**Full Stack Development with MERN**

**Project Documentation format**

* **Project Title:**

### GROCERY WEB APP

* **Team Members:**

Ramya S (412721205037)

NM ID : EC91C4C7EF65E51B2054E75850D11274

Abinaya P (412721205001)

NM ID : E316A5186D8FB999DE348F29678CB1F8

Indhumathi S (412721205017)

NM ID : 15C56AAF3481688809948309AB3B274E

Charumathi B (412721205006)

NM ID : 66E34BE425337A3BFA7419B7951FA0A8

DurgaDevi E (412721205010)

NM ID : 561A124CBCB1E2FA1C04D90000D244AC

**1. Project Overview**

**Purpose:**

The primary purpose of a grocery web application is to provide a convenient and user-friendly platform that enables customers to browse, select, and purchase groceries online. By offering features such as product search, detailed item descriptions, a shopping cart, secure payment processing, and delivery scheduling, the app aims to enhance the overall shopping experience and streamline the process for users.

**Goals:**

The overarching goal of the application is to make grocery shopping more accessible and efficient, allowing customers to shop from the comfort of their homes, saving time while enjoying a seamless and personalized experience. Additionally, the platform supports businesses in reaching a broader customer base and improving operational efficiency, particularly in inventory management.

**Features:**

A typical grocery web app incorporates several key features to optimize the user experience and support business operations. These include:

* Product Catalog: A comprehensive catalog with robust search and filtering capabilities, enabling users to easily find products.
* User Authentication: Secure login and account management, providing customers with personalized experiences and enabling order tracking.
* Shopping Cart & Checkout: An intuitive shopping cart and streamlined checkout process for efficient order placement.
* Order Management: A system for customers to track their purchases and manage order history.
* Real-Time Inventory Updates: Features that allow users to view live inventory status, ensuring accurate availability information.
* Delivery Scheduling: Options for customers to select preferred delivery time slots and manage delivery addresses.
* Customer Reviews & Ratings: A feature enabling users to share their experiences and evaluate products.
* Promotions & Discounts: Special offers, coupons, and discounts to encourage purchases and customer loyalty.
* Admin Panel: A back-end system for businesses to efficiently manage inventory, process orders, and monitor performance.
* Advanced Search Functionality: Enhanced search options that improve product discovery and allow for a more tailored shopping experience.

**2. Architecture Diagram and Workflow**

**Frontend Architecture:**

The frontend architecture of a grocery web application, developed using React, follows a component-based structure, ensuring modularity, scalability, and maintainability. Key components include:

* Header: Displays navigation and user-related information.
* ProductList: Displays the list of available products, often with search and filtering options.
* ProductCard: Represents individual product details, including pricing, description, and an "Add to Cart" option.
* Shopping Cart: A component that displays selected items and enables cart modifications.
* Checkout: Facilitates the final order process, including user authentication, payment, and delivery details.

React Router is used for managing page navigation, while global state management is handled through either Redux or Context API, which tracks global states like the shopping cart and user authentication status. For data fetching, the application uses Axios or the Fetch API to retrieve data from the backend.

The app’s user interface is styled using frameworks like Material-UI or Tailwind CSS for responsive design, ensuring compatibility across various screen sizes. Formik or React Hook Form is employed to manage forms, particularly for checkout and user authentication, improving form validation and handling.

This frontend architecture ensures a seamless and responsive user experience, making the app both scalable and easy to maintain.

**Backend Architecture:**

The backend of the grocery web application is built using Node.js with Express.js for routing and implementing a RESTful API that facilitates communication between the frontend and backend. This API manages crucial aspects such as user authentication, product management, cart functionality, and order processing.

* JWT (JSON Web Tokens) is used for secure user authentication and authorization.
* The data layer relies on either PostgreSQL or MongoDB for storing data. For MongoDB, Mongoose is employed as the Object-Relational Mapping (ORM) tool, while Sequelize is used for PostgreSQL. These ORMs simplify database interactions by providing an abstraction layer for CRUD operations.
* Middleware functions are incorporated for error handling, logging, and validation to ensure smooth operation and data integrity.
* Third-party payment integrations such as Stripe or PayPal handle secure payment processing, allowing for reliable transaction management.

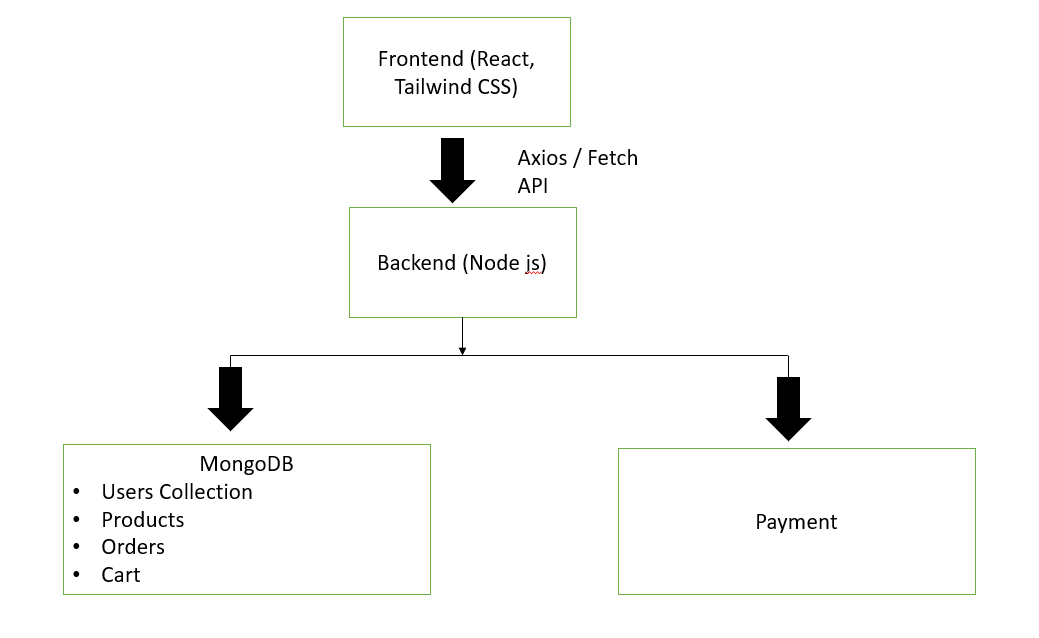
This backend setup ensures scalability, robustness, and security while supporting a seamless user experience and efficient business operations.

**Database Architecture:**

In a grocery web application using MongoDB, the database schema typically consists of several collections to manage user data, product listings, orders, and shopping carts. The primary collections include:

* Users Collection: Stores user information, including:
  + user\_id, name, email, password (hashed), order\_history.
* Products Collection: Holds product details, such as:
  + product\_id, name, description, price, stock\_quantity, category.
* Orders Collection: Tracks order details for each user, including:
  + order\_id, user\_id, product\_ids, quantity, total\_price, shipping\_address, order\_status.
* Cart Collection: Stores products added to the shopping cart before checkout, including:
  + cart\_id, user\_id, product\_ids, quantities**.**

**ARCHITECTURE DIAGRAM**

****

**3. Setup Instructions**

* **Prerequisites:** To build a grocery web app, you'll need proficiency in ‘React’ for the frontend, along with ‘HTML’, ‘CSS’, and ‘JavaScript’, and experience with ‘React Router’ for navigation and ‘Redux’ or ‘Context API’ for state management. On the backend, knowledge of ‘Node.js’ and ‘Express.js’ is essential for creating RESTful APIs, along with ‘JWT’ for user authentication. Familiarity with ‘MongoDB’ and ‘Mongoose’ is required for database management and CRUD operations. You'll also need experience with ‘Git’ for version control, and the ability to integrate payment systems like ‘Stripe’ or ‘PayPal’. Finally, understanding ‘Docker’ for containerization and deployment tools like ‘Heroku’ or ‘AWS’ is crucial for hosting the app. These skills together will enable you to build, manage, and deploy a fully functional grocery web app.
* **Installation:**

To set up the grocery web app:

**1.** **Clone the Repo:**

Clone the project from GitHub:

bash

git clone https://github.com/your-username/grocery-web-app.git

cd grocery-web-app

**2.** **Backend Setup:**

- Navigate to the backend folder **(cd backend)**, install dependencies with:

bash

npm install

- Create a **.env file** and add environment variables like MongoDB URI, JWT secret, and payment keys:

MONGO\_URI=mongodb://localhost:27017/grocerydb

JWT\_SECRET=your\_jwt\_secret

STRIPE\_SECRET\_KEY=your\_stripe\_secret\_key

PORT=5000

**- Start the backend server:**

bash

npm start

**3. Frontend Setup:**

- Navigate to the frontend folder (cd frontend), install dependencies:

bash

npm install

- Create a **.env file** and add variables like API URL:

REACT\_APP\_API\_URL=http://localhost:5000/api

REACT\_APP\_STRIPE\_KEY=your\_stripe\_publishable\_key

**- Start the React development server:**

bash

npm start

**4. Verify:**

- Backend should be running at http://localhost:5000, and frontend at <http://localhost:3000>.

This sets up the app locally with both backend and frontend running.

**4. Folder Structure**

* **Client:** The React frontend of a grocery web app follows a modular structure with key directories like **/public** for static files and **/src** for the app code. Core components reside in **/components** and **/pages**, with **/services** for API calls and **/redux (or /state)** for state management. Routing is handled with React Router, and global styles are applied via CSS or styled-components. The app is structured for scalability and maintainability, supporting features like product displays, a shopping cart, and more advanced functionalities.
* **Server:** The Node.js backend for a grocery web app is organized into several key directories for modularity and maintainability. **/config** handles environment settings, while **/controllers** processes requests related to products, orders, and users. **/models** defines the database structure, and **/routes** maps HTTP requests to the appropriate controller actions. **/services** contains business logic like payment processing, and **/middleware** is used for tasks like authentication and validation. **/utils** provides helper functions. The app is initialized in “app.js”, where Express is configured, and the database is connected, ensuring a clean, scalable architecture.

**5. 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.

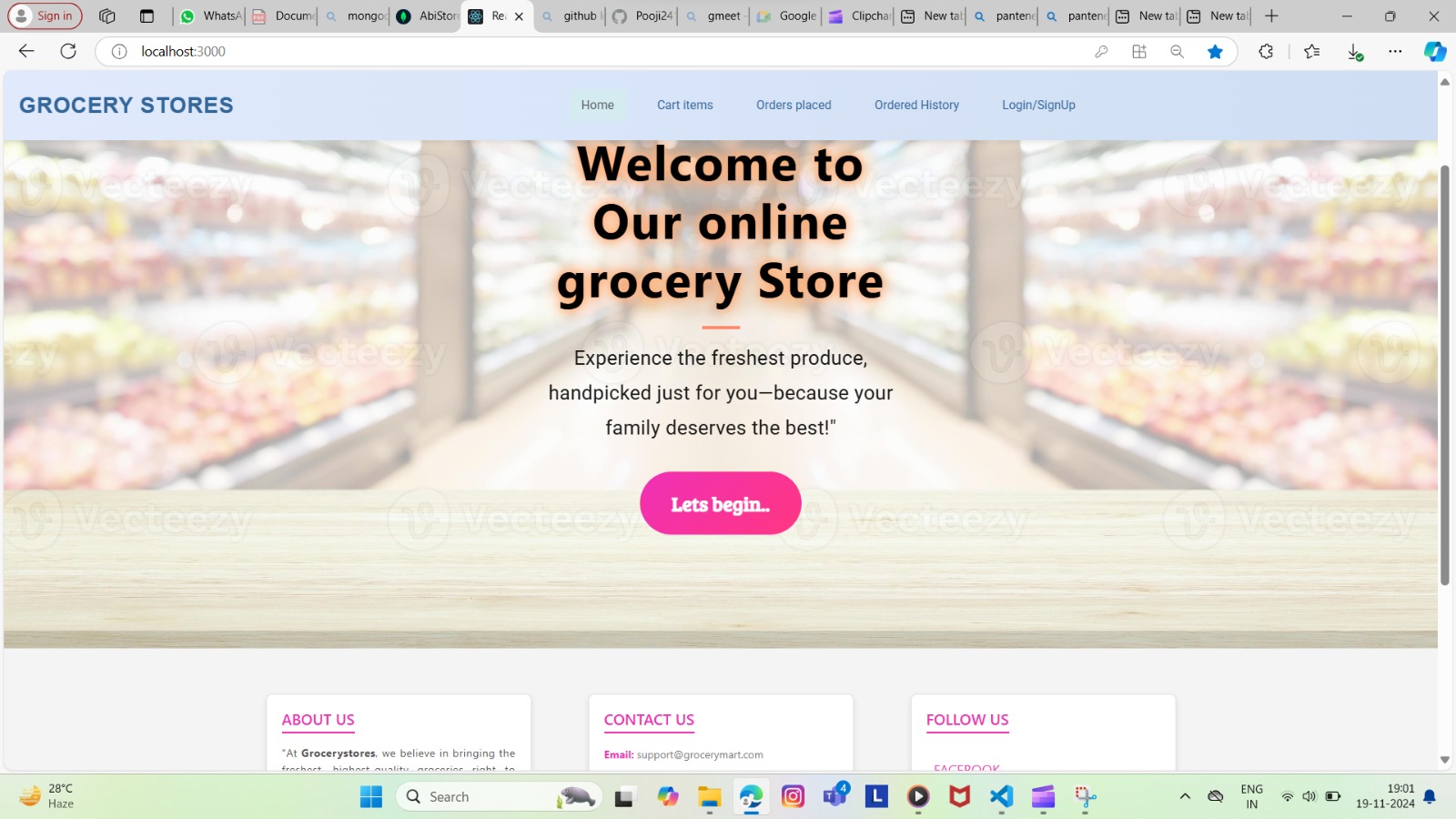
**6. API Documentation**

* The grocery web app backend exposes several API endpoints for managing user authentication, products, cart, orders, and payments. Key endpoints include user registration and login **(`/api/auth/register`, `/api/auth/login`**), profile retrieval **(`/api/users/{userId}`**), and cart management **(`/api/cart`)**, allowing users to add, update, and remove products. Product-related endpoints **(`/api/products`,** **`/api/products/{productId}`)** provide access to product listings and details. For orders, users can create, retrieve, and cancel orders through endpoints like **`/api/orders` and `/api/orders/{orderId}`**. Payment processing is handled by **`/api/payment/checkout**`, supporting various methods like credit cards. These endpoints are essential for managing user sessions, browsing products, and processing transactions, ensuring a seamless shopping experience.
* The grocery web app backend provides API endpoints for user authentication, product browsing, cart management, order processing, and payments. Key endpoints include user registration and login, cart operations (add, update, view), product retrieval, and order creation and cancellation. Payment processing is handled through a checkout endpoint. Each endpoint supports specific methods and parameters, returning responses in JSON format to ensure smooth user interactions, such as adding items to the cart or creating an order with status and payment details. These endpoints enable seamless shopping and order management.

**7. Authentication**

* In the grocery web app, ‘authentication’ is handled using JSON Web Tokens (JWT). Users sign up with a hashed password and log in to receive a JWT, which is stored in the client’s browser (e.g., in localStorage or cookies). This token is used to authenticate requests to protected routes, verified via middleware. ‘Authorization’ is managed with role-based access control (RBAC), where users are assigned roles like "user" or "admin" to restrict access to specific actions. Middleware checks the user's role before granting access to certain routes, such as product management for admins. Optionally, ‘refresh tokens’ can be used to extend sessions by allowing the user to obtain new access tokens when the original expires. The app can also implement password reset and email verification features for added security.
* In the grocery web app, ‘authentication’ is handled using ‘JWTs (JSON Web Tokens)’, which are issued upon login and stored in ‘localStorage’ or ‘HttpOnly cookie’. These tokens contain user data and an expiration time, and are used to authenticate API requests. ‘Refresh tokens’ are employed to extend user sessions, allowing the client to request new access tokens without re-login when the current token expires. ‘Role-based access control (RBAC)’ ensures users can only access specific resources based on their roles (e.g., user or admin). Middleware functions validate tokens and enforce role-based restrictions on protected routes. Additionally, the app supports ‘password reset’ and ‘email verification’ processes for enhanced security. The system is designed to securely manage user sessions, roles, and authentication using JWTs, refresh tokens, and cookies.

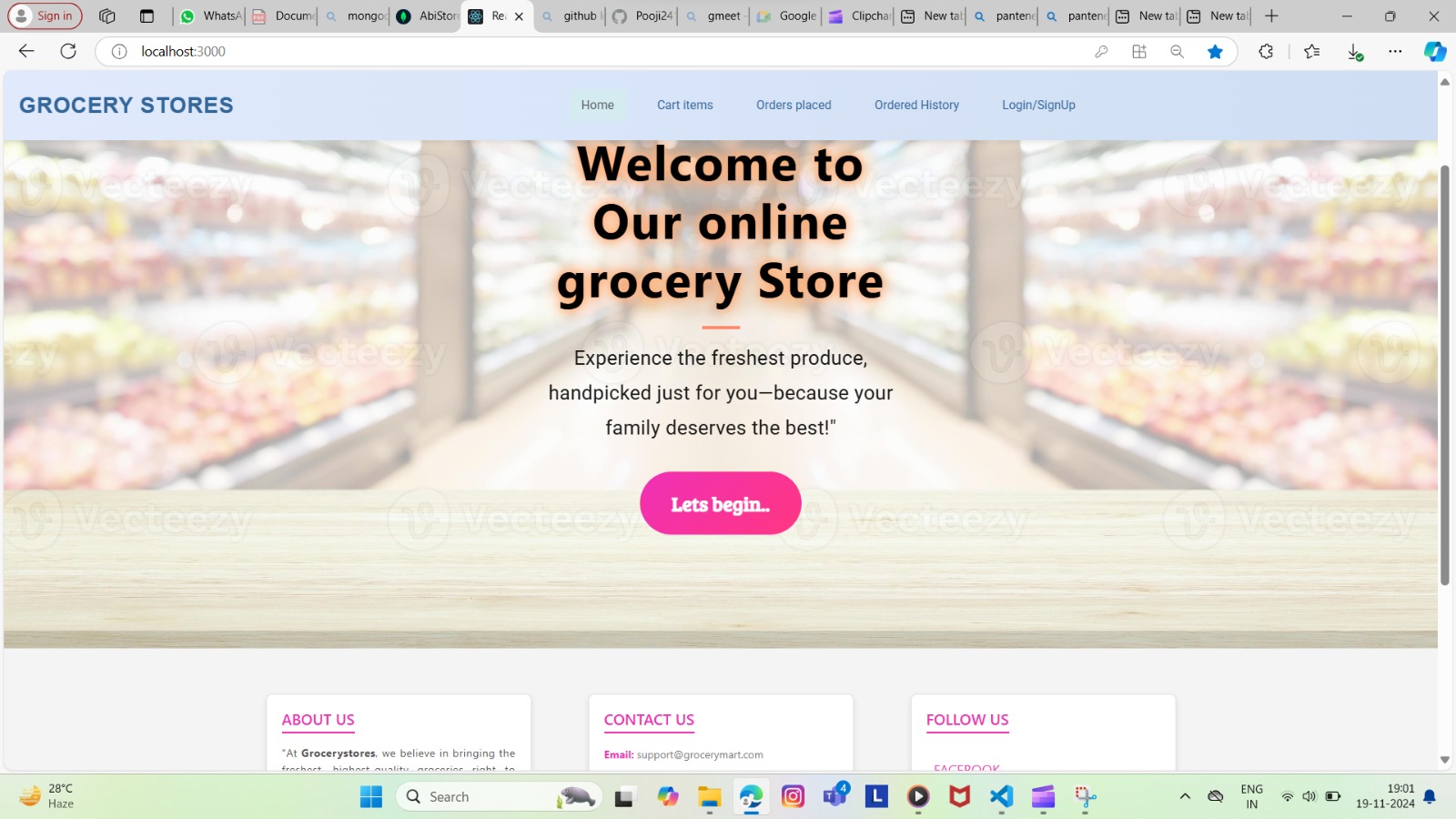
**8. User Interface**

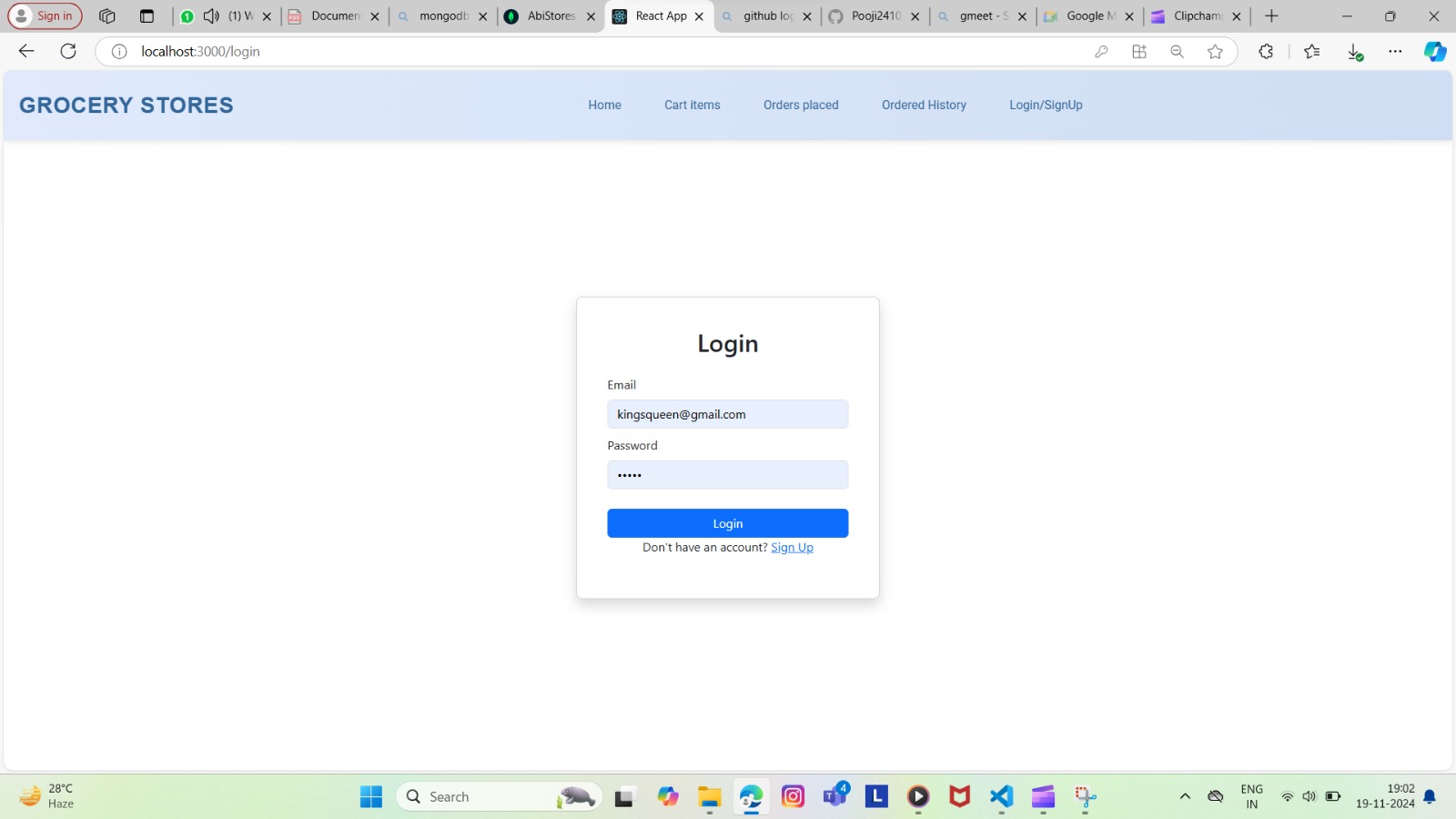
****

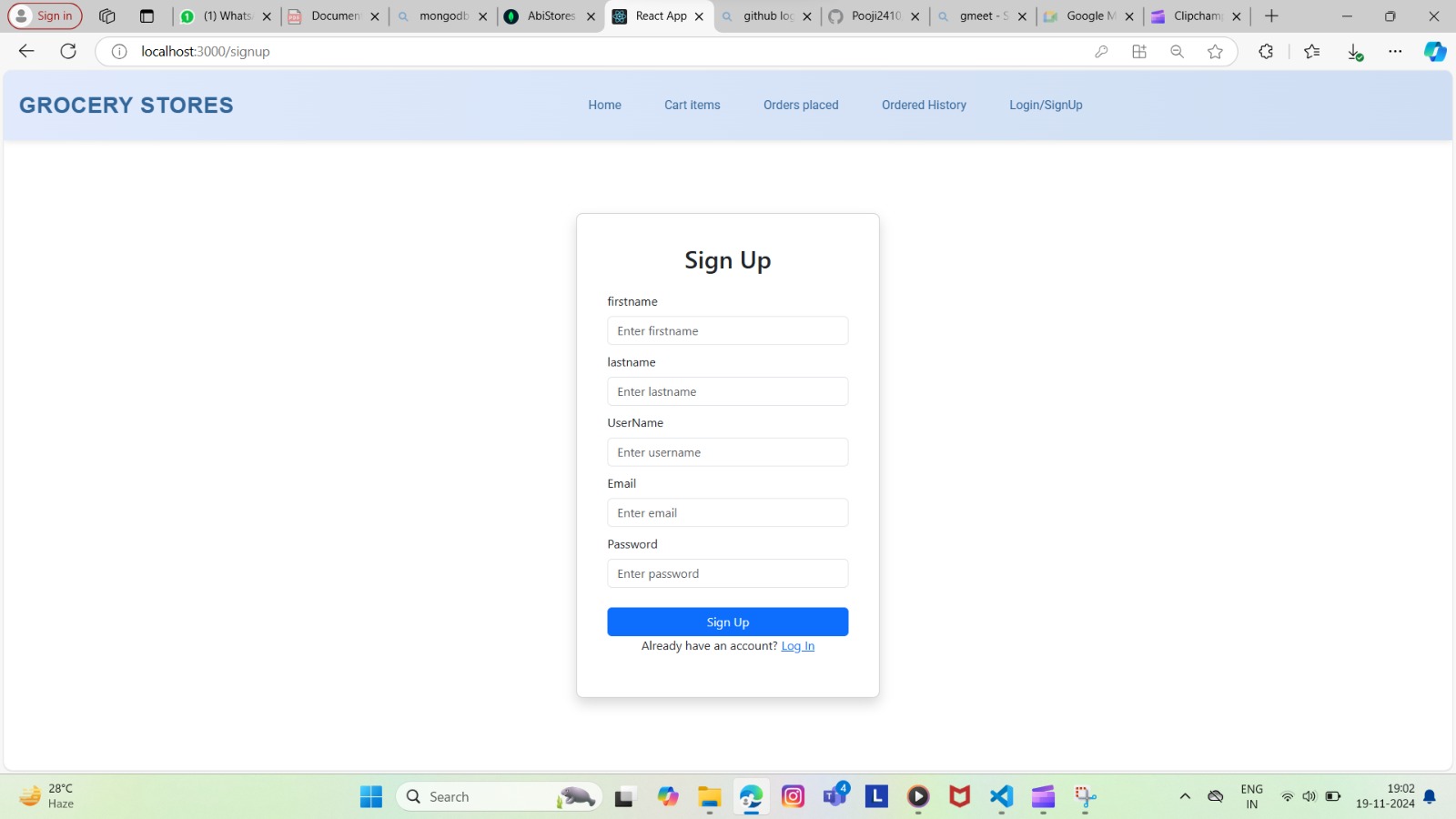
**8. Testing**

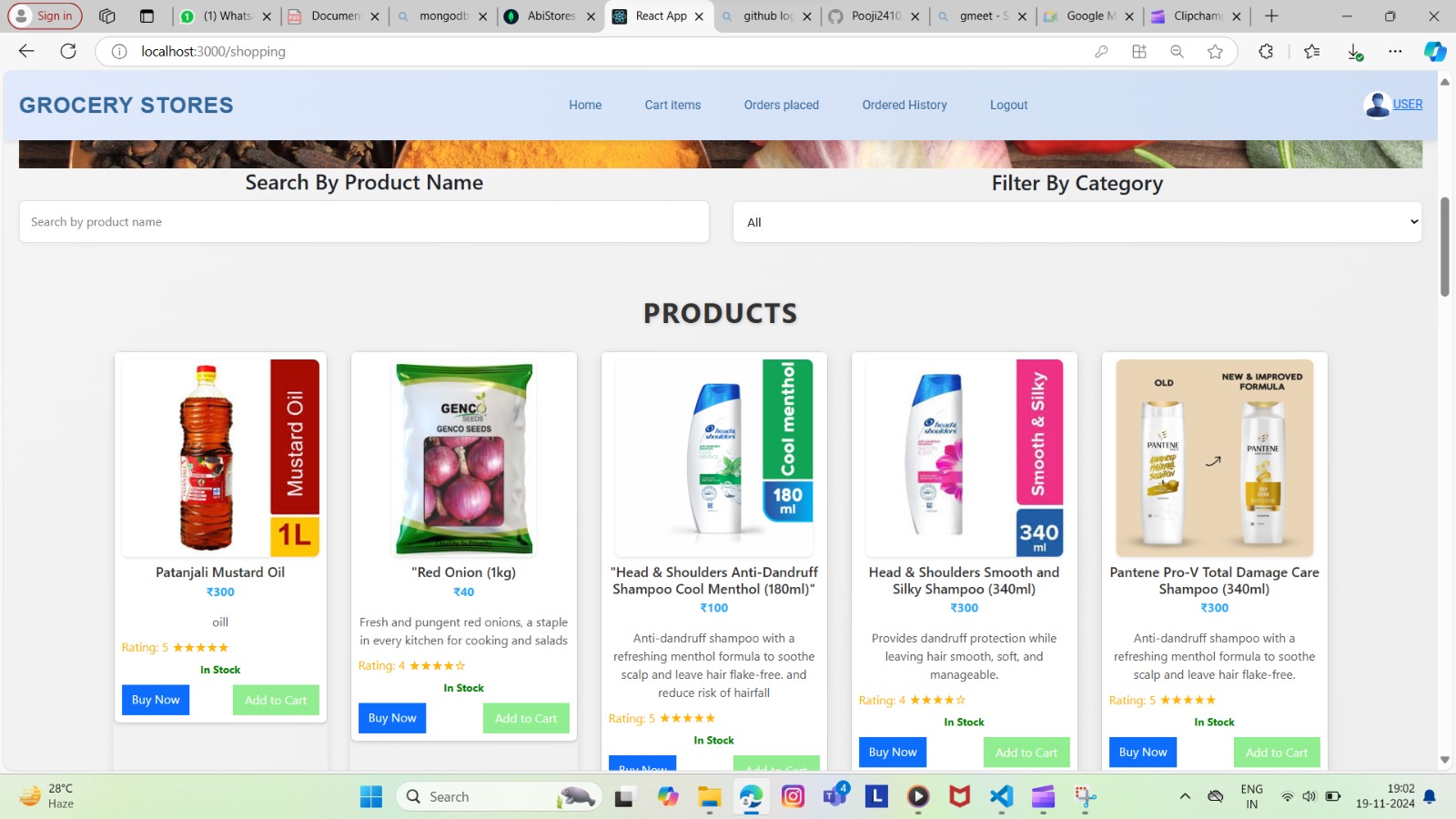
* The testing strategy for a grocery web app involves a multi-faceted approach to ensure functionality, performance, security, and user experience. Key testing types include **unit testing** to validate individual components, **integration testing** for checking interactions between modules, and **end-to-end (E2E) testing** to simulate real user journeys. **Performance testing** ensures the app can handle high traffic, while **security testing** safeguards against threats like SQL injection and XSS. **Usability** and **accessibility testing** focus on ensuring the app is intuitive and accessible to all users. Automation is crucial for efficient regression and **functional testing**, especially for repetitive tasks and cross-browser validation. Tools like **Cypress**, **Selenium**, and **Jest** are used for automated testing, while **Postman** and **JMeter** are used for API and load testing. The testing process is integrated into a CI/CD pipeline to ensure continuous validation. Manual testing is also performed for exploratory and user acceptance testing, ensuring all requirements are met before deployment.

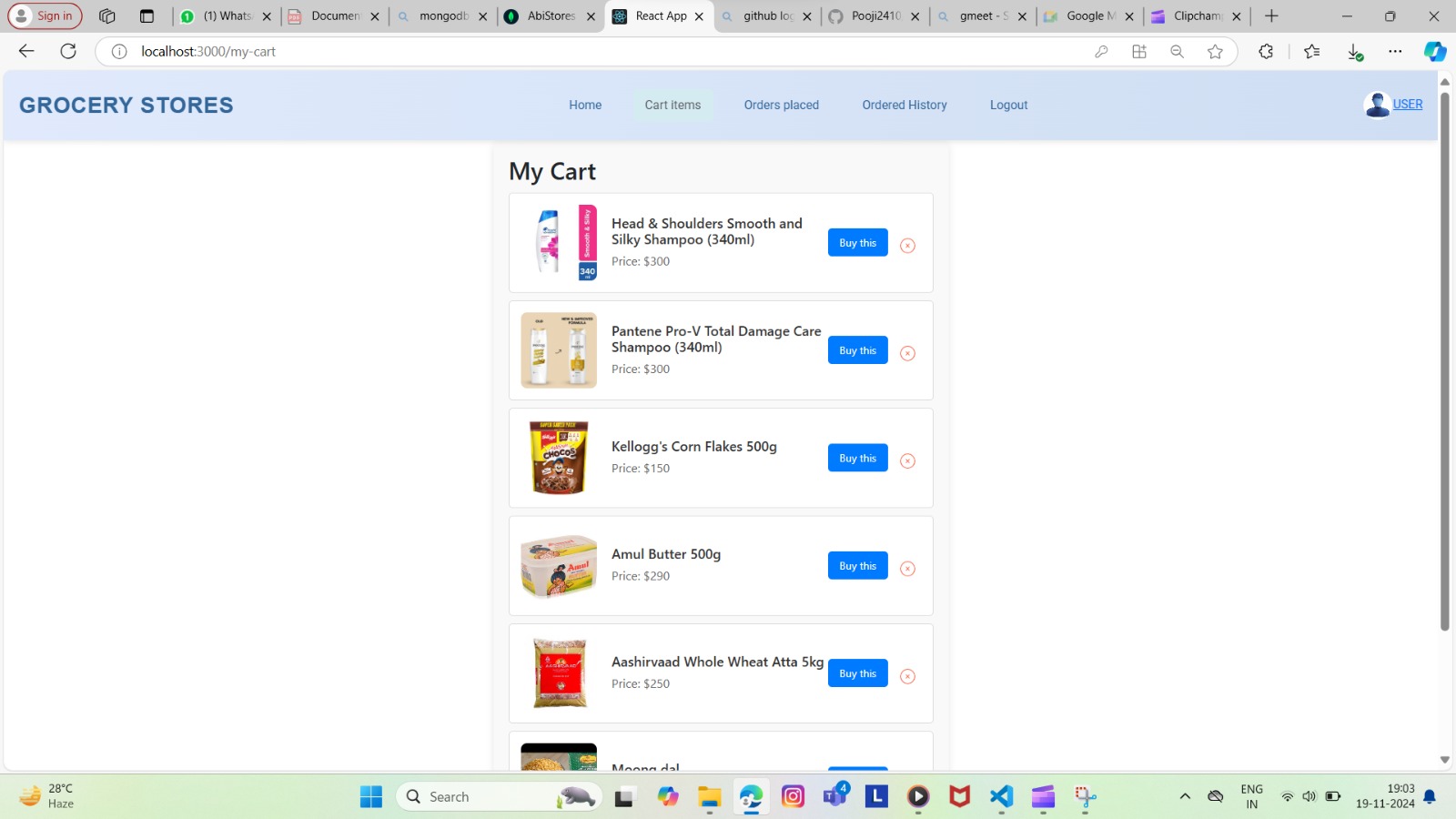
**9. Screenshots**

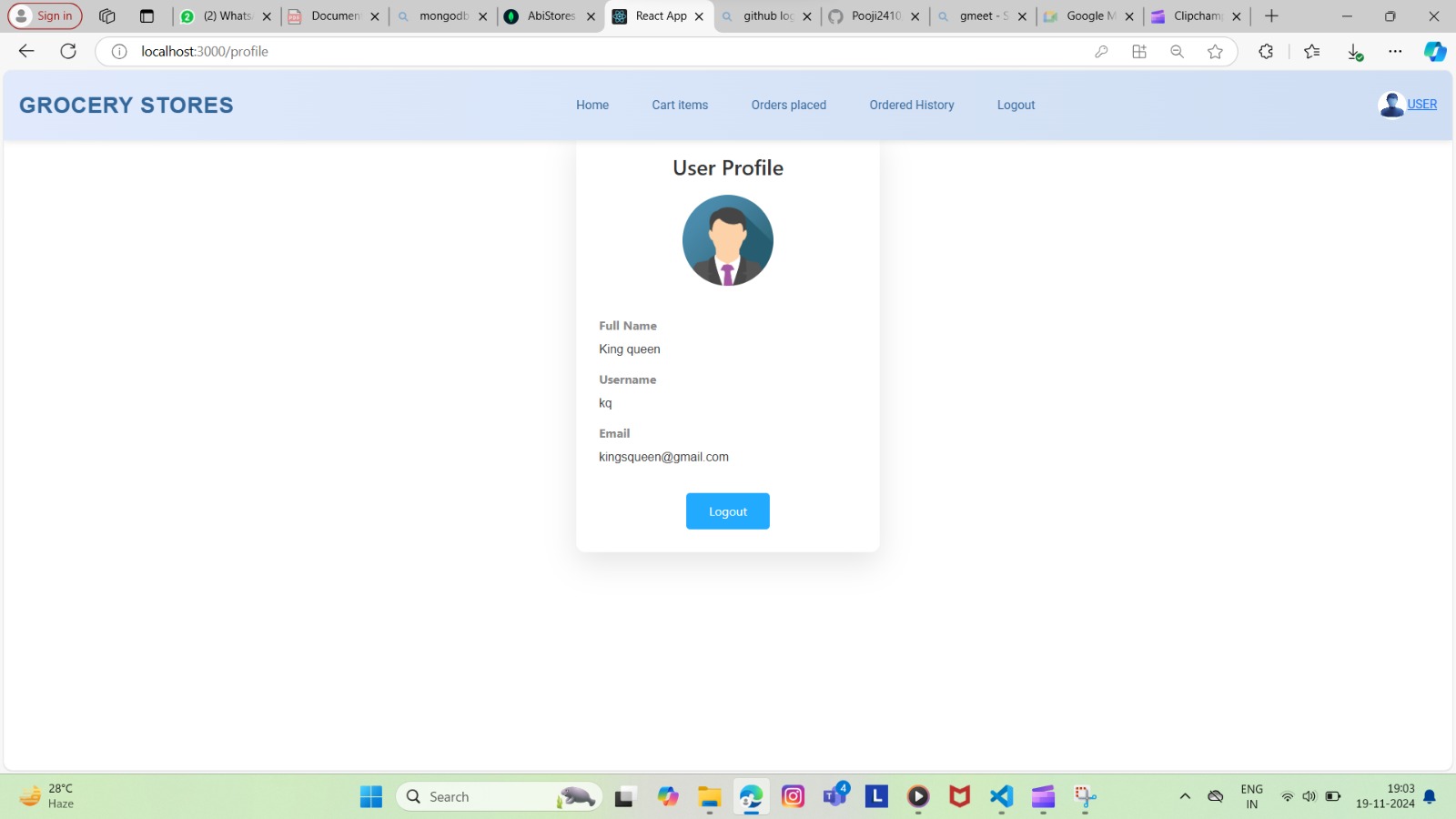
****

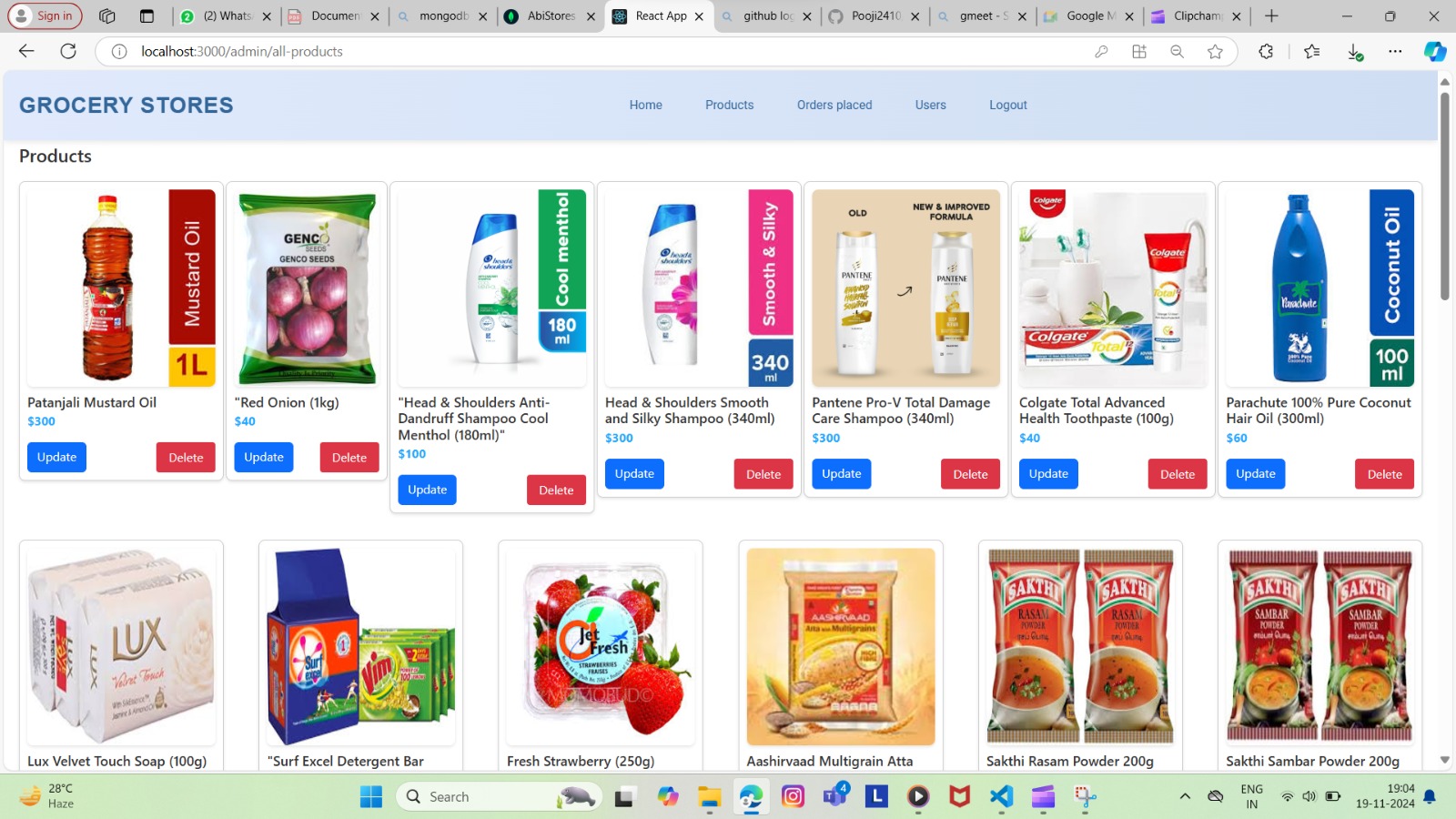
****

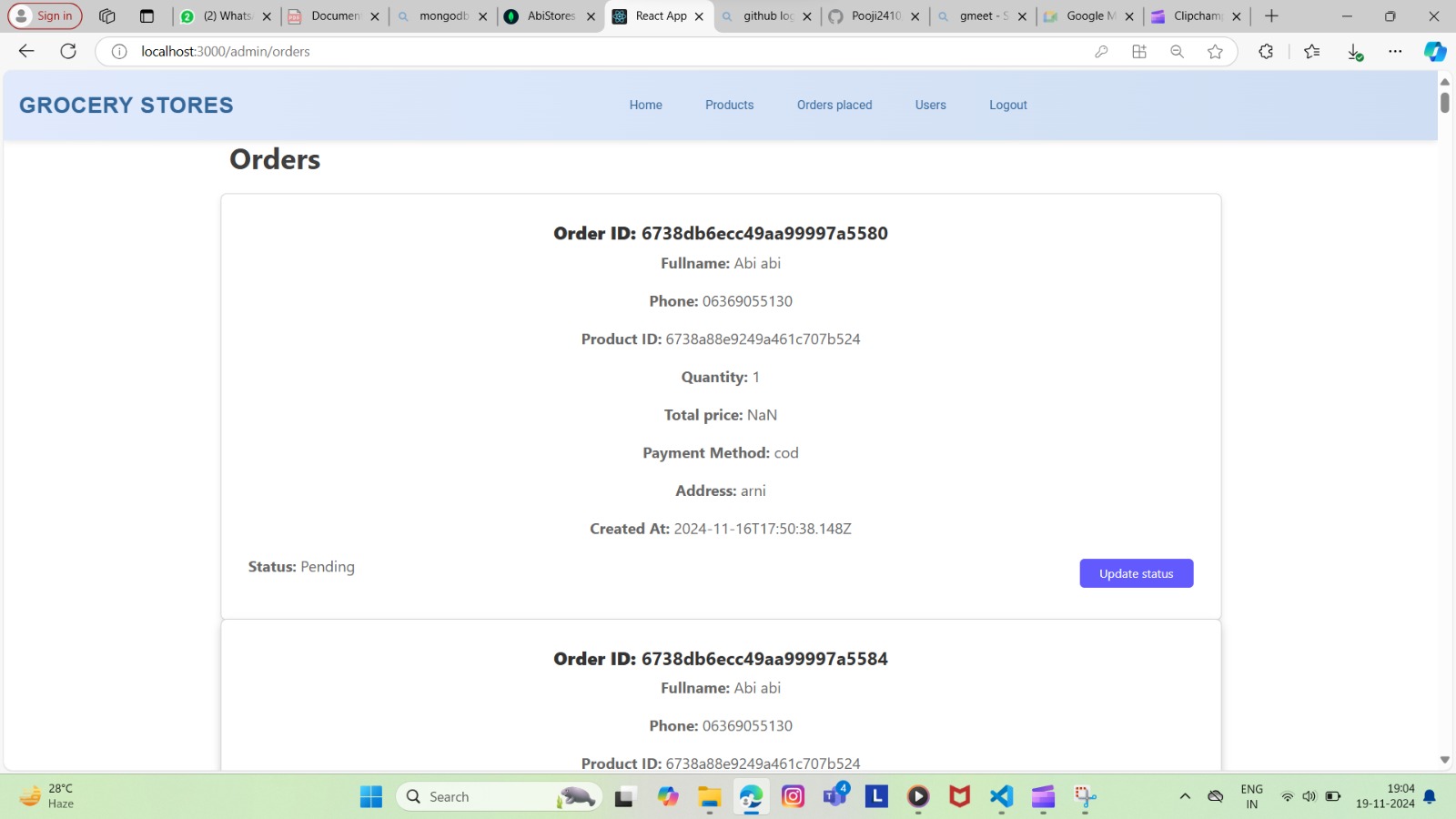
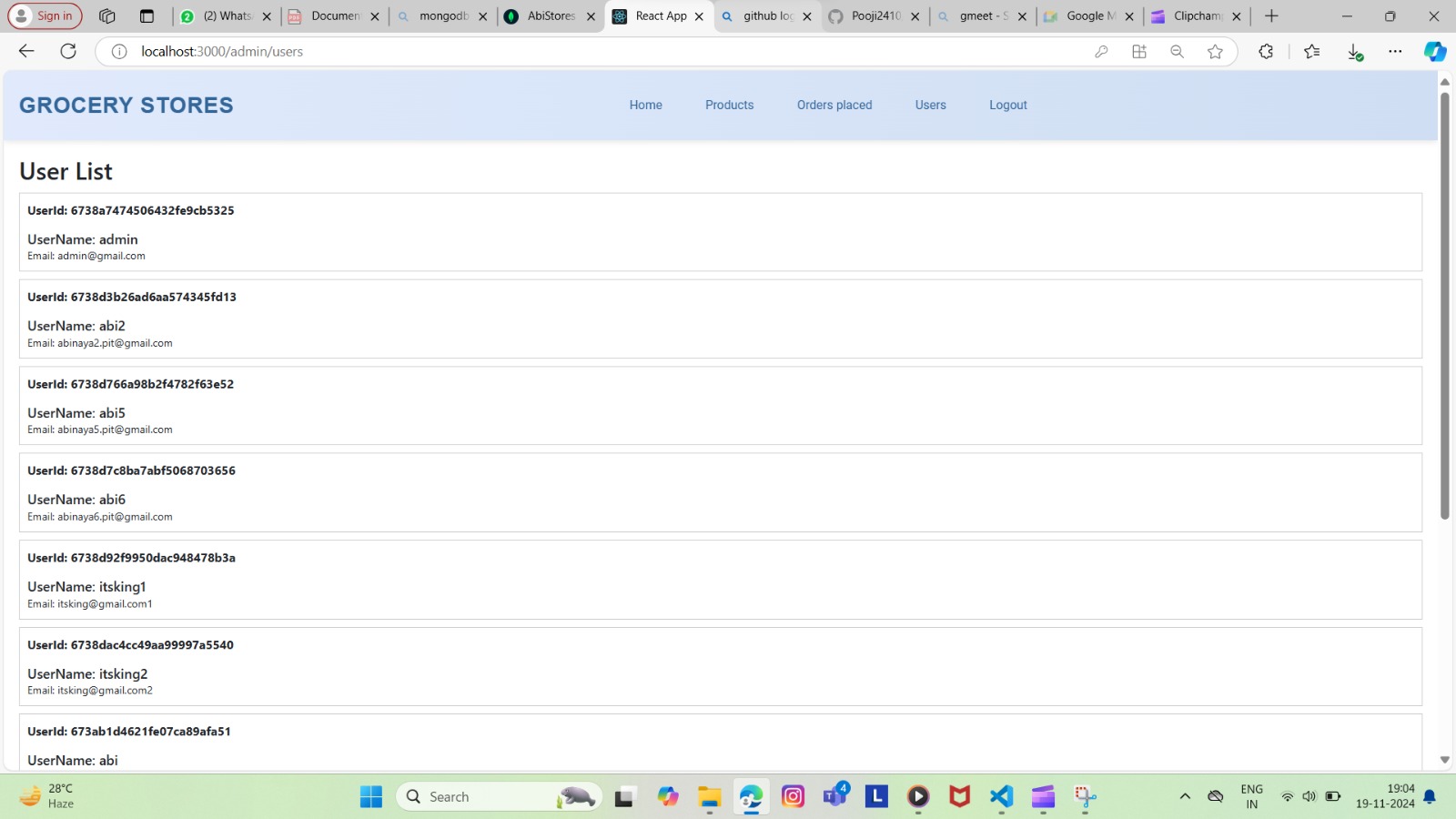
****

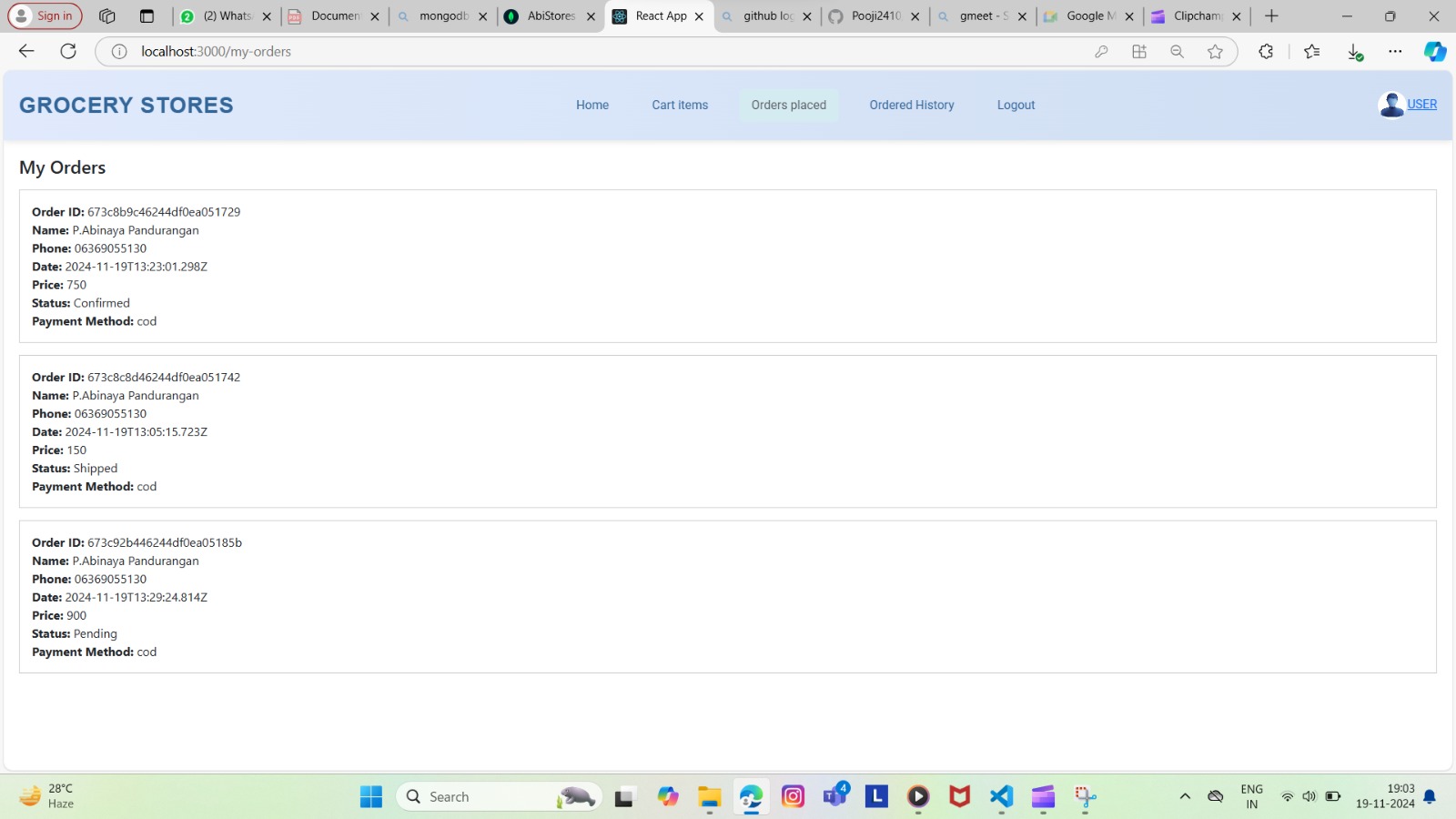
****

****

****

****

****

****

**12. Known Issues**

* There are several known issues with the grocery web app that users and developers should be aware of. Users may experience intermittent login failures due to session expiration, incorrect cart totals when applying discounts, and misaligned product images on mobile devices. The search function occasionally returns irrelevant results, and the checkout process sometimes times out during payment. PayPal integration issues have led to failed transactions, and product availability statuses may not update in real time. Shipping costs can be inaccurate for international orders, and product images may not refresh properly due to caching problems. Additionally, some accessibility issues persist with screen reader compatibility. Developers are actively working on fixes, and users are advised to follow suggested workarounds, such as clearing their cache or using alternative payment methods. Users are also encouraged to use modern browsers for better performance, and scheduled maintenance may temporarily affect the payment gateway.

**13. Future Enhancements**

* Future improvements for the grocery web app could include AI-powered search, personalized dashboards, real-time inventory updates, and mobile optimization. Enhancements like multi-currency and language support, subscription services, faster checkout, and eco-friendly product options would boost user satisfaction. Additional features like voice shopping, AI chatbots, advanced analytics, and more payment methods (e.g., buy-now-pay-later) would improve both customer experience and operational efficiency. Real-time delivery tracking, social media sharing, and scheduling options would further enhance convenience and engagement, helping drive sales and expand the app's reach.