Full Stack Development with MERN Project Documentation format

1. Introduction

• **Project Title:** shopmart

• **Team Members:** List team members and their roles.

2. Project Overview

• **Purpose:** Briefly describe the purpose and goals of the project. • **Features:** Highlight key features and functionalities.

3. Architecture

• **Frontend:** Describe the frontend architecture using React.

• **Backend:** Outline the backend architecture using Node.js and Express.js. • **Database:** Detail the database schema and interactions with MongoDB.

4. Setup Instructions

• **Prerequisites:** List software dependencies (e.g., Node.js, MongoDB). • **Installation:** Step-by-step guide to clone, install dependencies, and set up the environment variables.

5. Folder Structure

• **Client:** Describe the structure of the React frontend.

• **Server:** Explain the organization of the Node.js backend.

6. Running the Application

• Provide commands to start the frontend and backend servers locally.

o **Frontend:** npm start in the client directory.

o **Backend:** npm start in the server directory.

7. API Documentation

• Document all endpoints exposed by the backend.

• Include request methods, parameters, and example responses.

8. Authentication

• Explain how authentication and authorization are handled in the project. • Include details about tokens, sessions, or any other methods used.

9. User Interface

• Provide screenshots or GIFs showcasing different UI features.

10. Testing

• Describe the testing strategy and tools used.

11. Screenshots or Demo

• Provide screenshots or a link to a demo to showcase the application.

12. Known Issues

• Document any known bugs or issues that users or developers should be aware of.

13. Future Enhancements

• Outline potential future features or improvements that could be made to the project.

ShopSmart: Your Digital Grocery Store Experience

1. Introduction

Project Title: ShopSmart: Your Digital Grocery Store Experience

Team Leader : Bokka chinna sita ramudu

Team member : Kurupudi Rupa mallika

Team member : Gafoor Ayesha Shaik

Team member : avinash jogi

2. Project Overview :

Purpose of the Project

The purpose of the **ShopSmart** project is to design and develop a full-stack, role-based e-commerce web application that enables users to browse, search, and purchase grocery and retail products online, while allowing administrators to efficiently manage products, orders, and users through a secure dashboard.

This project aims to simulate a real-world online shopping platform similar to modern e-commerce systems. It focuses on building a scalable, secure, and user-friendly application using the MERN stack (MongoDB, Express.js, React.js, Node.js).

Goals of the Project

The major goals of the project are:

1. **To provide a seamless online shopping experience**
Users can explore products, filter by category, search by name, add items to cart, and complete purchases through a simple checkout process.

2. **To implement secure authentication and authorization**
The system supports role-based access control (User and Admin). Only authorized administrators can manage products and orders.

3. **To create an efficient admin management system**
The admin dashboard allows product management, order status updates, user control, and report generation (PDF export).

4. **To ensure real-world application architecture**
The project follows industry-standard practices such as JWT authentication, protected routes, REST APIs, modular backend structure, and structured database design.

5. **To enhance learning of full-stack development**
The project integrates frontend and backend technologies, database management, authentication, API integration, and dynamic UI rendering.

Features

Key Features and Functionalities of ShopSmart

The ShopSmart application provides a wide range of user-side and admin-side features designed to simulate a professional online shopping platform.

◈ User Features

1. **User Registration and Login**
   - Secure registration and login functionality.
   - JWT-based authentication.
   - Role-based redirection after login.
2. **Product Browsing**
   - View all available products.
   - Display product details including name, price, stock, and image.
3. **Search Functionality**
   - Search products by name.
   - Real-time search suggestions.
   - URL-based search query support.
4. **Category Filtering**
   - Filter products by category.
   - Dynamic category list generation from database.
5. **Cart Management**
   - Add products to cart.
   - Manage quantity.
   - Persistent cart through backend storage.
6. **Buy Now Option**
   - Direct checkout for single product purchase.
   - Bypasses cart for faster transactions.
7. **Stock Control Display**
   - Shows available stock.
   - Displays “Out of Stock” status dynamically.
8. **Checkout System**
   - Order placement functionality.
   - Order data stored in database with status tracking.

◈ Admin Features

1. **Admin Authentication**
   - Role-based secure login.

- Protected admin routes.

2. **Product Management**
   - Add new products.
   - Edit existing products.
   - Delete products.
   - View product stock levels.
3. **Order Management**
   - View all user orders.
   - Update order status (Placed, Shipped, Delivered).
   - Lock status after delivery.
4. **Report Generation**
   - Export all orders as PDF.
   - Downloadable report for record keeping.
5. **User Control (if implemented)**
   - Block/Unblock users.
   - Monitor user activity.

❖ Technical Features

1. RESTful API architecture
2. MongoDB database integration
3. Secure JWT token authentication
4. Role-based access control
5. Dynamic UI rendering with React
6. Modular backend structure
7. Clean separation of frontend and backend

3. Architecture

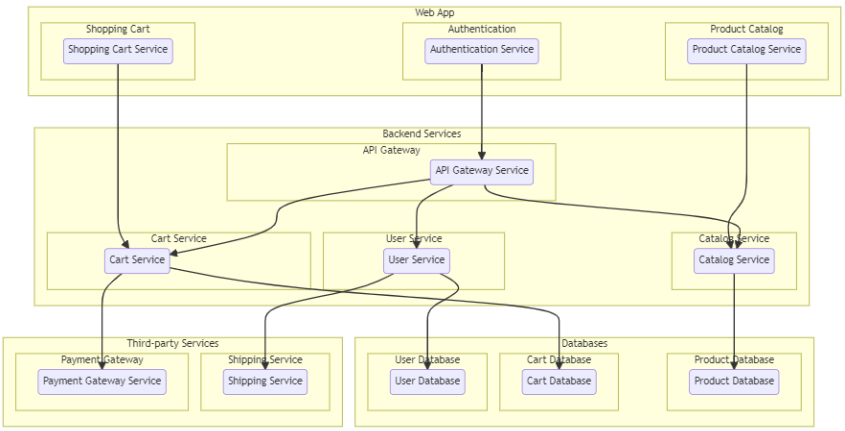

Fig 1 :- Main Architecture

I. Web App (Frontend Layer)

This is the user-facing interface where customers interact with the platform.

Shopping Cart: A UI component that tracks selected items and initiates checkout.

Authentication: The entry point for users to securely log in or sign up, often managed by a dedicated security service.

Product Catalog: The part of the website where users browse and search for products.

II. Backend Services (Logic Layer)

These services handle the core "behind-the-scenes" operations.

API Gateway: A central entry point that routes client requests to the correct microservice. It also handles "cross-cutting concerns" like security, rate limiting, and request aggregation to simplify the frontend.

User Service: Manages user profiles, account details, and session data.

Cart Service: Processes the logic for adding, removing, or updating items in a cart.

Catalog Service: Handles product data management, including listings, descriptions, and categories.

III. Databases (Data Storage Layer)

This architecture uses a database-per-service pattern to ensure services remain independent and decoupled.

User Database: Stores sensitive account information and credentials.

Cart Database: Usually a high-performance, low-latency store (like Redis) for temporary session data.

Product Database: A storage system (often a NoSQL database like MongoDB) optimized for quick retrieval of product details.

IV. Third-party Services (External Integrations)

The system connects to specialized external providers to avoid building complex logic from scratch.

Payment Gateway: A secure service (e.g., PayPal, Stripe) that processes financial transactions.

Shipping Service: An external logistics provider that manages package tracking and delivery.

4. Setup Instructions

• Prerequisites:

To develop a full-stack Ecommerce App for Furniture Tool using React js, Node.js,Express js and MongoDB, there are several prerequisites you should consider. Here are the key prerequisites for developing such an application:

Node.js and npm: Install Node.js, which includes npm (Node Package Manager), on your development machine. Node.js is required to run JavaScript on the server side.

• Download: https://nodejs.org/en/download/

• Installation instructions:https://nodejs.org/en/download/package-manager/

MongoDB: Set up a MongoDB database to store hotel and booking information. Install MongoDB locally or use a cloud-based MongoDB service.

• Download: https://www.mongodb.com/try/download/community

• Installation instructions:https://docs.mongodb.com/manual/installation/

Express.js: Express.js is a web application framework for Node.js. Install Express.js to handle server-side routing, middleware, and API development.

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

command: npm install express

React js: React is a JavaScript library for building client-side applications.

And Creating Single Page Web-Appliaction

Getting Started

Create React App is an officially supported way to create single-page React applications. It offers a modern build setup with no configuration.

Quik Start

npm create vite@latest

cd my-app

npm install

npm run dev

If you've previously installed create-react-app globally via npm install -g create-react-app, we recommend you uninstall the package using npm uninstall -g create-react-app or yarn global remove create-react-app to ensure that npx always uses the latest version.

Create a new React project:

• Choose or create a directory where you want to set up your React project.

• Open your terminal or command prompt.

• Navigate to the selected directory using the cd command.

• Create a new React project by running the following command: npx create-react-app your-app-name.Wait for the project to be created:

• This command will generate the basic project structure and install the necessary dependencies

Navigate into the project directory:

• After the project creation is complete, navigate into the project directory by running the following command: cd your-app-name

Start the development server:

• To launch the development server and see your React app in the browser, run the following command: npm run dev

• The npm start will compile your app and start the development server.

• Open your web browser and navigate to https://localhost:5173 to see your React app.

You have successfully set up React on your machine and created a new React project. You can now start building your app by modifying the generated project files in the src directory.

Please note that these instructions provide a basic setup for React. You can explore more ad- vanced configurations and features by referring to the official React documentation: https://react.dev/

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.

Front-end Library: Utilize React to build the user-facing part of the application, including products listings, booking forms, and user interfaces for the admin dashboard.

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, Sublime Text, or WebStorm.

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

• Sublime Text: Download from

https://www.sublimetext.com/download

• WebStorm: Download from https://www.jetbrains.com/webstorm/download

5. Folder Structure

Client:

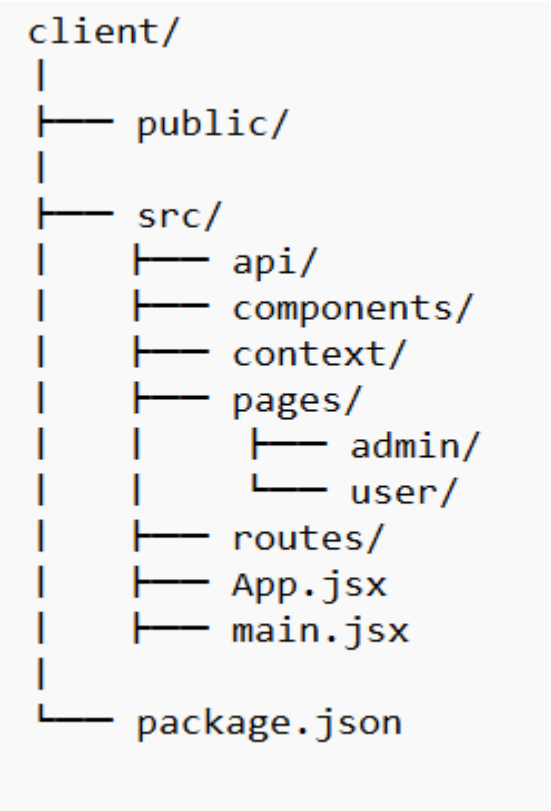

Fig 2 :- Client Structure

Server :

Fig 3 :- Server Structure

◈ Frontend (React Client) – Structure

The frontend of ShopSmart is developed using React.js and follows a component-based architecture. It is organized into structured folders such as pages, components, context, api, and routes. The pages folder contains main screens for users and admins, separated logically into subfolders. Reusable UI elements like Navbar and ProtectedRoute are stored inside the components folder. Authentication state is managed globally using the Context API inside the context folder. API calls are centralized using an Axios instance configured in the api folder. Routing is handled using React Router with role-based protection for admin routes. State management is implemented using React Hooks such as useState and useEffect. Search, filtering, and dynamic rendering are handled efficiently within components. This modular structure ensures scalability, maintainability, and clean separation of concerns.

◈ Backend (Node.js Server) – Structure

The backend of ShopSmart is built using Node.js and Express.js following a modular MVC architecture.
It is divided into folders such as models, routes, controllers, middleware, and config. The models folder defines MongoDB schemas using Mongoose for Users, Products, Orders, and Cart. The routes folder handles API endpoint definitions and maps them to controllers.

Controllers contain the core business logic for authentication, product management, and orders. Middleware is used for JWT authentication and role-based authorization control. The config folder manages database connections and external service configurations. RESTful APIs are used for communication between frontend and backend. Order reports and PDF exports are handled through backend utilities. This organized structure improves code readability, security, and scalability of the application.

6. Running the Application

To run the application successfully without any errors , we need to run the both backend and frontned at the same time , both should be in the running state simulatanously. To run these both , we need the commands , the commands are :

For Backend : First navigate into the folder using the command " cd folder_name(backend)

Next , run the command " npm start ".

For Frontend : First navigate into the folder using the command " cd folder_name(frontend)

Next , run the command " npm run dev ".

7. API Documentation

The ShopSmart backend exposes multiple RESTful API endpoints for authentication, product management, cart operations, order processing, and admin control. All APIs follow standard HTTP methods such as GET, POST, PUT, and DELETE, and responses are returned in JSON format.

🔐 1. Authentication APIs

1.1 Register User

Endpoint:
POST /api/auth/register

Description:
Registers a new user account.

Request Body:

```
{
  "name": "John Doe",
  "email": "john@gmail.com",
  "password": "123456"
}
```

Response:

```
{
  "message": "User registered successfully"
```

}

1.2 Login User

Endpoint:
POST /api/auth/login

Description:
Authenticates user and returns JWT token.

Request Body:

```
{
  "email": "john@gmail.com",
  "password": "123456"
}
```

Response:

```
{
  "token": "jwt_token_here",
  "user": {
    "_id": "12345",
    "name": "John Doe",
    "email": "john@gmail.com",
    "role": "user"
  }
}
```

---

2. Product APIs

2.1 Get All Products

Endpoint:
GET /api/products

Description:
Returns all available products.

Response:

```
[
  {
    "_id": "p1",
    "name": "Rice",
```

```
    "price": 50,
    "stock": 20,
    "category": "Grocery",
    "image": "image_url"
  }
]
```

2.2 Add Product (Admin Only)

Endpoint:
POST /api/products

Authorization: Admin Required

Request Body:


```
{
  "name": "Sugar",
  "price": 40,
  "stock": 30,
  "category": "Grocery",
  "image": "image_url"
}
```


Response:


```
{
  "message": "Product added successfully"
}
```

2.3 Update Product (Admin Only)

Endpoint:
PUT /api/products/:id

Description:
Updates product details.

Response:


```
{
  "message": "Product updated successfully"
}
```

2.4 Delete Product (Admin Only)

Endpoint:
DELETE /api/products/:id

Response:

```
{
  "message": "Product deleted successfully"
}
```

🛒 3. Cart APIs

3.1 Add to Cart

Endpoint:
POST /api/cart/add

Request Body:

```
{
  "productId": "p1",
  "quantity": 2
}
```

Response:

```
{
  "message": "Product added to cart"
}
```

3.2 Get User Cart

Endpoint:
GET /api/cart

Description:
Returns logged-in user's cart items.

📦 4. Order APIs

4.1 Place Order

Endpoint:
POST /api/orders

Description:
Creates a new order for the user.

Response:

{
"message": "Order placed successfully",
"orderId": "order123"
}

4.2 Get User Orders

Endpoint:
GET /api/orders/my

Description:
Returns logged-in user's order history.

4.3 Get All Orders (Admin Only)

Endpoint:
GET /api/orders

Description:
Returns all orders in the system.

4.4 Update Order Status (Admin Only)

Endpoint:
PUT /api/orders/:id

Request Body:

{
"orderStatus": "SHIPPED"
}

Response:

{
"message": "Order status updated"
}

---

5. Admin Report API

5.1 Export Orders as PDF (Admin Only)

Endpoint:
GET /api/admin/orders/export/pdf

Description:
Downloads a PDF report containing all orders.

Response:
Returns a downloadable PDF file.

🔐 Authentication Requirement

Protected endpoints require JWT token in request headers:

Authorization: Bearer <token>

8. Authentication

🔐 Authentication and Authorization in ShopSmart

The ShopSmart application implements secure authentication and role-based authorization using JSON Web Tokens (JWT). The system ensures that only authenticated users can access protected resources, and only administrators can access admin-specific functionalities.

🔹 Authentication Process

Authentication is handled using a token-based mechanism. When a user logs in successfully using valid email and password credentials, the backend verifies the user details stored in the MongoDB database. If the credentials are valid, the server generates a JSON Web Token (JWT) that contains the user's unique ID and role information.

This token is digitally signed using a secret key and sent back to the client. The frontend stores the token securely (typically in local storage) and includes it in future API requests for protected routes.

Example header format:

Authorization: Bearer <JWT_Token>

The backend middleware verifies this token before granting access to secured endpoints.

🔹 JWT (JSON Web Token) Usage

The JWT contains encoded user information such as:

- User ID
- User Role (user/admin)
- Token expiration time

The token ensures:

- Stateless authentication (no session stored on server)
- Secure request validation
- Reduced database calls for identity verification

Each protected route uses an authentication middleware that:

1. Extracts the token from request headers.
2. Verifies it using the secret key.

3. Decodes the payload.
4. Attaches user information to the request object.

If the token is invalid or expired, access is denied.

◈ Authorization (Role-Based Access Control)

Authorization is implemented using role-based access control (RBAC). Each user in the system has a role field, typically:

- user
- admin

After authentication, an additional middleware checks the user role before allowing access to certain endpoints.

For example:

- Only admins can add, edit, or delete products.
- Only admins can update order status.
- Only logged-in users can add items to cart or place orders.

If a non-admin user attempts to access admin routes, the server returns an unauthorized error response.

◈ Session Handling

The project primarily uses stateless authentication through JWT instead of traditional server-side sessions. This improves scalability and performance, as the server does not need to store session data.

In case social login (OAuth) is implemented, temporary sessions may be used during the authentication handshake process, but the final authentication is handled through JWT tokens.

◈ Security Measures Implemented

- Passwords are hashed before storing in the database.
- JWT tokens are signed using a secure secret key.
- Protected routes verify token authenticity.
- Role-based middleware restricts sensitive operations.
- Token expiration ensures automatic session invalidation.

9. User Interface

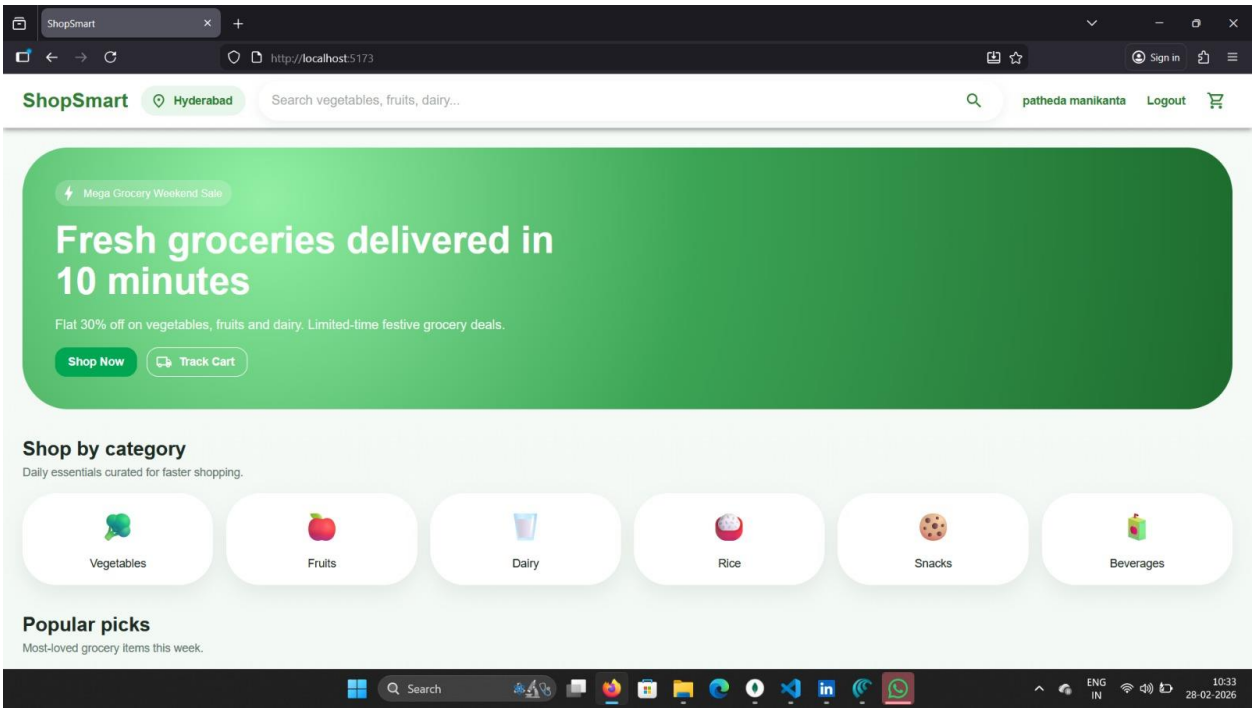

Fig 4 :- Public Home Page

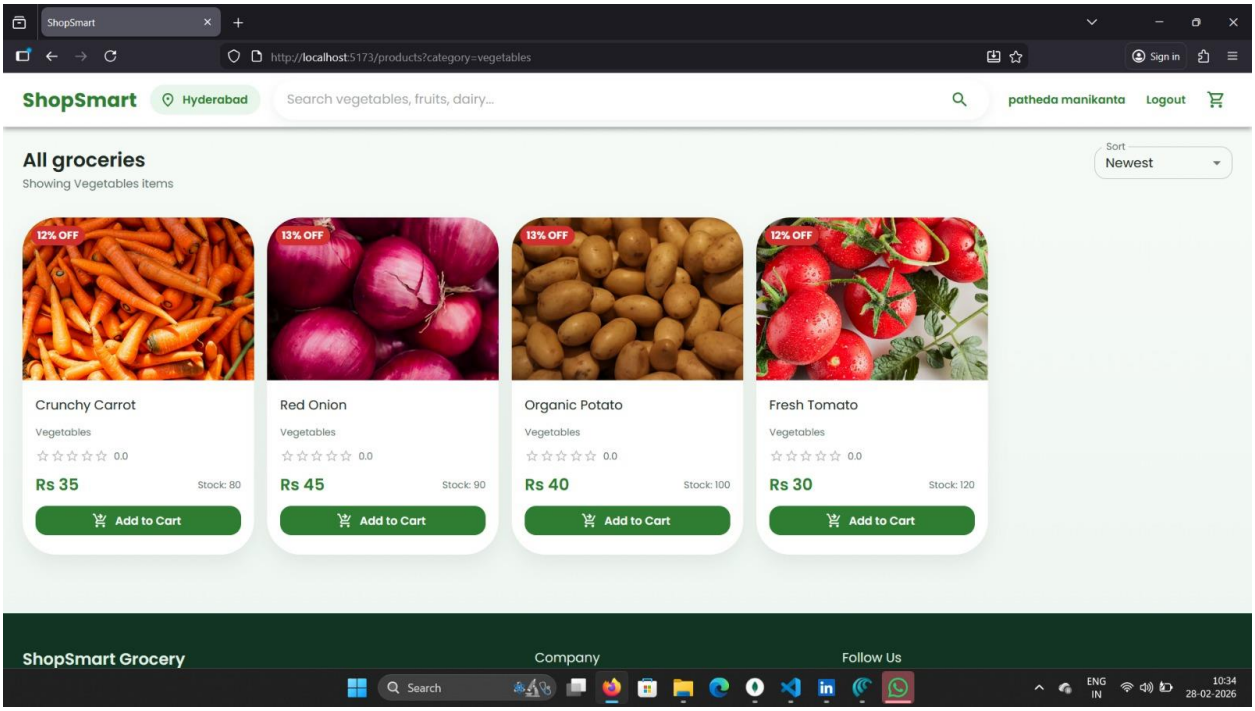

Fig 7 :- User Cart Page

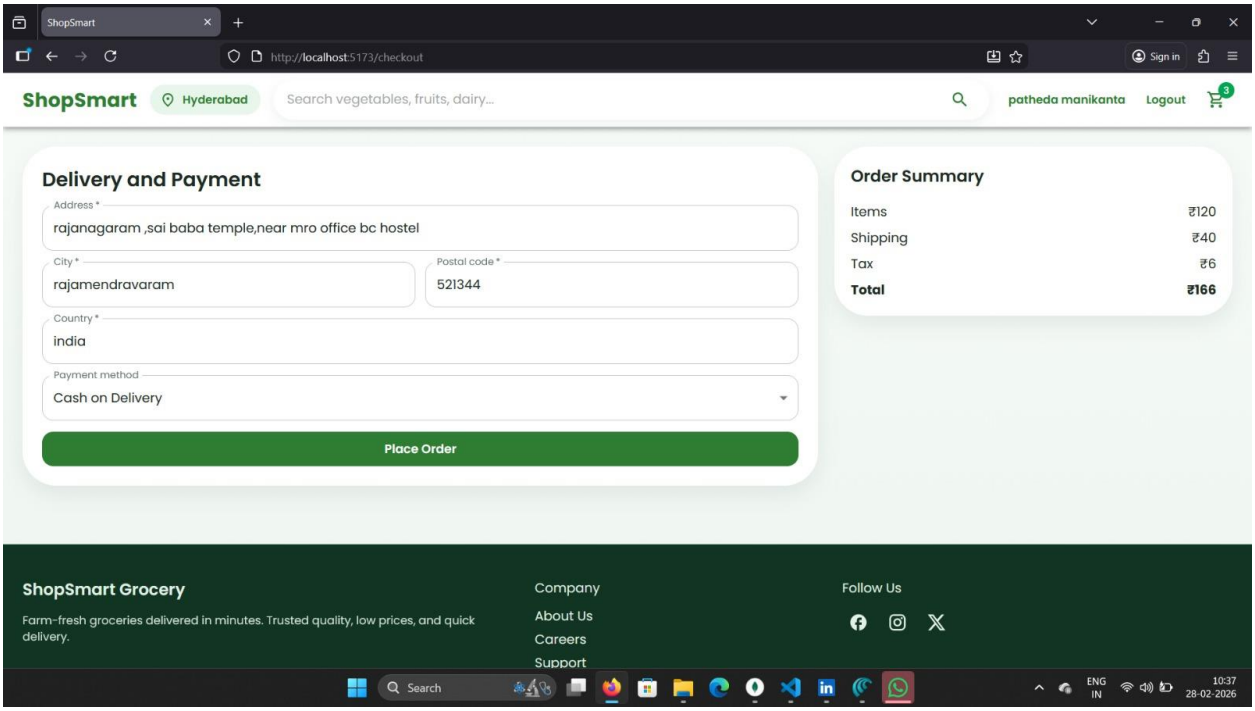

Fig 8 :- User Checkout Page

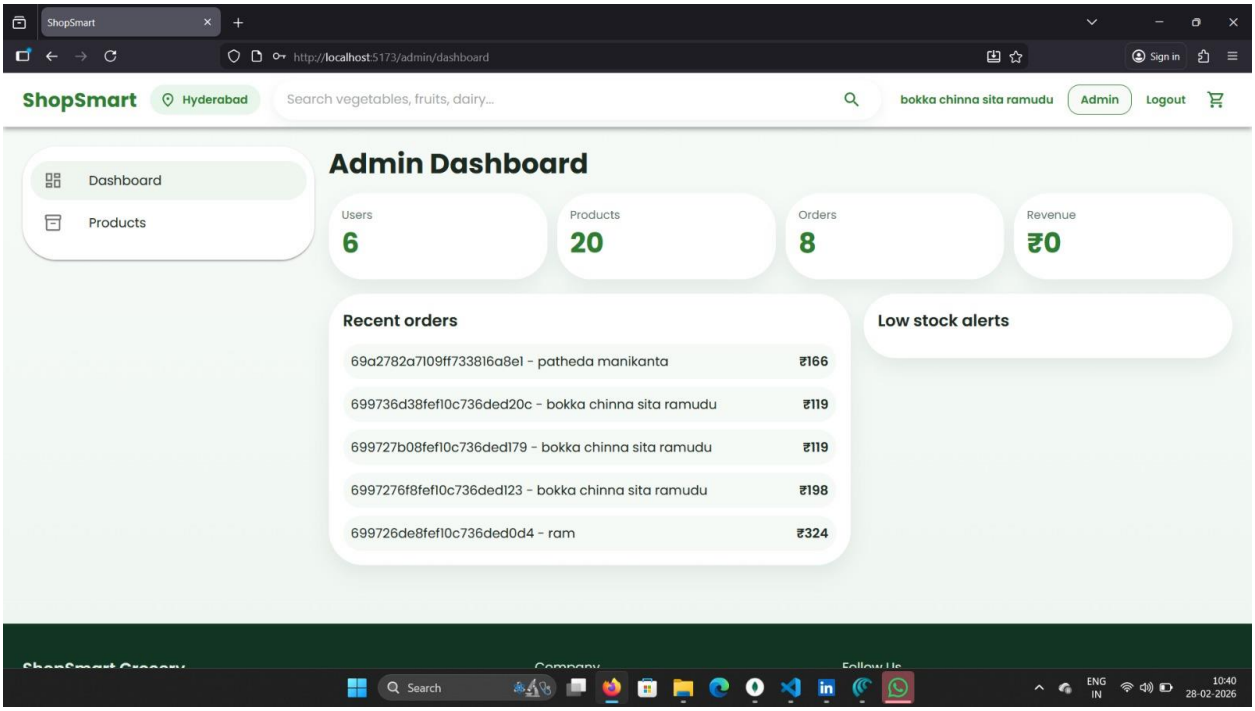

Fig 12 :- Admin Home Page

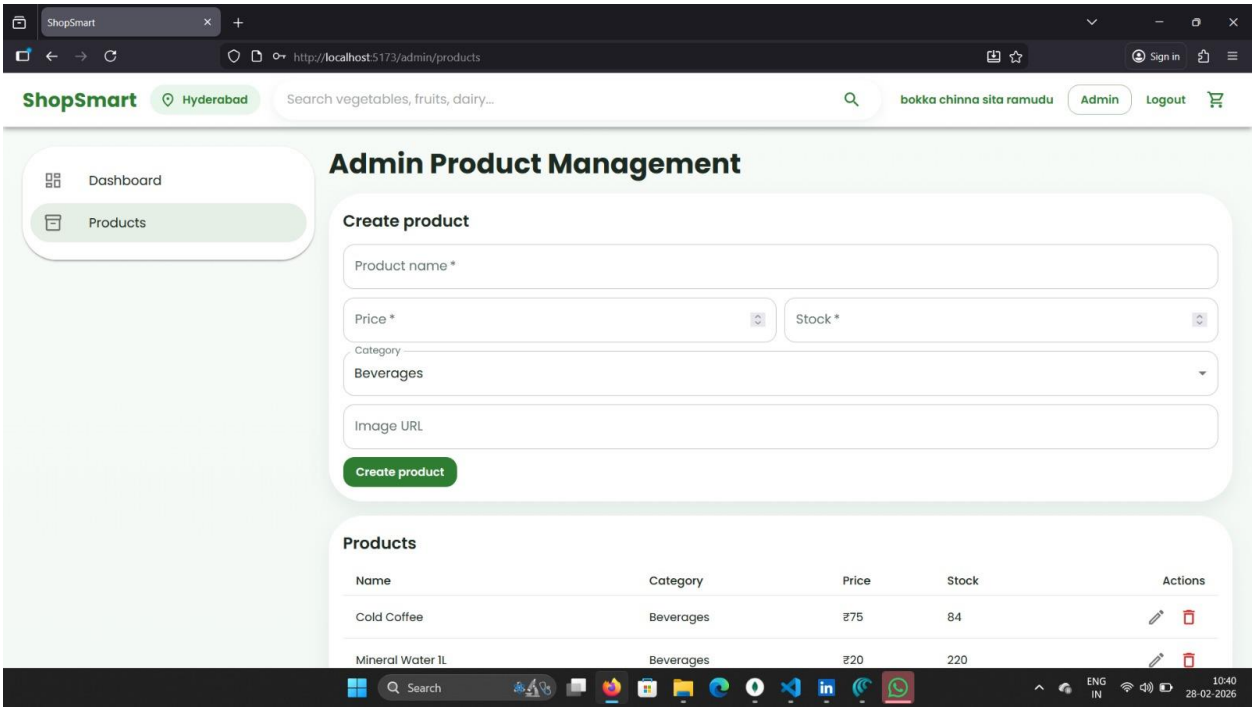

Fig 13 :- Admin Products Page

10. Testing

🔎 Testing Strategy and Tools Used

Testing in the ShopSmart project was performed to ensure that the application functions correctly, securely, and efficiently across both frontend and backend components. The testing strategy included manual testing, functional testing, API testing, and validation testing.

◈ 1. Manual Testing

Manual testing was performed throughout development to verify the correctness of features. Each module, such as authentication, product management, cart operations, and order processing, was tested individually. Test cases were created to check valid inputs, invalid inputs, and edge cases.

Examples:

- Testing login with correct and incorrect credentials.
- Checking product search and category filtering.
- Verifying order status updates by admin.
- Ensuring cart updates correctly when items are added or removed.

◈ 2. Functional Testing

Functional testing was conducted to ensure that all system functionalities behave according to the requirements. Each feature was tested from a user perspective to confirm that expected outputs were generated for given inputs.

Modules tested include:

- User Registration and Login
- Product CRUD operations
- Cart management
- Checkout and order placement
- Admin dashboard controls
- PDF report generation

◈ 3. API Testing

Backend APIs were tested using tools such as:

- Postman (for testing REST API endpoints)
- Browser developer tools (Network tab)

Each API endpoint was tested with:

- Valid request data
- Missing fields
- Unauthorized access attempts
- Invalid tokens

This ensured proper error handling and secure API behavior.

◈ 4. Authentication and Authorization Testing

JWT token verification was tested by:

- Accessing protected routes without a token.
- Using expired or invalid tokens.
- Attempting admin operations with normal user accounts.

The system correctly restricted unauthorized access, confirming proper role-based control.

◈ 5. UI Testing

Frontend UI testing was performed to ensure:

- Proper rendering of components.
- Responsive layout behavior.
- Correct display of product images and stock status.

- Real-time search suggestions.
- Error message display for failed operations.

◈ 6. Error Handling Testing

The system was tested for:

- Server errors
- Database connection failures
- Invalid inputs
- Network interruptions

Proper error messages were displayed to the user without crashing the application.

🔧 Tools Used for Testing

- Postman – for API endpoint testing
- Browser Developer Tools – for debugging frontend
- MongoDB Compass – for verifying database records
- Console Logging – for debugging application flow

12. Known Issues

Although the ShopSmart application is fully functional, there are certain known issues and limitations that users and developers should be aware of.

◈ 1. Limited Payment Integration

Currently, the system does not include integration with real-time payment gateways such as credit card or UPI systems. The checkout process simulates order placement without actual payment processing.

Impact:
The application is suitable for demonstration and academic purposes but requires payment gateway integration for production use.

◈ 2. Basic UI Design

The user interface focuses more on functionality than advanced styling. Some components may lack advanced responsiveness or professional UI enhancements.

Impact:
The system works correctly but can be improved with enhanced design frameworks such as Tailwind CSS or Material UI.

◈ 3. No Real-Time Notifications

The system does not currently support real-time notifications for order updates or stock changes.

Impact:
Users must manually refresh pages to view updated order status.

◈ 4. Token Storage in Local Storage

JWT tokens are stored in browser local storage for maintaining authentication state.

Impact:
Although suitable for small-scale applications, storing tokens in local storage may expose risks such as XSS attacks in large-scale production environments. More secure storage mechanisms like HTTP-only cookies can be implemented.

◈ 5. Social Login (If Under Development)

If social authentication (Google, Facebook, LinkedIn) is partially implemented, it may require proper OAuth configuration and production keys for full deployment.

Impact:
Requires proper environment configuration before deployment.

13. Future Enhancements

Although the ShopSmart application successfully implements core e-commerce functionalities, several enhancements can be made to improve performance, security, scalability, and user experience.

◈ 1. Payment Gateway Integration

Future versions can integrate secure payment gateways such as credit/debit card processing, UPI, or digital wallets. This would enable real-time online transactions and make the system production-ready.

◈ 2. Advanced UI/UX Improvements

The user interface can be enhanced using modern UI frameworks such as Tailwind CSS or Material UI. Responsive design improvements, animations, and better product display layouts can significantly enhance user experience.

◈ 3. Social Media Authentication

Full implementation of OAuth-based login using Google, Facebook, and LinkedIn can simplify the login process and improve user accessibility.

◈ 4. Real-Time Notifications

Implementing real-time updates using technologies like WebSockets or push notifications can notify users instantly about order status changes, offers, or stock updates.

◈ 5. Inventory Management System

An advanced inventory tracking system can be implemented to automatically update stock levels and notify admins when products are running low.

◈ 6. Secure Token Storage

Instead of storing JWT tokens in local storage, HTTP-only cookies can be implemented for improved security against cross-site scripting (XSS) attacks.

◈ 7. Performance Optimization

Future enhancements may include:

- Server-side caching
- Database indexing
- Code optimization
- Lazy loading of images
- Pagination for large product datasets

◈ 8. Mobile Application Version

A mobile application version using React Native or Flutter can be developed to extend the platform to Android and iOS users.

***** Thank You *****