**NAAN MUDHALVAN PROJECT**

**MERN STACK BY MONGODB**

**Project Title: GROCERY WEBAPP**

**Team Members:**

**DEEPIGHA SHRI CM - 113321243009**

**RAVURU VARSHITHA REDDY – 113321243041**

**PARVATHAREDDY HIMASREE - 113321243034**

**DEPARTMENT OF**

**ARTIFICIAL INTELLIGENCE AND DATA SCIENCE (FINAL YEAR)**

**VELAMMAL INSTITUTE OF TECHNOLOGY**

**PANCHETTI**

**CHENNAI – 601204**

**Index**

* **Project Overview**
* **Architecture**
* **Setup Instructions**
* **Folder Structure**
* **Running the Application**
* **API Documentation**
* **Authentication**
* **User Interface**
* **Testing**
* **Screenshots and Demo link**
* **Known Issues**
* **Future Enhancements**
* **Conclusion**

**1.Project Overview**

**Purpose:**The Grocery Webapp is designed to streamline grocery shopping by allowing users to browse items, add them to their cart, and place orders. The goal is to provide a seamless, user-friendly experience that simplifies grocery management, tracks order history, and provides real-time updates on stock and deals.

**Features:**

* + **Product Listings and Categories:** Organized sections for fruits, vegetables, dairy, and more.
  + **Shopping Cart and Checkout:** Secure cart management and payment options.
  + **User Authentication:** Sign-up, login, and account management.
  + **Order Tracking:** Access to past and current orders, with real-time status updates.
  + **Stock and Notification System:** Real-time stock status and notifications for offers.
  + **Dark and Light Mode:** Accessible interface with theme toggle.

### 2.Architecture

#### **Frontend (React)**

* **Component-Based Structure**: Organized with reusable components for modularity and scalability. Main components might include:
  + **ProductCard**: Displays product details and allows adding items to the cart.
  + **Cart**: Displays selected items, allows quantity adjustment, and calculates totals.
  + **OrderHistory**: Shows past orders and their statuses.
* **Pages and Routes**: Organized by routes using react-router-dom:
  + **Home**: Displays product categories and featured products.
  + **Product List**: Shows products by category, with filters and sorting options.
  + **Checkout**: Manages cart items and handles payment flow.
* **State Management**:
  + **Redux or Context API**: Manages global state for user data, cart items, product listings, and order statuses.
  + **Local Storage**: Stores cart data temporarily, even if the user leaves the page.
* **Responsive Design**:
  + **CSS and Flexbox/Grid**: Ensures responsive layout across desktop and mobile.

#### **Backend (Node.js and Express.js)**

* **RESTful API**: Organized with Express routes to manage various data operations:
  + **User Routes**: Handles registration, login, and profile management.
  + **Product Routes**: Manages product listings, categories, and stock availability.
* **Middleware**:
  + **Authentication Middleware**: Secures routes with JWT for user data protection.
  + **Error Handling Middleware**: Standardized error responses for debugging and user notifications.

#### **Database (MongoDB)**

* **Schema Design**:
  + **User Schema**: Stores user data, including encrypted passwords, contact information, and order history.
  + **Product Schema**: Defines product attributes (name, category, price, stock quantity, image URL).
* **Database Interactions**:
  + **Mongoose ORM**: Simplifies MongoDB queries and manages schema validation.
  + **Indexes**: Created on frequently queried fields, such as product categories and user email, for efficient data retrieval.

### 3. Setup Instructions

**Prerequisites:**

* + **Node.js**: JavaScript runtime.
  + **MongoDB**: NoSQL database for storing app data.
  + **Git**: For cloning the repository.

**Installation:**

* **Clone Repository**: git clone [repo URL].
* **Install Dependencies**: Run npm install in both /client and /server directories.
* **Environment Variables**:
* Create .env files in the root of /client and /server.
* Set up variables for MongoDB URI, JWT\_SECRET, API keys, and PORT.

### 4. Folder Structure

#### **Client (React)**

* /**src**: Main directory with organized subfolders:
* **/components**: Reusable UI components (e.g., Navbar, ProductCard, Cart).
* **/pages**: Main application pages (Home, Product List, Checkout).
* **/redux**: State management files (actions, reducers).
* **/styles**: Custom CSS or SCSS files.

#### **Server (Node.js and Express)**

* **/models**: Mongoose schemas (User.js, Product.js, Order.js).
* **/routes**: API route handlers (userRoutes.js, productRoutes.js, orderRoutes.js).
* **/controllers**: Business logic (userController.js, productController.js).
* **/middleware**: Authentication and error-handling middleware.
* **/config**: Configuration files (database connection, environment setup).

### 5. Running the Application

* **Frontend:** Run npm start in the /client directory.
* **Backend:** Run npm start in the /server directory.

### 6. API Documentation

* **GET /api/products**: Retrieves a list of products.
* **POST /api/cart**: Adds an item to the user's cart.
* **GET /api/orders/**: Retrieves order history.
* **Example**: Each endpoint includes request method, required parameters, headers, and sample responses.

### 7. Authentication

* **JWT Authentication**: Used for secure login sessions, storing tokens as HTTP-only cookies.
* **Authorization**: Role-based access control for admin and regular users.

### 8. User Interface

* **Screenshots and GIFs**: Display core pages (Home, Product List, Cart, Order History) to illustrate UI flow and theme options.

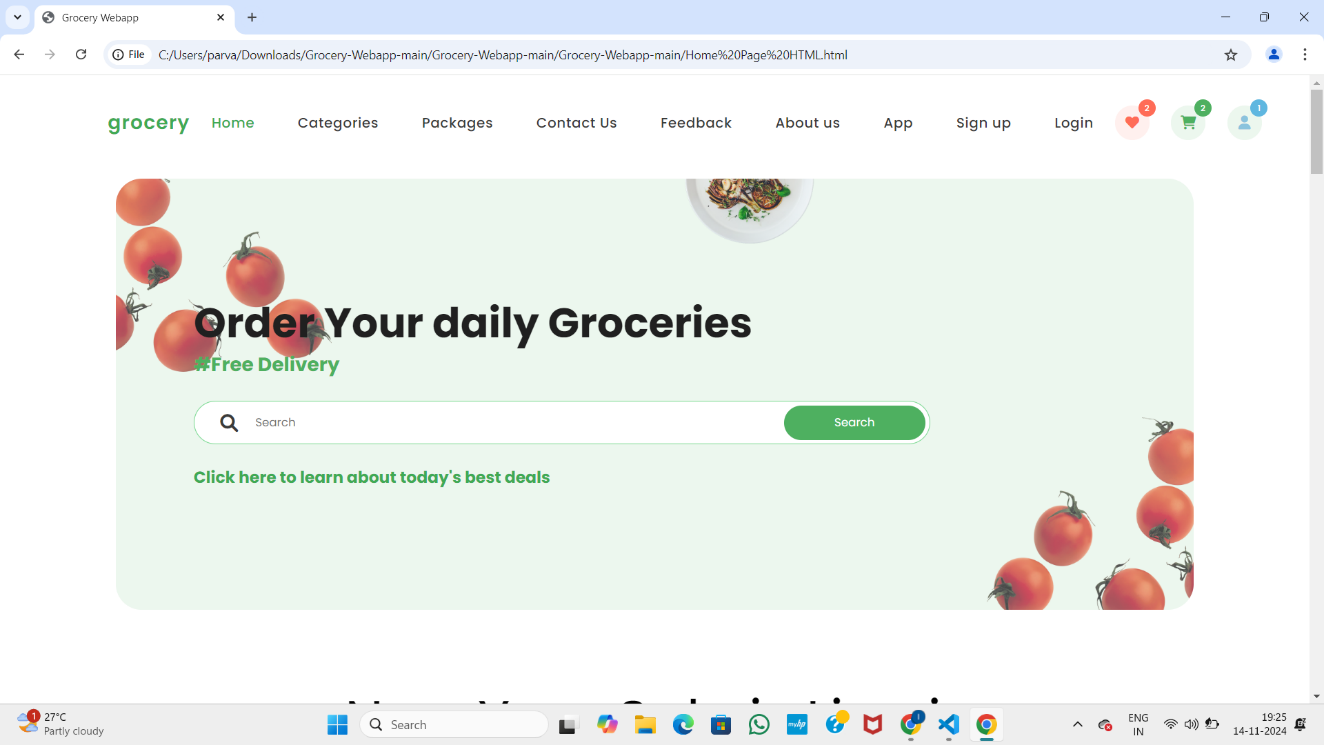
### 9. Testing

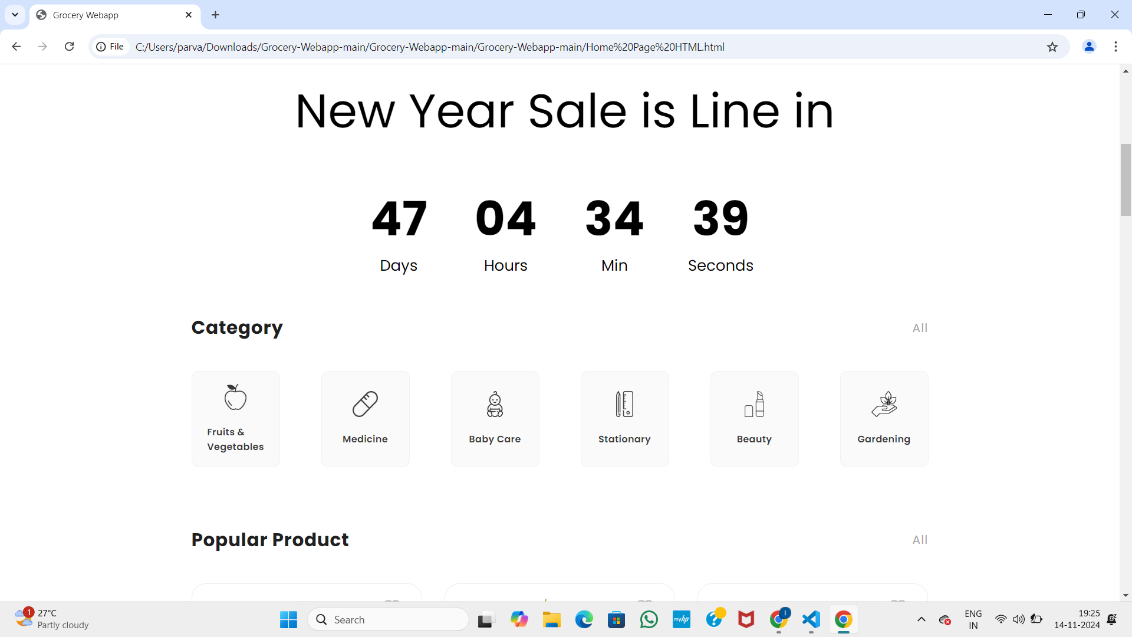
* **Frontend Testing**:
  + **React Testing Library**: For testing components in isolation.
  + **Jest**: For unit tests of Redux actions and reducers.
* **Backend Testing**:
  + **Mocha and Chai**: For testing API endpoints.
  + **Supertest**: For testing HTTP requests.

### 10. Screenshots and Demo link

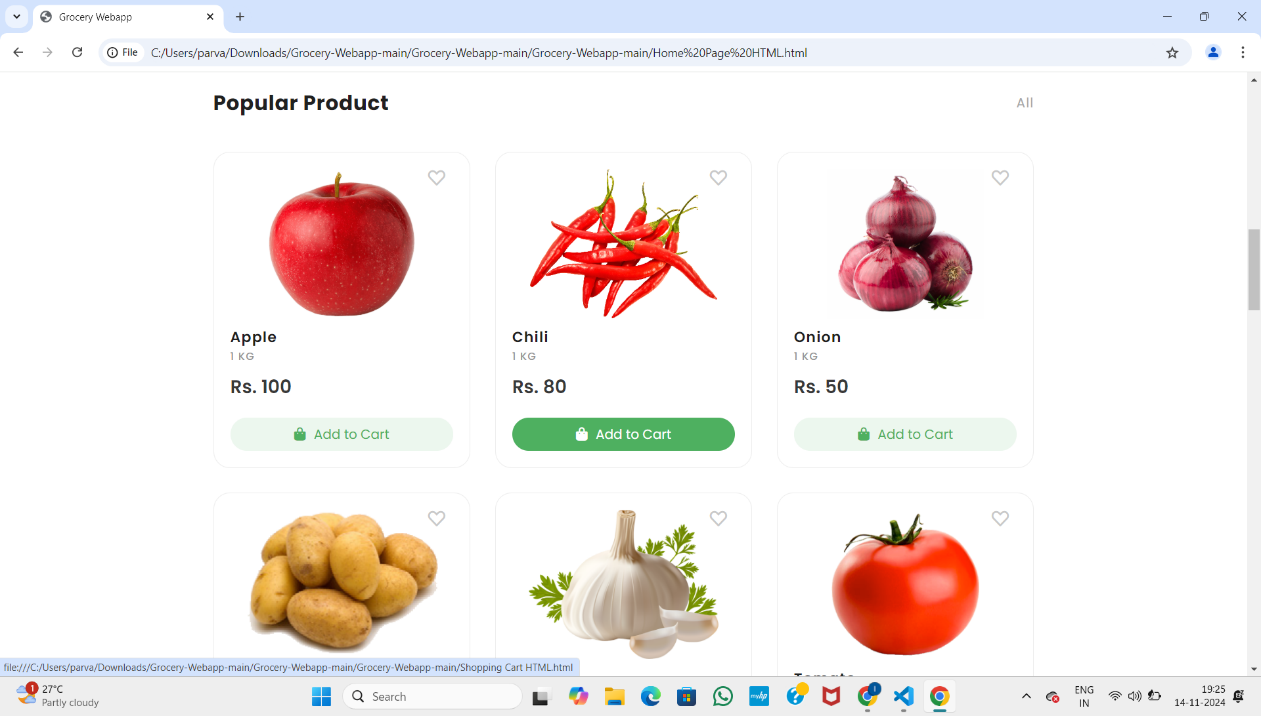
### Screenshots:

**HOME PAGE**

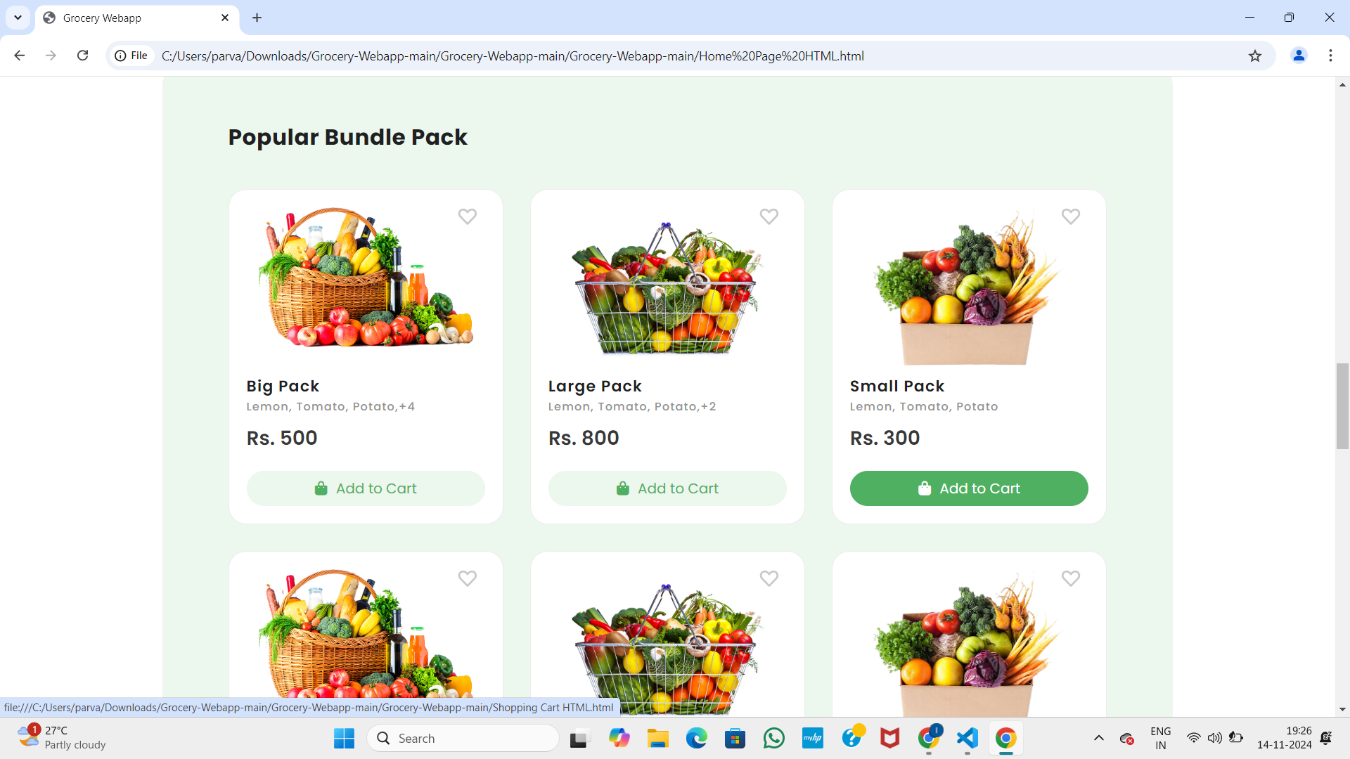


**CATEGORY and SALE COUNTDOWN PAGE**

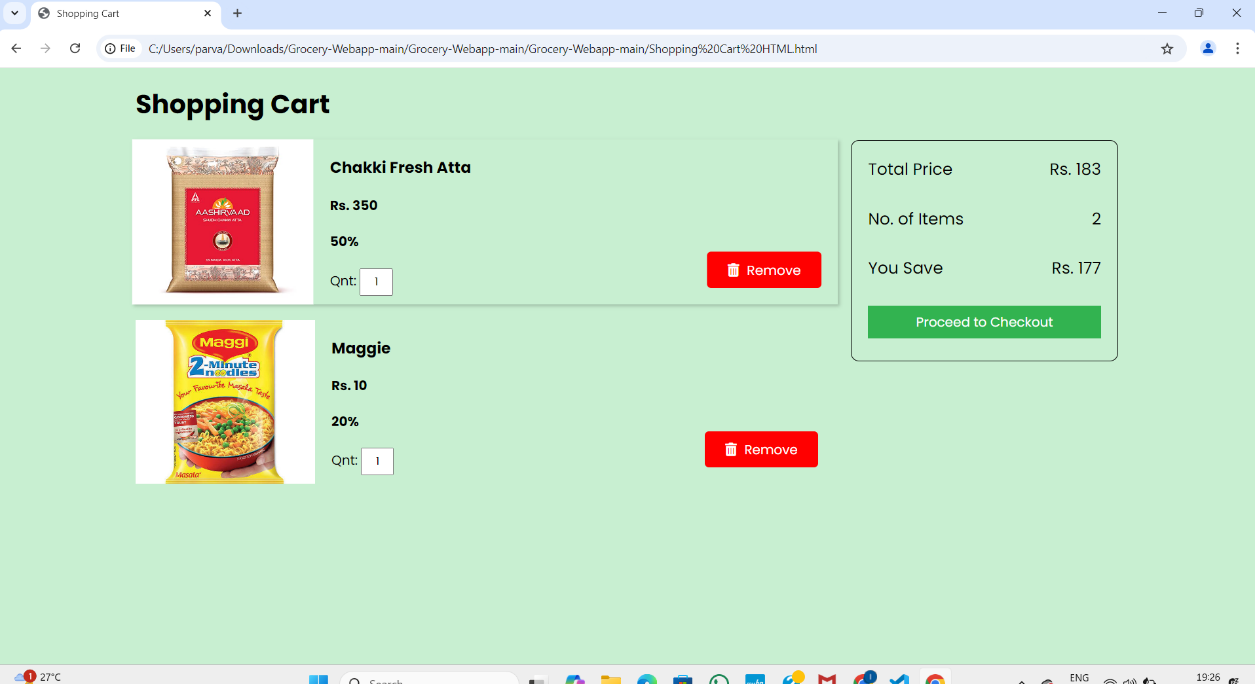
**POPULAR PRODUCT PAGE**



**POPULAR BUNDLE PACK PAGE**



**SHIPPING CART PAGE**

****

**DEMO LINK:**

[**GROCERY\_WEBAPP.MP4**](Grocery_WebApp.mp4)

### 11. Known Issues

* + **UI Glitches**: Certain layouts might need adjustments on smaller screens.
  + **Load Times**: Database query optimization might be needed for large datasets.

### 12. Future Enhancements

* **Wishlist**: Allow users to save items for future purchase.
* **Personalized Recommendations**: Suggest products based on previous purchases.
* **Delivery Tracking Integration**: Real-time delivery updates.
* **Voice Search**: Enable voice-activated product searches.
* **AI-Powered Deals**: Personalize discounts and notifications based on user behavior.

### 13.Conclusion

The Grocery Webapp showcases the MERN stack’s potential to build a modern, efficient, and user-focused shopping platform. Leveraging React for a dynamic, component-based frontend, Node.js and Express for a RESTful backend, and MongoDB for NoSQL database management, the application achieves seamless integration and effective data handling. Key features include organized product listings, cart and order management, real-time updates, JWT-based authentication for security, and a responsive, accessible UI with dark mode, stock notifications, and mobile compatibility. With a strong foundation and future plans for features like personalized recommendations, wishlist functionality, and delivery tracking, the Grocery Webapp is well-positioned to evolve with user needs and industry advancements.