Full Stack Development with MERN

Grocery Web-App

1. Introduction

Project Title: Grocery Web App

Team Members:

- S NAGA MEENAKSHI
- THUMMALURU CHANDANA

Welcome to our Grocery Web App, your one-stop shop for all your grocery needs! With our user-friendly interface and wide selection of high-quality products, we aim to make your grocery shopping experience convenient and enjoyable. Whether you're looking for fresh produce, pantry staples, or household essentials, our app has you covered. Explore our virtual aisles, add items to your cart with ease, and have your groceries delivered right to your doorstep.

2. Project Overview

Purpose:

ShopSmart is a modern, full-stack e-commerce application designed to provide users with a seamless online shopping experience. It allows users to browse products, add them to a shopping cart, place orders, and manage their purchase history. The application is built with a focus on a robust backend API and a dynamic, responsive frontend.

Key Features Implemented:

- User Management & Authentication
- User Registration: Allows new users to sign up for an account.

- User Login: Authenticates existing users, issuing a JSON Web Token (JWT) for secure session management.
- Logout: Securely terminates user sessions and clears local authentication data.
- Auth Middleware (Backend): Protects API routes, ensuring only authenticated users can access sensitive resources.
- HTTP Interceptor (Frontend): Automatically attaches the JWT to outgoing HTTP requests, simplifying authentication handling across the application.
- Product Catalog
- Product Listing: Displays a list of available products (assumed, as the frontend interacts with a /api/products route).
- Product Details: (Implicitly supported via cart population, as product details are fetched when viewing cart items).
- Shopping Cart Management
- Add to Cart: Users can add products to their shopping cart.
- Update Quantity: Users can increase or decrease the quantity of items in their cart.
- Remove Item: Users can remove specific products from their cart.
- Clear Cart: Users can empty their entire shopping cart.
- Persistent Cart: Cart contents are saved to the backend database, persisting across user sessions.
- Real-time Updates: Frontend cart state (items and total) is updated dynamically based on backend responses.
- Cart Count (Navbar): Displays the number of items in the cart in the application header.
- Order Processing
- Checkout Process: A dedicated page for users to review their cart, provide shipping details, and choose a payment method.
- Place Order: Submits the order to the backend, deducting product stock and creating an order record.
- Cash on Delivery (COD) Support: Orders can be placed without immediate Stripe payment processing.
- Order Status: Orders are assigned a status (e.g., 'pending', 'processing', 'cancelled').
- My Orders Page: Displays a list of all orders placed by the logged-in user.
- Cancel Order: Allows users to cancel orders within a specific status (e.g., 'pending', 'processing').

3. Architecture

- **Frontend:** Describe the frontend architecture using React.
- **Backend:** Outline the backend architecture using Node.js and Express.js.
- **Database:** Detail the database schema and interactions with MongoDB.

4. Setup Instructions

PREREQUISITES & INSTALLATIONS:

To develop a full-stack Ecommerce App for Furniture Tool using React js, Node.js, Express js and MongoDB, there are several prerequisites you should consider.

Here are the key prerequisites for developing such an application: Node.js and npm: Install Node.js, which includes npm (Node Package Manager), on your development machine. Node.js is required to run JavaScript on the server side.
• Download: https://nodejs.org/en/download/

• Installation manager/ instructions: https://nodejs.org/en/download/package MongoDB: Set up a MongoDB database to store hotel and booking information. Install MongoDB locally or use a cloud-based MongoDB service.

•Download: https://www.mongodb.com/try/download/community

•Installation instructions: https://docs.mongodb.com/manual/installation/ Express.js: Express.js is a web application framework for Node.js. Install Express.js to handle server-side routing, middleware, and API development.

• Installation: Open your command prompt or terminal and run the following command: npm install express

React js: React is a JavaScript library for building client-side applications. And Creating Single Page Web-Appliaction

Angular (Standalone Components): A modern, component-based framework for building dynamic single-page applications.

5. Folder Structure

• **Client:** Describe the structure of the React frontend.
• **Server:** Explain the organization of the Node.js backend.

6. Running the Application

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

7. API Documentation

Effective API documentation for our ShopSmart Grocery App backend is essential for developers. It concisely outlines all available API endpoints, their HTTP methods (GET, POST, PUT, DELETE), and required parameters.

For example, POST /api/products (for admins) needs a body with name, description, price, imageUrl, category (ID), and stock. POST /api/auth (login) requires email and password. The POST /api/users/admin-register endpoint additionally requires an admin_secret_key.

Crucially, documentation provides example JSON responses for success (e.g., a token, role, and user ID after login, or the newly created product details). It also clarifies which routes are protected by general authentication (auth middleware) and which require administrator privileges (adminAuth middleware), indicating that an x-auth-token header is necessary.

8. Authentication

1.Authentication (Verifying Who You Are):

- **Process:** When a user (or admin) logs in by sending their email and password to the POST /api/auth endpoint, the backend verifies their credentials against the database.
- **JWT Generation:** If the credentials are valid, the server generates a **JSON Web Token (JWT)**. This token contains a payload with essential user information, specifically their user.id and, crucially, their user.role (either 'user' or 'admin'). The token is then signed with a secret key (jwtSecret) stored on your server, ensuring its authenticity.
- **Frontend Storage:** This generated JWT is sent back to the frontend. Your Angular application's AuthService then stores this token (and the user's role) in the browser's localStorage.

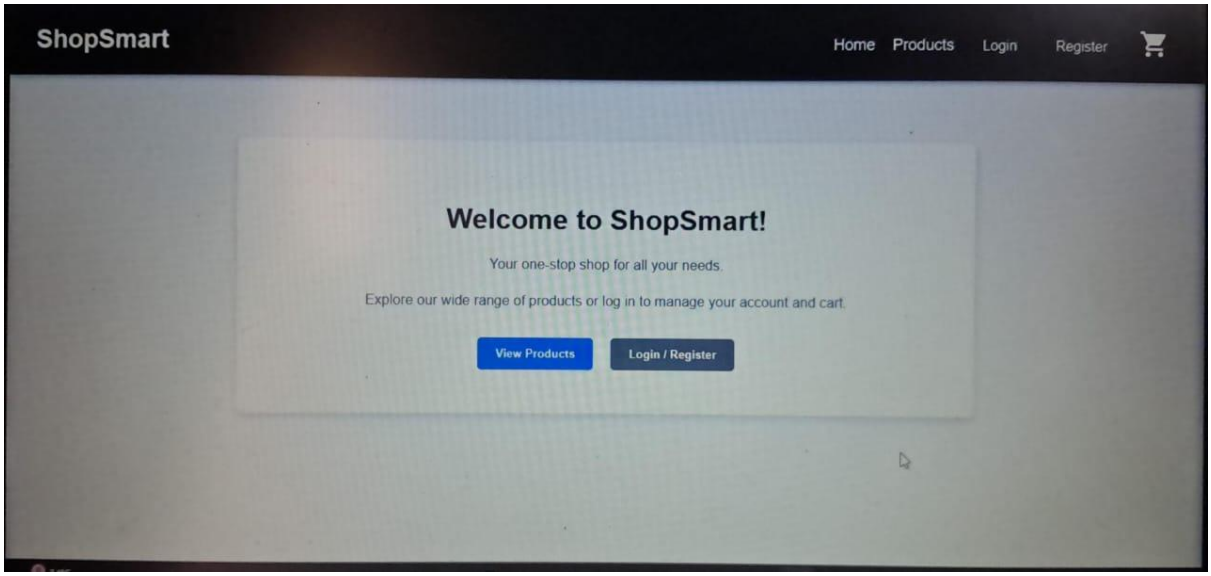
- **Subsequent Requests:** For nearly all subsequent requests to protected backend routes (like adding items to cart, fetching orders, or accessing admin features), the frontend includes this JWT in the x-auth-token header of the HTTP request.

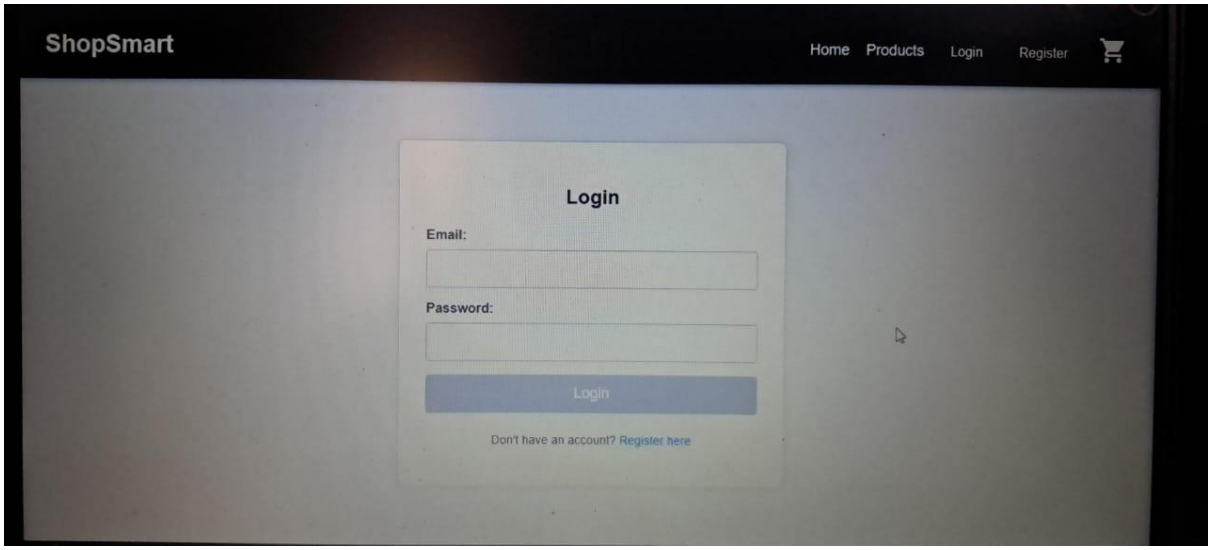
2. Authorization (What You're Allowed to Do):

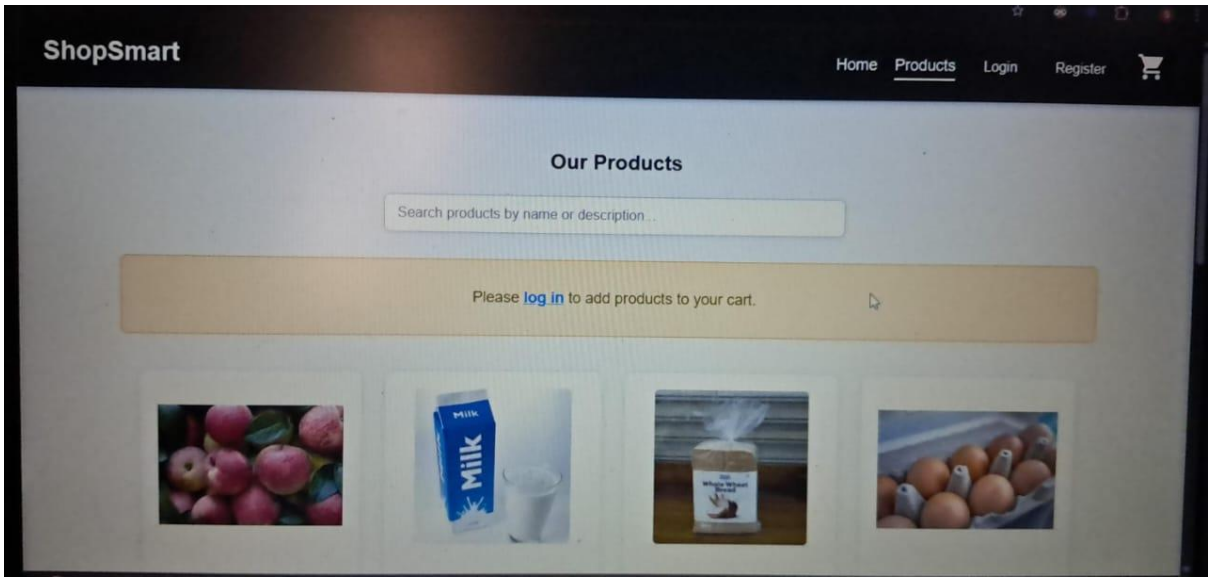
- **auth Middleware:** This is your general authentication middleware (from shopsmart-backend/middleware/auth.js). For any route that requires a user to be logged in (e.g., /api/cart, /api/orders), this middleware intercepts the request. It:
    1. Extracts the JWT from the x-auth-token header.
    2. Verifies the token's signature using your jwtSecret.
    3. If valid, it decodes the token and attaches the user's ID and role (e.g., req.user = { id: '...', role: '...' }) to the request object.
    4. Allows the request to proceed to the actual route handler. If the token is missing or invalid, it sends a 401 Unauthorized response.
- **adminAuth Middleware:** This is your specific authorization middleware (from shopsmart-backend/middleware/adminAuth.js). For routes that *only* administrators can access (e.g., POST /api/products, POST /api/users/admin-register), this middleware is used. It:

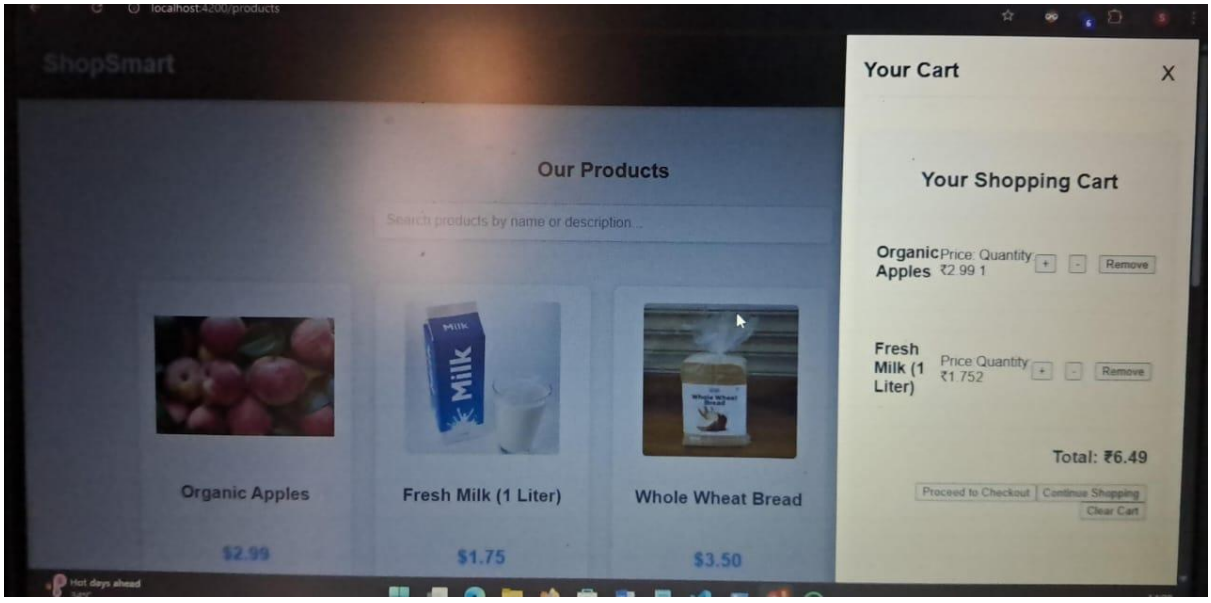
    1. Performs all the steps of the auth middleware first (authenticates the user).
    2. **Then, it checks the req.user.role property.** If req.user.role is not equal to 'admin', it immediately sends a 403 Forbidden ("Access denied: Admin role required") response, preventing non-admin users from accessing the route.
    3. Only if the user is authenticated *and* has the 'admin' role does the request proceed.
- **Frontend Role-Based UI:** On the frontend, your AuthService stores the user's role and exposes an observable (isAdmin$) which app.component.html subscribes to. This allows you to conditionally render UI elements, like the "Add Product" link, only when an admin is logged in.

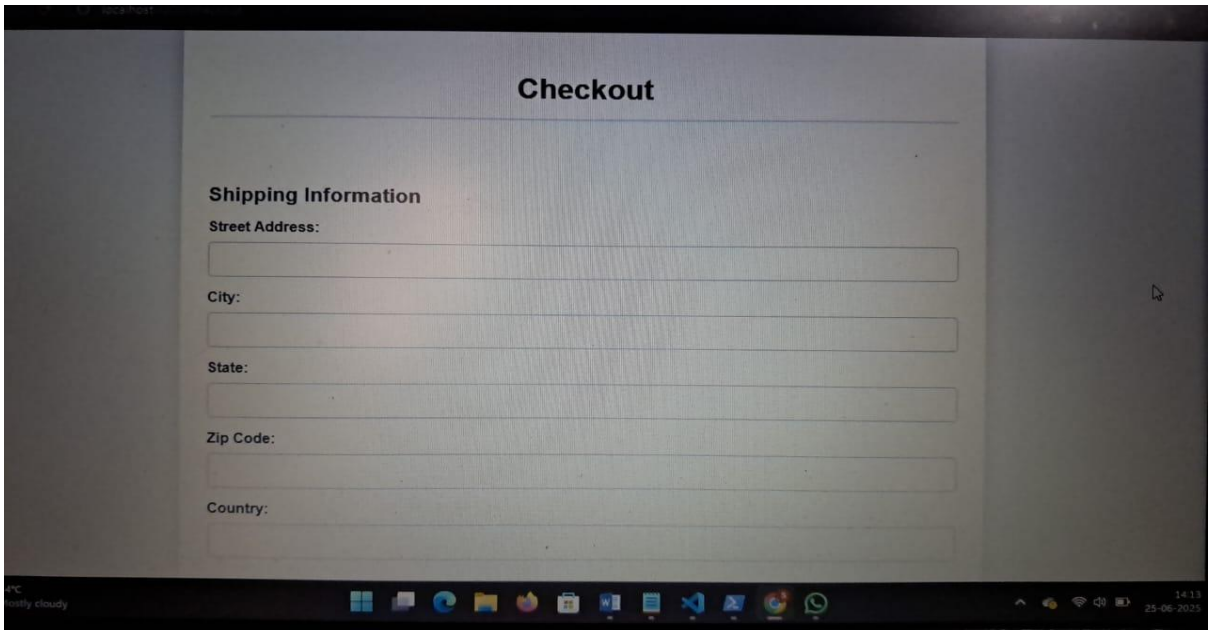
9. User Interface

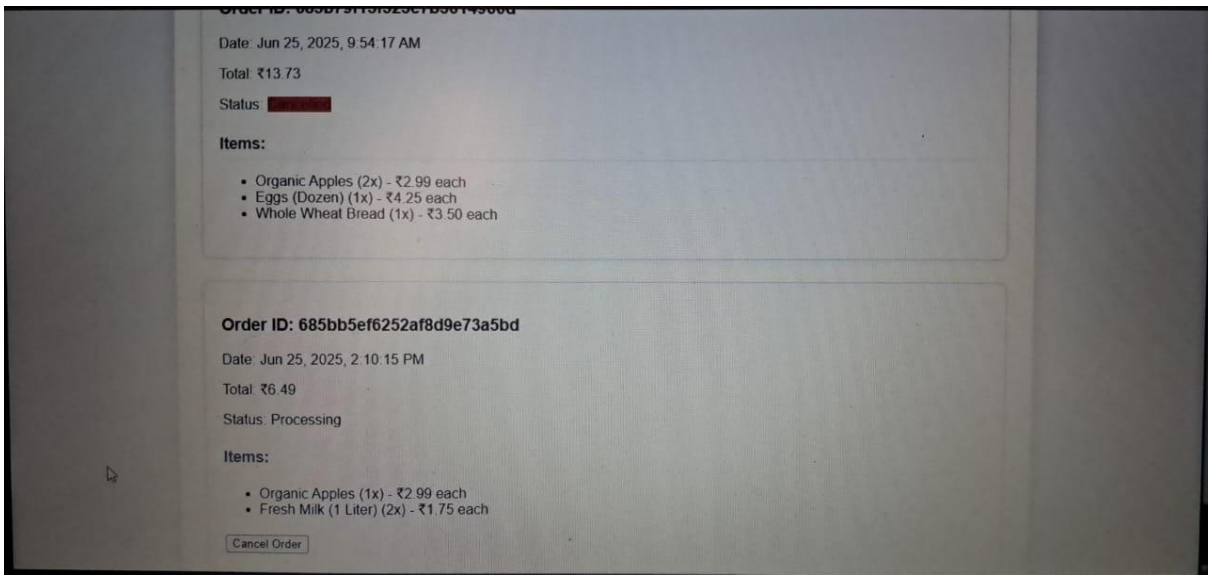

ShopSmart
Home
Products
Login
Register
Welcome to ShopSmart!
Your one-stop shop for all your needs.
Explore our wide range of products or log in to manage your account and cart.
View Products
Login / Register


ShopSmart
Home
Products
Login
Register
Login
Email:
Password:
Login
Don't have an account? Register here


ShopSmart
Home
Products
Login
Register
Our Products
Search products by name or description...
Please log in to add products to your cart.
Milk


ShopSmart
Our Products
Search products by name or description...
Organic Apples
$2.99
Fresh Milk (1 Liter)
$1.75
Whole Wheat Bread
$3.50
Your Cart
Your Shopping Cart
Organic Apples
Price: ₹2.99
Quantity: 1
Remove
Fresh Milk (1 Liter)
Price: ₹1.75
Quantity: 2
Remove
Total: ₹6.49
Proceed to Checkout
Continue Shopping
Clear Cart


Checkout
Shipping Information
Street Address:
City:
State:
Zip Code:
Country:


Date: Jun 25, 2025, 9:54:17 AM
Total: ₹13.73
Status:
Items:
Organic Apples (2x) - ₹2.99 each
Eggs (Dozen) (1x) - ₹4.25 each
Whole Wheat Bread (1x) - ₹3.50 each
Order ID: 685bb5ef6252af8d9e73a5bd
Date: Jun 25, 2025, 2:10:15 PM
Total: ₹6.49
Status: Processing
Items:
Organic Apples (1x) - ₹2.99 each
Fresh Milk (1 Liter) (2x) - ₹1.75 each
Cancel Order

10. Testing

The testing conducted during our development process has primarily been **manual and functional testing**. This involved:

- **Manual Backend Testing:** Using tools like Postman or Insomnia, or simple curl commands, to send requests to individual API endpoints (/api/users, /api/auth, /api/products, etc.) and verify that they return the expected responses and interact correctly with the MongoDB database. This was crucial for confirming successful user registration, login, product addition, cart management, and order processing.
- **Manual Frontend Testing:** Regularly running the Angular application in the browser (ng serve) and interacting with the UI elements (buttons, forms, navigation) to ensure they function as designed. This includes testing user registration, login, product browsing, adding items to the cart, modifying cart quantities, placing orders, viewing order history, and now, admin login and product submission.

While manual testing is essential for quick feedback during development, a robust application typically requires a more comprehensive automated testing strategy, which would involve:

1. **Unit Testing:**
    - **Backend (Node.js/Express): Mocha**, **Jest**, or **Chai** (for assertions) are common choices.
    - **Frontend (Angular): Jasmine** (the default testing framework for Angular CLI) and **Karma** (a test runner) are used for component, service, and pipe unit tests.
2. **Integration Testing:**
    - **Backend: Supertest** (often used with Mocha or Jest) for testing HTTP endpoints, verifying API responses, and checking database interactions.
    - **Frontend:** Angular's testing utilities allow for testing components and services in an integrated manner.
3. **End-to-End (E2E) Testing:**
    - **Frontend (with integration to backend): Cypress** is a popular modern choice for E2E testing Angular applications. Historically, **Protractor** was used with Angular but is being deprecated.

11. Demo

https://drive.google.com/file/d/18xWla1dgc26aun8iucLxe6nNnHMNLsbO/view?usp=sharing

12. Known Issues

- **No Password Reset/Forgot Password Functionality:** The application currently lacks a mechanism for users to reset or recover forgotten passwords. This is a standard feature for user convenience and security in production applications.
- **Client-Side Session Management:** User authentication relies on JWTs stored in localStorage. While convenient, localStorage is susceptible to XSS (Cross-Site Scripting) attacks. For heightened security, consider using HttpOnly cookies for storing JWTs or implementing refresh tokens.

13. Future Enhancements

Potential future enhancements for the Grocery Web App include implementing a robust product image upload system, expanding admin functionalities to include comprehensive category and user management UIs, and integrating real-time updates for product lists post-admin modifications. Enhancing user experience through features like personalized recommendations, advanced search filters, and a secure password reset mechanism would also be valuable. For a production environment, strengthening security by migrating JWT storage to HttpOnly cookies and implementing refresh tokens is a critical next step.