



**JEPPIAAR**  
ENGINEERING COLLEGE

# **GROCERY WEB APP**

## **A PROJECT REPORT**

**Submitted By:-**

- 1. Priya Dharshini A(310821104071)**
- 2.Jyothi V(310821104044)**
- 3.Yamini M(310821104119)**
- 4. Priyanka P(310821104072)**
- 5.Muneeswaran M(310821104304)**

in partial fulfillment for the award of the degree

of

**BACHELOR OF ENGINEERING**

in

**COMPUTER SCIENCE AND ENGINEERING**

**JEPPIAAR ENGINEERING COLLEGE, CHENNAI**

**ANNA UNIVERSITY:: CHENNAI 600 025**

**NOVEMBER 2024**



**ANNA UNIVERSITY: CHENNAI 600 025**

## **BONAFIDE CERTIFICATE**

Certified that this project report “ **GROCERY WEB APP USING MERN** ” is the bonafide work of “ **YAMINI M(310821104119)** ” who carried out the project work for Naan Mudhalavan.

MENTOR

HEAD OF THE DEPARTMENT

DATE:- \_\_\_\_\_

INTERNAL:-

EXTERNAL:-

## TABLE OF CONTENTS

S.NO	CONTENTS	PAGE NO
1.	Introduction	4
2.	Project Overview	5
3.	Architecture	7
4.	Setup Instructions	9
5.	Folder Structure	12
6.	Running the application	13
7.	API Documentation	16
8.	Authentication	22
9.	Authorisation	24
10.	User Interface	28
11.	Testing	29
12.	Known Issues	30
13.	Future Enhancements	31

## **1.INTRODUCTION:-**

### **PROJECT TITLE:-“GROCERY WEB APP”**

The Grocery Web App is a modern, responsive, and feature-rich web application designed to streamline the online grocery shopping experience. Built using the MERN stack (MongoDB, Express.js, React.js, and Node.js), the application provides a seamless platform for users to browse, search, add to cart, and purchase groceries. The admin panel allows efficient inventory management, orders, and user accounts.

It provides features like Secure registration and login. Users can browse, search, and filter products by category, price, and ratings. They can also add/remove items from a shopping cart and checkout with real-time price calculation. Users can track order status and view order history. Users can also rate and review products.

### **TEAM MEMBERS:-**

1. PRIYADHARSHINI A: Worked on the backend for handling the application's server-side logic, data management, security, and business rules.
2. JYOTHI V: Worked on the front end, responsible for the visual and interactive part of the application, where users interact with the system. It ensures seamless navigation, responsiveness, and accessibility.
3. YAMINI M: Focused on designing the platform's interface and creating intuitive and user-friendly experiences.
4. PRIYANKA P: Worked on both frontend and backend to ensure the overall functionality of the application.
5. MUNEESWARAN M: Handled the backend API development with Node.js, Express.js, and MongoDB.

## **2.PROJECT OVERVIEW:**

### **PURPOSE:-**

The primary purpose of the Grocery Web App is to create an efficient and user-friendly platform that modernizes and simplifies the grocery shopping experience. This project seeks to address the common pain points associated with traditional grocery shopping, such as long queues, lack of availability of preferred products, and the inconvenience of remembering shopping lists. By leveraging technology, the web app aims to provide users with a seamless and personalized online grocery shopping solution.

### **Ease of Use:**

- Develop an intuitive user interface that makes it simple for users to browse, search, and add items to their cart.
- Offer a smooth onboarding process for first-time users, with helpful tutorials or guides.

### **Comprehensive Product Catalog:**

- Maintain a well-organized inventory that includes various product categories (e.g., fresh produce, packaged goods, household items).
- Provide detailed product information, including descriptions, nutritional facts, and pricing.

### **Efficient Shopping Experience:**

- Enable advanced search features, including filters for brands, price ranges, dietary restrictions, and availability.
- Allow users to create and manage shopping lists for faster reordering.
- Provide options for subscription services for frequently purchased items.

### **FEATURES:**

## 1. User-Friendly Interface

- Clean and intuitive design for seamless navigation.
- Multi-language support to cater to diverse user bases.

## 2. Product Catalog and Search

- Comprehensive product listings with detailed descriptions, pricing, images, and nutritional information.
- Advanced search with filters for categories, brands, price ranges, dietary preferences, and availability.
- Real-time inventory updates to reflect stock levels.

## 3. Shopping Cart and List Management

- Add, edit, or remove items from the cart easily.
- Save multiple shopping lists for frequent orders (e.g., "Weekly Groceries" or "Party Supplies").
- Smart cart suggestions based on past purchases.

## 4. Personalized Recommendations

- Tailored product suggestions based on purchase history and browsing habits.
- Custom alerts for discounts or promotions on frequently purchased items.

## 5. Flexible Checkout and Payment

- Multiple payment options, including credit/debit cards, digital wallets, and cash on delivery.
- Secure transactions with encryption for user data.

## 6. Delivery and Pick-Up Options

- Flexible delivery scheduling, including same-day or next-day options.

- Real-time tracking of delivery status and estimated arrival times.

## 7. Ratings and Reviews

- Product reviews and ratings for informed purchasing decisions.
- Option to share feedback on the shopping experience.

## **3.ARCHITECTURE:-**

- **FRONTEND:**

The front-end architecture leverages React, a component-based JavaScript library, to create a modular, scalable, and efficient user interface for the Grocery Web App.

### Core Components

- Header: Includes navigation, search bar, and user authentication links.
- Footer: Contains contact information, social media links, and additional resources.
- Sidebar: For category navigation and quick filters.

### 1.2. Feature-Specific Components

- Product List: Displays the grocery items, fetched dynamically from the API.
- Product Card: A reusable component for each item, showing image, name, price, and "Add to Cart" functionality.
- Shopping Cart: Manages selected items, quantity updates, and total price calculations.
- Wishlist: Allows users to save products for later.
- Checkout Form: Captures user details like address, payment information, and delivery preferences.

- **BACKEND:-**

The backend is designed with Node.js and Express.js, leveraging their efficiency and scalability to handle API requests, business logic, and database interactions.

## 1. Core Architecture Overview

The backend follows a modular and layered architecture to ensure:

- Separation of concerns.
- Reusability of components.
- Scalability and maintainability.

## 2. Folder Structure

src/

├— controllers/	# Handles business logic and processes incoming requests
├— routes/	# Defines application routes and API endpoints
├— models/	# Database models/schemas
├— services/	# Contains reusable business logic
├— middlewares/	# Middleware functions (e.g., authentication, validation)
├— utils/	# Utility functions (e.g., helpers for formatting or error handling)
├— config/	# Configuration files (e.g., database connection, environment variables)
├— tests/	# Unit and integration tests
└ app.js	# Main application entry point

## Database



- Database Choice: MongoDB (via Mongoose) for flexible, document-based storage.
- Models: Defined in models/ with Mongoose schemas. Example:  
productModel.js  
javascript

```
const mongoose = require("mongoose");  
  
const productSchema = new mongoose.Schema({  
  
  name: String,  
  
  price: Number,  
  
  category: String,  
  
  stock: Number,  
  
  description: String,  
  
});  
  
module.exports = mongoose.model("Product", productSchema);
```

#### **4.SETUP INSTRUCTIONS:-**

##### PREREQUISITES:

- 1.Node.js and npm: Install Node.js, which includes npm (Node Package Manager).
- 2.MongoDB
3. Express.js
- 4.Angular
- 5.HTML, CSS, and JavaScript

#### **Step-by-Step Installation Guide: Clone, Install Dependencies, and Set Up Environment Variables:-**

## 1. Clone the Repository

### 1. Install Git

Ensure Git is installed on the machine.

### 2. Clone the Repository

Open a terminal or command prompt and run the following command:

bash:

```
git clone <repository-url>
```

Replace <repository-url> with the GitHub (or other) repository URL for the Grocery Web App.

Example:

```
git clone https://github.com/username/grocery-webapp.git
```

### 3. Navigate to the Project Directory

After cloning, navigate into the project folder:

bash:

```
cd grocery-webapp
```

### 2. Install Dependencies:-

#### 2.1 Backend dependencies

##### 1. Navigate to the backend folder (if applicable):

```
cd backend
```

##### 2. Install the required Node.js packages:

```
npm install
```

#### 2.2 Frontend Dependencies

##### 1. Navigate to the frontend folder:-

```
cd ../frontend
```

##### 2. Install the required Node.js packages:-

```
npm install
```

### 3. Set Up Environment Variables:-

#### 3.1. Backend Environment Variables

Environment variables are stored in a .env file for security and flexibility.

##### 1. Navigate to the backend folder:

```
cd backend
```

##### 2. Create a .env file in the root of the backend directory:

```
touch .env
```

##### 3. Add the following variables to the .env file (replace placeholders with actual values):

```
PORT=5000
```

```
MONGO_URI=mongodb+srv://<username>:<password>@<cluster-  
url>/<database-name>?retryWrites=true&w=majority
```

```
JWT_SECRET=your_jwt_secret_key
```

```
NODE_ENV=development
```

##### 4. Save the file.

#### 3.2. Frontend Environment Variables:-

##### 1. Navigate to the frontend folder:

```
cd ../frontend
```

##### 2. Create a .env file in the root of the frontend directory:

```
touch .env
```

##### 3. Add the following variables (replace placeholders with actual values):

```
REACT_APP_API_BASE_URL=http://localhost:5000/api
```

##### 4. Save the file.

#### **4. Start the Application:-**

##### **4.1. Start the Backend Server**

1. Navigate to the backend folder:

```
cd backend
```

2. Start the backend folder:

```
npm start
```

- If using Nodemon (for hot reloading during development):

```
npm run dev
```

##### **4.2. Start the Frontend Server**

Navigate to the frontend folder:

```
cd ../frontend
```

1. Start the frontend React app:

```
npm start
```

#### **5. Access the Application**

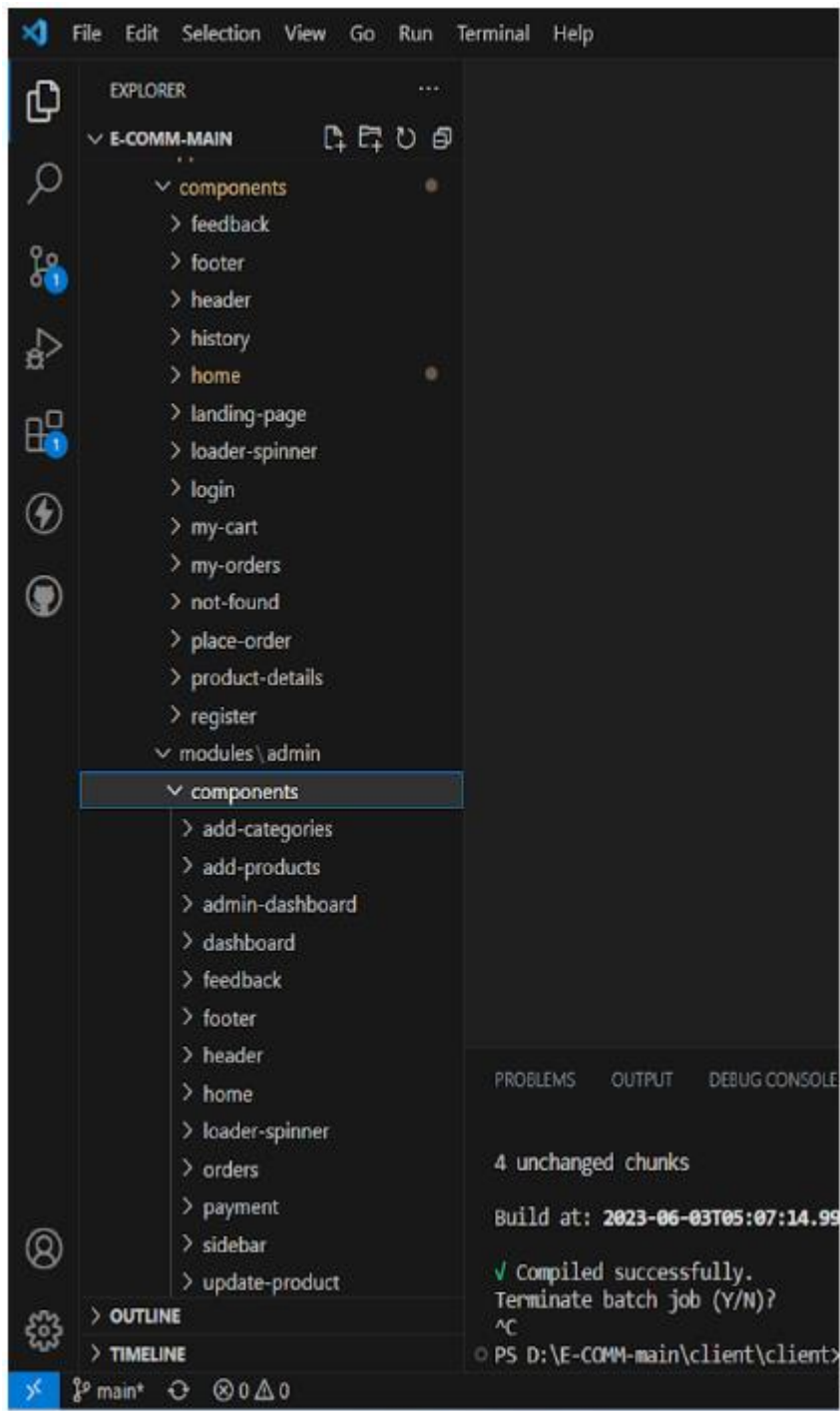
Open your browser and navigate to:

`http://localhost:3000`

The backend API will be running on:

<http://localhost:5000>

#### **5.FOLDER STRUCTURE:-**



## **6.RUNNING THE APPLICATION:-**

Step 1: Setup Frontend (Client):

1. Navigate to the client/ directory:

```
cd client
```

2. Initialize a new package.json for the frontend:

```
npm init -y
```

3. Update the client/package.json to include the following:-

```
{  
  "name": "client",  
  "version": "1.0.0",  
  "scripts": {  
    "start": "lite-server"  
  },  
  "devDependencies": {  
    "lite-server": "^2.6.1"  
  }  
}
```

4. Move back to the root directory:

```
cd..
```

Step 2: Setup Backend (Server):-

1. Navigate to the server/ directory:

```
cd server
```

2. Initialize a new package.json for the backend:

```
npm init -y
```

3. Add dependencies for the backend:

```
npm install express mongoose body-parser cors
```

4. Add dependencies for the backend:

```
npm install express mongoose body-parser cors
```

5. Update the server/package.json to include the following:

```
{  
  "name": "server",  
  "version": "1.0.0",  
  "scripts": {  
    "start": "node index.js"  
  },  
  "dependencies": {  
    "express": "^4.18.2",  
    "mongoose": "^7.3.2",  
    "body-parser": "^1.20.2",  
    "cors": "^2.8.5"  
  }  
}
```

5. Move back to the root directory:

```
cd..
```

Step 3:Starting the servers:

1.Start the frontend server:

```
cd client
```

```
npm start
```

2.Start the backend server:

```
cd server
```

```
npm start
```

This will start the Express server on <http://localhost:5000>.

## **7.API DOCUMENTATION:-**

BASE URL:

<https://api.groceryapp.com/v1>

### 1.User Registration

Endpoint: POST /auth/register

Description: Register a new user.

Headers: Content-Type: application/json

Request Body:

json

```
{  
  "name": "John Doe",  
  "email": "johndoe@example.com",  
  "password": "password123"  
}
```

Response:

- 201 Created
- 400 Bad Request

json

```
{  
  "error": "Email already exists."  
}
```

### 2.User Login



Endpoint: POST /auth/login

Description: Authenticate user and retrieve an access token.

Headers: Content-Type: application/json

Request Body:

json

```
{  
  "email": "johndoe@example.com",  
  "password": "password123"  
}
```

Response:

- 200 OK

json

```
{  
  "token": "your-access-token",  
  "user_id": "12345"  
}
```

- 401 Unauthorized

json

```
{  
  "error": "Invalid credentials."  
}
```

Product Management

3.Get All Products

Endpoint: GET /products

Description: Retrieve a list of available grocery products.

Headers: Authorization: Bearer {token}

Query Parameters:

- category (optional): Filter by product category.
- limit (optional): Number of items per page (default: 10).

Response:

200 OK

json:

```
{  
  "products": [  
    {  
      "id": "1",  
      "name": "Apple",  
      "price": 1.2,  
      "category": "Fruits",  
      "stock": 50  
    },  
    {  
      "id": "2",  
      "name": "Milk",  
      "price": 2.5,  
      "category": "Dairy",  
      "stock": 30  
    }  
  ],  
}
```

```
"total": 100,  
"page": 1,  
"limit": 10  
}
```

#### 4. Get product details:-

Endpoint: GET /products/{id}

Description: Retrieve detailed information about a product.

Headers: Authorization: Bearer {token}

Response:

- 200 OK

```
{  
  "id": "1",  
  "name": "Apple",  
  "description": "Fresh red apples.",  
  "price": 1.2,  
  "category": "Fruits",  
  "stock": 50  
}
```

- 404 Not Found

json

```
{  
  "error": "Product not found."  
}
```

#### 5. Add to cart:

Endpoint: POST /cart

Description: Add a product to the cart.

Headers: Authorization: Bearer {token}

Request Body:

```
{  
  "product_id": "1",  
  "quantity": 3  
}
```

Response:

- 200 OK

json

```
{  
  "message": "Product added to cart.",  
  "cart_id": "98765"  
}
```

- 404 Bad request

```
{  
  "error": "Insufficient stock."  
}
```

## 6. Remove from Cart

Endpoint: DELETE /cart/{id}

Description: Remove a product from the cart.

Headers: Authorization: Bearer {token}

Response:

- 200 OK

```
{  
  "message": "Product removed from cart."  
}
```

- 404 Not found:-

```
{  
  "error": "Cart item not found."  
}
```

## Order Management

### 7.Place Order

Endpoint: POST /orders

Description: Place an order for items in the cart.

Headers: Authorization: Bearer {token}

Request Body:

```
{  
  "address": "123 Main St, City, Country",  
  "payment_method": "credit_card"  
}
```

Response:

- 201 Created

```
{  
  "message": "Order placed successfully.",  
  "order_id": "54321",  
  "total": 35.6
```

}

## 8. AUTHENTICATION:-

When implementing authentication for a grocery web app, you need a secure system to manage user sign-up, login, and account management. Here's a general outline of how to set up authentication:

### 1. User Registration (Sign-Up)

- Collect basic information from users (e.g., name, email, password).
- **Password Handling:** Use bcrypt or Argon2 to hash passwords before storing them in the database to prevent plaintext password storage.
- **Email Verification:** Send a confirmation email with a verification link to ensure that the provided email is valid.

### 2. User Login

- **Login Form:** Users provide their email and password.
- **Password Verification:** Compare the hashed password stored in the database with the one provided by the user using bcrypt or Argon2.
- **Session or Token-based Authentication:**
  - **Session-based Authentication:** Store user session information (e.g., in a cookie). This is typically used for traditional web applications.
  - **Token-based Authentication:** Use JWT (JSON Web Tokens) for modern, stateless applications (often with mobile or API backends). The user is issued a token after logging in, which they include in requests to authenticate.

### 3. Password Recovery

- Allow users to reset their passwords if they forget them. This usually involves:
  - Asking for the email address.

- Sending a password reset link to the user's email.
- Allowing the user to choose a new password.

#### 4. User Authorization (Role Management)

- **Admin vs. User Roles:** If your app has different user roles (e.g., admin, regular user), implement role-based access control (RBAC).
- For example, admins can manage product listings, while regular users can place orders.

#### 5. Account Management

- Allow users to update their profile (email, password, etc.).
- Add options to deactivate or delete an account.

#### 6. Security Considerations

- **HTTPS:** Ensure the entire web app runs over HTTPS to encrypt communication.
- **Session Expiration:** Sessions or tokens should expire after a certain period for security.
- **Two-Factor Authentication (2FA):** For enhanced security, implement 2FA, where the user must provide a second verification method (e.g., code sent via email or SMS).
- **Rate Limiting:** Protect endpoints from brute-force attacks by limiting the number of login attempts.

#### Example of a Basic Authentication Flow (JWT)

1. **User logs in:** Sends email and password to the backend.
2. **Backend validates credentials:** If correct, issue a JWT token.
3. **User receives JWT:** Store this token on the client side (usually in localStorage or sessionStorage).

4. **User makes authenticated requests:** Include the JWT in the Authorization header as a Bearer token for protected API routes.

## 7. Tech Stack Examples

- **Backend:** Node.js with Express, Django, or Flask.
- **Frontend:** React or Angular for SPA (Single Page Application), or traditional HTML forms.
- **Database:** MySQL, PostgreSQL, or MongoDB for user data.
- **Authentication Libraries:**
  - For Node.js: passport.js, jsonwebtoken, bcryptjs.

## 9.AUTHORISATION:-

Authorization in a grocery web app is about controlling what authenticated users are allowed to do within the app. It ensures that users can only access certain resources or perform certain actions if they have the necessary permissions.

Here's how you can structure authorization for your grocery web app:

### 1. Define Roles and Permissions

Before implementing authorization, define the roles and permissions within your app. Common roles for a grocery app could include:

- **Admin:** Full access to the app, including product management, user management, order management, and settings.
- **Customer:** Can browse products, add items to the cart, and place orders.
- **Guest:** A user who is not logged in but can browse products (usually with limited access).

Each role has a set of permissions associated with it. For instance:

- **Admin Permissions:**



- View, add, edit, and delete products.
- View and manage all customer orders.
- View and manage user accounts (including the ability to promote/demote users).
- **Customer Permissions:**
  - View product details.
  - Add products to the cart.
  - Place orders.
  - View order history.
- **Guest Permissions:**
  - View product listings (but cannot add items to the cart or place orders).

## 2. Role-Based Access Control (RBAC)

The simplest and most common approach to authorization is **Role-Based Access Control (RBAC)**, where each user is assigned a role (e.g., admin, customer), and access to resources or actions is based on that role.

Here's how to implement RBAC:

### 1. User Role Assignment:

- When a user signs up, you can assign them a default role (e.g., customer).
- Admins may manually assign roles or use a specific sign-up process for admin users.

### 2. Protect Routes Based on Roles:

- When the user tries to access a specific route or perform an action, check if they have the necessary permissions based on their role.

### 3. Example Flow for Admin Authorization:

- **Route Protection:** Protect routes like adding or deleting products with an authorization check:

javascript

```
app.post('/admin/products', (req, res) => {  
  if (req.user.role !== 'admin') {  
    return res.status(403).send('Access Denied');  
  }  
  // Allow product creation  
});
```

- **Admin Dashboard:** Only admins should be able to view the dashboard to manage users, orders, and products:

Javascript:

```
app.get('/admin/dashboard', (req, res) => {  
  if (req.user.role !== 'admin') {  
    return res.status(403).send('Access Denied');  
  }  
  // Render admin dashboard  
});
```

### Best Practices for Authorization

- **Least Privilege:** Grant users the minimum level of access they need to perform their tasks.
- **Token-based Access:** If using JWT tokens, embed the user's role or permissions in the token. This helps to manage authorization without needing to query the database on every request.

Example:

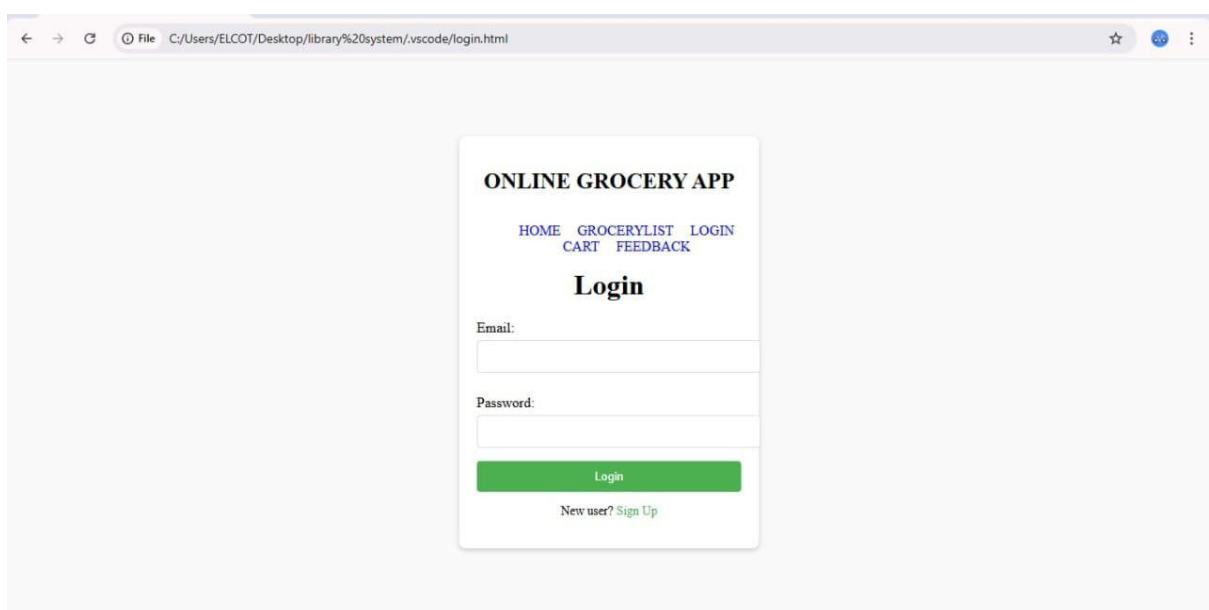
javascript

```
const jwt = require('jsonwebtoken');
```

```
function generateAuthToken(user) {  
  const token = jwt.sign(  
    { userId: user.id, role: user.role },  
    'your_jwt_secret_key',  
    { expiresIn: '1h' }  
  );  
  return token;  
}
```

## 10.USER INTERFACE:-

### LOGIN PAGE:-



The screenshot shows a web browser window with the address bar displaying 'File C:/Users/ELCOT/Desktop/library%20system/vscode/login.html'. The page content is a login form for an 'ONLINE GROCERY APP'. The form has a title 'ONLINE GROCERY APP' and a navigation menu with links: 'HOME', 'GROCERYLIST', 'LOGIN', 'CART', and 'FEEDBACK'. Below the menu is a 'Login' heading. The form contains two input fields: 'Email:' and 'Password:'. A green 'Login' button is positioned below the password field. At the bottom of the form, there is a link that says 'New user? Sign Up'.

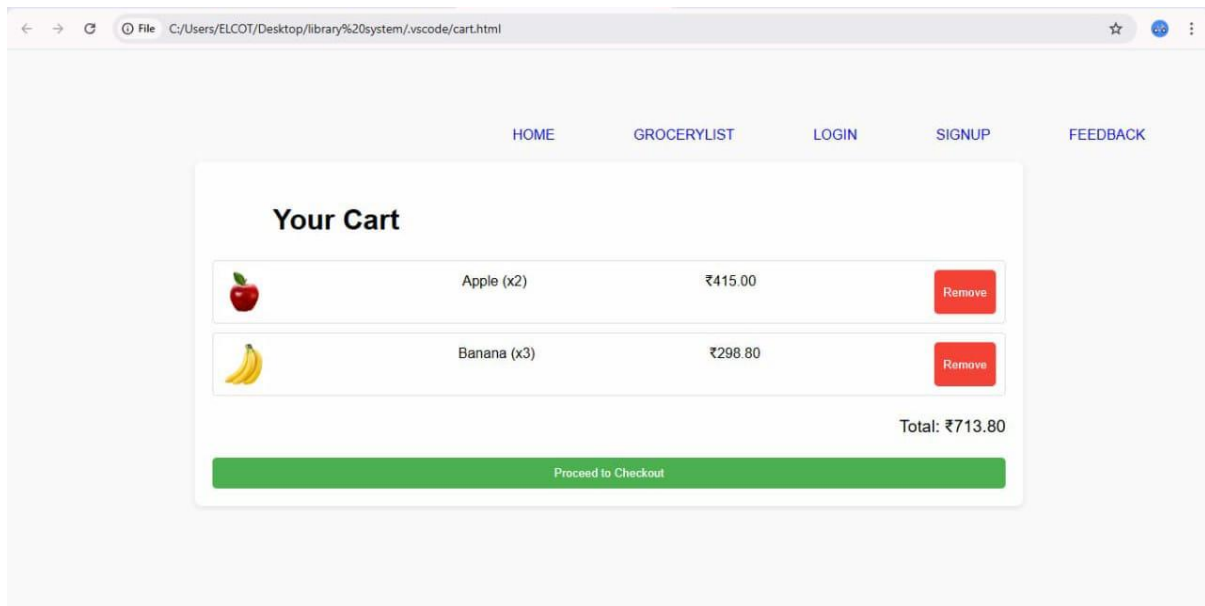
## SIGNUP PAGE:-

The screenshot shows a web browser window with the address bar displaying 'File C:/Users/ELCOT/Desktop/library%20system/vscode/signup.html'. The page title is 'ONLINE GROCERY APP'. Below the title is a navigation bar with links: HOME, GROCERYLIST, LOGIN, CART, and FEEDBACK. On the right side, there is a 'Sign Up' section with three input fields labeled 'Full Name:', 'Email:', and 'Password:'. Below these fields is a green button labeled 'Create New Account'.

## HOME PAGE:-

The screenshot shows a web browser window with the address bar displaying 'File C:/Users/ELCOT/Desktop/library%20system/vscode/index.html'. The page title is 'ONLINE GROCERY APP'. Below the title is a navigation bar with links: HOME, GROCERYLIST, LOGIN, CART, and FEEDBACK. The 'HOME' link is highlighted. Below the navigation bar, there is a 'Grocery List' section. It includes a label 'Items:' followed by a dropdown menu showing 'Milk'. Below this is an 'Add Item' button.

## CART PAGE:-



## 11. TESTING:-

### 1. Unit Testing

Validation of business logic (e.g., calculating discounts, checking stock availability).

API endpoint responses for correct inputs and errors.

Tools:

Jest (JavaScript-based apps)

JUnit (Java-based backends)

### 2. Integration Testing

Communication between the frontend and backend (e.g., API calls to fetch product lists or update cart).

External services like payment gateways, notification systems, or inventory management.

Tools:

Postman (for API testing)

### 3. Security Testing

Identify vulnerabilities in the application.

Secure handling of user data (e.g., encryption, secure session management).

Preventing SQL injection, XSS, CSRF, and other common vulnerabilities.

Tools:

OWASP ZAP

Burp Suite

## **12.KNOWN ISSUES:-**

Despite careful planning and development, grocery web apps may encounter various challenges that affect functionality, user experience, and scalability.

Below are common issues categorized by technical, user experience, and business aspects.

### **Data Integrity Issues**

- **Inventory Mismatch:** Products may show as "in stock" when they are not due to delayed inventory updates.
- **Order Duplication:** Race conditions during checkout can cause duplicate orders.

### **Payment Gateway Problems**

- **Transaction Failures:** Errors during payment processing can lead to incomplete transactions.
- **Security Vulnerabilities:** Improper implementation of secure protocols (e.g., HTTPS, tokenization) might expose sensitive payment details.

### **API Failures**

- **Backend Downtime:** If the server crashes or APIs fail, the app may become unusable.
- **Third-Party Dependencies:** Issues with integrated services (e.g., payment gateways, SMS notifications) can disrupt functionality.

## Authentication and Security

- **Weak Authentication:** Poor password policies or token mismanagement may lead to account breaches.
- **Data Breaches:** Insufficient encryption of sensitive user data (e.g., addresses, payment details) can lead to leaks. **Customer Support**
- **Delayed Responses:** Lack of real-time support for order-related issues can harm customer satisfaction.

## 13.FUTURE ENHANCEMENTS:-

As the grocery web app continues to grow and evolve, there are various areas where future enhancements can improve user experience, performance, and scalability. These improvements can cater to both business needs and technical advancements. Below are potential enhancements that can be considered for the app's future development:

### Voice Search

- **Description:** Implement a voice-based search feature that allows users to search for products using voice commands. This can enhance accessibility for users who are unable to type or prefer voice interaction.
- **Benefits:** Improves accessibility and convenience, making the app more user-friendly.

### Real-Time Chatbot Support

- **Description:** Integrate a chatbot that can help users find products, answer FAQs, or assist with order-related issues in real-time.
- **Benefits:** Reduces customer service workload, increases responsiveness, and improves user satisfaction

### Push Notifications for Offers and Updates

- **Description:** Implement push notifications to inform users of offers, discounts, new arrivals, or the status of their orders.

- **Benefits:** Increases user engagement, boosts sales during promotions, and keeps customers informed.

### **Corporate and Bulk Orders**

- **Description:** Introduce functionality for businesses or organizations to place bulk orders, manage multiple accounts, and track corporate deliveries.
- **Benefits:** Opens new business opportunities, increases average order value, and attracts B2B customers.