Sachin Sam JACOB

Sachin_Main_Project_Report_.docx

Project

MCA_Project

Amal Jyothi College of Engineering

Document Details

Submission ID

trn:oid:::1:3191391174

Submission Date

Mar 23, 2025, 1:25 PM GMT+5:30

Download Date

Mar 24, 2025, 2:04 PM GMT+5:30

File Name

Sachin_Main_Project_Report_.docx

File Size

9.8 MB

126 Pages

15,879 Words

106,258 Characters

*% detected as AI

AI detection includes the possibility of false positives. Although some text in this submission is likely AI generated, scores below the 20% threshold are not surfaced because they have a higher likelihood of false positives.

Caution: Review required.

It is essential to understand the limitations of AI detection before making decisions about a student's work. We encourage you to learn more about Turnitin's AI detection capabilities before using the tool.

Frequently Asked Questions

How should I interpret Turnitin's AI writing percentage and false positives?

The percentage shown in the AI writing report is the amount of qualifying text within the submission that Turnitin's AI writing detection model determines was either likely AI-generated text from a large-language model or likely AI-generated text that was likely revised using an AI-paraphrase tool or word spinner.

False positives (incorrectly flagging human-written text as AI-generated) are a possibility in AI models.

AI detection scores under 20%, which we do not surface in new reports, have a higher likelihood of false positives. To reduce the likelihood of misinterpretation, no score or highlights are attributed and are indicated with an asterisk in the report (*%).

The AI writing percentage should not be the sole basis to determine whether misconduct has occurred. The reviewer/instructor should use the percentage as a means to start a formative conversation with their student and/or use it to examine the submitted assignment in accordance with their school's policies.

What does 'qualifying text' mean?

Our model only processes qualifying text in the form of long-form writing. Long-form writing means individual sentences contained in paragraphs that make up a longer piece of written work, such as an essay, a dissertation, or an article, etc. Qualifying text that has been determined to be likely AI-generated will be highlighted in cyan in the submission, and likely AI-generated and then likely AI-paraphrased will be highlighted purple.

Non-qualifying text, such as bullet points, annotated bibliographies, etc., will not be processed and can create disparity between the submission highlights and the percentage shown.

CHAPTER 1

INTRODUCTION

1.1 PROJECT OVERVIEW

The Smart Grocery platform builds an advanced AI-based system which uses artificial intelligence (AI) together with machine learning and automated inventory management and dynamic pricing to transform the grocery shopping interaction. Users benefit from Smart Grocery because it brings AI-generated recipe recommendations and optimized supply chain management alongside personalized recommendations through its service model which differs from traditional online grocery shopping platforms.

The fast way we live now causes customers to prioritize fast delivery services while looking for shopping platforms that offer personalized recommendations. The Smart Grocery digital marketplace resolves user needs through a combined solution of easy grocery shopping and personalized meal planning based on selected cart items. Each group of users on this platform obtains unique capabilities through their assigned role either as administrator or customer or supplier or district operations manager to manage operations successfully.

The AI-Powered Recipe Generation system allows users to receive food preparation recommendations depending on what groceries they have in their basket to achieve better value for their purchase and minimize food excess. Although Dynamic Pricing and Personalized Discounts form part of the system they function through the adjustment of product prices depending on stock levels and customer behavior data and market demand. Users can utilize Voice-Activated Assistance for hands-free shopping on the platform because the system includes an AI-powered voice command feature.

General operational efficiency comes from the Admin Panel through real-time analytics which shows administrators user activity data together with product performance metrics and inventory numbers. Automated inventory tracking together with restocking alerts through the system enables suppliers and district operations managers to minimize supply chain disruptions. The Secure Payment & Checkout system enables customers to conduct their business with secure payment choices through a range of options.

AI-powered algorithms within Smart Grocery enable platform users to receive custom recommendations and operate more efficiently. AI suggests recommended products by leveraging user behavior data and purchase history as well as predictive analytics regulates product inventories to avoid stockout issues.

The Smart Grocery platform is set to transform the online grocery market through its ability to make recommendations, personalize prices and operate AI-based menus and sustain live supply chain operations. The integration of modern technology together with intuitive interfaces helps this project establish the most efficient and fastest grocery shopping experience available today.

1.2 PROJECT SPECIFICATION

This platform named Smart Grocery improves regular grocery shopping through innovative technological methods together with AI functionality and personal features for users. Through its streamlined online service the platform delivers individualized shopping capability and operational excellence for inventory control. The platform contains dedicated user roles named Administrators, Customers, Suppliers, and District Operations Managers for achieving proper functionality together with effective management.

User Roles & Functionalities

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 and also secure login with Face Authentication for enhanced security
- **Profile Management:** Manage personal details, delivery addresses, and payment methods.
- **Product Browsing & Shopping:** Browse groceries, search, and filter products based on brand, price, and category.
- **Smart Shopping Cart:** Add/remove items, adjust quantity, and save products for later.
- **AI-Powered Recipe Generation:** Generate meal plans based on cart items and suggest alternative ingredients.
- **Secure Checkout:** Multiple payment options, including credit/debit cards, UPI, wallets, and cash on delivery.
- **Feedback & Reviews:** Leave product ratings and reviews to enhance shopping experiences.
-

District Operations Managers:

- **Inventory Management:** Monitor stock levels and ensure sufficient supply for assigned regions.
- **Order Fulfillment & Logistics:** Coordinate deliveries, returns, and supply chain processes.
- **Region Management:** Modify service coverage by adding/removing delivery zones (by pincodes).
- **Supplier Coordination:** Communicate with suppliers to manage restocking and stock updates.
- **Inventory Oversight:** Track stock levels and coordinate with suppliers for restocking.
- **Report Generation:** View system analytics, sales trends, and user activity reports.

Suppliers:

- **Product Management:** Add, update, and manage product listings with stock availability.
- **Order Fulfillment:** Process bulk orders received from district operations managers.
- **Automated Restocking Alerts:** Receive automatic notifications when stock levels drop below a threshold.

CHAPTER 2

SYSTEM STUDY

2.1 INTRODUCTION

The Smart Grocery brings together artificial intelligence with modern technology features that enhance grocery shopping capabilities through AI-based recommendation systems combined with voice guidance functions and dynamic price control mechanisms and automated inventory automation. The system delivers individualized streamlined operations to administrators and suppliers and district operations managers alongside convenient shopping convenience to customers.

Smart Grocery operates a complete online grocery marketplace which allows users to look for and acquire their necessary grocery items across multiple product categories. The platform serves different user groups including customers while providing administrators and suppliers and district operations managers their own enhanced operational tools to improve their work and outcome quality.

The application enables Face-Authenticated Login for authentication security and offers features such as a Product Catalog with Smart Shopping Cart and AI-Powered Recipe Generation that generates cooking recommendations from cart contents. The pricing strategy of Dynamic Pricing & Personalized Discounts bases product prices through connection with available stock amounts and consumer demand patterns. The system features built-in voice-activated assistance which enables users to communicate with the platform through voice commands.

The administrative team observes user profiles enables backup control of product catalogs together with system performance evaluation. District Operations Managers conduct supply chain management activities as well as handle inventory operations within their regions and oversee supplier commitments. The system automatically notifies suppliers to handle bulk order management and provides them restocking information.

Predictive stock management and smart meal planning allow Smart Grocery Shopping to establish new industry standards through their highly intelligent automated personalized shopping solution.

2.2 EXISTING SYSTEM

Smart Grocery intends to resolve the multiple drawbacks found in traditional grocery shopping performed both in stores and through conventional e-commerce channels. The traditional grocery shopping process requires consumers to undertake tedious activities including visiting store aisles and doing manual product comparison and managing their grocery lists thus making the experience time-consuming for busy shoppers. The introduction of online shopping through grocery chains and retailers has failed to include personalized recommendation systems along with dynamic pricing features and integrated meal planning solutions because of which users face an imperfect 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 standouts 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 behaviors, 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 main function of feasibility studies aims to evaluate if proposed projects will succeed in fulfilling organizational objectives while utilizing available resources along with labor and time constraints. A crucial examination of the project enables developers alongside decision-makers to acquire vital forecasting about project viability and potential outcomes. The examination of multiple system aspects during the feasibility study determines project feasibility alongside potential success while evaluating organizational impact and resource optimization.

Multiple essential factors form the basis for determining the feasibility of a proposed project during the decision-making phase.

- Technical Feasibility
- Behavioral Feasibility
- Economic Feasibility

An effective feasibility study creates knowledge for decision-makers who can use this information to assess project success possibilities. A feasibility study helps stakeholders discover potential project aspects so they can create proper risk management solutions.

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 behaviors 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 planned system Smart Grocery provides consumers and suppliers with a full-scale e-grocery shopping service through its platform. The system will launch with three core capabilities including registration features for users, product listing administration and transaction security options. The project's student-design emphasizes growth potential so developers can later include subscription services together with community functions and analytical tools.

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?

Prabhu P Kalesan, Branch Store Manager at SMART BAZAAR.

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

1. **How do you currently manage inventory in your grocery store?**

   We manage inventory systematically using a stock management system. Each product has a predefined Minimum Base Quantity (MBQ). When stock levels drop below the MBQ, an automatic restocking request is sent to the supplier, ensuring timely replenishment and preventing stockouts.

2. **What challenges do you face in tracking stock levels and reordering products?**

   Our automated system helps track stock levels efficiently, but occasional challenges arise when suppliers delay shipments or when there are sudden spikes in demand. We monitor these patterns and adjust the MBQ for frequently purchased products to reduce shortages**.**

3. **How do you gather information about customer preferences and shopping habits?**

   We track sales data and customer purchase history through our system. This allows us to identify best-selling products, seasonal trends, and customer preferences. We also collect feedback directly from customers to improve our stock selection.

4. **Do you offer any form of online shopping, and if so, what challenges do you face?**

   Yes, we offer online shopping with home delivery and in-store pickup options. Managing real-time stock updates across both online and offline platforms can be challenging, but we synchronize inventory data to prevent overselling or stock mismatches.

5. **How do you manage deliveries, and what logistical issues do you encounter?**

   We have a delivery management system that assigns delivery slots and tracks orders in real-time. Challenges include route optimization, ensuring perishable goods remain fresh during transit, and managing high delivery demands during peak hours.

6. **What payment methods do you currently accept, and how secure are they?**

   We accept multiple payment methods, including cash, credit/debit cards, UPI, and digital wallets. Our system integrates with secure payment gateways to ensure encryption and fraud protection, reducing risks associated with online transactions.

7. **How do you handle customer inquiries and complaints?**

   We have a dedicated customer support system that allows customers to reach us via phone, email, and live chat. Complaints are logged into our system, and each query is tracked until resolution, ensuring timely responses and improved customer satisfaction.

8. **How often do you update your product catalog, and how do you decide on the products to stock?**

   Our product catalog is updated regularly based on sales data, seasonal demands, and supplier availability. The system suggests adding or removing products based on customer demand and purchasing patterns.

9. **What marketing strategies do you use to attract and retain customers?**

   We use digital marketing, personalized promotions, and loyalty programs. Customers receive targeted discounts based on their shopping history, and we send promotional offers through SMS, emails, and mobile app notifications.

10. **Are you open to adopting new technologies to improve your operations, and what concerns might you have?**

    Yes, we are always looking for ways to improve efficiency. Our main concerns include ensuring smooth integration with existing systems, training staff to adapt to new technologies, and managing implementation costs while maintaining business continuity.

3.1.5 Geotagged Photograph

3.2 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, Python
Database	-	MongoDB Atlas
Client on PC	-	Windows 10 and above.
Technologies used	-	JS, HTML5, CSS, Cloudinary, Face API, Gemini API, Hugging Face API, AWS EC2 Instance, Firebase Authentication, Image Recognition, Natural Language Processing, TensorFlow.

3.3 SOFTWARE DESCRIPTION

3.3.1 React JS

React (React. js) is an open-source JavaScript library for building UI, specifically full immersion responsive single page applications (SPAs) and high responsiveness with good-performing user experiences. React was created by Facebook and is still maintained by them. Second, React allows developers to create massive web applications and modify the state of data without reloading the page. The idea is component-based — in other words, what a developer can do is break an application UI into tiny reusable parts (we have names for them) components that work autonomously with their own state and logic.

The React virtual DOM — so that it can quickly render just the portions of your page you need to, shortening the change for every little bit of the page. Rather than whole DOM being updated after every state change in the framework, React first dismantle (un-build) the stateful component to this inside out shapped construct of the virtual DOM in memory.

With data mutating, React does a "diff" between the actual DOM and this virtual DOM for everything that changed, allowing the render to update only what needs updated less occasionally; this is how DOM updates become less costly.

React likes it that way: purely declarative programming, so you can deduce pretty quickly the way the app works. A declarative syntax, we say what UI should be for current state and React will render it. Way nicer and more reliable especially with big apps.

3.3.2 MongoDB Atlas

MongoDB Atlas is a fully managed NoSQL database service in cloud which aims at simplifying the scale, flexibility and ease for Developers.As a Database-as-a-Service solution it does all of the heavy lifting in database management by managing things like provisioning, scaling up backups and security intentions. Unlike the restrictive structured tables of traditional relational databases (RDBMS) where one stores information in MongoDB Atlas, BSON (a binary file format for structuring data) like formatted documents, which is more free-form and dynamic evolving along with application needs.

MongoDB Atlas is known as being exceptionally easy to scale in a way that most wouldn´t dream of scaling their applications on unstructured (or semi-structured) data. It is capable of horizontal scaling (sharding), meaning that provisioning can be spread out across more cloud regions and instances for massive cloud, availability as well increased performance as workloads increase. As a distributed environment, developers can most truly easily deploy and manage databases with the help of multi-cloud-abilities for AWS, Google Cloud (GCP) and Microsoft Azure.

MongoDB Atlas: MongoDB Atlas provides you with comprehensive querying abilities, allowing you to carry out complex searches, aggregations, and real-time analysis over big datasets. High Performance data retrieval using indexing, text search and geospatial queries as well It also comes with its faulty replica set architecture to provide automatic failover and continuous backups which deliver an extremely high data redundancy.

MongoDB Atlas has native security in that encryption of data at rest or in flight with end-to-end encryption, automated compliance per industry standard, RBAC (Role Based Access Control).

3.3.3 Node Js and Express Js

Node.js represents an open-source environment which operates as a server-side JavaScript runtime platform for developers to execute JavaScript code on server capabilities. Node uses the V8 engine from Chrome to establish its JavaScript runtime environment which enables developers to create high-speed server-side applications using JavaScript code. Node.js leverages an event-driven and non-blocking I/O model for operating efficient multiple requests simultaneously pertaining to real-time apps and API and chat-intensive applications.

Nodejs implements an Asynchronous single-threaded model through Express as its main framework structure while operating differently from standard server environments that depend on multi-threaded architecture and use callbacks for async functions to process requests. With its capabilities Node handles thousands of simultaneous requests at a time therefore making it appropriate for applications demanding fast processing and scalability. The package ecosystem of Node JavaScript brings many benefits through its management system maintained by npm (Node Package Manager) which stands as one of the world's largest open-source repositories for developers to incorporate easily.

Express.js operates as a minimalistic web framework which runs above Nodejs so developers can handle web servers and APIs effectively. Node.js provides basic server sides JavaScript capabilities yet Express.js brings an application framework which automates common coding tasks for HTTP requests as well as routing and middleware implementation. The combination of simplicity with performance along with flexibility allows Express to become the leading choice compared to other web application frameworks.

Express.js developer routing functions let users connect specific handler functions to particular URL routes through its main routing feature. Express enables developers to create modular code that handles HTTP methods (GET, POST, PUT, DELETE) as well as paths more easily. The middleware functionality enables developers to separate functionality into distinct components for request-response operations including authentication logging and request parsing and error handling facilities.

Node.js teams up with Express.js to create an effective system that handles full-stack JavaScript applications by enabling Node.js operation as a server-side execution environment alongside Express.js implementation for API development.

CHAPTER 4

SYSTEM DESIGN

4.1 INTRODUCTION

Engineering systems start their development during the design stage by using creative methods. The creation of well-dueled designs functions as an essential factor to guarantee system success. Design represents the methodology of implementing diverse structural frameworks and rules to document sufficient operational details needed for physical system development. Multiple approaches are utilized to generate information about a system or machine that provides enough specifics for building it. Every software development method depends on a fundamental design stage which stands as an essential requirement throughout the entire process. System design constitutes the production of architectural drawings to construct products or machine systems. The development of software requires detailed planning which leads to optimal operational performance together with accurate results. The programmers and those handling the database become the primary focus in the design phase after the user-focused period ends. System development requires completion of two major procedures: Logical Design and Physical Design.

4.2 UML DIAGRAM

The standardized language called Unified Modeling Language helps understand software system elements through specification and visualization along with system construction and documentation. OMG established itself as the governing body which developed UML when they showed the initial draft of UML 1.0 specification to their members in January 1997. UML functions differently than conventional programming languages due to its non-language nature contrary to C++, Java and COBOL. As a graphic language UML functions as a building device to develop software blueprints.

Software systems use UML as their general-purpose modeling language to create visual diagrams for system representation and documentation and specification and construction development. UML serves its main purpose by modeling software systems but demonstrates applications outside this basic usage scope. organizations use this method both to model and comprehend software processes and other non-computing operational systems such as production workflows.

UML operates as a modeling tool instead of a programming language although various tools can use diagram data to create source code for multiple programming languages. The core component of UML includes eight main diagrams which help developers display system characteristics.

- 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 serves as an organizational tool to understand requirements for system development specifically when building product delivery websites. Use case diagrams in the Unified Modeling Language serve as tools to represent these models which standardize model creation for systems and real-world entities.

Main elements in a use case diagram include the following components:

- The boundary component defines system areas to differentiate between system contents and outside elements.
- The diagram contains Actors to depict entities and people who carry out defined roles in the system.
- The interactions between different people or elements in specific scenarios or problems.
- Use case diagrams serve mainly to establish functional specifications of systems as their primary objective.

To establish successful use case diagrams guidelines, need to be followed:

- The diagram needs use cases and actors which receive descriptive names.
- The technical design team ensures all relationships along with dependencies receive proper definition.
- Including only necessary relationships for the diagram's clarity.
- Additional explanatory notes serve to explain necessary system information when needed.

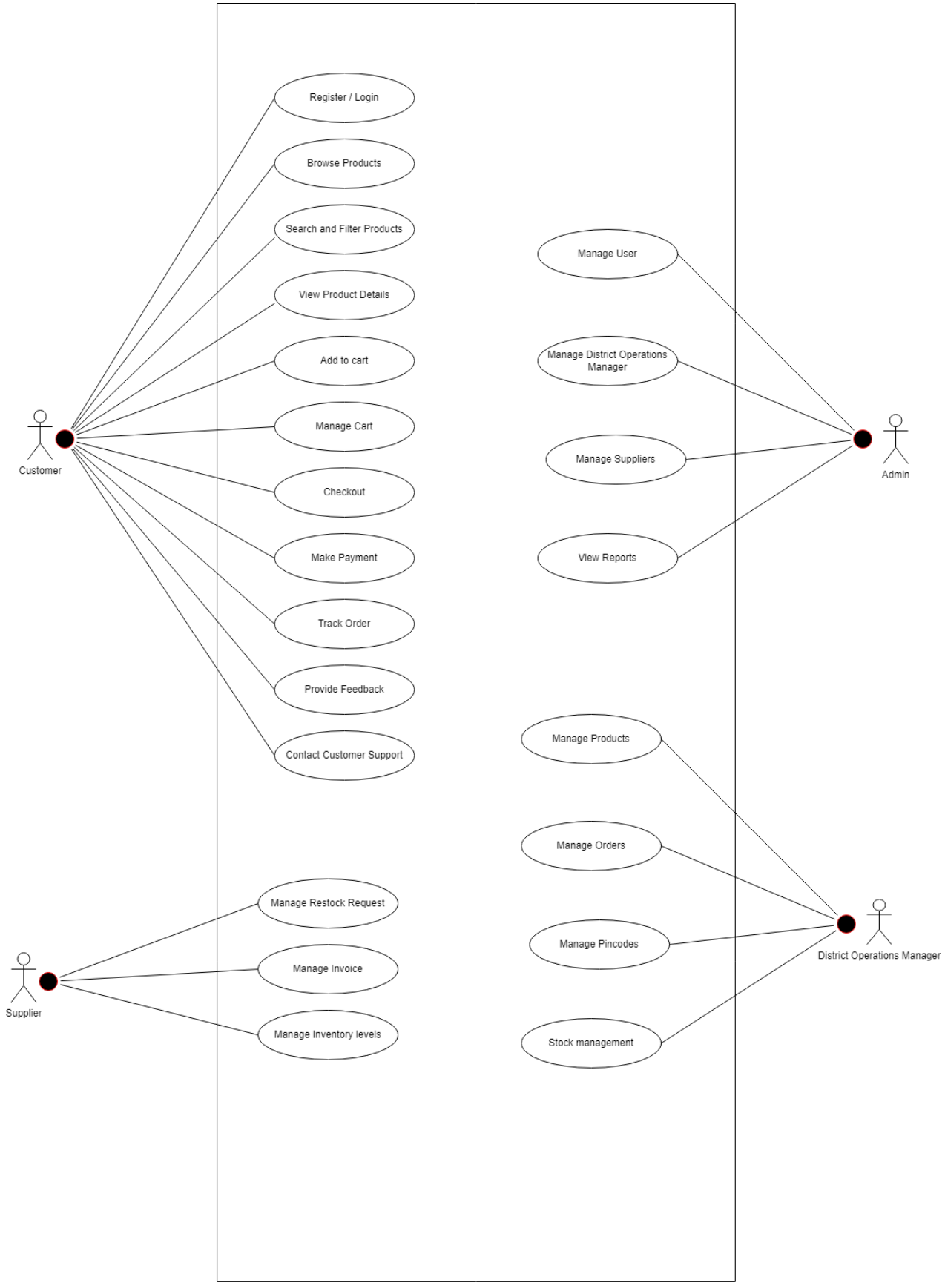


Fig 4.2.1 Use Case Diagram

4.2.2 SEQUENCE DIAGRAM

The sequence diagram represents how objects interact one after the other to present the sequential process of events. The diagram serves multiple names as event diagrams and event scenarios. Sequence diagrams provide the necessary understanding of which system components and their chronological behavior in collaboration with each other. Sequence diagrams serve business professionals together with software developers for requirements understanding and visual representation of new and existing system needs.

Sequence Diagram Notations:

I. Within UML diagrams actors depict all users who access system functions through its components. During UML diagram depiction actors do not show up because they remain outside the boundaries of the modeling system. They fulfill role-based participation by representing human participants alongside outside organizations in story-based efforts. Sequences in diagram boxes show actors as simple stick figures. Through diagramming it becomes possible to show multiple actors during the sequence-based representation of events.

II. The top section of sequence diagrams displays components belonging to lifelines using a representational format.

III. Objects transmit messages to each other through a sequential order which runs along the lifeline for communication purposes. The core structure of a sequence diagram depends on arrows to display communication messages which serve as its essential elements.

Guards in UML serve as the mechanism for showing different system conditions. Software developers utilize guards to control message distribution based on particular conditions thereby obtaining a better understanding of system or process regulations.

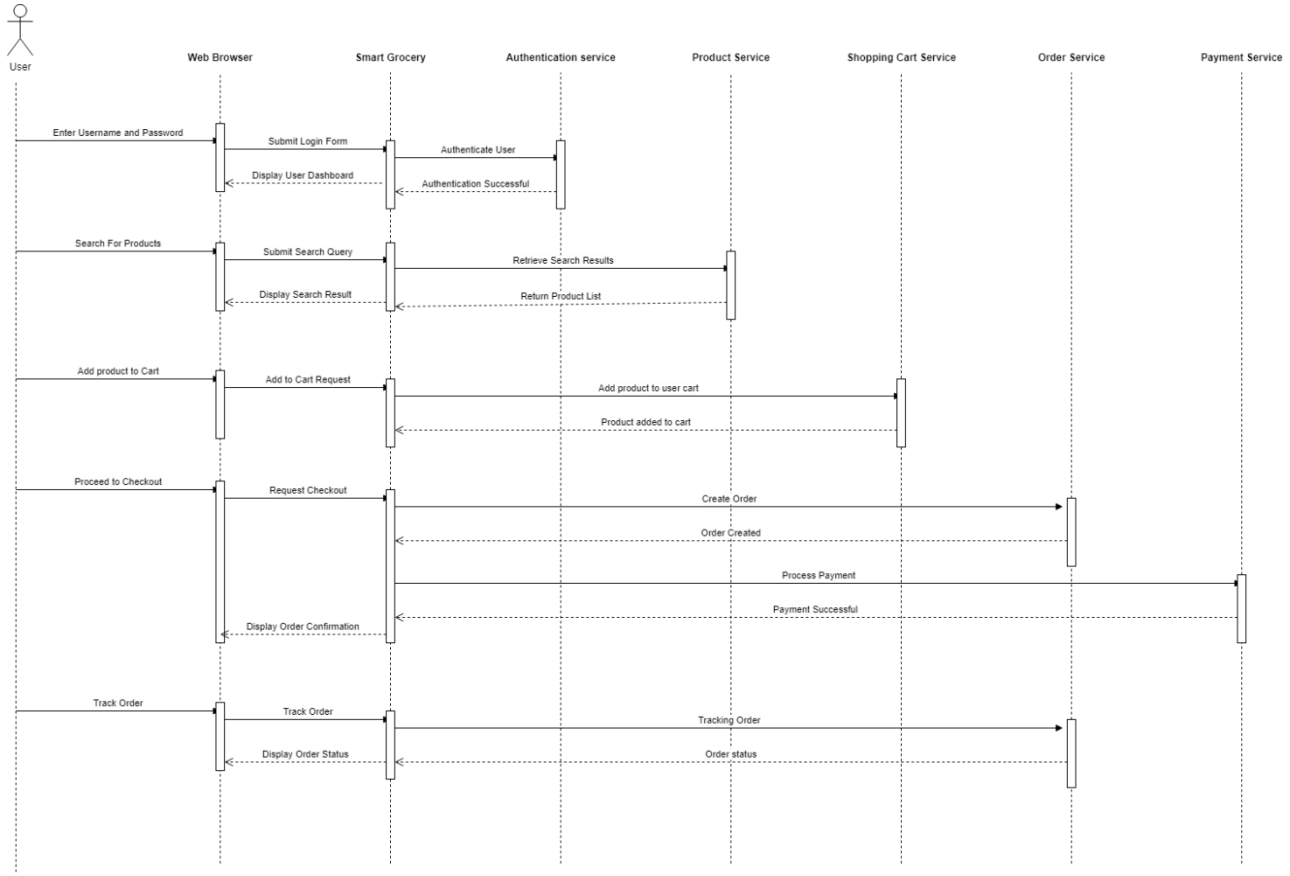


Fig 4.2.2 Sequence Diagram

4.2.3 State Chart Diagram

A state machine diagram which is known as a state chart provides visuals of object states throughout systems along with their state transition order. Through this record the system maintains its behavior to show how different entities work together no matter if they represent teams or students or large gatherings or whole organizations. State machine diagrams function as vital depiction methods which show systems' component interactions through clear object evolution tracking upon events while demonstrating the multiple circumstances that affect each entity or component.

Notations within a state machine diagram encompass:

- A black circle acts as the first symbol which indicates when processes begin.
- A final state represents the complete process termination through the use of a filled circle which rests inside another circle.
- The decision box takes diamond form to assist guard evaluation during decision processes.
- Any change affecting authority or state because of an event constitutes a transition. Each transition contains labels on its arrows which explain the triggering events.

- A state box shows the current status of an element inside a group at a particular time. The graphical representation shows these elements inside rounded-corner rectangular shapes.

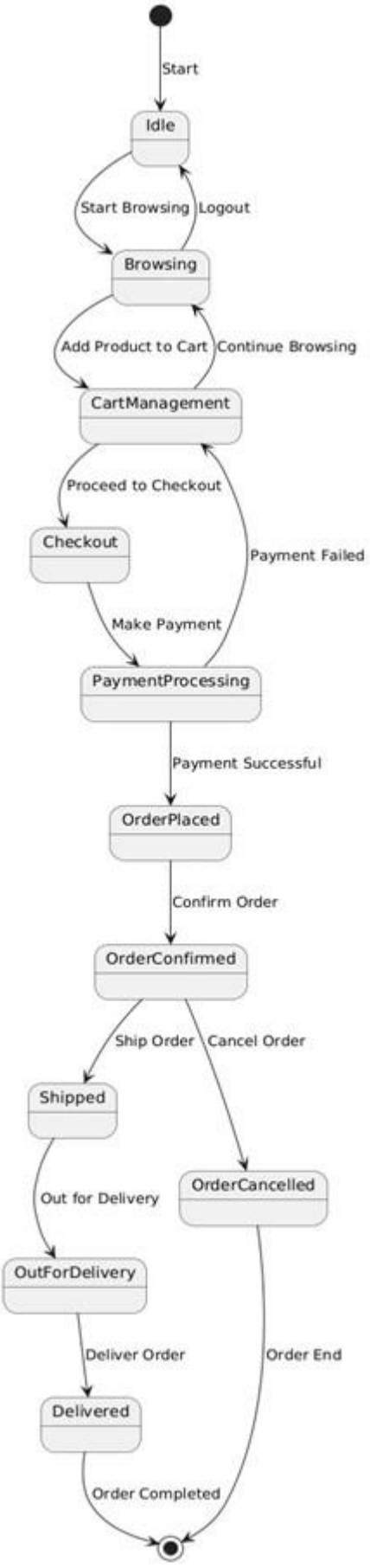


Fig 4.2.3 State Chart Diagram

4.2.4 Activity Diagram

The visual representation known as an activity diagram shows the progression of combined or sequential events. The diagram functions to display how processes link with each other while showing how activities move between tasks. The nature of activity diagrams revolves around sequencing responsibilities because these visuals demonstrate three different task sequences including sequential processing alongside parallel processing and alternative workflow paths. The flow elements within activity diagrams consist of forks and join nodes which support systems that demonstrate specific system functionalities.

Activity diagrams are built from specific essential elements including:

- The method of grouping behavioral units into multiple interconnected steps constitutes activities in diagram design. The lines between these actions display the logical order of activities that occur step by step. A process consists of several elements which include its main functions together with controlling factors and necessary resources that perform the tasks.
- Activity Partition/Swim Lane serves to organize tasks according to types through rows or columns thus improving diagram modularity. Each activity diagram does not require swim lanes which can be placed either vertically or horizontally.
- Task segments can operate at the same time through the implementation of fork nodes. The diagram represents an expansion point from a single input to multiple outputs as it mirrors several elements that shape decisions.
- Join nodes differ from fork nodes through a Logical AND operation that ensures all incoming data streams meet at one collective point for synchronization purposes.

The primary visual symbols used in Activity Diagrams consist of:

- Initial State: Depicting the beginning or first step in the process.
- A heading located at the end signifies that all procedures have finished with no upcoming developments.
- The Decision Box helps maintain structured paths for activities during the process.
- An Action Box displays the particular operational tasks and actions which need execution throughout the process.

Fig 4.2.4 Activity Diagram

4.2.5 Class Diagram

A dormant system requires class diagrams as static application blueprints which represent its components together with their relations. A class diagram reveals system structure through an illustration that depicts all its elements and their connection patterns. A class diagram functions as a visual documentation tool in software development to help teams generate operational applications.

Important Parts of The Class Diagram

- System Architecture A class diagram is a top-level view of the software system or software part that shows its structure, relationships and interactions. It is an organizational container for names, attributes and methods that makes things easier in software development,
- Structural Visualization: The class diagram is a visual representation from Structural Diagrams consolidate classes, associations and constraints to describe architecture of the identified system.

Class Diagram Components

The class diagram is made up of three main areas:

- Top Part— The upper part lists the class name, which represents a set of objects that may have shared attributes, behaviors and roles. Specifics on how to represent groups of objects are capitalizing the first letter in class name, centering it, writing in bold typeface and slanted titles for abstract classes.
- Middle Part: here declare the class attributes with visibility indicators static (+), private (-), protected (#) or package (~)
- Beside it is the bottom section describes with a sequence how class operate or methods in list as each method on a different line. It shows how the class uses data.

Relationships in class diagram from UML are of three types:

- Change One will Affect the other, representing Dependency
- Generalization: This is a hierarchical relationship between one class (parent) and another (child).
- Association: Representation of relationship between things.
- Required Quantity: Defining the restriction of how many instances should contain an particular attribute, multiplicity has one by Default if it is not given.

- Aggregation: An aggregation is the association of a collection, that is involved in relationship called group.
- Composition: "Composition" is just a subset of "aggregation.". This will tell in the way that it need each other (Parent and child), means if removed one another won´t work.

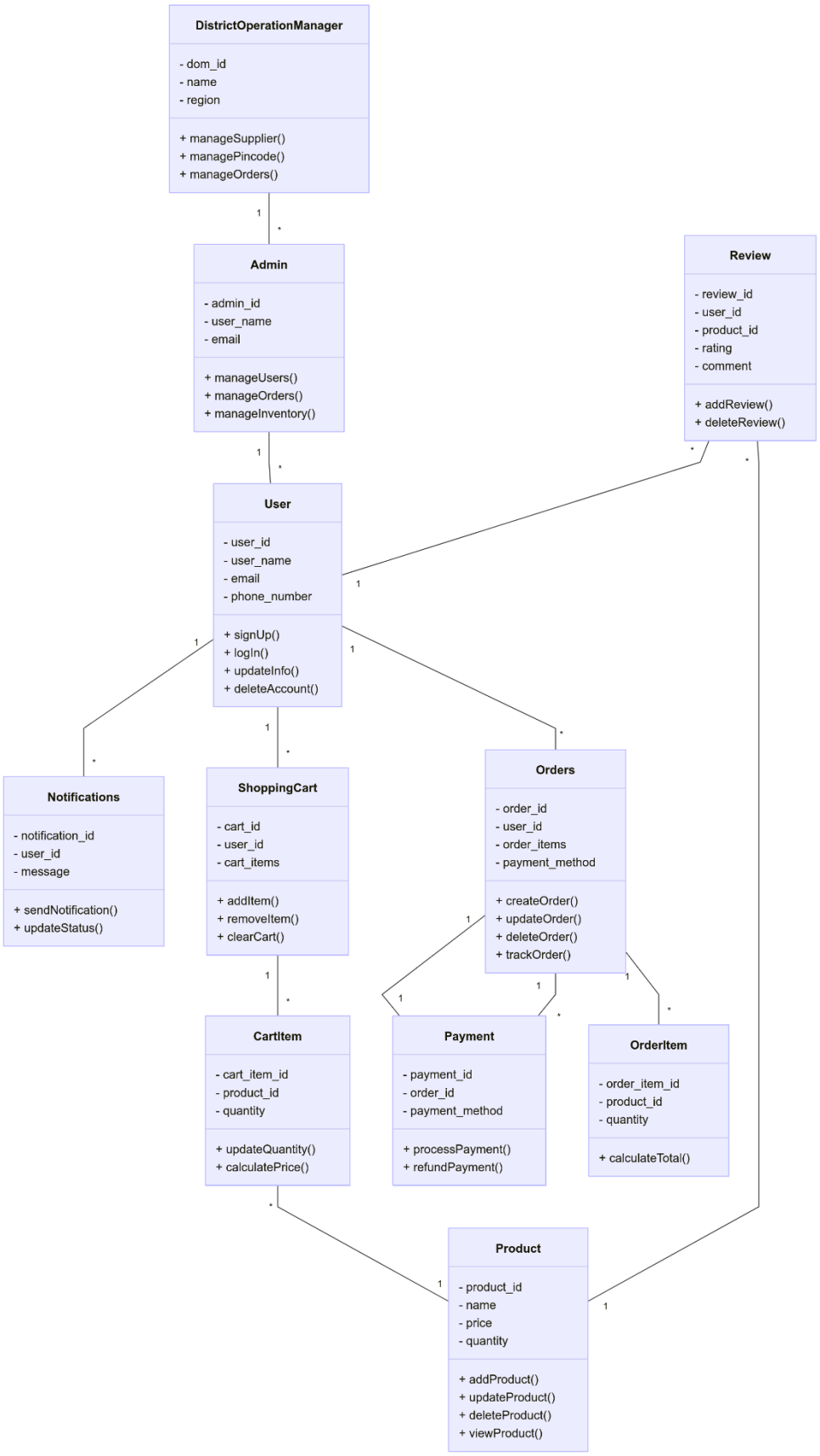


Fig 4.2.5 Class Diagram

4.2.6 Object Diagram

The creation of Object diagrams comes from applying Class diagrams to develop a visualization of entities. The diagrams display objects as points while representing their class type by symbols. Object diagrams present the situation of objects existing in object-oriented systems at specific temporary points.

Although object diagrams stem from class diagrams both models have distinctive features that make them different from each other. The abstraction in class diagrams makes them generic while they avoid showing instance objects. Class diagrams employ abstraction methods which lower the complexity for learning system function and organizational structure.

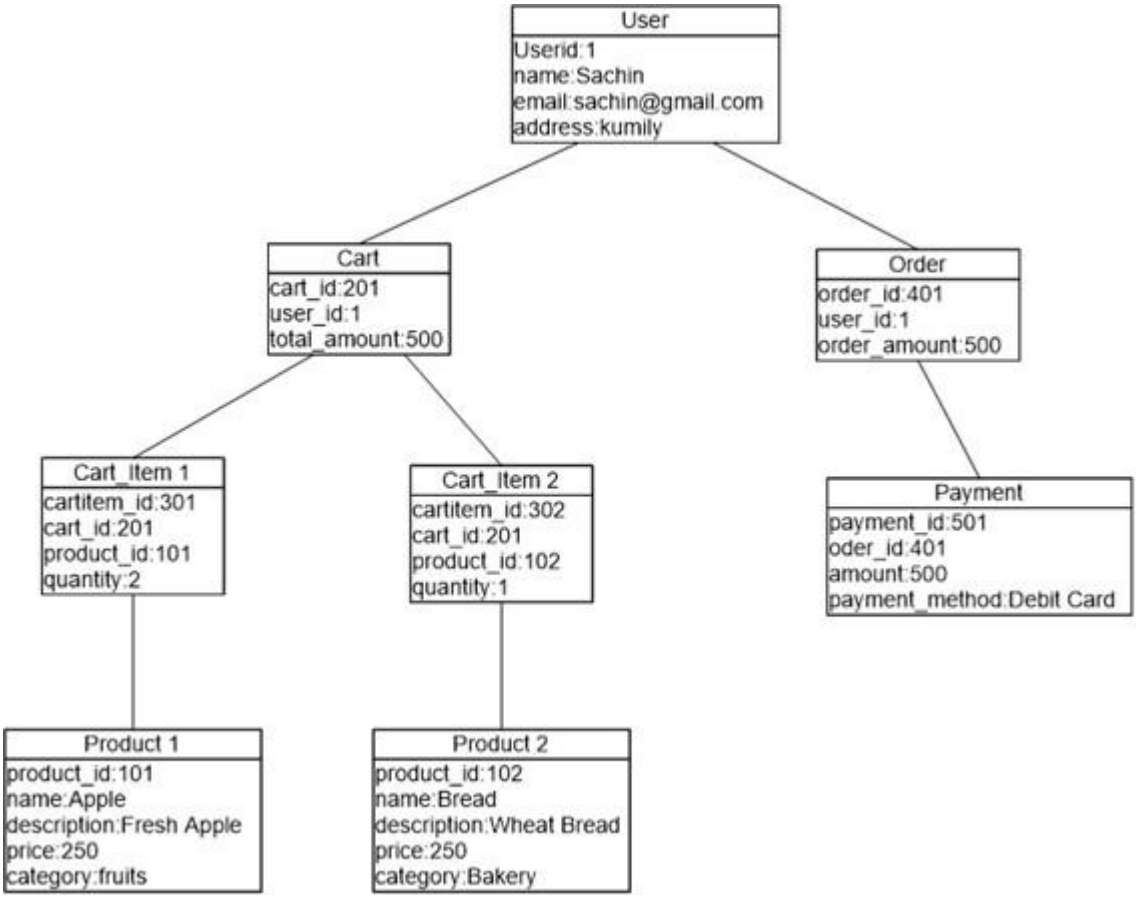


Fig 4.2.6 Object Diagram

4.2.7 Component Diagram

The component diagram is used to break up a large object-oriented system into less complex components. It shows a graphical view for components, programs and documents within log nodes that forms the system.

A component diagram also reveals how things in the system relates to each other and combine into

an operational system.

In the perspective of component diagram, component is nothing but a part of system that can be changed that works by itself. Being a Blackbox process also remains its secret internals and needs the exact operation to be performed, it does execute only when used rightly.

Component Diagram Notation:

- A component
- A node

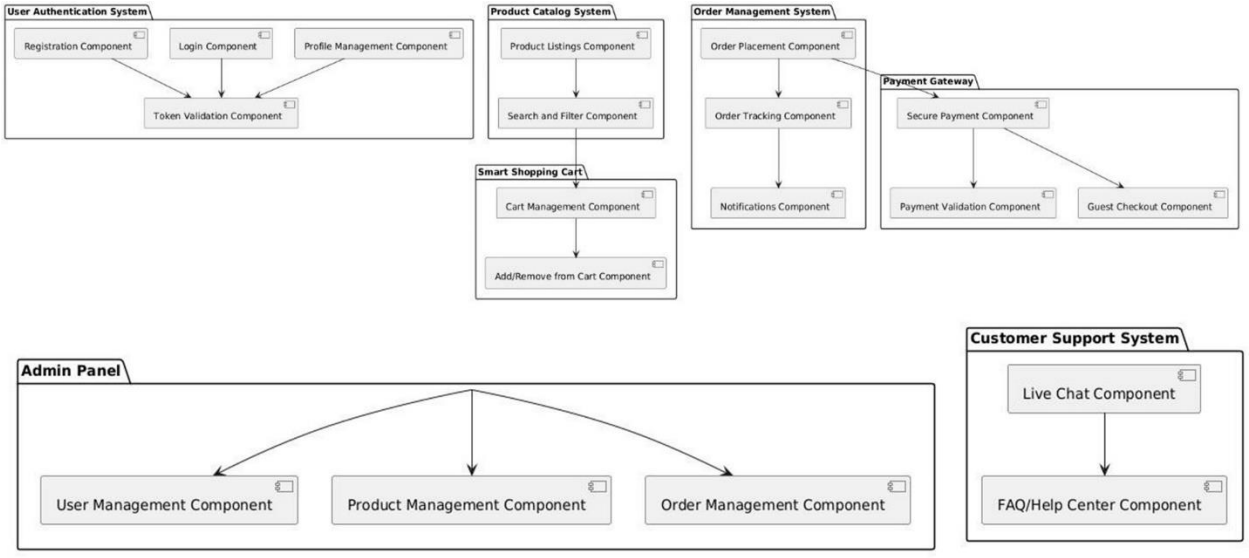


Fig 4.2.7 Component Diagram

4.2.8 Deployment Diagram

The deployment diagram gives a graphic illustration of software placement onto actual physical computer systems. This diagram presents the system at a fixed moment to display how nodes interact through their network connections.

The diagram delves into program placement on computers to explain the construction method software requires for correspondence with physical computer systems.

Notations in a Deployment Diagram include:

- A component
- An artifact
- An interface

- A node

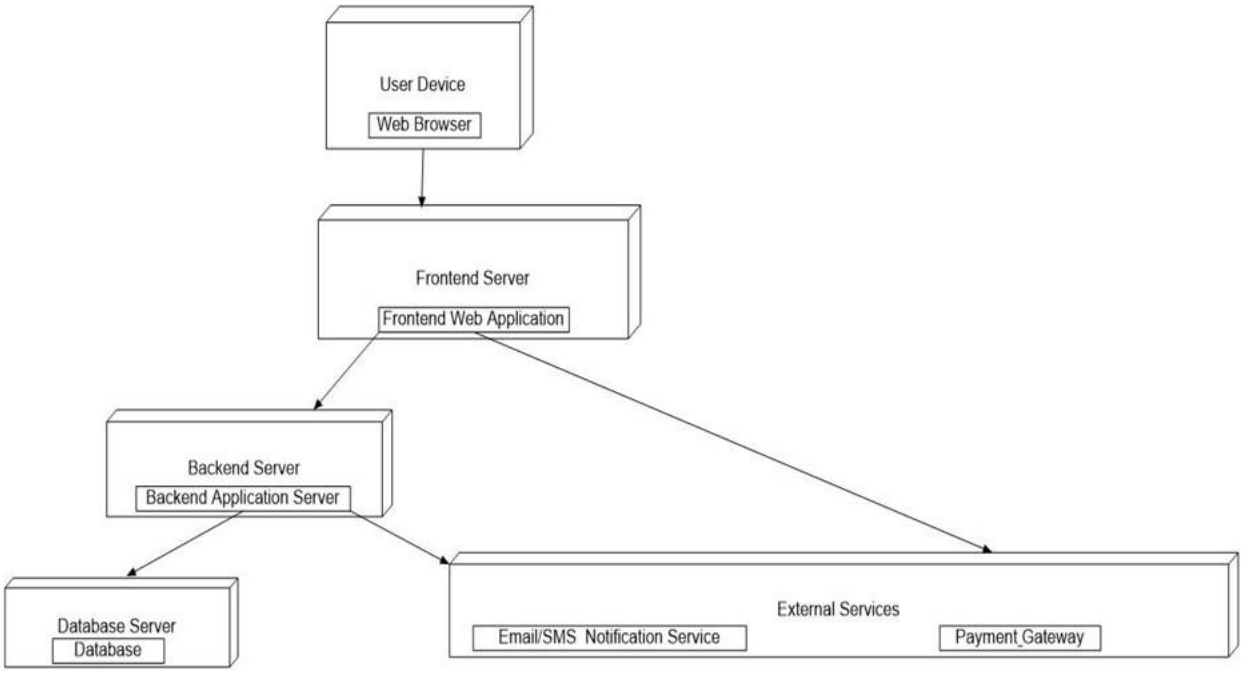


Fig 4.2.8 Deployment Diagram

4.3 USER INTERFACE DESIGN USING FIGMA

4.3.1. Login Page

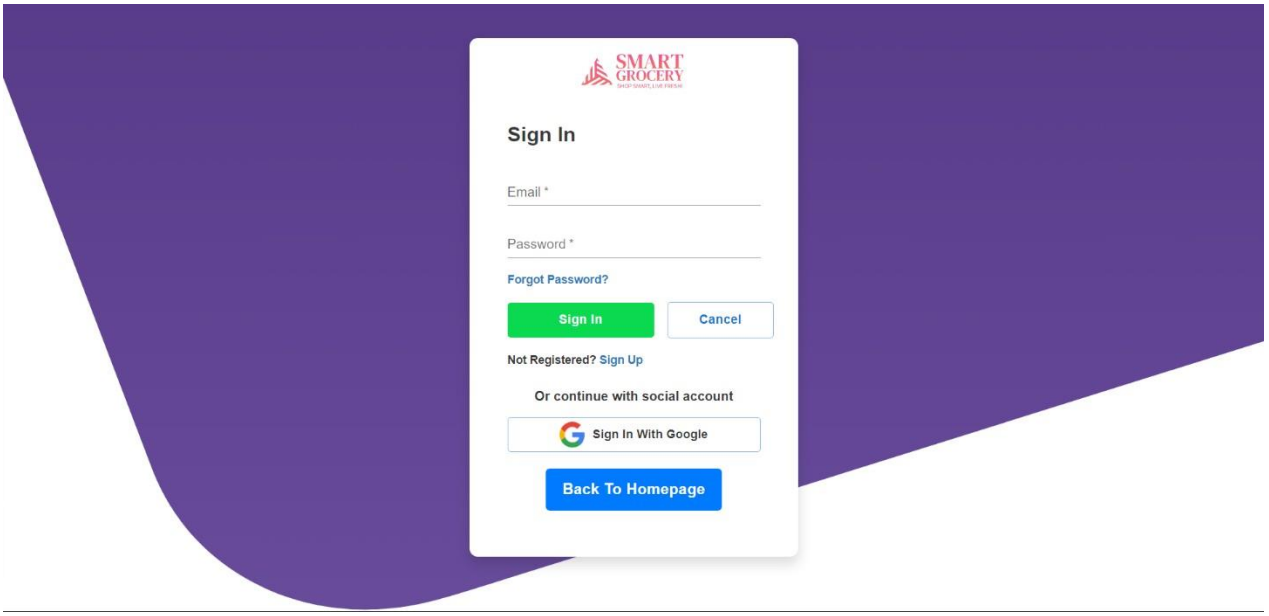


Fig 4.3.1. Login Page

4.3.2. Registration Page

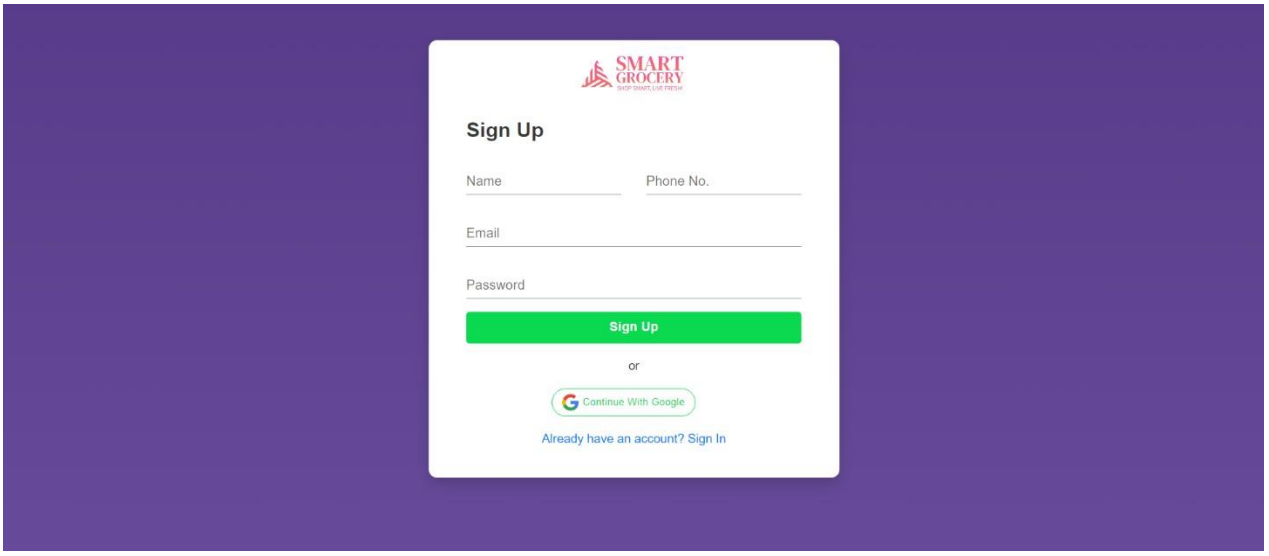


Fig 4.3.2. Registration Page

4.3.3. Homepage

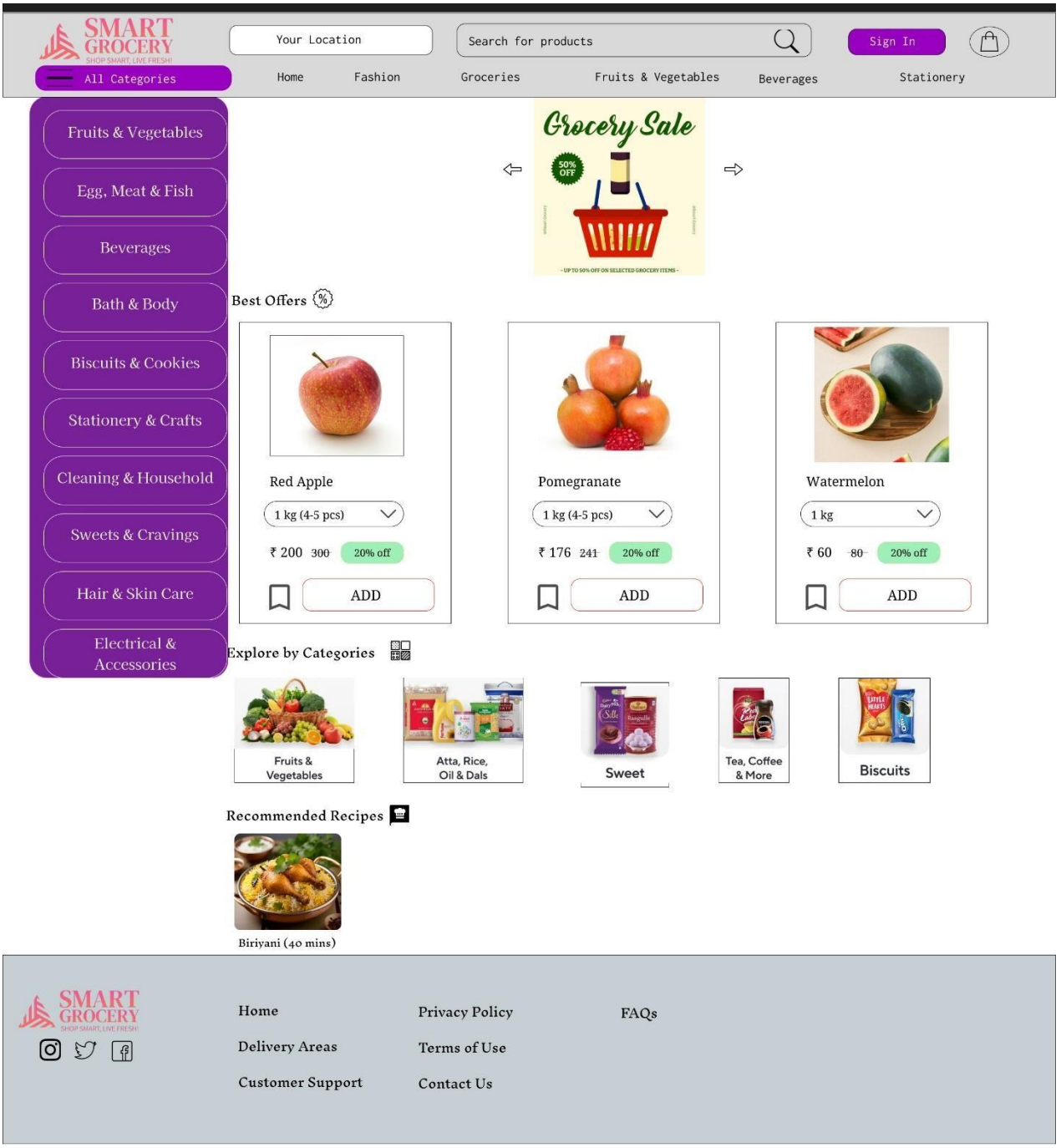


Fig 4.3.3. Homepage

4.3.4. Product View Page

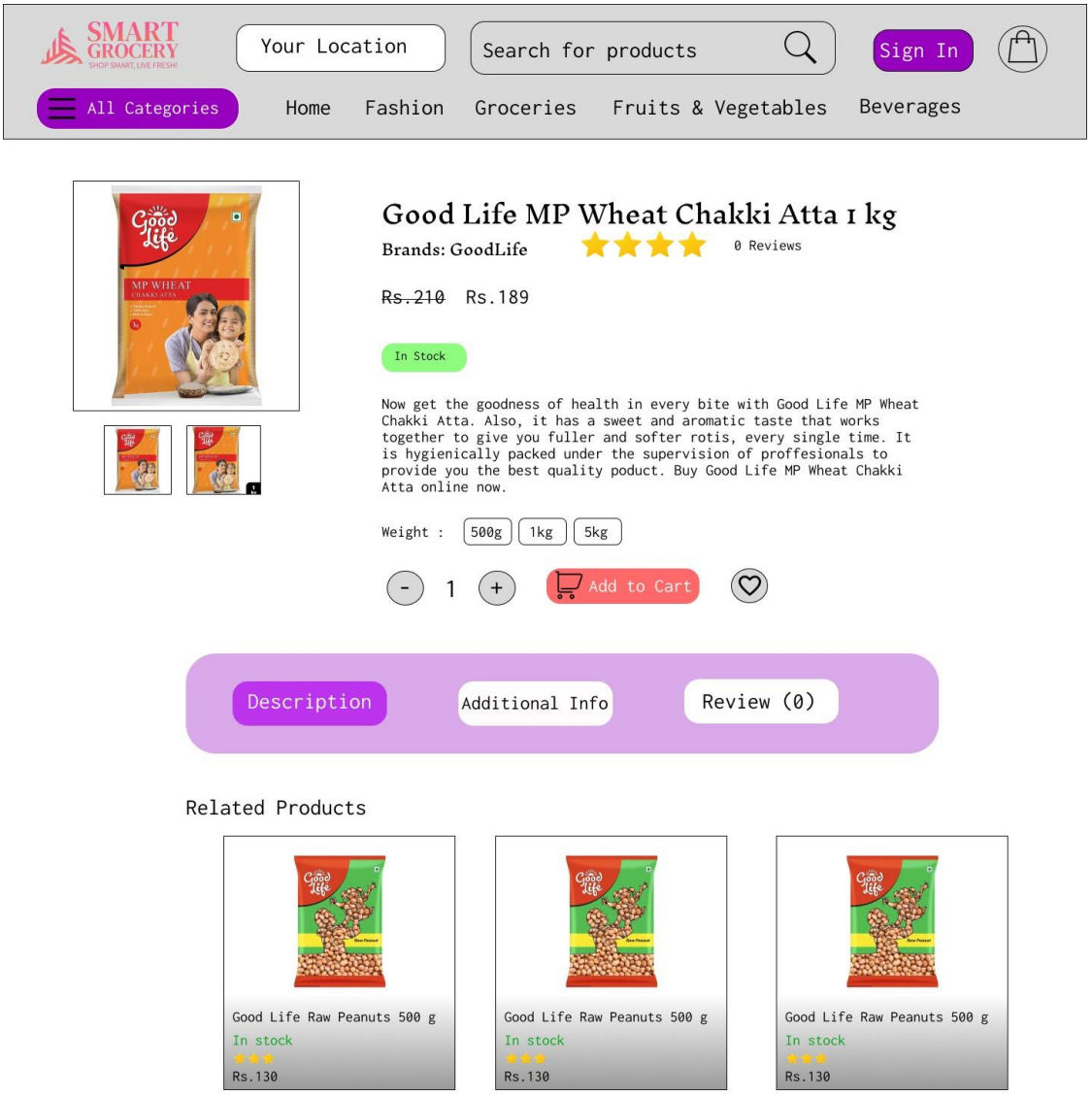


Fig 4.3.4. Product View Page

4.3.5. Cart Page

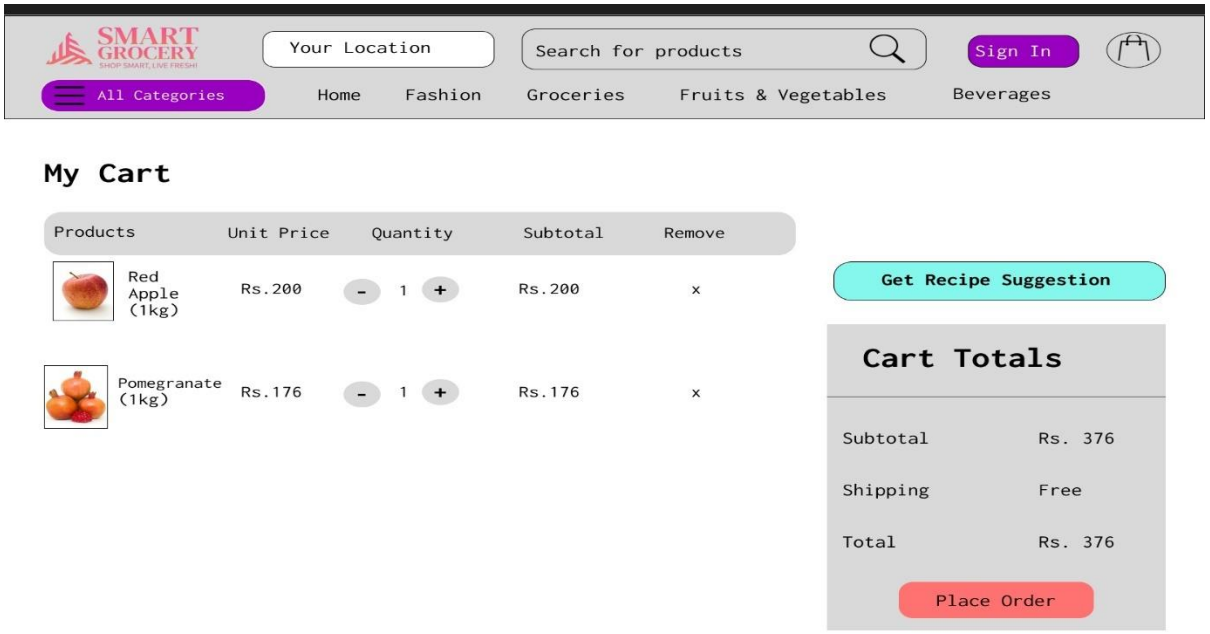


Fig 4.3.5. Cart Page

4.3.6. Buy Now Page

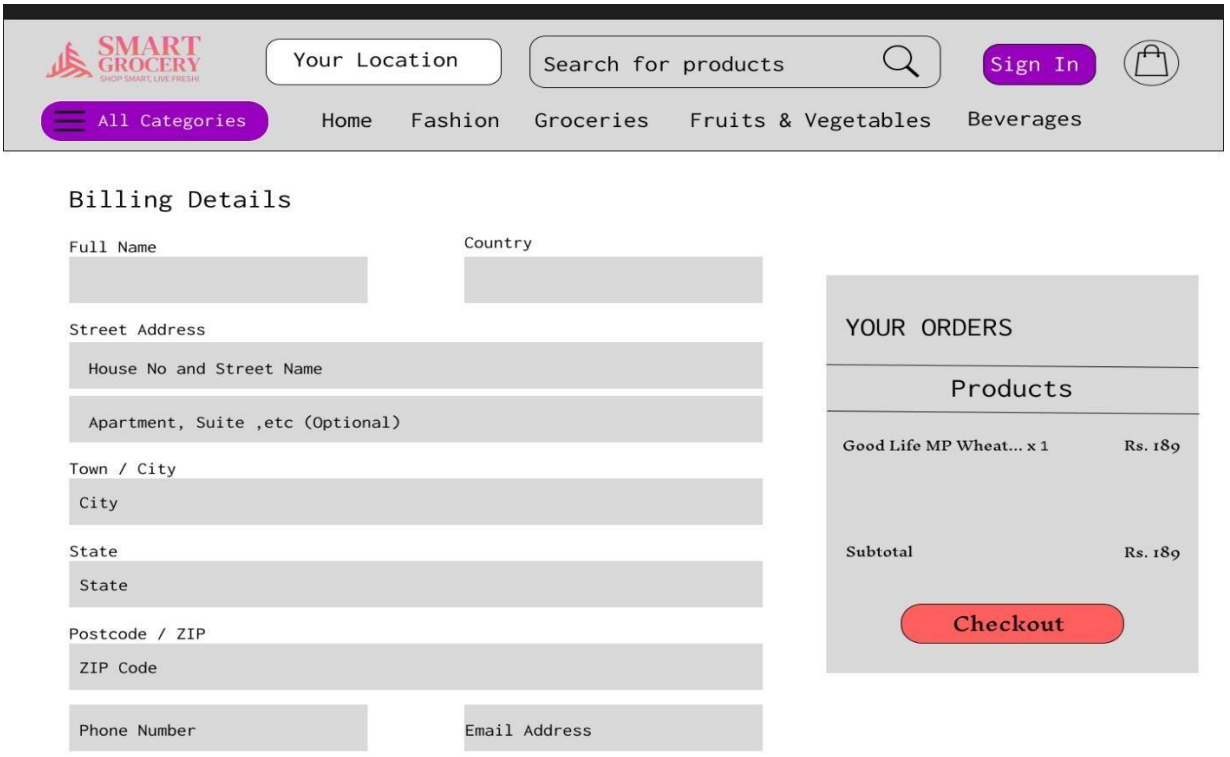


Fig 4.3.6. Buy Now Page

4.3 DATABASE DESIGN

The structured information system which stores data functions as a secure database that enhances the manageability of data updates through efficient design methodologies. Information-level design begins first in the process since it collects user needs for building a database which perfectly matches user requirements. The specific DBMS design follows this phase which operates independently of database management systems. A physical-level design follows by taking into account the characteristics of the selected database management system (DBMS).

4.4.1 MONGODB DATABASE MANAGEMENT SYSTEM (MDBMS)

The data management system MongoDB plays a vital position for handling structured data within its role as a NoSQL database management system. Data stored in MongoDB follows a different approach than RDBMS since it uses flexible JSON like documents in its collections. The database consists of documents that contain structures organized through attributes which define their properties. Data integrity is maintained through MongoDB and the system enables authors to use dynamic schemas to achieve scalability. The ability of MongoDB to process intricate data structures leads developers to select it for their applications since it provides adaptable infrastructure for handling multiple data formats.

4.4.2 Normalization

The relational database management system (RDBMS) technique Normalization maintains significance in MongoDB although it belongs to the NoSQL database category. The principles of normalization operate in MongoDB even though it functions as a NoSQL database. The data organization in MongoDB consists of distributed collections that connect through reference points. The application of this method to structural design reduces duplicated entries while keeping data consistent by conforming to MongoDB's structural design independence approach. The MongoDB design team implements normalization techniques properly to achieve both data consistency and sped-up database operations and structure efficiency.

4.4.3 Sanitization

The Smart Grocery platform depends on sanitization procedures because they serve to protect both data security and integrity while preventing SQL injection and cross-site scripting (XSS) attacks. The Smart Grocery platform applies frontend React.js and backend Node.js components that execute sanitization methods against malicious inputs and to maintain data consistency.

React.js uses controlled components to validate data in real-time so that server input gets filtered before processing. The use of Yup and Formik libraries helps users validate their input data at the client-side which maintains correct data patterns to enhance security as well as improve user

experience.

The backend solution uses express-validator as a Node.js module to perform API request sanitization for user registration processes together with payments. The DOMPurify library defends against XSS attacks through its function of cleaning HTML contents. The schema-based data model of MongoDB blocks the storage of potentially dangerous information in the system.

The combined use of various security layers provides both safe and clean data processing for customers of the Smart Grocery e-commerce solution.

4.4.4 Indexing

An index tracks information and its grouped counterparts when arranged in value order. Proper arrangement of index entries allows users to locate information both exactly or within specified ranges with maximum efficiency. Users can locate data in a database instantly with indexes without the need to search all records during each access. The index provides a guide which helps users locate information within a database system. The indexing system allows fast data retrieval and provides efficient record locating by specific ordering. The index can operate from one or several columns contained within 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)
15	isSupplier	Boolean	Indicates if the user is a Supplier

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

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	SubCat	String	Name of the sub category
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
21	basePrice	Number	Indicates the base price of the product

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	reason	String	Indicates reason for cancellation

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	customerName	String	Name of the Customer
3	productId	ObjectId	References tbl_Products (_id)
4	customerId	ObjectId	References tbl_Users (_id)
5	rating	Number	Rating given by the customer
6	review	String	Review content

4.5.15 Tbl_DynamicPricing

Primary Key : **_id**

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

No.	Field Name	Data Type	Description
1	_id	ObjectId	Unique identifier for the dynamic pricing
2	productId	ObjectId	References tbl_Products (_id)
3	originalPrice	Number	Indicates actual price of the product
4	currentPrice	Number	Indicates current price of the product
5	stockLevel	Number	Indicates the stock level of the product
6	demandScore	Number	Indicates the demand score of the product
7	salesVelocity	Number	Sales velocity of the product
8	priceHistory	Array	List of price updations made
9	lastUpdated	Date	Indicates the last updated date

4.5.16 Tbl_FaceData

Primary Key: **_id**

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

No.	Field Name	Data Type	Description
1	_id	ObjectId	Unique identifier for the face data
2	userId	ObjectId	References tbl_Users (_id)
3	faceDescriptor	Array	Coordinates of the face
4	createdAt	Date	Indicates the date the face is registered

4.5.17 Tbl_Report

Primary Key: **_id**

No.	Field Name	Data Type	Description
1	_id	ObjectId	Unique identifier for the report
2	type	String	Type of User
3	period	String	Period of report
4	dateRange	Date	Start and End date of report
5	location	String	Location of the District Operation Manager
6	createdAt	Date	Date of creation of report

4.5.17 Tbl_stockManagement

Primary Key: **_id**

Foreign Key: **productId** reference **Tbl_Products(_id)**

No.	Field Name	Data Type	Description
1	_id	ObjectId	Unique identifier for the stock management
2	productId	ObjectId	References Tbl_Products(_id)
3	type	String	Indicates Demand type
4	threshold	Number	Indicates threshold
5	currentStock	Number	Indicates current stock level
6	auutoOrderEnabled	Boolean	Indicates whether auto-order is enabled or Not
7	location	String	Indicates location
8	status	String	Indicates status of the stock
9	createdAt	Date	Date of creation of report

4.5.17 Tbl_stockOrder

Primary Key: **_id**

Foreign Key: **productId** reference **Tbl_Products(_id)**

Foreign Key: **supplierId** reference **Tbl_Users(_id)**

No.	Field Name	Data Type	Description
1	_id	ObjectId	Unique identifier for the stock order
2	productId	ObjectId	References Tbl_Products(_id)
3	supplierId	ObjectId	References Tbl_Users(_id)
4	quantity	Number	Indicates the quantity of stock
5	status	String	Indicates the status of the stock
6	autoOrdered	Boolean	Indicates the stock is autoOrdered or Not

7	location	String	Indicates the location
8	orderDate	Date	Indicates the order date of the stock
9	paymentStatus	String	Indicates the status of payment
10	paymentMethod	String	Indicates the payment method
11	paymentDate	Date	Date of payment done
12	invoiceNumber	String	Indicates the invoice number of stock
13	totalAmount	Number	Indicates the total amount of stock ordered

4.5.17 Tbl_supplierProducts

Primary Key: **_id**

Foreign Key: **supplierId** reference **Tbl_Users(_id)**

No.	Field Name	Data Type	Description
1	_id	ObjectId	Unique identifier for the review
2	supplierId	ObjectId	References Tbl_Users(_id)
3	name	String	Name of the product
4	description	String	Description of the product
5	price	Number	Price of the product
6	quantity	Number	Quantity of product
7	quantityType	String	Type of Quantity
8	category	String	Category of Product
9	minStockAlert	Number	Minimum Stock of product
10	productCode	String	Indicates a unique code for the product
11	status	String	Indicates status of the product
13	createdAt	Date	Date of creation of product

CHAPTER 5
SYSTEM TESTING

5.1 INTRODUCTION

The verification of computer programs against intended functionalities happens through essential testing procedures. The testing procedure aims to verify software execution alongside required standards and prescribed requirements. Software verification happens through official checks that confirm programs meet the criteria defined in specifications. Program assessment as a testing method conducts performance validation through combination techniques which include code inspection and program walkthroughs. The software validation process verifies that systems match what users expect in their requirements. Multiple guiding principles together with specific test objectives direct the entire software testing process:

1. The execution of programs with error identification as the main purpose constitutes testing.
2. Test cases that demonstrate high potential to detect new faults within the software system are considered effective.
3. Software testing generates success through the detection of new errors.

A successful test case operates to meet its objectives which enables it to discover software faults. The successful operation of the computer program indicates its proper functioning and excellent performance. Such assessment includes three main components:

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

5.2 TEST PLAN

Test plans consist of detailed instructions which direct users throughout different forms of testing operations. The document functions like a directional map which details the testing approach to evaluate a computer program. The programming team designs complete usage guidelines that manage program data organization for correct operation. The team makes sure that every program subsection executes its designed operations. The testing of the software undergoes complete verification by an organization called the ITG (Information Technology Group) to confirm its operational capabilities rather than depending on the developers for testing alone.

Testing objectives need to exist as clear and measurable statements. A structured test plan needs to include information regarding failure frequency and repair expenses as well as the frequency of encountered issues and complete testing duration. Testing levels follow a specific order which contains:

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

5.2.1 Unit Testing

A software design's smallest element can be checked by unit testing since it begins with testing software components or modules. A test of crucial control paths in the module follows the design guide to discover potential issues. Each tiny portion inside the program follows specific difficulty in testing criteria as well as indicating untested areas. Unit testing serves as an examination method which studies program internals while allowing parallel evaluation across different program sections.

Testing must commence by validating data transmission between all elements of the computer program. All other verification checks become useless when information transfer fails to work properly. Before creating something, designers should envision potential issues ahead of time and put together strategies for their resolution. The system requires either directing the process differently or terminating the process altogether.

The developers evaluated the Smart Grocery System through individual part examinations followed by various testing procedures. During module design various mistakes were detected for which programmers implemented corrective steps. The checking process begins after writing instructions for each individual part before moving on to separate tests. Our team deleted surplus codes while validating that all operations function correctly.

5.2.2 Integration Testing

Software development requires integration testing as a crucial process to develop programs which identifies errors between different program components while in progress. The main purpose involves using already tested components to build programs which follow the original design. The examination screens every component of the program to validate its correct operation.

Integration testing identifies problems which cause the development of new issues that force repeated cycles between testing and refinement. After complete examination of each individual system component its components are integrated to verify their unified operation. All programs undergo standardization work to prevent the occurrence of multiple different versions across the system.

5.2.3 Validation Testing or System Testing

The system tests all components through a thorough examination to confirm that assorted building blocks and instruction types connect properly with each other. System testing also known as Black Box testing constitutes the last testing approach.

The purpose of Black Box testing helps confirm if software performs according to specifications. Its mechanism aids software engineers in detecting all program issues through multiple types of input. The evaluation scope of Black Box testing includes functional, interface, data access, performance testing in addition to process initialization and termination. Software verification through this essential approach confirms that developed applications fulfill their specifications with accurate operation.

5.2.4 Output Testing or User Acceptance Testing

A user satisfaction test along with company requirement verification occurs during system testing. End-user contact remains essential for computer program development or update requirements. These elements enable the connection according to the following points:

- Input Screen Designs.
- Output Screen Designs.

Testing the system requires utilizing different kinds of data. The development of test data stands as an essential element during this phase of work. The collected test data functions as a tool for evaluating the system after its collection. The evaluation process creates identified issues which developers solve by following standard operating procedures. The corrections are documented for future use in order to enhance the system. The process maintains operational effectiveness and user and organizational requirements satisfaction of the system.

5.2.5 Automation Testing

Before formal implementation software verification occurs through Automated testing which ensures proper software functionality together with standard compliance. Testing tools execute

written instructions to carry out this type of testing. TI automation testing utilizes particular tools for performing automated testing operations. The creation of scripts serves to automate necessary checks instead of having humans perform manual clicking within the application for functional verification. Multiple computers benefit from automated testing processes since the same test can run across them simultaneously which leads to faster and more uniform testing outcomes.

5.2.6 Selenium Testing

The free resource Selenium serves as an important tool to conduct automated testing of websites. The tool functions as an essential element in web developer work environments through its ability to simplify testing operations. The method of using Selenium to conduct automation testing is called Selenium automation testing. The Selenium automation toolkit consists of various testing units which execute separate tasks to benefit testers during automation testing procedures. The fundamental requirement of development applications relies on manual testing even though this approach leads to tedious and repeated work. Employee Jason Huggins at Thoughtworks created a system for automatic testing procedures to replace manual work. Jason Huggins started by developing the JavaScriptTestRunner which later received a new name as Selenium in 2004.

Test Case 1: User Login

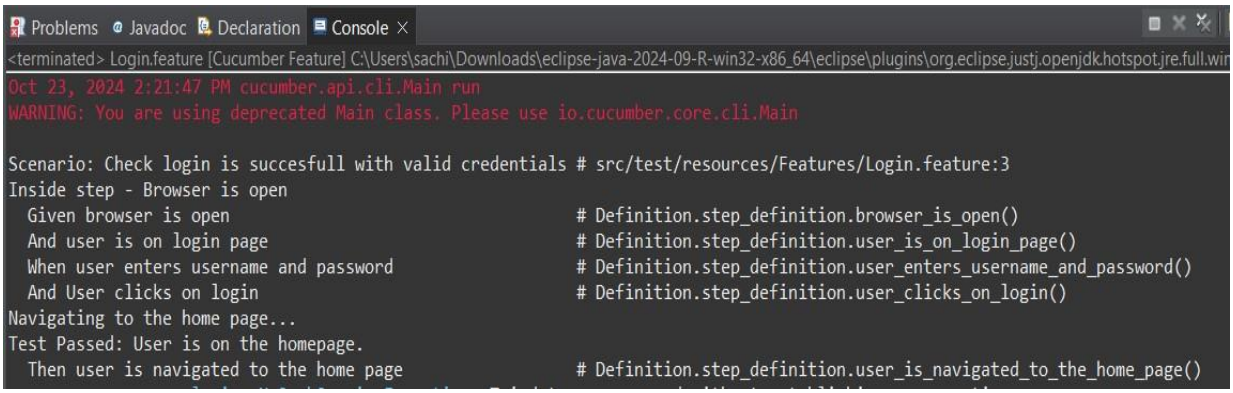
Code:

```
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\\eclipseworkspace
\\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: 16-03-2025**
Module Name: Login Module	**Test Executed By: Ms. Sona Maria Sebastian**
Test Title: User Login	**Test Execution Date: 17-03-2025**
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@123**			
4	**Click on login button**				
Post-Condition: User is validated with database and successfully logs into Homepage.					

Test Case 2: Face Authentication

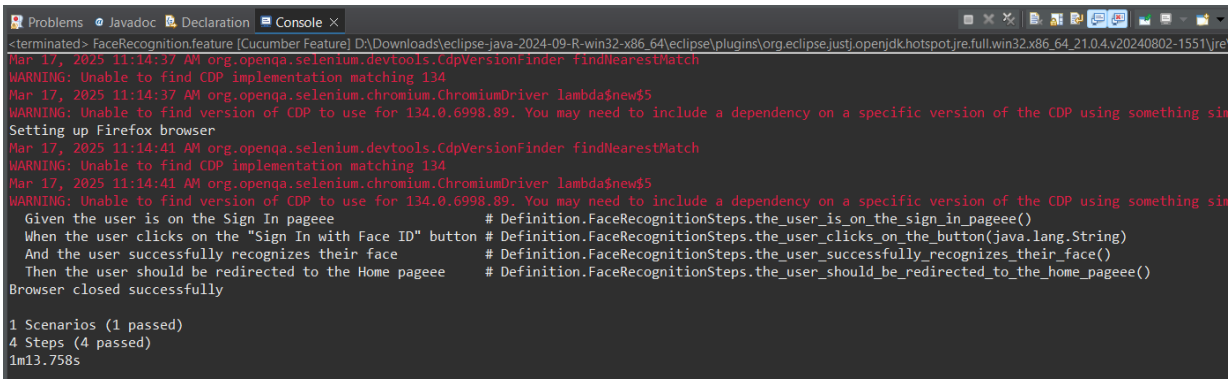
Code:

```
package Definition;

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 io.cucumber.java.After;
import io.cucumber.java.Before;
import io.cucumber.java.en.Then;
import io.cucumber.java.en.When;
```

```
import java.time.Duration;

public class FaceRecognitionSteps {
    private WebDriver driver;
    private WebDriverWait wait;

    @Before
    public void setup() {
        System.setProperty("webdriver.chrome.driver", "C:\\Users\\sachi\\eclipse-
workspace\\1234\\src\\test\\resources\\drivers\\chromedriver.exe");

        // Launch Chrome
        driver = new ChromeDriver();

        // Initialize explicit wait
        wait = new WebDriverWait(driver, Duration.ofSeconds(10));

        // Maximize browser window
        driver.manage().window().maximize();
    }

    @When("the user is on the Sign In pageee")
    public void the_user_is_on_the_sign_in_pageee() {
        driver.get("http://localhost:3008/signin"); // Adjust the URL as necessary
    }

    @When("the user clicks on the {string} button")
    public void the_user_clicks_on_the_button(String buttonText) {
        WebElement faceIdButton =
wait.until(ExpectedConditions.visibilityOfElementLocated(By.id("faceIdLogin")));
        faceIdButton.click();
        WebElement faceIdButton1 =
wait.until(ExpectedConditions.visibilityOfElementLocated(By.id("startFaceIdLogin")));
        faceIdButton1.click();
```

```
    faceIdButton1.click();
    try {
        Thread.sleep(60000); // 5000 milliseconds = 5 seconds
    } catch (InterruptedException e) {
        e.printStackTrace();
    }
}

@When("the user successfully recognizes their face")
public void the_user_successfully_recognizes_their_face() {
    // Simulate the face recognition process
    // This may involve waiting for the face detection to complete
    // You may need to adjust this based on your application's behavior
   String currentUrl = driver.getCurrentUrl();
    assert currentUrl.equals("http://localhost:3008/");    }

@Then("the user should be redirected to the Home pageee")
public void the_user_should_be_redirected_to_the_home_pageee() {
    String currentUrl = driver.getCurrentUrl();
    assert currentUrl.equals("http://localhost:3008/"); // Adjust the URL as necessary
}

@After
public void tearDown() {
    if (driver != null) {
        driver.quit();
    }
}
```

Screenshot



Test Report

Test Case 2

Project Name: Smart Grocery

Face Authentication Test Case

Test Case ID: 2	**Test Designed By: Sachin Sam Jacob**
Test Priority (Low/Medium/High): High	**Test Designed Date: 16-03-2025**
Module Name: Face Authentication Module	**Test Executed By: Ms. Sona Maria Sebastian**
Test Title: Face Authentication Login	**Test Execution Date: 17-03-2025**
Description: Checks face and Logins to homepage	

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	Clicks on Sign in with Face ID		Detects the face of the user	User logs in	Pass

Post-Condition: User is validated with the Face Coordinates stored in the database and successfully logs into Homepage.

Test Case 3: Recipe Generation

Code:

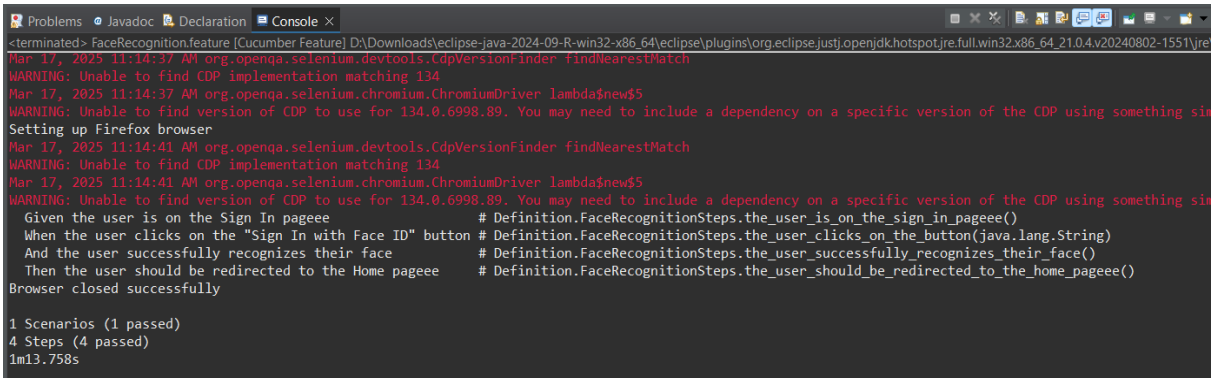
```
package Definition;

import java.time.Duration;
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.support.ui.ExpectedConditions;
import org.openqa.selenium.support.ui.WebDriverWait;
import io.cucumber.java.After;
import io.cucumber.java.Before;
import io.cucumber.java.en.Given;
import io.cucumber.java.en.Then;
import io.cucumber.java.en.When;

public class RecipeGenerationSteps {
    private WebDriver driver;
    private WebDriverWait wait;

    // Setup method to initialize WebDriver with Chrome
    @Before
    public void launchBrowser() {
        // Set path to ChromeDriver
        System.setProperty("webdriver.chrome.driver", "C:\\Users\\sachi\\eclipse-
workspace\\1234\\src\\test\\resources\\drivers\\chromedriver.exe");

        // Launch Chrome
        driver = new ChromeDriver();

        // Initialize explicit wait
```

```
        wait = new WebDriverWait(driver, Duration.ofSeconds(10));

        // Maximize browser window
        driver.manage().window().maximize();
    }
    @Given("the shopper is on the Sign In page")
    public void the_shopper_is_on_the_sign_in_page() {
        // Navigate to the Sign In page
        driver.get("http://localhost:3008/signin");    }

    @When("shopper provides valid login details")
    public void shopper_provides_valid_login_details() {
        WebElement emailBox =
wait.until(ExpectedConditions.visibilityOfElementLocated(By.name("email")));
        WebElement passwordBox =
wait.until(ExpectedConditions.visibilityOfElementLocated(By.name("password")));

        emailBox.sendKeys("sachinsamjacob@gmail.com"); // Replace with valid email
        passwordBox.sendKeys("Password@123"); // Replace with valid password



    }

    @When("shopper selects the Sign In button")
    public void shopper_selects_the_sign_in_button() {
        WebElement signInBtn = driver.findElement(By.id("login"));
        signInBtn.click();
    }

    @Then("shopper is navigated to the dashboard")
    public void shopper_is_navigated_to_the_dashboard() {
        String currentUrl = driver.getCurrentUrl();
        assert currentUrl.equals("http://localhost:3008/");
        try {
            Thread.sleep(6000); // 5000 milliseconds = 5 seconds
```

```
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }

    @When("shopper clicks on the basket icon in the navbar")
    public void shopper_clicks_on_the_basket_icon_in_the_navbar() {
       wait.until(ExpectedConditions.visibilityOfElementLocated(By.className("header")));
        WebElement cartButton = driver.findElement(By.id("cart"));

        cartButton.click();
        try {
            Thread.sleep(4000); // 5000 milliseconds = 5 seconds
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }

    @Then("shopper should view the cart section")
    public void shopper_should_view_the_cart_section() {
       String currentUrl = driver.getCurrentUrl();
        assert currentUrl.equals("http://localhost:3008/cart");
    }

    @When("shopper accesses the recipe suggestion feature")
    public void shopper_accesses_the_recipe_suggestion_feature() {
        WebElement recipeButton =
wait.until(ExpectedConditions.visibilityOfElementLocated(By.xpath("//button[contains(text
(), 'Get Recipe Suggestions')]")));
        recipeButton.click();
        try {
            Thread.sleep(20000); // 5000 milliseconds = 5 seconds
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
```

```
    }

    @Then("shopper should be directed to the recipe suggestion page")
    public void shopper_should_be_directed_to_the_recipe_suggestion_page() {
        closeBrowser();
    }

    // Cleanup method to close the browser
    @After
    public void closeBrowser() {
        if (driver != null) {
            driver.quit();
        }
    }
}
```

Screenshot



Test Report

Test Case 3					
Project Name: Smart Grocery					
Recipe Generation Test Case					
Test Case ID: 3			**Test Designed By: Sachin Sam Jacob**		
Test Priority (Low/Medium/High): High			**Test Designed Date: 16-03-2025**		
Module Name: Recipe Generation Module			**Test Executed By: Ms. Sona Maria Sebastian**		
Test Title: Recipe Generation			**Test Execution Date: 17-03-2025**		
Description: Generates recipes based on the products in the cart					
Pre-Condition: User has should have at least two products in their 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@123			
4	Click on login button				
5	Click on cart		Cart should be viewed	Cart is Viewed	Pass
6	Clicks on Get Recipe Suggestion		Recipes should be Generated	Recipes are generated	Pass

Post-Condition: : User is validated with database and successfully logs into Homepage , clicks on cart option, clicks on get recipe suggestion and the recipes are generated .

Test Case 4: Visual Search

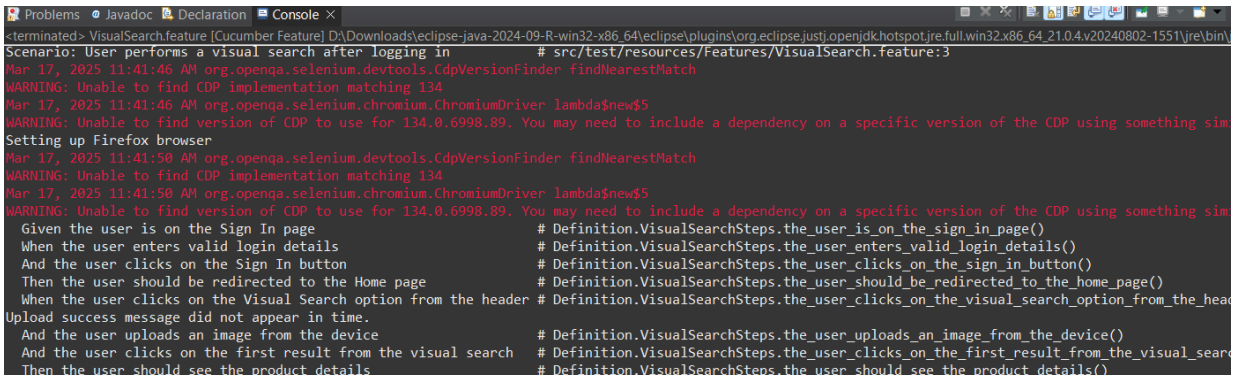
Code:

```
package Definition;

import java.time.Duration;

import org.openqa.selenium.By;
import org.openqa.selenium.JavascriptExecutor;
import org.openqa.selenium.TimeoutException;
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 io.cucumber.java.en.*;

public class VisualSearchSteps {
   private WebDriver driver;
   private WebDriverWait wait;

   public VisualSearchSteps() {
         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));
   }

   @Given("the user is on the Sign In page")
   public void the_user_is_on_the_sign_in_page() {
      driver.get("http://localhost:3008/signin"); // Adjust URL as necessary
   }

   @When("the user enters valid login details")
   public void the_user_enters_valid_login_details() {
      // Wait for the email field to be visible
      WebElement emailField =
wait.until(ExpectedConditions.visibilityOfElementLocated(By.name("email")));
      WebElement passwordField =
wait.until(ExpectedConditions.visibilityOfElementLocated(By.name("password")));

      emailField.sendKeys("sachinsamjacob@gmail.com"); // Replace with valid email
```

```
        passwordField.sendKeys("Password@123"); // Replace with valid password
    }

    @When("the user clicks on the Sign In button")
    public void the_user_clicks_on_the_sign_in_button() {
        WebElement signInButton =
wait.until(ExpectedConditions.elementToBeClickable(By.xpath("//button[contains(text(),'Sign
In')]")));
        signInButton.click();
    }

    @Then("the user should be redirected to the Home page")
    public void the_user_should_be_redirected_to_the_home_page() {
        String currentUrl = driver.getCurrentUrl();
        assert currentUrl.equals("http://localhost:3008/"); // Adjust URL as necessary
        try {
            Thread.sleep(6000); // 5000 milliseconds = 5 seconds
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }

    @When("the user clicks on the Visual Search option from the header")
    public void the_user_clicks_on_the_visual_search_option_from_the_header() {

        wait.until(ExpectedConditions.visibilityOfElementLocated(By.className("header")));
        WebElement visualSearchIcon =
wait.until(ExpectedConditions.elementToBeClickable(By.xpath("//button[@aria-label='Search
by image']")));
        visualSearchIcon.click();
    }

    @When("the user uploads an image from the device")
    public void the_user_uploads_an_image_from_the_device() {
        WebElement uploadInput = driver.findElement(By.cssSelector("input[type='file']"));
        uploadInput.sendKeys("C:\\Users\\sachi\\Downloads\\Mango-Sindhura.jpg"); // Replace
with the path to the test image

        // Wait for a different condition that indicates the upload is complete
        try {
            // Example: Wait for an element that indicates the upload is complete
            wait.until(ExpectedConditions.visibilityOfElementLocated(By.cssSelector(".uploaded-
image"))); // Adjust selector as necessary
        }
        catch (TimeoutException e) {
            System.out.println("Upload success message did not appear in time.");
            // Optionally, take a screenshot or log the current page state for debugging
        }

        // Trigger a change event on the file input
```

```
        ((JavascriptExecutor) driver).executeScript("arguments[0].dispatchEvent(new
Event('change'));", uploadInput);

        // Optionally, hide the file input after upload
        ((JavascriptExecutor) driver).executeScript("arguments[0].style.display='none';",
uploadInput);
    }
    @When("the user clicks on the first result from the visual search")
    public void the_user_clicks_on_the_first_result_from_the_visual_search() {
        WebElement firstResult = driver.findElement(By.className("product-info"));
        firstResult.click();
    }

    @Then("the user should see the product details")
    public void the_user_should_see_the_product_details() {
        WebElement productDetails = driver.findElement(By.cssSelector(".product-info"));
        assert productDetails.isDisplayed();
    }}
```

Screenshot



Test Report

Test Case 4	
Project Name: Smart Grocery	
Visual Search Test Case	
Test Case ID: 4	**Test Designed By: Sachin Sam Jacob**
Test Priority (Low/Medium/High): High	**Test Designed Date: 16-03-2025**
Module Name: Visual Search Module	**Test Executed By: Ms. Sona Maria Sebastian**
Test Title: Visual Search	**Test Execution Date: 17-03-2025**

Description: Search products based on the image uploaded					
Pre-Condition: User has should have a valid email and password and a photo of a product					
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@123			
4	Click on login button				
5	Click on Visual Search option		Visual Search pop up box should be appeared	Visual search option is appeared	Pass
6	Uploads an image		Image should be uploaded and related product should be shown	Image is uploaded and related products are shown	Pass
Post-Condition: : User is validated with database and successfully logs into Homepage , clicks on visual search and image is uploaded and related products are shown.					

Test Case 4: Product Description Generation

Code:

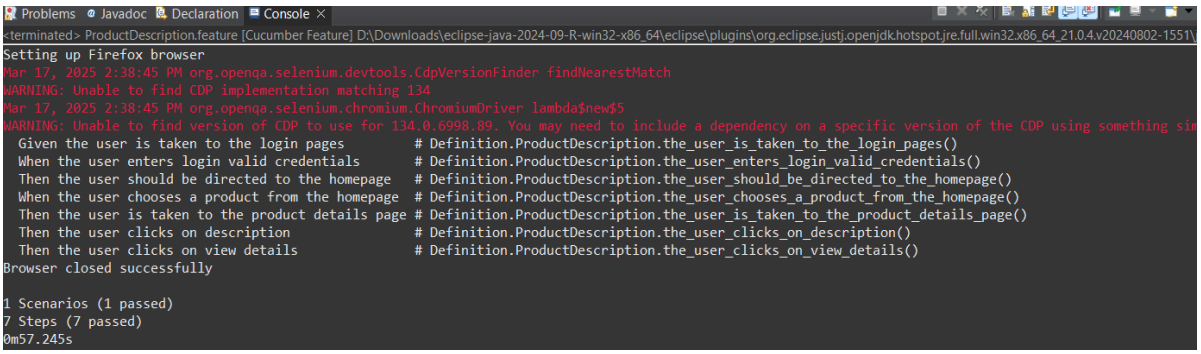
```
package Definition;

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.firefox.FirefoxDriver;
```

```
import org.openqa.selenium.support.ui.ExpectedConditions;
import org.openqa.selenium.support.ui.WebDriverWait;
import java.time.Duration;

public class ProductDescription {
   WebDriver driver;
   WebDriverWait wait;

   @Given("the user is taken to the login pages")
   public void the_user_is_taken_to_the_login_pages() {
      System.setProperty("webdriver.gecko.marionette", "C:\\Users\\sachi\\eclipse-
workspace\\1234\\src\\test\\resources\\drivers");
      driver = new FirefoxDriver();
      wait = new WebDriverWait(driver, Duration.ofSeconds(10));
      driver.get("http://localhost:3008/signIn");
   }

   @When("the user enters login valid credentials")
   public void the_user_enters_login_valid_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");

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

   @Then("the user should be directed to the homepage")
   public void the_user_should_be_directed_to_the_homepage() {
      wait.until(ExpectedConditions.urlContains("http://localhost:3008/"));
      try {
         Thread.sleep(5000);
      } catch (InterruptedException e) {
         e.printStackTrace();
      }
   }

   @When("the user chooses a product from the homepage")
   public void the_user_chooses_a_product_from_the_homepage() {
      WebElement product =
wait.until(ExpectedConditions.visibilityOfElementLocated(By.cssSelector(".productItem")));
      try {
         Thread.sleep(3000);
      } catch (InterruptedException e) {
         e.printStackTrace();
      }
      product.click();
```

```
    }

    @Then("the user is taken to the product details page")
    public void the_user_is_taken_to_the_product_details_page() {
        wait.until(ExpectedConditions.urlContains("http://localhost:3008/product/"));
        WebElement productName =
wait.until(ExpectedConditions.visibilityOfElementLocated(By.cssSelector(".hd.text-
capitalize")));
        assert productName.isDisplayed();
    }

    @Then("the user clicks on description")
    public void the_user_clicks_on_description() {
        // Wait for the tabs to be visible

wait.until(ExpectedConditions.visibilityOfElementLocated(By.className("customTabs")));

        // Use a more precise XPath to find the "Additional info" button
        WebElement additionalInfoTab = wait.until(ExpectedConditions.elementToBeClickable(
            By.xpath("//div[@class='customTabs']//button[contains(text(), 'Additional info')]")));

        // Add a small delay to ensure the page is fully loaded
        try {
            Thread.sleep(2000);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }

        // Click the button
        additionalInfoTab.click();

        // Add a delay to ensure the tab content is loaded
        try {
            Thread.sleep(2000);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }


    @Then("the user clicks on view details")
    public void the_user_clicks_on_view_details() {
        // Wait for the AI Description button to be visible and clickable
        WebElement descriptionGenerator =
wait.until(ExpectedConditions.elementToBeClickable(
            By.xpath("//button[contains(text(), 'View AI Details') or contains(text(), 'Get AI-
Powered Insights')]")));

        descriptionGenerator.click();
```

```
        // Wait for the AI description to be generated
        try {
            Thread.sleep(20000);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }

    @After
    public void closeBrowser() {
        if (driver != null) {
            driver.quit();
        }
    }
}
```

Screenshot



Test Report

Test Case 5	
Project Name: Smart Grocery	
Visual Search Test Case	
Test Case ID: 5	**Test Designed By: Sachin Sam Jacob**
Test Priority (Low/Medium/High): High	**Test Designed Date: 16-03-2025**
Module Name: Product Description Generation Module	**Test Executed By: Ms. Sona Maria Sebastian**
Test Title: Product Description Generation	**Test Execution Date: 17-03-2025**

Description: Generates product description using AI					
Pre-Condition: User has should have a valid email 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@123			
4	Click on login button				
5	Click on a product		Product should be viewed	Product is viewed	Pass
6	Clicks on Description and clicks on view details		Description should be generated	Description is generated	Pass
Post-Condition: : User is validated with database and successfully logs into Homepage , clicks on a product, clicks on description and description is generated.					

Test Case 6: Order of Product

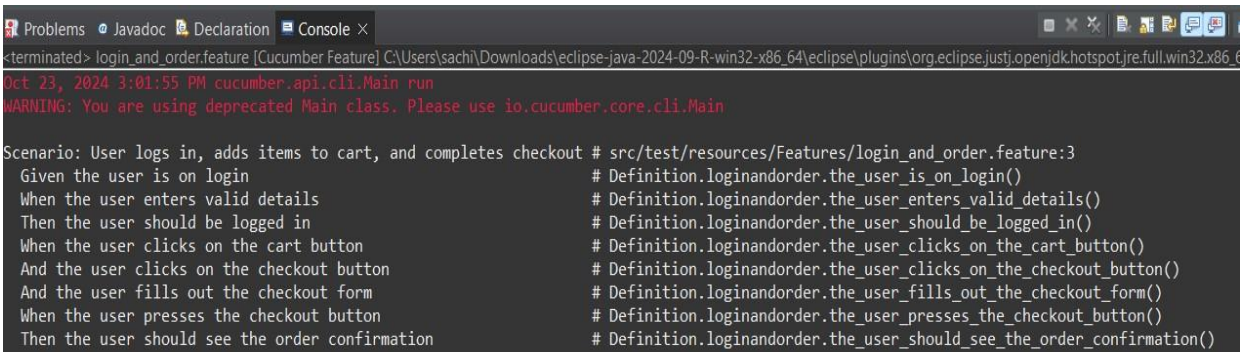
Code:

```
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 6					
Project Name: Smart Grocery					
Checkout Test Case					
Test Case ID: Test_6			**Test Designed By: Sachin Sam Jacob**		
Test Priority (Low/Medium/High): High			**Test Designed Date: 16-03-2025**		
Module Name: Checkout Module			**Test Executed By:** **Ms. Sona Maria Sebastian**		
Test Title: Order of Product			**Test Execution Date: 17-03-2025**		
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@123			
4	Click on login button				
5	Click on cart		User should checkouts the products in cart	User checkouts the products in the cart	Pass
6	Click on Checkout				

turnitin Page 85 of 128 - AI Writing Submission Submission ID trn:oid:::1:3191391174

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.					

CHAPTER 6

IMPLEMENTATION

6.1 INTRODUCTION

A planned system transitions into its operational shape during implementation. The system requires users to feel confident and trust that the solution will work properly to succeed. User training plus informative materials creation constitute the crucial part of this phase. The transition normally takes place throughout and after user training sessions. Organizations activate new systems during implementation by utilizing design plans to produce operational functionality.

During implementation users execute a transition from their present working procedures into new operational methods which might implement full system replacement or introduce small modifications in the system. A proper execution of the process leads to establishing a system that fits the organization's requirements exactly. 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

Implementing software requires both software installation at its destination point and verification of proper functionality. An organization may select someone not directly using the software to serve as the final decision authority for the software development project. Special attention must be given to dispel initial software doubts which could otherwise lead to powerful rejection from users.

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.

The success evaluation requires verification that the server program operates actively on the server. The expected outcomes cannot be reached when the server fails to operate.

6.2.1 User Training

Suitable user training represents an essential element to guarantee people develop skills for utilizing and adjusting to new systems. User training stands essential in developing system comfort and building user confidence during system interaction. Additional complexity in a system leads to a greater necessity for training. All user training sessions teach current and future employees how to enter data as well as how to manage errors and query their system databases and utilize their report customization tools and access system capabilities. The program seeks to endow users with important information and abilities for establishing effective system relationships.

6.2.2 Training on the Application Software

After covering basic computer skills, the following stage involves teaching users how to operate new software programs. Users must receive training about the new system that teaches screen navigation and help resource access and covers errors and their resolutions. The training system gives users and groups both the required know-how and abilities to successfully operate the system and its parts. The training process must differentiate according to user or role groups because organizations need customized solutions to fulfill each member's requirements.

6.2.3 System Maintenance

The process of keeping a system functioning properly presents a complicated challenge in software development activities. The software performs vital functions using smooth operations during the software life cycle maintenance phase. A system starts operating after development but needs sustained caretaking to protect its top operational state. The development process requires software maintenance because it enables systems to transform according to their environmental requirements. Software maintenance requires tasks that surpass the identification process of code errors and their correction. The system requires multiple tasks which collaborate for stability enhancement along with operational effectiveness.

6.2.4 Hosting

Smart Grocery platform uses **Render** as its hosting platform because it offers dependable services along with superior performance alongside simple integration capabilities. The Render platform serves to manage both backend servers and frontend interfaces that serve customers and admin users as well as district operation managers. The integration of these platforms results in a scalable user experience that satisfies the visitors of the website.

Render

The cloud-based platform Render provides deployment tools specifically designed to simplify the process of launching full stack applications when building server-side applications such as backends APIs. The main reasons behind choosing Render involve:

- **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

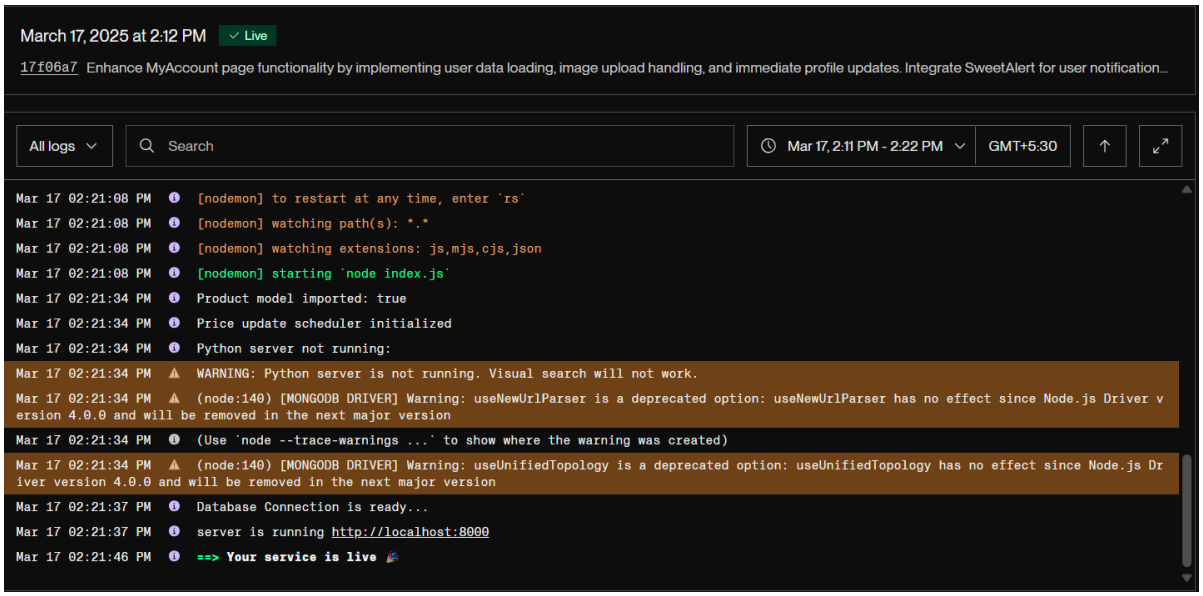
Render handles the build and deployment process, making the backend server live on a secure URL.

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-h9jw.onrender.com

Screenshot



Hosting Process on Render (Frontend)

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 Static Site Setup:

- Select "New Static Site" and choose the connected repository.
- Configure the environment
- Build Command: npm install;npm run build;
- Publish Directory: build
- 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 (Customer): https://smartgrocery.onrender.com

Hosted Link (Admin): https://smartgroceryadmin.onrender.com

Hosted Link (District Operations Manager): https://smartgrocerydom.onrender.com

Hosted Link (Supplier): https://smart-grocery-1.onrender.com

Hosted QR Code (Customer):

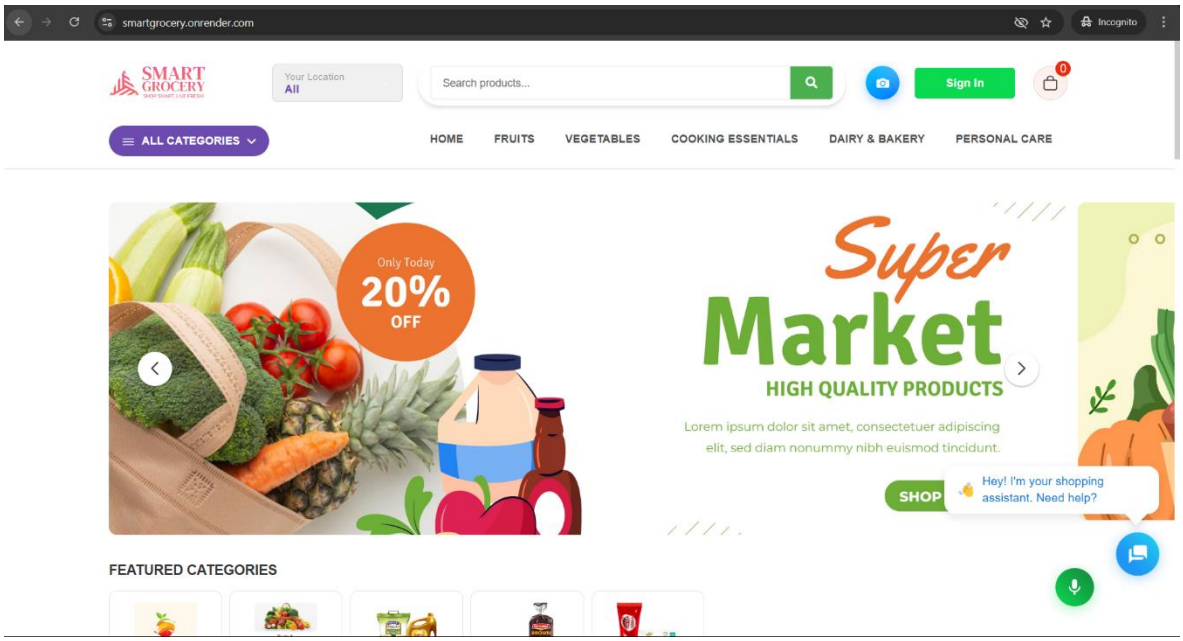
Hosted QR Code (Admin):

Hosted QR Code (District Operation Manager):

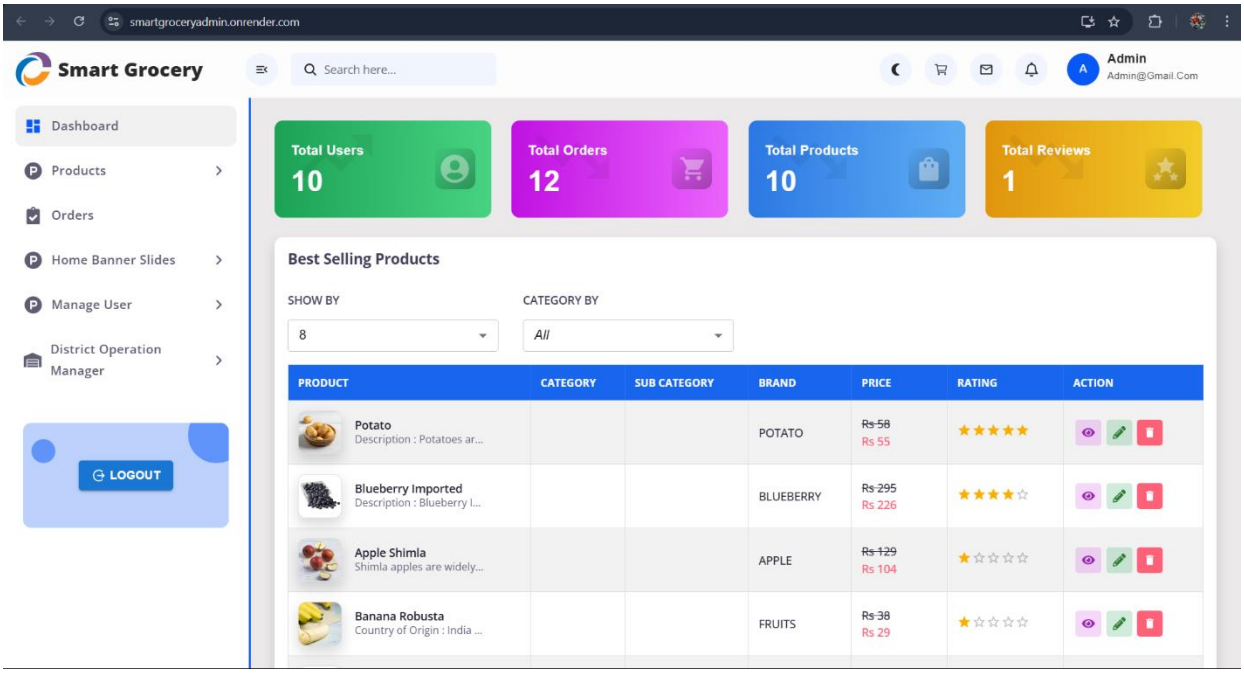
Hosted QR Code (Supplier):

SUPPLIER

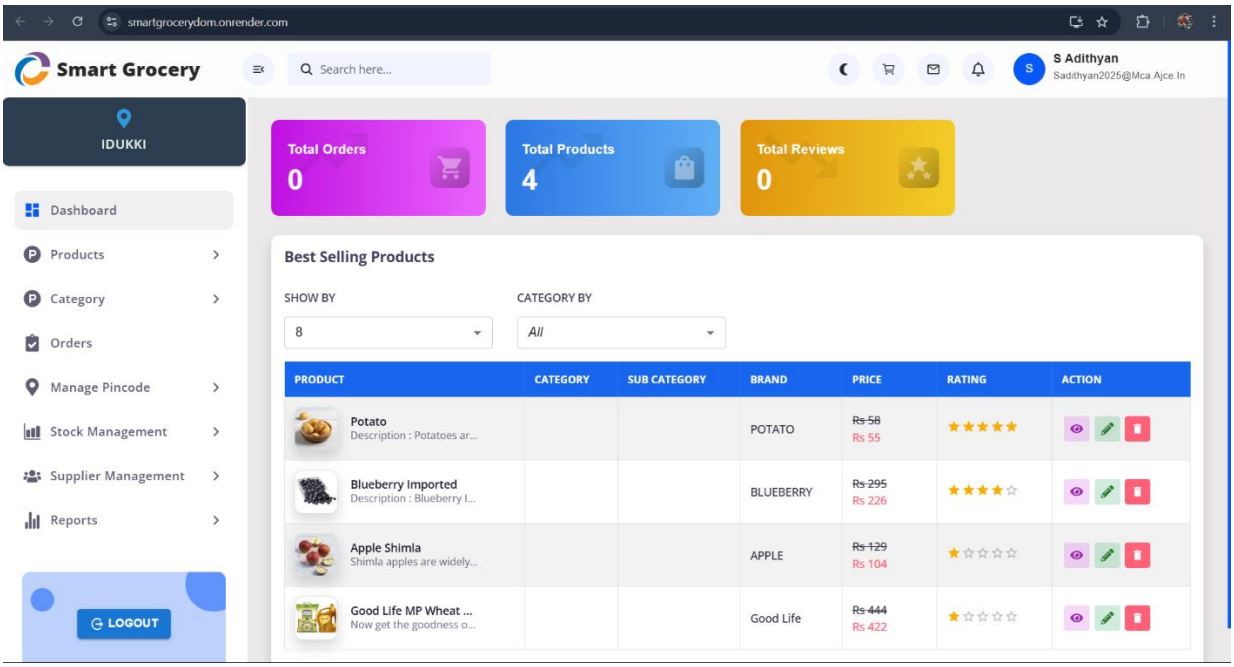
SCREENSHOT (CUSTOMER):



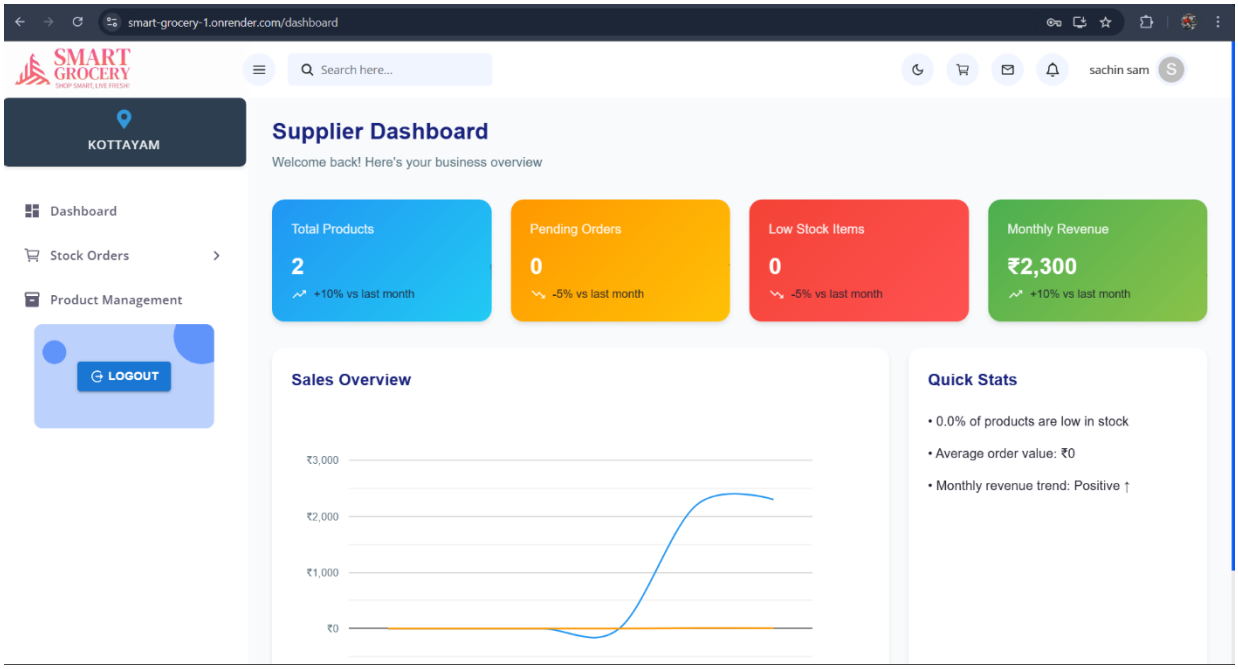
SCREENSHOT (ADMIN):



SCREENSHOT (DISTRICT OPERATION MANAGER):



SCREENSHOT (SUPPLIER):



CHAPTER 7

CONCLUSION AND FUTURE SCOPE

7.1 CONCLUSION

The evaluation shows that Smart Grocery Shopping Platform functions as an advanced AI-powered service that aims to revolutionize contemporary grocery shopping practices. Modern shopping requirements are satisfied through the platform which employs AI-based suggestion systems and dynamic pricing algorithms and voice control protocols and automated stock control for an improved personalized technology-based shopping process.

Seamless shopping meets user security requirements and inventory optimization is possible due to the platform's combination of features like face-authenticated login with AI-powered recipe generation and personalized recommendations and a smart shopping cart and secure payment integration. The system provides users with an easy-to-use interface that shows real-time order monitoring and computerized notifications which result in smooth shopping experiences.

The platform gives administrators detailed control through its full dashboard for handling platform users alongside products and inventory items and active orders. This system gives administrators and district operations managers the tools to conduct effective supply chain logistics monitoring as well as sales trend analysis and stock replenishment optimization using automated alert systems and AI forecast predictions. The platform delivers real-time stock updates together with bulk order management which supports effective product distribution for suppliers.

The Smart Grocery Shopping platform establishes new production standards within the online grocery business through its combination of artificial intelligence and predictive analytics and automation systems. The successful deployment of this platform increases productivity together with security measures and customer interaction which creates advantages for consumers and administrators as it shapes the evolution of grocery retail operations.

7.2 FUTURE SCOPE

The implementation of AI-driven with automation-based features through Smart Grocery platform improves the grocery shopping experience successfully. Several advanced functionalities exist to enhance both efficiency and personalization and security features for future development.

The system will get future improvements through Nutritional Analysis and Diet-Based Recommendations which provide AI-driven item suggestions based on dietary needs along with health objectives. Through blockchain technology the tracking of all products becomes real-time which results in trust and authentic product verification. The implementation of AI-Driven Fraud Detection System helps protect transactions through its ability to detect abnormal

purchasing behavior.

Users will benefit from improved customer engagement through Augmented Reality (AR)-Based Virtual Grocery Shopping which enables them to view products in three dimensions prior to buying them. Last-mile logistics receives an enhancement from drone-based and robotic delivery systems operated autonomously for distribution. The technology identifies opportunities for reducing food waste through recommendations about item transformation and donation just before expiration.

Technology improvements of AI, blockchain along with automation will transform Smart Grocery into an enhanced intelligent efficient sustainable system that establishes innovative standards in online grocery operations.

.

CHAPTER 8

BIBLIOGRAPHY

BIBLIOGRAPHY:

- 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, useLocation } 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 Swal from 'sweetalert2';
import 'animate.css';
import './styles.css';
import { firebaseApp } from "../../firebase"; // Ensure firebaseApp is correctly initialized
import FaceLogin from '../../Components/FaceLogin/FaceLogin';
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
  const location = useLocation();
  const [showFaceLogin, setShowFaceLogin] = useState(false);

  useEffect(() => {
    // Check if user is already logged in
    const token = localStorage.getItem("token");
    if (token) {
      // Redirect to home page if already logged in
      window.location.replace("/");
    }
    context.setisHeaderFooterShow(false);
  }, [navigate]);

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

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

  const handleSuccessfulLogin = (user, token, isAdmin, isStockManager) => {
    localStorage.setItem("token", token);
    localStorage.setItem("user", JSON.stringify(user));

    Swal.fire({
```

```
      title: 'Login successful!',
      icon: 'success',
      timer: 1500,
      timerProgressBar: true,
    });

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

    // Use replace: true to prevent going back to login page
    setTimeout(() => {
      if (isAdmin) {
        window.location.replace("http://localhost:3002/");
      } else if (isStockManager) {
        window.location.replace("/stockmanager-dashboard");
      } else {
        window.location.replace("/");
      }
    }, 2000);
  };

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

    if (formfields.email === "") {
        Swal.fire({
            title: '<span style="color: #dc3545">Email Required</span>',
            html: '<div class="custom-error-message">Please enter your email address</div>',
            icon: 'warning',
            background: '#fff',
            customClass: {
                popup: 'animated fadeInDown error-popup',
                title: 'error-title',
                htmlContainer: 'error-container',
            },
            showConfirmButton: true,
            confirmButtonText: 'OK',
            confirmButtonColor: '#dc3545',
            padding: '2em',
            borderRadius: '15px',
            showClass: {
                popup: 'animate__animated animate__fadeInDown'
            },
            hideClass: {
                popup: 'animate__animated animate__fadeOutUp'
            }
        });
        return false;
    }

    if (formfields.password === "") {
        Swal.fire({
            title: '<span style="color: #dc3545">Password Required</span>',
            html: '<div class="custom-error-message">Please enter your password</div>',
            icon: 'warning',
```

```
            background: '#fff',
            customClass: {
                popup: 'animated fadeInDown error-popup',
                title: 'error-title',
                htmlContainer: 'error-container',
            },
            showConfirmButton: true,
            confirmButtonText: 'OK',
            confirmButtonColor: '#dc3545',
            padding: '2em',
            borderRadius: '15px',
            showClass: {
                popup: 'animate__animated animate__fadeInDown'
            },
            hideClass: {
                popup: 'animate__animated animate__fadeOutUp'
            }
        });
        return false;
    }

    setIsLoading(true);
    postData("/api/user/signin", formfields)
        .then((res) => {
            if (!res.error) {
                const user = {
                    name: res.user?.name,
                    email: res.user?.email,
                    userId: res.user?.id,
                    isStockManager: res.user?.isStockManager,
                    location: res.user?.location,
                };
                handleSuccessfulLogin(user, res.token, res.user.isAdmin, res.user.isStockManager);
            } else {
                throw new Error(res.msg);
            }
        })
        .catch((error) => {
            let errorMessage = error.message;
            // Handle specific error responses
            if (error.message.includes('HTTP error! status: 400')) {
                errorMessage = 'Incorrect password. Please try again.';
            } else if (error.message.includes('HTTP error! status: 404')) {
                errorMessage = 'No account found with this email address.';
            } else if (error.message.includes('HTTP error! status: 403')) {
                errorMessage = 'Your account has been blocked. Please contact support.';
            }

            Swal.fire({
                title: '<span style="color: #dc3545">Error</span>',
                html: `<div class="custom-error-message">${errorMessage}</div>`,
                icon: 'error',
                background: '#fff',
                customClass: {
                    popup: 'animated fadeInDown error-popup',
```

```
                title: 'error-title',
                htmlContainer: 'error-container',
            },
            showConfirmButton: true,
            confirmButtonText: 'OK',
            confirmButtonColor: '#dc3545',
            padding: '2em',
            borderRadius: '15px',
            showClass: {
                popup: 'animate__animated animate__fadeInDown'
            },
            hideClass: {
                popup: 'animate__animated animate__fadeOutUp'
            }
        });
        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) {
            const user = {
              name: res.user?.name,
              email: res.user?.email,
              userId: res.user?.id,
            };

            handleSuccessfulLogin(user, res.token, res.user.isAdmin, res.user.isStockManager);
          } else {
            Swal.fire({
              title: 'Error!',
              text: res.msg,
              icon: 'error',
              timer: 1500,
              timerProgressBar: true,
            });
            setIsLoading(false);
          }
        } catch (error) {
```

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

      Swal.fire({
        title: 'Login successful!',
        icon: 'success',
        timer: 1500,
        timerProgressBar: true,
      });
    })
    .catch((error) => {
      Swal.fire({
        open: true,
        error: true,
        msg: error.message,
      });
    });
};

const handleFaceLogin = async (faceDescriptor) => {
  try {
    setIsLoading(true);
    console.log('Attempting face login...');
    console.log(faceDescriptor);
    const response = await postData('/api/face/login', { faceDescriptor });
    console.log(response);
    if (response.success) {
      const user = {
        name: response.user.name,
        email: response.user.email,
        userId: response.user.id,
        isStockManager: response.user.isStockManager,
        location: response.user.location,
      };

      handleSuccessfulLogin(
        user,
        response.token,
        response.user.isAdmin,
        response.user.isStockManager
      );
    } else {
      throw new Error(response.message || 'Face login failed');
    }
  } catch (error) {
    console.error('Face login error:', error);
    Swal.fire({
      title: 'Face Login Failed',
      text: 'Face ID not recognized. Please try again or use password.',
      icon: 'error',
      showCancelButton: true,
      confirmButtonText: 'Try Again',
      cancelButtonText: 'Use Password'
```

```
    }).then((result) => {
      if (result.isConfirmed) {
        setShowFaceLogin(true); // Retry face login
      } else {
        setShowFaceLogin(false); // Switch to password login
      }
    });
  } finally {
    setIsLoading(false);
  }
};

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

const renderFaceLoginModal = () => {
  if (!showFaceLogin) return null;

  return (
    <div className="face-login-modal">
      <div className="face-login-modal-content">
        <button
          className="close-modal-btn"
          onClick={() => setShowFaceLogin(false)}
        >
          <i className="fas fa-times"></i>
        </button>
        <FaceLogin
          onFaceDetected={handleFaceLogin}
          mode="login"
        />
      </div>
    </div>
  );
};

return (
  <section className="section signInPage">
    {/* Add Back to Homepage Button */}
    <div className="back-to-home-wrapper">
      <button
        onClick={handleBackToHomepage}
        className="back-to-home-btn"
      >
        <i className="fas fa-arrow-left"></i>
        <span>Back to Homepage</span>
      </button>
    </div>

    <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"

           variant="standard"
           className="w-100"
           name="email"
           onChange={onchangeInput}
          />
         </div>
         <div className="form-group">
          <TextField
           id="passwords"
           label="Password"
           type="password"

           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>

        <div className="login-options mt-4">
          <Button
            className="loginWithGoogle mb-3 w-100"
            variant="outlined"
            onClick={signInWithGoogle}
          >
            <img src={GoogleImg} alt="Google" /> Sign In with Google
          </Button>

          <Button
            className="faceIdLogin w-100"
            variant="outlined"
            id="faceIdLogin"
            onClick={() => setShowFaceLogin(true)}
            startIcon={<i className="fas fa-face-viewfinder"></i>}
          >
            Sign In with Face ID
          </Button>
        </div>
      </form>
    </div>
  </div>

  {/* Render face login modal */}
  {renderFaceLoginModal()}
 </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: "No account found with this email address",
                details: "Please check your email or create a new account"
            });
        }

        if (existingUser.isBlocked) {
            return res.status(403).json({
                error: true,
                msg: "Account Blocked",
                details: "Your account has been blocked due to unauthorized activities. Please contact
support."
            });
        }

        const isPasswordCorrect = await bcrypt.compare(password, existingUser.password);
        if (!isPasswordCorrect) {
            return res.status(400).json({
                error: true,
                msg: "Incorrect Password",
                details: "The password you entered is incorrect. Please try again."
            });
        }

        const token = jwt.sign(
            {
                email: existingUser.email,
                id: existingUser._id,
                location: existingUser.location
            },
            process.env.JSON_WEB_TOKEN_SECRET_KEY
        );

        return res.status(200).send({
            user: existingUser,
            token: token,
            msg: "Login Successful",
            details: "Welcome back!"
        });

    } catch (error) {
        res.status(500).json({
            error: true,
```

```
            msg: "Server Error",
            details: "Something went wrong. Please try again later."
        });
    }
});
```


turnitin Page 111 of 128 - AI Writing Submission Submission ID trn:oid:::1:3191391174

9.2 Screen Shots

9.2.1. Login Page

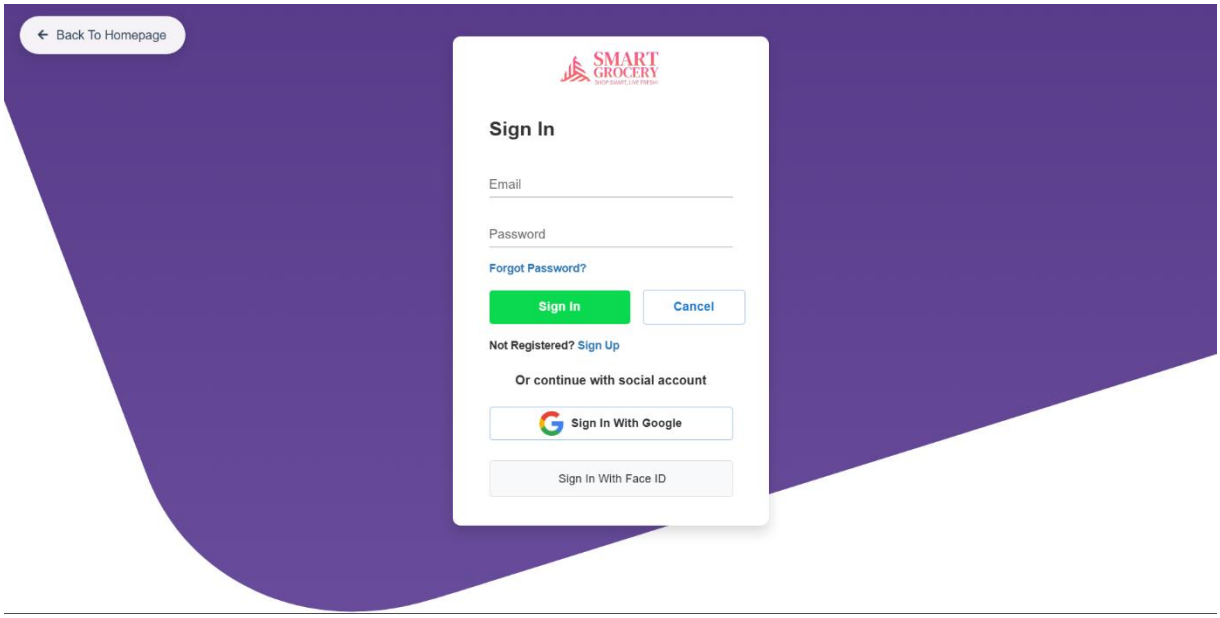


Fig 9.2.1 Login Page

9.2.2. Registration Page

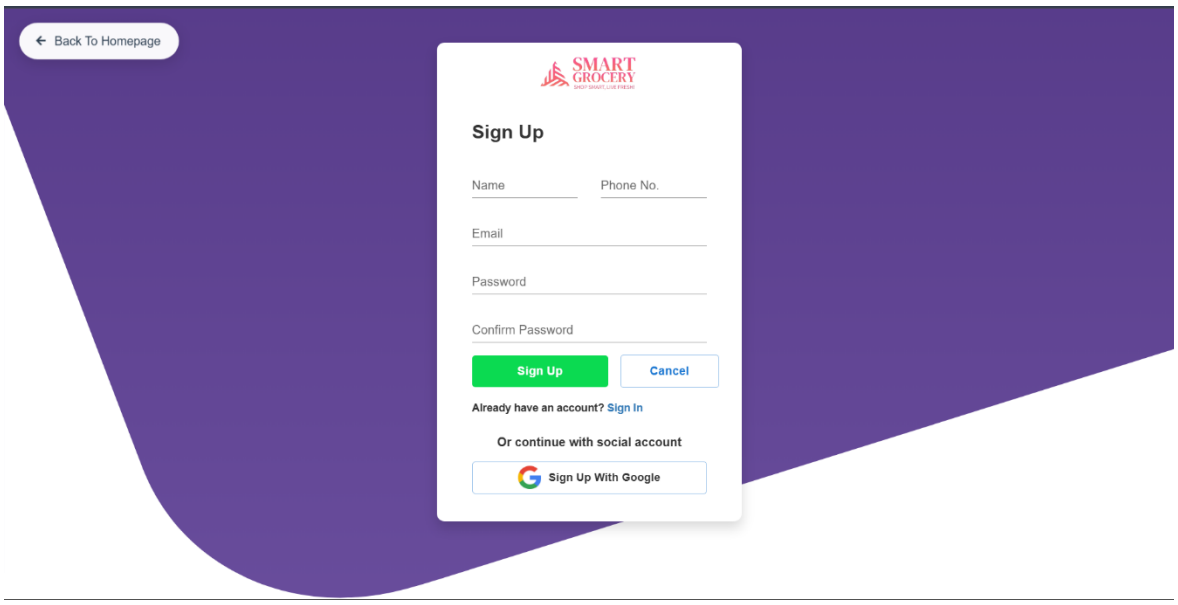
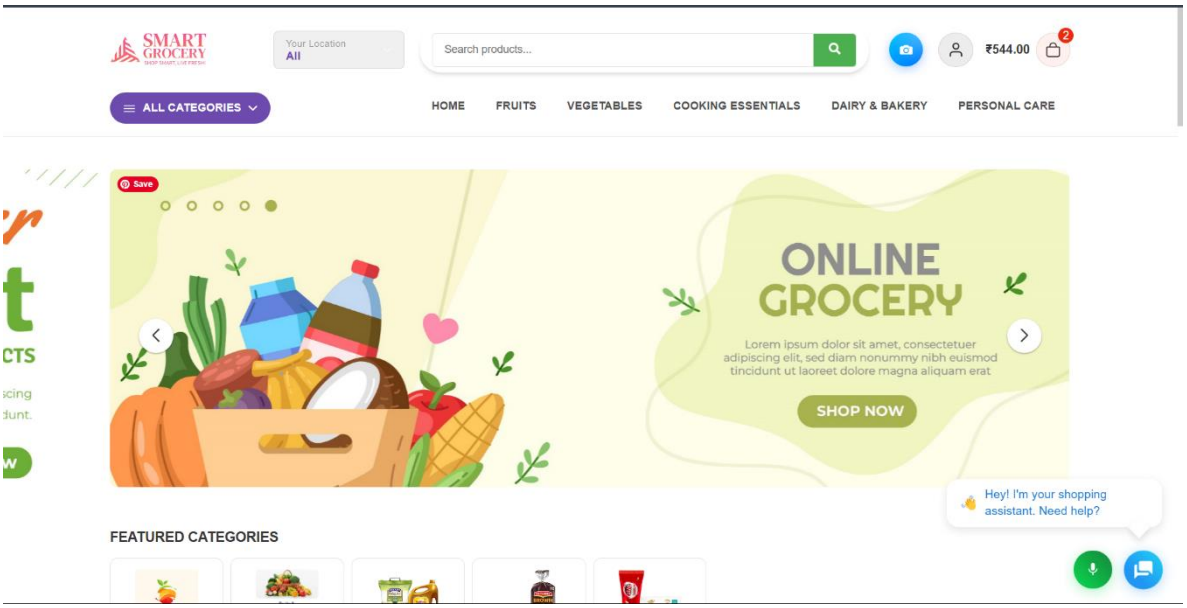


Fig 9.2.2 Registration Page

9.2.3. HomePage



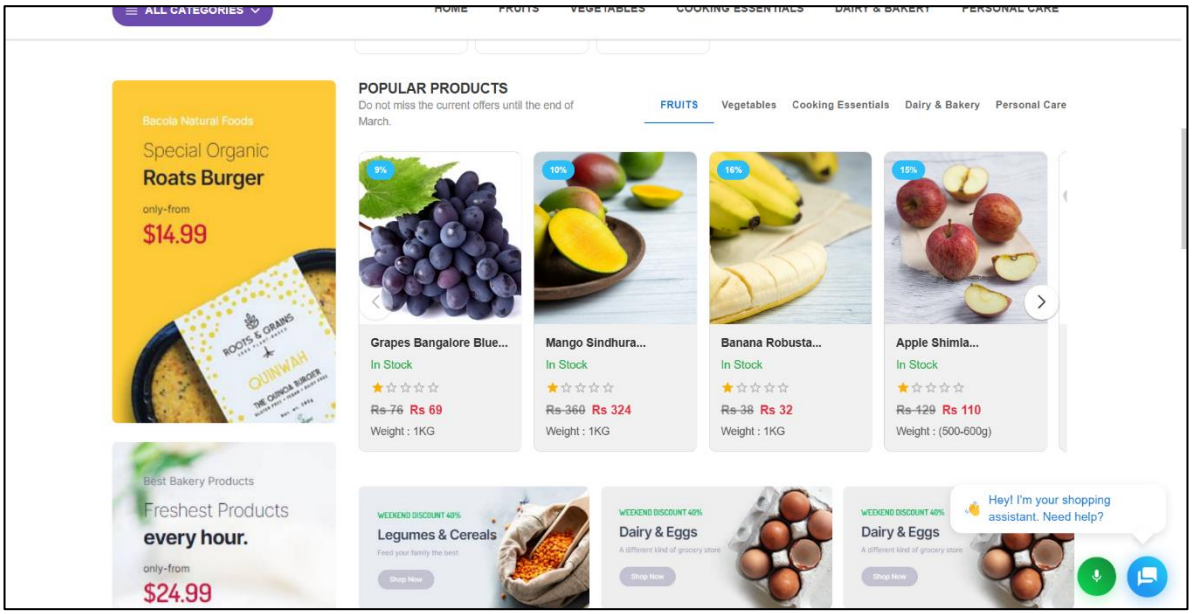


Fig 9.2.3 HomePage

9.2.4. ProductView Page

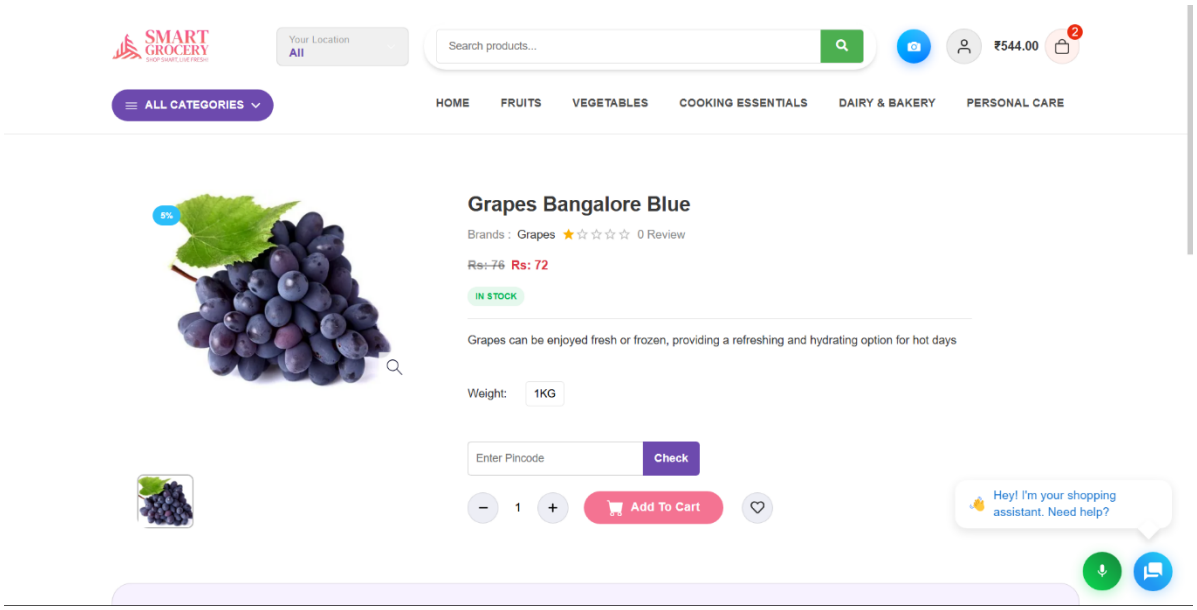


Fig 9.2.4. ProductView Page

9.2.5. Wishlist Page

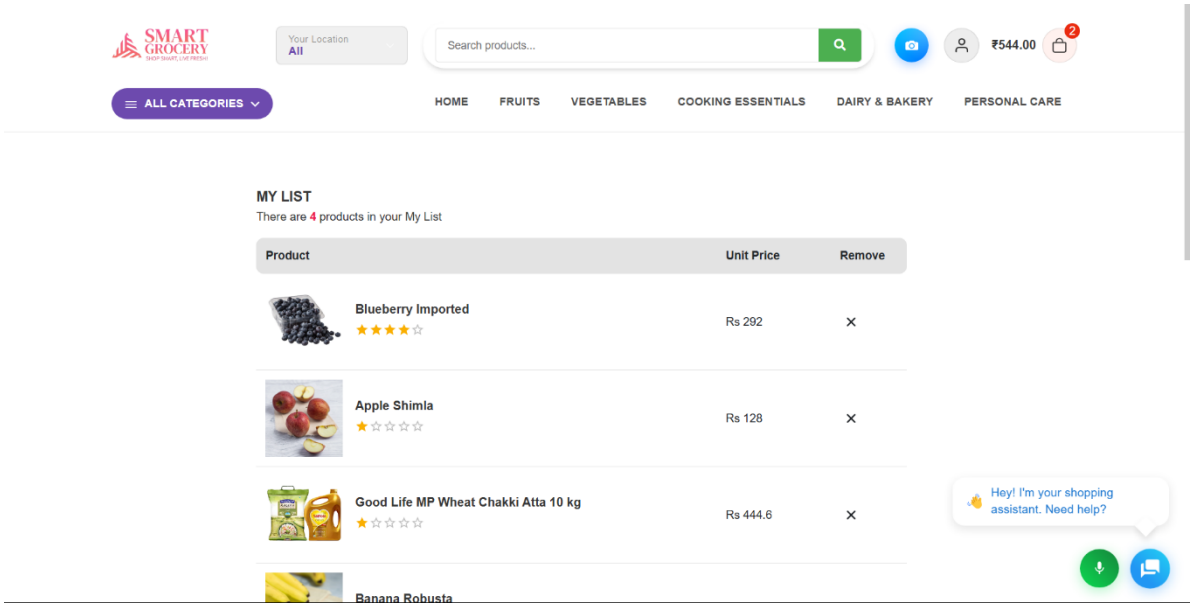


Fig 9.2.5. Wishlist Page

9.2.6. Cart Page

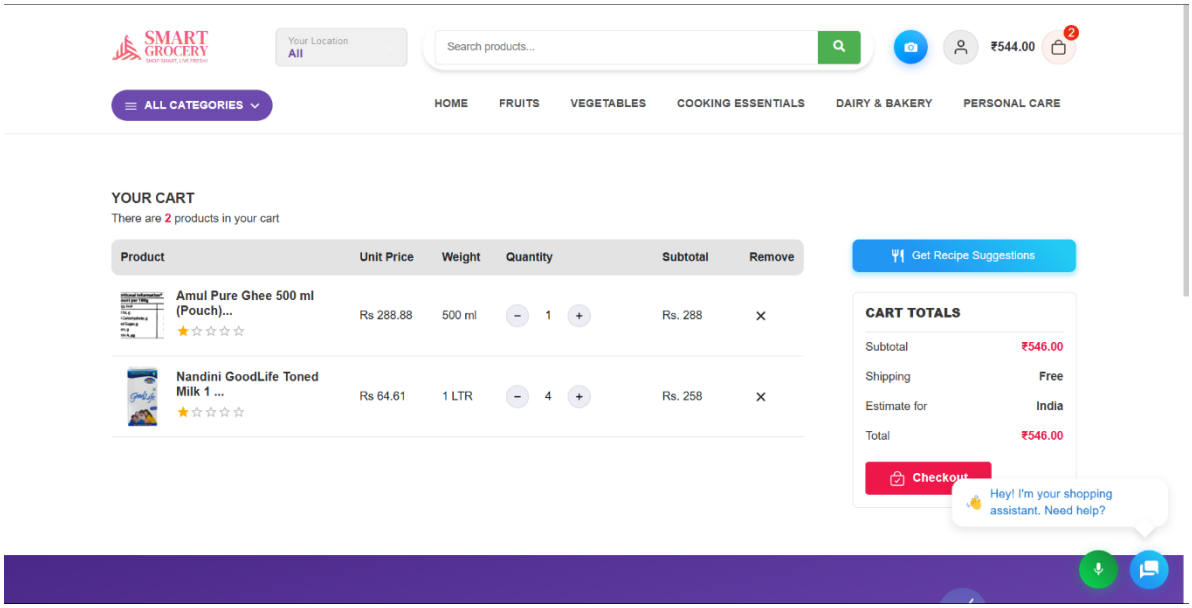


Fig 9.2.6. Cart Page

9.2.7. Orders Page

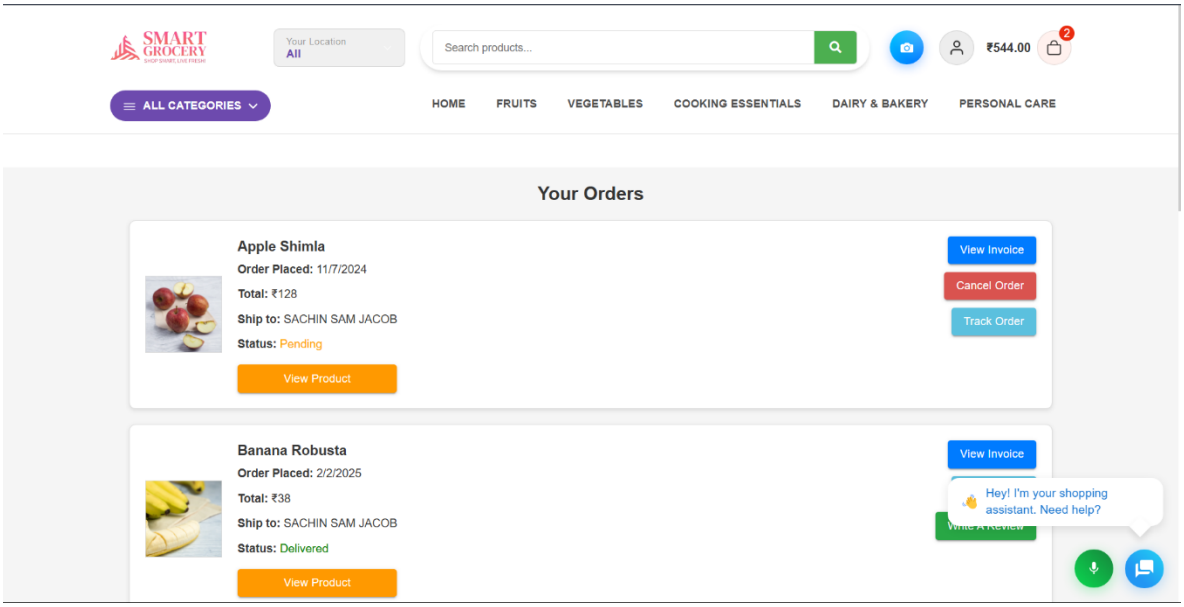


Fig 9.2.7. Orders Page

9.2.8.Checkout Page

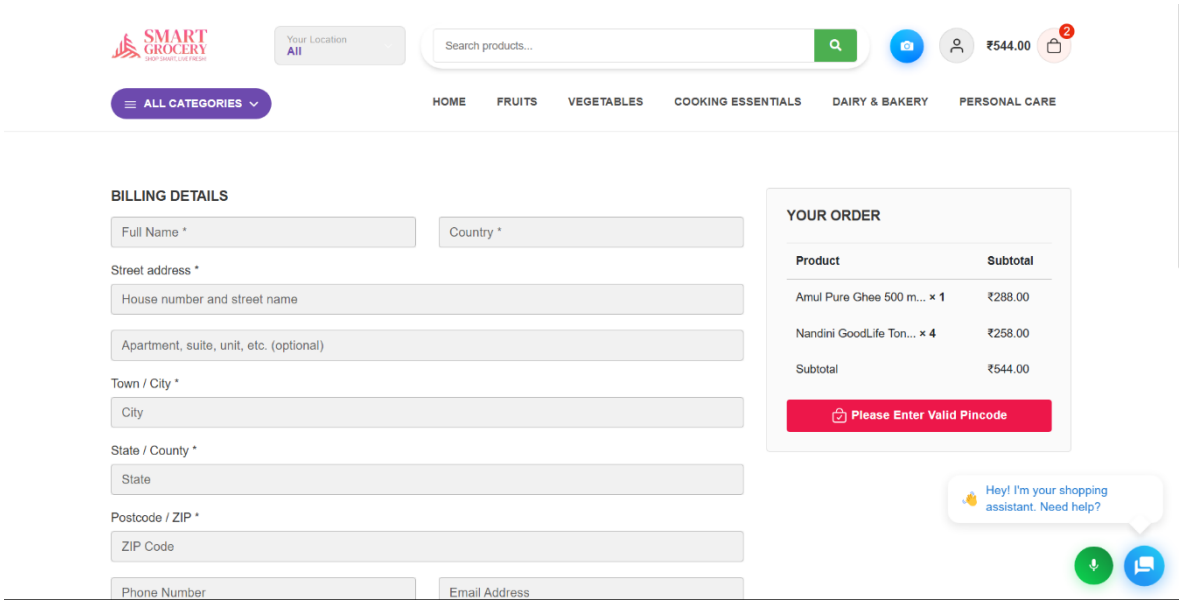


Fig 9.2.8. Checkout Page

9.2.9. Update Profile Page

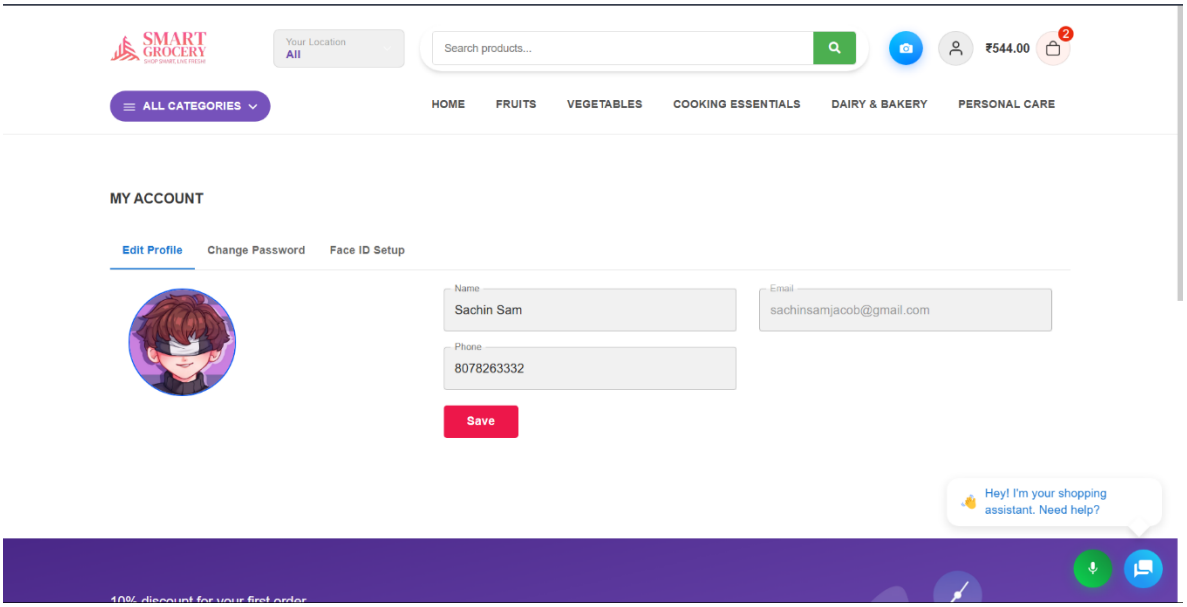


Fig 9.2.9. Update Profile Page

9.2.10. Change Password Page

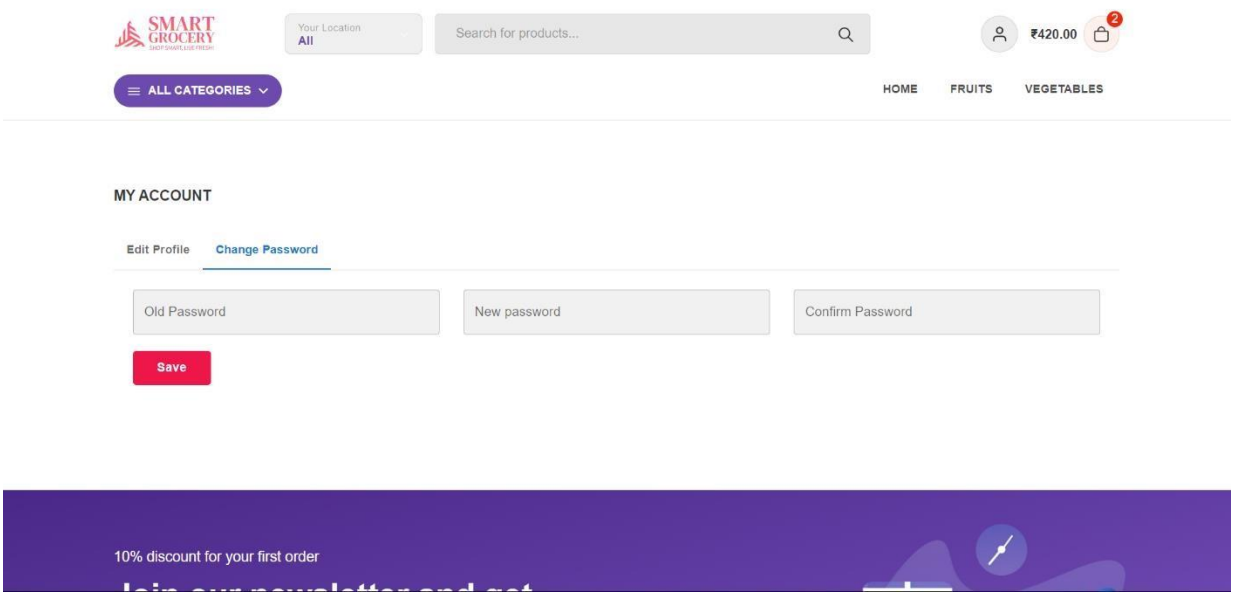


Fig 9.2.10. Change Password Page

9.2.11. Admin Dashboard

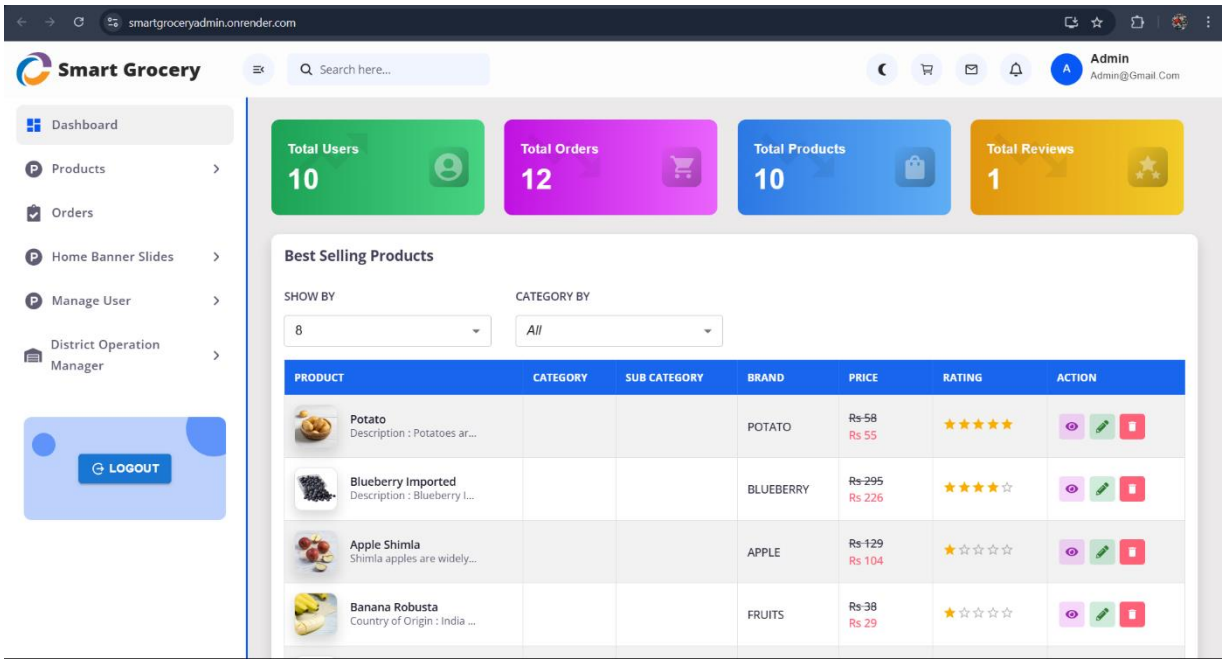


Fig 9.2.11. Admin Dashboard

9.2.12. District Operations Manager Dashboard

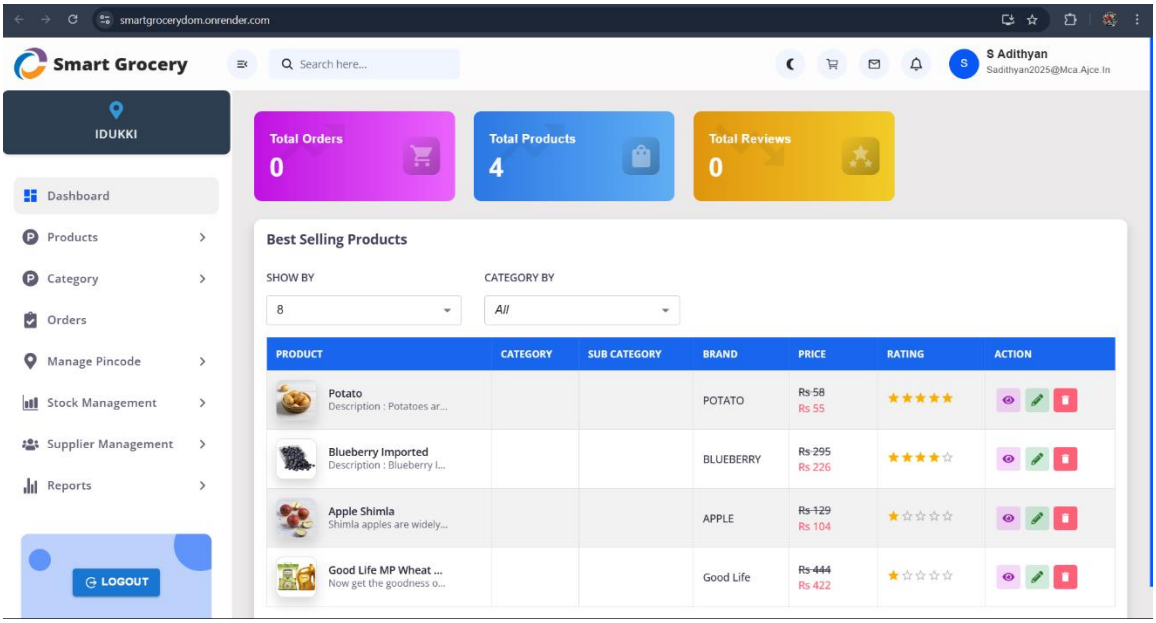


Fig 9.2.12. District Operations Manager Dashboard

9.2.13 Supplier Dashboard

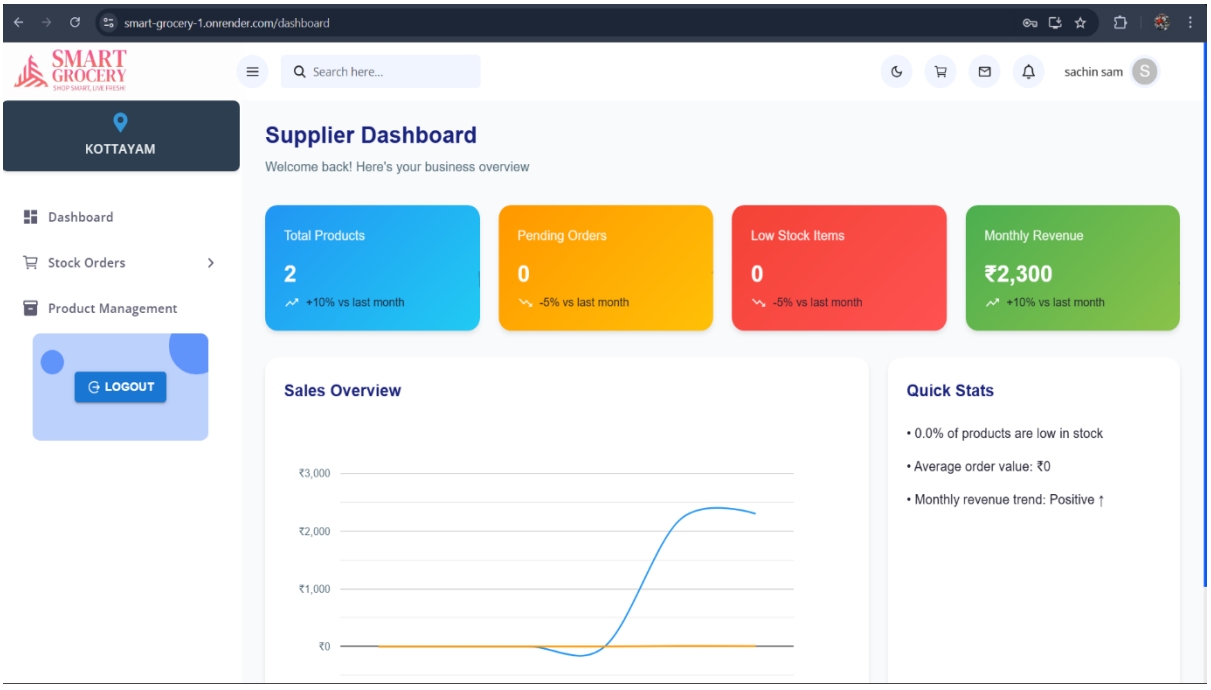


Fig 9.2.13. Supplier Dashboard

9.3 CERTIFICATES

9.3.1. MERN: Advanced Mern Development

Fig 9.3.1. Mern: Advanced Mern Development

9.3.2. Fundamentals of AI & ML: Introduction to Artificial Intelligence

Fig 9.3.2. Fundamentals of AI & ML: Introduction to Artificial Intelligence

9.3.3. Introduction to Express JS

Fig 9.3.3. Introduction to Express Js

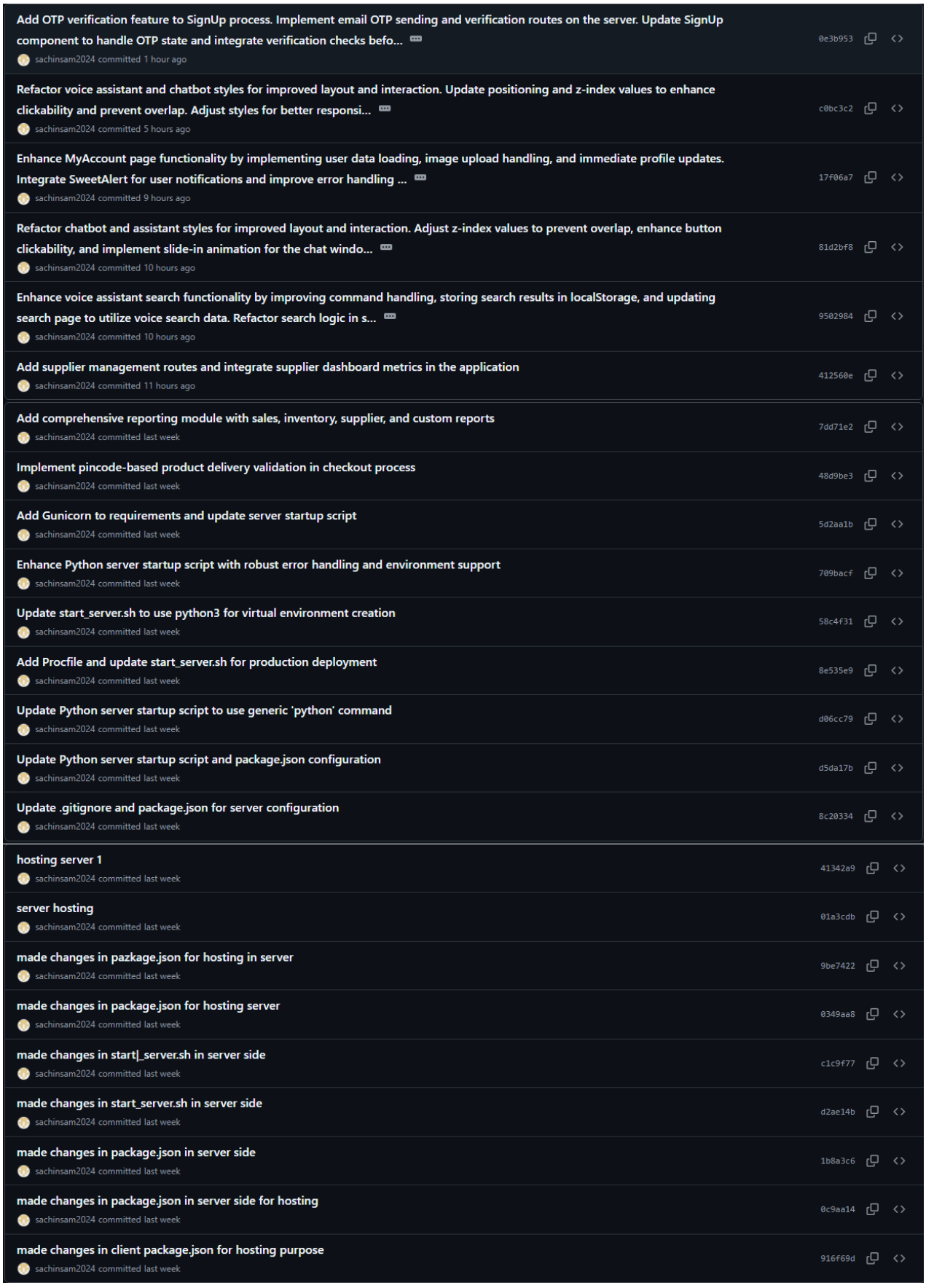
9.3.4. Introduction to MongoDB

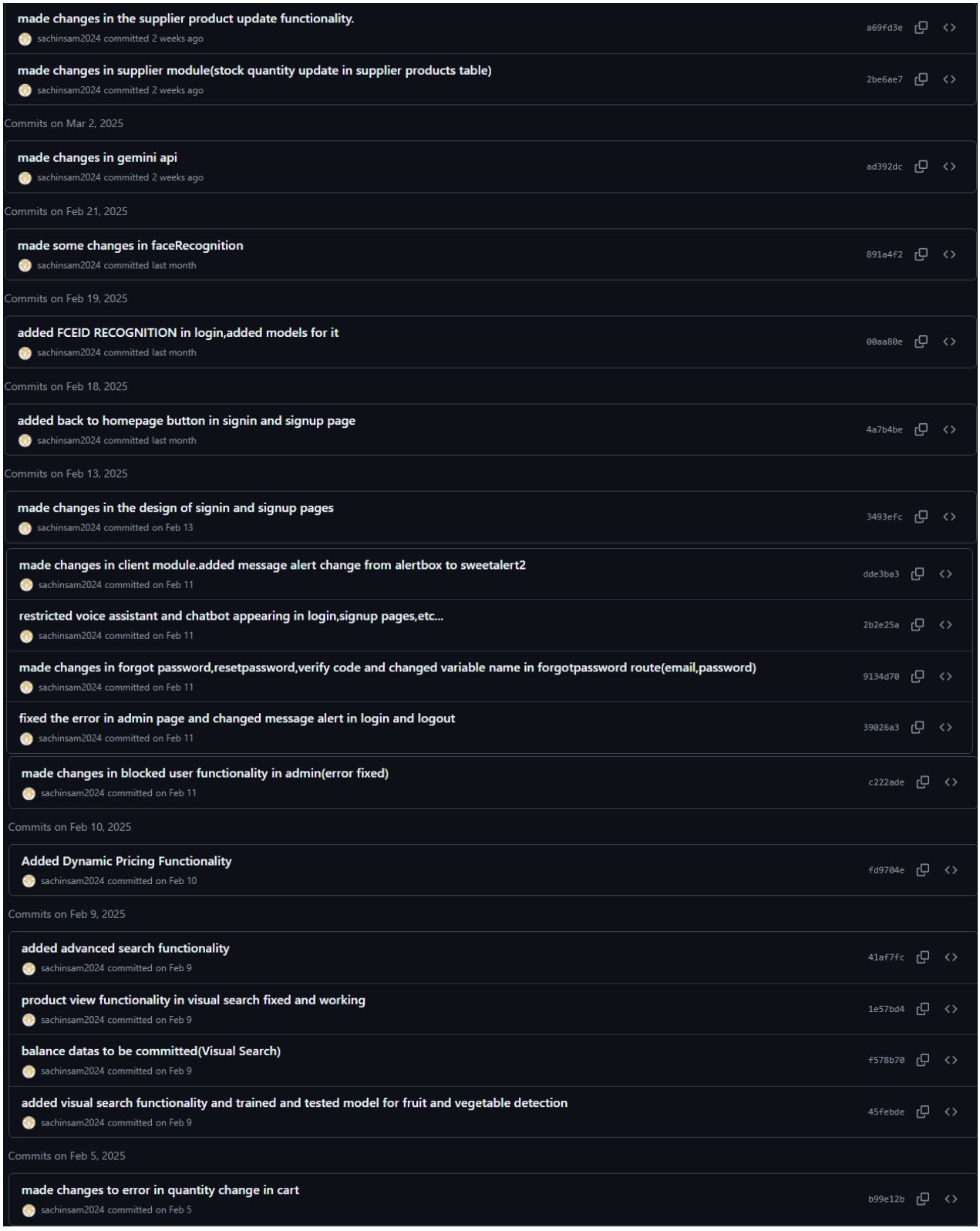
Fig 9.3.4. Introduction to MongoDB

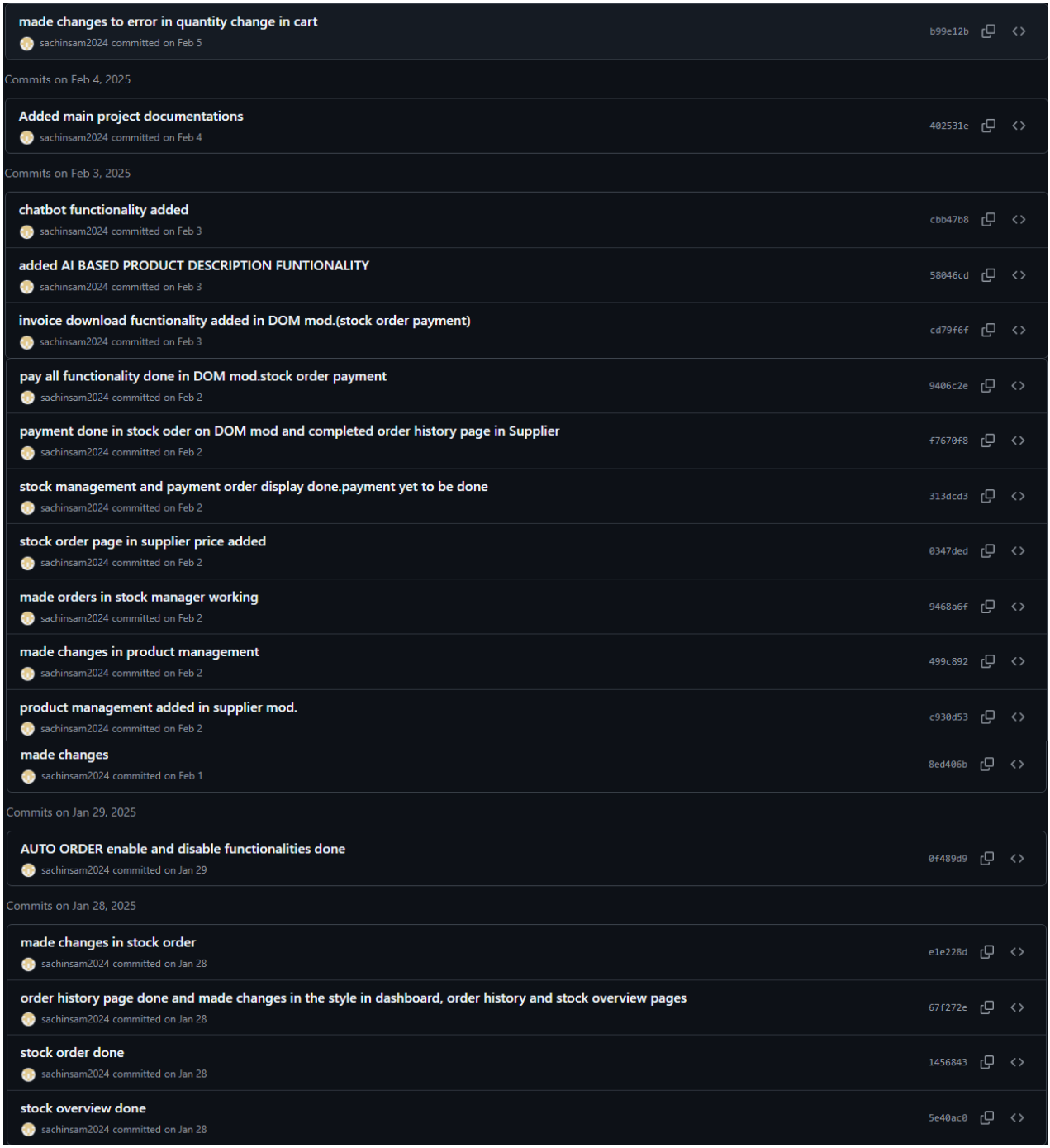
9.3.5. Getting Started with NodeJS

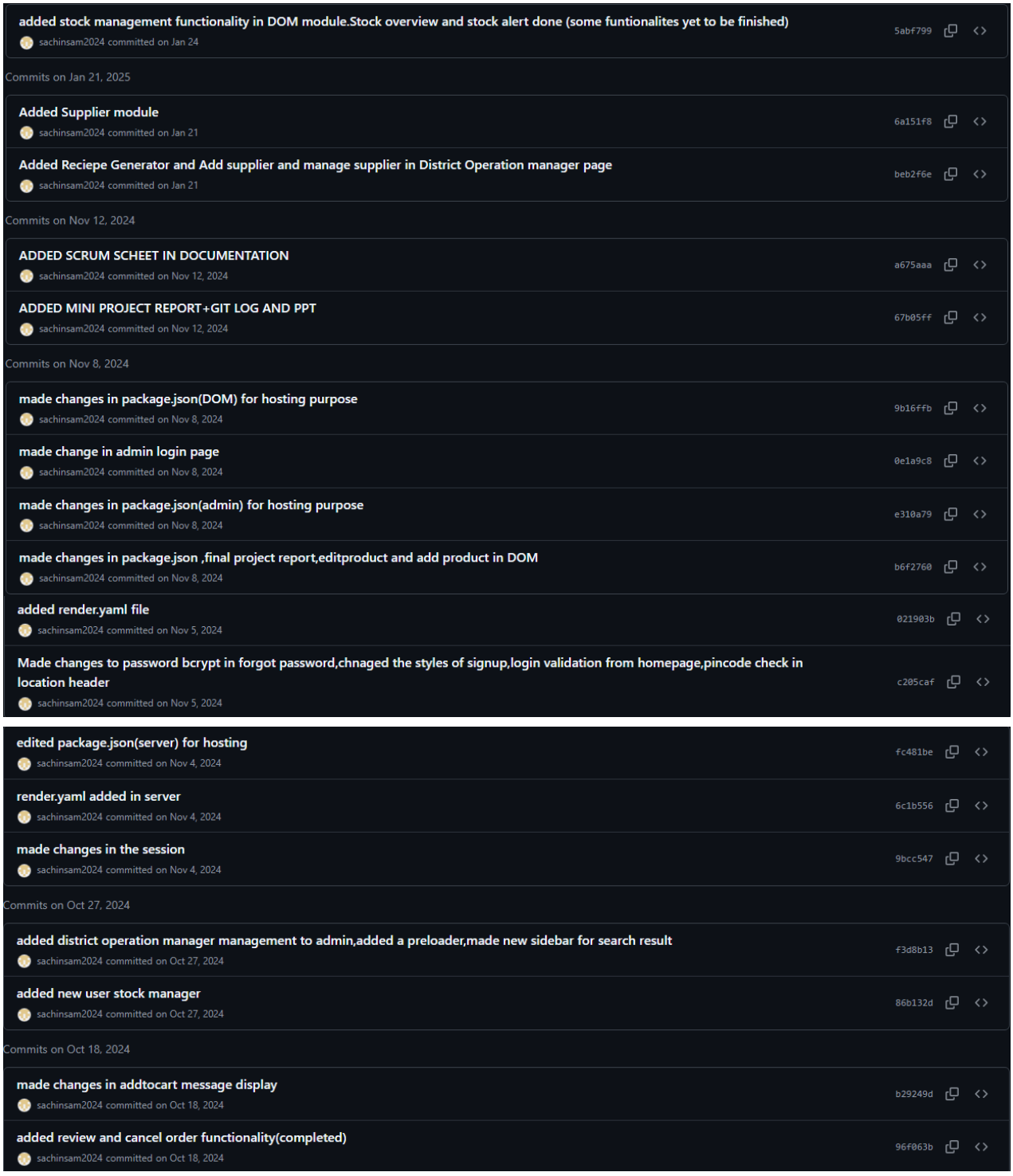
Fig 9.3.5. Getting Started with NodeJS

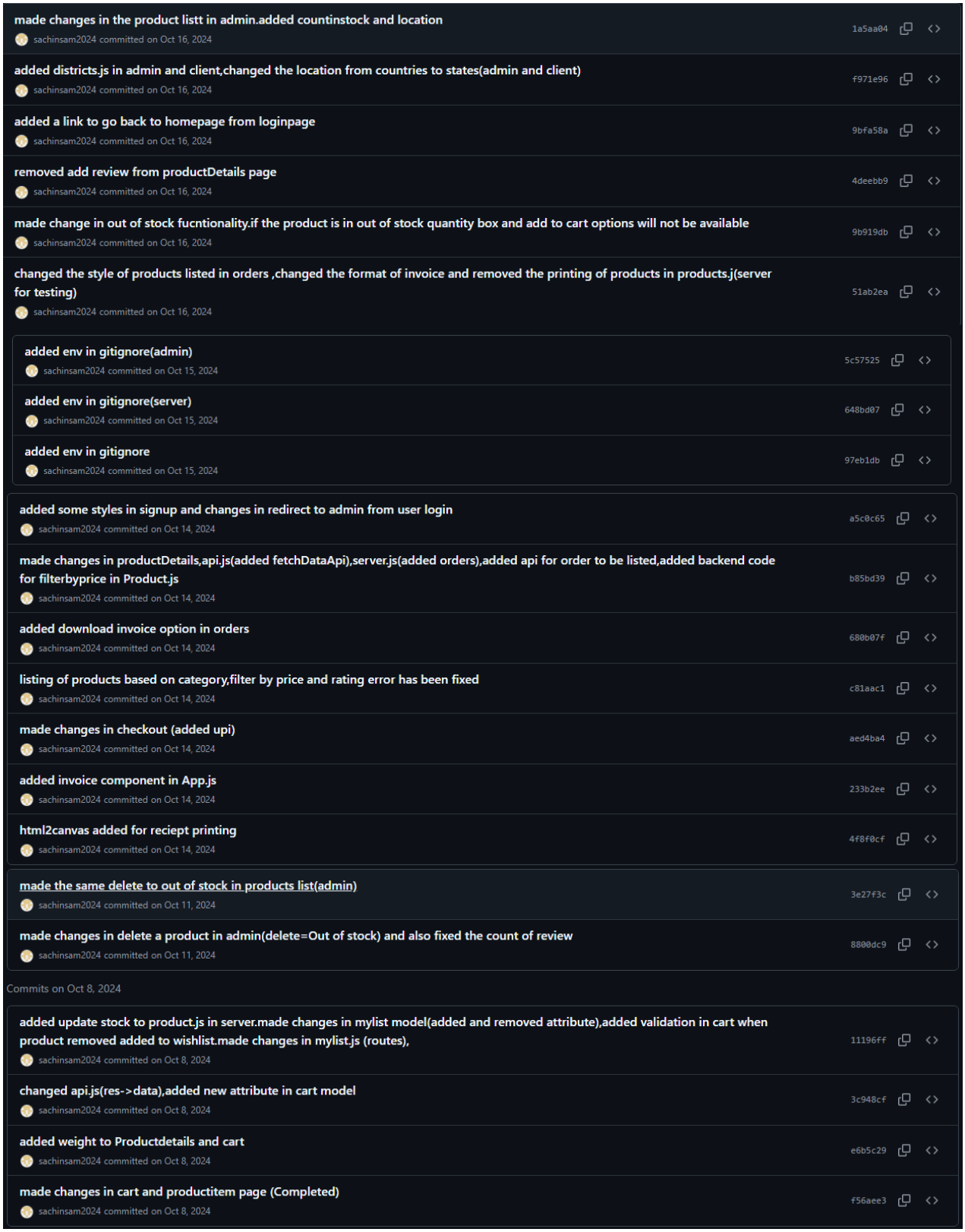
9.4 GIT LOG

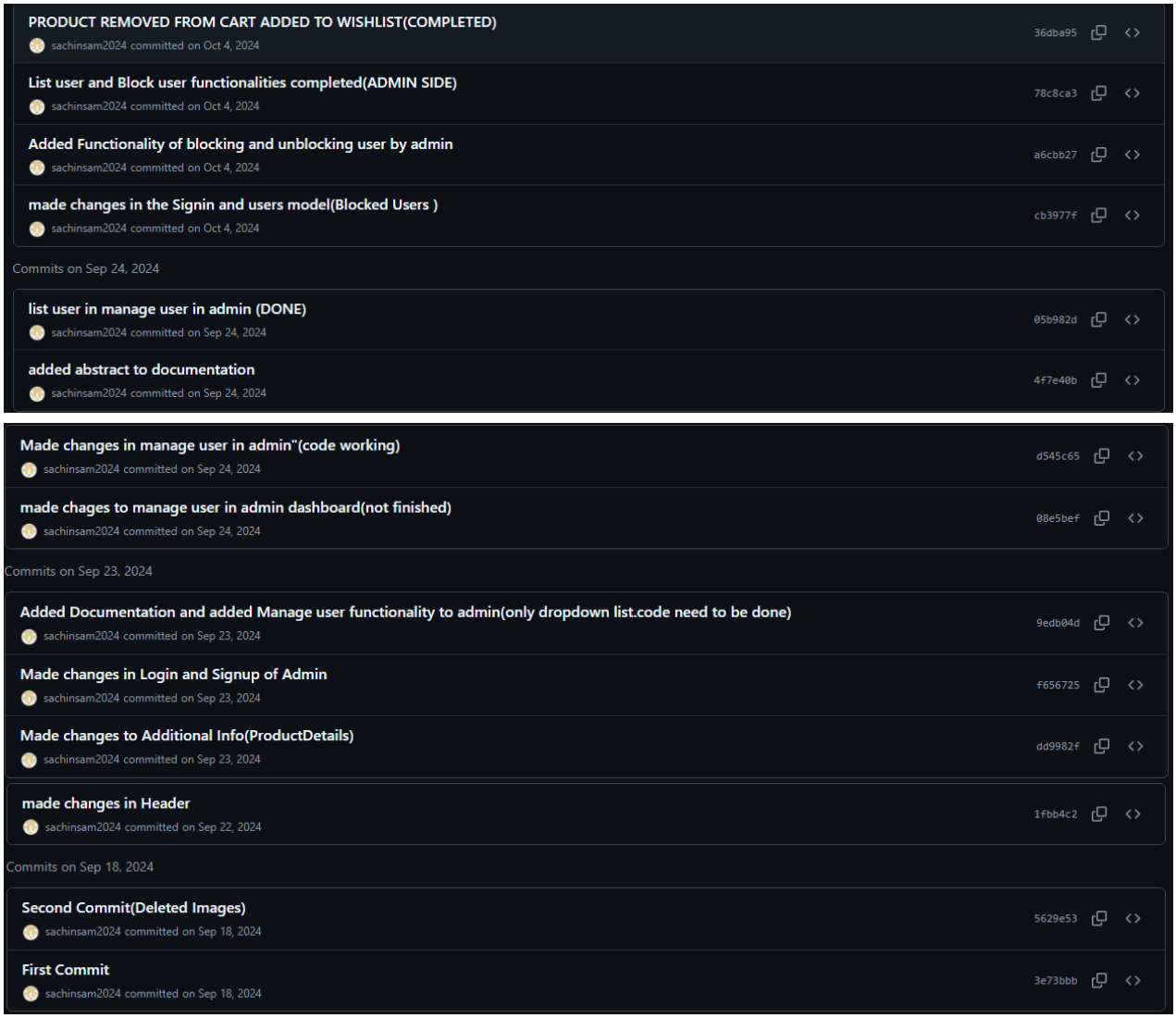

Add OTP verification feature to SignUp process. Implement email OTP sending and verification routes on the server. Update SignUp component to handle OTP state and integrate verification checks befo... 0e3b953
sachinsam2024 committed 1 hour ago
Refactor voice assistant and chatbot styles for improved layout and interaction. Update positioning and z-index values to enhance clickability and prevent overlap. Adjust styles for better responsi... c0bc3c2
sachinsam2024 committed 5 hours ago
Enhance MyAccount page functionality by implementing user data loading, image upload handling, and immediate profile updates. Integrate SweetAlert for user notifications and improve error handling ... 17f06a7
sachinsam2024 committed 9 hours ago
Refactor chatbot and assistant styles for improved layout and interaction. Adjust z-index values to prevent overlap, enhance button clickability, and implement slide-in animation for the chat windo... 81d2bf8
sachinsam2024 committed 10 hours ago
Enhance voice assistant search functionality by improving command handling, storing search results in localStorage, and updating search page to utilize voice search data. Refactor search logic in s... 9502984
sachinsam2024 committed 10 hours ago
Add supplier management routes and integrate supplier dashboard metrics in the application 412560e
sachinsam2024 committed 11 hours ago
Add comprehensive reporting module with sales, inventory, supplier, and custom reports 7dd71e2
sachinsam2024 committed last week
Implement pincode-based product delivery validation in checkout process 48d9be3
sachinsam2024 committed last week
Add Gunicorn to requirements and update server startup script 5d2aa1b
sachinsam2024 committed last week
Enhance Python server startup script with robust error handling and environment support 709bacf
sachinsam2024 committed last week
Update start_server.sh to use python3 for virtual environment creation 58c4f31
sachinsam2024 committed last week
Add Procfile and update start_server.sh for production deployment 8e535e9
sachinsam2024 committed last week
Update Python server startup script to use generic 'python' command d06cc79
sachinsam2024 committed last week
Update Python server startup script and package.json configuration d5da17b
sachinsam2024 committed last week
Update .gitignore and package.json for server configuration 8c20334
sachinsam2024 committed last week
hosting server 1 41342a9
sachinsam2024 committed last week
server hosting 01a3cdb
sachinsam2024 committed last week
made changes in pazkage.json for hosting in server 9be7422
sachinsam2024 committed last week
made changes in package.json for hosting server 0349aa8
sachinsam2024 committed last week
made changes in start|_server.sh in server side c1c9f77
sachinsam2024 committed last week
made changes in start_server.sh in server side d2ae14b
sachinsam2024 committed last week
made changes in package.json in server side 1b8a3c6
sachinsam2024 committed last week
made changes in package.json in server side for hosting 0c9aa14
sachinsam2024 committed last week
made changes in client package.json for hosting purpose 916f69d
sachinsam2024 committed last week


made changes in the supplier product update functionality.
sachinsam2024 committed 2 weeks ago
a69fd3e
made changes in supplier module(stock quantity update in supplier products table)
sachinsam2024 committed 2 weeks ago
2be6ae7
Commits on Mar 2, 2025
made changes in gemini api
sachinsam2024 committed 2 weeks ago
ad392dc
Commits on Feb 21, 2025
made some changes in faceRecognition
sachinsam2024 committed last month
891a4f2
Commits on Feb 19, 2025
added FCEID RECOGNITION in login,added models for it
sachinsam2024 committed last month
00aa80e
Commits on Feb 18, 2025
added back to homepage button in signin and signup page
sachinsam2024 committed last month
4a7b4be
Commits on Feb 13, 2025
made changes in the design of signin and signup pages
sachinsam2024 committed on Feb 13
3493efc
made changes in client module.added message alert change from alertbox to sweetalert2
sachinsam2024 committed on Feb 11
dde3ba3
restricted voice assistant and chatbot appearing in login,signup pages,etc...
sachinsam2024 committed on Feb 11
2b2e25a
made changes in forgot password,resetpassword,verify code and changed variable name in forgotpassword route(email,password)
sachinsam2024 committed on Feb 11
9134d70
fixed the error in admin page and changed message alert in login and logout
sachinsam2024 committed on Feb 11
39026a3
made changes in blocked user functionality in admin(error fixed)
sachinsam2024 committed on Feb 11
c222ade
Commits on Feb 10, 2025
Added Dynamic Pricing Functionality
sachinsam2024 committed on Feb 10
fd9704e
Commits on Feb 9, 2025
added advanced search functionality
sachinsam2024 committed on Feb 9
41af7fc
product view functionality in visual search fixed and working
sachinsam2024 committed on Feb 9
1e57bd4
balance datas to be committed(Visual Search)
sachinsam2024 committed on Feb 9
f578b70
added visual search functionality and trained and tested model for fruit and vegetable detection
sachinsam2024 committed on Feb 9
45febde
Commits on Feb 5, 2025
made changes to error in quantity change in cart
sachinsam2024 committed on Feb 5
b99e12b


made changes to error in quantity change in cart
sachinsam2024 committed on Feb 5
b99e12b
Commits on Feb 4, 2025
Added main project documentations
sachinsam2024 committed on Feb 4
402531e
Commits on Feb 3, 2025
chatbot functionality added
sachinsam2024 committed on Feb 3
cbb47b8
added AI BASED PRODUCT DESCRIPTION FUNTIONALITY
sachinsam2024 committed on Feb 3
58046cd
invoice download fucntionality added in DOM mod.(stock order payment)
sachinsam2024 committed on Feb 3
cd79f6f
pay all functionality done in DOM mod.stock order payment
sachinsam2024 committed on Feb 2
9406c2e
payment done in stock oder on DOM mod and completed order history page in Supplier
sachinsam2024 committed on Feb 2
f7670f8
stock management and payment order display done.payment yet to be done
sachinsam2024 committed on Feb 2
313dcd3
stock order page in supplier price added
sachinsam2024 committed on Feb 2
0347ded
made orders in stock manager working
sachinsam2024 committed on Feb 2
9468a6f
made changes in product management
sachinsam2024 committed on Feb 2
499c892
product management added in supplier mod.
sachinsam2024 committed on Feb 2
c930d53
made changes
sachinsam2024 committed on Feb 1
8ed406b
Commits on Jan 29, 2025
AUTO ORDER enable and disable functionalities done
sachinsam2024 committed on Jan 29
0f489d9
Commits on Jan 28, 2025
made changes in stock order
sachinsam2024 committed on Jan 28
e1e228d
order history page done and made changes in the style in dashboard, order history and stock overview pages
sachinsam2024 committed on Jan 28
67f272e
stock order done
sachinsam2024 committed on Jan 28
1456843
stock overview done
sachinsam2024 committed on Jan 28
5e40ac0


added stock management functionality in DOM module.Stock overview and stock alert done (some funtionalites yet to be finished)
sachinsam2024 committed on Jan 24
5abf799
Commits on Jan 21, 2025
Added Supplier module
sachinsam2024 committed on Jan 21
6a151f8
Added Reciepe Generator and Add supplier and manage supplier in District Operation manager page
sachinsam2024 committed on Jan 21
beb2f6e
Commits on Nov 12, 2024
ADDED SCRUM SCHEET IN DOCUMENTATION
sachinsam2024 committed on Nov 12, 2024
a675aaa
ADDED MINI PROJECT REPORT+GIT LOG AND PPT
sachinsam2024 committed on Nov 12, 2024
67b05ff
Commits on Nov 8, 2024
made changes in package.json(DOM) for hosting purpose
sachinsam2024 committed on Nov 8, 2024
9b16ffb
made change in admin login page
sachinsam2024 committed on Nov 8, 2024
0e1a9c8
made changes in package.json(admin) for hosting purpose
sachinsam2024 committed on Nov 8, 2024
e310a79
made changes in package.json ,final project report,editproduct and add product in DOM
sachinsam2024 committed on Nov 8, 2024
b6f2760
added render.yaml file
sachinsam2024 committed on Nov 5, 2024
021903b
Made changes to password bcrypt in forgot password,chnaged the styles of signup,login validation from homepage,pincode check in location header
sachinsam2024 committed on Nov 5, 2024
c205caf
edited package.json(server) for hosting
sachinsam2024 committed on Nov 4, 2024
fc481be
render.yaml added in server
sachinsam2024 committed on Nov 4, 2024
6c1b556
made changes in the session
sachinsam2024 committed on Nov 4, 2024
9bcc547
Commits on Oct 27, 2024
added district operation manager management to admin,added a preloader,made new sidebar for search result
sachinsam2024 committed on Oct 27, 2024
f3d8b13
added new user stock manager
sachinsam2024 committed on Oct 27, 2024
86b132d
Commits on Oct 18, 2024
made changes in addtocart message display
sachinsam2024 committed on Oct 18, 2024
b29249d
added review and cancel order functionality(completed)
sachinsam2024 committed on Oct 18, 2024
96f063b


made changes in the product listt in admin.added countinstock and location
sachinsam2024 committed on Oct 16, 2024
1a5aa04
added districts.js in admin and client,changed the location from countries to states(admin and client)
sachinsam2024 committed on Oct 16, 2024
f971e96
added a link to go back to homepage from loginpage
sachinsam2024 committed on Oct 16, 2024
9bfa58a
removed add review from productDetails page
sachinsam2024 committed on Oct 16, 2024
4deebb9
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
sachinsam2024 committed on Oct 16, 2024
9b919db
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)
sachinsam2024 committed on Oct 16, 2024
51ab2ea
added env in gitignore(admin)
sachinsam2024 committed on Oct 15, 2024
5c57525
added env in gitignore(server)
sachinsam2024 committed on Oct 15, 2024
648bd07
added env in gitignore
sachinsam2024 committed on Oct 15, 2024
97eb1db
added some styles in signup and changes in redirect to admin from user login
sachinsam2024 committed on Oct 14, 2024
a5c0c65
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
sachinsam2024 committed on Oct 14, 2024
b85bd39
added download invoice option in orders
sachinsam2024 committed on Oct 14, 2024
680b07f
listing of products based on category,filter by price and rating error has been fixed
sachinsam2024 committed on Oct 14, 2024
c81aac1
made changes in checkout (added upi)
sachinsam2024 committed on Oct 14, 2024
aed4ba4
added invoice component in App.js
sachinsam2024 committed on Oct 14, 2024
233b2ee
html2canvas added for reciept printing
sachinsam2024 committed on Oct 14, 2024
4f8f0cf
made the same delete to out of stock in products list(admin)
sachinsam2024 committed on Oct 11, 2024
3e27f3c
made changes in delete a product in admin(delete=Out of stock) and also fixed the count of review
sachinsam2024 committed on Oct 11, 2024
8800dc9
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),
sachinsam2024 committed on Oct 8, 2024
11196ff
changed api.js(res->data),added new attribute in cart model
sachinsam2024 committed on Oct 8, 2024
3c948cf
added weight to Productdetails and cart
sachinsam2024 committed on Oct 8, 2024
e6b5c29
made changes in cart and productitem page (Completed)
sachinsam2024 committed on Oct 8, 2024
f56aee3


PRODUCT REMOVED FROM CART ADDED TO WISHLIST(COMPLETED)
sachinsam2024 committed on Oct 4, 2024
36dba95
List user and Block user functionalities completed(ADMIN SIDE)
sachinsam2024 committed on Oct 4, 2024
78c8ca3
Added Functionality of blocking and unblocking user by admin
sachinsam2024 committed on Oct 4, 2024
a6cbb27
made changes in the Signin and users model(Blocked Users)
sachinsam2024 committed on Oct 4, 2024
cb3977f
Commits on Sep 24, 2024
list user in manage user in admin (DONE)
sachinsam2024 committed on Sep 24, 2024
05b982d
added abstract to documentation
sachinsam2024 committed on Sep 24, 2024
4f7e40b
Made changes in manage user in admin"(code working)
sachinsam2024 committed on Sep 24, 2024
d545c65
made chages to manage user in admin dashboard(not finished)
sachinsam2024 committed on Sep 24, 2024
08e5bef
Commits on Sep 23, 2024
Added Documentation and added Manage user functionality to admin(only dropdown list.code need to be done)
sachinsam2024 committed on Sep 23, 2024
9edb04d
Made changes in Login and Signup of Admin
sachinsam2024 committed on Sep 23, 2024
f656725
Made changes to Additional Info(ProductDetails)
sachinsam2024 committed on Sep 23, 2024
dd9982f
made changes in Header
sachinsam2024 committed on Sep 22, 2024
1fbb4c2
Commits on Sep 18, 2024
Second Commit(Deleted Images)
sachinsam2024 committed on Sep 18, 2024
5629e53
First Commit
sachinsam2024 committed on Sep 18, 2024
3e73bbb