
- Middle Section: In this part, the class's attributes are detailed, including their visibility indicators, denoted as public (+), private (-), protected (#), or package (~).
- Lower Section: The lower section elaborates on the class's methods or operations, presented in a list format with each method on a separate line. It outlines how the class interacts with data.

In UML, relationships within a class diagram fall into three categories:

- Dependency: Signifying the influence of one element's changes on another.
- Generalization: Representing a hierarchical relationship where one class acts as a parent, and another serves as its child.

- Association: Indicating connections between elements.
- Multiplicity: Defining constraints on the number of instances allowed to possess specific characteristics, with one being the default value when not specified.
- Aggregation: An aggregation is a group that is a part of a relationship called association.
- Composition: "Composition" is like a smaller part of "aggregation.". This describes how a parent and child need each other, so if one is taken away, the other won't work anymore.

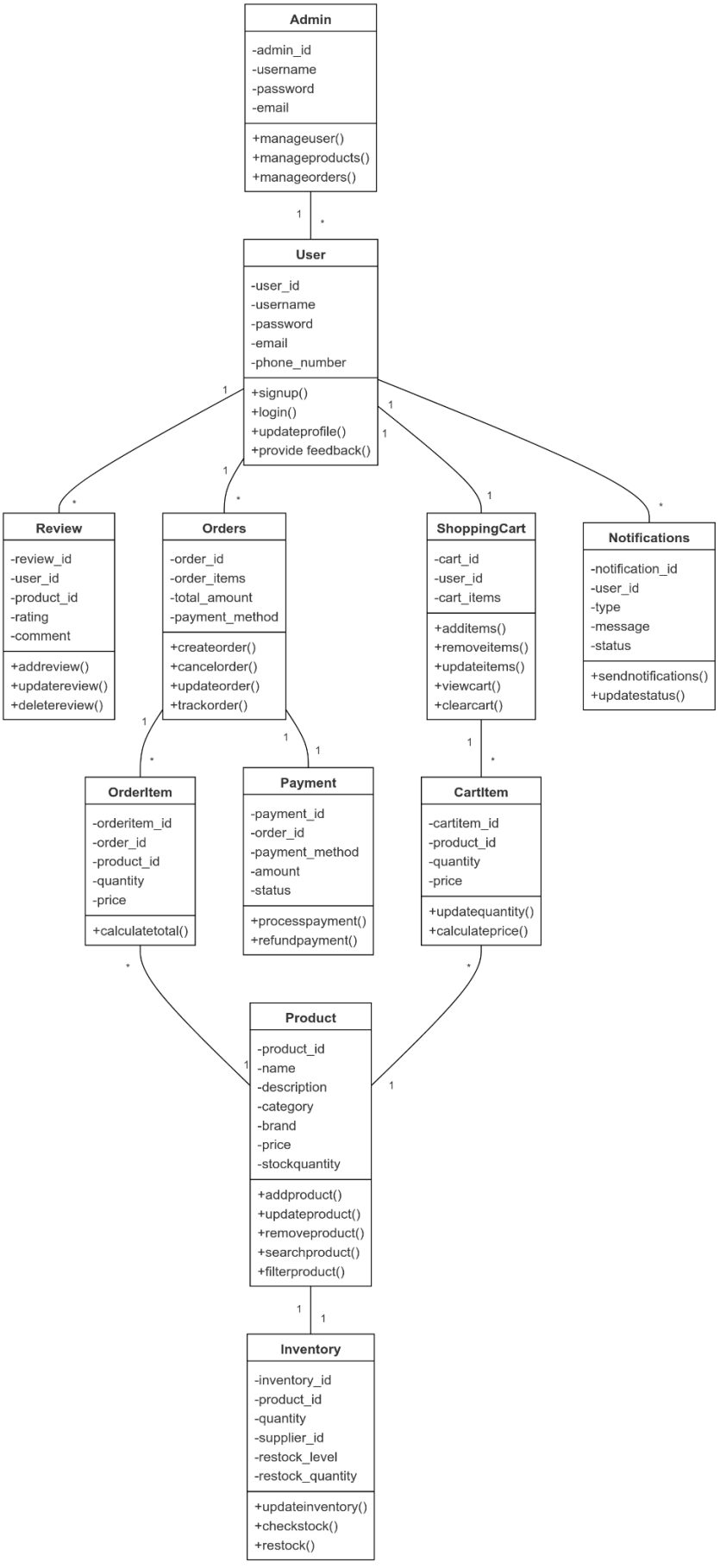


Fig 4.2.5: Class Diagram

4.2.6 Object Diagram

Object diagrams are derived from class diagrams and rely on them to provide a visual representation. They offer an illustration of a collection of objects related to a particular class. Object diagrams provide a snapshot of objects in an object-oriented system at a specific point in time.

Object diagrams and class diagrams share similarities, but they also have distinctions. Class diagrams are more generalized and do not portray specific objects. This abstraction in class diagrams simplifies the comprehension of a system's functionality and structure.

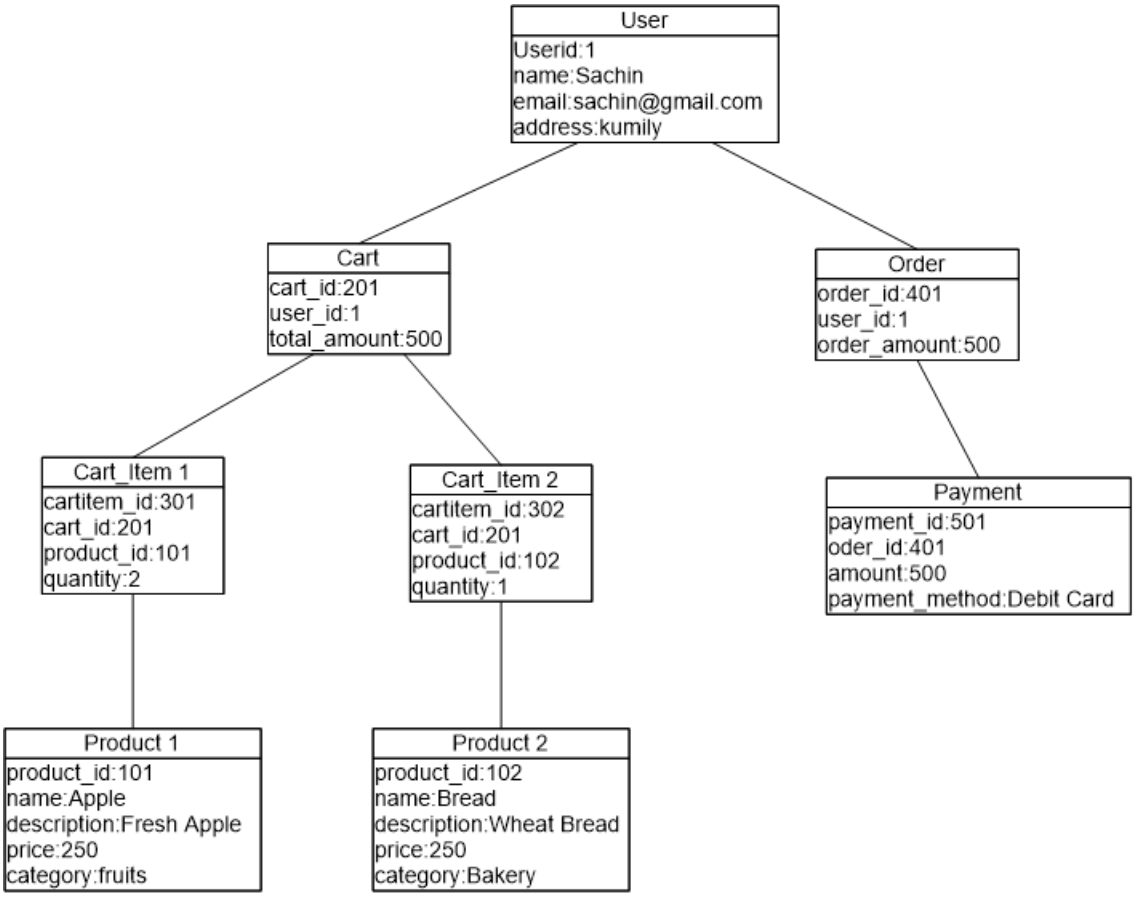


Fig 4.2.6: Object Diagram

4.2.7 Component Diagram

A component diagram serves the purpose of breaking down a complex system that utilizes objects into more manageable segments. It offers a visual representation of the system, showcasing its internal components such as programs, documents, and tools within the nodes.

This diagram elucidates the connections and organization of elements within a system, resulting in the creation of a usable system.

In the context of a component diagram, a component refers to a system part that can be modified and operates independently. It retains the secrecy of its internal operations and requires a specific method to execute a task, resembling a concealed box that functions only when operated correctly.

Notation for a Component Diagram includes:

- A component
- A node

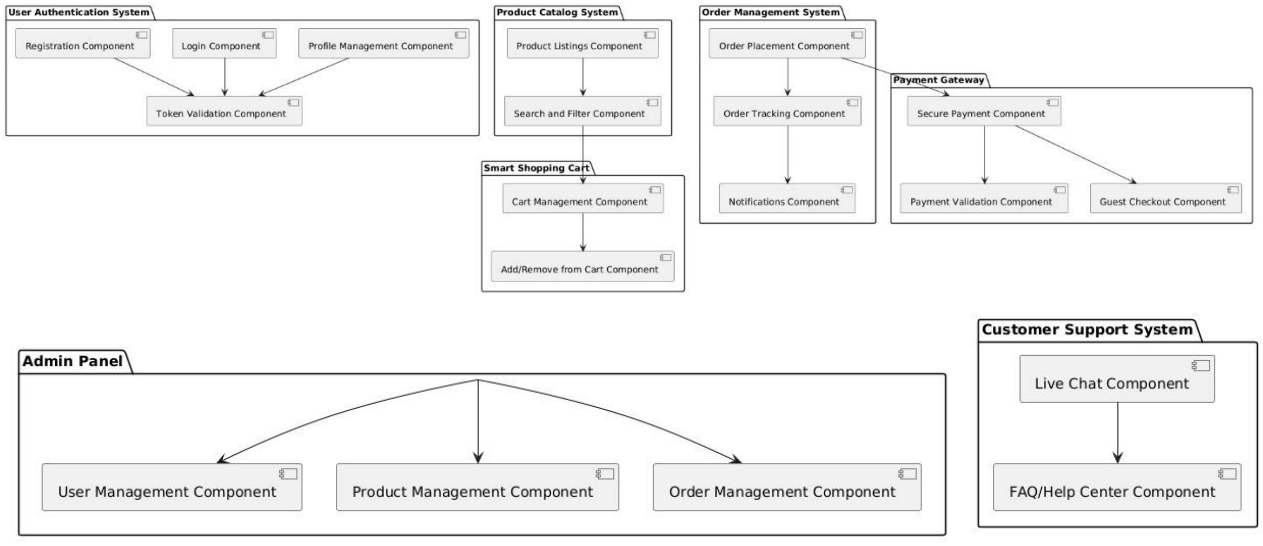


Fig 4.2.7: Component Diagram

4.2.8 Deployment Diagram

A deployment diagram provides a visual representation of how software is positioned on physical computers or servers. It depicts the static view of the system, emphasizing the arrangement of nodes and their connections.

This type of diagram delves into the process of placing programs on computers, elucidating how software is constructed to align with the physical computer system.

Notations in a Deployment Diagram include:

- A component
- An artifact
- An interface
- A node

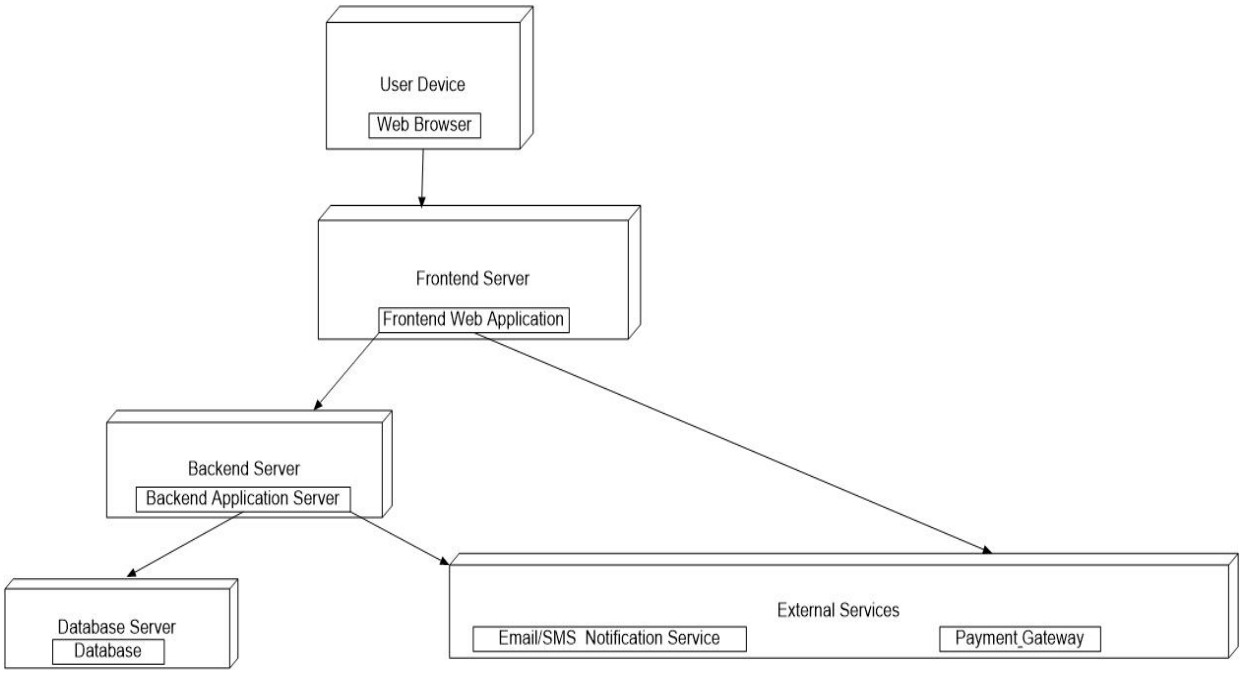


Fig 4.2.8: Deployment Diagram

4.3 USER INTERFACE DESIGN USING FIGMA

4.3.1. Login Page

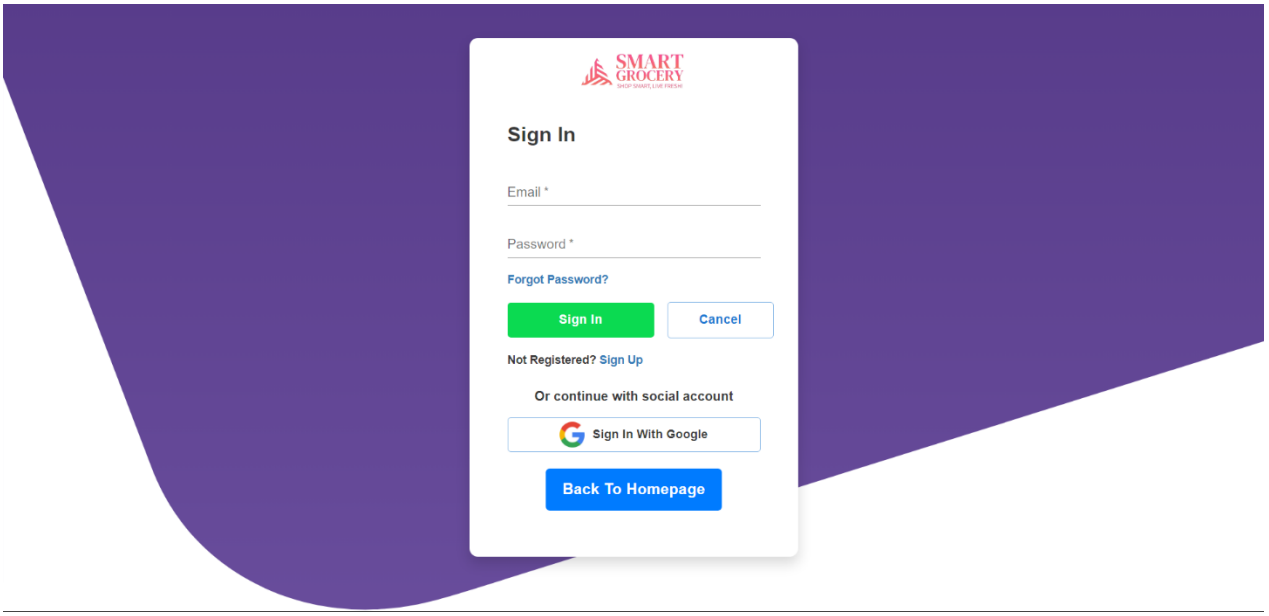


Fig 4.3.1. Login Page

4.3.2. Registration Page

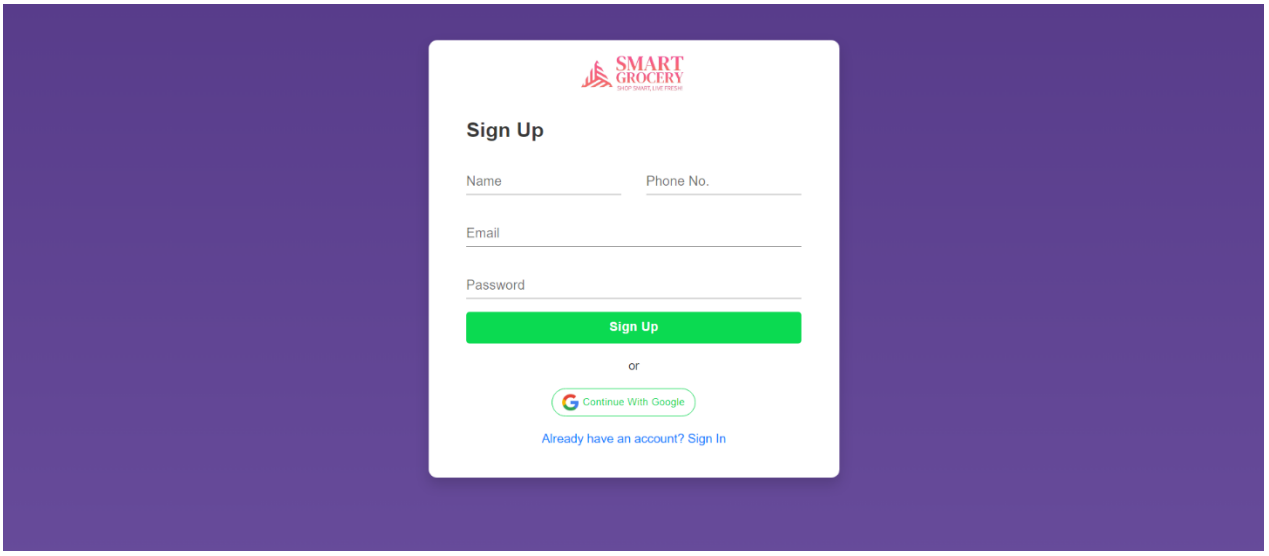


Fig 4.3.2. Registration Page

4.3.3. Homepage

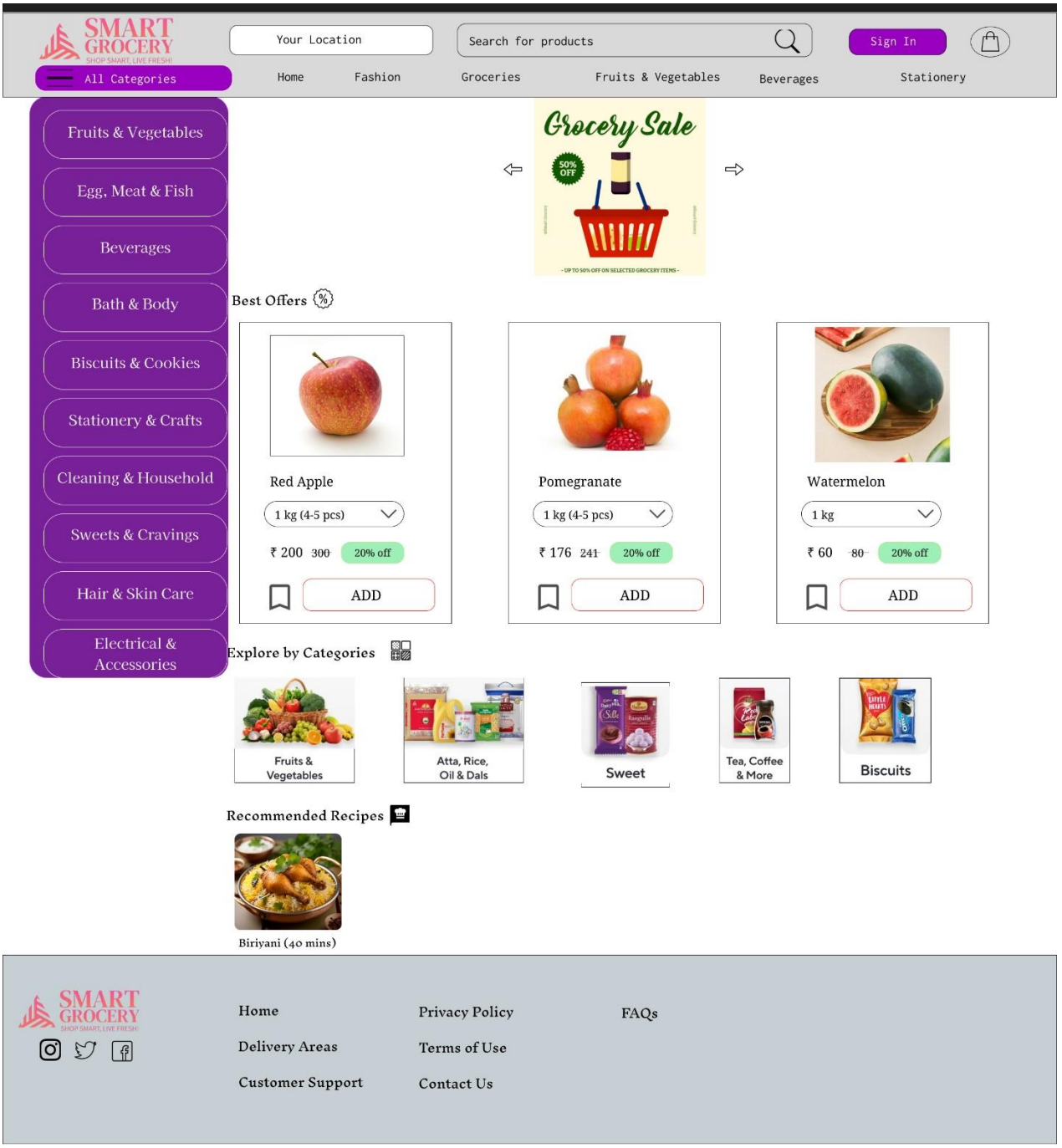


Fig 4.3.3. Homepage

4.3.4. Product View Page

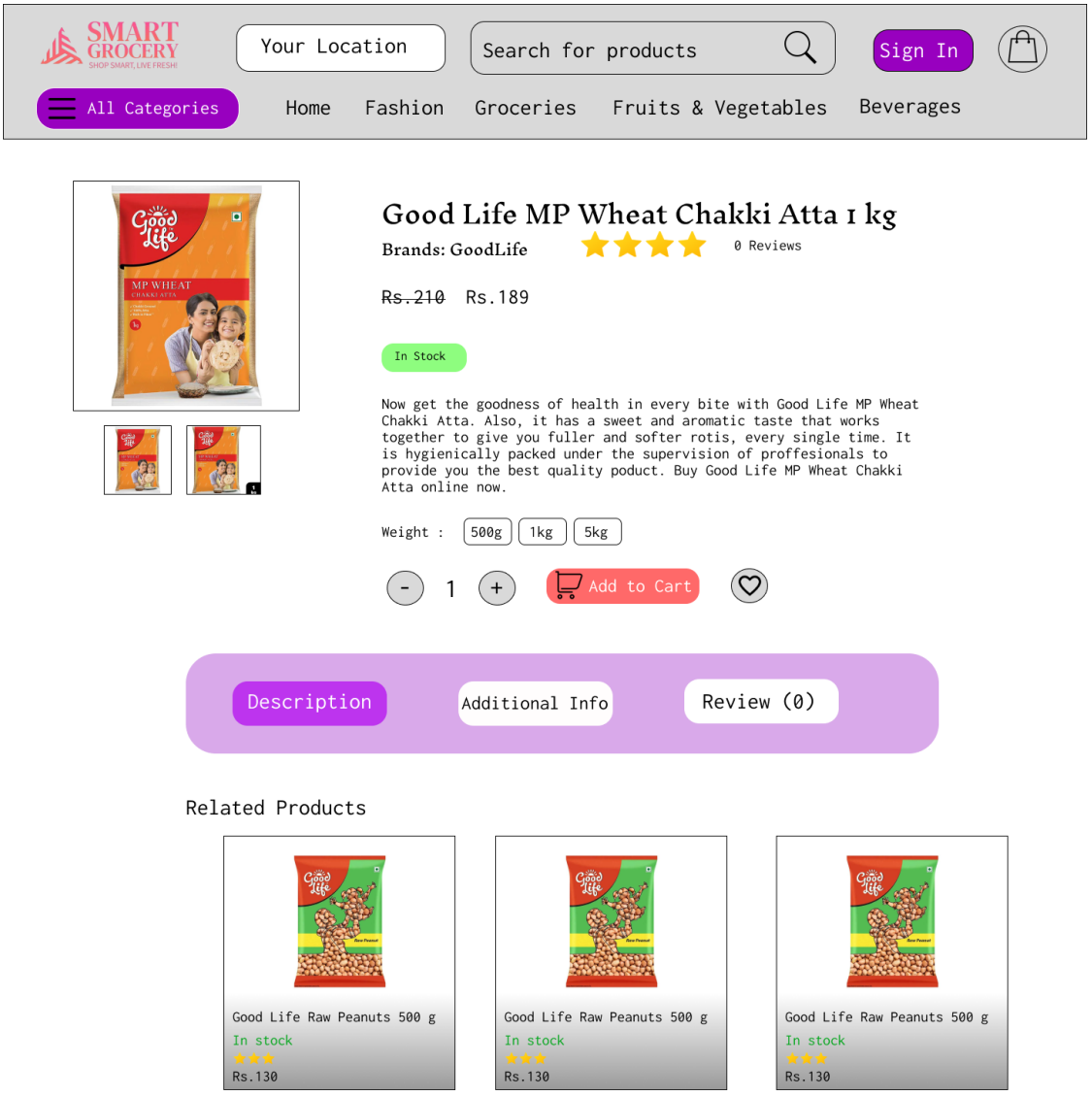


Fig 4.3.4. Product View Page

4.3.5. Cart Page

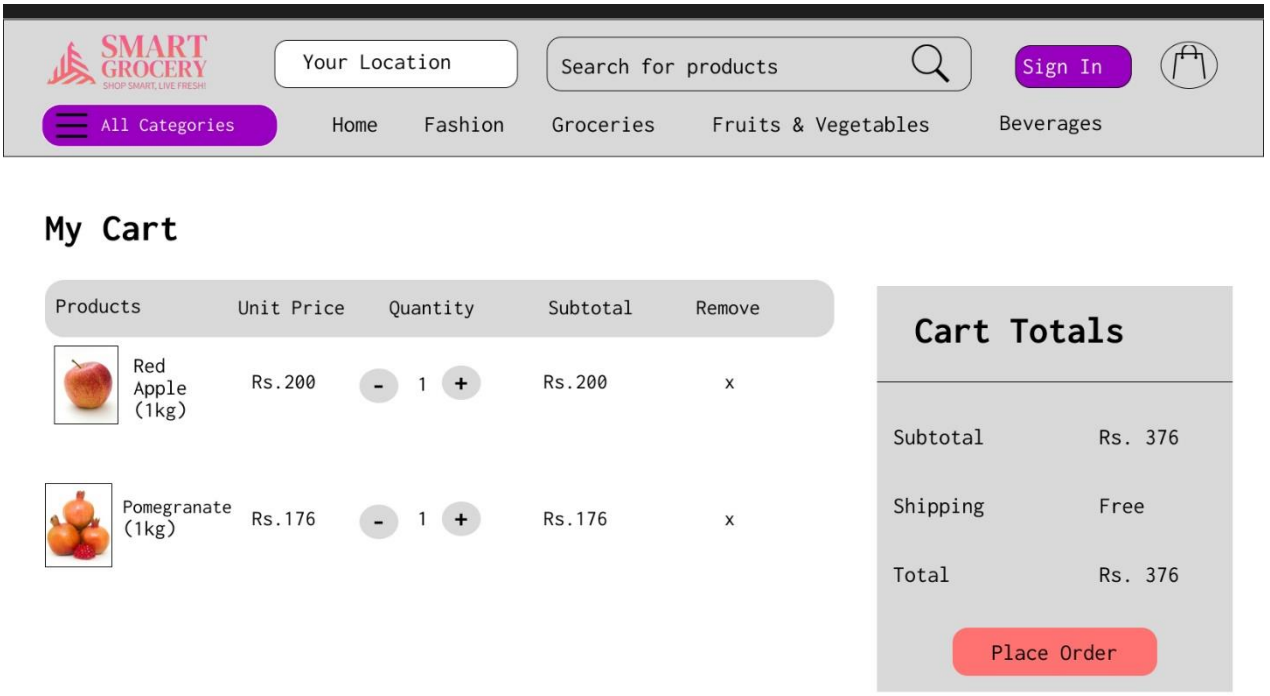


Fig 4.3.5. Cart Page

4.3.6. Buy Now Page

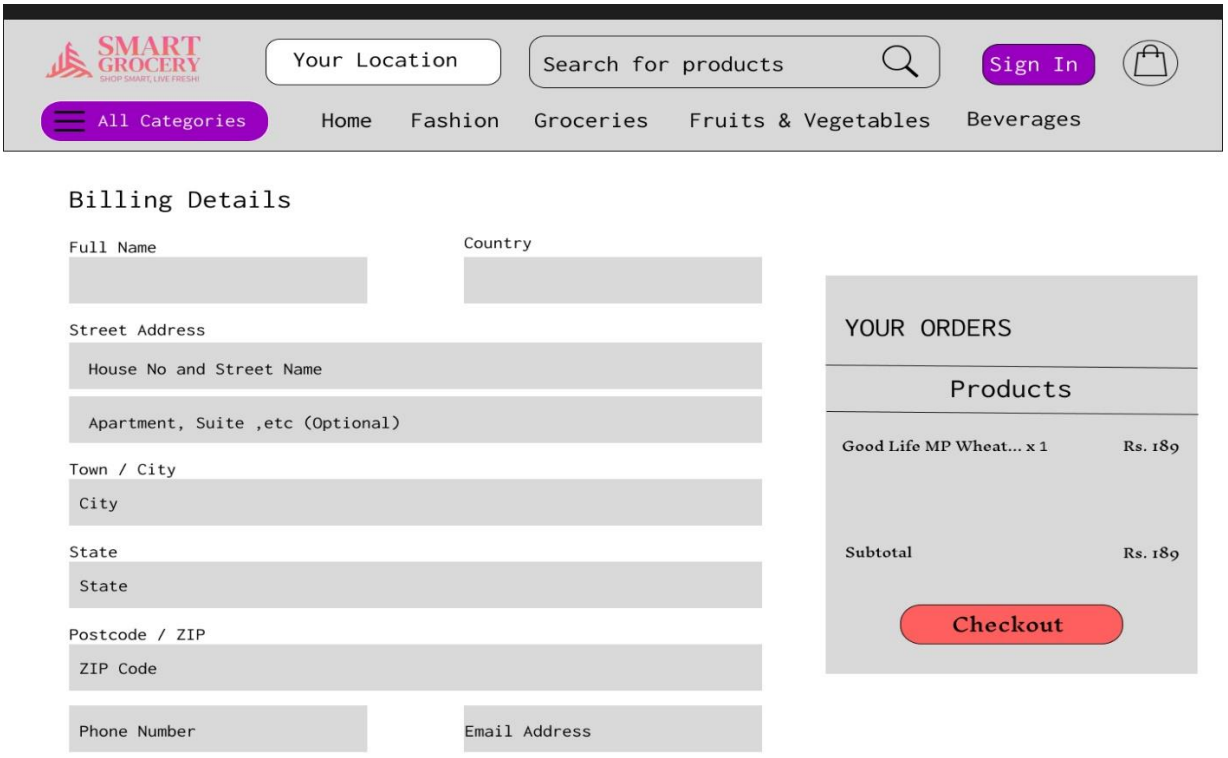


Fig 4.3.6. Buy Now Page

4.4 DATABASE DESIGN

A database serves as an organized repository of data, designed to facilitate efficient management and updates while prioritizing information security. The database design process unfolds in two distinct steps, commencing with information-level design aimed at gathering user requirements and crafting a database that impeccably aligns with user needs. Executed independently of any Database Management System (DBMS), this phase transitions to the specific DBMS design for system development. The subsequent physical-level design considers the characteristics of the chosen

DBMS.

4.4.1 MONGODB DATABASE MANAGEMENT SYSTEM (MDBMS)

MongoDB, a NoSQL database management system, assumes a pivotal role in organizing and managing structured data. Unlike traditional RDBMS, MongoDB stores data in flexible, JSON-like documents within collections. Each document represents a record, and attributes or fields define the document's structure. MongoDB ensures data integrity and supports dynamic schema for scalability. MongoDB's proficiency in handling complex data structures makes it a preferred choice for modern applications, offering flexibility and scalability in managing diverse data types.

4.4.2 Normalization

Normalization, a database design technique ingrained in relational database management systems (RDBMS), also finds relevance in MongoDB. Although MongoDB is a NoSQL database, the principles of normalization – minimizing data redundancy and ensuring data integrity – hold significance. In MongoDB, normalization involves structuring data across multiple collections and establishing relationships using references. This approach eliminates redundancy and maintains data consistency, aligning with MongoDB's schema-less architecture. MongoDB designers apply normalization principles judiciously, balancing data redundancy with query performance to create efficient database structures.

4.4.3 Sanitization

Sanitization is vital for maintaining data security and integrity in the Smart Grocery platform, protecting against threats like SQL injection and cross-site scripting (XSS). Both the React.js frontend and Node.js backend implement sanitization strategies to prevent malicious inputs and ensure data consistency.

On the frontend, controlled components in React.js enable real-time validation, filtering user

input before it reaches the server. Client-side validation, using libraries like Yup or Formik, ensures data like emails and delivery addresses are correctly formatted, improving security and user experience.

On the backend, Node.js uses express-validator to sanitize incoming API requests, ensuring data integrity during processes like user registration and payments. Libraries like DOMPurify prevent XSS attacks by cleaning any HTML content. MongoDB further enforces input validation at the schema level, preventing the storage of harmful data.

This layered approach ensures secure and clean data processing, making the Smart Grocery platform a safe and reliable e-commerce solution.

4.4.4 Indexing

The index keeps track of a certain piece of information or group of information, arranged in order of its value. Arranging the index entries helps find things quickly and easily that match exactly or are within a certain range. Indexes make it easy to find information in a database without having to search through every record every time the database is used. An index is like a roadmap for finding information in a database. It helps you look up data quickly and also makes it easy to find records that are in a certain order. It can be based on one or more columns in the table.

4.5 TABLE DESIGN

4.5.1. Tbl_Users

Primary key: **_id**

No.	Field Name	Data Type	Description
1	_id	ObjectId	Unique identifier for the user
2	name	String	Name of the user
3	phone	String	User's phone number
4	email	String	User's email address
5	password	String	User's password
6	images	Array	Array of image URLs
7	isAdmin	Boolean	Indicates if the user is an admin
8	isBlocked	Boolean	Indicates if the user is blocked
9	reason	String	Reason for blocking the user
10	resetCode	String	Code for password reset
11	resetCodeExpiration	Date	Expiration date for reset code
12	isStockManager	Boolean	Indicates if the user is a stock manager
13	location	String	User's location
14	role	String	Role of the user (default: customer)

4.5.2. Tbl_Category

Primary key: **_id**

No.	Field Name	Data Type	Description
1	_id	ObjectId	Unique identifier for the category
2	name	String	Name of the category
3	slug	String	URL-friendly version of the name
4	images	Array	Array of image URLs
5	color	String	Color associated with the category
6	parentId	String	ID of the parent category (if any)

4.5.3. Tbl_SubCategory

Primary Key: **_id**

Foreign Key: **category** references **tbl_Category (_id)**

No.	Field Name	Data Type	Description
1	_id	ObjectId	Unique identifier for the sub-category
2	category	ObjectId	References tbl_Category (_id)
3	subCat	String	Name of the sub-category

4.5.4. Tbl_Order

Primary key: **_id**

Foreign Key**: userId** references **tbl_Users (_id)**

No.	Field Name	Data Type	Description
1	_id	ObjectId	Unique identifier for the order
2	name	String	Name of the customer
3	phoneNumber	String	Customer's phone number
4	address	String	Delivery address
5	pincode	String	Pincode for the delivery location
6	amount	String	Total amount for the order
7	paymentId	String	Payment identifier
8	email	String	Customer's email address
9	userId	ObjectId	References tbl_Users (_id)
10	products	Array	List of products in the order
11	status	String	Current status of the order (default: pending)
12	date	Date	Date of the order

4.5.5. Tbl_Products

Primary Key: **_id**

Foreign Key: **catId** references **tbl_Category (_id)**

Foreign Key: **subCatId** references **tbl_SubCategory (_id)**

Foreign Key: **category** references **tbl_Category (_id)**

No.	Field Name	Data Type	Description
1	_id	ObjectId	Unique identifier for the product
2	name	String	Name of the product
3	description	String	Description of the product
4	images	Array	Array of image URLs
5	brand	String	Brand of the product
6	price	Number	Current price of the product
7	oldPrice	Number	Previous price of the product
8	catName	String	Name of the category
9	catId	ObjectId	References tbl_Category (_id)
10	subCatId	ObjectId	References tbl_SubCategory (_id)
11	category	ObjectId	References tbl_Category (_id)
12	countInStock	Number	Number of items available
13	rating	Number	Average rating of the product
14	isFeatured	Boolean	Indicates if the product is featured
15	discount	Number	Discount percentage
16	productRam	Array	Array of RAM options
17	size	Array	Array of size options
18	productWeight	Array	Array of weight options
19	location	String	Location of the product
20	dateCreated	Date	Date when the product was created

4.5.6. Tbl_Cart

Primary Key: **_id**

Foreign Key: **userId** references **tbl_Users (_id)**

No.	Field Name	Data Type	Description
1	_id	ObjectId	Unique identifier for the cart item
2	productTitle	String	Title of the product
3	image	String	Image URL of the product
4	rating	Number	Product rating
5	price	Number	Price of the product
6	quantity	Number	Quantity of the product in the cart
7	subTotal	Number	Subtotal for the product
8	productId	String	Unique identifier for the product
9	countInStock	Number	Number of items in stock
10	userId	ObjectId	References tbl_Users (_id)
11	weight	String	Weight of the product

4.5.7. Tbl_MyList

Primary key: **_id**

Foreign Key: **userId** references **tbl_Users (_id)**

No.	Field Name	Data Type	Description
1	_id	ObjectId	Unique identifier for the item in the list
2	productTitle	String	Title of the product
3	image	String	Image URL of the product
4	rating	Number	Product rating
5	price	Number	Price of the product
6	productId	ObjectId	Unique identifier for the product
7	userId	ObjectId	References tbl_Users (_id)

4.5.8. Tbl_Pincode

Primary Key: **_id**

No.	Field Name	Data Type	Description
1	_id	ObjectId	Unique identifier for the district
2	name	String	Name of the district
3	pincodes	Array	List of pincode objects
4	code	String	Code of the district

4.5.9. Tbl_ProductWeight

Primary key: **_id**

No.	Field Name	Data Type	Description
1	_id	ObjectId	Unique identifier for the weight entry
2	productWeight	String	Weight of the product

4.5.10. Tbl_RecentlyViewed

Primary Key: **_id**

Foreign Key: **productId** references **tbl_Products (_id)**

Foreign Key: **userId** references **tbl_Users (_id)**

No.	Field Name	Data Type	Description
1	_id	ObjectId	Unique identifier for the recently viewed item
2	productId	ObjectId	References tbl_Products (_id)
3	userId	ObjectId	References tbl_Users (_id)

4.5.11. Tbl_CancelledOrders

Primary Key: **_id**

Foreign Key: **orderId** references **tbl_Orders (_id)**

Foreign Key: **userId** references **tbl_Users (_id)**

No.	Field Name	Data Type	Description
1	_id	ObjectId	Unique identifier for the cancelled order
2	orderId	ObjectId	References tbl_Orders (_id)
3	userId	ObjectId	References tbl_Users (_id)

4.5.12. Tbl_HomeBanner

Primary Key**: _id**

No.	Field Name	Data Type	Description
1	_id	ObjectId	Unique identifier for the home banner
2	image	String	Image URL of the banner
3	redirectTo	String	URL to redirect to when clicked

4.5.13. Tbl_ImageUpload

Primary Key: **_id**

No.	Field Name	Data Type	Description
1	_id	ObjectId	Unique identifier for the uploaded image
2	imageURL	String	URL of the uploaded image

4.5.14. Tbl_ProductReview

Primary Key: **_id**

Foreign Key: **productId** references **tbl_Products (_id)**

Foreign Key: **customerId** references **tbl_Users (_id)**

No.	Field Name	Data Type	Description
1	_id	ObjectId	Unique identifier for the review
2	productId	ObjectId	References tbl_Products (_id)
3	customerId	ObjectId	References tbl_Users (_id)
4	rating	Number	Rating given by the customer
5	review	String	Review content

CHAPTER 5

SYSTEM TESTING

5.1 INTRODUCTION

Software testing is an essential procedure employed to verify if a computer program functions as intended. It is conducted to ensure that the software performs its designated tasks accurately and complies with the prescribed requirements and standards. Validation is the process of inspecting and assessing software to confirm its adherence to the specified criteria. Software testing is a method for assessing the performance of a program, often in conjunction with techniques like code inspection and program walkthroughs. Validation ensures that the software aligns with the user's expectations and requirements.

Several principles and objectives guide the process of software testing, including:

1. Testing is the practice of executing a program with the primary aim of identifying errors.
2. An effective test case is one that has a high likelihood of uncovering previously undiscovered errors.
3. A successful test is one that exposes previously undiscovered errors.

When a test case operates effectively and accomplishes its objectives, it can detect flaws within the software. This demonstrates that the computer program is functioning as intended and is performing well. The process of evaluating a computer program encompasses three primary aspects:

1. Correctness assessment
2. Evaluation of implementation efficiency
3. Examination of computational complexity

5.2 TEST PLAN

A test plan serves as a comprehensive set of instructions for conducting various types of tests. It can be likened to a map that outlines the steps to follow when evaluating a computer program. Software developers create instructions for both using and organizing the necessary information for the program's proper functionality. They ensure that each component of the program performs its intended functions. To ensure the thorough testing of the software, a group known as the ITG (Information Technology Group) is responsible for verifying its functionality, instead of relying solely on the software's creators for testing.

The objectives of testing should be clearly defined and measurable. A well-structured test plan should encompass details about the frequency of failures, the associated repair costs, the occurrence rate of issues, and the time required for complete testing. The levels of testing typically include:

- Unit testing
- Integration testing
- Data validation testing
- Output testing

5.2.1 Unit Testing

Unit testing checks the smallest part of a software design - the software component or module. Testing important control paths within a module using the design guide to find errors. This means how difficult the tests are for each small part of a program and what parts of the program haven't been tested yet. Unit testing is a type of testing that looks at how the code works inside and can be done at the same time for different parts of the program.

Before starting any other test, we need to check if the data flows correctly between different parts of the computer program. If the information doesn't move in and out correctly, all other checks are pointless. When designing something, it's important to think about what could go wrong and make a plan for how to deal with those problems. This can mean redirecting the process or stopping it completely.

The Smart Grocery System was tested by looking at each part by itself and trying different tests on it. Some mistakes in the design of the modules were discovered and then fixed. After writing the instructions for different parts, each part is checked and tried out separately. We got rid of extra code and made sure everything works the way it should.

5.2.2 Integration Testing

Integration testing is a critical process in software development that involves constructing a program while simultaneously identifying errors in the interaction between different program components. The primary objective is to utilize tested individual parts and assemble them into a program according to the initial plan. This comprehensive testing approach assesses the entire program to ensure its proper and correct functionality.

As issues are identified and resolved during integration testing, it's not uncommon for new problems to surface, leading to an ongoing cycle of testing and refinement. After each individual component of the system is thoroughly examined, these components are integrated

to ensure they function harmoniously. Additionally, efforts are made to standardize all programs to ensure uniformity rather than having disparate versions.

5.2.3 Validation Testing or System Testing

The final phase of testing involves a comprehensive examination of the entire system to ensure the correct interaction of various components, including different types of instructions and building blocks. This testing approach is referred to as Black Box testing or System testing.

Black Box testing is a method employed to determine if the software functions as intended. It assists software engineers in identifying all program issues by employing diverse input types. Black Box testing encompasses the assessment of errors in functions, interfaces, data access, performance, as well as initialization and termination processes. It is a vital technique to verify that the software meets its intended requirements and performs its functions correctly.

5.2.4 Output Testing or User Acceptance Testing

System testing is conducted to assess user satisfaction and alignment with the company's requirements. During the development or update of a computer program, it's essential to maintain a connection with the end-users. This connection is established through the following elements:

- Input Screen Designs.
- Output Screen Designs.

To perform system testing, various types of data are utilized. The preparation of test data plays a crucial role in this phase. Once the test data is gathered, it is used to evaluate the system under investigation. When issues are identified during this testing, they are addressed by following established procedures. Records of these corrections are maintained for future reference and improvement. This process ensures that the system functions effectively and meets the needs of both users and the organization.

5.2.5 Automation Testing

Automated testing is a method employed to verify that software functions correctly and complies with established standards before it is put into official use. This type of testing relies on written instructions that are executed by testing tools. Specifically, UI automation testing involves the

use of specialized tools to automate the testing process. Instead of relying on manual interactions where individuals click through the application to ensure its proper functioning, scripts are created to automate these tests for various scenarios. Automating testing is particularly valuable when it is necessary to conduct the same test across multiple computers simultaneously, streamlining the testing process and ensuring consistency.

5.2.6 Selenium Testing

Selenium is a valuable and free tool designed for automating website testing. It plays a crucial role for web developers as it simplifies the testing process. Selenium automation testing refers to the practice of using Selenium for this purpose. Selenium isn't just a single tool; it's a collection of tools, each serving distinct functions in the realm of automation testing. Manual testing is a necessary aspect of application development, but it can be monotonous and repetitive. To alleviate these challenges, Jason Huggins, an employee at ThoughtWorks, devised a method for automating testing procedures, replacing manual tasks. He initially created a tool named the JavaScriptTestRunner to facilitate automated website testing, and in 2004, it was rebranded as Selenium.

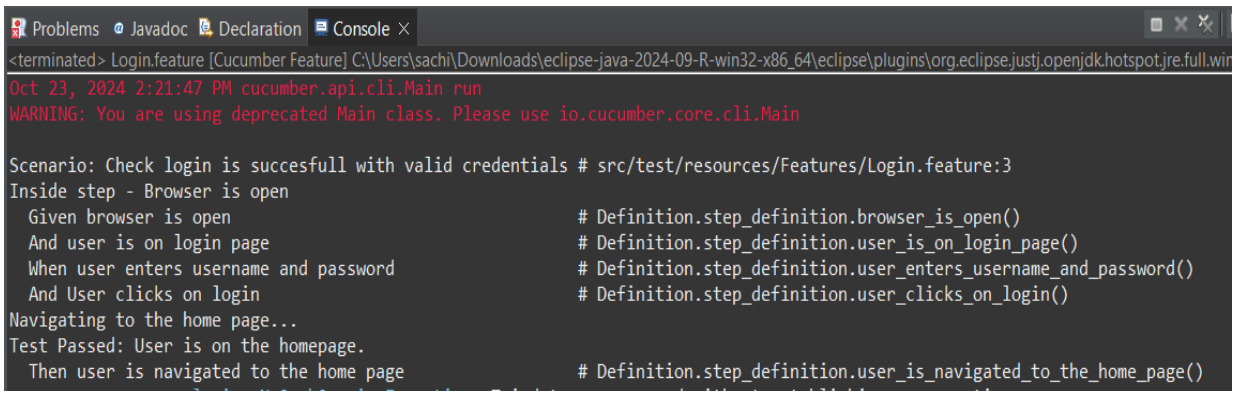
Test Case 1: User Login

Code:

```
package Definition;
import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.firefox.FirefoxDriver;
import io.cucumber.java.en.And;
import io.cucumber.java.en.Given;
import io.cucumber.java.en.Then;
import io.cucumber.java.en.When;
public class step_definition {
	WebDriver driver=null;
	@Given("browser is open")
	public void browser_is_open() {
		System.out.println("Inside step-Browser is open");
		System.setProperty("webdriver.gecko.marionette","C:\\Users\\sachi\\eclipse-
workspace\\1234\\src\\test\\resources\\drivers\\geckodriver.exe");
		driver=new FirefoxDriver();
		driver.manage().window().maximize();

	}

	@And("user is on login page")
	public void user_is_on_login_page() throws Exception {
		driver.navigate().to("http://localhost:3008/signIn");
		Thread.sleep(2000);

	}

	@When("user enters username and password")
public void user_enters_username_and_password() throws Throwable{
		driver.findElement(By.id("emailid")).sendKeys("sachinsamjacob@gmail.com");
		driver.findElement(By.id("passwords")).sendKeys("Password@123");
```

```
        }

        @When("User clicks on login")
        public void user_clicks_on_login() {
                driver.findElement(By.id("login")).click();     }

        @Then("user is navigated to the home page")
        public void user_is_navigated_to_the_home_page() throws Exception {
                driver.findElement(By.id("submitbutton")).isDisplayed();
           Boolean isLogoutDisplayed = driver.findElement(By.id("logout")).isDisplayed();

           if (isLogoutDisplayed) {
              System.out.println("Login successful and user is on the home page");
           } else {
              System.out.println("Login failed or not navigated to the home page");
           }

                Thread.sleep(2000);
                driver.close();
                driver.quit();
        }
}
```

Screenshot



Test Report

Test Case 1

Project Name: Smart Grocery

Login Test Case

Test Case ID: 1	**Test Designed By: Sachin Sam Jacob**
Test Priority (Low/Medium/High): High	**Test Designed Date: 22-10-2024**
Module Name: **Login Module**	**Test Executed By: Ms. Sona Maria Sebastian**
Test Title: User Login	**Test Execution Date: 22-10-2024**
Description: User has a valid email/password	

Pre-Condition: User has valid username and password

Step	Test Step	Test Data	Expected Result	Actual Result	Status(Pass/ Fail)
1	Navigate to login page		Login form should be displayed	Login form is displayed	Pass
2	Provide valid Email	Email : sachinsamjacob@gmail.com	User should be able to login	User logs in	Pass
3	Provide valid password	Password: Password@1234			
4	Click on login button				

Post-Condition: User is validated with database and successfully logs into Homepage.

Test Case 2: Add to Cart

Code:

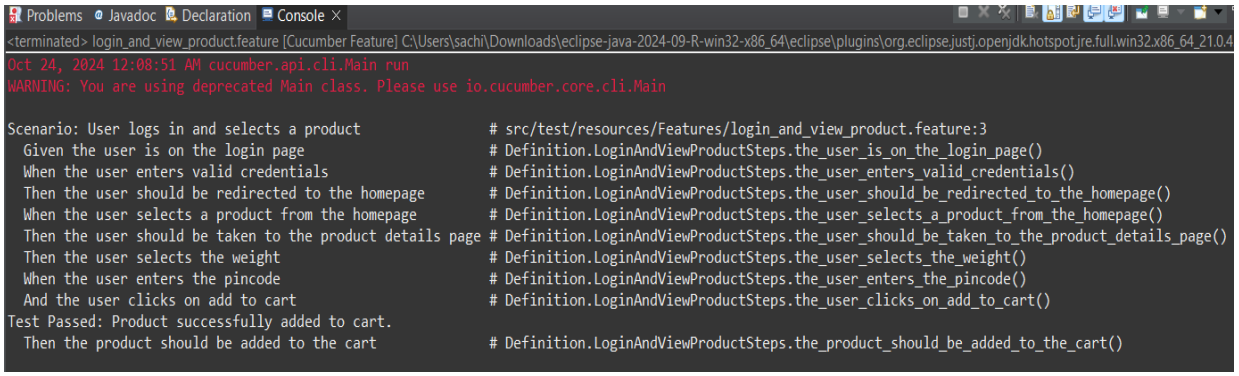
```
package Definition;

import io.cucumber.java.After;
import io.cucumber.java.en.*;
import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.chrome.ChromeDriver;
import org.openqa.selenium.firefox.FirefoxDriver;
import org.openqa.selenium.support.ui.ExpectedConditions;
import org.openqa.selenium.support.ui.WebDriverWait;

import java.time.Duration;

public class LoginAndViewProductSteps {
   WebDriver driver;
   WebDriverWait wait;

   @Given("the user is on the login page")
   public void the_user_is_on_the_login_page() {
      System.setProperty("webdriver.gecko.marionette",
"C:\\Users\\sachi\\eclipse-workspace\\1234\\src\\test\\resources\\drivers");      //
Update the path to your geckodriver
      driver = new FirefoxDriver();
      wait = new WebDriverWait(driver, Duration.ofSeconds(10));

      // Navigate to the login page
      driver.get("http://localhost:3008/signIn");
   }

   @When("the user enters valid credentials")
```

```
    public void the_user_enters_valid_credentials() {
        // Locate the email and password fields and enter the credentials
        WebElement emailField =
wait.until(ExpectedConditions.visibilityOfElementLocated(By.id("emailid")));
        WebElement passwordField = driver.findElement(By.id("passwords"));

        emailField.sendKeys("sachinsamjacob@gmail.com");
        passwordField.sendKeys("Password@123");

        // Click the login button
        WebElement loginButton =
driver.findElement(By.xpath("//button[contains(text(),'Sign In')]"));
        loginButton.click();
    }

    @Then("the user should be redirected to the homepage")
    public void the_user_should_be_redirected_to_the_homepage() {
        // Verify that the user is on the homepage
        wait.until(ExpectedConditions.urlContains("http://localhost:3008/"));
        try {
            Thread.sleep(5000); // 5000 milliseconds = 5 seconds
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }

    @When("the user selects a product from the homepage")
    public void the_user_selects_a_product_from_the_homepage() {
        // Locate the first product element and click on it
        WebElement product =
wait.until(ExpectedConditions.visibilityOfElementLocated(By.cssSelector(".pr
oductItem")));
        try {
            Thread.sleep(3000); // 5000 milliseconds = 5 seconds
```

```
            } catch (InterruptedException e) {
                e.printStackTrace();
            }// Adjust the selector as needed
            product.click();
        }

        @Then("the user should be taken to the product details page")
        public void the_user_should_be_taken_to_the_product_details_page() {
            // Verify that the user is on the product details page

    wait.until(ExpectedConditions.urlContains("http://localhost:3008/product/"));
    // Adjust the URL as needed

        }
        @Then("the user selects the weight")
        public void the_user_selects_the_weight() {
            WebElement WeightButton =
    driver.findElement(By.id("weightchoose"));
            WeightButton.click();
        }

        @When("the user enters the pincode")
        public void the_user_enters_the_pincode() {
            WebElement pincodeField =
    wait.until(ExpectedConditions.visibilityOfElementLocated(By.id("enterpincod
    e")));
            pincodeField.sendKeys("686518"); // Replace with actual pincode

            // Optionally, click a button to check availability based on pincode
            WebElement checkPincodeButton =
    driver.findElement(By.id("checkpincode"));
            checkPincodeButton.click();
        }
```

```
    @And("the user clicks on add to cart")
    public void the_user_clicks_on_add_to_cart() {
        WebElement addToCartButton =
wait.until(ExpectedConditions.visibilityOfElementLocated(By.id("addtocart"))
);
        addToCartButton.click();
    }

    @Then("the product should be added to the cart")
    public void the_product_should_be_added_to_the_cart() {

          wait.until(ExpectedConditions.visibilityOfElementLocated(By.className("header")));
          WebElement cartButton = driver.findElement(By.id("cart"));

        cartButton.click();
        try {
            Thread.sleep(8000); // 5000 milliseconds = 5 seconds
        } catch (InterruptedException e) {
            e.printStackTrace();
        }// Adjust class name as needed
        System.out.println("Test Passed: Product successfully added to cart.");

        driver.quit();
    }
}
```

Screenshot:



Test report

Test Case 2

Project Name: Smart Grocery					
Add to Cart Test Case					
Test Case ID: Test_2			**Test Designed By: Sachin Sam Jacob**		
Test Priority (Low/Medium/High): High			**Test Designed Date: 22-10-2024**		
Module Name: Add to Cart Module			**Test Executed By: Ms. Sona Maria Sebastian**		
Test Title: Add to Cart			**Test Execution Date: 22-10-2024**		
Description: User can add product to Cart.					
Pre-Condition: User has to be logged in.					
Step	**Test Step**	**Test Data**	**Expected Result**	**Actual Result**	**Status(Pass/ Fail)**
1	Navigate to login page		Login form should be displayed	Login form is displayed	Pass
2	Provide valid Email	Email: sachinsamjacob@gmail.com	User should be able to login	User logs in	Pass
3	Provide valid password	Password: Password@1234			
4	Click on login button				
5	Clicks on a product		The product should be displayed	The product is displayed	Pass
6	Selects the Weight				

7.	Enters the pincode and checks the product is available for delivery to the location	Pincode: 686518	Checks the product is Deliverable or Not	The Product is Deliverable	Pass
8	Clicks on Add to Cart		Product is to be added to Cart	Product is Added to Cart	Pass
9	Cart is Viewed				
Post-Condition: User is validated with database and successfully logs into Homepage and the product is selected and Pincode is entered to check availability to the entered location and product is added to cart.					

Test Case 3: Order of Product

Code:

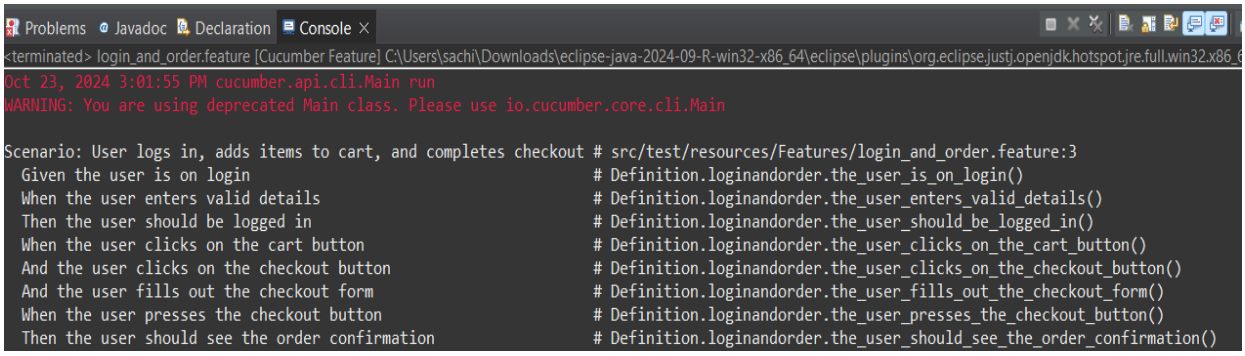
```
package Definition;

import io.cucumber.java.en.*;
import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.firefox.FirefoxDriver;
import org.openqa.selenium.support.ui.ExpectedConditions;
import org.openqa.selenium.support.ui.WebDriverWait;

import java.time.Duration;

public class loginandorder {
   WebDriver driver;
   WebDriverWait wait;
   @Given("the user is on login")
   public void the_user_is_on_login() {
```

```
            System.setProperty("webdriver.gecko.marionette",
"C:\\Users\\sachi\\eclipse-workspace\\1234\\src\\test\\resources\\drivers");      //
Update the path to your geckodriver
        driver = new FirefoxDriver();
        driver.manage().timeouts().implicitlyWait(Duration.ofSeconds(10));
        driver.get("http://localhost:3008/signIn"); // Replace with your login URL
    }

    @When("the user enters valid details")
    public void the_user_enters_valid_details() {
        // Locate and fill in the login form

driver.findElement(By.id("emailid")).sendKeys("sachinsamjacob@gmail.com"
); // Replace with your email field name
        driver.findElement(By.id("passwords")).sendKeys("Password@123");    //
Replace with your password field name
        driver.findElement(By.cssSelector("button[type='submit']")).click();       //
Adjust selector as needed
    }

    @Then("the user should be logged in")
    public void the_user_should_be_logged_in() {
        // Verify login success (adjust the condition as per your application)
        wait = new WebDriverWait(driver, Duration.ofSeconds(10));

wait.until(ExpectedConditions.visibilityOfElementLocated(By.className("hea
der"))); // Adjust the locator for a visible element after login
    }

    @When("the user clicks on the cart button")
    public void the_user_clicks_on_the_cart_button() {
        // Click on the cart button
        WebElement cartButton = driver.findElement(By.id("cart"));
```

```
        try {
            Thread.sleep(3000); // 5000 milliseconds = 5 seconds
        } catch (InterruptedException e) {
            e.printStackTrace();
        }// Adjust class name as needed
        cartButton.click();
    }

    @When("the user clicks on the checkout button")
    public void the_user_clicks_on_the_checkout_button() {
        // Click on the checkout button
            WebElement checkoutButton = driver.findElement(By.id("checkout"));
// Adjust class name as needed
             try {
                  Thread.sleep(3000); // 5000 milliseconds = 5 seconds
               } catch (InterruptedException e) {
                  e.printStackTrace();
               }
        checkoutButton.click();
    }

    @When("the user fills out the checkout form")
    public void the_user_fills_out_the_checkout_form() {
        // Fill out the checkout form
        driver.findElement(By.name("fullName")).sendKeys("Sachin          Sam
Jacob");
        driver.findElement(By.name("country")).sendKeys("India");
        driver.findElement(By.name("streetAddressLine1")).sendKeys("Konil
House ");

driver.findElement(By.name("streetAddressLine2")).sendKeys("Thekkady");
        driver.findElement(By.name("city")).sendKeys("Kumily");
        driver.findElement(By.name("state")).sendKeys("Kerala");
        driver.findElement(By.name("zipCode")).sendKeys("685509");
```

```
driver.findElement(By.name("phoneNumber")).sendKeys("8078263332");

driver.findElement(By.name("email")).sendKeys("sachinsamjacob@gmail.co
m");
    }

    @When("the user presses the checkout button")
    public void the_user_presses_the_checkout_button() {
        try {
        Thread.sleep(3000); // 5000 milliseconds = 5 seconds
      } catch (InterruptedException e) {
        e.printStackTrace();
      }

      // Press the checkout button
      driver.findElement(By.xpath("//button[contains(text(),
'Checkout')]")).click(); // Adjust the locator as needed
    }

    @Then("the user should see the order confirmation")
    public void the_user_should_see_the_order_confirmation() {
      // Verify order confirmation (adjust the condition as per your application)
          try {
              Thread.sleep(5000); // 5000 milliseconds = 5 seconds
            } catch (InterruptedException e) {
              e.printStackTrace();
            } // Adjust the locator for confirmation
      driver.quit(); // Close the browser
    }
}
```

Screenshot



Test Report

Test Case 3

Project Name: Smart Grocery					
Checkout Test Case					
Test Case ID: Test_3			**Test Designed By: Sachin Sam Jacob**		
Test Priority (Low/Medium/High): High			**Test Designed Date: 22-10-2024**		
Module Name: Checkout Module			**Test Executed By: Ms. Sona Maria Sebastian**		
Test Title: Order of Product			**Test Execution Date: 22-10-24**		
Description: User checkout the products in Cart					
Pre-Condition: User has to be logged in and should contain products in cart.					
Step	**Test Step**	**Test Data**	**Expected Result**	**Actual Result**	**Status(Pass/ Fail)**
1	Navigate to login page		Login form should be displayed	Login form is displayed	Pass
2	Provide valid Email	Email : sachinsamjacob @ gmail.com	User should be able to login	User logs in	Pass
3	Provide valid password	Password: Password@1234			
4	Click on login button				
5	Click on cart		User should checkouts the products in	User checkouts the products in the cart	Pass

			cart		
6	Click on Checkout				
7	Enters the checkout details	Full Name:Sachin Sam Jacob, Country: India, StreetAddressLine 1:Konil House, StreetAddressLine 2:Thekkady, City:Kumily, State:Kerala, Zipcode:685509, Phone Number:8078263 332, Email:sachinsamj acob@gmail.com			
8	Enters the payment gateway		Payement Gateway should be opened	Payement Gateway is Opened	Pass
Post-Condition: User is validated with database and successfully logs into Homepage and clicks on cart option and checkouts the products in the cart and the payment gateway is opened.					

Test Case 4: Product View

Code:

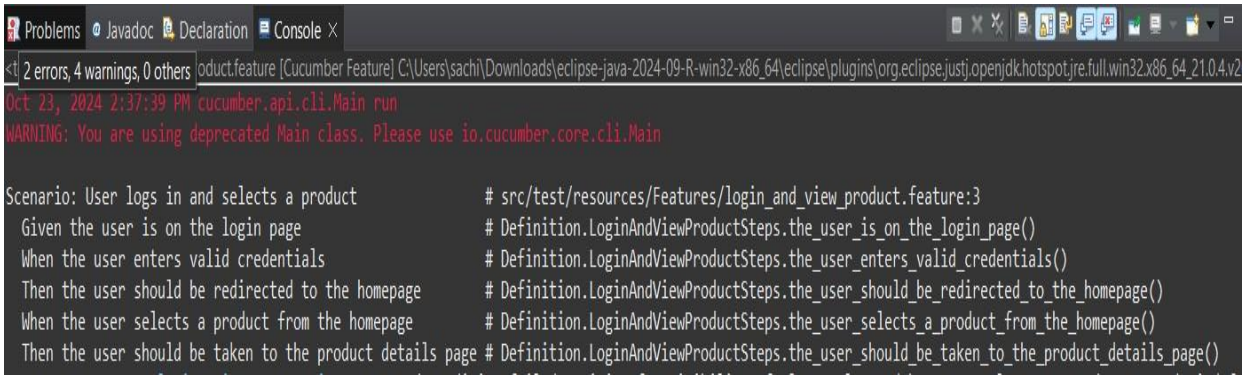
```
package Definition;

import io.cucumber.java.After; import
io.cucumber.java.en.*; import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver; import
org.openqa.selenium.WebElement; import
org.openqa.selenium.chrome.ChromeDriver; import
org.openqa.selenium.firefox.FirefoxDriver; import
```

```
org.openqa.selenium.support.ui.ExpectedConditions;
import org.openqa.selenium.support.ui.WebDriverWait;

import java.time.Duration;

public class LoginAndViewProductSteps {
   WebDriver driver;
   WebDriverWait wait;

   @Given("the user is on the login page")     public
void the_user_is_on_the_login_page() {
System.setProperty("webdriver.gecko.marionette",
"C:\\Users\\sachi\\eclipse-workspace\\1234\\src\\test\\resources\\drivers");    //
Update the path to your geckodriver         driver = new FirefoxDriver();
      wait = new WebDriverWait(driver, Duration.ofSeconds(10));

      // Navigate to the login page
driver.get("http://localhost:3008/signIn");
   }

   @When("the user enters valid credentials")
public void the_user_enters_valid_credentials() {
      // Locate the email and password fields and enter the credentials
      WebElement emailField =
wait.until(ExpectedConditions.visibilityOfElementLocated(By.id("emailid")));
      WebElement passwordField = driver.findElement(By.id("passwords"));

      emailField.sendKeys("sachinsamjacob@gmail.com");
passwordField.sendKeys("Password@123");

      // Click the login button
      WebElement loginButton =
driver.findElement(By.xpath("//button[contains(text(),'Sign In')]"));
loginButton.click();
   }
```

```
   @Then("the user should be redirected to the homepage")
public void the_user_should_be_redirected_to_the_homepage() {
      // Verify that the user is on the homepage
      wait.until(ExpectedConditions.urlContains("http://localhost:3008/"));
try {
         Thread.sleep(5000); // 5000 milliseconds = 5
seconds       } catch (InterruptedException e) {
e.printStackTrace();
      }
   }

   @When("the user selects a product from the homepage")
   public void the_user_selects_a_product_from_the_homepage() {
      // Locate the first product element and click on it
      WebElement                              product                              =
wait.until(ExpectedConditions.visibilityOfElementLocated(By.cssSelector(".pr
oductItem")));
      try {
         Thread.sleep(3000); // 5000 milliseconds = 5
seconds       } catch (InterruptedException e) {
e.printStackTrace();       }// Adjust the selector as needed
product.click();
   }

   @Then("the user should be taken to the product details page")    public
void the_user_should_be_taken_to_the_product_details_page() {
      // Verify that the user is on the product details page

wait.until(ExpectedConditions.urlContains("http://localhost:3008/product/"));
// Adjust the URL as needed

      // Optionally, you can verify the product name or other details
      WebElement                              productName                          =
```

```
wait.until(ExpectedConditions.visibilityOfElementLocated(By.cssSelector(".
pr oduct-name"))); // Adjust the selector as needed        assert
productName.isDisplayed();
    }
    @After     public void
tearDown() {        //
Close the browser        if
(driver != null) {
driver.quit();
        }
    }
}
```

Screenshot:



Test report

Test Case 4

Project Name: Smart Grocery	
Product View Test Case	
Test Case ID: Test_4	**Test Designed By: Sachin Sam Jacob**
Test Priority (Low/Medium/High): High	**Test Designed Date: 22-10-2024**
Module Name: Product View Module	**Test Executed By:** **Ms. Sona Maria Sebastian**
Test Title: Product View	**Test Execution Date: 22-10-2024**
Description: User can view the product	

Pre-Condition : User has to be logged in.					
Step	**Test Step**	**Test Data**	**Expected Result**	**Actual Result**	**Status(Pass/ Fail)**
1	Navigate to login page		Login form should be displayed	Login form is displayed	Pass
2	Provide valid Email	Email : sachinsamjacob@gmail.com	User should be able to login	User logs in	Pass
3	Provide valid password	Password: Password@1234			
4	Click on login button				
5	Clicks on a product		The product should be displayed	The product is displayed	Pass
6	Redirects to product view of that selected product				
Post-Condition: User is validated with database and successfully logs into Homepage and the product is selected and is viewed.					

CHAPTER 6

IMPLEMENTATION

6.1 INTRODUCTION

Implementation is the stage where a planned system transforms into a tangible and operational entity. Building trust and instilling confidence in users is of paramount importance for the success of the system. This is a critical phase that places a strong emphasis on user training and the creation of informative materials. The actual transition often occurs during or after user training. Implementation signifies the act of putting a new system into action after its design phase, turning a conceptual design into a functional one.

Implementation involves the process of transitioning from the old method of operation to the new one, whether it replaces the old system entirely or makes incremental changes. Ensuring that the process is executed accurately is vital to establish a system that aligns well with the organization's requirements. System implementation encompasses the following tasks:

- Meticulous planning.
- Assessment of the existing system and its constraints.
- Designing methods to facilitate the transition.

6.2 IMPLEMENTATION PROCEDURES

Software implementation involves the installation of software in its intended location and ensuring that it functions as intended. In some organizations, an individual who is not directly involved in using the software may oversee and approve the project's development. Initially, there may be some skepticism regarding the software, and it's essential to address these concerns to prevent strong resistance.

To ensure a successful transition, several key steps are important:

- Communicating Benefits: Users need to understand why the new software is an improvement over the old one, instilling trust in its capabilities.
- User Training: Providing training to users is crucial to make them feel confident and comfortable using the application.

To evaluate the outcome, it is essential to verify that the server program is running on the server. If the server is not operational, the expected outcomes will not be achieved.

6.2.1 User Training

User training is a critical component of ensuring that individuals know how to use and adapt to a new system. It plays a vital role in fostering user comfort and confidence with the system. Even when a system becomes more complex, the need for training becomes even more apparent. User training covers various aspects, including inputting information, error handling, querying databases, and utilizing tools for generating reports and performing essential tasks. It aims to empower individuals with the knowledge and skills needed to effectively interact with the system.

6.2.2 Training on the Application Software

Once the fundamental computer skills have been covered, the next step is to instruct individuals on using a new software program. This training should provide a comprehensive understanding of the new system, including navigation through screens, accessing help resources, error management, and resolution procedures. The training aims to equip users or groups with the knowledge and skills necessary to effectively utilize the system or its components. It's important to note that training may be tailored differently for various groups of users and individuals in different roles within the organization to cater to their specific needs and requirements.

6.2.3 System Maintenance

Maintaining a system's functionality is a complex challenge in software development. In the maintenance phase of the software life cycle, the software performs critical functions and operates smoothly. Once a system is operational, it requires ongoing care and maintenance to ensure its continued optimal performance. Software maintenance is a crucial aspect of the development process as it ensures that the system can adapt to changes in its environment. Maintenance involves more than just identifying and correcting errors in the code. It encompasses various tasks that contribute to the system's stability and effectiveness.

6.2.4 Hosting

Smart Grocery Shopping platform is hosted on **Render** and **Netlify**, chosen for their reliability, performance, and ease of integration with the development stack. Render is utilized for backend server deployment, while Netlify hosts the frontend interfaces for customers, admin, and district operation managers. These platforms together enable a smooth, scalable user experience for the website’s visitors.

Render

Render is a cloud-based hosting platform that offers a streamlined approach for deploying full-stack applications, particularly well-suited for server-side applications like our backend API. Render provides us with a robust hosting environment for our **Node.js backend server**, which supports secure, high-performance API calls to the frontends hosted on Netlify. Key reasons for choosing Render include:

- **Automatic Deployments**: Render automatically builds and deploys from GitHub repositories upon new commits, ensuring that the backend server is always up-to-date.
- **SSL Support**: Render offers free SSL certificates, securing data transactions between our server and client applications.
- **Scalability**: Render allows for scalable deployments, ensuring the backend can handle high traffic and API requests effectively as our user base grows.

Hosting Process on Render (Backend)

Step 1: Create a Render Account

Sign up and log into Render (render.com).

Step 2: Connect GitHub Repository

Link the GitHub repository containing the backend code.

Step 3: New Web Service Setup:

- Select "New Web Service" and choose the connected repository.
- Configure the environment
- Build Command: npm install
- Start Command: node index.js
- Add the necessary Environment Variables
- Enable auto-deploy for continuous integration upon new commits.

Step 4: Specify Environment Variables

Set environment variables for database connections, API keys, and other configurations.

Step 5: Deploy the Service

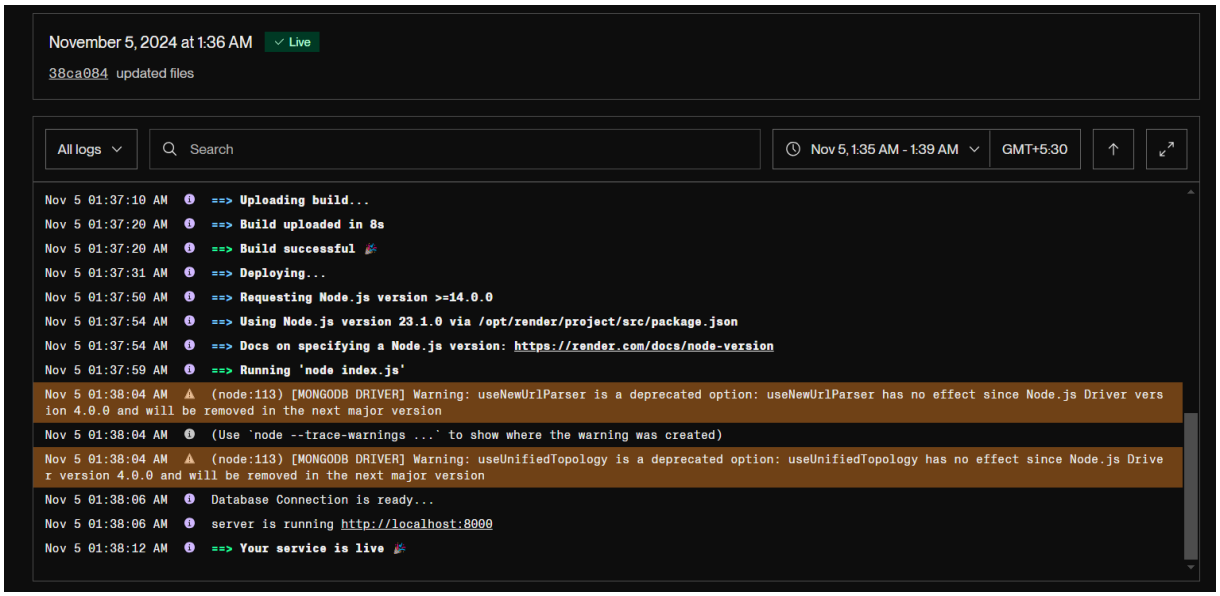
Render handles the build and deployment process, making the backend server live on a secure URL.

Step 6: Test and Monitor

Use Render's built-in logs and analytics to monitor the backends performance and error logs for optimal functionality.

Hosted Link: https://smart-grocery-server.onrender.com

Screenshot



Netlify

Netlify is a powerful hosting platform for static websites and client applications, ideal for hosting our **customer, admin, and district operation manager interfaces**. Netlify was chosen for its excellent support for **React applications** and ease of deployment, particularly beneficial for our frontend applications. Some of its benefits include:

- **Continuous Deployment**: Netlify's CI/CD pipeline integrates with GitHub, automatically deploying any updates to the frontend.
- **Global CDN**: Netlify uses a global content delivery network (CDN) for faster loading times and high availability worldwide.
- **Built-in SSL**: Free SSL certificates are provided, ensuring secure data exchanges for all users.
- **Custom Domain and DNS Management**: Netlify supports custom domain configuration and efficient DNS management for reliable access.

Hosting Process on Netlify (Frontend)

Step 1: Create a Netlify Account

Sign up and log into Netlify (netlify.com).

Step 2: Connect GitHub Repository

Link each GitHub repository for the customer, admin, and district operation manager frontends.

Step 3: New Site Setup

- Select "New Site from Git" and choose the relevant repository.
- Configure the build settings, including commands like npm run build and publish directory settings.

Step 4: Set Environment Variables

Define any environment variables necessary for client functionality (e.g., API URLs for connecting to the Render server).

Step 5: Deploy Site

Netlify automatically builds and deploys each frontend, providing unique URLs.

Step 6: Domain Configuration

Configure custom domains and SSL certificates for secure and branded access.

Hosted Link (Customer): https://smartgrocerysite.netlify.app/

Hosted Link (Admin): https://smartgroceryadmin.netlify.app/

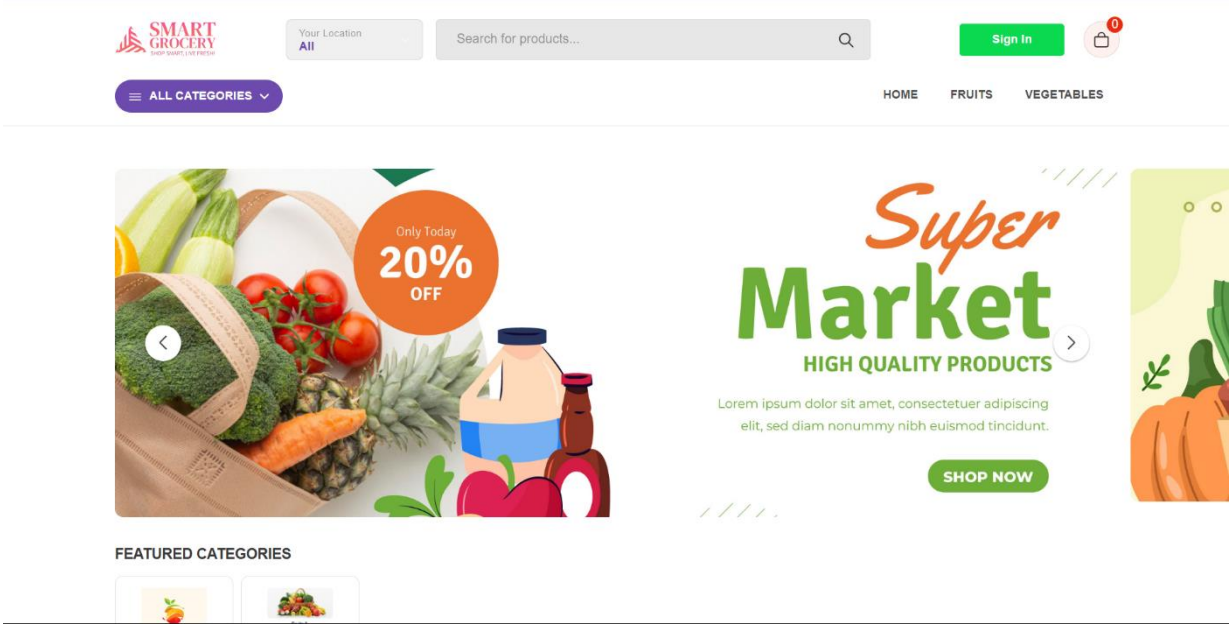
Hosted Link (District Operations Manager): https://smartgrocerydom.netlify.app/

Hosted QR Code (Customer):

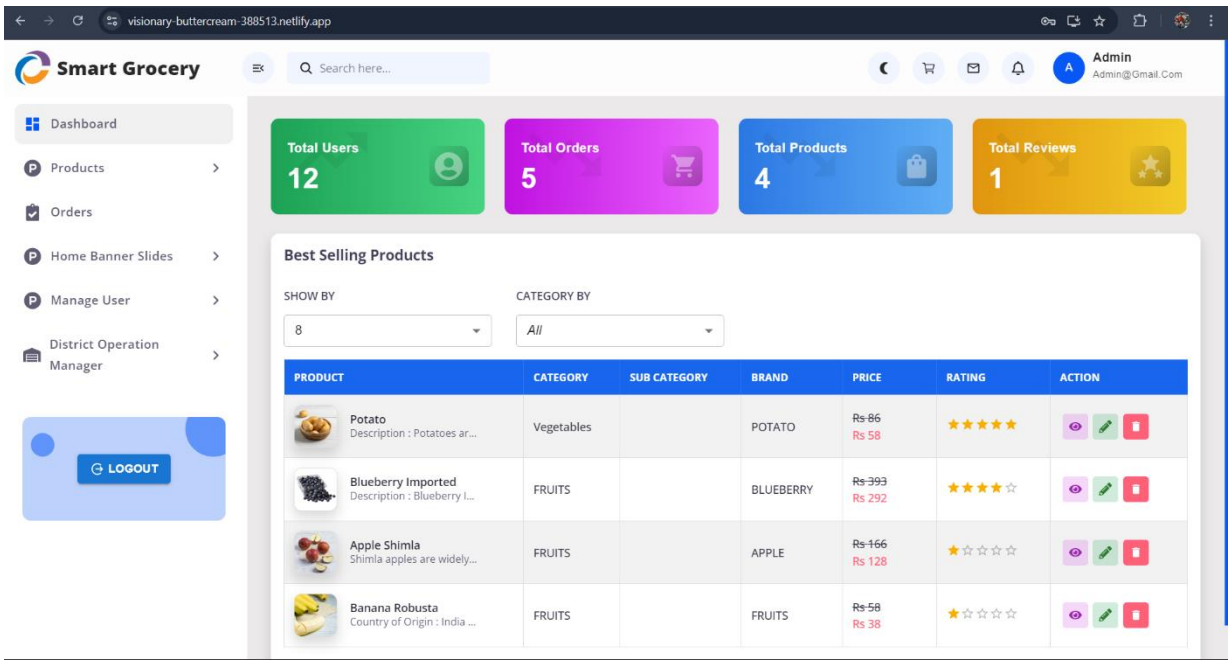
Hosted QR Code (Admin):

Hosted QR Code (District Operation Manager):

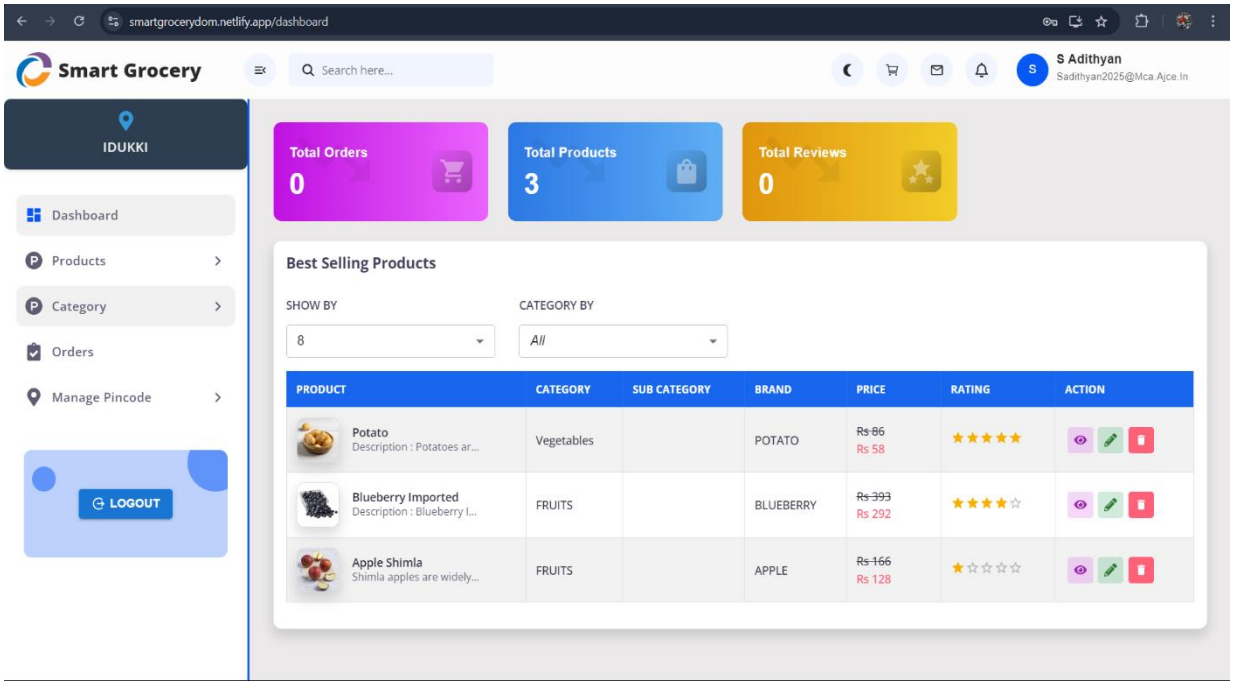
Screenshot (Customer):



Screenshot (Admin):



Screenshot (District Operations Manager):



CHAPTER 7

CONCLUSION AND FUTURE SCOPE

7.1 CONCLUSION

The analysis of the **Smart Grocery** platform reveals that it is a well-designed and innovative solution for modern grocery shopping. It addresses the growing demand for a user-centric and technology-driven shopping experience, providing a wide range of features to ensure efficiency, convenience, and personalization for users. The platform's key features, including user authentication, product catalog and search, smart shopping cart, and secure payment, contribute to its effectiveness in simplifying the grocery shopping process.

The **Smart Grocery** project focuses on creating a user-friendly interface with features like an intuitive product catalog and search functionality. Users can manage their profiles, track their orders, and receive notifications, ensuring a smooth and convenient shopping journey. From the admin perspective, the system offers a robust dashboard for managing users, products, and orders. Admins have the ability to oversee and maintain the entire platform, adding or updating products, managing inventory, and addressing customer queries, all while leveraging real-time data and analytics. The successful implementation of this system can significantly enhance the grocery shopping experience, benefiting consumers and administrators alike by streamlining processes and offering modern solutions to common challenges in the grocery retail industry.

7.2 FUTURE SCOPE

The future scope of the **Smart Grocery** project includes several advanced features to enhance both user experience and operational efficiency. Key modules for future scope will introduce Personalized Recommendations, offering tailored product suggestions based on user preferences and behaviour, Delivery Management and Supplier Management. Smart Shopping Assistant will provide AI-driven, voice-activated assistance, guiding users with real-time suggestions and product comparisons. The Custom Meal Planning module will recommend recipes and automatically add ingredients to shopping carts, simplifying meal preparation. The Dynamic Pricing and Deals module will adjust prices based on stock levels, location, and time-sensitive offers, ensuring users receive competitive pricing. An Enhanced Admin Panel will give administrators tools to monitor user activity, manage inventory, update stock, schedule deliveries, and maintain secure payment gateways.

CHAPTER 8
BIBLIOGRAPHY

REFERENCES:

- Shelly, G. B., & Rosenblatt, H. J. (2009). System Analysis and Design. Course Technology.
  This book provided foundational knowledge on system design and analysis, which was essential in structuring the Smart Grocery platform's architecture.

- Hunt, A., & Thomas, D. (2008). The Pragmatic Programmer: From Journeyman to Master. Pearson India, 1st Edition.Concepts from this book were applied to ensure best practices in coding and software development during the Smart Grocery website's implementation.

- Schwaber, K., & Beedle, M. (2008). Agile Software Development with Scrum. Pearson.This reference guided the development process, particularly in applying agile methodologies for iterative improvements to the platform's features.

- Crispin, L., & Gregory, J. (2008). Agile Testing: A Practical Guide for Testers and Agile Teams. Addison-Wesley Professional, 1st Edition.Testing strategies from this guide were crucial in validating the various modules such as user authentication, payment gateways, and personalized recommendations.

WEBSITES:

- https://www.mongodb.com/docs/atlas/
- https://reactjs.org/docs/getting-started.html
- https://nodejs.org/en/docs/
- https://www.jiomart.com/
- https://www.bigbasket.com/
- https://chat.openai.com/chat

CHAPTER 9

APPENDIX

9.1 Sample Code

LOGIN PAGE

```
import { useContext, useEffect, useState } from "react";
import Logo from "../../assets/images/MainLogo.png";
import { MyContext } from "../../App";
import TextField from "@mui/material/TextField";
import Button from "@mui/material/Button";
import { Link, useNavigate } from "react-router-dom";
import GoogleImg from "../../assets/images/googleImg.png";
import CircularProgress from "@mui/material/CircularProgress";
import { postData } from "../../utils/api";
import { getAuth, signInWithPopup, GoogleAuthProvider } from "firebase/auth";
import { firebaseApp } from "../../firebase"; // Ensure firebaseApp is correctly initialized
const auth = getAuth(firebaseApp);
const googleProvider = new GoogleAuthProvider();

const SignIn = () => {
  const [isLoading, setIsLoading] = useState(false);
  const context = useContext(MyContext);
  const navigate = useNavigate(); // useNavigate instead of history

  useEffect(() => {
    context.setisHeaderFooterShow(false);
  }, []);

  const [formfields, setFormfields] = useState({
    email: "",
    password: "",
  });

  const onchangeInput = (e) => {
    setFormfields(() => ({
      ...formfields,
      [e.target.name]: e.target.value,
    }));
  };

  const login = (e) => {
    e.preventDefault();

    if (formfields.email === "") {
      context.setAlertBox({
        open: true,
        error: true,
        msg: "Email cannot be blank!",
      });
      return false;
    }
```

```
if (formfields.password === "") {
  context.setAlertBox({
    open: true,
    error: true,
    msg: "Password cannot be blank!",
  });
  return false;
}

setIsLoading(true);
postData("/api/user/signin", formfields).then((res) => {
  try {
    if (res.error !== true) {
      localStorage.setItem("token", res.token);

      const user = {
        name: res.user?.name,
        email: res.user?.email,
        userId: res.user?.id,
        isStockManager: res.user?.isStockManager,
        location:res.user?.location,
      };

      localStorage.setItem("user", JSON.stringify(user));

      context.setAlertBox({
        open: true,
        error: false,
        msg: res.msg,
      });

      context.setIsLogin(true);
      setIsLoading(false);

      setTimeout(() => {
        if (res.user.isAdmin) {
          window.location.href = "http://localhost:3002/"; // Redirect to admin page
        } else if (res.user.isStockManager) {
          navigate("/stockmanager-dashboard", { replace: true }); // Use replace: true here
        } else {
          navigate("/"); // Redirect to user home page
        }
      }, 2000);
    } else {
      // Handle error messages
      context.setAlertBox({
        open: true,
        error: true,
        msg: res.msg,
      });
```

```
        setIsLoading(false);
       }
      }
      catch (error) {
       console.log(error);
       setIsLoading(false);
      }
     });
    };

    const signInWithGoogle = () => {
     signInWithPopup(auth, googleProvider)
      .then((result) => {
       const credential = GoogleAuthProvider.credentialFromResult(result);
       const token = credential.accessToken;
       const user = result.user;

       const fields = {
        name: user.providerData[0].displayName,
        email: user.providerData[0].email,
        password: null,
        images: user.providerData[0].photoURL,
        phone: user.providerData[0].phoneNumber,
       };

       postData("/api/user/authWithGoogle", fields).then((res) => {
        try {
         if (res.error !== true) {
          localStorage.setItem("token", res.token);

          const user = {
           name: res.user?.name,
           email: res.user?.email,
           userId: res.user?.id,
          };

          localStorage.setItem("user", JSON.stringify(user));

          context.setAlertBox({
           open: true,
           error: false,
           msg: res.msg,
          });

          setTimeout(() => {
           navigate("/"); // Use navigate instead of history
           context.setIsLogin(true);
           setIsLoading(false);
           context.setisHeaderFooterShow(true);
          }, 2000);
         } else {
```

```
            context.setAlertBox({
              open: true,
              error: true,
              msg: res.msg,
            });
            setIsLoading(false);
          }
        } catch (error) {
          console.log(error);
          setIsLoading(false);
        }
      });

      context.setAlertBox({
        open: true,
        error: false,
        msg: "User authentication successful!",
      });
    })
    .catch((error) => {
      context.setAlertBox({
        open: true,
        error: true,
        msg: error.message,
      });
    });
};

// const handleForgot=(res) => {
//   navigate(`/forgotpassword`)
// };

const handleBackToHomepage = () => {
  window.location.href = "/"; // Navigate to homepage and reload
};

return (
  <section className="section signInPage">
    <div className="shape-bottom">
      <svg
        fill="#fff"
        id="Layer_1"
        x="0px"
        y="0px"
        viewBox="0 0 1921 819.8"
        style={{ enableBackground: "new 0 0 1921 819.8" }}
      >
        <path
          className="st0"
          d="M1921,413.1v406.7H0V0.5h0.4l228.1,598.3c30,74.4,80.8,130.6,152.5,168.6c107.6
,57,212.1,40.7,245.7,34.4 c22.4-4.2,54.9-13.1,97.5-26.6L1921,400.5V413.1z"
```

```
    ></path>
  </svg>
</div>

<div className="container">
  <div className="box card p-3 shadow border-0">
    <div className="text-center">
      <img style={{width:'130px',height:'50px'}} src={Logo} alt="Logo" />
    </div>

    <form className="mt-3" onSubmit={login}>
      <h2 className="mb-4">Sign In</h2>

      <div className="form-group">
        <TextField
          id="emailid"
          label="Email"
          type="email"
          required
          variant="standard"
          className="w-100"
          name="email"
          onChange={onchangeInput}
        />
      </div>
      <div className="form-group">
        <TextField
          id="passwords"
          label="Password"
          type="password"
          required
          variant="standard"
          className="w-100"
          name="password"
          onChange={onchangeInput}
        />
      </div>
      <Link to="/forgotpassword">
      <a className="border-effect cursor txt" >
       Forgot Password?
      </a>
      </Link>
      <div className="d-flex align-items-center mt-3 mb-3">
        <Button type="submit" id="login" className="btn-blue col btn-lg btn-big">
          {isLoading === true ? <CircularProgress /> : "Sign In"}
        </Button>
        <Link to="/">
          <Button
            className="btn-lg btn-big col ml-3"
            variant="outlined"
            onClick={() => context.setisHeaderFooterShow(true)}
```

```
    >
      Cancel
    </Button>
  </Link>
</div>



<p className="txt">
  Not Registered?{" "}
  <Link to="/signUp" className="border-effect">
    Sign Up
  </Link>
</p>

<h6 className="mt-4 text-center font-weight-bold">
  Or continue with social account
</h6>

<Button
  className="loginWithGoogle mt-2"
  variant="outlined"
  onClick={signInWithGoogle}
>
  <img src={GoogleImg} alt="Google" /> Sign In with Google
</Button>

   {/* Add Back to Homepage Button */}
   <div style={{ textAlign: 'center', margin: '20px 0' }}>
  <Button
    onClick={handleBackToHomepage}
    style={{
      textDecoration: 'none',
      color: 'white',
      backgroundColor: '#007bff',
      fontWeight: 'bold',
      fontSize: '18px',
      padding: '10px 20px',
      border: 'none',
      borderRadius: '5px',
      transition: 'background-color 0.3s',
    }}
    onMouseEnter={(e) => {
      e.target.style.backgroundColor = '#0056b3'; // Darker blue on hover
    }}
    onMouseLeave={(e) => {
      e.target.style.backgroundColor = '#007bff'; // Original blue
    }}
  >
    Back to Homepage
  </Button>
```

```
          </div>
        </form>
      </div>
    </div>
  </section>
 );
};

export default SignIn;
```

BACKEND CODE

```
router.post(`/signin`, async (req, res) => {
  const { email, password } = req.body;

  try {
    const existingUser = await User.findOne({ email: email });
    if (!existingUser) {
      return res.status(404).json({ error: true, msg: "User not found!" });
    }

    // Check if user is blocked
    if (existingUser.isBlocked) {
      return res.status(403).json({ error: true, msg: "User is blocked by the admin due to
unauthorized activities." });
    }

    // Compare plain text passwords directly
    if (password !== existingUser.password) {
      return res.status(400).json({ error: true, msg: "Incorrect password" });
    }

    const token = jwt.sign({ email: existingUser.email, id: existingUser._id,location:
existingUser.location }, process.env.JSON_WEB_TOKEN_SECRET_KEY);
    console.log("ex",existingUser);
    return res.status(200).send({
      user: existingUser,
      token: token,
      msg: "User Login Successfully!"
    });

  } catch (error) {
    res.status(500).json({ error: true, msg: "Something went wrong" });
  }
});
```

9.2 Screen Shots

9.2.1. Login Page

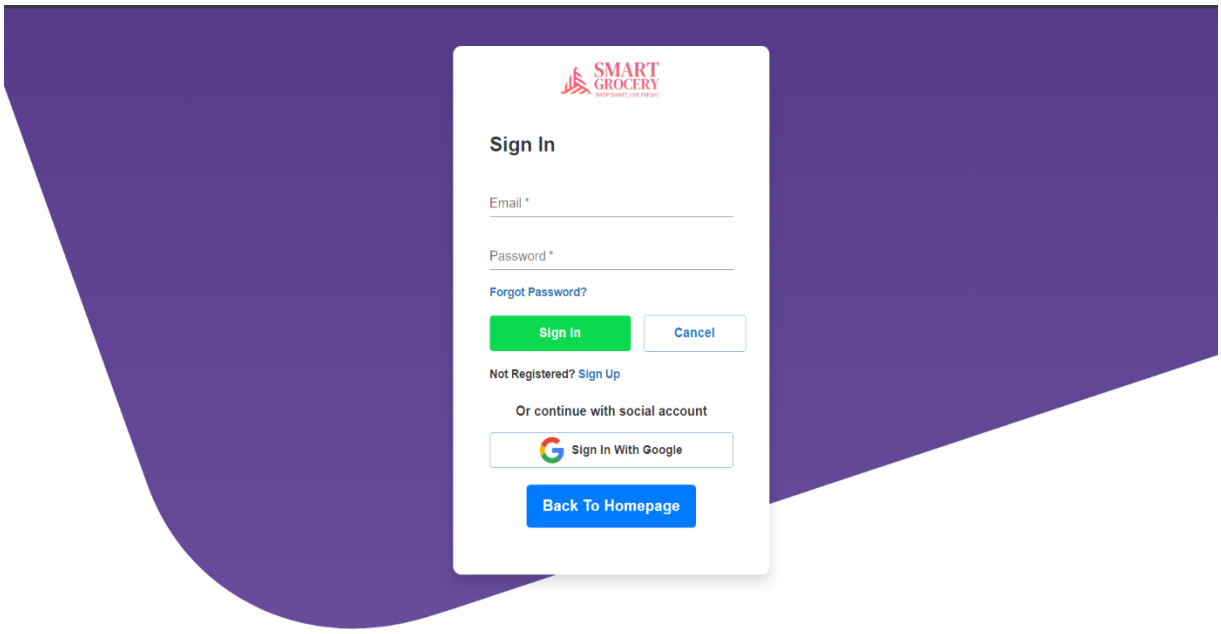


Fig 9.2.1 Login Page

9.2.2. Registration Page

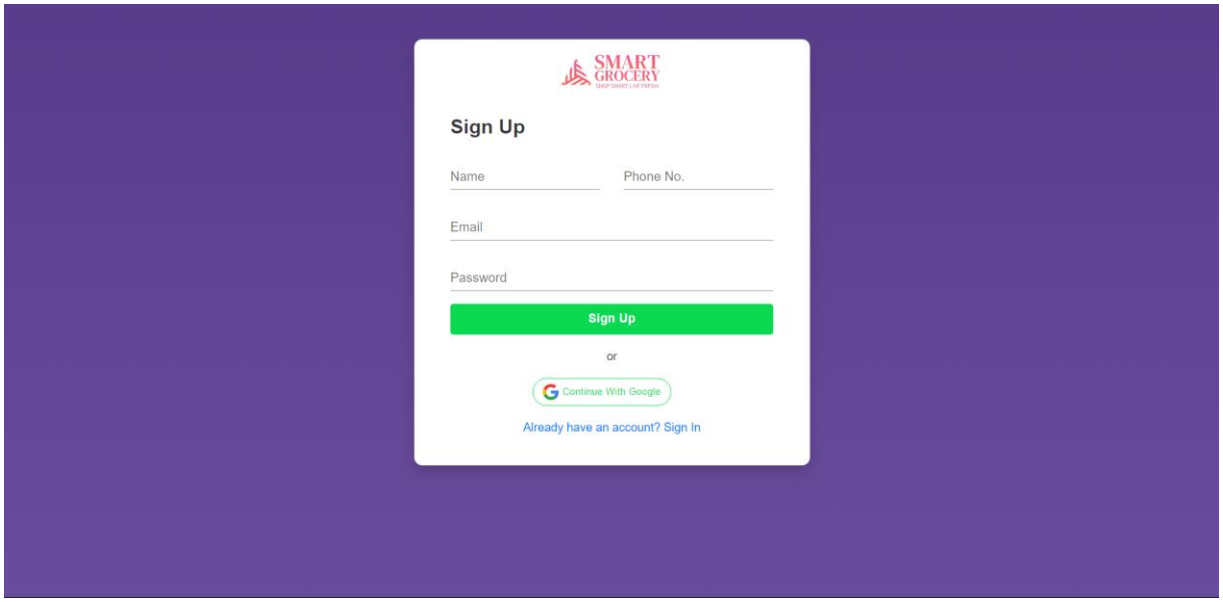


Fig 9.2.2 Registration Page

9.2.3. HomePage

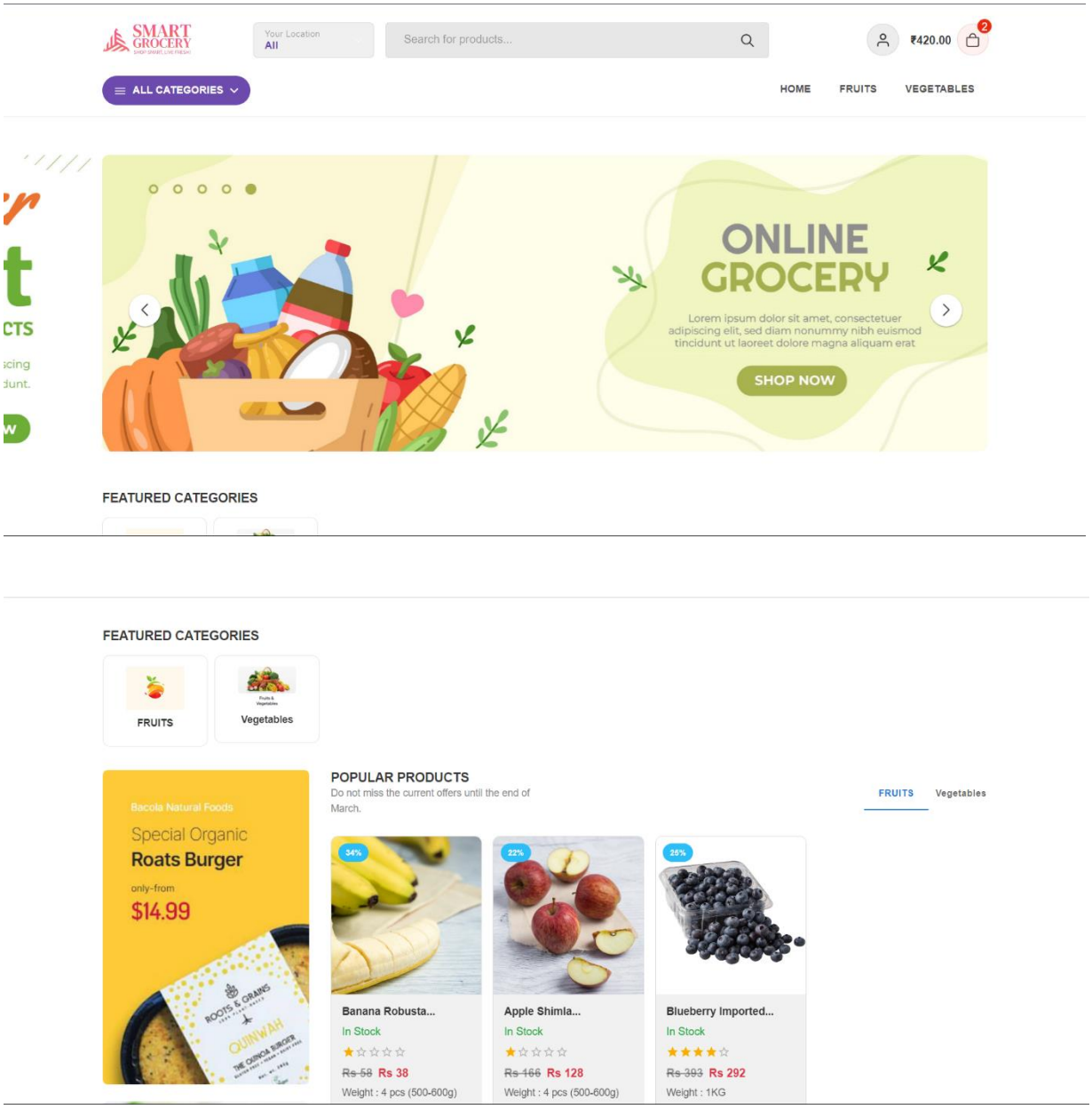


Fig 9.2.3 HomePage

9.2.4. ProductView Page

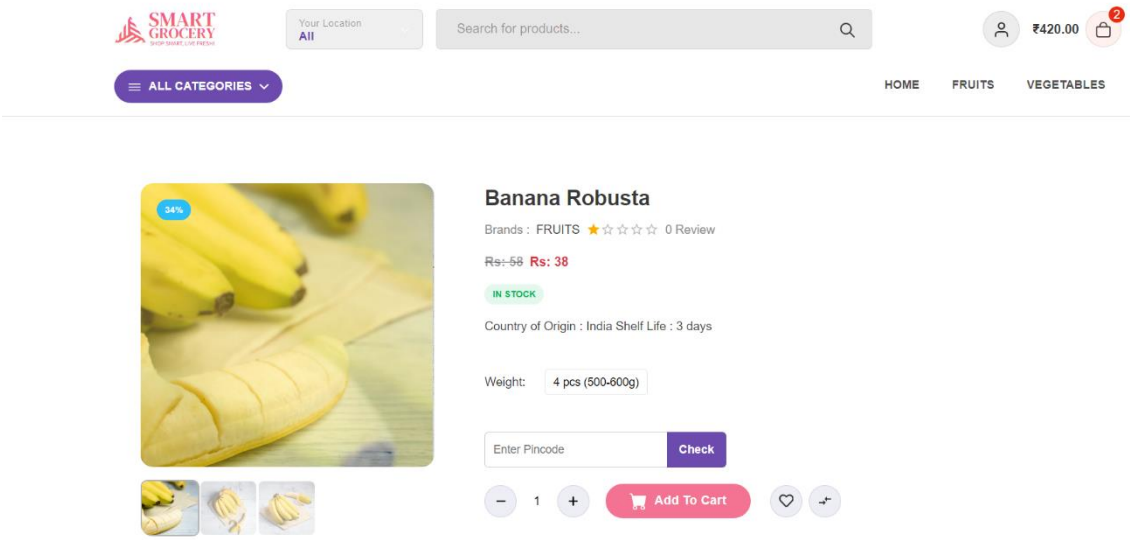


Fig 9.2.4. ProductView Page

9.2.5. Wishlist Page

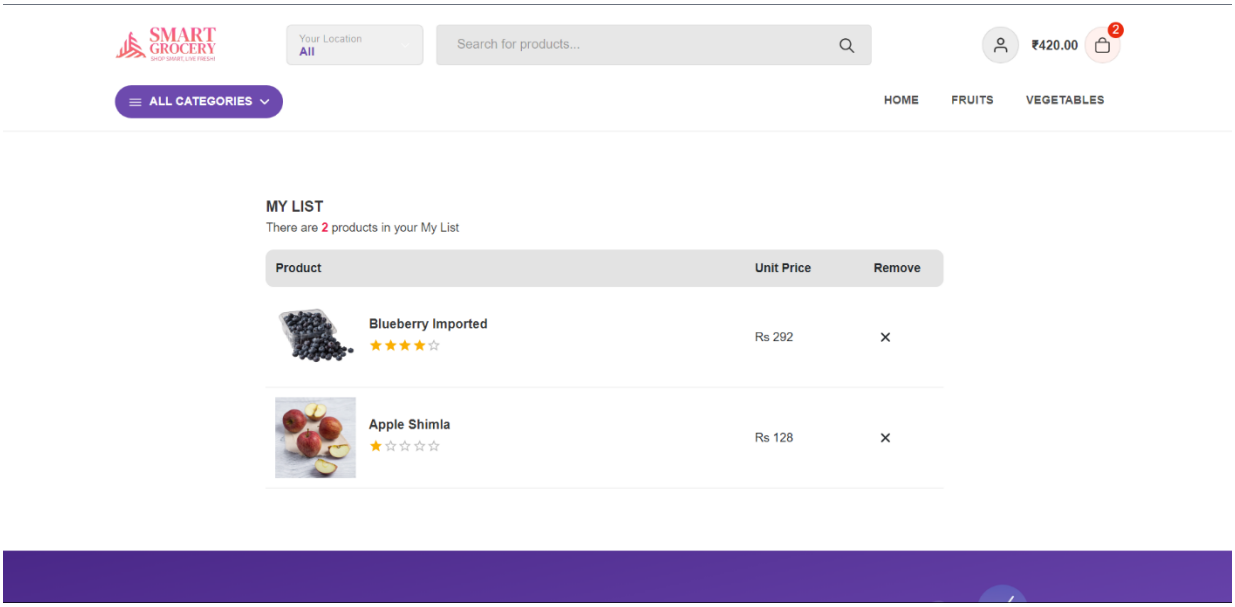


Fig 9.2.5. Wishlist Page

9.2.6. Cart Page

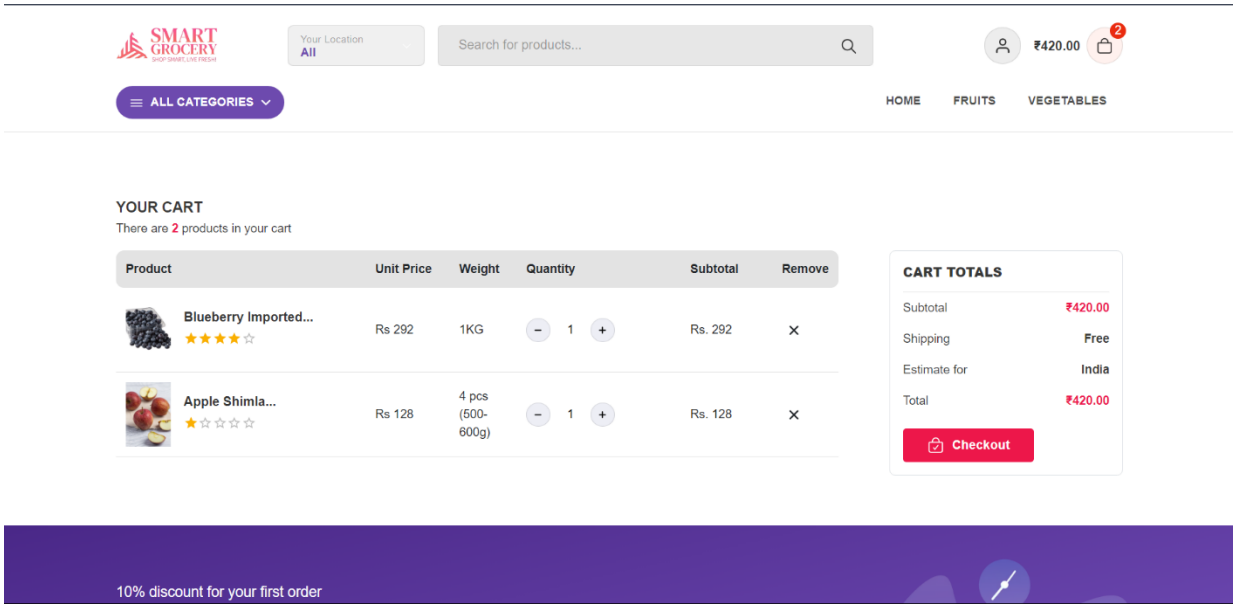


Fig 9.2.6. Cart Page

9.2.7. Orders Page

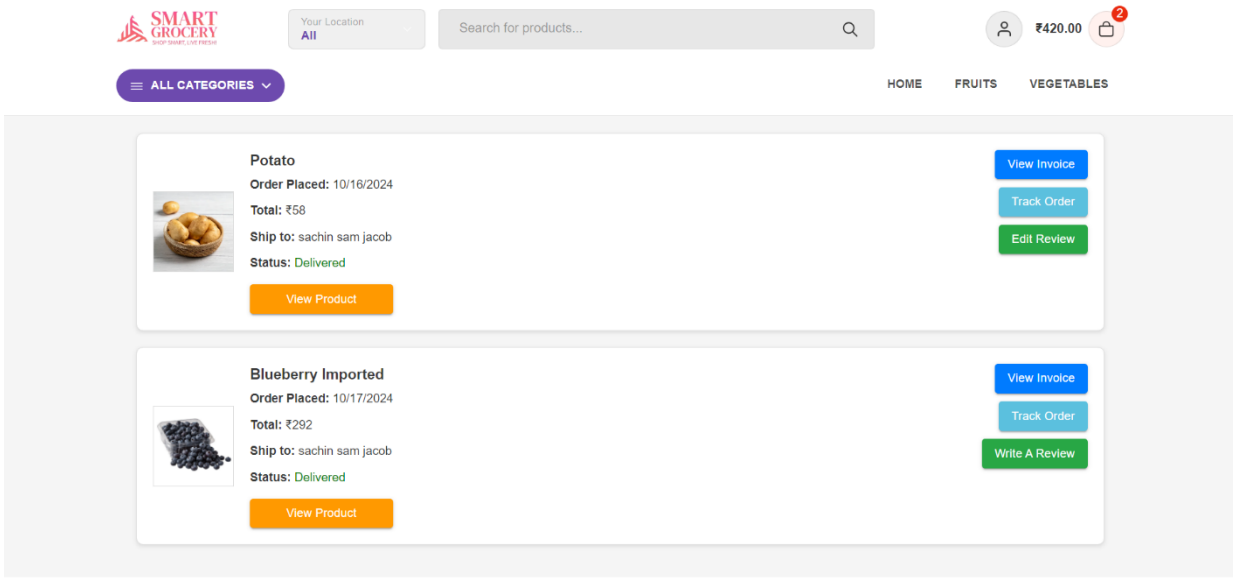


Fig 9.2.7. Orders Page

9.2.8.Checkout Page

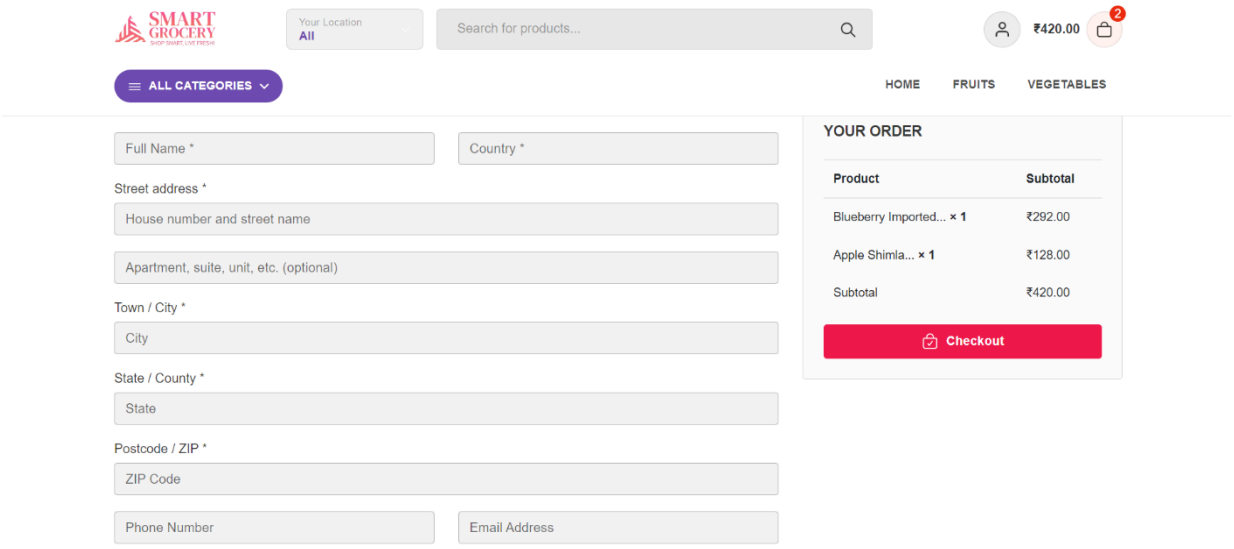


Fig 9.2.8. Checkout Page

9.2.9. Update Profile Page

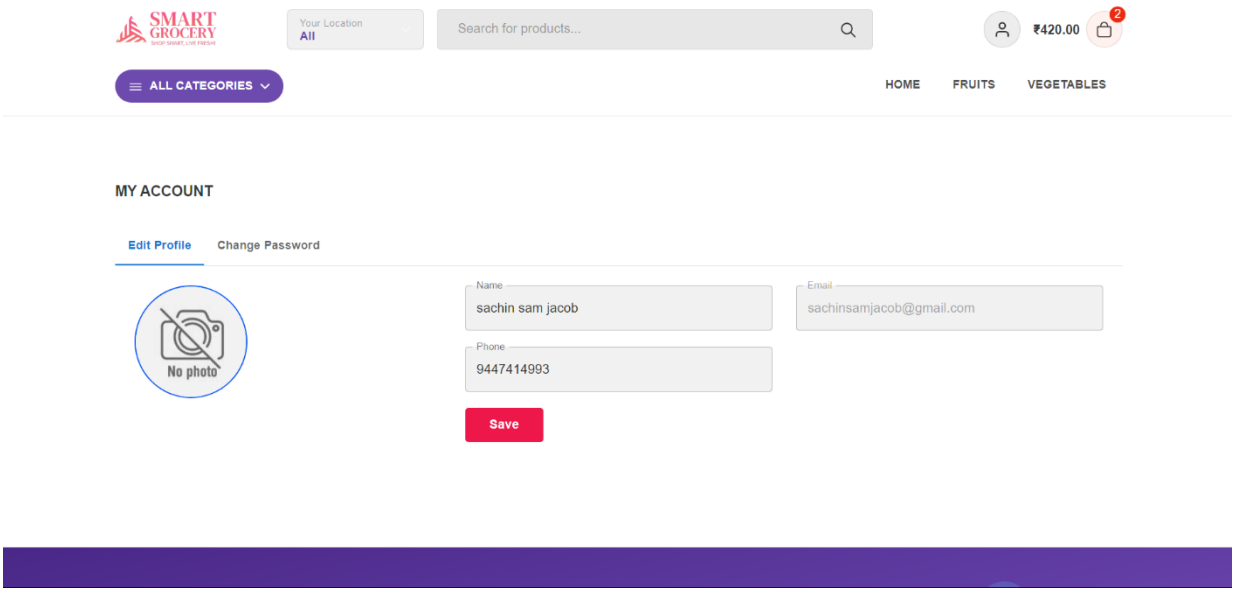


Fig 9.2.9. Update Profile Page

9.2.10. Change Password Page

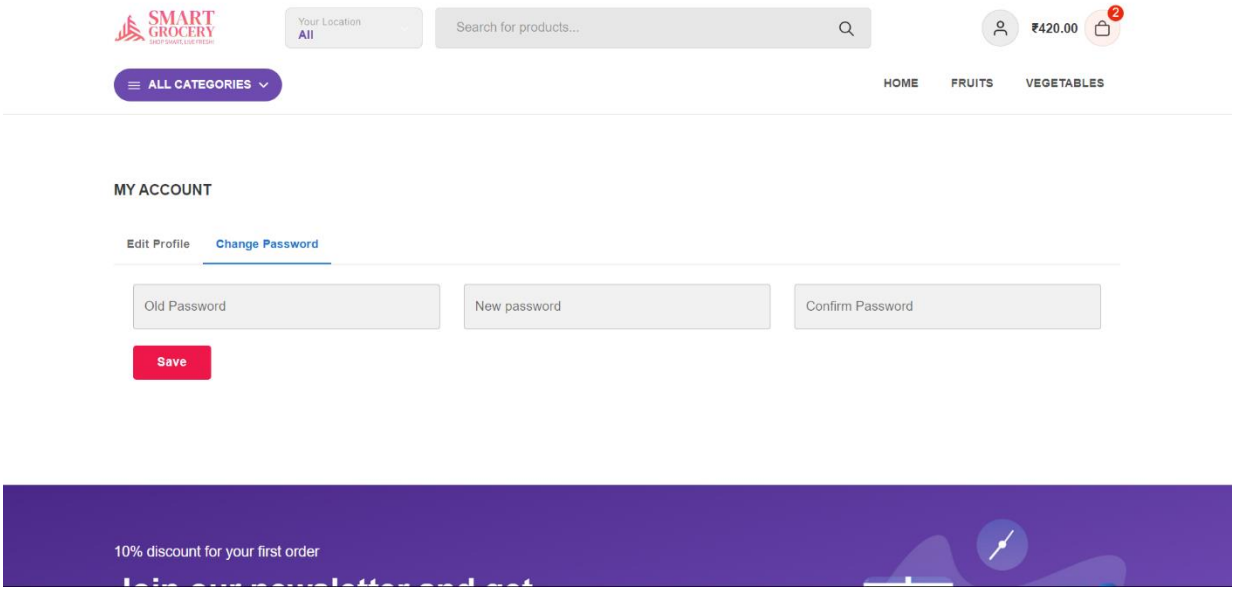


Fig 9.2.10. Change Password Page

9.2.11. Admin Dashboard

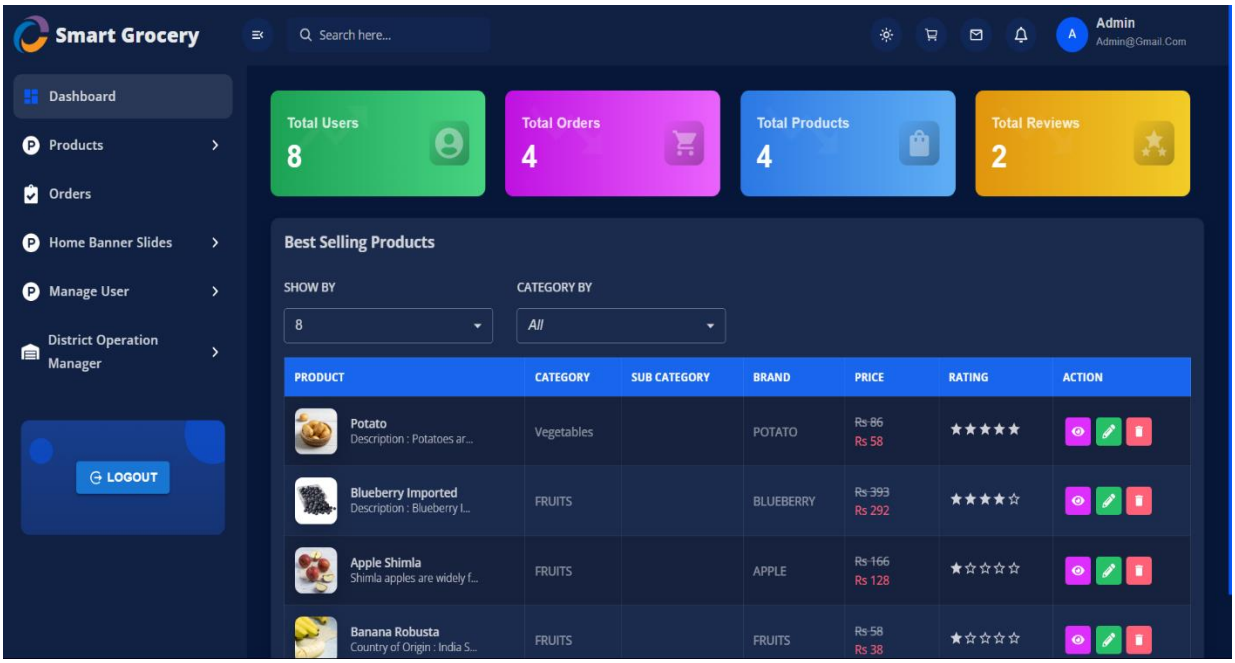


Fig 9.2.11. Admin Dashboard

9.2.12. District Operations Manager Dashboard

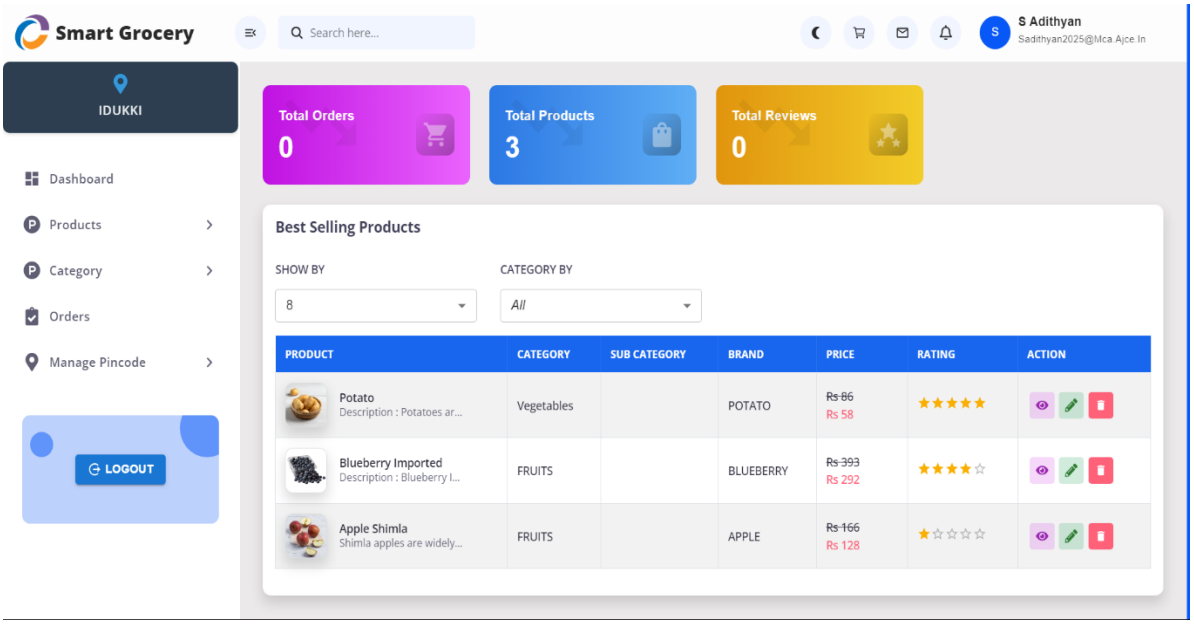


Fig 9.2.12. District Operations Manager Dashboard

Commits

main All users All time

Commits on Nov 8, 2024

made change in admin login page

`0e1a9c8`

sachinsam2024 committed 6 hours ago

made changes in package.json(admin) for hosting purpose

`e310a79`

sachinsam2024 committed 6 hours ago

made changes in package.json ,final project report,editproduct and add product in DOM

`b6f2760`

sachinsam2024 committed 6 hours ago

Commits on Nov 5, 2024

added render.yaml file

`021903b`

sachinsam2024 committed 2 days ago

Made changes to password bcrypt in forgot password,chnaged the styles of signup,login validation from homepage,pincode check in location header

`c205caf`

sachinsam2024 committed 2 days ago

Commits on Nov 4, 2024

edited package.json(server) for hosting

`fc481be`

sachinsam2024 committed 3 days ago

render.yaml added in server

6c1b556

sachinsam2024 committed 3 days ago

made changes in the session

9bcc547

sachinsam2024 committed 3 days ago

Commits on Oct 27, 2024

added district operation manager management to admin,added a preloader,made new sidebar for search result

f3d8b13

sachinsam2024 committed 2 weeks ago

added new user stock manager

86b132d

sachinsam2024 committed 2 weeks ago

Commits on Oct 18, 2024

made changes in addtocart message display

b29249d

sachinsam2024 committed 3 weeks ago

added review and cancel order functionality(completed)

96f063b

sachinsam2024 committed 3 weeks ago

Commits on Oct 16, 2024

made changes in the product listt in admin.added countinstock and location

1a5aa04

sachinsam2024 committed 3 weeks ago

added districts.js in admin and client,changed the location from countries to states(admin and client)

f971e96

sachinsam2024 committed 3 weeks ago

added a link to go back to homepage from loginpage

9bfa58a

sachinsam2024 committed 3 weeks ago

removed add review from productDetails page

4deebb9

sachinsam2024 committed 3 weeks ago

made change in out of stock fucntionality.if the product is in out of stock quantity box and add to cart options will not be available

9b919db

sachinsam2024 committed 3 weeks ago

changed the style of products listed in orders ,changed the format of invoice and removed the printing of products in products.j(server for testing)

51ab2ea

sachinsam2024 committed 3 weeks ago

Commits on Oct 15, 2024

added env in gitignore(admin)

5c57525

sachinsam2024 committed 3 weeks ago

added env in gitignore(server)

648bd07

sachinsam2024 committed 3 weeks ago

added env in gitignore

97eb1db

sachinsam2024 committed 3 weeks ago

Commits on Oct 14, 2024

added some styles in signup and changes in redirect to admin from user login

a5c0c65

sachinsam2024 committed 3 weeks ago

made changes in productDetails,api.js(added fetchDataApi),server.js(added orders),added api for order to be listed,added backend code for filterbyprice in Product.js

b85bd39

sachinsam2024 committed last month

added download invoice option in orders

680b07f

sachinsam2024 committed last month

listing of products based on category,filter by price and rating error has been fixed

c81aac1

sachinsam2024 committed last month

made changes in checkout (added upi)

aed4ba4

sachinsam2024 committed last month

added invoice component in App.js

233b2ee

sachinsam2024 committed last month

html2canvas added for reciept printing

4f8f0cf

sachinsam2024 committed last month

Commits on Oct 11, 2024

made the same delete to out of stock in products list(admin)

3e27f3c

sachinsam2024 committed last month

made changes in delete a product in admin(delete=Out of stock) and also fixed the count of review

8800dc9

sachinsam2024 committed last month

Commits on Oct 8, 2024

added update stock to product.js in server.made changes in mylist model(added and removed attribute),added validation in cart when product removed added to wishlist.made changes in mylist.js (routes),

11196ff

sachinsam2024 committed on Oct 8

changed api.js(res->data),added new attribute in cart model

3c948cf

sachinsam2024 committed on Oct 8

added weight to Productdetails and cart

e6b5c29

sachinsam2024 committed on Oct 8

made changes in cart and productitem page (Completed)

f56aee3

sachinsam2024 committed on Oct 8

Commits on Oct 4, 2024

PRODUCT REMOVED FROM CART ADDED TO WISHLIST(COMPLETED)

36dba95

sachinsam2024 committed on Oct 4

Previous Next >

Commits

main All users All time

Commits on Oct 4, 2024

List user and Block user functionalities completed(ADMIN SIDE)
78c8ca3
sachinsam2024 committed on Oct 4

Added Functionality of blocking and unblocking user by admin
a6cbb27
sachinsam2024 committed on Oct 4

made changes in the Signin and users model(Blocked Users)
cb3977f
sachinsam2024 committed on Oct 4

Commits on Sep 24, 2024

list user in manage user in admin (DONE)
05b982d
sachinsam2024 committed on Sep 24

added abstract to documentation
4f7e40b
sachinsam2024 committed on Sep 24

Made changes in manage user in admin"(code working)
d545c65
sachinsam2024 committed on Sep 24

made chages to manage user in admin dashboard(not finished)
08e5bef
sachinsam2024 committed on Sep 24

Commits on Sep 23, 2024

Added Documentation and added Manage user functionality to admin(only dropdown list.code need to be done)

9edb04d

sachinsam2024 committed on Sep 23

Made changes in Login and Signup of Admin

f656725

sachinsam2024 committed on Sep 23

Made changes to Additional Info(ProductDetails)

dd9982f

sachinsam2024 committed on Sep 23

Commits on Sep 22, 2024

made changes in Header

1fbb4c2

sachinsam2024 committed on Sep 22

Commits on Sep 18, 2024

Second Commit(Deleted Images)

5629e53

sachinsam2024 committed on Sep 18

First Commit

3e73bbb

sachinsam2024 committed on Sep 18

Previous Next