ShopSmart: Your Digital Grocery Store Experience – Project Documentation

This document provides a comprehensive overview of the "ShopSmart" full-stack MERN (MongoDB, Express.js, React, Node.js) application, detailing its purpose, architecture, setup, and key functionalities.

1. Introduction

- **Project Title:** ShopSmart: Your Digital Grocery Store Experience
- **Team ID :** LTVIP2026TMIDS35290
- **Team Size :** 4
- **Team Leader :** Navara Sai Divakar
- **Team member :** Mahesh Gangula
- **Team member :** Jayasurya Velidi
- **Team member :** Vigneswara Reddy Sabbella

2. Project Overview

- **Purpose:** ShopSmart is an e-commerce web application designed to offer users a convenient and efficient online platform for grocery shopping. Its primary goals are to facilitate product browsing, streamline the shopping cart and checkout process, and provide robust administrative tools for managing the store's inventory and user base.
- **Features:**
  - **User Management:**
    - Secure User Registration and Login/Logout functionality.
    - Password hashing for security.
    - User Profile viewing.
  - **Product Catalog:**
    - Browse and view a wide range of grocery products.
    - Search functionality to quickly find specific items.
    - Categorization of products for organized browsing.
  - **Shopping Cart:**
    - Add, remove, and update quantities of items in the shopping cart.
    - Persistent cart state for logged-in users.
  - **Order Management:**
    - Place new orders for items in the cart.
    - View detailed order history for individual users.
  - **Admin Panel:**
    - Dedicated administrator login for secure access.
    - **Product CRUD:** Create, Read, Update, and Delete products from the inventory.
    - **User CRUD:** View, create, update, and delete user accounts.

        - **Order Viewing:** Access and manage all customer orders.
    - **Secure Routing:** Implementation of protected routes and authentication middleware to ensure authorized access for users and administrators.

3. Architecture

- **Frontend:** The client-side of ShopSmart is built using **React.js**, a popular JavaScript library for building user interfaces.
    - **Component-Based:** The UI is structured using a modular, component-based architecture, promoting reusability and maintainability.
    - **React Router:** Handles client-side routing, enabling navigation between different pages without full page reloads (e.g., Home, Products, Cart, Admin Dashboard).
    - **Context API:** Utilized for global state management, providing shared data (e.g., user authentication status, cart items) across components without prop drilling.
    - **API Consumption:** Interacts with the backend via RESTful API calls to fetch data, send user inputs, and manage state.
- **Backend:** The server-side is developed with **Node.js** and the **Express.js** framework.
    - **RESTful API:** Provides a set of well-defined API endpoints that the frontend consumes.
    - **Middleware:** Employs Express middleware for request parsing, authentication (JWT verification), authorization (role-based access control), and error handling.
    - **Controllers:** Contains the core business logic for processing requests, interacting with the database, and sending responses.
    - **Routes:** Defines the API endpoints and maps them to their respective controller functions.
    - **Models:** Uses Mongoose (an ODM for MongoDB) to define data schemas and interact with the MongoDB database.
- **Database:MongoDB** is used as the primary NoSQL database for storing all application data.
    - **Cloud Hosting:** Leverages **MongoDB Atlas**, a cloud-based database service, for scalability, reliability, and ease of management.
    - **Collections (Schema Overview):**
        - **users:** Stores user details including username, email, password (hashed), role (e.g., 'user', 'admin'), and timestamps.
        - **products:** Stores information about each product, such as name, description, price, category, imageUrl, stockQuantity, and timestamps.
        - **categories:** Stores product category names.
        - **carts:** (Often implicitly handled via user/product interactions or temporary storage) Could store current user cart items.

        - **orders:** Stores details of customer orders, including userId, products (array of product IDs and quantities), totalAmount, shippingAddress, status (e.g., 'pending', 'shipped', 'delivered'), and timestamps.

4. Setup Instructions

To get the ShopSmart application running on your local machine, follow these steps:

- **Prerequisites:**
    - **Node.js:** Ensure Node.js (LTS version recommended) is installed. Download from https://nodejs.org/. npm (Node Package Manager) comes bundled with Node.js.
    - **MongoDB Atlas Account:** You will need an account and a cluster set up on MongoDB Atlas. Obtain your connection string (usually starts with mongodb+srv://).
    - **Git:** Install Git from https://git-scm.com/downloads.
- **Installation:**
    1. **Clone the repository:** Open your terminal or command prompt and run:
    2. git clone https://github.com/Dharmateja15/ShopSmart-Your-Digital-Grocery-Store-Experience.git

    3. **Navigate into the project root directory:**
    4. cd "ShopSmart-Your-Digital-Grocery-Store-Experience"

    5. **Navigate into the Project files directory:** All core application code resides here.
    6. cd "Project files"

    7. **Backend Setup:**
        - Navigate into the Backend directory:
        - cd Backend

        - **Install backend dependencies:**
        - npm install

        - **Create .env file:** In the Backend directory, create a new file named .env. This file will store your sensitive environment variables. **Important:** This file is listed in .gitignore and **will not be committed to GitHub**. Add the following content to your .env file, replacing the placeholder values:
        - MONGO_URI=your_mongodb_atlas_connection_string_here
        - JWT_SECRET=a_very_strong_random_secret_key_for_jwt
        - PORT=5100

(Replace your_mongodb_atlas_connection_string_here with your actual MongoDB Atlas connection string. Replace a_very_strong_random_secret_key_for_jwt with a long, random string. You can generate one online if needed.)

8. **Frontend Setup:**
    - Navigate back up to the Project files directory and then into Frontend:
    - cd ../Frontend

    - **Install frontend dependencies:**
    - npm install

    - **Configure Frontend Proxy:** To allow the frontend to communicate with the backend during development without CORS issues, open Frontend/package.json in your code editor. Add the following line at the top level of the JSON object (e.g., just after the "name" property):
    - "proxy": "http://localhost:5100",

    Save the package.json file.

    - **Create .env file (Optional):** If your frontend needs any public environment variables (e.g., for a payment gateway's *public* key), create a .env file in the Frontend directory (e.g., REACT_APP_PUBLIC_KEY=your_public_key).

5. Folder Structure

The project is organized into logical directories to separate concerns and improve maintainability.

- **Root Project Directory (ShopSmart-Your-Digital-Grocery-Store-Experience/)**
    - Document/: Contains project-related non-code documents, such as wireframes, design mockups, or project planning notes.
        - readme.md (Placeholder for documentation index)
    - Project files/: This directory houses the core application's backend and frontend codebases.
        - **Backend/ (Server-side Node.js/Express.js application)**
            - db/: Database connection configuration (connect.js) and initial data seeding/schema definitions (products.js, schema.js).
            - node_modules/: All installed Node.js packages. **(Ignored by Git)**
            - src/: Primary source code for the backend.
            - controllers/: Contains functions that handle the business logic for each API route.
            - models/: Defines Mongoose schemas for MongoDB collections (e.g., User, Product, Order).

            - routes/: Defines the API endpoints and maps them to the appropriate controller functions.
            - middleware/: Custom Express middleware for tasks like authentication (JWT verification), authorization, and error handling.
        - index.js: The main entry point file for starting the Express server.
        - package.json: Lists project metadata, scripts, and npm dependencies.
        - package-lock.json: Records the exact version of dependencies installed.
        - .env: Stores environment-specific variables like database URI and JWT secret. **(Ignored by Git)**
- **Frontend/ (Client-side React.js application)**
    - public/: Static assets that are served directly (e.g., index.html, favicon.ico, logo192.png, logo512.png, manifest.json, robots.txt).
    - src/: Contains all the React application's source code.
    - admin_components/: Reusable React components specifically designed for the administrative dashboard (e.g., AddCategory, AddProduct, AdminLogin, AdminNavbar, AdminProtectedRoute, AdminSignup, Dashboard, Orders, ProductItem, Products, Update, Users).
    - components/: General reusable React components used across the user-facing part of the application (e.g., About, Checkout, Contact, Footer, Header, History, Home, LoaderSpinner, Login, MyCart, MyOrders, NotFound, ProductItem, ProtectedRoute, Registration, products).
    - context/: Implements React Context API for global state management (e.g., context.js for user authentication, cart state).
    - assets/: (Potentially, if you add an assets folder) Images, icons, or other media.
    - App.js: The root component of the React application, defining overall layout and routing.
    - index.js: The entry point for rendering the React application to the DOM.
    - App.css, index.css: Global CSS styles.
    - logo.svg, reportWebVitals.js, setupTests.js: Standard files from Create React App.
    - node_modules/: All installed npm packages for the frontend. **(Ignored by Git)**
    - package.json: Lists project metadata, scripts, and npm dependencies.
    - package-lock.json: Records the exact version of dependencies installed.
    - .env: Stores environment variables for the frontend. **(Ignored by Git)**

- Video demo/: Contains multimedia files like video demonstrations or screen recordings of the application in action.
    - readme.md (Placeholder for demo index)

    - .gitignore: Crucial file specifying files and directories that Git should explicitly ignore (e.g., node_modules/, .env files).
    - README.md: The main README file for the entire project, providing a high-level overview, setup instructions, and deployment information.

6. Running the Application

To run the ShopSmart application locally, you need to start both the backend and frontend servers independently. Ensure you have followed all steps in the Setup Instructions section.

- **1. Start the Backend Server:**
    - Open your first terminal or command prompt window.
    - Navigate to the backend project directory:
    - cd "ShopSmart-Your-Digital-Grocery-Store-Experience/Project files/Backend"
    - Execute the command to start the Node.js Express server:
    - npm start

    You should see output similar to Server running at http://localhost:5100 and Connection successful (indicating database connection). Keep this terminal open and running.
- **2. Start the Frontend Application:**
    - Open a **second, separate** terminal or command prompt window.
    - Navigate to the frontend project directory:
    - cd "ShopSmart-Your-Digital-Grocery-Store-Experience/Project files/Frontend"
    - Execute the command to start the React development server:
    - npm start

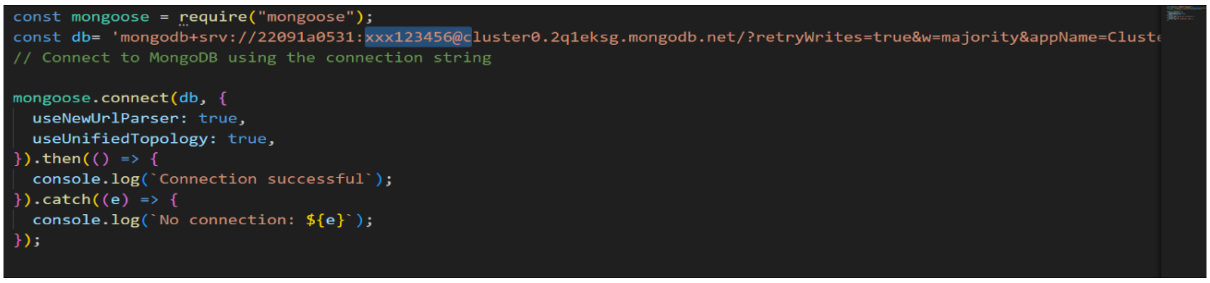
    This command will compile your React application and typically automatically open a new tab in your default web browser to http://localhost:3000 (or another available port).

    - Once both servers are running, you can interact with the ShopSmart application in your browser. The frontend will communicate with the backend API.

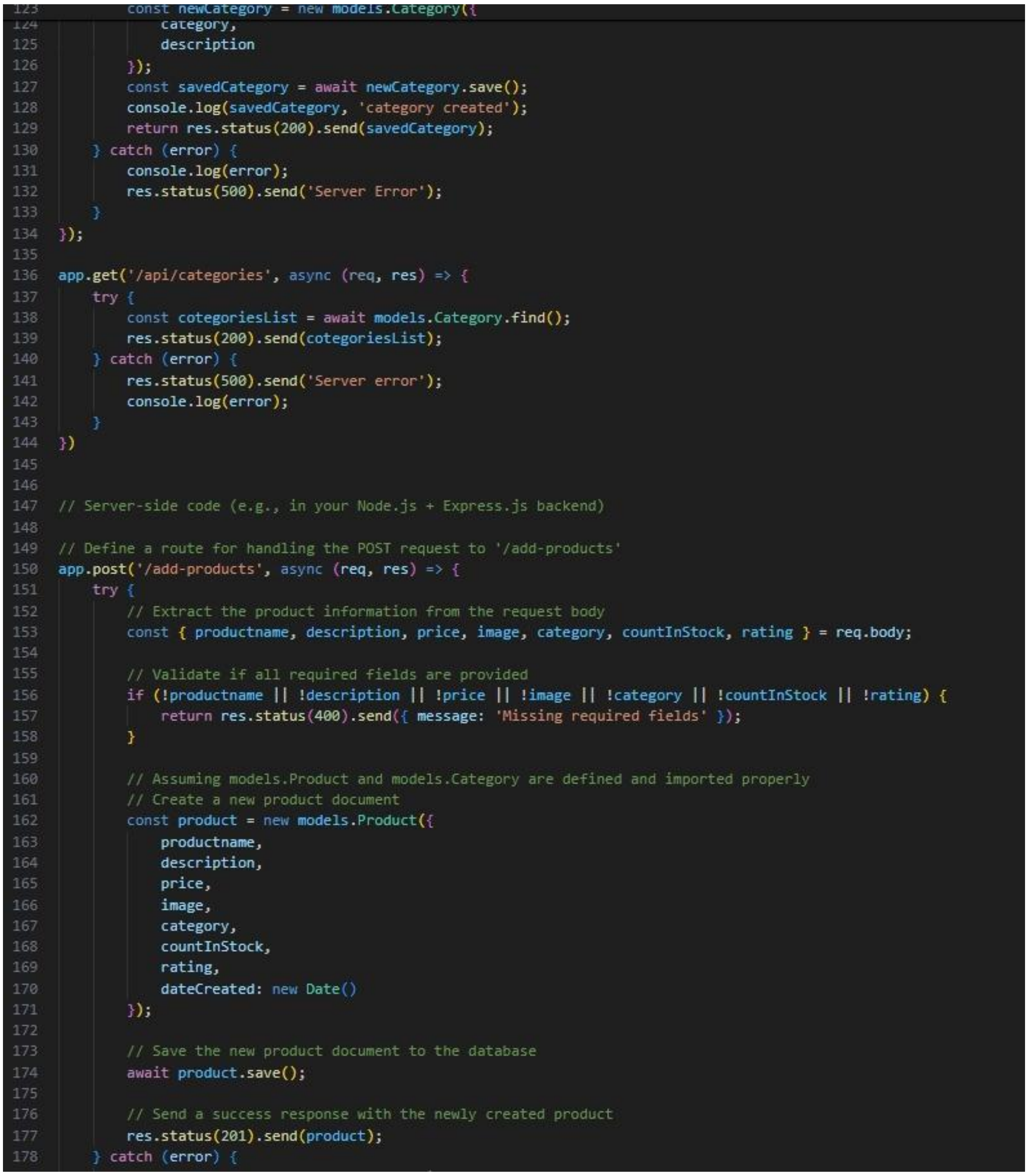
7. API Documentation

This section details the RESTful API endpoints exposed by the Node.js/Express.js backend. All requests should be sent to the base URL (during local development, this is http://localhost:5100/api).

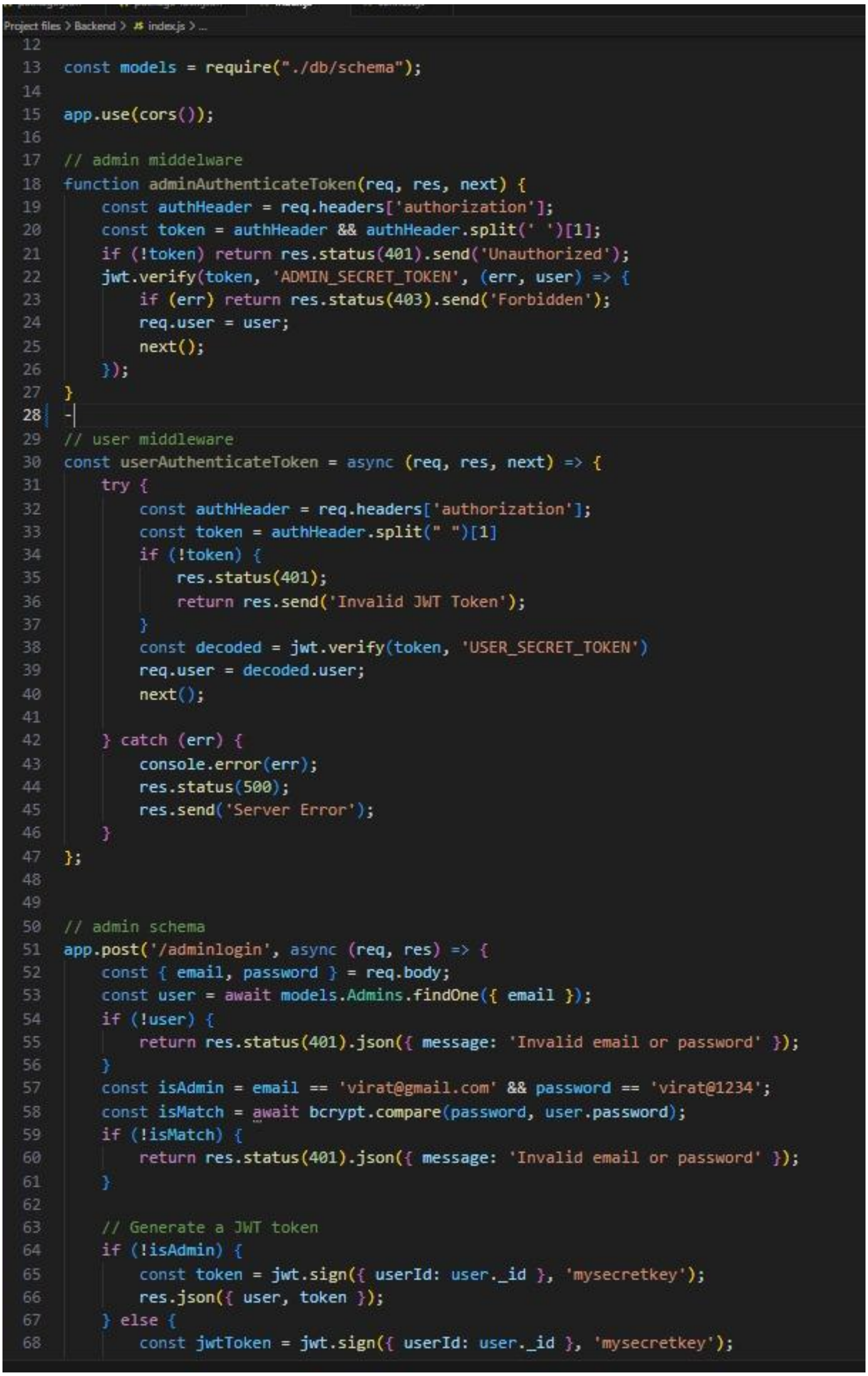
Common Headers (for Protected Routes): Authorization: Bearer <JWT_TOKEN> (Replace <JWT_TOKEN> with the token obtained from login)



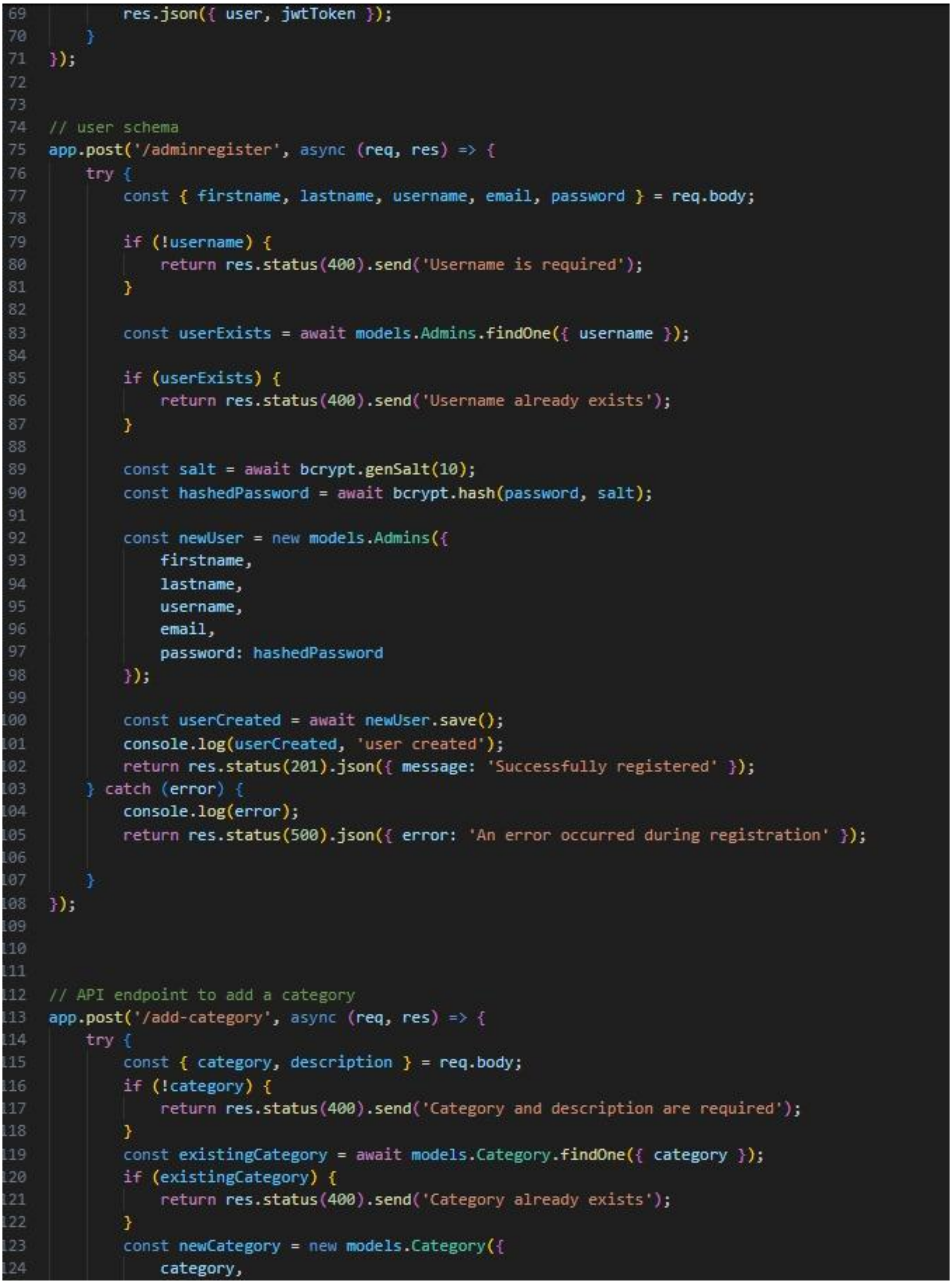
```
const mongoose = require("mongoose");
const db= 'mongodb+srv://22091a0531:xxx123456@cluster0.2q1eksg.mongodb.net/?retryWrites=true&w=majority&appName=Clust
// Connect to MongoDB using the connection string

mongoose.connect(db, {
  useNewUrlParser: true,
  useUnifiedTopology: true,
}).then(() => {
  console.log(`Connection successful`);
}).catch((e) => {
  console.log(`No connection: ${e}`);
});
```



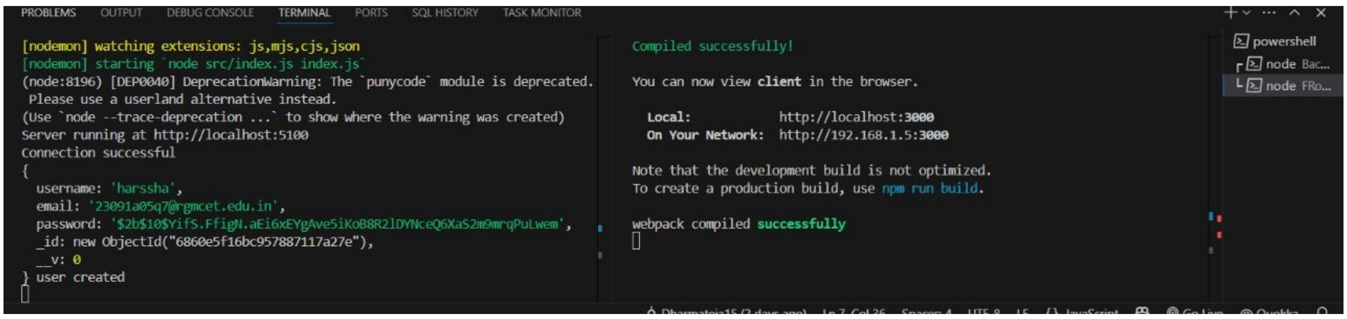
```
        const newCategory = new models.Category({
            category,
            description
        });
        const savedCategory = await newCategory.save();
        console.log(savedCategory, 'category created');
        return res.status(200).send(savedCategory);
    } catch (error) {
        console.log(error);
        res.status(500).send('Server Error');
    }
});

app.get('/api/categories', async (req, res) => {
    try {
        const cotegoriesList = await models.Category.find();
        res.status(200).send(cotegoriesList);
    } catch (error) {
        res.status(500).send('Server error');
        console.log(error);
    }
})


// Server-side code (e.g., in your Node.js + Express.js backend)

// Define a route for handling the POST request to '/add-products'
app.post('/add-products', async (req, res) => {
    try {
        // Extract the product information from the request body
        const { productname, description, price, image, category, countInStock, rating } = req.body;

        // Validate if all required fields are provided
        if (!productname || !description || !price || !image || !category || !countInStock || !rating) {
            return res.status(400).send({ message: 'Missing required fields' });
        }

        // Assuming models.Product and models.Category are defined and imported properly
        // Create a new product document
        const product = new models.Product({
            productname,
            description,
            price,
            image,
            category,
            countInStock,
            rating,
            dateCreated: new Date()
        });

        // Save the new product document to the database
        await product.save();

        // Send a success response with the newly created product
        res.status(201).send(product);
    } catch (error) {
```


Project files > Backend > JS index.js > ...

```
const models = require("./db/schema");

app.use(cors());

// admin middelware
function adminAuthenticateToken(req, res, next) {
    const authHeader = req.headers['authorization'];
    const token = authHeader && authHeader.split(' ')[1];
    if (!token) return res.status(401).send('Unauthorized');
    jwt.verify(token, 'ADMIN_SECRET_TOKEN', (err, user) => {
        if (err) return res.status(403).send('Forbidden');
        req.user = user;
        next();
    });
}
-
// user middleware
const userAuthenticateToken = async (req, res, next) => {
    try {
        const authHeader = req.headers['authorization'];
        const token = authHeader.split(" ")[1]
        if (!token) {
            res.status(401);
            return res.send('Invalid JWT Token');
        }
        const decoded = jwt.verify(token, 'USER_SECRET_TOKEN')
        req.user = decoded.user;
        next();

    } catch (err) {
        console.error(err);
        res.status(500);
        res.send('Server Error');
    }
};


// admin schema
app.post('/adminlogin', async (req, res) => {
    const { email, password } = req.body;
    const user = await models.Admins.findOne({ email });
    if (!user) {
        return res.status(401).json({ message: 'Invalid email or password' });
    }
    const isAdmin = email == 'virat@gmail.com' && password == 'virat@1234';
    const isMatch = await bcrypt.compare(password, user.password);
    if (!isMatch) {
        return res.status(401).json({ message: 'Invalid email or password' });
    }

    // Generate a JWT token
    if (!isAdmin) {
        const token = jwt.sign({ userId: user._id }, 'mysecretkey');
        res.json({ user, token });
    } else {
        const jwtToken = jwt.sign({ userId: user._id }, 'mysecretkey');
```



```
        res.json({ user, jwtToken });
    }
});


// user schema
app.post('/adminregister', async (req, res) => {
    try {
        const { firstname, lastname, username, email, password } = req.body;

        if (!username) {
            return res.status(400).send('Username is required');
        }

        const userExists = await models.Admins.findOne({ username });

        if (userExists) {
            return res.status(400).send('Username already exists');
        }

        const salt = await bcrypt.genSalt(10);
        const hashedPassword = await bcrypt.hash(password, salt);

        const newUser = new models.Admins({
            firstname,
            lastname,
            username,
            email,
            password: hashedPassword
        });

        const userCreated = await newUser.save();
        console.log(userCreated, 'user created');
        return res.status(201).json({ message: 'Successfully registered' });
    } catch (error) {
        console.log(error);
        return res.status(500).json({ error: 'An error occurred during registration' });

    }
});



// API endpoint to add a category
app.post('/add-category', async (req, res) => {
    try {
        const { category, description } = req.body;
        if (!category) {
            return res.status(400).send('Category and description are required');
        }
        const existingCategory = await models.Category.findOne({ category });
        if (existingCategory) {
            return res.status(400).send('Category already exists');
        }
        const newCategory = new models.Category({
            category,
```



```
[nodemon] watching extensions: js,mjs,cjs,json
[nodemon] starting `node src/index.js index.js`
(node:8196) [DEP0040] DeprecationWarning: The `punycode` module is deprecated.
 Please use a userland alternative instead.
(Use `node --trace-deprecation ...` to show where the warning was created)
Server running at http://localhost:5100
Connection successful
{
  username: 'harssha',
  email: '23091a05q7@rgmcet.edu.in',
  password: '$2b$10$YifS.FfigN.aEi6xEYgAve5iKoB8R2lDYNceQ6XaS2m9mrqPuLwem',
  _id: new ObjectId("6860e5f16bc957887117a27e"),
  __v: 0
} user created
```

```
Compiled successfully!

You can now view client in the browser.

  Local:            http://localhost:3000
  On Your Network:  http://192.168.1.5:3000

Note that the development build is not optimized.
To create a production build, use npm run build.

webpack compiled successfully
```

8. Authentication

Authentication and authorization in ShopSmart are primarily handled using **JSON Web Tokens (JWTs)**, providing a stateless and secure mechanism for user verification.

- **Login/Registration Flow:**
    1. When a user registers or logs in via the frontend, their credentials are sent to the backend's authentication (/api/auth) endpoints.
    2. The backend verifies the credentials against the users collection in MongoDB.
    3. Upon successful verification, the backend generates a signed JWT. This token contains essential user information (e.g., _id, email, role) but *not* sensitive data like passwords.
    4. The generated JWT is sent back to the frontend.
- **Token Storage & Usage (Frontend):**
    1. The frontend stores the received JWT, typically in localStorage or sessionStorage for persistence across browser sessions/tabs.
    2. For subsequent requests to protected backend API endpoints, the frontend includes this JWT in the Authorization header of the HTTP request, in the format Bearer <YOUR_JWT_TOKEN>.
- **Authentication Middleware (Backend):**
    1. The backend utilizes custom middleware (e.g., authMiddleware.js) that intercepts requests to protected routes.
    2. This middleware extracts the JWT from the Authorization header.
    3. It then verifies the token's authenticity and expiration using the JWT_SECRET stored in the backend's .env file.
    4. If the token is valid, the decoded user information (ID, role) is attached to the req.user object, making it accessible to subsequent controller functions.
    5. If the token is missing or invalid, the middleware responds with a 401 Unauthorized status.
- **Authorization (Role-Based Access Control - Backend):**
    1. Beyond authentication, an authorization middleware (e.g., adminAuthMiddleware.js) checks the role property from the decoded JWT (available in req.user).
    2. For routes requiring administrator privileges (e.g., product creation, user deletion), this middleware ensures that req.user.role is 'admin'.
    3. If the user's role does not meet the requirement, a 403 Forbidden status is returned.

- **Frontend Protected Routes:**
    1. React Router is configured with custom ProtectedRoute components (e.g., components/ProtectedRoute/index.js, admin_components/AdminProtectedRoute/index.js).
    2. These components check for the presence and a basic validity (e.g., token existence) of the JWT in localStorage before rendering the protected UI components. If the token is not present or invalid, they redirect the user to the login page.

9. User Interface

(This section is designed for visual representation of your application. You would embed screenshots or GIFs here.)

- **Login Page:***(Insert Screenshot/GIF)*
    - Description: Displays the user authentication form for existing users.
- **Registration Page:***(Insert Screenshot/GIF)*
    - Description: Allows new users to create an account.
- **Home Page (Product Catalog):***(Insert Screenshot/GIF)*
    - Description: The main landing page showcasing available grocery products.
- **Product Detail Page:***(Insert Screenshot/GIF)*
    - Description: Provides detailed information about a selected product.
- **Shopping Cart:***(Insert Screenshot/GIF)*
    - Description: Shows items currently added to the user's cart, with options to adjust quantities or remove items.
- **Checkout Flow:***(Insert Screenshot/GIF)*
    - Description: Steps involved in placing an order, including shipping details and payment method selection.
- **User Order History:***(Insert Screenshot/GIF)*
    - Description: Displays a list of all past orders placed by the logged-in user.
- **Admin Dashboard:***(Insert Screenshot/GIF)*
    - Description: The central hub for administrators, providing navigation to manage products, users, and orders.
- **Admin Product Management:***(Insert Screenshot/GIF)*
    - Description: Interface for administrators to add new products, update existing ones, or delete products.
- **Admin User Management:***(Insert Screenshot/GIF)*
    - Description: Interface for administrators to view, edit, or delete user accounts.

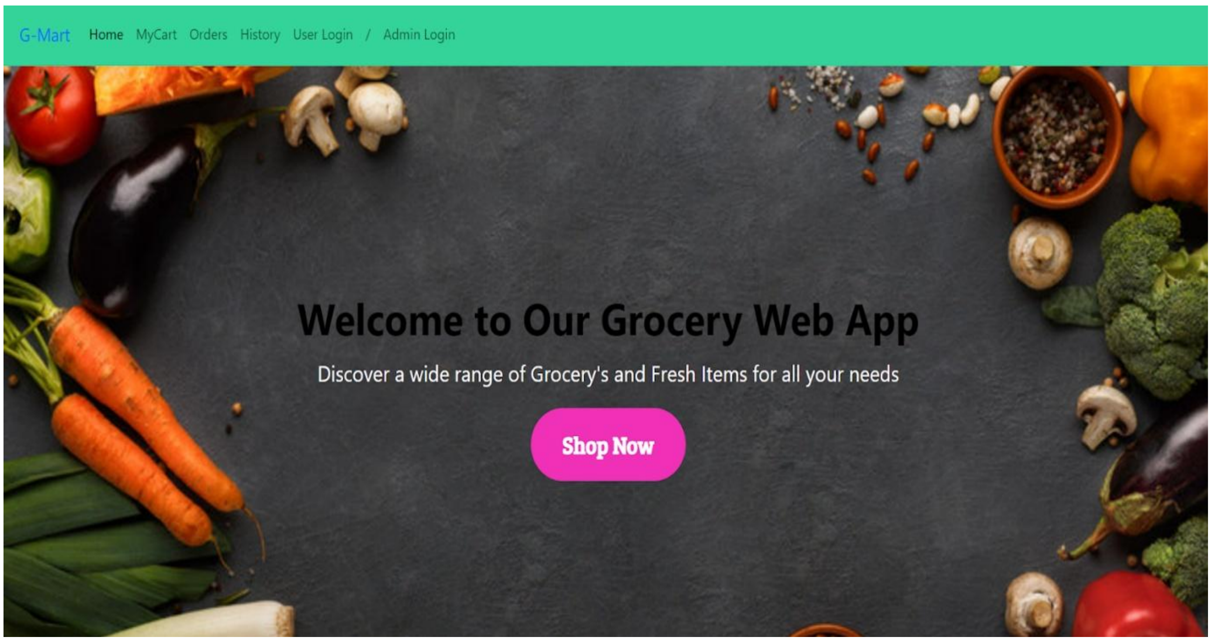
10. Testing

(This section depends heavily on what testing you've actually implemented. If no formal testing framework was used, state that and perhaps mention manual testing, or indicate it as a future enhancement.)

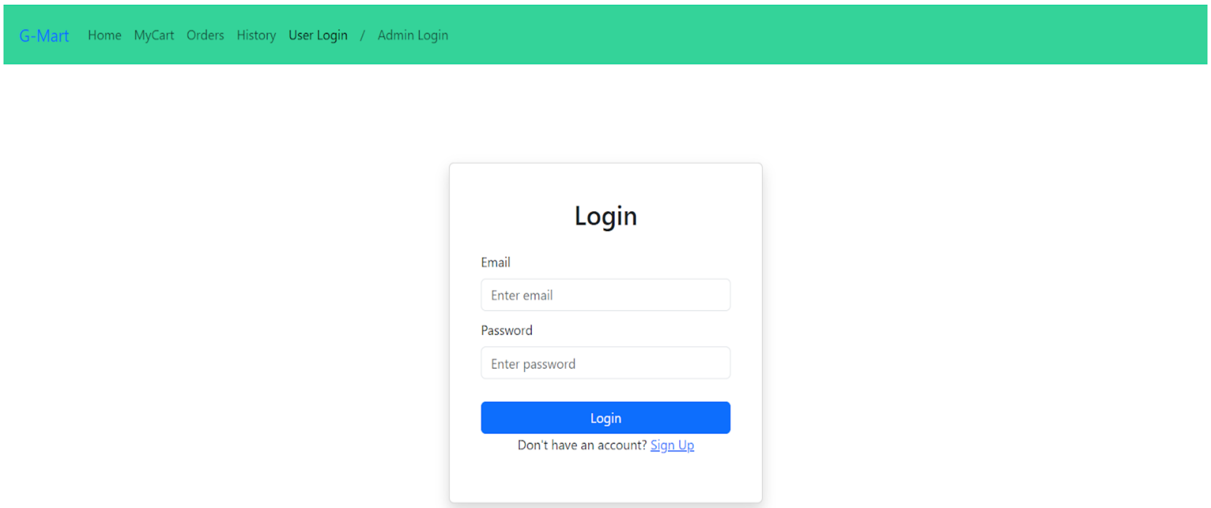
- **Manual Testing:** The application has undergone extensive manual testing across various user flows and roles (guest, regular user, admin) to ensure all core functionalities (registration, login, browsing, cart operations, ordering, admin management) work as expected and that the user interface is responsive and intuitive.
- **Unit Testing (Optional - if implemented):**
  - **Frontend (React):** Jest and React Testing Library for testing individual React components in isolation.
  - **Backend (Node.js/Express):** Mocha, Chai, and Supertest for testing API endpoints and backend logic.
- **Integration Testing (Optional - if implemented):**
  - Focus on testing the interaction between different backend modules (e.g., controller-model interactions) or between the frontend and backend APIs.
- **End-to-End Testing (Optional - if implemented):**
  - Cypress or Playwright to simulate full user journeys through the application from start to finish.

11. Screenshots or Demo

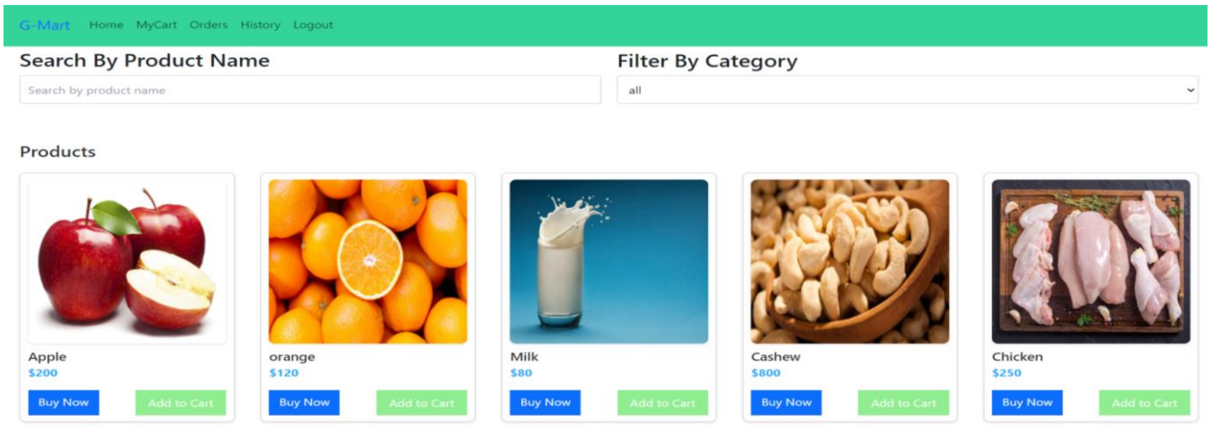
- **Screenshots:**

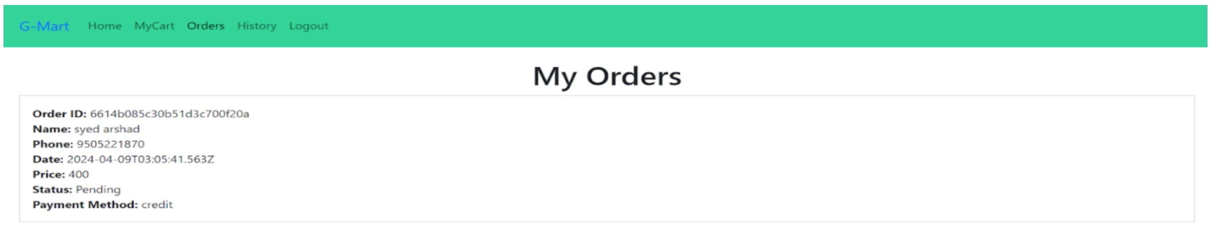

G-Mart
Home
MyCart
Orders
History
User Login
/
Admin Login
Welcome to Our Grocery Web App
Discover a wide range of Grocery's and Fresh Items for all your needs
Shop Now

Login page:



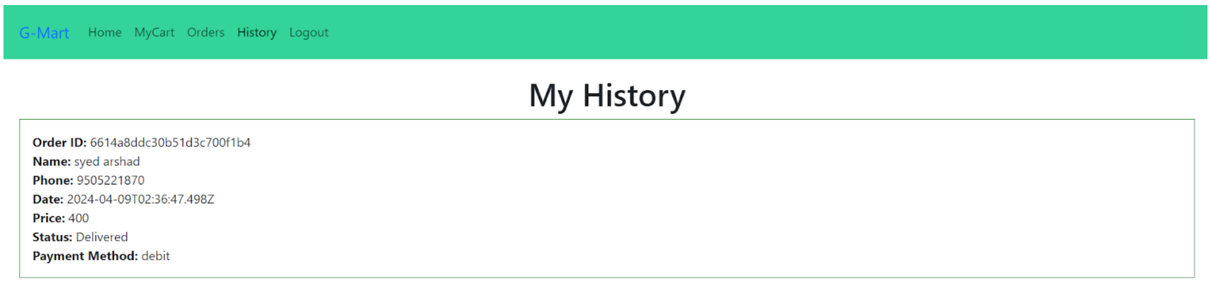
Itempage:





My Orders

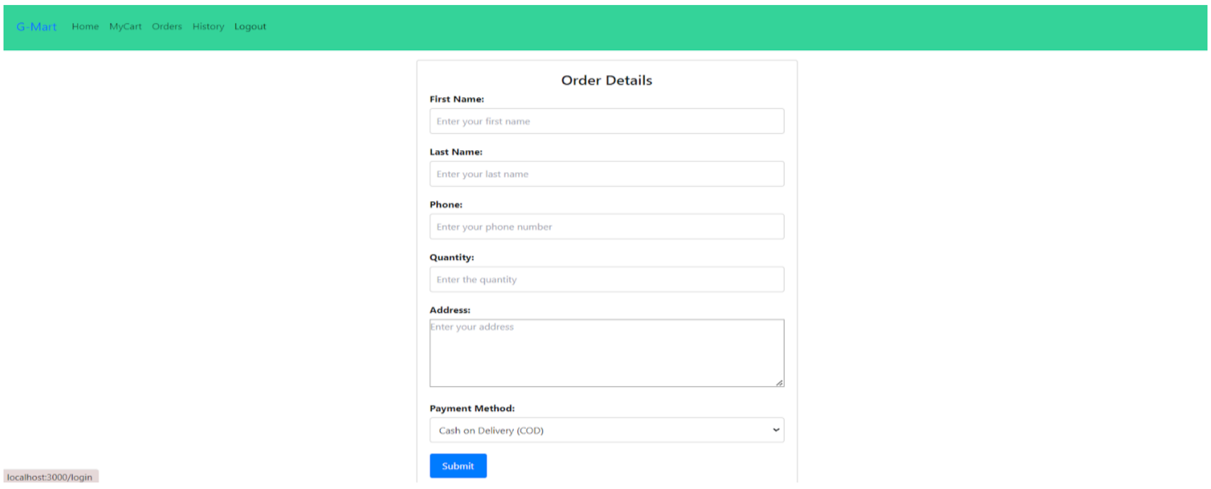
Order ID: 6614b085c30b51d3c700f20a
Name: syed arshad
Phone: 9505221870
Date: 2024-04-09T03:05:41.563Z
Price: 400
Status: Pending
Payment Method: credit



My History

Order ID: 6614a8ddc30b51d3c700f1b4
Name: syed arshad
Phone: 9505221870
Date: 2024-04-09T02:36:47.498Z
Price: 400
Status: Delivered
Payment Method: debit

Placeorderpage:



Order Details

First Name:
Enter your first name

Last Name:
Enter your last name

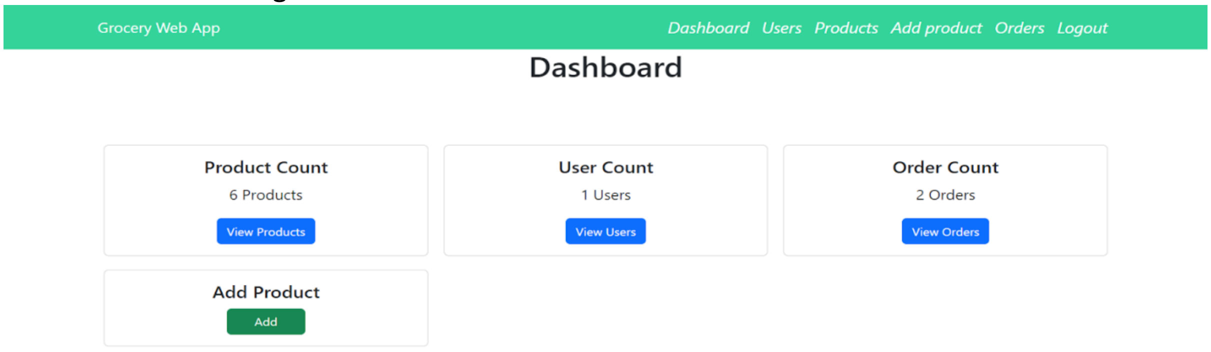
Phone:
Enter your phone number

Quantity:
Enter the quantity

Address:
Enter your address

Payment Method:
Cash on Delivery (COD)

Admin Dashboard Page:



Dashboard

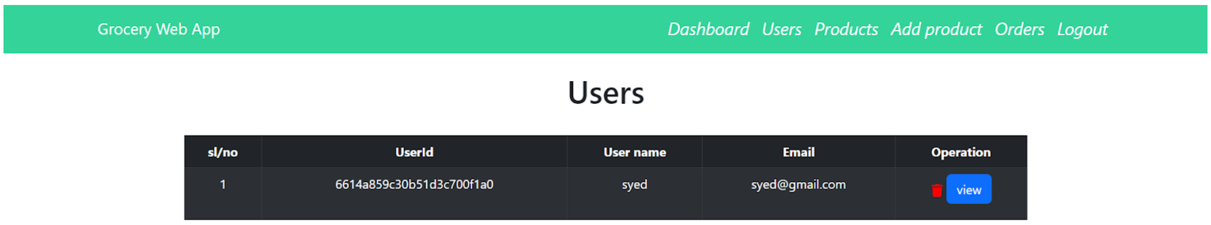
Product Count
6 Products

User Count
1 Users

Order Count
2 Orders

Add Product

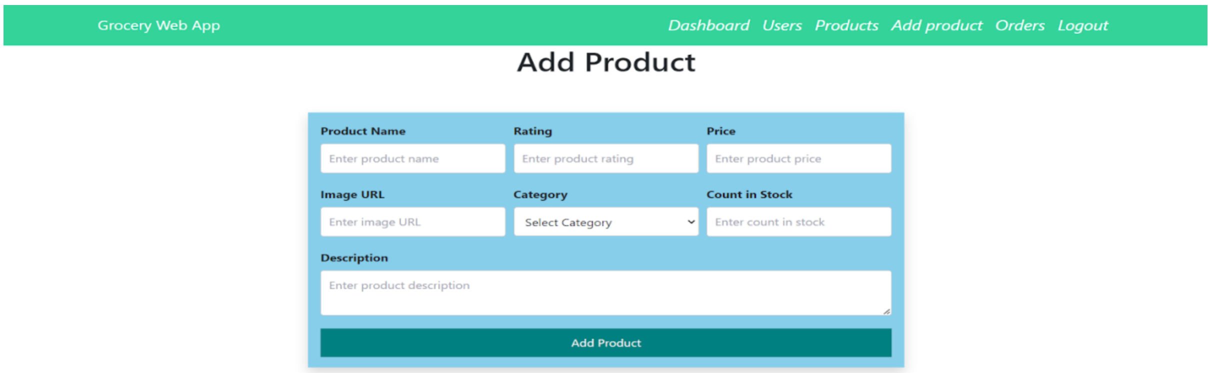
Users Page:-



Users

sl/no	UserId	User name	Email	Operation
1	6614a859c30b51d3c700f1a0	syed	syed@gmail.com	view

Add Product page:-

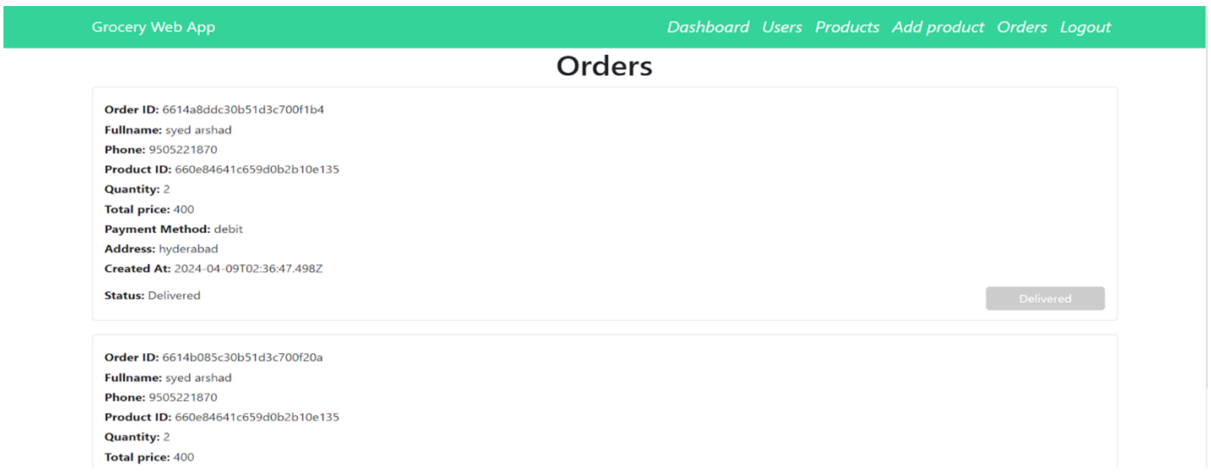


Add Product

Product Name	Rating	Price
Enter product name	Enter product rating	Enter product price
Image URL	**Category**	**Count in Stock**
Enter image URL	Select Category	Enter count in stock

Description: Enter product description

Admin Orders Page:-



Orders

Order ID: 6614a8ddc30b51d3c700f1b4
Fullname: syed arshad
Phone: 9505221870
Product ID: 660e84641c659d0b2b10e135
Quantity: 2
Total price: 400
Payment Method: debit
Address: hyderabad
Created At: 2024-04-09T02:36:47.498Z
Status: Delivered

Order ID: 6614b085c30b51d3c700f20a
Fullname: syed arshad
Phone: 9505221870
Product ID: 660e84641c659d0b2b10e135
Quantity: 2
Total price: 400

The demo of the app is available at:-

Video Demo: A comprehensive video demonstration showcasing the key features and user flows of the ShopSmart application is available here:

https://drive.google.com/drive/folders/1eKSsV003gl8ecX1EVuPD2oW7uVC6ZxAZ?usp=sharing

12. Known Issues

This section lists any identified bugs, limitations, or areas that could cause unexpected behavior.

- **CORS Configuration for Production Deployment:** While a proxy is used for local development, proper CORS headers will need to be explicitly configured on the backend for production environments where the frontend and backend are hosted on different domains.
- **Comprehensive Input Validation:** Basic input validation is in place, but more robust and granular server-side validation for all incoming data (especially for product creation/updates and user inputs) could be added to enhance security and data integrity.
- **Generic Error Messages:** Some error messages returned to the frontend are generic. More specific, user-friendly error messages could be implemented for a better user experience.
- **UI Responsiveness (Minor Adjustments):** While generally responsive, some UI elements or layouts may require minor adjustments for optimal display across all device sizes and screen orientations.
- **Image Handling:** Currently, image URLs are stored. A dedicated image upload service (e.g., Cloudinary, AWS S3) and local file handling should be integrated for more robust image management.
- **Password Reset Functionality:** A formal "Forgot Password" feature with email verification is not yet implemented.
- **Limited Search & Filtering:** Current product search and filtering capabilities are basic. More advanced options (e.g., by price range, brand, multiple categories) are needed.

13. Future Enhancements

The following features and improvements are planned or considered for future development to enhance ShopSmart's functionality, performance, and user experience:

- **Payment Gateway Integration:** Implement a secure third-party payment gateway (e.g., Stripe, PayPal, Razorpay) to enable real-time online transactions.
- **Advanced User Profile Management:** Allow users to update their profile details (name, shipping address, contact number) from the frontend.
- **Product Reviews and Ratings:** Enable users to submit reviews and assign ratings to products, with average rating display.
- **Wishlist Functionality:** Allow users to save products to a personal wishlist for future purchase.
- **Email Notifications:** Integrate an email service for order confirmations, shipping updates, password resets, and promotional communications.
- **Admin Analytics Dashboard:** Develop more comprehensive analytics tools for administrators, including sales reports, popular products, and user engagement metrics.
- **Product Variants:** Support for different product variants (e.g., different sizes or weights for a grocery item) and their respective stock levels.
- **Deployment Automation:** Set up Continuous Integration/Continuous Deployment (CI/CD) pipelines (e.g., GitHub Actions) for automated testing and deployment.
- **Real-time Updates:** Implement WebSocket for real-time updates (e.g., live chat support, order status updates, stock changes).
- **Search Engine Optimization (SEO):** Implement SEO best practices for better visibility in search engines.
- **Accessibility Improvements:** Enhance the application to meet WCAG guidelines for accessibility.
- **Customer Support Chat:** Integrate a live chat feature for immediate customer assistance.