MEENAKSHI SUNDARARAJAN ENGINEERING COLLEGE

Kodambakkam, Chennai-600024

(An Autonomous Institution)

NM1042 – MERN STACK POWERED BY MONGODB

DEPARTMENT OF INFORMATION TECHNOLOGY

PROJECT TITLE: GROCERY WEBAPP

TEAM ID : NM2024TMID11968

FACULTY MENTOR : Mrs. G.Aswini

Project submitted by:

NAAN MUDHALVAN ID	NAME	REGISTER NUMBER
F4AA0AECD6F59FC864E4FCCB6B03A3B1	*HARIKRISHNAN M R*	*311521205017*
401FD38A7AC6D53A77E43B2AD87C1B1D	*SATHISH G*	*311521205048*
434F7994A66F74BBFDD4E72FCA5D4112	*YOKESHWARAN K*	*311521205063*
C741B3E603E3023BF4C48C9EB163F69B	*DINESH KUMAR S*	*311521205012*

ANNA UNIVERSITY: CHENNAI 600 025

BONAFIDE CERTIFICATE

Certified that this project report "**GROCERY- WEB APP**" is the bonafide work of "**HARI KRISHNAN M R (311521205017), SATHISH G (311521205048), YOKESHWARAN K (311521205063) & DINESH KUMAR S (311521205012)"** Naan Mudhalvan Team ID "**NM2024TMID11968**" who carried out the project work under my supervision.

SIGNATURE	**SIGNATURE**
Mrs. G. Aswini M. TECH	**DR. A. KANIMOZHI M.E, Ph.D.**
ASSISTANT PROFESSOR	**HEAD OF DEPARTMENT**
INFORMATION TECHNOLOGY	**INFORMATION TECHNOLOGY**
MEENAKSHI SUNDARARAJAN ENGINEERING COLLEGE	**MEENAKSHI SUNDARARAJAN ENGINEERING COLLEGE**
CHENNAI 600024	**CHENNAI 600024**

Submitted for the project viva voce of Bachelor of Engineering in Information Technology held on ___________________.

INTERNAL EXAMINER **EXTERNAL EXAMINER**

ACKNOWLEDGEMENT

First and foremost, we express our sincere gratitude to our Respected Correspondent **Dr. K. S. Lakshmi**, our beloved Secretary **Mr. N. Sreekanth**, Principal**Dr. S. V. Saravanan** for their constant encouragement, which has been our motivation to strive towards excellence.

Our primary and sincere thanks goes to **Dr. A. Kanimozhi,** Head of the Department, Department of Information Technology, for her profound inspiration, kind cooperation and guidance.

We are grateful to **Mrs. G.Aswini**, Internal Guide, Assistant Professor as our project coordinator for their invaluable support in completing our project. We are extremely thankful and indebted for sharing expertise, and sincere and valuable guidance and encouragement extended to us.

Above all, we extend our thanks to God Almighty without whose grace and blessings itwould not have been possible.

ABSTRACT

The **Grocery-Web App** is an innovative solution designed to simplify and enhance the online shopping experience for users. It provides a seamless platform for customers to explore and purchase a wide variety of products, catering to diverse needs such as daily essentials, fashion items, and tech gadgets. The app features user-friendly navigation and an intuitive interface, allowing customers to effortlessly browse categories, view product details, add items to their cart, and complete secure transactions.

For sellers and administrators, the app offers powerful backend functionalities, including product management, inventory tracking, order processing, customer support, and performance monitoring. With a strong emphasis on security and privacy, the platform ensures that customer data is protected and transactions remain safe.

This project demonstrates a commitment to creating a robust, efficient, and secure e-commerce platform that delivers a delightful shopping experience for users and streamlines operations for sellers and administrators.

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

1.1 PROJECT TITLE

GROCERY-WEB APP

1.2 TEAM MEMBERS

Hari Krishnan M.R (311521205017)

Sathish G (311521205048)

Yokeshwaran K (311521205063)

Dinesh Kumar S (311521205012)

.

CHAPTER 2

PROJECT OVERVIEW

OBJECTIVE:

The primary objective of the Grocery-Web App is to develop a seamless, secure, and user-friendly platform that enhances the online shopping experience for customers while streamlining operations for sellers and administrators. The app aims to provide customers with intuitive navigation, diverse product categories, and a hassle-free checkout process, ensuring convenience and satisfaction. For sellers, the platform offers efficient tools to manage product listings, inventory, and orders, while administrators benefit from streamlined customer support, payment processing, and performance monitoring functionalities. By prioritizing security and privacy, the app seeks to build trust with users, ensuring safe transactions and the protection of personal data. Overall, the Grocery-Web App is designed to cater to the diverse needs of its users while delivering a robust and reliable shopping solution.

TECHNOLOGY STACK:

1. **Frontend Development:**
   - HTML5, CSS3, JavaScript: For building the user interface and ensuring responsive design.
   - React.js: For creating a dynamic and interactive user experience.
2. **Backend Development:**
   - Node.js: For building the server-side application.
   - Express.js: For handling routing and API requests.
3. **Database:**
   - MongoDB: For storing and managing product details, user data, and order information.
4. **Authentication and Security:**
   - JWT (JSON Web Tokens): For secure user authentication.
   - BCrypt: For password encryption.
5. **Payment Integration:**
   - Stripe or Razorpay: For secure and reliable payment processing.

6. **Hosting and Deployment:**

   - AWS or Heroku: For deploying the backend and frontend applications.
   - Netlify or Vercel: For hosting the frontend.

7. **Version Control:**

   - Git and GitHub: For code collaboration and version management.

8. **Additional Tools:**

   - Postman: For API testing.
   - Firebase (optional): For real-time database or authentication services.

KEY FEATURES:

The Grocery-Web App is designed with a range of features to enhance the shopping experience for customers, streamline operations for sellers, and provide efficient management tools for administrators. Customers can enjoy a user-friendly interface, seamless navigation, detailed product pages, and a secure checkout process. Sellers benefit from robust tools to manage product listings, track inventory, and process orders efficiently.

Administrators have access to a comprehensive dashboard for handling customer queries, managing transactions, and monitoring app performance. The app prioritizes security with encrypted authentication, secure payment gateways, and data privacy protection. Additionally, features like a mobile-friendly design, real-time notifications, and advanced search functionality ensure a smooth and convenient experience.

CHALLENGES & SOLUTIONS:

- **Challenge:** Ensuring a User-Friendly Interface
- **Problem**: Designing a clean, intuitive layout to enhance user experience without overwhelming users with options.
- **Solution**: Implemented a responsive and minimalistic design using React.js with clear navigation and well-structured categories to make browsing seamless.
- Real-Time Functionality: Socket.IO was implemented to deliver order status updates to users, enhancing the user experience.

CONCLUSION:

The Grocery-Web App successfully addresses the needs of both customers and sellers, providing a seamless and secure online shopping experience. With a user-friendly interface, robust backend functionalities, and a strong focus on security, the app delivers a reliable and efficient platform for browsing, purchasing, and managing products.

2.1 PURPOSE

The purpose of the Grocery-Web App is to provide a convenient, secure, and efficient platform for online shopping, catering to the needs of diverse customers while offering robust tools for sellers and administrators. By simplifying the shopping process with an intuitive interface and seamless checkout experience, the app aims to enhance customer satisfaction and loyalty. It also empowers sellers to manage their products, inventory, and orders effectively, driving operational efficiency. Additionally, the app focuses on maintaining high standards of data security and privacy to ensure safe transactions and build trust with users. Ultimately, the goal is to create a comprehensive and reliable e-commerce platform that connects customers and sellers in a secure, efficient, and user-friendly environment.

For sellers, the app provides a powerful backend system to manage product listings, track inventory, process orders, and monitor performance, helping them streamline their operations and increase sales. By offering real-time insights and efficient tools, the app enables sellers to focus on growth and customer satisfaction.

Additionally, the app seeks to foster trust and reliability in e-commerce by prioritizing security, implementing secure payment gateways, and ensuring customer data protection. The ultimate purpose is to build a platform that not only meets the current demands of online shopping but also adapts to the evolving needs of both consumers and businesses, providing a seamless, secure, and rewarding shopping experience for everyone.

2.2.1.User Account Management

- **Sign Up / Login**: Allow users to create accounts or sign in throughemail, phone number, or social media platforms.
- **Profile Management**: Users can update their personal information,payment methods, and order history.
- **Order History**: Displays past orders, allowing users to reordereasily.

2.2.2.Search and Discover

- **Restaurant Listings:** List restaurants based on location, ratings,cuisine type, popularity, etc.
- **Menu Browsing:** Detailed menu listings with item descriptions,prices, and availability.
- **Filtering and Sorting:** Filter restaurants by ratings, delivery time,cuisines, and cost.
- **Search Bar:** Keyword-based search to find specific dishes orrestaurants.

2.2.3.Menu Customization and Ordering

- **Add to Cart:** Users can add items to a cart, modify quantities, orremove items before checkout.
- **Special Instructions:** Users can add special instructions.
- **Customizable Options:** Allow users to modify items (e.g., extratoppings, spice level).

2.2.4.Payment Options

- **Multiple Payment Methods:** Accept credit/debit cards, digital wallets,UPI, and cash-on-delivery.
- **Saved Cards:** Save card details securely for fast checkouts.

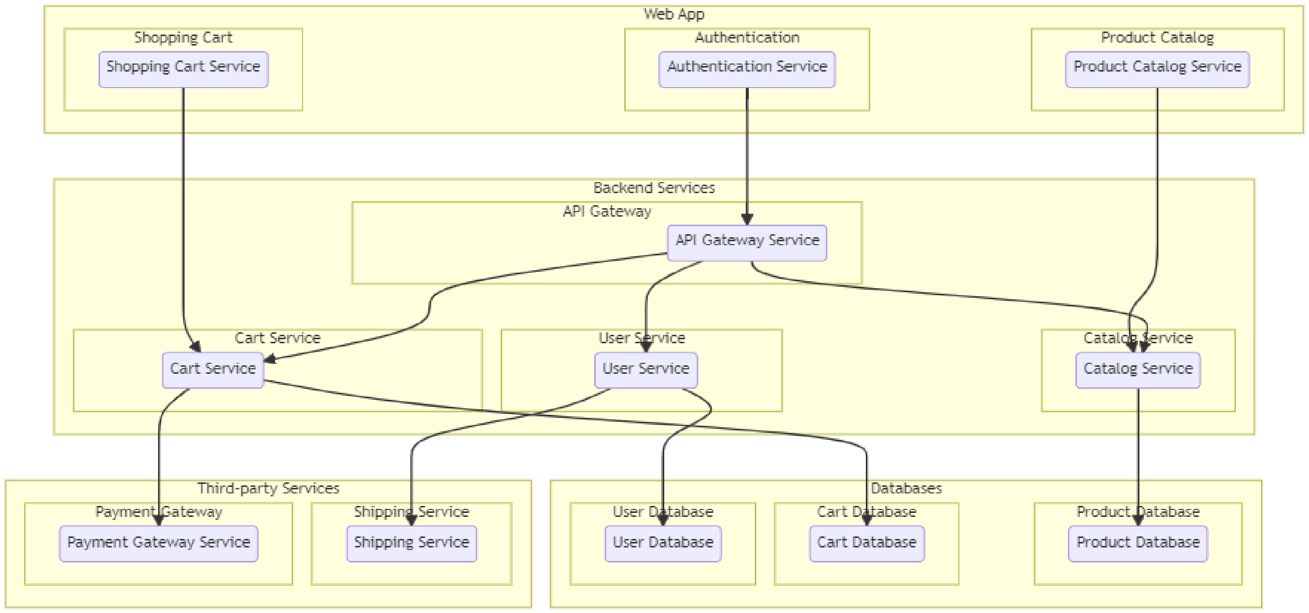
- **Split Payment:** Allows group ordering and splitting of bills amongfriends.

2.2.5.Admin Panel for Restaurant

- **Menu Management:** Add, remove, and update menu items andprices.
- **Order Management:** View and manage incoming orders with real-time updates.
- **Analytics Dashboard:** View insights on sales, popular items,customer feedback, etc.

CHAPTER 3

ARCHITECTURE



3.1 FRONTEND ARCHITECTURE

The **Web App** represents the user-facing interface of the Grocery Web App, which allows customers to interact with the system. It consists of the following key elements:

1. **Authentication Service**:
   - The front-end communicates with the Authentication Service for user login, registration, and session management. This service handles tasks like user authentication, managing tokens, and ensuring secure access to the application.

2. **Product Catalog Service**:
    - The front-end interacts with the Product Catalog Service to display the products available in the app. This service provides the product details (name, price, description, etc.) that are shown to customers. Users can browse through different product categories, view detailed product pages, and add products to their shopping cart.
3. **Shopping Cart**:
    - The front-end communicates with the Shopping Cart Service to handle the shopping cart functionality. Customers can add products to the cart, update quantities, remove items, and view the total cost of their cart. The front-end ensures the cart is synchronized in real-time with the back-end services.

3.2 BACKEND ARCHITECTURE

The backend architecture of the Grocery Web App follows a **microservices architecture** where each service is responsible for specific functionality, allowing the application to scale, maintain, and update individual services independently. Here's a breakdown of each backend component and its role:

1. **API Gateway**:
    - **Function**: Acts as the single entry point for all incoming requests from the front-end. The API Gateway routes requests to the appropriate backend services (such as Authentication, User, Cart, Product Catalog). It also handles tasks like load balancing, request routing, authentication, and rate limiting.
    - **Role in Backend**: Ensures all communication between the front-end and backend services goes through a central point, enabling easier management and monitoring.
2. **Authentication Service**:
    - **Function**: Handles user authentication and authorization. It ensures that users are properly authenticated when they log in, register, or access restricted resources. This service is responsible for managing user credentials, generating authentication tokens (like JWT), and verifying users' identities.
    - **Role in Backend**: It communicates with the **User Database** to verify login credentials and maintain user sessions. This service ensures that only authenticated users can access certain functionalities (like making purchases).

3. **Product Catalog Service**:
    - **Function**: Manages the list of available products. It handles the retrieval and display of product information such as names, descriptions, prices, and stock levels. This service queries the **Product Database** to fetch product details based on the user's request (e.g., browsing by category or searching for specific items).
    - **Role in Backend**: Acts as the main controller for managing and serving product data to the front-end. It interacts with the **Product Database** for adding, updating, or deleting products from the catalog.
4. **Cart Service**:
    - **Function**: Manages the shopping cart of users, handling the adding, removing, and updating of products in the cart. It maintains the cart's contents for each user, calculates the total cost, and prepares the cart for checkout.
    - **Role in Backend**: Communicates with the **Cart Database** to store and retrieve cart information and with the **User Service** to tie the cart to specific users. It ensures that users' carts are persisted and synchronized across sessions.
5. **User Service**:
    - **Function**: Responsible for managing user information. This service handles user profiles, updating user details, and

providing access to user-related actions like order history, shipping address, and payment preferences.

   - **Role in Backend**: Interacts with the **User Database** to fetch or update user data. It also communicates with other services like the Cart Service and Authentication Service for user-specific actions (e.g., fetching cart items, managing user credentials).

6. **Catalog Service**:
   - **Function**: Similar to the Product Catalog Service, it handles additional catalog-related operations, such as organizing products into categories, filtering products based on user preferences, and providing advanced search capabilities.
   - **Role in Backend**: Works closely with the Product Catalog Service to ensure the product data is categorized and searchable in a structured manner.
7. **Payment Gateway Service** (Third-party service):
   - **Function**: Handles the processing of payment transactions. It connects to external payment providers (such as Stripe or PayPal) to securely process payment information and confirm successful transactions.
   - **Role in Backend**: This service interacts with the **Cart Service** to calculate the total payment due and then processes the payment through a secure third-party gateway.

8. **Shipping Service** (Third-party service):
    - **Function**: Manages the shipping and delivery process. It connects with third-party shipping services to calculate shipping costs, determine delivery times, and handle tracking of shipments.
    - **Role in Backend**: After a user completes the purchase, the Shipping Service helps in calculating the cost and delivery options for the selected products.

3.3 DATABASE

The database uses MongoDB with a schema defined through Mongoose. Collections are created to manage restaurants, menu items, orders, and user- specific elements like cart and order history.

3.3.1. DATABASE STRUCTURE

Fields:

- **username:** A unique identifier for each user (string).
- **password:** A hashed password for secure user authentication (string). The password is hashed using bcrypt for encryption.
- **email:** A unique email address for communication and account verification (string). This must be a valid email format and is used for login and notifications.

- **userType**: Specifies the role of the user (e.g., "customer", "admin", "seller"). This helps in differentiating the functionality available to the user (string).
- **address:** Stores one or more shipping addresses for easy order processing (array of objects). Each object may contain street, city, zipCode, and country.
- **phone:** A contact number for the user, which can be used for notifications and order updates (string).
- **created_at:** Timestamp for when the user account was created (Date)

DATABASE OPERATIONS:

The database layer is powered by MongoDB, designed to perform efficient data operations specifically tailored for users, restaurants, menu items, orders, carts, and favorites. Typical operations within the database include the standard CRUD functions—Create, Read, Update, and Delete—along with specialized actions that address the unique requirements of food ordering systems.

CHAPTER 4
SETUP INSTRUCTION

To set up the Grocery-web application, follow these steps:

4.1 PREREQUISITES:

- **Node.js (Version 20.x):** Download and install from Node.js officialwebsite, which includes npm (Node Package Manager).
- **MongoDB:** Set up MongoDB using MongoDB Compass for a localinstance or create a MongoDB Atlas account for a cloud-based setup.
- **Git:** Install Git from Git downloads for version control.
- **npm -** Node Package Manager (comes with Node.js).
- **Code Editor -** Visual Studio Code or another preferred IDE.

4.2 INSTALLATION:

1. Download Project Files:

Place all project files in a dedicated project directory on the local machine.

2.Install Dependencies:

Open a terminal, navigate to the project directory, and install the required dependencies:

npm install

3.Set Up Environment Variables:

In the project root, create a .env file to store environment-specific variables suchas database connection strings and JWT secrets. Contents in the .env file is as follows:

```
PORT=3001
MONGO_URI=<mongodb-
connection-string>
JWT_SECRET=<jwt-secret>
```

1. Run the Application:

Start the application in development mode:

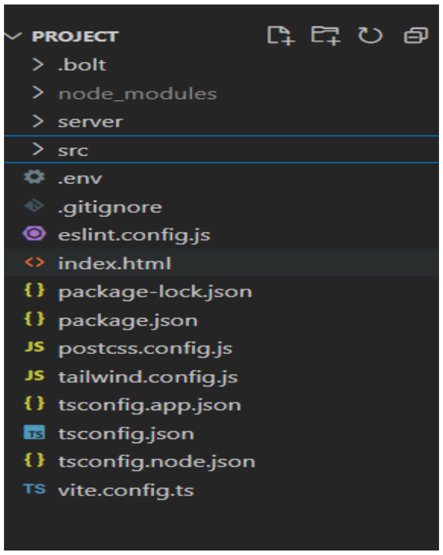
npm start

Now the application can be accessed locally at http://localhost:5174/

CHAPTER 5

FOLDER STRUCTURE

The QuickBite Food Ordering application follows a well-organized folder structure for both the **Frontend** (React) and **Backend** (Node.js) to ensure clarity, scalability, and maintainability of the project.



```
PROJECT
  > .bolt
  > node_modules
  > server
  > src
  .env
  .gitignore
  eslint.config.js
  index.html
  package-lock.json
  package.json
  postcss.config.js
  tailwind.config.js
  tsconfig.app.json
  tsconfig.json
  tsconfig.node.json
  vite.config.ts
```

Figure .Overall Folder Structure

5.1 CLIENT:

React Frontend Structure

1. public/

Contains static files that are publicly accessible, like index.html, favicon.ico, and assets that do not require processing by Webpack (e.g., logos, icons).

2. src/

This is where the core React application resides.

components/: Contains reusable components such as buttons, form fields, product cards, and UI elements used throughout the application. These components are designed to be generic so they can be used in different parts of the app.

pages/: Each file in this folder represents a page or view within the application. For example, the HomePage, ProductDetailPage, and CheckoutPage represent distinct pages that users will navigate to.

context/: This folder manages global state using the React Context API. It handles the global data shared across the app, like user authentication status and shopping cart state. By using Context, you can avoid prop drilling and manage app-wide state efficiently.

styles/: This folder contains CSS files or custom styling for the components. By modularizing styles, you ensure that they are maintainable and reusable across different components and pages.

images/: This folder stores image assets like product images, icons, logos, and other media that are used in the application. By centralizing images here, you can easily manage and reference them .

App.js: This is the root component that initializes the app, handles routing (via react-router-dom), and sets up the main layout (such as navigation, header, footer).

index.js: The entry point of the React app where ReactDOM.render() is used to render the App.js component into the root div element in index.html.

routes.js: Centralizes routing logic using react-router-dom to define paths and associated components. This helps manage navigation throughout the app and improves the maintainability of the app's routing structure.

package.json:This file manages project dependencies, scripts, and other configurations for the frontend application.

5.2 SERVER:

Node.js Backend Structure

1. models/

Contains Mongoose schemas to define the structure of the data stored in MongoDB. Each schema corresponds to an entity in your application, such as Users, Products, Orders, Cart, and Addresses.

2. routes/

Contains the route definitions that map HTTP requests to specific controller functions. Each file corresponds to a specific resource (like products, users, orders).

3. controllers/

This folder contains the logic for handling requests. The controllers interact with the models (from the models/ folder) and return responses.

CHAPTER 6

RUNNING THE GROCERY WEB APPLICATION

To run the Grocery Web Application locally, follow these steps to start both the frontend and backend servers:

6.1 FRONTEND COMMANDS

1.Navigate to the Client Directory:

- Open a terminal window on your system.
- Use the cd (change directory) command to navigate to the client folder, which contains all the frontend code for the application.

2.Install Frontend Dependencies:

Before starting the frontend server, ensure that all required dependencies are installed. If this is the first time running the project, or if dependencies have been added or updated, run the following command:

```
npm install
```

This will install all the necessary packages listed in the package.json file, including React, React Router, and other dependencies.

3.Start the Frontend Server:

After installing the dependencies, use the following command to start the React development server:

```
npm start
```

This will compile the React application and start a local development server.

4.Access the Application:

Once the server is running, open a web browser and go to:

arduino

```
http://localhost:3000
```

The frontend application will be accessible at this URL. You should see the Grocery Web Application user interface, and the frontend is now live and connected to the backend server for full functionality, allowing users to explore and purchase products, browse categories, and securely complete transactions.

6.2 BACKEND COMMANDS

1.Navigate to the Server Directory:

- Open a terminal window.
- Use the cd (change directory) command to navigate to the server folder, which contains all the backend code for the application.

2.Install Backend Dependencies:

If this is the first time running the backend server, ensure all required dependencies are installed by running:

```
npm install
```

This will install all the necessary backend packages, including Express, Mongoose, and other dependencies as specified in the package.json file.

3.Start the Backend Server:

Once the dependencies are installed, use the following command to start the backend server:

```
node index.js
```

This will initialize the backend server and connect it to the database.

4.Access the Backend:

The backend server will now be running and can be accessed via:

arduino

```
http://localhost:5000
```

This URL handles all API requests and will process interactions such as product management, inventory control, user authentication, order processing, and more. It also ensures secure transactions for users, protecting their personal information and payment details.

CHAPTER 7

AUTHENTICATION

The **Grocery Web Application** follows a secure and efficient authentication and authorization process to manage user access, ensuring data protection and role-based control. It uses **JSON Web Tokens (JWT)**, **password hashing**, and **session management** to ensure secure user authentication and effective session control.

1. User Authentication with JWT Tokens

- **Description**: Upon successful user login or registration, the system generates a **JSON Web Token (JWT)**. This token contains encoded user-specific details, such as user ID and role (e.g., customer or admin), and is signed with a secret key to ensure authenticity and security.
- **Process**:
  On successful login, the backend generates the JWT token, which is sent to the client.
  The token is stored in the client's **session storage**.
  This token is included in the HTTP headers for all **protected requests**. It allows the backend to authenticate the user on each request.
  The backend decodes the token to retrieve user details and validate its integrity.

2. Token-Based Authorization

- **Description**: The system implements **token-based authorization** by validating the JWT token on each protected request. This ensures that only authenticated users can access specific parts of the application.
- **Role-Based Authorization**:
- **Customer**: Can access the product catalog, manage their cart, place orders, and view their order history.
- **Admin**: Has access to the **Admin Dashboard**, where they can manage products, view all users, and modify orders.
- Each role's permissions are verified through the information embedded in the JWT token (such as user ID and role).
- **Protected Routes**:
- Only authenticated users with valid JWT tokens are allowed to access routes like /api/orders, /api/cart, and /api/admin (for admin users).

3. Password Hashing

- **Description**: **Password hashing** is applied to secure user passwords before they are stored in the database. This ensures that even if the database is compromised, user credentials remain safe.
- **Process**:
- **bcrypt** is used to securely hash passwords before storing them.

- When a user logs in, the system hashes the entered password and compares it with the stored hash.
- If the hashes match, the user is authenticated; otherwise, an error is returned.

4. Session Management

- **Description**: The application manages user sessions through **session storage**, allowing users to stay logged in across multiple browser tabs.
- **User Data**:
- User details such as **user ID**, **username**, **email**, and **user type** are stored in session storage, providing a consistent experience across different parts of the app.
- **Session Persistence**: As long as the session is active and the token is valid, users will remain logged in.
- **Token Expiry**:
- JWT tokens include an **expiry timestamp**, after which they are no longer valid.
- The system automatically logs users out when the token expires, improving security by preventing unauthorized access after inactivity.
- **Security Measures**
- **Cross-Site Request Forgery (CSRF)**: Tokens are stored in **session storage**, which prevents them from being automatically included in requests from other websites, reducing CSRF risks.

- **Cross-Origin Resource Sharing (CORS)**: The application uses CORS headers to restrict cross-origin requests, preventing unauthorized sites from making requests to the backend.
- **Encryption**: All communication between the client and server is secured using **HTTPS**, ensuring data encryption during transmission.
- ity by preventing unauthorized access.

CHPTER 8

USER INTERFACE

The **Grocery Web Application** is designed with a user-friendly, responsive, and aesthetically pleasing interface to ensure smooth navigation for both customers and administrators. Built using **React**, the UI enables seamless interactions and an engaging experience, guiding customers through browsing products, managing orders, and completing transactions while offering admins robust control over the platform's operations.

1. Customer Experience

The **Grocery Web Application** provides a simple and enjoyable shopping experience for users, emphasizing ease of use and accessibility:

- **Homepage**: The homepage features a clean and intuitive layout with key sections for product categories, promotional offers, and a prominent search bar to allow users to quickly find products. Popular grocery items, new arrivals, and deals are prominently displayed with clear product images, names, and prices.
- **Product Listings**: Users can browse a wide variety of grocery items, including fresh produce, pantry staples, and specialty products. Each product is displayed with essential details, such as name, description, price, and an image. Filters and sorting options allow users to narrow down search results by category, price range, or brand.

- **Product Details**: Clicking on a product provides more detailed information, including specifications, available sizes, and nutrition facts. Users can choose quantities, customize selections (e.g., size or packaging), and view estimated delivery costs before adding items to their cart.
- **Cart Management**: The shopping cart allows users to view their selected items, adjust quantities, remove products, and see the total cost, including taxes and delivery fees. The checkout process is streamlined for ease, offering multiple payment methods (e.g., credit card, cash on delivery).
- **Profile and Order History**: Customers can manage personal details, delivery addresses, and track past orders. The "Order History" feature provides easy access to previous purchases for reordering or tracking the status of ongoing orders.
- **Wishlist**: Users can add products to their wishlist to save items for future purchases, helping them easily find items they wish to buy later.

2. Admin Interface

The **Admin Dashboard** offers a comprehensive suite of management tools to control and monitor the platform:

- **User and Product Management**: Admins can view and manage user accounts, modify product listings, and approve or reject new product submissions from vendors. They can edit product details (such as pricing, description, and images) and remove outdated or unavailable items.

- **Order Management**: The admin interface allows admins to track orders placed by customers, view order statuses (e.g., pending, shipped, delivered), and assist in resolving any order issues. Admins can view detailed customer information, including shipping addresses and purchase history.
- **Vendor and Inventory Monitoring**: Admins can manage vendors and suppliers, ensuring that product availability is up-to-date. The inventory management system helps track stock levels and alerts admins to restock products that are running low.
- **Analytics and Reporting**: The dashboard provides real-time analytics on key performance metrics such as order volumes, customer activity, popular products, and revenue generation. These insights help admins make data-driven decisions and optimize operations.
- **Customer Support**: Admins can respond to customer queries, handle complaints, and manage support tickets, ensuring timely issue resolution and a positive customer experience.

3. Vendor Interface

The **Vendor Dashboard** is designed to provide a seamless experience for grocery suppliers and store owners to manage their product offerings and track orders:

- **Product and Pricing Management**: Vendors can add new products to the catalog, edit existing products, and adjust pricing based on demand and supply. Vendors can also set availability statuses for items (in-stock or out-of-stock) to keep the product listing updated.
- **Order Tracking**: Vendors can view incoming orders, track the status of shipments, and manage order fulfillment. They have the ability to update order statuses and ensure timely delivery.
- **Customer Insights**: Vendors can view customer preferences, top-selling products, and feedback, allowing them to adjust their offerings accordingly.
- **Inventory Monitoring**: The interface provides tools for vendors to monitor stock levels, receive alerts for low-stock products, and ensure that high-demand items remain available.

4. Responsiveness and Aesthetics

The UI is designed to provide a smooth and engaging experience across all devices, ensuring that customers and admins can use the application seamlessly on mobile phones, tablets, or desktops:

- **Responsive Design**: The layout automatically adjusts to different screen sizes, ensuring consistent user experience on all devices. **Bootstrap** is utilized for its grid system, providing flexibility and scalability for various screen resolutions.
- **Aesthetic Design**: The visual design emphasizes simplicity and modernity. The color scheme is clean and neutral, with vibrant

product images to make browsing and shopping visually appealing. High-quality images of grocery items, along with intuitive icons and buttons, create a pleasant and organized shopping environment.

- **Smooth Interactions**: The UI ensures smooth transitions between pages, and customers can easily interact with the application using well-designed buttons, form fields, and navigation menus. The interface is designed to reduce friction, allowing users to quickly complete their shopping without confusion.

CHAPTER 9

TESTING

The testing strategy for the **Grocery Web Application** focuses on validating the functionality, security, and performance of the platform using various testing techniques. Postman is used to conduct thorough API testing, ensuring that all backend services are robust, reliable, and secure. The testing approach also includes error handling validation, performance testing, and automation for repeatability.

1. API Testing with Postman

Postman is an essential tool for testing the backend API endpoints of the **Grocery Web Application**, which includes user authentication, product catalog management, order processing, and cart functionalities. Each API endpoint is tested to ensure that the system responds correctly to requests and behaves as expected under different scenarios.

- **Authentication Endpoints**: The API endpoints for user login, registration, and token validation are tested by sending various login credentials and verifying the correct generation of JSON Web Tokens (JWTs). The tokens are then validated to ensure token-based access control works as expected for restricted routes. This ensures that only authenticated users can access protected resources like personal data or order history.
- **Product and Catalog Management Endpoints**: These endpoints are tested to ensure that vendors can successfully add, update, and

retrieve product details (e.g., product names, descriptions, prices, and images). The API is also tested to confirm that any updates to the catalog are correctly reflected in the system. Test cases verify that the endpoints handle valid and invalid product data correctly and that appropriate error responses are generated when required fields are missing or data is incorrect.

- **Order and Cart Management Endpoints**: These API endpoints are crucial for managing customer orders and cart items. Test cases are created for adding and removing products from the cart, updating quantities, calculating total costs, and placing an order. The order status endpoints (e.g., Order Received, In-Process, Out for Delivery, Delivered) are also thoroughly tested to ensure the correct transition of order statuses and the proper handling of special scenarios, such as order cancellations or delays.

2. Error Handling and Validation

Proper error handling is crucial for delivering a seamless and user-friendly experience. During testing, responses from API calls are checked to ensure that they return accurate status codes (e.g., 200 for success, 400 for bad request, 401 for unauthorized access, and 404 for not found). This ensures that errors are clearly communicated to users and developers.

- **Positive and Negative Test Cases**:
    - **Positive Test Cases**: Valid inputs (e.g., correct login credentials, valid product IDs, and correct API parameters) are

tested to ensure that the system behaves as expected and responds with success status codes.

    - **Negative Test Cases**: Edge cases and invalid scenarios are also tested, such as incorrect login credentials, missing parameters (e.g., missing product IDs or quantities), unauthorized access attempts, and invalid data formats. These tests ensure that the system responds appropriately with error messages, helping developers identify and fix issues efficiently.

- **Security Testing**: The application undergoes tests to ensure that unauthorized users cannot access sensitive data. This includes testing endpoints that require authentication or specific roles, such as admin or vendor, and verifying that access control mechanisms are correctly enforced.

3. Test Automation and Repeatability

To ensure that the application remains reliable as it evolves, Postman collections are used to automate and organize the API testing process. The following practices are employed:

- **Postman Collections**: A set of Postman collections is created for each module of the application (e.g., user authentication, product management, order processing). These collections group relevant tests and allow for easy execution of multiple test cases at once.

- **Automated Test Execution**: Automated tests are run regularly to check for regressions or issues that might arise during development or after updates. Automated execution ensures that the APIs continue to function as expected after code changes or updates to the system.
- **Continuous Integration (CI) Setup**: The Postman collections can be integrated into the Continuous Integration (CI) pipeline, allowing tests to run automatically with every deployment. This ensures that any issues are detected early and that the API performs consistently across different environments and stages of development.
- **Repeatability and Consistency**: The Postman tests are designed to be repeatable and consistent, allowing developers and QA engineers to quickly validate endpoint behavior. This increases efficiency and ensures that the application maintains reliable performance as new features or changes are introduced.

4. Performance Testing

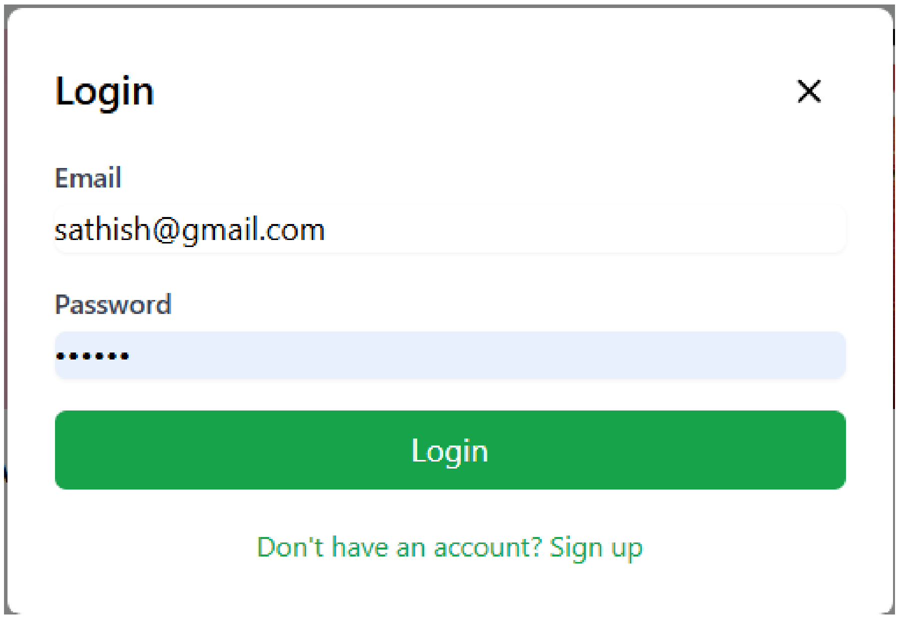
In addition to functional API testing, performance testing is conducted to ensure the system performs well under various load conditions. This includes:

- **Load Testing**: The application's ability to handle a large number of simultaneous users (e.g., many customers placing orders at the same time) is tested. This helps identify any performance bottlenecks or areas where the system may struggle under high demand.

- **Stress Testing**: The system is tested under extreme conditions, such as simulating a sudden spike in traffic or a high volume of simultaneous orders, to determine its breaking point and ensure it can recover gracefully.
- **Latency Testing**: API response times are tested to ensure that they meet performance expectations and provide users with a smooth experience when interacting with the system.

CHAPTER 10

SCREENSHOTS



Login

Email

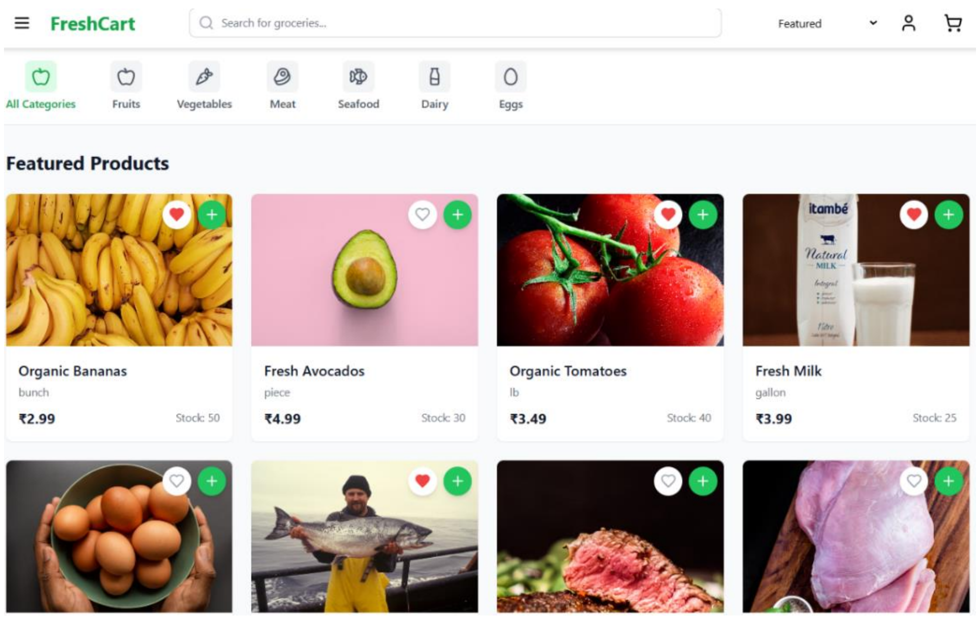
sathish@gmail.com

Password

••••••

Login

Don't have an account? Sign up



FreshCart

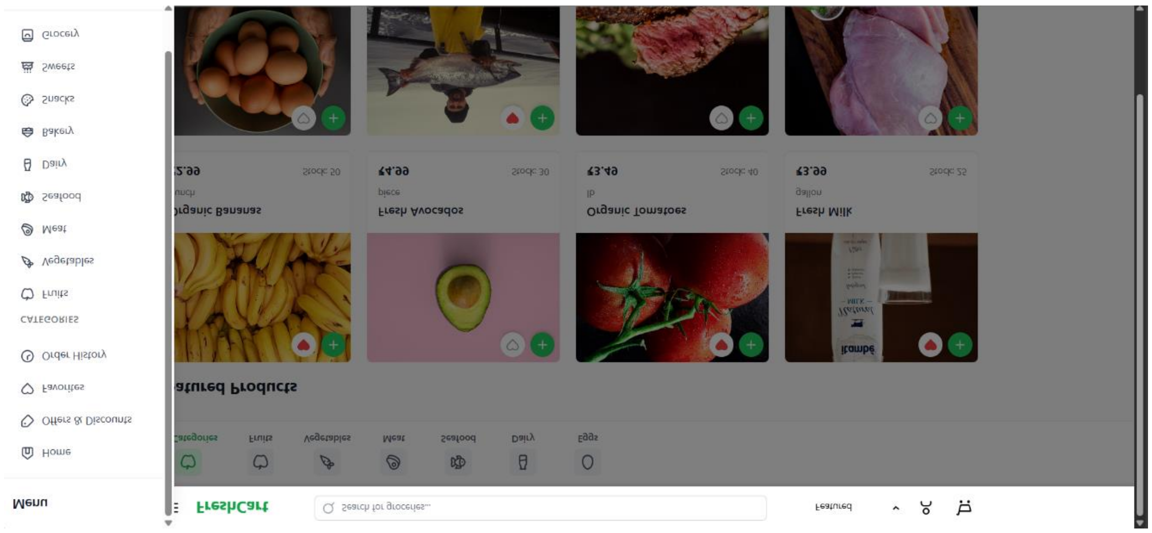
Search for groceries...

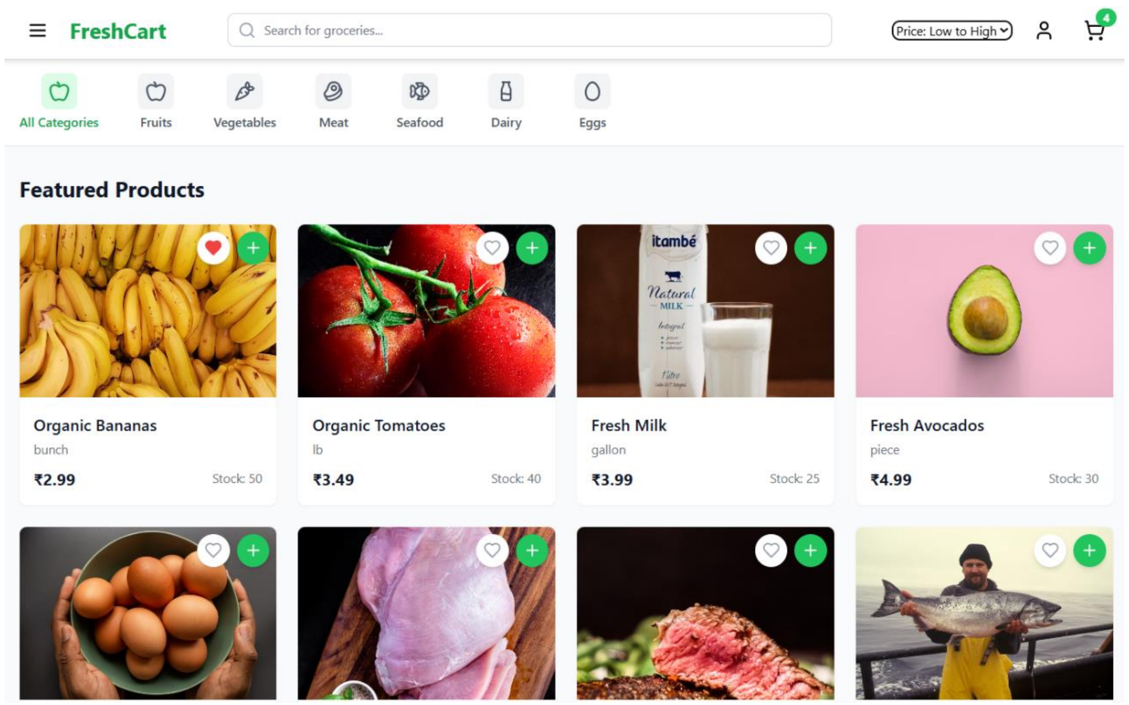
Featured

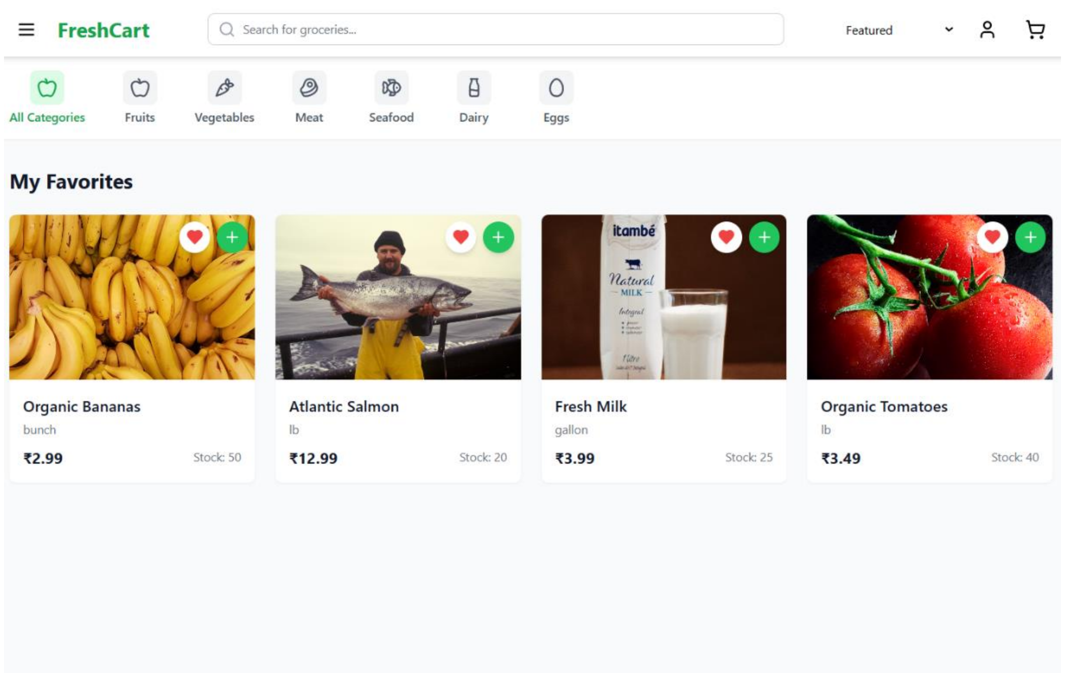
All Categories | Fruits | Vegetables | Meat | Seafood | Dairy | Eggs

Featured Products

Product	Unit	Price	Stock
Organic Bananas	bunch	₹2.99	Stock: 50
Fresh Avocados	piece	₹4.99	Stock: 30
Organic Tomatoes	lb	₹3.49	Stock: 40
Fresh Milk	gallon	₹3.99	Stock: 25


Menu
Home
Offers & Discounts
Favorites
Order History
CATEGORIES
Fruits
Vegetables
Meat
Seafood
Dairy
Bakery
Snacks
Sweets
Grocery
FreshCart
Search for groceries...
Featured
Categories
Fruits
Vegetables
Meat
Seafood
Dairy
Eggs
atured Products
rganic Bananas
unch
2.99
Stock: 50
Fresh Avocados
piece
₹4.99
Stock: 30
Organic Tomatoes
lb
₹3.49
Stock: 40
Fresh Milk
gallon
₹3.99
Stock: 25


FreshCart
Search for groceries...
Price: Low to High
4
All Categories
Fruits
Vegetables
Meat
Seafood
Dairy
Eggs
Featured Products
Organic Bananas
bunch
₹2.99
Stock: 50
Organic Tomatoes
lb
₹3.49
Stock: 40
Fresh Milk
gallon
₹3.99
Stock: 25
Fresh Avocados
piece
₹4.99
Stock: 30



FreshCart

Search for groceries...

Featured

All Categories | Fruits | Vegetables | Meat | Seafood | Dairy | Eggs

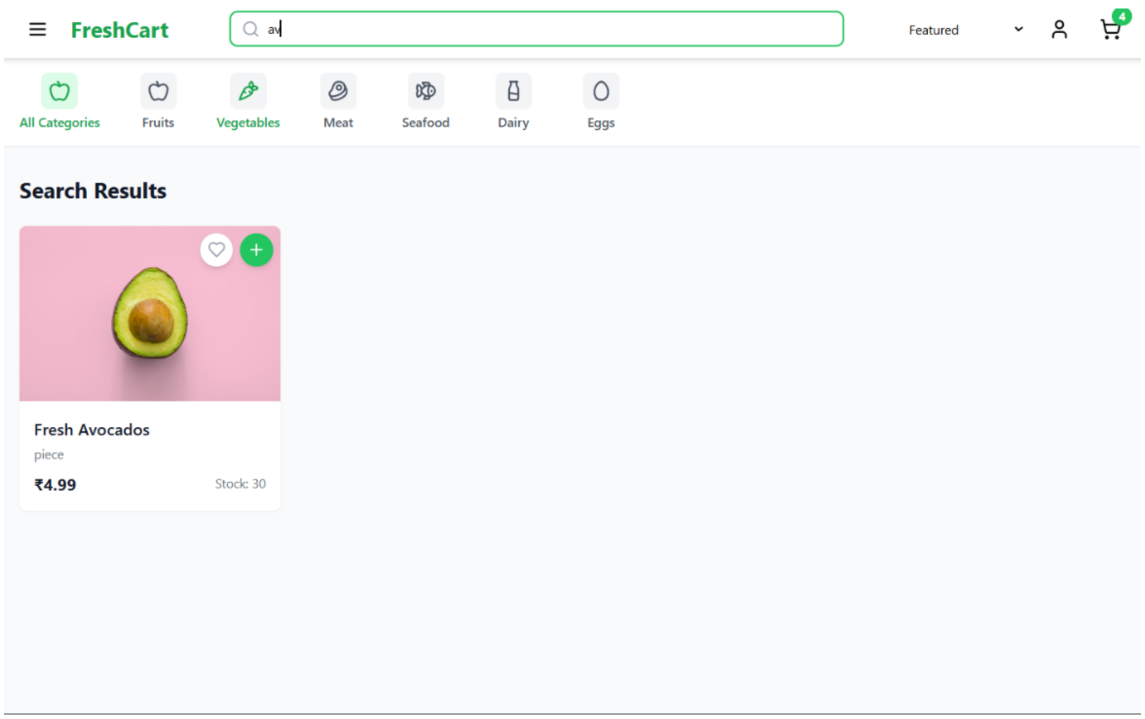
My Favorites

Organic Bananas
bunch
₹2.99 Stock: 50

Atlantic Salmon
lb
₹12.99 Stock: 20

Fresh Milk
gallon
₹3.99 Stock: 25

Organic Tomatoes
lb
₹3.49 Stock: 40



FreshCart

av

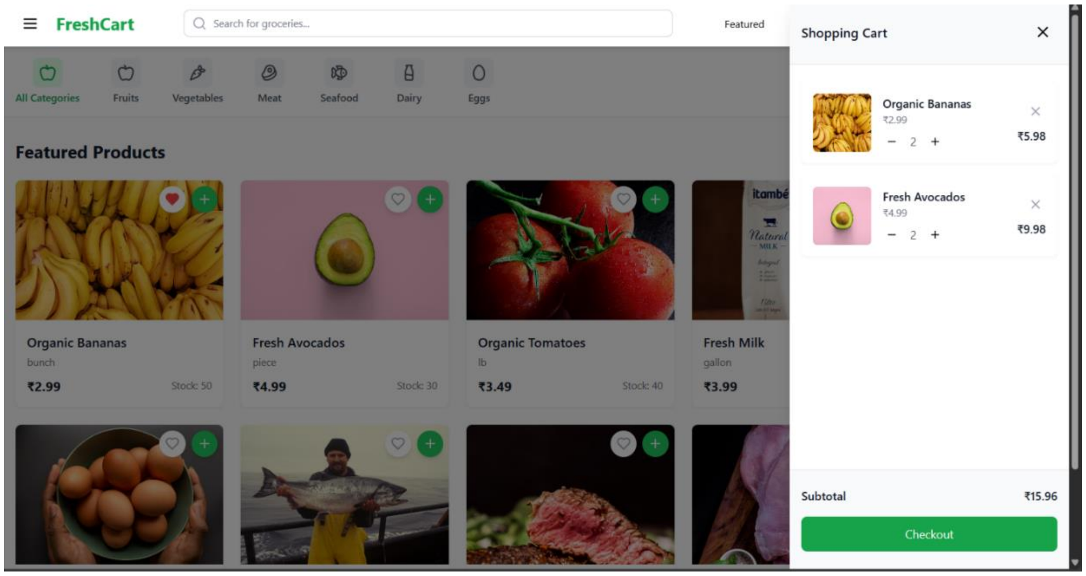
Featured

4

All Categories | Fruits | Vegetables | Meat | Seafood | Dairy | Eggs

Search Results

Fresh Avocados
piece
₹4.99 Stock: 30



FreshCart

Search for groceries...

Featured

All Categories · Fruits · Vegetables · Meat · Seafood · Dairy · Eggs

Featured Products

Organic Bananas
bunch
₹2.99 Stock: 50

Fresh Avocados
piece
₹4.99 Stock: 30

Organic Tomatoes
lb
₹3.49 Stock: 40

Fresh Milk
gallon
₹3.99

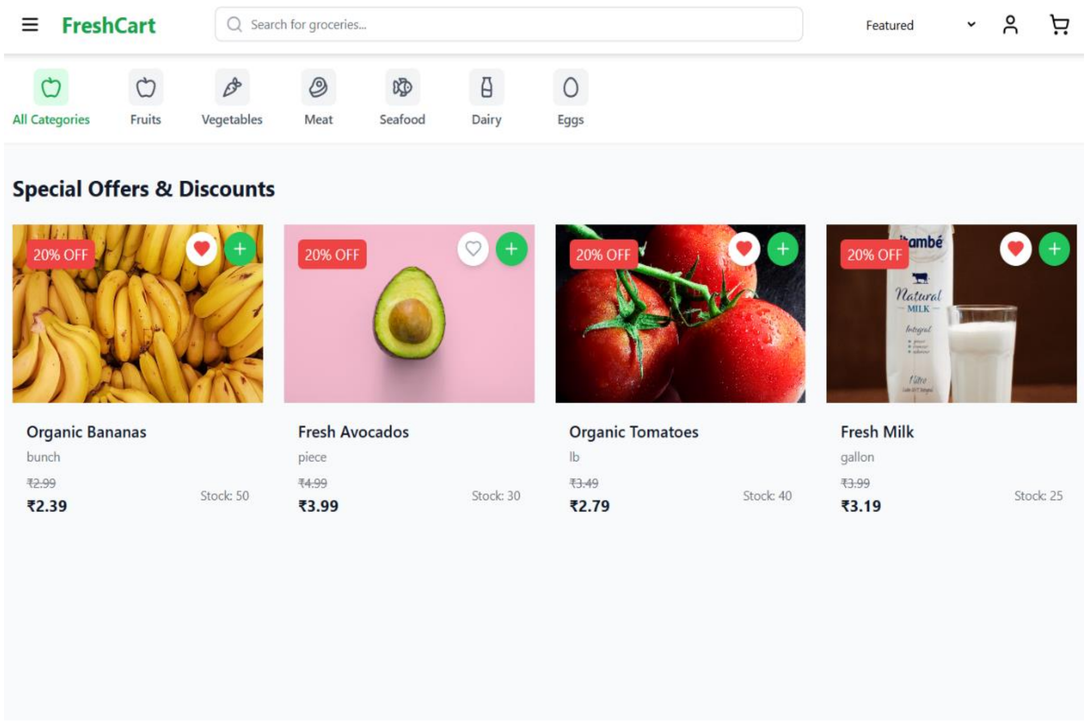
Shopping Cart

Organic Bananas
₹2.99
− 2 + ₹5.98

Fresh Avocados
₹4.99
− 2 + ₹9.98

Subtotal ₹15.96

Checkout



FreshCart

Search for groceries...

Featured

All Categories · Fruits · Vegetables · Meat · Seafood · Dairy · Eggs

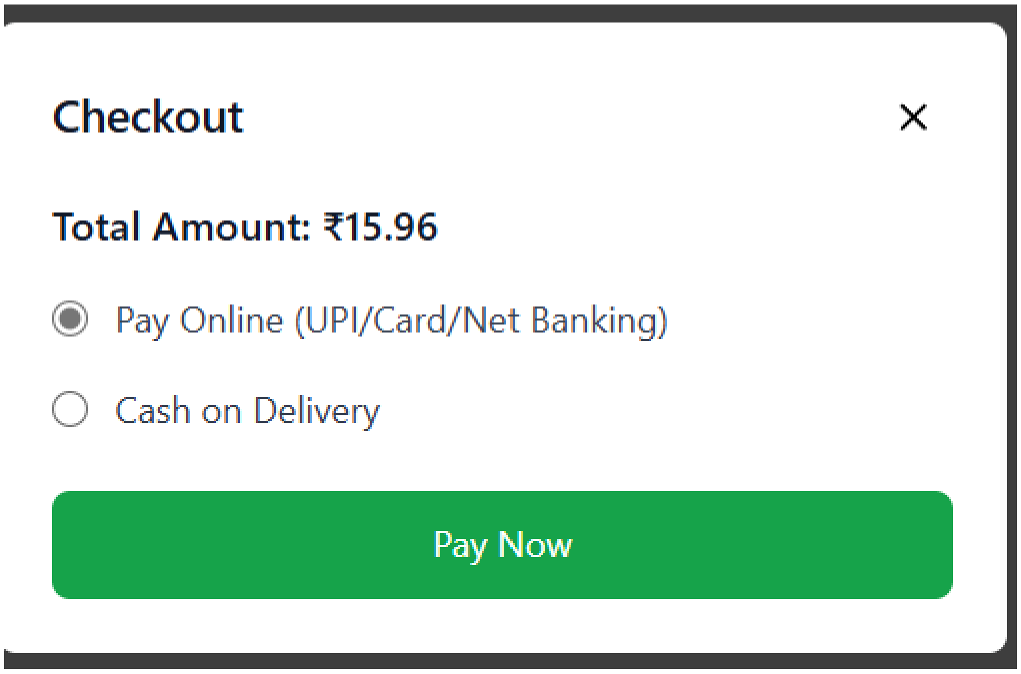
Special Offers & Discounts

20% OFF
Organic Bananas
bunch
~~₹2.99~~
₹2.39 Stock: 50

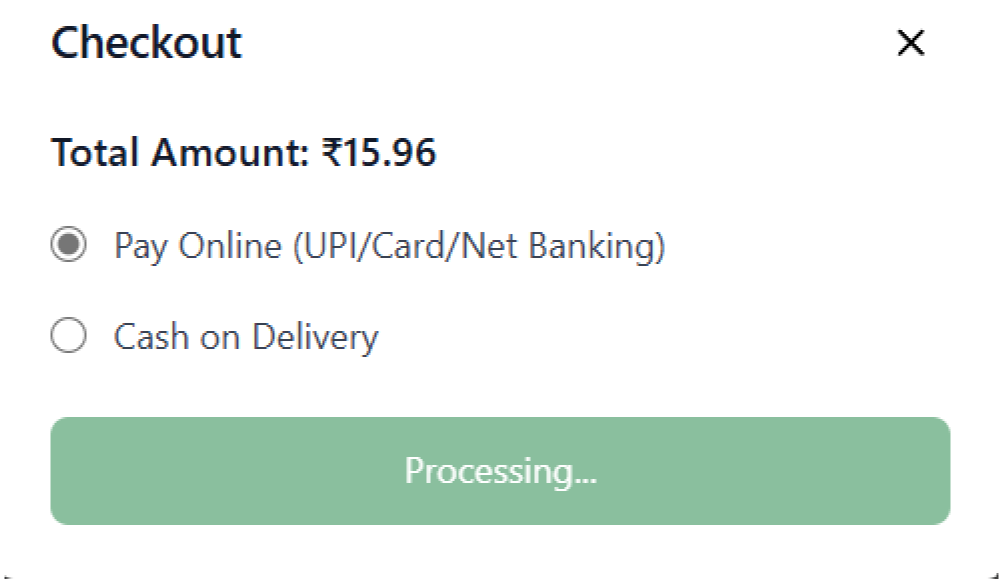
20% OFF
Fresh Avocados
piece
~~₹4.99~~
₹3.99 Stock: 30

20% OFF
Organic Tomatoes
lb
~~₹3.49~~
₹2.79 Stock: 40

20% OFF
Fresh Milk
gallon
~~₹3.99~~
₹3.19 Stock: 25


Checkout

Total Amount: ₹15.96


Checkout

Total Amount: ₹15.96

CHAPTER 11
KNOWN ISSUES

1. **Forgot Password Functionality:**
   - **Description:** The forgot password functionality does not include an email verification step to ensure the validity of the email address before sending the reset link.
   - **Impact:** If a user enters an incorrect email address, they will not receive any feedback, leading to confusion as the password reset link will not be delivered to them. This may cause frustration and unnecessary support requests from users.
   - **Resolution:** A verification step will be added to check the validity of the email address before sending the reset link. Additionally, error messages will be displayed to inform users if the entered email does not exist in the system.
2. **Cross-browser Compatibility:**
   - **Description:** Certain features, such as the image carousel and advanced filtering options, do not function properly in older browsers, including Internet Explorer and certain versions of Safari.
   - **Impact:** Users accessing the application through older or unsupported browsers may experience broken layouts, non-

functional features, or degraded performance, which could result in a poor user experience and loss of engagement.

- **Resolution:** A review and update of the application's CSS and JavaScript will be conducted to ensure better compatibility with older browser versions. Polyfills and fallbacks will be used where necessary to support outdated browsers and ensure consistent behavior across all platforms.

3. **Mobile Responsiveness:**
   - **Description:** Some sections of the application, such as the Admin Dashboard and product listings page, may not fully adapt to smaller screens on mobile devices.
   - **Impact:** Users accessing these sections on mobile devices may encounter issues with page layout, misaligned elements, or broken UI components, negatively affecting usability and accessibility.
   - **Resolution:** The application will undergo a thorough mobile responsiveness audit to ensure that all pages, including the Admin Dashboard and product views, adjust appropriately to various screen sizes. UI components will be redesigned for better alignment and ease of use on smaller devices.
4. **Authentication Token Expiry Handling:**
   - **Description:** The application does not consistently manage token expiry, leading to situations where users remain logged

in after their session has expired, or they are required to manually refresh the page to reauthenticate.

- **Impact:** This inconsistency may result in users experiencing unexpected logouts or access issues when attempting to interact with the application after their session has expired. Users may also face frustration if they are unable to access features or need to log in repeatedly.
- **Resolution:** The system will be updated to handle token expiry more consistently. This will include automatic redirection to the login screen when a token expires and the addition of clear messaging to guide users through the reauthentication process. Improvements will also ensure that users are logged out securely when their session ends.

CHAPTER 12
FUTURE ENHANCEMENTS

- **AI-Powered Personalization**
- **Smart Recommendations:** Implement machine learning algorithms that analyze past orders, browsing behavior, and user preferences to suggest menu items tailored to individual users. This will help increase user engagement and boost sales by offering personalized options.
- **Dynamic Promotions:** Utilize user data to deliver location-based, time-sensitive discounts and promotions. For example, a user might receive a discount for ordering lunch during specific hours or for ordering from a restaurant within their vicinity.
- **Order Prediction:** Leverage predictive analytics to offer users the ability to reorder their previous meals with a single click. This feature will enhance the convenience and speed of ordering for frequent users.
- **Real-Time Order Tracking with Live Updates**
- **Map-Based Tracking:** Integrate live GPS tracking so users can see the real-time location of their food delivery, providing transparency and reducing uncertainty.

- **ETA Updates:** Use real-time data to generate dynamic Estimated Time of Arrival (ETA) updates, adjusting based on traffic patterns, weather conditions, and order preparation times.
- **In-App Notifications:** Implement a notification system to keep users informed about each step of their order (e.g., order confirmed, preparing, out for delivery). This could be delivered via push notifications or SMS, keeping users updated and improving their overall experience.
- **Enhanced Customer Support**
- **In-App Chatbot:** Introduce an AI-driven chatbot capable of answering common customer queries, handling cancellations, and assisting with order-related questions. For more complex issues, the chatbot would escalate to live customer support agents.
- **Order Modification:** Allow users to modify their orders within a short time frame after placing them, such as adding items, modifying delivery details, or adjusting customization options.
- **Feedback and Issue Reporting:** Enable users to rate items individually, provide feedback on the ordering process, and report issues directly within the app. This will help improve

customer satisfaction and allow for quick resolution of problems.

- **Advanced Search and Filtering Options**
- **Voice Search:** Integrate voice search functionality to make it easier for users to find specific dishes or restaurants without needing to type.
- **Dietary and Allergy Filters:** Provide filters that allow users to search for dishes based on dietary preferences (e.g., vegan, gluten-free) and allergens (e.g., nuts, dairy).
- **Ingredient-Based Search:** Allow users to search for dishes that contain specific ingredients, making it easier to find meals that align with their preferences or dietary restrictions.
- **Group Ordering and Splitting Bills**
- **Group Orders:** Enable users to create group orders where multiple people can contribute to a shared cart. This will be particularly useful for office or family group orders, simplifying the process of ordering together.
- **Bill Splitting:** Implement a feature that allows users to split the total bill among multiple people, with the option for each person to pay via different payment methods. This feature will make group ordering more convenient and flexible.
- **Sustainable and Eco-Friendly Options**

- **Eco-Friendly Packaging Option:** Provide an option for users to select eco-friendly packaging for their orders, encouraging sustainability. This could include recyclable or biodegradable containers, reducing environmental impact.
- **Food Waste Management:** Offer discounts on items that are nearing expiration or create "chef’s special" options that help reduce food waste by utilizing surplus ingredients.
- **Carbon Footprint Tracking:** Include a feature that tracks the carbon footprint of each order and encourages users to choose meals and delivery options with lower environmental impact.
- **Health and Nutrition Tracking**
- **Caloric and Nutritional Information:** Display detailed nutritional facts and caloric breakdowns for each menu item to help users make informed decisions based on their dietary needs.
- **Health Goals Integration:** Allow users to set health goals (e.g., low-calorie, high-protein) and recommend menu items that align with those goals, offering a personalized health-conscious dining experience.
- **Integration with Fitness Apps:** Sync with popular fitness and health apps (e.g., Fitbit, MyFitnessPal) to allow users to track their calories, nutritional intake, and exercise regimen directly from their food orders.