GROCERY WEBAPP

THE PROJECT REPORT

Submitted by

VINSHI.J

PAVITHRA MANIKANDAN

VIJAYASARATHY.PS

SANTHAKUMAR.R

In partial fulfillment for the award of the degree of

BACHELOR OF ENGINEERING

In

COMPUTER SCIENCE AND ENGINEERING

JEPPIAAR ENGINEERING COLLEGE

ANNA UNIVERSITY : CHENNAI 600 025

ANNA UNIVERSITY : CHENNAI 600 025

BONAFIDE CERTIFICATE

Certified that this project report "**GROCERY WEB APPLICATION**" is the bonafide work of "_____" who carried out the project work under my supervision.

SIGNATURE: SIGNATURE:

HEAD OF THE DEPARTMENT SUPERVISOR

TABLE OF CONTENTS:

7. API Documentation

7.1 GET/API/PRODUCTS

7.2 POST/API/CART

7.3 POST/API/ORDERS

8. Authentication

8.1 Authentication

8.2 Authorization

8.3 Using Middleware for JWT and Authorization

8.4 Security Enhancements

9. User Interface

9.1 UI-User Interface

10. Testing

10.1 Testing

11. Screenshots or Demo

11.1 Output Screenshots

11.2 Demo link

12. Known Issues

12.1 Issues and Solutions

13. Future Enhancements

13.1 Future Improvements

1. INTRODUCTION:

1.1 Project Title: GROCERY WEBAPP

A Grocery Web Application is an online platform that allows users to browse, select, and purchase grocery items from a wide range of categories. It typically includes features like product search, filtering by price and category, a shopping cart, secure checkout options, and delivery tracking. Customers can shop for everyday essentials, place orders for home delivery or pickup, and manage their accounts for a seamless, convenient grocery shopping experience.

1.2 Team Members:

- *Vinshi J – Team Lead*

  The team lead has effectively managed communication, time, and tasks throughout the project while contributing to both front-end and back-end development. They ensured clear communication among team members and kept the project on schedule. The team lead actively participated in coding, overseeing both the user interface and server-side functionality. Their strong leadership and technical skills helped maintain project quality, resolve issues, and ensure timely deliver.
- *Pavithra Manikandan - Frontend Developer*

  The front-end developer was responsible for designing and implementing the user interface, ensuring an intuitive and responsive experience across devices. She worked with HTML, CSS, and JavaScript (or relevant front-end frameworks like React or Angular) to build interactive features, such as product browsing, search functionality, and a smooth checkout process. Their focus was on creating a user-friendly, visually appealing design that aligned with the project's goals, ensuring ease of navigation and an optimal shopping experience for customers.
- *Vijayasarathy PS - Backend Developer*

  The back-end developer was responsible for building and maintaining the server-side logic, databases, and APIs that power the application. He worked with technologies like Node.js, React.js, or Mongo database to manage user authentication, product data, inventory, and order processing. By ensuring seamless integration between the front-end and the back end, they focused on optimizing performance, ensuring data security, and enabling real-time updates, ultimately providing a smooth and reliable experience for users and admins alike.
- *Santhakumar R - UIUX developer*

  The UI/UX designer was responsible for creating an intuitive, user-friendly interface and ensuring a seamless user experience. he designed the layout, navigation, and visual elements, focusing on ease of use, accessibility, and an attractive, responsive

design. Their goal was to optimize the shopping experience, making it easy for users to browse, select products, and complete purchases smoothly across all devices.

2. PROJECT OVERVIEW:

2.1 Purpose and Goals:

2.1.1 Purpose of the Project

The primary purpose of the "Grocery Web App" is to streamline the process of purchasing groceries by providing an efficient, user-friendly online platform. The app aims to connect customers with a wide range of grocery products, enabling them to browse, select, and order items from the comfort of their homes. It seeks to enhance convenience, reduce shopping time, and offer a seamless experience through features like product categorization, search functionality, and easy payment methods. By incorporating modern technology, the app serves as a solution to the challenges of traditional shopping, such as long queues, limited product availability, and geographical constraints.

2.1.2 Goals of the Project

1. Enhanced User Convenience:

Provide an intuitive and accessible platform that simplifies grocery shopping for users of all age groups. The app will allow customers to shop at any time and from anywhere, eliminating the need for physical store visits.

2. Efficient Order Management:

Develop features for smooth order processing, including cart management, order tracking, and delivery scheduling. This will ensure a hassle-free shopping experience.

3. Broad Product Availability:

Offer a diverse range of products, including fresh produce, packaged goods, and household essentials, catering to varied customer needs and preferences.

4. Personalization and Recommendations:

Utilize data-driven algorithms to provide personalized product recommendations and offers based on users' shopping history, enhancing customer satisfaction**.**

5. Cost and Time Savings:

Help users save money through discounts, deals, and loyalty programs, while also minimizing the time spent on grocery shopping.

6. Business Growth and Scalability:

Build a scalable platform that supports vendor partnerships, enabling the onboarding of local and regional grocery stores. This fosters business growth while offering more variety to customers.

7. Sustainability:

Promote eco-friendly practices, such as paperless receipts and minimal packaging, contributing to a greener environment.

The "Grocery Web App" is designed to bridge the gap between technology and daily essentials, making grocery shopping a more efficient and enjoyable experience for users.

2.2 Features:

2.2.1 User Features

1.User Registration and Login

- Secure account creation and login system.
- Option for social media or Google account integration.

2. Product Browsing and Search

- Organized categories for groceries (e.g., fruits, vegetables, dairy).
- Advanced search with filters for price, brand, or availability.

3. Smart Shopping Cart

- Add, remove, or update items in the cart.
- Display estimated total price, taxes, and delivery fees.

4. Personalized Recommendations

- Suggestions based on purchase history and preferences.
- Display top deals and trending products.

5. Multiple Payment Options

- Support for credit/debit cards, UPI, e-wallets, and cash on delivery.
- Secure and encrypted payment gateway integration.

6. Order Tracking and Management

- Real-time order status updates (e.g., processed, shipped, out for delivery).
- Option to cancel or reschedule orders.

7. Loyalty Program and Discounts

- Reward points for purchases.
- Regular deals, promo codes, and exclusive discounts.

2.2.2 Admin Features

1. Product Management

- Add, update, or delete products with details like price, stock, and images.
- Manage product categories and subcategories.

2. Order and Customer Management

- Track customer orders and resolve disputes.
- Access customer purchase history for better service.

3. Inventory Management

- Real-time stock updates and alerts for low inventory.
- Bulk inventory uploads for large-scale operations

5. Marketing and Promotions

- Manage promotions, banners, and discount campaigns.
- Push notifications and email campaigns for marketing.

2.2.3 Vendor Features

1. Vendor Registration and Dashboard

- Allow grocery store owners to register and list their products.
- Dedicated dashboard to manage orders and payments.

2. Performance Analytics

- Access sales data and product performance metrics.

3. Communication Tools

- In-app messaging for customer support or order updates.

3. ARCHITECHTURE:

3.1 Frontend:

The front-end architecture of the Grocery Web App defines how the user interface (UI) components are structured, designed, and interact with the back-end services.

Key Components of the Front-End Architecture:

1. Technology Stack

- Framework/Library: Popular choices like React.js, Angular, or node.js for building dynamic, interactive user interfaces.

- HTML, CSS, and JavaScript: Core technologies for structuring, styling, and enabling functionality on the web pages.

- CSS Frameworks: Tools like Bootstrap or Tailwind CSS for responsive and consistent UI design.

2. Component-Based Architecture

- The UI is divided into reusable components such as Header, Footer, Product Card, Cart, and Search Bar.

- Each component handles a specific function and can be independently developed and maintained.

3. State Management

- Use of libraries like Redux (for React) or Vuex (for Vue.js) to manage the application's global state.

- Ensures consistency across components for actions like adding items to the cart or updating user preferences.

4. Routing

- Implemented using libraries like React Router or Vue Router to create a single-page application (SPA) experience.

- Allows smooth navigation between pages (e.g., Home, Product Details, Cart) without reloading the entire page.

5. API Integration

- Front-end communicates with the back-end using RESTful APIs or GraphQL.

- Fetches data such as product listings, user details, and order statuses dynamically.

6. Responsive Design

- Ensures the web app is accessible and user-friendly across devices (desktops, tablets, and mobile phones).

- Uses media queries and responsive grids for layout adaptability.

7. Error Handling and Notifications

- Displays user-friendly error messages for failed actions (e.g., payment issues, network errors).

- Real-time notifications for actions like order confirmations or updates**.**

8. Performance Optimization

- Techniques like lazy loading (loading images and components only when needed) and code splitting to enhance speed.

- Use of Content Delivery Networks (CDNs) to serve static assets like images and stylesheets faster.

9. **Security Features**

- Protection against cross-site scripting (XSS) and other vulnerabilities.

- Use of HTTPS to secure communication and cookies for session management.

3.2 Backend:

The back-end architecture of a Grocery Web App is responsible for handling data, business logic, user requests, and communication with the database and front-end.

1. Server-Side Framework

The back-end is built using a server-side framework like Node.js, React.js, Mongo db depending on the chosen technology stack.

2. Database Layer

- *Database Management System (DBMS):*

  The back-end interacts with databases to store and retrieve data. Common choices include:

  - Relational Databases (SQL): MySQL or PostgreSQL for structured data like product details, user accounts, and orders.

  - NoSQL Databases: MongoDB or Firebase for flexible storage of dynamic data like user preferences

3. API Layer

- RESTful APIs or GraphQL APIs serve as a communication bridge between the front-end and back-end, allowing users to interact with the app.

- Example API Endpoints:

- GET /products to fetch product details.
- POST /cart to add items to the shopping cart.
- GET /orders/{id} to retrieve order status.

4. Business Logic Layer

This layer handles the core functionalities of the app, such as:

- Validating user inputs (e.g., checking if payment details are valid).
- Processing business rules (e.g., applying discounts).
- Managing inventory updates after a purchase

5. Authentication and Security

i. *Authentication Systems:*

Handles user login and registration using JWT (JSON Web Tokens) or OAuth**.**

ii. *Security Measures:*

- Encryption of sensitive data like passwords and payment details.

- Prevention of common vulnerabilities like SQL Injection, XSS, and CSRF.

3.3 Database:

The database architecture for a Grocery Web App is designed to efficiently manage data related to users, products, orders, and transactions.

1. **Database Type**

- Relational Database Management System (RDBMS): Suitable for structured data, using tables with relationships (e.g., MySQL, PostgreSQL).

- Optional NoSQL Database: For scalability and flexibility in handling unstructured data, like product images or logs (e.g., MongoDB).

2. Key Tables and Their Structure

Users Table stores user information.

- Fields:

- user_id (Primary Key)
- name, email, password_hash
- phone, address, user_role (e.g., customer, admin)
- created_at, updated_at

Products Table

Manages product details.

- Fields:

- product_id (Primary Key)
- name, description, category_id (Foreign Key)
- price, stock, image_url
- vendor_id (Foreign Key), created_at, updated_at

Categories Table

Organizes products into categories.

- Fields:

- category_id (Primary Key)
- name, description

Orders Table

Tracks customer orders.

- Fields:

- order_id (Primary Key)
- user_id (Foreign Key)
- order_status (e.g., pending, shipped, completed)
- total_price, order_date, delivery_date

Order Items Table

Details products within an order.

- Fields:

- order_item_id (Primary Key)
- order_id (Foreign Key), product_id (Foreign Key)
- quantity, price_per_item

Payments Table

Logs payment transactions.

- Fields:

- payment_id (Primary Key)
- order_id (Foreign Key)
- payment_method, amount, status, payment_date

Vendors Table

Stores vendor-specific details.

- Fields:

- vendor_id (Primary Key)
- name, email, phone, address

3. Relationships Between Tables

a. Users ↔ Orders: A user can place multiple orders (1-to-many).
b. Orders ↔ Order Items: An order contains multiple items (1-to-many).
c. Products ↔ Categories: Each product belongs to a category (many-to-1).
d. Orders ↔ Payments: Each order is associated with one payment (1-to-1).
e. Vendors ↔ Products: Vendors supply multiple products (1-to-many).

4. Features of the Database Design

a. *Normalization:* Data redundancy is minimized using normalization techniques.
b. *Indexing:* Frequently queried fields (e.g., user_id, product_id) are indexed for faster access.
c. *Scalability:* Can handle increased users/products with replication and sharding.
d. *Data Security:* Passwords are encrypted, and sensitive data is stored securely.
e. *Backup and Recovery:* Regular backups ensure data integrity and availability during failures.

4. SETUP INSTRUCTIONS:

4.1 Prerequisites:

To set up a Grocery Web Application you'll need a combination of development tools, frameworks, and dependencies.

1. Development Environment:

A computer with a modern operating system (Windows, macOS, Linux)**.**

2. Platform Overview:

The platform operates through three primary layers:

→ *USER INTERFACE (FRONTEND):* Built with React, this layer manages user interactions and client-side rendering.
→ *SERVER-SIDE LOGIC (BACKEND):* Implemented using Node.js and Express.js, it handles request processing, business logic, and database connectivity.
→ *DATA STORAGE (DATABASE):* Mongo DB is used to store data related to customers, products, orders and inventory management.

3. Frameworks & Libraries:

i. *FRONTEND FRAMEWORK*:
- *Modular Design:* React components enable reusability and structured development.
- *State Control:* Redux is utilized for handling the applications global state effectively.
- *Navigation:* Seamless routing is achieved with React Router.

ii. *BACKEND STRUCTURE:*

- *API Development:* REST APIs are designed for smooth client-server communication.
- *Middleware Functions:* Responsible for handling authentication and data validation tasks

iii. *DATABASE SCHEMA:*

- *User Data:* Profiles are stored in the user's collection.
- *Product Information:* The products collection holds details of product names, categories, prices, and descriptions.
- *Order Tracking:* Order and delivery records are kept in the orders collection.

4. Tools:

- Code Editor/IDE: Visual Studio Code.
- Version Control: Git and GitHub/GitLab for managing your codebase.

5. Other Requirements:

- Web browser for testing.
- Basic understanding of REST APIs if applicable.

4.2 Setup Instructions

Before starting the development of a GROCERY WEBAPP using the **MERN** stack (MongoDB, Express.js, React, Node.js), ensure the following prerequisites are met. These include the software, tools, and basic knowledge required for a smooth development process.

1. Software Prerequisites:

a. Node.js

Why Required?

Node.js is the runtime environment for executing JavaScript on the server-side. It is essential for running the backend services of the platform and managing dependencies using npm (Node Package Manager).

Installation:

Download and install Node.js from Node.js Official Website

b. MongoDB

Why Required?

MongoDB is a NoSQL database used for storing and managing the application data. It provides a flexible schema design, making it ideal for handling various types of data, such as user details, courses, and progress tracking.

Installation:

Download and install MongoDB Community Edition from MongoDB Official Website Alternatively, use a cloud-based service like MongoDB Atlas for hosting the database online.

c. Code Editor (e.g., VS Code)

Why Required?

A code editor like Visual Studio Code helps in writing, editing, and managing the codebase efficiently. It offers features like syntax highlighting, debugging, and integration with version control.

Installation:

Download and install Visual Studio Code from VS Code Official Website

d. Git and GitHub

Why Required?

Git is used for version control, enabling collaboration and maintaining code history. GitHub (or any similar platform) is used for hosting the repository and facilitating team collaboration.

Installation:

Download Git from Git Official Website and create a GitHub account.

e. Web Browser (e.g., Google Chrome)

Why Required?

A modern web browser is necessary for testing and debugging the frontend application. Tools like Chrome DevTools assist in inspecting and debugging the HTML, CSS, and JavaScript code.

f. Postman or Similar API Testing Tool

Why Required?

Postman is used for testing backend APIs during development. It helps validate API endpoints, request/response formats, and error handling.

Installation:

Download Postman from Postman Official Website

2. Knowledge Prerequisites

a. Basics of HTML, CSS, and JavaScript

Why Required?

To build and design the frontend of the application, you need to understand:

- HTML: For structuring the content.
- CSS: For styling and creating responsive designs.
- JavaScript: For adding interactivity to the user interface.

b. Understanding of MERN Stack

- MongoDB: Basic knowledge of creating, querying, and managing databases.
  Example: Using commands like find(), insertOne(), or connecting via Mongoose.
- Express.js: Familiarity with setting up a Node.js server and defining RESTful API routes.
  Example: app.get('/api/courses', (req, res) => res.send(courses));.
- React.js: Understanding of React components, state, props, and hooks for building interactive UIs.
  Example: useState for managing local component state and useEffect for lifecycle methods.
- Node.js: Knowledge of creating a server, handling requests, and interacting with the database. Example: Setting up middleware with app.use().

c. RESTful API Design

Why Required?

To design and implement API endpoints for communication between the frontend and backend.

d. Git Version Control

Why Required?

To track code changes, work collaboratively, and manage branches during development.

e. Responsive Design and Cross-Browser Compatibility

Why Required?

To ensure the platform works seamlessly across all devices and browsers.

f. Basic Understanding of Authentication and Authorization

Why Required?

To implement user authentication (e.g., login, signup) and protect API routes using JWT or similar technologies.

3. System Requirements

- Operating System: Windows, macOS, or Linux.
- Processor: Minimum dual-core (quad-core recommended).
- RAM: 8GB (minimum), 16GB or more for better performance.
- Disk Space: At least 10GB for installing software and storing project files.

4. Development Workflow Prerequisites:

- Package Managers: Familiarity with npm or yarn for installing project dependencies. Example: npm install to install dependencies from package.json.
- Environment Variables: Knowledge of setting up .env files to securely manage sensitive information like database credentials, API keys, and JWT secrets.
- Error Handling and Debugging: Ability to use tools like Chrome DevTools, console logs, and Node.js debugging tools.

4.3 Installation:

The installation process involves setting up the project on your local machine, installing necessary dependencies, and configuring the environment.

1. Prerequisites

Check Ensure the following are installed on your system:

- Node.js (download from Node.js)
- MongoDB (download from MongoDB)

- Git (download from Git)
- A code editor, such as Visual Studio Code (download from VS Code)

2. Clone the Repository

i. Open your terminal or command prompt.
Navigate to the directory where you want to set up the project:
cd /path/to/your/project/directory.

ii. Clone the repository using Git:
git clone
https://github.com/your-repo/Grocery-MERN.git

iii. Navigate into the project folder:
cd Grocery-MERN

3. Install Dependencies

a. Backend Setup

Navigate to the server folder:

cd backend

1. Install the backend dependencies:

npm install

b. Frontend Setup

Navigate to the client folder:

cd frontend

1. Install the frontend dependencies:

npm install

4. Configure Environment Variables

a. Backend Environment (.env File)

Navigate to the server folder:

Cd backend

1. Create a .env file in the server directory:

bash

Copy code

touch .env

2. Add the following configuration details to the .env file:

PORT=5000

MONGO_URI=mongodb://localhost:27017/Grocery-MERN

JWT_SECRET=your_jwt_secret

b. Frontend Environment (.env File)

Navigate to the client folder:

cd frontend

1. Create a .env file in the client directory:

touch .env

2. Add the following configuration details to the .env file:

REACT_APP_API_URL=http://localhost:5000/api

5. Start the Application

a. Start the Backend Server

Navigate to the server folder:

cd backend

1. Start the server:

npm start

2. Confirm the server is running on http://localhost:5000.

b. Start the Frontend Server

1. Open a new terminal.

Navigate to the client folder:

cd /path/to/Grocery-MERN/frontend

2. Start the React development server:

bash

Copy code

npm start

3. Confirm the frontend is running on http://localhost:3000.

6. Access the Application

1. Open your web browser.
2. Visit http://localhost:3000 to view the frontend.
3. Verify that the backend API is working by visiting http://localhost:5000/api.

7. Testing the Setup

1. Check if the frontend communicates with the backend (e.g., user registration or product listing functionalities).
2. Use Postman or cURL to test API endpoints on the backend.

8. Optional: Run MongoDB

Ensure MongoDB is running on your local machine:

Mongod

Alternatively, if using MongoDB Atlas, ensure your MONGO_URI in the .env file is correctly set up.

9. Common Issues and Fixes

→ Port Conflict: If 5000 is in use, update the. env or configuration files to use a different port.

→ Missing Dependencies: Re-run npm install in both server and client folders.

→ MongoDB Not Running: Ensure MongoDB is running locally or that the connection string points to a valid database.

5. FOLDER STRUCTURE:

This structure maintains a clear separation between client and server functionality, with directories for components, routes, models, controllers, and configurations in the backend, and reusable React components organized by role in the frontend. Let me know if you need further clarification or adjustments!

FRONTEND

```
frontend/
├── public/
|   └── index.html          # Main HTML file for the React app
├── src/
|   ├── assets/             # Images, icons, and other static assets
|   ├── components/
|   |   ├── admin/          # Components for admin functionality
|   |   |   ├── AdminHome.jsx
|   |   |   └── AllProducts.jsx
```

```
|   |   ├── common/          # Components shared across roles
|   |   |   ├── Header.jsx
|   |   |   ├── Footer.jsx
|   |   |   ├── Navbar.jsx
|   |   |   └── Login.jsx
|   |   ├── customer/        # Components for customer functionality
|   |   |   ├── ProductList.jsx
|   |   |   ├── Cart.jsx
|   |   |   └── CustomerHome.jsx
|   |   └── seller/          # Components for seller functionality
|   |       ├── AddProduct.jsx
|   |       └── SellerHome.jsx
|   ├── pages/               # Pages for different routes
|   |   ├── HomePage.jsx
|   |   ├── AboutPage.jsx
|   |   ├── ContactPage.jsx
|   |   ├── CartPage.jsx
|   |   └── ProductPage.jsx
|   ├── redux/               # Redux store and slices
|   |   ├── store.js
|   |   ├── productSlice.js
|   |   └── userSlice.js
|   ├── App.js               # Main app component
|   ├── App.css              # Global styles
|   ├── index.js             # Main entry point for React app
|   └── package.json         # Frontend dependencies and scripts
```

BACKEND

backend/

```
├── config/
│   └── connect.js            # MongoDB connection setup
├── controllers/
│   ├── adminController.js    # Logic for admin actions
│   ├── customerController.js # Logic for customer actions
│   └── sellerController.js   # Logic for seller actions
├── middlewares/
│   └── authMiddleware.js     # Authentication middleware
├── models/
│   ├── userModel.js          # User schema
│   ├── productModel.js       # Product schema
│   ├── cartModel.js          # Cart schema
│   └── orderModel.js         # Order schema
├── routers/
│   ├── adminRoutes.js        # Routes for admin actions
│   ├── customerRoutes.js     # Routes for customer actions
│   └── sellerRoutes.js       # Routes for seller actions
├── uploads/                  # Folder for storing uploaded files
├── .env                      # Environment variables (MongoDB URI, JWT secret, etc.)
├── index.js                  # Main server entry point
└── package.json              # Backend dependencies and scripts
```

6. RUNNING THE APPLICATION:

6.1 Frontend

To start the React frontend:

Navigate to the client directory:

```
cd frontend
```

1. Install dependencies:

```
npm install
```

2. Start the development server:

npm start

3. The frontend will usually run on http://localhost:3000 by default.

6.2 Backend

To start the Node.js backend:

Navigate to the server directory:

cd backend

1. Install dependencies:

bash

Copy code

npm install

2. Set up environment variables:

Create a .env file in the server directory if it doesn't already exist. Add necessary configurations like:

PORT=5000

MONGO_URI=mongodb://localhost:27017/yourDatabase

JWT_SECRET=yourSecretKey

Start the server:

npm start

3. The backend will usually run on http://localhost:5000 by default.

6.3 Running Both Frontend and Backend Concurrently

For ease of development, you can run both the frontend and backend servers simultaneously. If you are using a tool like concurrently:

Install it in the root directory:

npm install concurrently --save-dev

1. Update your root package.json scripts:

"scripts": {

"start": "concurrently \"npm start --prefix frontend\" \"npm start --prefix backend\""

}

2. Run both servers:

npm start

3. Now both the frontend and backend will be running together!

7. API DOCUMENTATIONS:

7.1 GET /API/PRODUCTS

Description: Retrieve all available products.

- **Endpoint:** `/api/products`
- **Method:** GET
- **Headers:** None
- **Request Body:** None
- **Response:**
  - **Status Code:** `200 OK`
  - **Content:**

```
{
 "success": true,
 "products": [
  {
   "id": "productId1",
   "name": "Product Name",
   "price": 10.99,
   "category": "Fruits",
   "image": "image_url",
   "description": "Product description"
  },
  {
   "id": "productId2",
   "name": "Another Product",
   "price": 5.49,
   "category": "Vegetables",
   "image": "image_url",
   "description": "Product description"
  }
 ] }
```

- **Error Response:**
  - **Status Code:** `500 Internal Server Error`

- **Content:**

```
{
 "success": false,
 "message": "Failed to fetch products" }
```

7.2 POST /API/CART

Description: Add a product to the user's cart.

- **Endpoint:** `/api/cart`
- **Method:** POST
- **Headers:**
  - `Authorization: Bearer <JWT_TOKEN>` (for authenticated users)
- **Request Body:**

```
{
 "productId": "productId1",
 "quantity": 2
}
```

- **Response:**
  - **Status Code:** `200 OK`
  - **Content:**

```
{
 "success": true,
 "message": "Product added to cart successfully",
 "cart": {
  "id": "cartId",
  "userId": "userId",
  "items": [
   {
    "productId": "productId1",
    "name": "Product Name",
    "quantity": 2,
    "price": 10.99,
    "total": 21.98
   }
  ]
 }
}
```

- **Error Response:**
  - **Status Code:** `400 Bad Request`
  - **Content:**

```
{
 "success": false,
 "message": "Product not found"
```

```
}
```

- **Status Code:** `401 Unauthorized`
- **Content:**

```
{
 "success": false,
 "message": "Authentication required"
}
```

7.3 POST /api/orders

Description: Place an order for the items in the cart.

- **Endpoint:** `/api/orders`
- **Method:** POST
- **Headers:**
  - `Authorization: Bearer <JWT_TOKEN>` (for authenticated users)
- **Request Body:**

```
{
 "cartId": "cartId",
 "paymentMethod": "Credit Card",
 "shippingAddress": {
  "street": "123 Grocery Lane",
  "city": "Foodville",
  "state": "CA",
  "zipcode": "90001"
 }
}
```

- **Response:**
  - **Status Code:** `201 Created`
  - **Content:**

```
{
 "success": true,
 "message": "Order placed successfully",
 "order": {
  "orderId": "orderId1",
  "userId": "userId",
  "items": [
   {
    "productId": "productId1",
    "name": "Product Name",
    "quantity": 2,
    "price": 10.99,
    "total": 21.98
```

```
    }
   ],
   "totalAmount": 21.98,
   "paymentMethod": "Credit Card",
   "shippingAddress": {
    "street": "123 Grocery Lane",
    "city": "Foodville",
    "state": "CA",
    "zipcode": "90001"
   },
   "status": "Pending"
  }
}
```


- **Error Response:**
  - **Status Code:** `400 Bad Request`
  - **Content:**

    ```
    {
     "success": false,
     "message": "Invalid cart ID"
    }
    ```

  - **Status Code:** `401 Unauthorized`
  - **Content:**

    ```
    {
     "success": false,
     "message": "Authentication required"
    }
    ```

  - **Status Code:** `500 Internal Server Error`
  - **Content:**

    ```
    {
     "success": false,
     "message": "Failed to process order"
    }
    ```


8. AUTHENTICATION:

8.1 Authentication

Authentication ensures users are who they claim to be by verifying their credentials.

8.1.1 User Registration

- **Purpose:** Allow users to sign up.
- **Features:**

- Collect details: Name, email, phone number, password, etc.
- Hash passwords securely using a tool like bcrypt.
- Validate email/phone via OTP or email verification.

8.1.2 User Login Workflow

1. **Verify Credentials:**
    a. User provides email/username and password.
    b. Backend validates credentials against the database.
2. **Generate JWT Token:**
    a. On successful login:
        i. **Payload:** Include user ID, role, and other non-sensitive info.
        ii. **Secret Key:** Use a secure, environment-specific secret key.
        iii. **Expiry:** Set a short token expiration time (e.g., 15 minutes).
3. **Send Token to Client:**
    a. Return the token in the response body or as an HttpOnly cookie.

8.1.3 Token Refresh Workflow

- Use **refresh tokens** for reissuing JWT tokens when they expire:
    - Store refresh tokens securely (e.g., HttpOnly cookie).
    - Assign a longer expiration time (e.g., 7 days) for refresh tokens.
    - Validate refresh tokens via a dedicated API endpoint.

8.1.4 Password Management

- Provide password reset options via email or phone OTP.
- Enforce strong password policies.
- Allow password updates from the user profile after verifying the old password.

8.2 Authorization

Authorization ensures users can access only the resources they are allowed to.

8.2.1 Role-Based Access Control (RBAC)

- **Roles:** Define user roles like Admin, Vendor, and Customer.
- **Permissions:**
    - **Admin:** Manage users, inventory, and orders.
    - **Vendor:** Manage their products and stock.
    - **Customer:** Browse products, add to cart, and place orders.

8.2.2 Middleware for Authorization

Middleware functions are used to verify JWT tokens and roles before granting access to protected resources.

8.3 Using Middleware for JWT and Authorization

8.3.1 JWT Verification Middleware

- **Purpose:** Validate the JWT token sent by the client.
- **Implementation:**

```
const jwt = require('jsonwebtoken');
 const verifyToken = (req, res, next) => {
   const token = req.headers['authorization']?.split(' ')[1]; // Extract token from Bearer
header
   if (!token) {
      return res.status(401).json({ message: 'Access Denied. Token Missing.' });
   }

   try {
      const decoded = jwt.verify(token, process.env.JWT_SECRET);
      req.user = decoded; // Attach decoded data (e.g., userId, role) to the request
      next(); // Pass control to the next middleware
   } catch (error) {
      return res.status(403).json({ message: 'Invalid or Expired Token.' });
   }
};
 module.exports = verifyToken;
```

8.3.2 Role-Based Authorization Middleware

- **Purpose:** Ensure only users with specific roles can access certain resources.
- **Implementation:**

```
const authorizeRole = (roles) => {
   return (req, res, next) => {
      if (!roles.includes(req.user.role)) {
         return res.status(403).json({ message: 'Forbidden. Insufficient Permissions.' });
      }
      next(); // User has the required role; proceed
   };
};  module.exports = authorizeRole;
```

8.3.3 Applying Middleware in Routes

- Use middleware to secure routes based on authentication and roles.

```
const express = require('express');
const router = express.Router();
const verifyToken = require('./middlewares/verifyToken');
const authorizeRole = require('./middlewares/authorizeRole');
router.post('/login', loginController);
router.post('/register', registerController);
router.get('/user/profile', verifyToken, userProfileController);
router.post('/admin/add-product', verifyToken, authorizeRole(['admin']),
addProductController);
router.get('/vendor/products', verifyToken, authorizeRole(['vendor']),
vendorProductsController);
```

8. 4. Security Enhancements

8.4.1 Secure JWT Implementation

- **JWT Payload Structure:**

```
{
  "id": "12345",
  "role": "admin",
  "email": "user@example.com",
  "iat": 1699989600, // Issued at
  "exp": 1699993200  // Expiry
}
```

- **Generate JWT in Node.js:**

```
const jwt = require('jsonwebtoken');
 const generateToken = (user) => {
   return jwt.sign(
     { id: user.id, role: user.role }, // Payload
     process.env.JWT_SECRET,         // Secret key
     { expiresIn: '15m' }           // Expiry
   );
};
```

8.4.2 Securing Tokens

- Store JWT tokens in **HttpOnly cookies** to protect against XSS attacks.
- Use **HTTPS** to encrypt all communications between client and server.
- Implement **rate limiting** to prevent brute-force attacks on login endpoints.
- Use CSRF tokens if JWT is stored in cookies.

8.4.3 Logging and Monitoring

- Log authentication and authorization events.
- Monitor suspicious activities like multiple failed login attempts.

9. USER INTERFACE:

9.1 UI-User Interface

For a Grocery Web Application, the user interfaces (UIs) are the points of interaction between the users and the application.

1. Home Page

Purpose: Introduce the application and showcase the primary features.

Features:

- Search bar for quick product searches.
- Categories or featured products section.
- Navigation menu (e.g., Home, Products, Cart, Contact, etc.).

2. Product Listing Page

Purpose: Display all available grocery items.

Features:

- Filters (e.g., category, price range, brand).
- Sort options (e.g., by price, popularity, or freshness).
- Grid or list views of products with images, names, prices, and "Add to Cart" buttons.

3. Product Details Page

Purpose: Provide detailed information about a selected product.

Features:

- Product images.
- Description, nutritional information, and price.
- Quantity selection and "Add to Cart" button.
- Related products or suggestions.

4. Cart Page

Purpose: Allow users to review and modify their selected items before purchasing.

Features:

- List of selected products with quantity and price.

- Update or remove items from the cart.
- Display of subtotal, tax, and total cost.
- "Proceed to Checkout" button.

5. Checkout Page

Purpose: Handle the payment and order confirmation process.

Features:

- User shipping information form.
- Payment options (e.g., credit card, PayPal, COD).
- Order summary.
- "Place Order" button.

6. User Registration/Login Page

Purpose: Authenticate users to access their personalized data.

Features:

- Login form with email and password fields.
- Registration form for new users.

8. Search Results Page

Purpose: Display products that match the user's search query.

Features:

- List of matching products with the same layout as the product listing page.
- Option to refine the search with filters.

10. TESTING:

Testing a grocery web app involves various methodologies to ensure its reliability, functionality, usability, security, and performance. Here's how testing is typically performed, broken down into specific categories:

a.Functional Testing

Ensures that the app's features work as expected.

Key Areas to Test:

1. User Registration and Login:
   - Validate registration with valid/invalid credentials.
   - Test login with JWT token generation and validation.
2. Product Management:
   - Add, edit, delete, and search products.
   - Verify product details display correctly

3. Shopping Cart:
    - Add/remove items to/from the cart.
    - Update quantities.
    - Verify cart total and discount calculations.
4. Order Placement:
    - Test the checkout process, including payment gateway integration.
    - Ensure order confirmation is sent to the user.
5. Search and Filters:
    - Validate product search and filters (category, price, etc.).

b.Usability Testing

Ensures the app is user-friendly and intuitive.

Key Areas to Test:

1. Navigation:
    - Test how easily users can navigate the app.
    - Ensure menus, buttons, and links are accessible.
2. Visual Consistency:
    - Check for consistent fonts, colors, and alignment.

c.Performance Testing

Checks the app's performance under different conditions.

Key Areas to Test:

1. Load Testing:
    - Simulate high traffic to ensure the app handles concurrent users smoothly.
2. Stress Testing:
    - Test how the app performs under extreme conditions (e.g., traffic spikes).
3. Database Performance:
    - Verify the database can handle high-volume read/write operations efficiently.

d.Security Testing

Ensures the app is protected against vulnerabilities.

Key Areas to Test:

1. Authentication:
    - Test password strength and encryption.
    - Validate JWT token generation and expiration.
2. Authorization:
    - Verify users can access only the resources allowed for their role
3. Data Security:
    - Check for secure handling of sensitive data (e.g., payment information).
4. Common Vulnerabilities:

- Test for SQL injection, XSS, CSRF, and insecure API endpoints

e.Integration Testing

Ensures different modules and systems work together as expected.

Key Areas to Test:

1. API Integration:
   - Validate backend services for login, product listings, orders, and payments.
2. Middleware:
   - Verify middleware like JWT-based authentication and role-based authorization.

f.Database Testing

Ensures database operations are accurate and efficient.

Key Areas to Test:

1. Validate CRUD operations (Create, Read, Update, Delete).

g.Regression Testing

Ensures new features or changes don't break existing functionality.

Approach:

1. Create and maintain a comprehensive test suite.
2. Automate repetitive test cases to save time.

h.User Acceptance Testing (UAT)

Validates the app against real-world use cases before release.

Process:

1. Involve end-users to simulate real-life scenarios.
2. Collect feedback on usability, speed, and overall satisfaction.
3. Perform tests in staging or pre-production environments.

i.Automation Testing

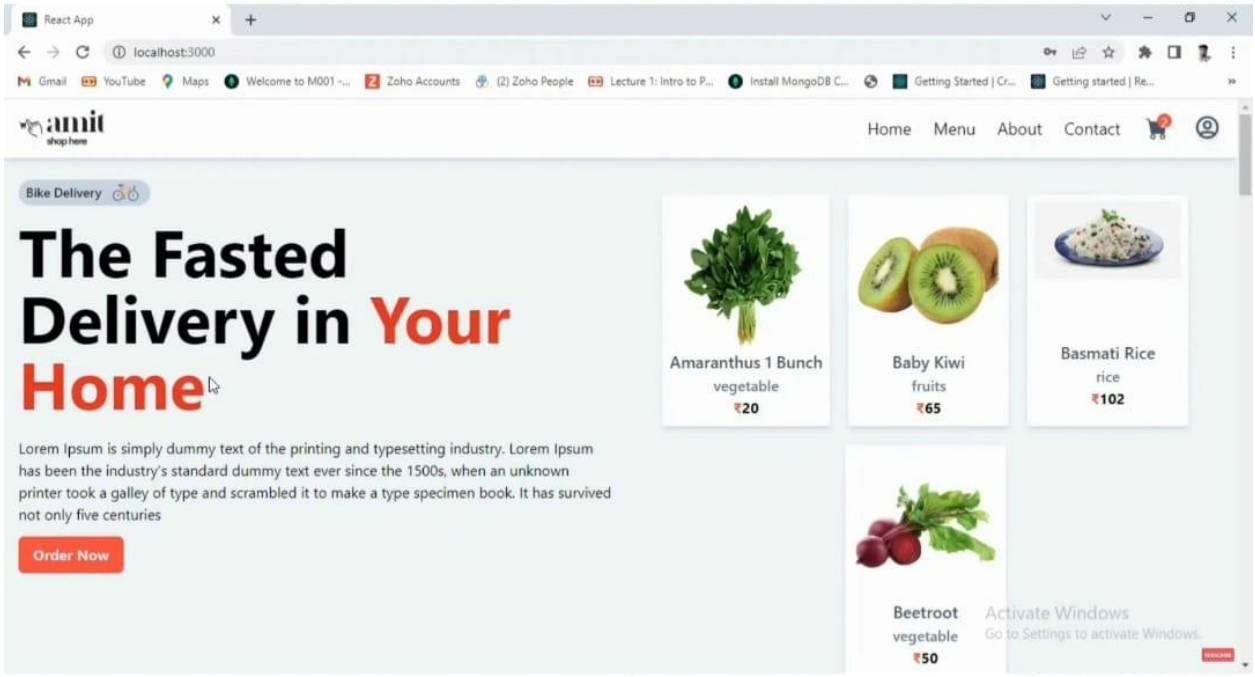
Automates repetitive tasks to save time and reduce manual effort.

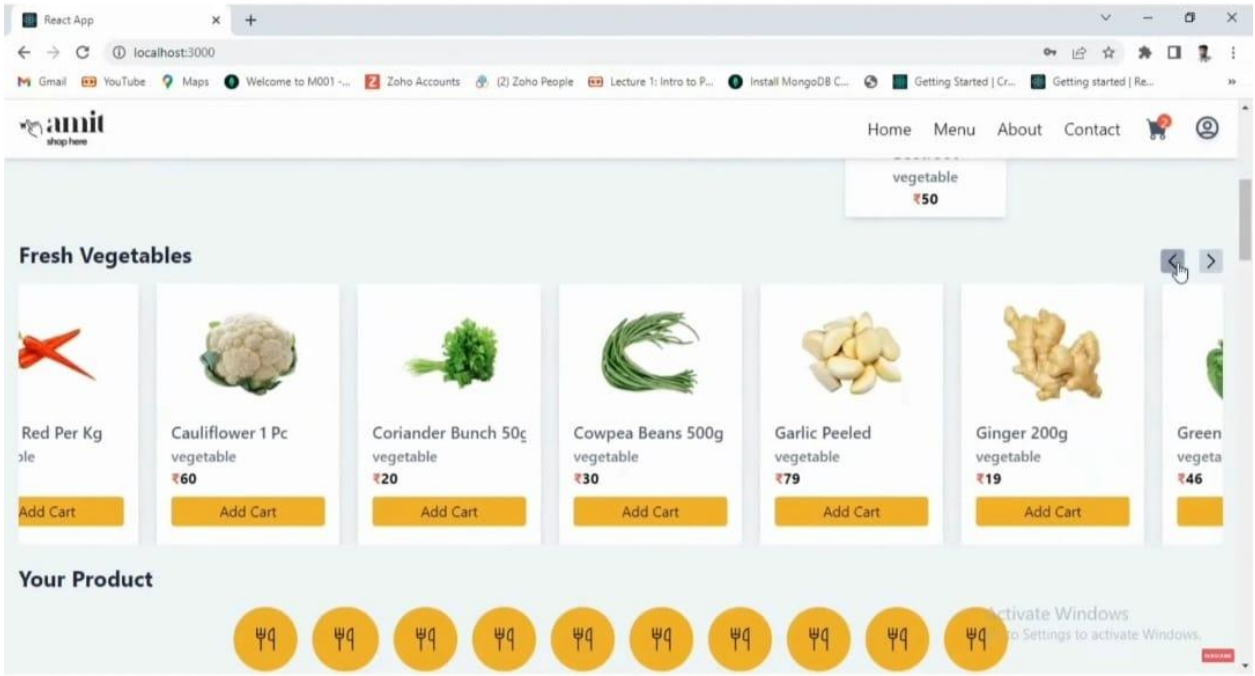
Key Areas to Automate:

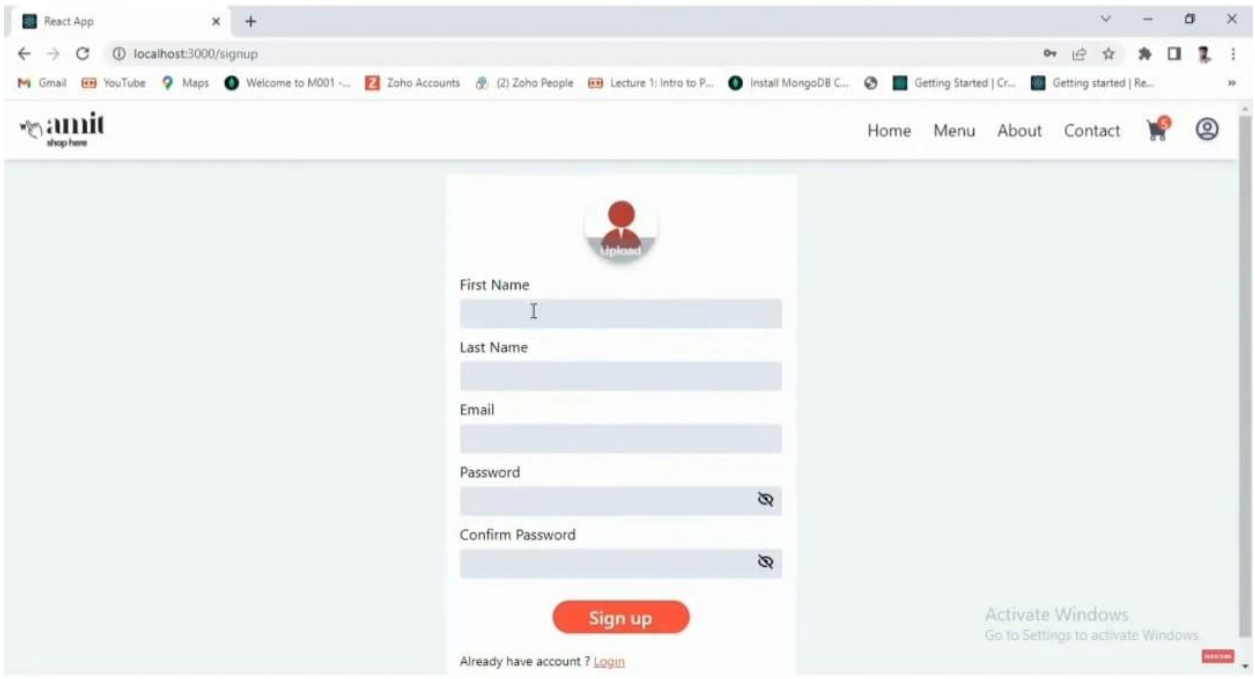
1. Login/logout workflows.
2. Product search and filter functionality.

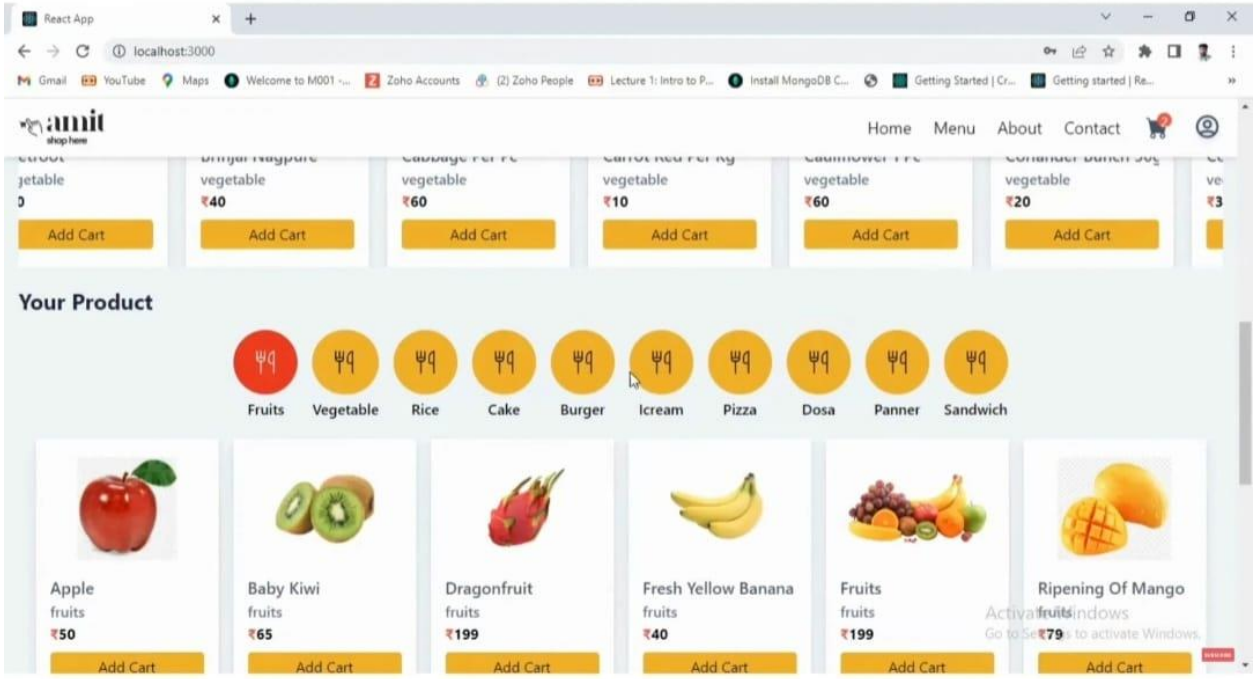
11.OUTPUT SCREENSHOTS AND DEMOLINK

11.1 Output Screenshots

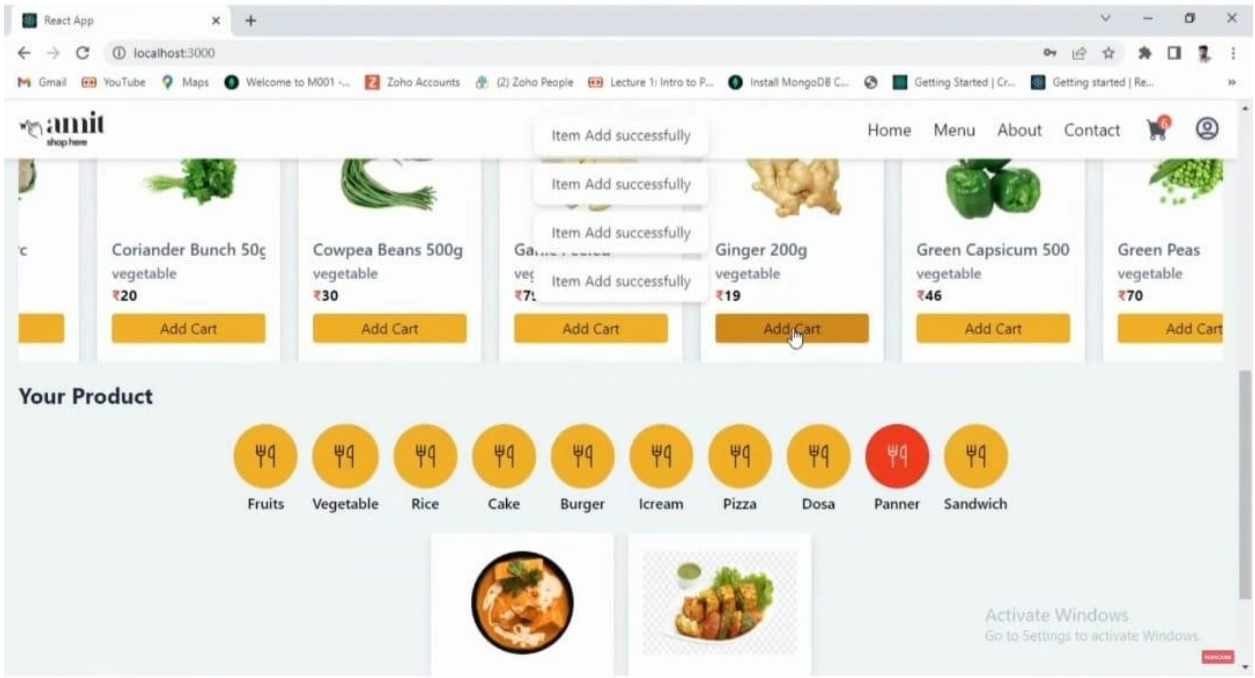

React App
localhost:3000
Gmail
YouTube
Maps
Welcome to M001 -...
Zoho Accounts
(2) Zoho People
Lecture 1: Intro to P...
Install MongoDB C...
Getting Started | Cr...
Getting started | Re...
amit
shop here
Home
Menu
About
Contact
Bike Delivery
The Fasted Delivery in Your Home
Lorem Ipsum is simply dummy text of the printing and typesetting industry. Lorem Ipsum has been the industry's standard dummy text ever since the 1500s, when an unknown printer took a galley of type and scrambled it to make a type specimen book. It has survived not only five centuries
Order Now
Amaranthus 1 Bunch
vegetable
₹20
Baby Kiwi
fruits
₹65
Basmati Rice
rice
₹102
Beetroot
vegetable
₹50
Activate Windows
Go to Settings to activate Windows.

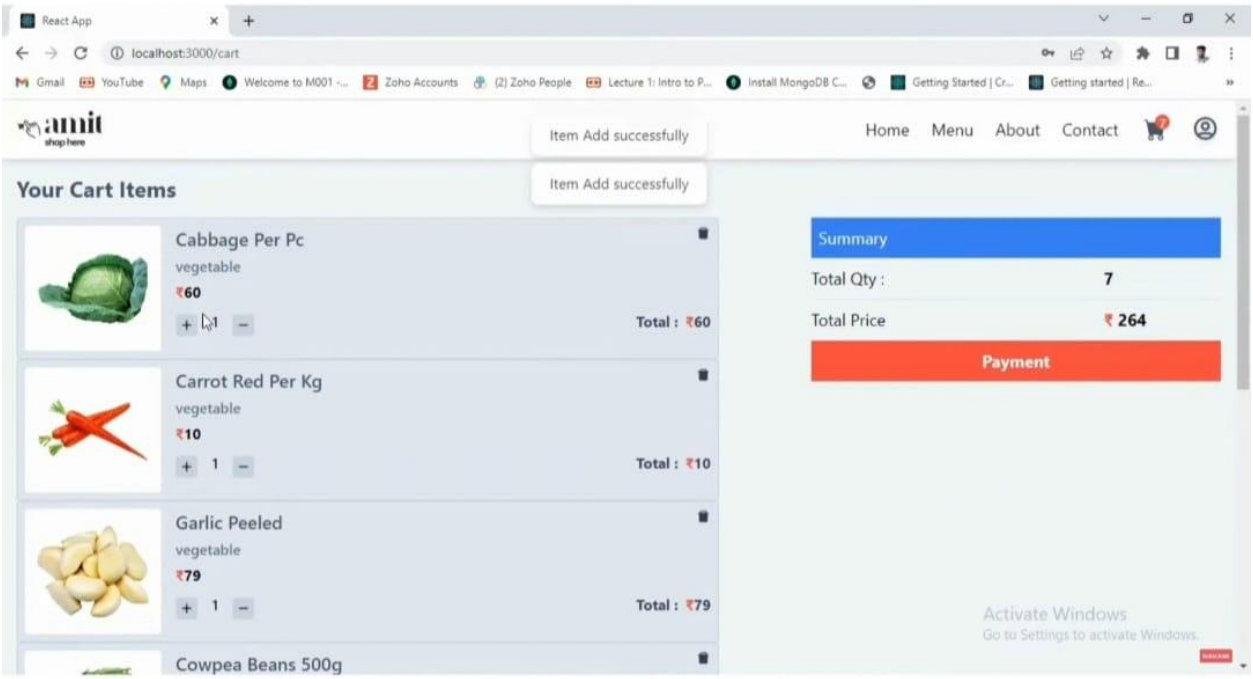

React App
localhost:3000
Gmail
YouTube
Maps
Welcome to M001 -...
Zoho Accounts
(2) Zoho People
Lecture 1: Intro to P...
Install MongoDB C...
Getting Started | Cr...
Getting started | Re...
amit
shop here
Home
Menu
About
Contact
vegetable
₹50
Fresh Vegetables
Red Per Kg
Add Cart
Cauliflower 1 Pc
vegetable
₹60
Add Cart
Coriander Bunch 50g
vegetable
₹20
Add Cart
Cowpea Beans 500g
vegetable
₹30
Add Cart
Garlic Peeled
vegetable
₹79
Add Cart
Ginger 200g
vegetable
₹19
Add Cart
Green
vegeta
₹46
Your Product
Activate Windows


React App
localhost:3000/signup
Gmail
YouTube
Maps
Welcome to M001 -...
Zoho Accounts
(2) Zoho People
Lecture 1: Intro to P...
Install MongoDB C...
Getting Started | Cr...
Getting started | Re...
amit
shop here
Home
Menu
About
Contact
Upload
First Name
Last Name
Email
Password
Confirm Password
Sign up
Already have account ? Login
Activate Windows
Go to Settings to activate Windows.


React App
localhost:3000
Gmail
YouTube
Maps
Welcome to M001 -...
Zoho Accounts
(2) Zoho People
Lecture 1: Intro to P...
Install MongoDB C...
Getting Started | Cr...
Getting started | Re...
amit
shop here
Home
Menu
About
Contact
vegetable
₹40
Add Cart
vegetable
₹60
Add Cart
vegetable
₹10
Add Cart
vegetable
₹60
Add Cart
vegetable
₹20
Add Cart
Your Product
Fruits
Vegetable
Rice
Cake
Burger
Icream
Pizza
Dosa
Panner
Sandwich
Apple
fruits
₹50
Add Cart
Baby Kiwi
fruits
₹65
Add Cart
Dragonfruit
fruits
₹199
Add Cart
Fresh Yellow Banana
fruits
₹40
Add Cart
Fruits
fruits
₹199
Add Cart
Ripening Of Mango
fruits
₹79
Add Cart


React App
localhost:3000/menu/63f0fd983bcc2f97fa53d25b
amit
shop here
Home
Menu
About
Contact
Baby Kiwi
fruits
₹65
Buy
Add Cart
Description :
Baby kiwi fruit are small berries, just shy of the size and similar in shape to a grape, with a razor thin fuzz-free smooth skin
Related Product
Fruits
Vegetable
Rice
Cake
Burger
Icream
Pizza
Dosa
Panner
Sandwich


React App
localhost:3000
amit
shop here
Home
Menu
About
Contact
Item Add successfully
Item Add successfully
Item Add successfully
Item Add successfully
Coriander Bunch 50g
vegetable
₹20
Add Cart
Cowpea Beans 500g
vegetable
₹30
Add Cart
Add Cart
Ginger 200g
vegetable
₹19
Add Cart
Green Capsicum 500
vegetable
₹46
Add Cart
Green Peas
vegetable
₹70
Add Cart
Your Product
Fruits
Vegetable
Rice
Cake
Burger
Icream
Pizza
Dosa
Panner
Sandwich



11.2 Demo Link

Demo Video Link: DEMO

12. KNOWN ISSUES:

12.1 Issues and Solutions

Here are common issues in a grocery web app along with potential solutions:

1. Authentication and Authorization Issues

Issue:

- Weak password policies lead to unauthorized access.
- JWT tokens not expiring properly.
- Improper role-based access controls (e.g., customers accessing admin features).

Solution:

- Enforce strong password rules (minimum length, special characters, etc.).
- Set short expiry times for JWT access tokens and use refresh tokens securely.
- Implement middleware to validate JWT tokens and roles on every request.
- Regularly test for vulnerabilities like session hijacking or token replay attack.

2. Poor Performance During High Traffic

Issue:

- Slow page loading or system crashes during sales or promotions.

Solution:

- Use a content delivery network (CDN) to cache static content.
- Implement database indexing and optimize queries.
- Scale server resources during peak times using load balancers.
- Use caching mechanisms like Redis for frequently accessed data (e.g., product catalogs).
- Test scalability with load testing tools like Apache JMeter.

3. Search and Filter Functionality Issues

Issue:

- Search returns irrelevant or incomplete results.
- Filters don't update dynamically or display incorrect product counts.

Solution:

- Use a full-text search engine like Elasticsearch for better search relevance.
- Ensure filters are dynamically updated based on available inventory.
- Test all combinations of filters and search terms to validate results.

4. Shopping Cart Errors

Issue:

- Items disappear from the cart unexpectedly.
- Incorrect total calculations due to discounts or taxes.

Solution:

- Save the cart data in session storage or user profiles on the server side.
- Validate price calculations on both the client and server sides to avoid inconsistencies.
- Implement automated tests for cart functionality, including adding, removing, and updating items.

5. User Experience (UX) Issues

Issue:

- Slow page transitions or unresponsive UI.
- Difficult navigation or cluttered layout.

Solution:

- Optimize front-end code for performance (minify CSS, JavaScript).
- Use frameworks like React or Angular for smooth UI updates.
- Conduct usability testing with real users and incorporate their feedback.
- Ensure responsive design for mobile and tablet users.

13. FUTURE ENHANCEMENT:

13.1 Future Improvements

For a grocery web application, future enhancements can help improve user experience, increase efficiency, and expand functionality.

1. **Personalized Recommendations**:

   Using AI algorithms, the app can suggest products based on the user's purchase history, preferences, or dietary needs. It could also offer special promotions or discounts tailored to individual users.

2. **Multi-Language Support**:
   Adding support for multiple languages could help cater to a more diverse customer base, expanding the reach of the platform.

3. **Mobile App Integration:**

   Developing mobile app to complement the web version would provide users with an easy to-use interface for shopping on the go,along with push notifications for discounts or special offer.

4. **Real-Time Inventory Updates:**
   Implementing a real-time inventory tracking system would reduce issues like ordering out-of-stock items and ensure that customers have accurate information.

5. **Sustainability Features**:
   Including options like carbon footprint tracking for deliveries, eco-friendly product suggestions, or offering discounts for using reusable bags could appeal to environmentally conscious customers.

6. **AI-Powered Chatbot**:
   Implementing an AI chatbot to assist with customer service inquiries or offer product suggestions would enhance the customer experience and reduce support costs.

7. **Enhanced Payment Options:**
   Offering a wider range of payment methods, including cryptocurrencies or buy-now-pay-later services, could appeal to a broader audience and improve the checkout experience.