Project Title: Grocery WebApp

Team members:

Devanesan R   IT-A - 210821205020
Bhavanesh M IT-A - 210821205015
Ragul K IT-B - 210821205081
Rajesh R IT-B - 210821205082

Purpose:

A grocery web app provides convenience, time savings, and personalized experiences for shoppers by allowing them to browse, order, and schedule delivery or pickup of groceries online. It enhances user experience through features like personalized recommendations, shopping lists, and customizable orders. The app helps users save money with price comparisons, discounts, and subscription services. It also promotes healthier choices with dietary filters and nutritional info. For businesses, it improves efficiency with inventory and order management and offers customer insights. Sustainability features, accessibility, and loyalty programs further enhance its value for both users and retailers.

Features:

A grocery web app offers a range of features to enhance the shopping experience, including a comprehensive product catalog with search and personalized recommendations, secure payment options, and flexible delivery or pickup choices. Users can customize orders, apply discounts, and set up recurring subscriptions for frequently bought items. Additional features like detailed product information, nutrition filters, customer reviews, and order tracking help users make informed decisions. The app also supports multi-device access, offers customer support, and sends push notifications for deals and order updates.

Architecture:

Frontend:

The frontend architecture of a grocery web app using React is component-based, where UI elements like product cards, shopping carts, and search bars are built as reusable components. State is managed using React's `useState` or `useReducer` for local state and Context API or Redux for global state (e.g., cart and user data). **React Router** handles page navigation, while API calls (using Axios or Fetch) fetch dynamic data like product listings and user info.

For styling, **Material-UI** or **Chakra UI** can be used for consistent, accessible components, and **CSS-in-JS** or **CSS Modules** for scoped styling. The app is responsive, optimized with techniques like **code splitting** and **lazy loading** for better performance, especially on mobile. **JWT** handles user authentication, ensuring secure access to protected routes (like checkout). Finally, **unit tests** with **Jest** and **React Testing Library** ensure reliability.

This architecture is scalable, efficient, and provides a smooth user experience across devices.

Backend:

The backend architecture for a grocery web app using **Node.js** and **Express.js** involves setting up a RESTful API to handle product listings, user authentication, cart management, and order processing. **Express.js** routes API requests, while a **NoSQL database** (e.g., MongoDB) stores data like products and user profiles. **JWT** handles secure authentication, and middleware ensures input validation, error handling, and route protection.

Key API endpoints include product CRUD operations, user management, cart actions, and order processing. Payment integration with **Stripe** or **PayPal** supports secure transactions. The backend is scalable, with options for **Docker** containers and cloud deployment (e.g., AWS, Heroku). This architecture ensures a secure, efficient, and scalable backend for the web app.

Database:

For a grocery web app using **MongoDB**, the database schema includes several key collections:

1. **Product** (`products`): Stores product details like name, price, description, stock, and images. CRUD operations allow adding, updating, and retrieving products.
2. **User** (`users`): Stores user information (name, email, password, address) and order history. Supports user registration, login, and profile management.
3. **Cart** (`carts`): Tracks each user's shopping cart with product IDs, quantities, and total price. Supports adding/removing items and fetching the current cart.
4. **Order** (`orders`): Stores order details such as user, items, shipping address, and payment status. Manages order creation and status updates (e.g., pending, shipped, delivered).
5. **Review** (`reviews`, optional): Allows users to leave product reviews with ratings and comments.

Setup Instructions:

Grocery Web App Setup Overview (Angular + MongoDB)

The **Grocery Web App** is a simple e-commerce platform that lets users browse products, add them to their cart, and place orders. The app uses **Angular** for the frontend (user interface) and **MongoDB** for storing data (products, user accounts, orders).

1. What You Need to Get Started

To run the app, make sure you have these tools installed:

Node.js: Required to run the backend and manage dependencies.

- **Angular CLI**: A tool to help you create and run Angular projects.

- **MongoDB**: A database to store product and user data. You can use **MongoDB Atlas** (cloud-based) or install **MongoDB** locally.

Git: A version control tool for managing the code.

2. How the App Works

- **Frontend (Angular)**: This is the part the user interacts with. It displays products, handles the shopping cart, and lets users place orders. It communicates with the backend to get product data and send order details.

- **MongoDB (Database)**: Stores all the data for the app, such as products, users, and orders. MongoDB is flexible and easy to work with.

3. Setting Up the App

1. **Clone the Project**: Download the app's code from a Git repository.

2. **Install Dependencies**:

   - In the **frontend** folder, run `npm install` to install the necessary libraries (Angular components, etc.).

3. **Run the Backend**:

   - Set up a backend server to handle requests (for example, using **Node.js** and **Express**) and connect to **MongoDB**.

4. **Run the Frontend**:

   - Use Angular's command `ng serve` to run the frontend, which will show the user interface in your web browser.

4. Key Features

- **Frontend (Angular)**: Displays products, shopping cart, and order form. Handles user interactions like adding products to the cart and placing orders.

- **Database (MongoDB)**: Stores product data (e.g., product names, prices) and user data (e.g., usernames, orders).

Conclusion

This app uses **Angular** for the frontend, **MongoDB** for the database, and communicates between the two to let users shop online. Once set up, you can browse products, add them to the cart, and place orders.

Folder Structure:

Frontend (React) Structure:

- **/public**: Contains the `index.html` and static assets.
- **/src**: All frontend code:
  - **/components**: Reusable UI elements (e.g., ProductCard, Navbar).
  - **/pages**: App pages (e.g., Home, Cart, Checkout).
  - **/services**: API calls to the backend (e.g., fetchProducts).
  - **/context**: Global state management (e.g., CartContext).
  - **/styles**: CSS or styled-components for UI styling.

Backend (Node.js) Structure:

- **/config**: Configuration files (e.g., database setup).
- **/controllers**: Business logic (e.g., handling product, order requests).
- **/models**: Mongoose models for MongoDB schemas (e.g., Product, Order).
- **/routes**: API endpoints for handling requests (e.g., productRoutes).
- **/middleware**: Middleware functions (e.g., authentication checks).
- **/utils**: Helper functions (e.g., JWT token generation).

This structure keeps the project modular and organized, separating concerns between the frontend, backend, and database.

Running the Application:

Running the Grocery Web App Locally

1. Backend:

Navigate to the **server** directory:
bash
Copy code
```
cd server
```

- Install dependencies:
  bash
  Copy code
  ```
  npm install
  ```
- Start the backend:
  bash
  Copy code
  ```
  npm start
  ```
- Backend runs on `http://localhost:5000` (or custom port).

2. Frontend:

Navigate to the **client** directory:
bash
Copy code
```
cd client
```

- Install dependencies:
  bash
  Copy code
  `npm install`
- Start the frontend:
  bash
  Copy code
  `npm start`
-
    - Frontend runs on `http://localhost:3000`.

3. Access the App:

Open `http://localhost:3000` in your browser to view the app.

This will run both the **React frontend** and **Node.js backend** locall

API Documentation:

1. Products:

- **GET /api/products**: Retrieve all products.
    -
    - **Response**: List of products (name, price, description).
- **GET /api/products/**
  : Retrieve product details by ID.
    - **Response**: Product details (name, price, description).
- **POST /api/products**: Add a new product.
    - **Request**: Product details (name, price, description).
    - **Response**: Success message with new product.

2. Orders:

- **POST /api/orders**: Place an order.
    - **Request**: User ID, items (product ID, quantity), total price.
    - **Response**: Order confirmation.
- **GET /api/orders/**
  : Retrieve orders by user ID.
    - **Response**: List of user orders.

3. User Authentication:

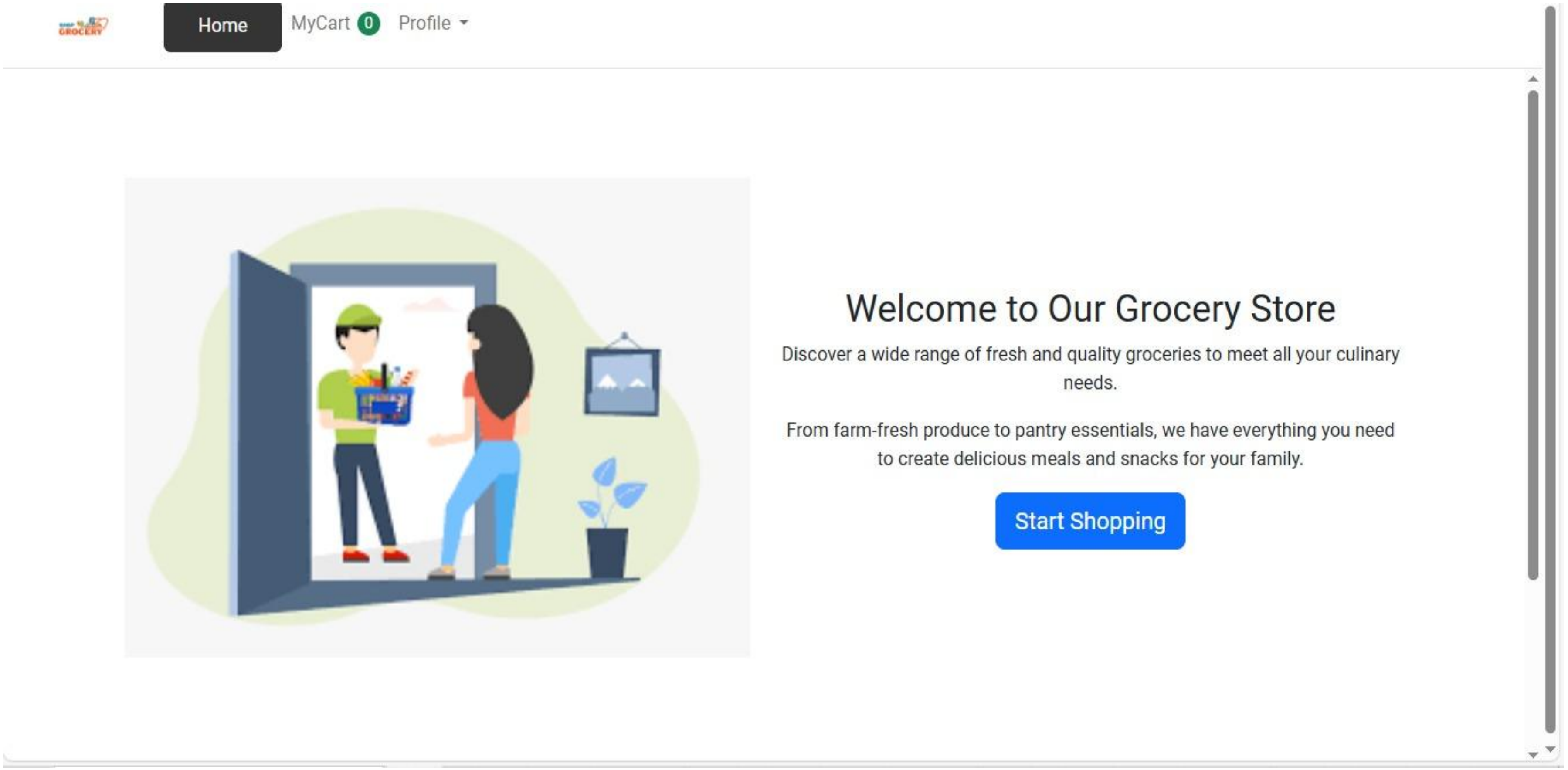
- **POST /api/auth/register**: Register a new user.
    - **Request**: Username, email, password.
    - **Response**: Success message with user details.
- **POST /api/auth/login**: Log in an existing user.
    - **Request**: Email, password.
    - **Response**: JWT token.

Authentication:

- **Authentication (Login & Registration)**:
  - **JWT Tokens** are used to authenticate users.
  - On **registration**, passwords are hashed and stored securely.
  - On **login**, valid credentials return a **JWT token** to the user.
- **Authorization**:
  - Protected routes (e.g., placing orders) require a valid JWT token.
  - The **JWT token** is included in the **Authorization header** for requests.
  - **Middleware** verifies the token and grants access to protected resources.
- **Token Expiry**:
  - JWT tokens have an expiry time (e.g., 1 hour). Users must log in again or use a refresh token.
- **Security**:
  - Passwords are hashed with **bcrypt**.
  - Communication is over **HTTPS** for security.
  - **JWT secret** ensures secure signing and verification of tokens.

This system ensures secure user access and protects sensitive data across the app.

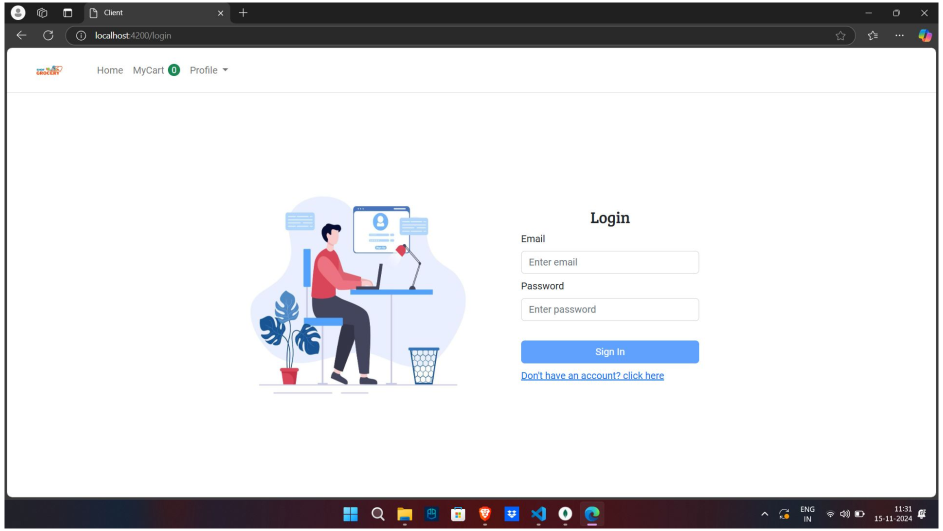
User Interface:

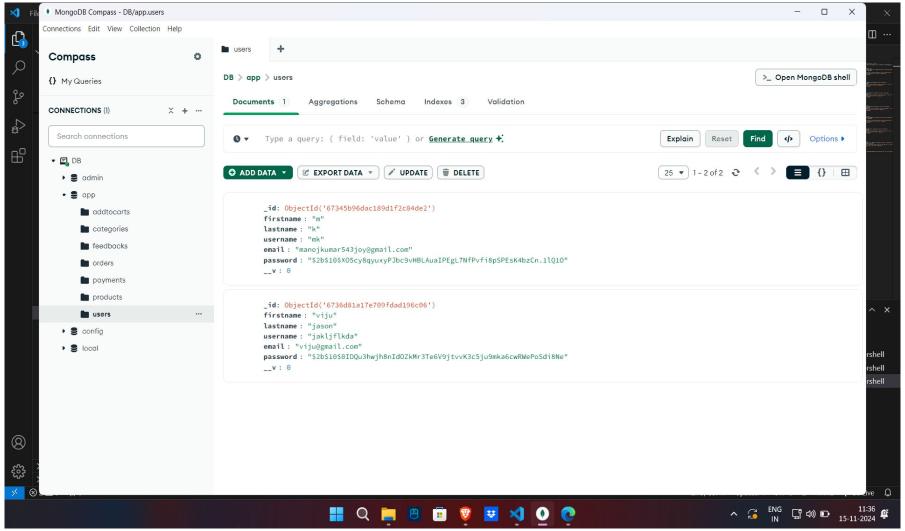


Welcome to Our Grocery Store

Discover a wide range of fresh and quality groceries to meet all your culinary needs.

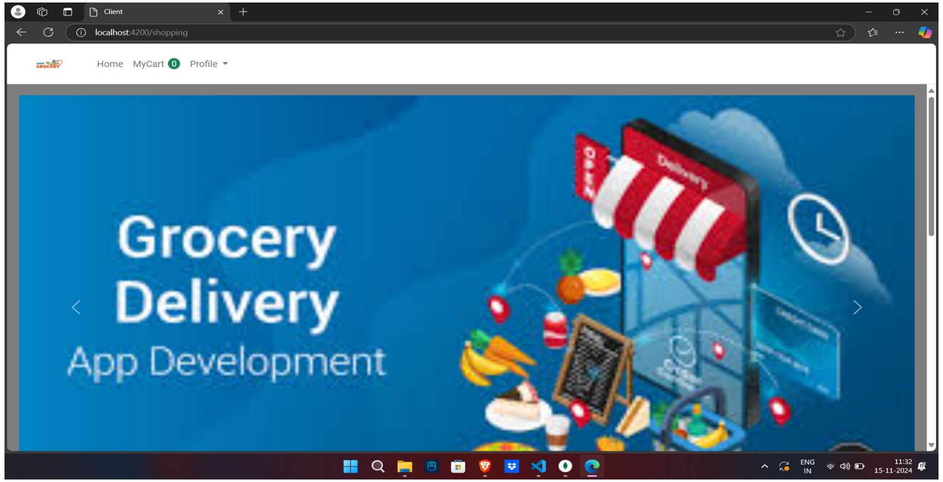
From farm-fresh produce to pantry essentials, we have everything you need to create delicious meals and snacks for your family.

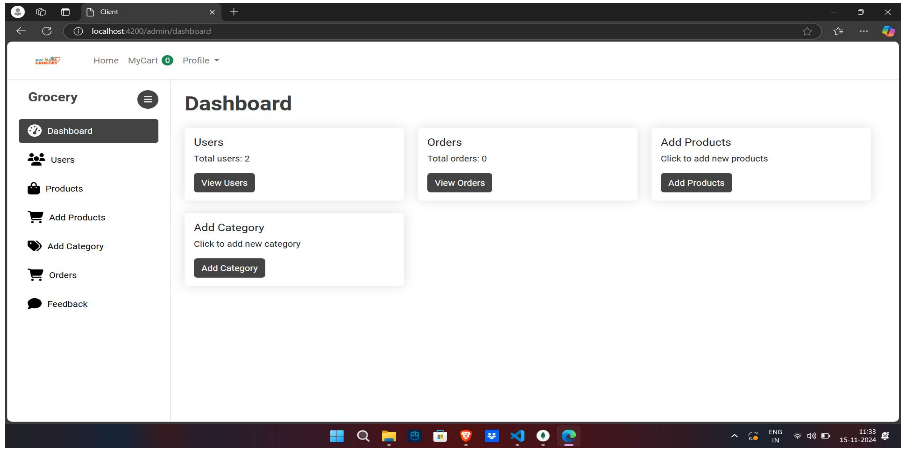




```
_id: ObjectId('67345b96dac189d1f2c84de2')
firstname : "m"
lastname : "k"
username : "mk"
email : "manojkumar543joy@gmail.com"
password : "$2b$10$XO5cy8qyuxyPJbc9vHBLAuaIPEgL7NfPvfi8pSPEsK4bzCn.1lQ1O"
__v : 0

_id: ObjectId('6736d81a17e709fdad196c06')
firstname : "viju"
lastname : "jason"
username : "jakljflkda"
email : "viju@gmail.com"
password : "$2b$10$0IDQu3hwjh8nIdOZkMr3Te6V9jtvvK3c5ju9mka6cwRWePoSdi8Ne"
__v : 0
```





Dashboard

Users
Total users: 2
View Users

Orders
Total orders: 0
View Orders

Add Products
Click to add new products
Add Products

Add Category
Click to add new category
Add Category

Testing:

1. Frontend Testing (React):

Tools: **Jest**, **React Testing Library**

- **Focus**:
    - Unit tests for React components (e.g., Cart, ProductList).
    - Mocking API calls for isolated testing.
    - Component interaction and rendering.

2. Backend Testing (Node.js/Express):

- **Tools**: **Jest**, **Supertest**
- **Focus**:
    - Unit tests for backend logic and functions.
    - API endpoint testing (e.g., GET/POST requests).
    - Mocking database interactions.

3. End-to-End (E2E) Testing:

- **Tool**: **Cypress**
- **Focus**:
    - Full user flow testing (e.g., login, shopping, checkout).
    - UI/UX, responsiveness, and error handling.

4. Manual Testing:

- **Focus**:
    - Exploratory testing, cross-browser, and mobile testing.

5. CI/CD Integration:

- **Tools**: **GitHub Actions**, **Jenkins**
- **Focus**: Automated tests on each code change to ensure code quality.

This approach ensures thorough testing across all layers of the app—frontend, backend, and user interactions.

Known Issues:

User Interface is not good.

Future Enhancements:

1. **User Account Management**: Profile customization, order history, and reordering options.
2. **Advanced Search & Filters**: Improved search with auto-suggestions, filters, and voice search.
3. **Recommendation System**: Personalized product suggestions based on user behavior.
4. **Real-time Inventory Updates**: Stock alerts and real-time product availability.
5. **Subscription & Recurring Orders**: Grocery subscriptions and scheduled orders.
6. **Multi-language & Currency Support**: Localization and currency options for global users.
7. **Mobile App**: Develop native iOS/Android apps with offline capabilities.
8. **Payment Gateway Integration**: Add more payment options (e.g., Google Pay, PayPal, Crypto).
9. **Customer Reviews**: Product ratings and review filtering.
10. **Admin Dashboard**: Advanced analytics, order management, and insights.
11. **Sustainability Features**: Eco-friendly products and carbon footprint tracking.
12. **Delivery Enhancements**: Same-day delivery and real-time tracking.