**Project Title**

**ShopSmart: Your Digital Grocery Store Experience**

**Team Members:**

Here List team members and their roles:

1. **A Ratna Varsha** (Full Stack Developer): Combines both frontend and backend responsibilities, ensuring smooth communication between the two. This role also handles bug fixing, feature integration, and overall system performance.

**2. P S S Manikanta** (Frontend Developer): Responsible for designing the user interface using React.js. This role focuses on ensuring a responsive, user-friendly design, as well as integrating the frontend with backend APIs.

**3. CH Nikhitha** (Backend Developer): Develops the backend server using Node.js and Express.js, ensuring the creation of secure, scalable RESTful APIs, as well as handling authentication, data processing,and business logic.

**4. V Sagar (**Database Administration): Manages the MongoDB

database, focusing on schema design, data integrity, and database optimization to ensure efficient data storage and retrieval.

**Introduction:**

Tired of juggling long grocery lists, comparing prices, and managing your shopping trips? Introducing ShopSmart, a cutting-edge grocery store web application built using the MERN stack (MongoDB, Express.js, React.js, and Node.js). Designed to simplify and enhance the grocery shopping experience, ShopSmart offers seamless navigation, personalized features, and real-time updates.

**Project Overview**

ShopSmart is a modern grocery store web application designed to revolutionize the way users shop for groceries.It combines convenience, efficiency, and personalization to create a seamless shopping experience.We prioritize your privacy and security. Our app utilizes encryption to protect your data, ensuring that all communications remain confidential and shielded from unauthorized access.

ShopSmart aims to simplify grocery shopping by addressing common pain points such as managing lists, finding deals, and saving time. By integrating real-time updates and a user-friendly interface, it offers a smarter way to shop for essentials

**Goals of Project:**

* Allow users to browse and buy groceries online easily.
* Provide a simple and user-friendly shopping experience.
* Let users add items to a cart and place orders smoothly.
* Ensure secure user authentication (login/signup).
* Store product details in a database for easy management.
* Make the app fast and responsive for all users..

**Features:**

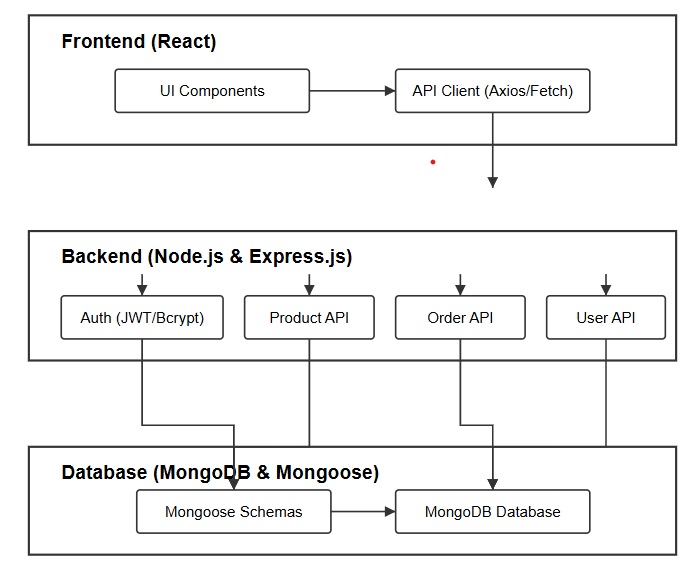
1. **Product Browsing:** Users can explore a variety of grocery items with categories for easy navigation.
2. **Cart Management:** Users can add/remove items from the cart and view the total price before checkout.
3. **User Authentication:** Secure login and signup system for personalized shopping experiences.
4. **Order Placement:** Customers can place orders and view order details.
5. **Simple Admin Panel:** Allows easy management of products, including adding, updating, and removing items.

**Technical Architecture:**

he technical architecture of our Grocery Store Web App follows a client-server model, utilizing the MERN stack along with Firebase authentication and JWT for secure user management. The frontend, built using React, serves as the user interface and communicates with the backend through a REST API. Users can browse grocery items, add them to their cart, and place orders seamlessly. Authentication is handled via Firebase and JWT, ensuring secure login and access control.

The backend, powered by Node.js and Express.js, manages API requests, processes orders, and handles user authentication. Secure user sessions are maintained using JWT, while Firebase enhances authentication reliability

**Architecture**



The backend interacts with MongoDB using Mongoose, efficiently storing product details, user accounts, and order information. The app also integrates real-time inventory management to prevent stock issues and enhance user experience. Additionally, a simple admin panel allows store owners to manage products and track orders effortlessly. Together, the frontend, backend, authentication system, and database form a robust technical architecture that ensures smooth grocery shopping, secure transactions, and efficient order management. information, is stored and retrieved from a database such as MongoDB, ensuring efficient storage and retrieval of data.

Together, the frontend, backend, REST API, socket.io, Express.js, and database (e.g., MongoDB) form a comprehensive technical architecture for our social media app. This architecture enables real-time messaging, seamless post uploading, authentication, and secure data storage, providing users with a dynamic and interactive social media experience.

**PRE-REQUISITES**:

Here are the key prerequisites for developing a full-stack application using Node.js, Express.js, MongoDB, React.js, Socket.io:

* **Node.js and npm**:

Node.js is a powerful JavaScript runtime environment that allows you to run JavaScript code on the server-side. It provides a scalable and efficient platform for building network applications.

Install Node.js and npm on your development machine, as they are required to run JavaScript on the server-side.

* Download: <https://nodejs.org/en/download/>
* Installation instructions: <https://nodejs.org/en/download/package-manager/>
* **Express.js**:

Express.js is a fast and minimalist web application framework for Node.js. It simplifies the process of creating robust APIs and web applications, offering features like routing, middleware support, and modular architecture. Install Express.js, a web application framework for Node.js, which handles server-side routing, middleware, and API development.

Installation: Open your command prompt or terminal and run the following command:

**npm install express**

* **MongoDB**:

MongoDB is a flexible and scalable NoSQL database that stores data in a JSON-like format. It provides high performance, horizontal scalability, and seamless integration with Node.js, making it ideal for handling large amounts of structured and unstructured data.

Set up a MongoDB database to store your application's data.

* Download: <https://www.mongodb.com/try/download/community>
* Installation instructions: <https://docs.mongodb.com/manual/installation/>
* **React.js**:

React.js is a popular JavaScript library for building user interfaces. It enables developers to create interactive and reusable UI components, making it easier to build dynamic and responsive web applications.

Install React.js, a JavaScript library for building user interfaces.

Follow the installation guide: <https://reactjs.org/docs/create-a-new-react-app.html>

* **HTML, CSS, and JavaScript**: Basic knowledge of HTML for creating the structure of your app, CSS for styling, and JavaScript for client-side interactivity is essential.
* **Database Connectivity**: Use a MongoDB driver or an Object-Document Mapping (ODM) library like Mongoose to connect your Node.js server with the MongoDB database and perform CRUD (Create, Read, Update, Delete) operations. To Connect the Database with Node JS go through the below provided link:

• [https://www.section.io/engineering-education/nodejs- mongoosejs-mongodb/](https://www.section.io/engineering-education/nodejs-%20mongoosejs-mongodb/)

* **Version Control**: Use Git for version control, enabling collaboration and tracking changes throughout the development process. Platforms like GitHub or Bitbucket can host your repository.

• Git: Download and installation instructions can be found at: <https://git-scm.com/downloads>

* **Development Environment**: Choose a code editor or Integrated Development Environment (IDE) that suits your preferences, such as Visual Studio Code.

• Visual Studio Code: Download from <https://code.visualstudio.com/download>

To run the ShopSmart app download the code from Github

Firstly, download the code from the Github link provided below

<https://github.com/Ratnavarsha/ShopSmart>

**Install Dependencies:**

• Navigate into the cloned repository directory:

**cd ShopSmart**

• Install the required dependencies by running the following commands:

**cd frontend**

**npm install**

**cd ../backend**

**npm install**

* **Start the Development Server**:

• To start the development server, execute the following command:

**npm run dev**

• The ShopSmart app will be accessible at http://localhost:5173

* **Access the App:**

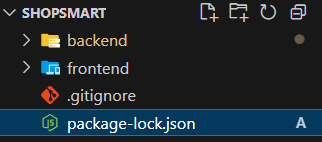
• Open your web browser and navigate to http://localhost:5173

• You should see the web site

You have successfully installed and set up the ShopSmart app on your local machine. You can now proceed with further customization, development, and testing as needed.

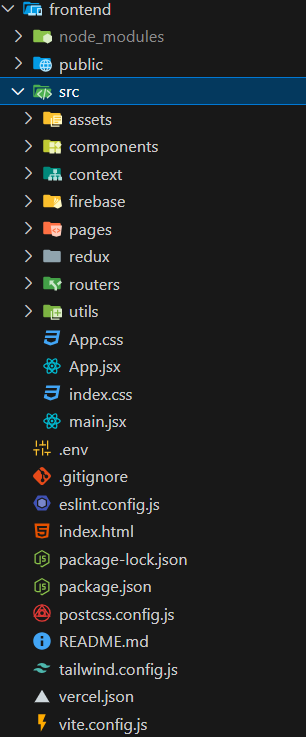
**Project structure:**

* Inside the ShopSmart directory, we have the following folders



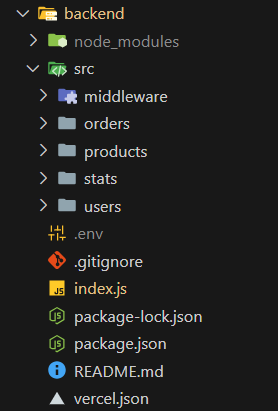
* **Frontend directory:**

The below directory structure represents the directories and files in the client folder (front end) where, react Js is used along with Api’s such as socket.io.



* **Backend directory:**

The below directory structure represents the directories and files in the server folder (back end) where, node js, express js and mongodb are used along with socket.io.

****

**Project Flow:**

**Milestone 1: Project setup and configuration.**

* **Folder setup**:

## Now, firstly create the folders for frontend and backend to write the respective code and install the essential libraries.

* + Frontend folders.
  + Backend folders
* **Installation of required tools**:

Open the frontend folder to install necessary tools. For frontend (client), we use:

|  |  |
| --- | --- |
| Tools/libraries | Installation command |
| React Js | npx create-react-app . |
| Router-dom | npm install react-router-dom |
| Bootstrap | npm install bootstrap |
| Axios | npm install axios |
| Firebase | npm install firebase |
| swiper | npm install swiper |

Open the backend folder to install necessary tools. For backend (server), we use:

|  |  |
| --- | --- |
| Tools/libraries | Installation command |
| Express Js | npm install express |
| Mongoose | npm install mongoose |
| Bcrypt | npm install bcrypt |
| Body-parser | npm install body-parser |
| Cors | npm install cors |
| Dotenv | npm install dotenv |
| nodemon | npm install nodemon |
| JWT | npm install jwt |

**Milestone 2: Backend development**

#### **Set Up Project Structure:**

* Create a new directory for your project and set up a package.json file using npm init command.
* Install necessary dependencies such as Express.js, Mongoose, and other required packages.

#### **Set Up Project Structure:**

* + Create a new directory for your project and set up a package.json file using npm init command.
  + Install necessary dependencies such as Express.js, Mongoose, and other required packages.

#### **Create Express.js Server:**

* + Set up an Express.js server and serve API endpoints.
  + Configure middleware such as body-parser for parsing request bodies and cors for handling cross-origin requests.

#### **Define API Routes:**

* + Create separate route files for different API functionalities such as authentication, create post, upload stories, chats, etc.,
  + Implement route handlers using Express.js to handle requests and interact with the database.

#### **Implement Data Models:**

* + Define Mongoose schemas for the different data entities like posts, chats, users, etc.,
  + Create corresponding Mongoose models to interact with the MongoDB database.
  + Implement CRUD operations (Create, Read, Update, Delete) for each model to perform database operations.

#### **User Authentication:**

* + Implement user authentication using strategies like JSON Web Tokens (JWT) or session-based authentication.
  + Create routes and middleware for user registration, login, and logout.
  + Set up authentication middleware to protect routes that require user authentication.

#### **Handle new chats and posts:**

* + Allow users to chat with other users using the userId.
  + Also the users will make the posts on social timeline.

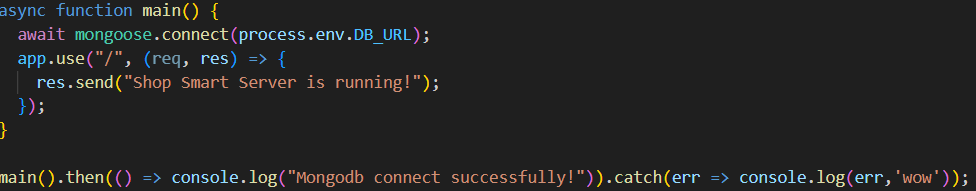
#### **Error Handling:**

* + Implement error handling middleware to catch and handle any errors that occur during the API requests.
  + Return appropriate error responses with relevant error messages and HTTP status codes.

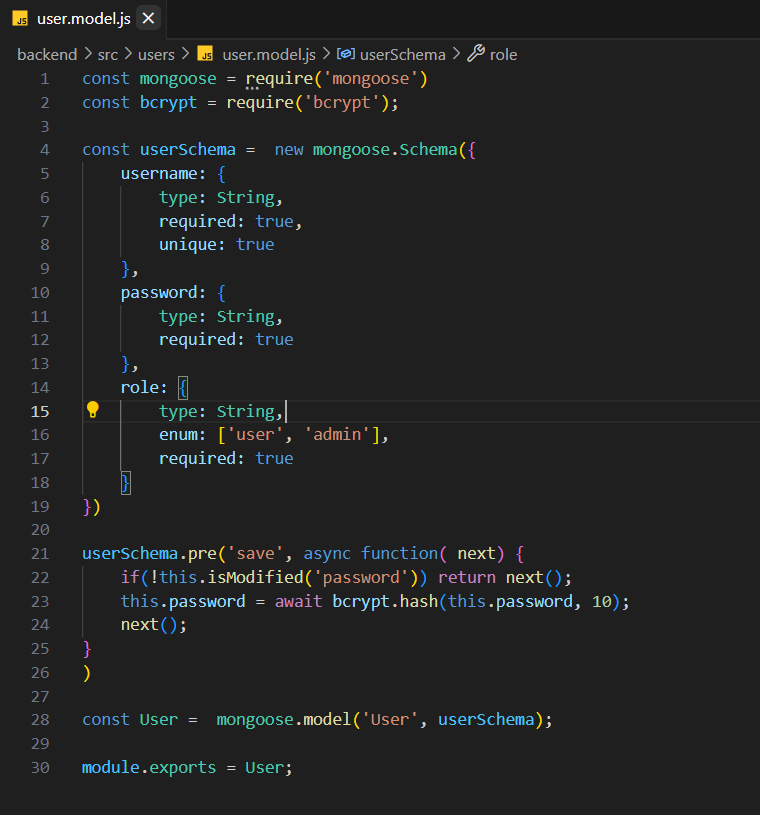
**Milestone 3: Database development:**

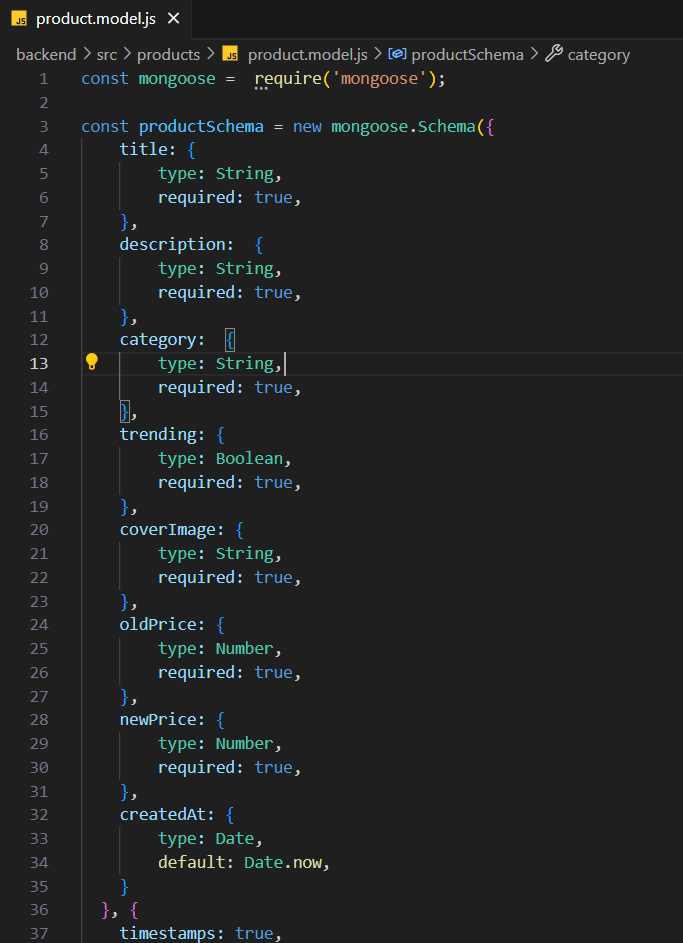
* Set up a MongoDB database either locally or using a cloud-based MongoDB service like MongoDB Atlas.
* Create a database and define the necessary collections for users, posts, chats, etc.,

The code for the database connection to the server is



**The code for the schemas in the database is provided below**





****

**Milestone 4: Frontend Development & Integration**

**1. Establish Real-Time Cart & Order Updates**

After successful authentication using Firebase and JWT, establish a WebSocket (Socket.io) connection between the client and the server.

Use Socket.io to ensure seamless updates when users add or remove items from the cart.

Implement real-time order status tracking, where users receive instant notifications about order confirmations, dispatch, and delivery updates.

**2. Add Product Management & Upload Feature**

Allow admin users to add new grocery items (product name, price, category, and images).

Upload product images to Firebase Storage and store the file links in MongoDB.

Retrieve and display the product listings dynamically for users to browse.

**3. Implement User Profile & Order History**

Add a profile page for users where they can view and update their details.

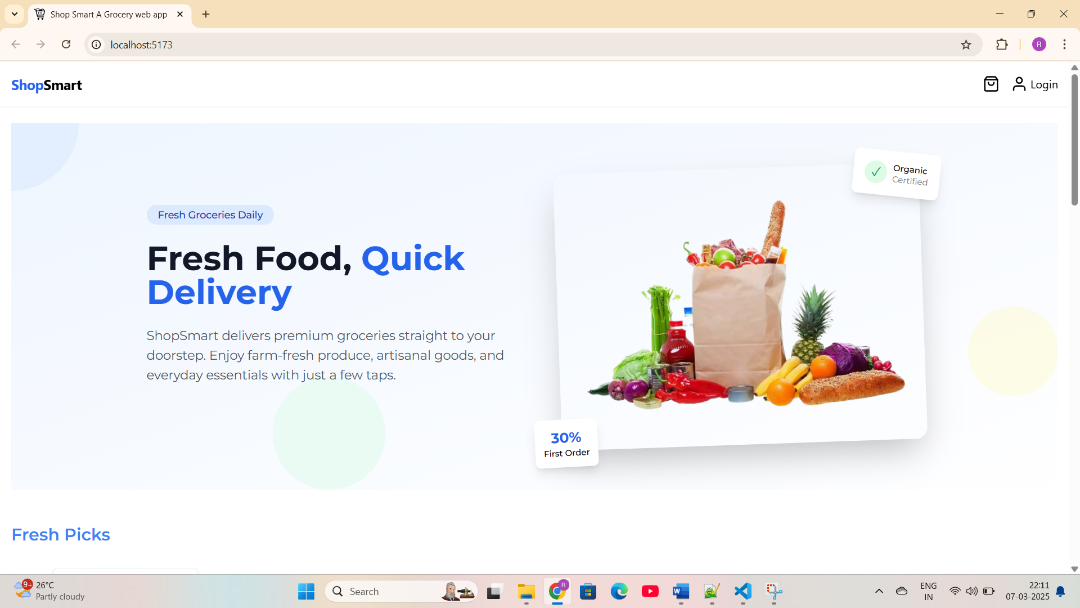
Display user order history, including past and ongoing orders.

Allow users to update details like profile picture, address, and phone number.

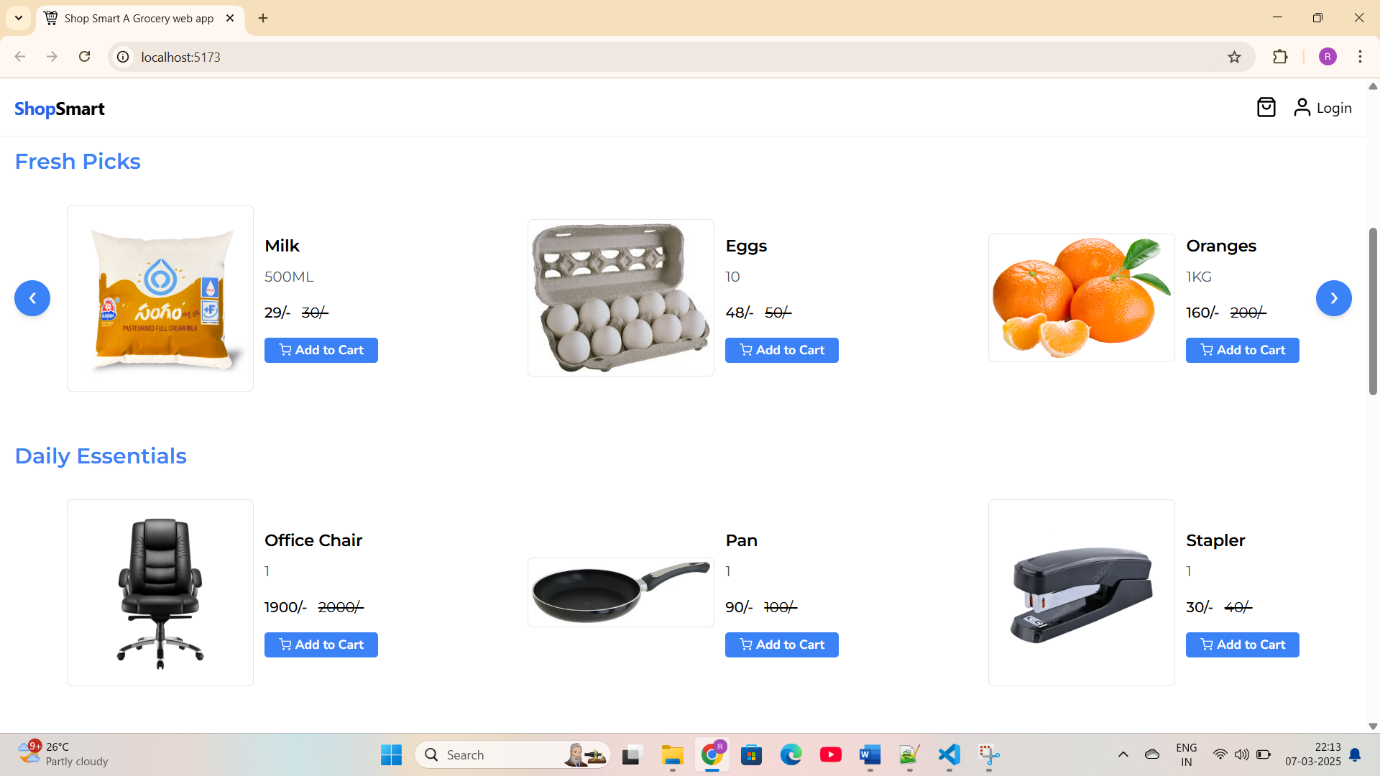
**Milestone 5: Project Implementation:**

On completing the development part, we then run the application one last time to verify all the functionalities and look for any bugs in it. The user interface of the application looks a bit like the one’s provided below.

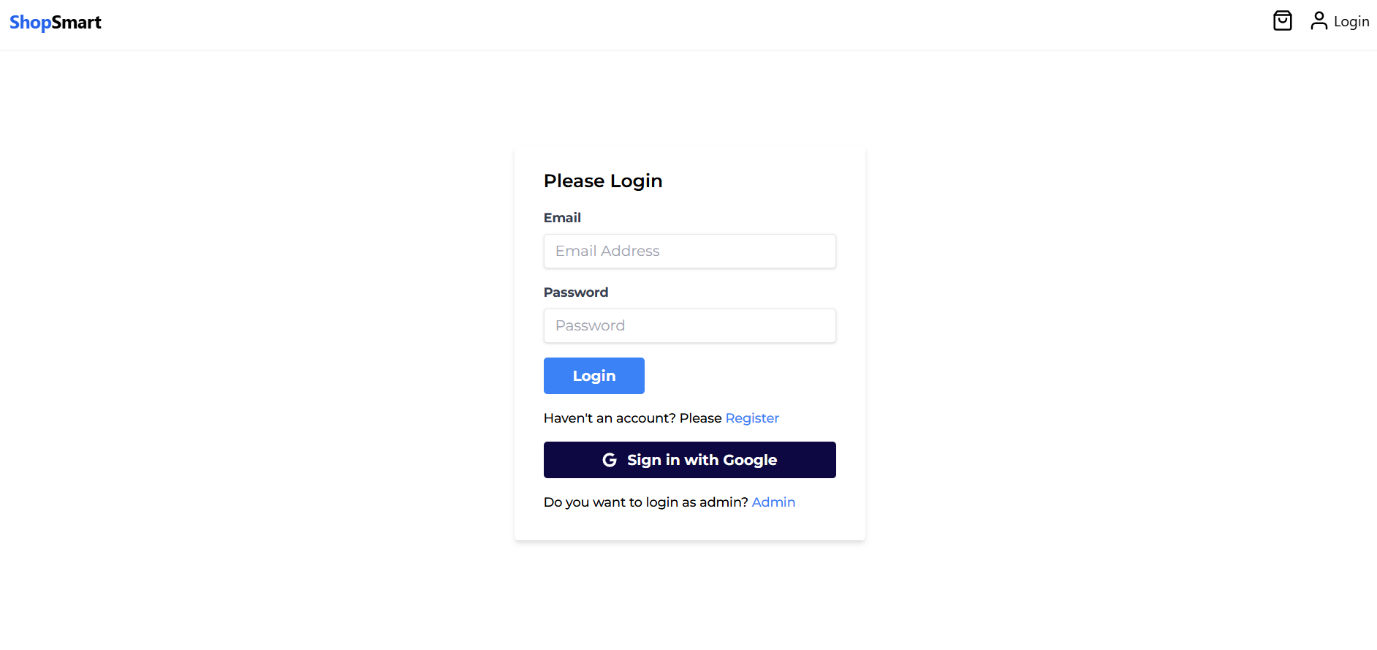
* **Landing page**

****

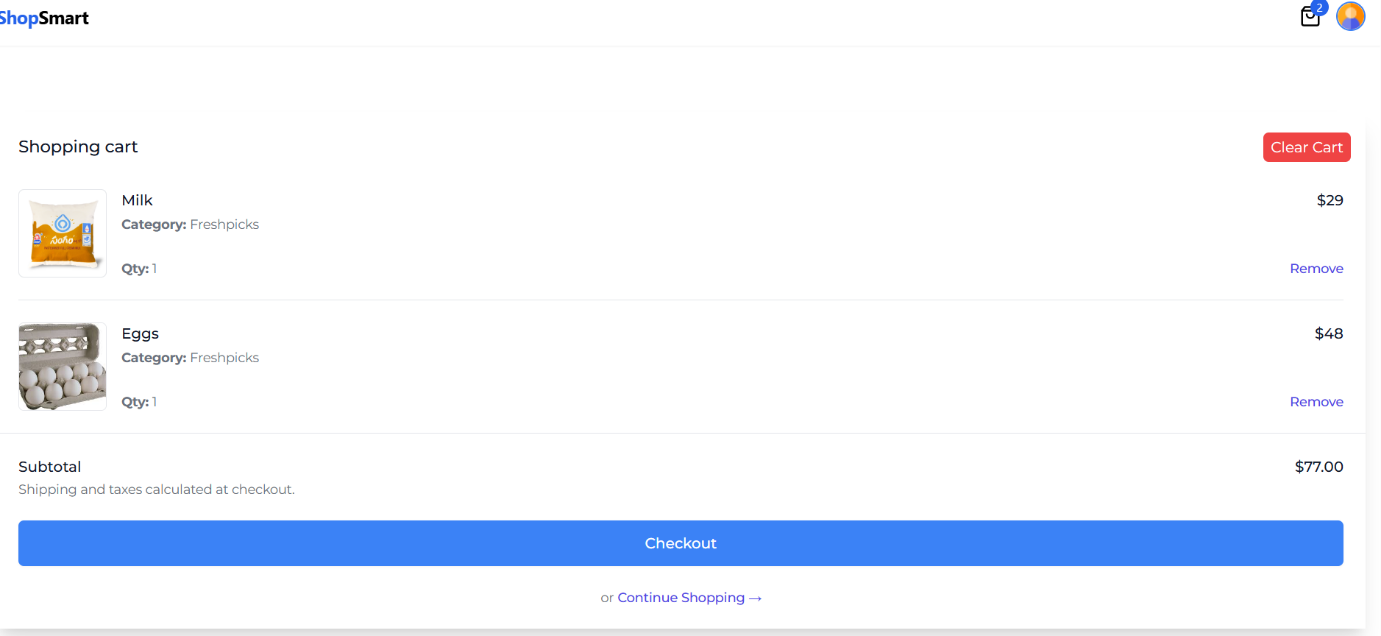
**Home page**



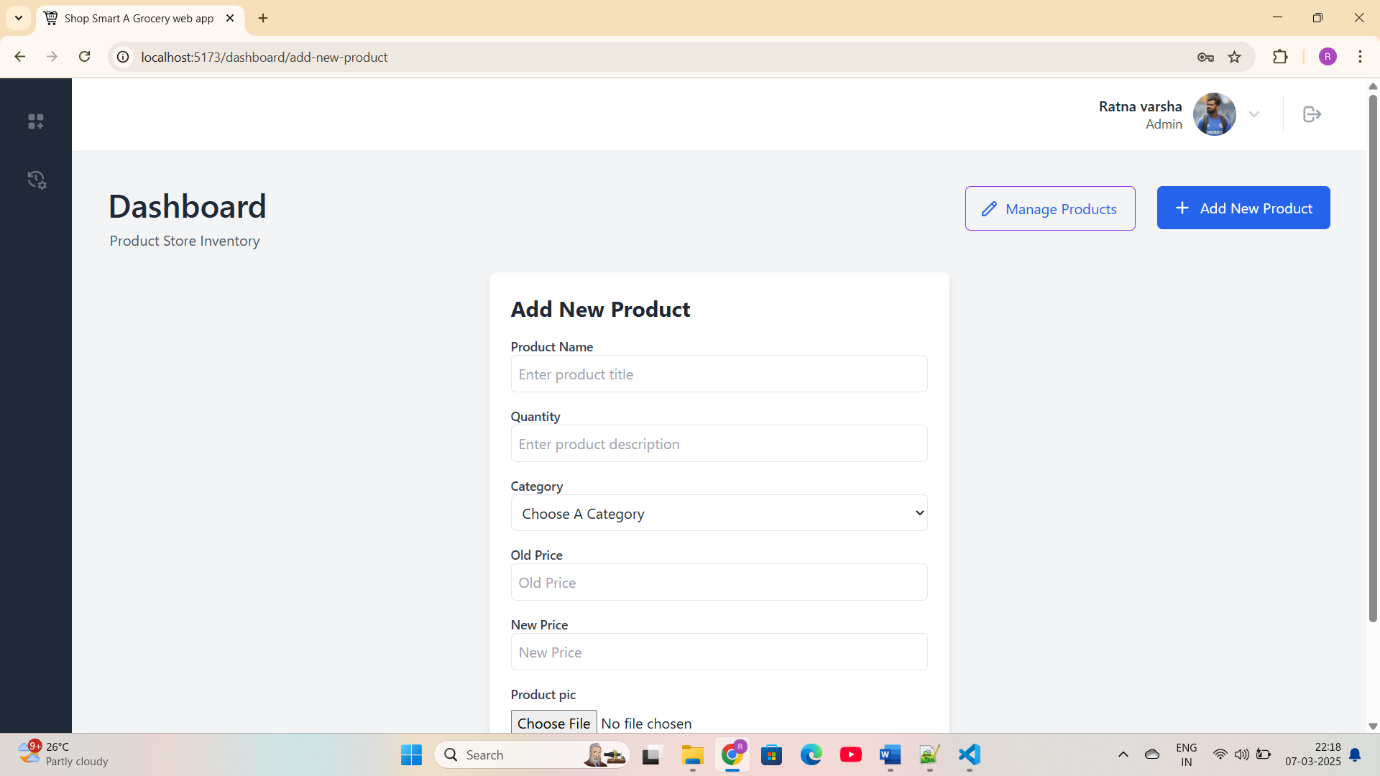
**Login Page**



**Cart page**



**Admin profile**



**Testing:**

**1. Unit Testing:**

* **Description:** Focuses on testing individual components or functions in isolation to ensure they work as intended.
* **Example:** Testing a React component to verify that it renders correctly based on props.

**2.Integration Testing:**

* **Description:** Tests how different modules or components work together, ensuring that integrated parts of the application function correctly**.**
* **Example:** Testing an API endpoint to verify that it correctly interacts with the database and returns the expected response.

**Known Issues:**

1. **Session Management**: Users may experience session timeouts or loss of state when navigating between pages or after refreshing.
2. **Performance Lag**: The application can slow down with a large number of posts or user interactions, particularly due to unoptimized database queries.
3. **Error Handling**: Generic error messages for failed API calls can confuse users; more specific messages are needed.

**Future Enhancements:**

* **Enhanced User Profiles**: Add more customizable profile options, including themes, cover photos, and bio sections.
* **Real-Time Notifications**: Implement real-time notifications for updates

**\*\*\* Happy coding!! \*\*\***