GROCERY WEBAPP POWERED BY MERN STACK

Project Documentation

1. Introduction

- **Project Title:** Grocery WebApp – MERN stack powered by MongoDB.
- **Team Members:** Gowtham S, Hemalakshmi D, Jaiththira R, Keerthana S, Likitha R

2. Project Overview

- **Purpose:** A grocery web app built using the MERN stack (MongoDB, Express.js, React.js, Node.js) offers significant benefits to both consumers and businesses. It helps to increase the convenience of the customers to buy the necessary needs. The web application ensures scalability, reliability, and adaptability to meet the needs of both consumers and suppliers.
- **Features:** To improve the convenient, efficient and user-friendly platform for purchasing groceries online. It aims to enhance user convenience, streamline shopping experience, promote vendor accessibility, optimize delivery system and encourage sustainability.

3. Architecture

- **Frontend:** React is frontend framework used to create a dynamic, user-friendly interfaces for browsing, searching and managing grocery items effectively.
- **Backend:** Node JS and Express JS power the backend of the MERN stack grocery WebApp, handling server-side logic, API routing and database interactions seamlessly.
- **Database:** MongoDB serves as the database in grocery WebApp efficiently storing and managing data such as product details, user profiles, and order histories.

4. Setup Instructions

- **Prerequisites:** Software dependencies (React JS, Node JS, Express JS, MongoDB, VS code, Git).
- **Installation:** Step-by-Step Installation of Prerequisites for a MERN Stack Grocery Web App

    1. **Install Node.js:**
       Download Node.js from the official website. Install using the default installer for your operating system.

       Verify the installation:
       node -v
       npm -v

2. **Install MongoDB:** Download MongoDB Community Edition from the official MongoDB website. Follow the installation instructions for your operating system. Start the MongoDB service:

Windows: Run mongodb in the Command Prompt.

macOS/Linux: Use brew services start mongodb/brew/mongodb-community or sudo service mongodb start.

3. **Install Visual Studio Code (Optional):** Download Visual Studio Code and install it. Install recommended extensions:
   ESLint
   Prettier
   MongoDB for VS Code

4. **Install Git (Optional):**
   Download and install Git from Git's official website. Verify installation:

   git –version.

5. **Set Up the Project Directory Open a terminal and create a new directory:**

   mkdir grocery-webapp
   cd grocery-webapp

6. **Initialize a Node.js Project:**
   Initialize a new Node.js project:

   npm init -y

   This creates a package.json file to manage dependencies.

7. Install Required Backend Packages

Install Express.js, Mongoose, CORS, and dotenv:

npm install express mongoose cors dotenv

8. Install Nodemon for Backend Development

Install Nodemon globally or as a dev dependency:

npm install --save-dev nodemon

Update package.json to add a script:

"scripts": {

"start": "node server.js",

"dev": "nodemon server.js"

}

9. Set Up React Frontend

Navigate back to the project directory and create the React app:

npx create-react-app client

Start the React development server:

cd client

npm start

10. Configure MongoDB

Create a database in MongoDB (optional):

Use the Mongo shell or Compass GUI to create a database called groceryDB.

Create a .env file in the backend:

MONGO_URI=mongodb://localhost:27017/groceryDB

Ensure your backend connects to MongoDB using mongoose.connect() in server.js.

11. Verify Setup

Start the backend server:

npm run dev

Start the frontend React app:

cd client

npm start

Now ready to start developing your grocery web app using the MERN stack.

5. Folder Structure

- **Client:** The client-side folder structure for a grocery web app using React in the MERN stack typically includes directories for components, pages, hooks, contexts, services, assets, and styles, with key files like App.js, index.js, and individual component files such as Header.js, ProductList.js, and Cart.js.

- **Server:** The server-side folder structure for a grocery web app using the MERN stack typically includes directories for models, routes, controllers, middleware, and configuration, with key files like server.js, app.js, db.js, and individual route files such as productRoutes.js and cartRoutes.js.

6. Running the Application

- Provide commands to start the frontend and backend servers locally.
    - **Frontend:** npm start in the client directory.
    - **Backend:** npm start in the server directory.

7. API Documentation

- The API documentation for a grocery web app using React typically includes details on the available RESTful endpoints, authentication methods, request/response formats, and error handling. For example, the GET /api/products endpoint retrieves a list of all grocery items, while GET /api/products/:id returns the details of a specific product. Users can add items to their cart via the POST /api/cart endpoint, and view their current cart with GET /api/cart. For checkout, the POST /api/checkout endpoint processes the purchase.
- Each API call expects specific request payloads, such as product ID and quantity for adding to the cart or payment details for completing a purchase. Responses are typically in JSON format, including success messages or error codes like 400 Bad Request or 404 Not Found. Authentication may be handled via JWT tokens, which must be included in the Authorization header of protected requests.

8. Authentication

- Authentication in a grocery web app using React and the MERN stack typically relies on JSON Web Tokens (JWT). When users register (POST /api/auth/register) or log in (POST /api/auth/login), the server generates a JWT, which is returned to the client and stored in localStorage or sessionStorage. This token is included in the Authorization header as Bearer <jwt_token> for subsequent requests to protected endpoints, such as adding items to the cart or completing a purchase.
- The server verifies the token using middleware to ensure the user is authenticated. If the token expires, the user can refresh it using a refresh token (POST /api/auth/refresh). On logout, the token is removed from the client's storage. This ensures secure access to sensitive features while maintaining session persistence across the app.

9. User Interface

- The user interface (UI) of a grocery web app using React is designed to be intuitive and user-friendly, with a clean and responsive layout. The homepage features product categories, promotional banners, and a search bar for quick product lookup.
- Users can browse products on the product listing page, where they can filter, sort, and add items to their cart. The product details page provides in-depth information like images, descriptions, and pricing, along with options to adjust quantities and add to the cart.

- The shopping cart page displays selected items with quantities, prices, and a total, offering users the ability to modify their cart before proceeding to checkout. The checkout page allows users to review their order, enter shipping details, and provide payment information. For authentication, users can register, log in, and manage their profile.
- The app also includes a global search function for quick product discovery, and it is built to be fully responsive, ensuring a smooth experience across devices. The UI is composed of modular React components, with state management handled by hooks and context, providing a seamless and interactive shopping experience.

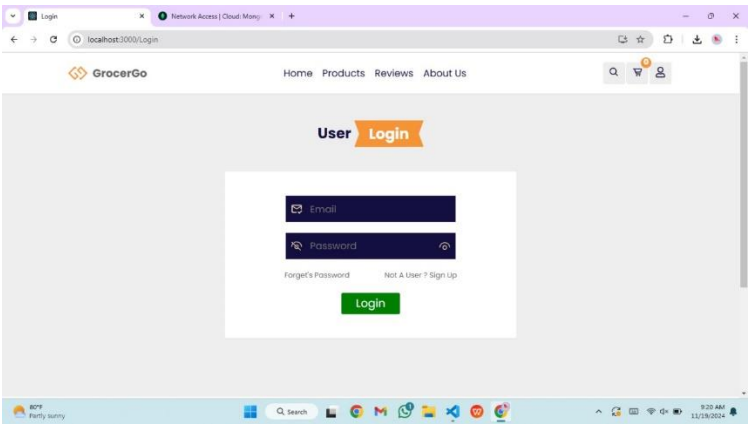
10. Testing

- Testing a grocery web app with **Cypress** involves using it for end-to-end testing to simulate real user interactions, such as browsing products, adding items to the cart, and completing the checkout process.
- It ensures the app's functionality works as expected across different user scenarios.

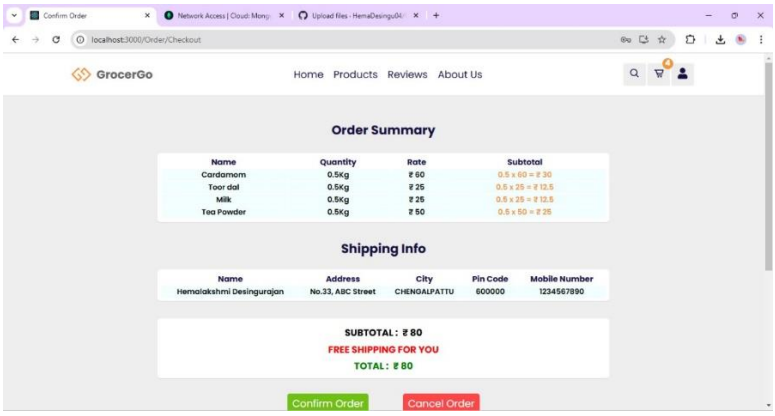
11. Screenshots or Demo

For User:

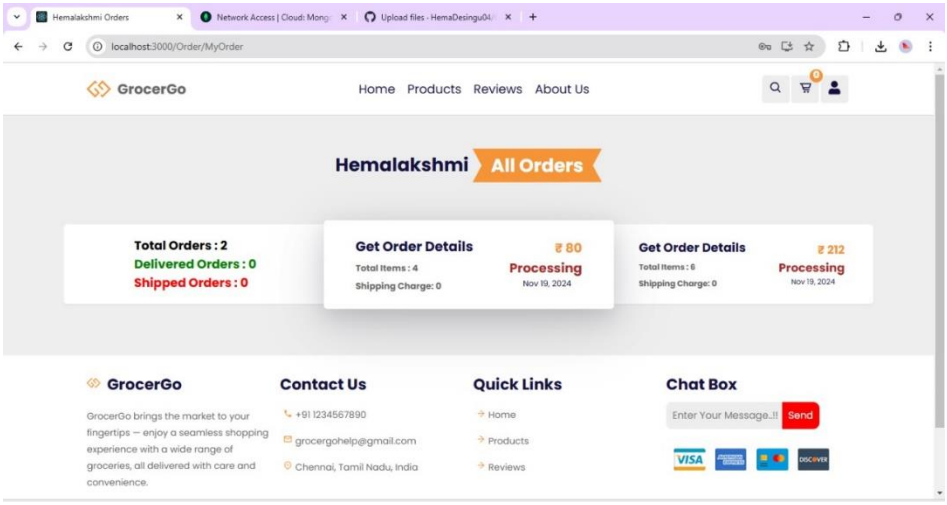
- Login Page of our project.



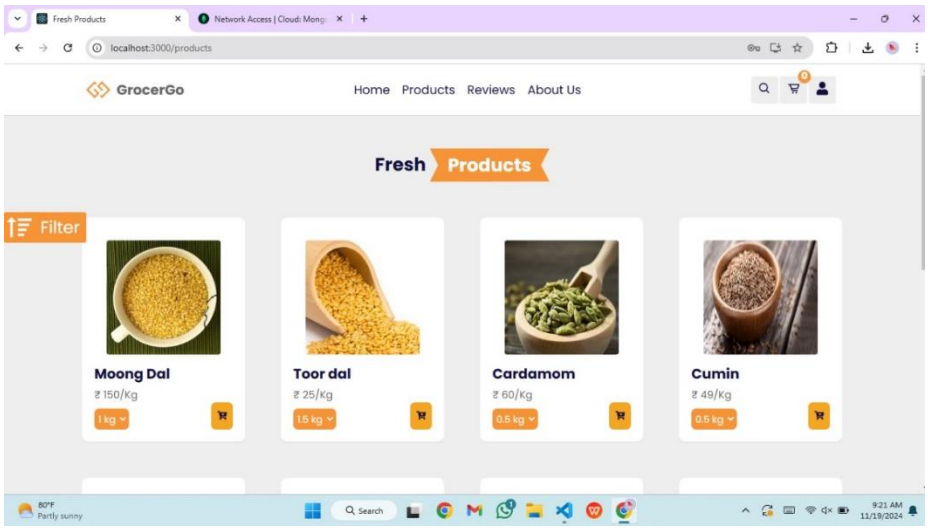
- Order summary and shipping information with total amount is displayed.



- Displaying the number of orders the customer have ordered previously.

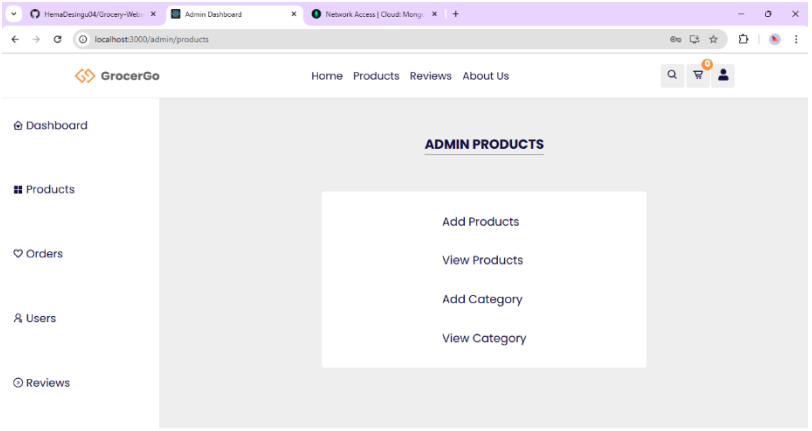


- Displaying the products list

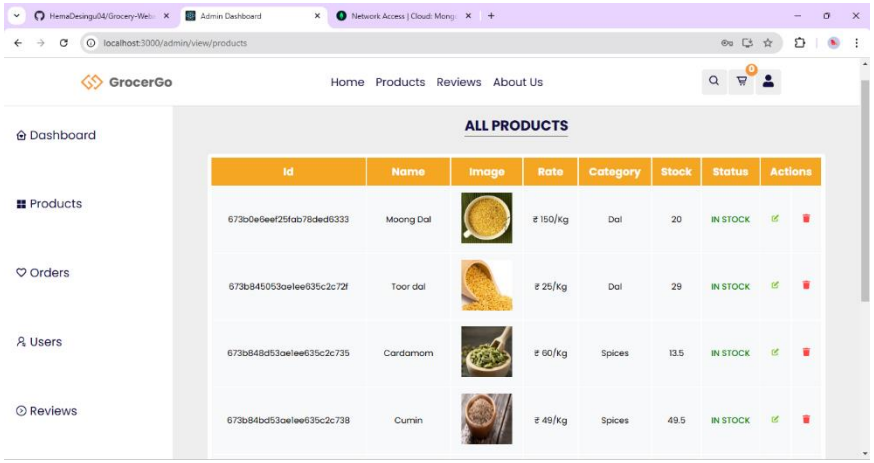


For Admin Dashboard

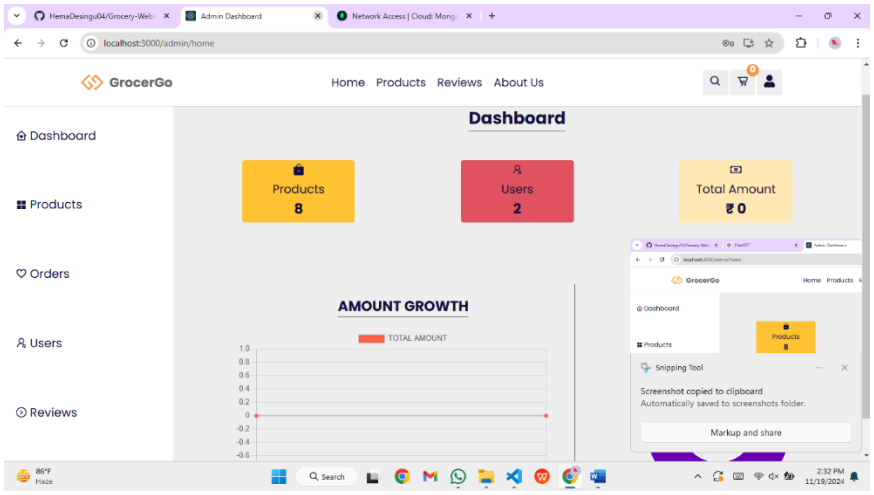
- Adding category and products



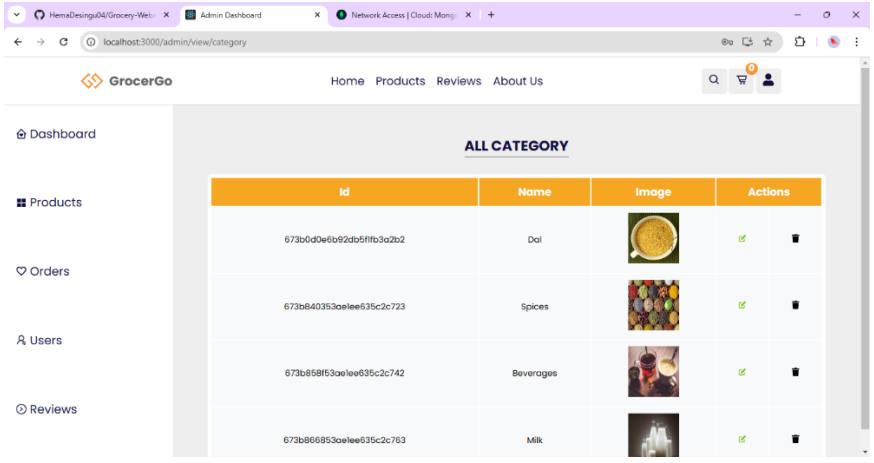
- Added products



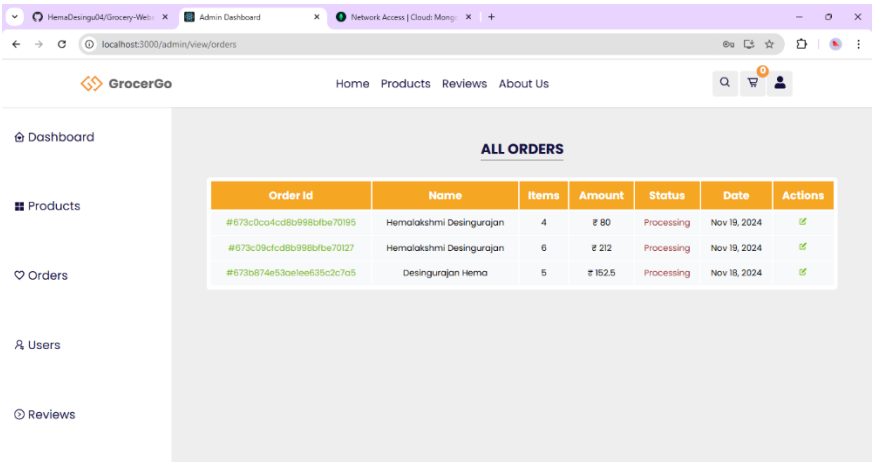
- Admin dashboard



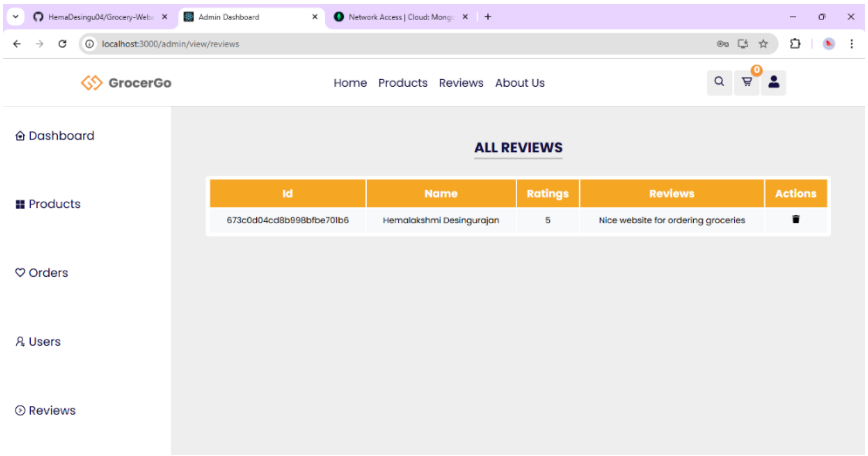
- Added Categories



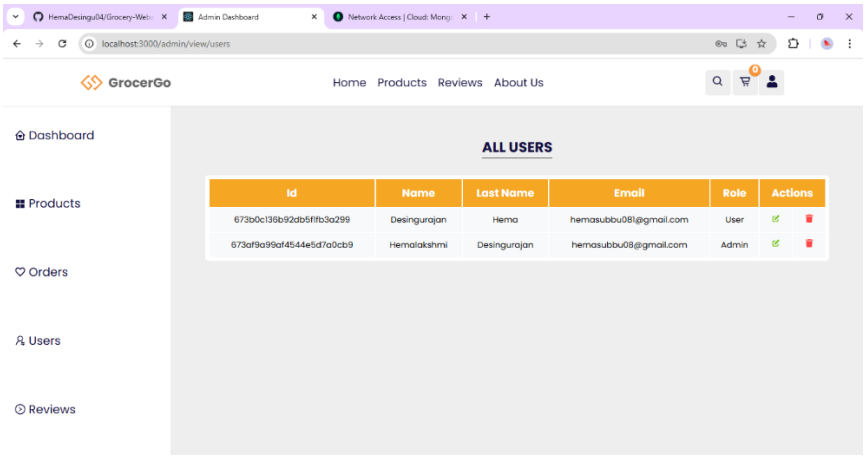
- All orders



- Review page



- Users of the webpage



12. Known Issues

- A known issue in the grocery web app could be inconsistent product images loading on slower networks, causing a delay in the page rendering.

13. Future Enhancements

- Our future enhancement for the grocery web app could include implementing real-time inventory updates to reflect stock changes instantly, improving the user experience during shopping.
- Additionally, integrating personalized product recommendations using machine learning algorithms could help users discover relevant products based on their browsing and purchase history.