***Full Stack Development With MERN***

***...............................................................***

**INTRODUCTION**

**----------------------------**

**Project Title**: Order On The Go: On-Dem and Food Ordering Solution

**Team Members:**

1. Chamarthy Venkata Swathi - 248A1F0065

2 . Baddepudi Venkata Sai Lakshmi - 248A1F0061

3 . Jagarlamudi Anusri - 248A1F0073

4. Shaik Afrin - 248A1F0087

**Project Overview**

**------------------------**

**Purpose:** **"Order On The Go"** is an innovative, on-demand food ordering and delivery platform designed to connect customers with nearby restaurants and food outlets in real time. The solution allows users to browse menus, place orders, make payments, and track deliveries seamlessly through a mobile app or website.

The platform provides a centralized system for customers, restaurants, and delivery partners, enhancing the food ordering experience through speed, convenience, and transparency.

**Features:**

**For Customers:**

* User-friendly mobile app and website
* Search and browse nearby restaurants
* Customize and place orders
* Multiple payment options (card, wallet, UPI, etc.)
* Real-time order and delivery tracking
* Ratings and reviews

**For Restaurants:**

* Dashboard to manage menus, prices, and offers
* Real-time order notifications
* Order history and analytics
* Control over availability and delivery zones

**For Delivery Agents:**

* Delivery app with order assignment
* GPS navigation to restaurants and customer locations
* Order status updates
* Delivery performance tracking

**For Admins:**

* Centralized control panel
* User, restaurant, and delivery partner management
* Commission and payment management
* Reporting and analytics dash board

**Business Impact:**

“Order On The Go” aims to modernize the food ordering experience, boost local restaurant visibility, and create job opportunities for delivery agents, while offering customers a faster and more reliable way to enjoy their favourite meals.

**Scenario:** Using "Order On The Go" – A Real-Life Use Case

1. **Customer Journey**

Sarah is working late in the office and feels hungry. She doesn’t want to cook or drive to a restaurant. She opens the **"Order On The Go"** mobile app on her phone.

* She browses nearby restaurants offering dinner.
* She filters by cuisine (Italian) and sees a list of available options.
* She selects a restaurant, views the menu, customizes her pasta order, and adds a soft drink.
* She selects "Home Delivery," adds her address, and pays using her credit card through the app.
* The app confirms the order and shows a real-time tracking map.

1. **Restaurant Experience**

Luigi's Italian Kitchen, the restaurant Sarah ordered from, receives the order via their dashboard.

* A notification pops up with all order details.
* The kitchen starts preparing the meal.
* Once ready, the restaurant marks the order as "Ready for Pickup."

1. **Delivery Agent Workflow**

Mike, a delivery agent nearby, gets a notification on the delivery app.

* He accepts the order and navigates to Luigi’s using in-app GPS.
* He picks up the order, updates the status to "Out for Delivery."
* Sarah watches the delivery progress in real-time.

1. **Order Completion**

* Mike delivers the food to Sarah’s office within 30 minutes.
* She receives a notification and rates both the restaurant and delivery service.
* The app logs the transaction and sends a digital receipt.

This scenario demonstrates how "Order On The Go" delivers a smooth, real-time food ordering and delivery experience by connecting customers, restaurants, and delivery partners in a single, integrated digital ecosystem.

**Architecture**

**------------------**

**Frontend:** The frontend is represented by the "Frontend" section, including user interface components such as User Authentication, Cart, Food Items, Profile, Admin dashboard, etc.,

**Components:** Break down the UI into reusable components (e.g. Food Item list, cart, login form , button).

**State Management:** Use a state management library like Redux or React Context API or Zustand (e.g. user authentication, cart contents)

**Routing:** It use React Router for client-side routing (e.g. navigating between item pages, cart, login).

**API Interactions:** Use Axios or Fetch API to make requests to the backend for data (e.g. fetching food items, posting orders).

|  |  |
| --- | --- |
| **API Calls** | : It uses Axios or Fetch API |

|  |  |
| --- | --- |
| **Forms** | : React Hook Form or Formik + Yup |

|  |  |
| --- | --- |
| **Authentication** | : It uses JWT + Local Storage or Context API |

|  |  |
| --- | --- |
| **Real-time Updates** | : It uses Web Sockets or Firebase or Socket.IO |

|  |  |
| --- | --- |
| **Map Integration** | : It uses Google Maps JavaScript API |

|  |  |
| --- | --- |
| **Notifications** | : It uses Toastify or Notistack or OneSignal |

**Backend:** The backend is built on a RESTful API architecture using Node.js and Express.js. It acts as the central engine powering all business logic, API routes, data operations, authentication, real-time tracking, and communication between the frontend and the database.

**API Endpoints:** API Endpoints: Define RESTful API endpoints for frontend-backend interactions (e.g., / api /food items, / api /orders /api /auth /login).

**Authentication:** Implement authentication using strategies like JWT (JSON Web Tokens) for securing endpoints.

- **Middleware:** Use Express.js middleware for handling requests, authentication checks, and error handling.

- **Error Handling:** Implement centralized error handling for API responses.

|  |  |
| --- | --- |
| **Payment Gateway** | : It uses Stripe or Razorpay |

|  |  |
| --- | --- |
| **File Storage (Images)** | : It uses Cloudinary or AWS S3 |

|  |  |
| --- | --- |
| **Email/SMS Notifications** | : It uses Node mailer or Twilio |

**Database Schema and Interactions (MongoDB):** The Database section represents the database that stores the items for Users, cart, Orders and Food Items.

**Collections:**

**-Items :** Store food item details (e.g., name, cost, description, discount).

**- Users:** Store user details (e.g., email, password hash, role).

- **Orders:** Store order details (e.g., user ID, items, total, status).

- **Interactions:**

**-** Use Mongoose or MongoDB driver for Node.js to interact with MongoDB.

- Perform CRUD operations (create, read, update, delete) for items, users, orders.

**Setup Instructions**

the setup instructions for the **Order On The Go** online food application, including prerequisites and a step-by-step installation guide.

**## Prerequisites**

To set up and run the Order On The Go application, you'll need the following software dependencies installed on your machine:

**- Node.js:** Version 16 or later (for running the backend and potentially build scripts for the frontend).

**- MongoDB:** For storing application data (items, users, orders).

**- npm or yarn:** For managing dependencies and running scripts.

**## Installation**

Follow these steps to clone, install dependencies, and set up the environment variables for the Order On The Go application:

**### Step 1: Clone the Repository**

git clone <https://github.com/your-repo/Order> On The Go.git

cd Order On The Go

(Replace <https://github.com/your-repo/Order> On The Go.git with the actual repository URL if different.)

**### Step 2: Install Dependencies**

Assuming you have a typical Node.js project structure with a backend or server and possibly a frontend or client folder:

**# Install backend dependencies**

cd server

npm install

**# If there's a client folder using React, install its dependencies too**

cd client

npm install react-scripts

**### Step 3: Set Up Environment Variables**

You'll need to set up environment variables for the application. Typically, this involves creating a .env file in the backend directory with variables like:

**# .env in backend folder**

**MONGO\_URI**=mongodb://localhost:27017/shopEZ

**JWT\_SECRET**=your-secret-key-for-jwt

**PORT**=3001

**- MONGO\_URI:** Connection string for your MongoDB instance.

**- JWT\_SECRET:** Secret key for signing JWT tokens for authentication.

**- PORT:** Port on which the backend server will run.

**### Step 4:** Run the Application

# Start the backend server

cd server

npm install -g nodemon

nodemon index.js

# In a separate terminal, if applicable, start the frontend development server

cd client

npm update

npm start

**## Next Steps**

**- Populate the database:** You might need to add initial data (like products) to MongoDB for testing.

**- Test the application:** Verify that both frontend and backend are working as expected by navigating to the application in a browser (usually http://localhost:3000 for React frontend if using default settings).

**## Folder Structure**

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

The client folder contains the structure for the React frontend. Here's what's in it:

**- public:** Typically holds public assets and the main entry point for the app.

**- Src :** Contains the source code for the React application.

**- .gitignore :** Specifies files to ignore in version control.

**- Package-lock.json :** Automatically generated for npm operations.

**- Package.json :** Holds metadata and dependencies for the project.

**- README.md :** Documentation for the project.

**### Server (Node.js Backend)**

The server folder is organized for the Node.js backend. Here's the breakdown:

**- controllers:** Handles business logic for routes.

**- middleware:** Contains custom middleware functions for the app.

**- models:** Defines database models (likely for MongoDB given the context of Order On The Go).

**- routes:** Defines API routes for the backend.

**- .env:** Stores environment variables for the project.

**- index.js:** Main entry point for the Node.js server.

**- Package-lock.json:** Automatically generated for npm operations.

**- Package.json:** Holds metadata and dependencies for the project.

**## Running the Application**

To run the Order On The Go application locally, you'll need to start both the frontend and backend servers. Here are the commands:

**### Start the Frontend Server**

**Navigate to the client directory and run:**

cd client

npm start

This will start the React development server, typically making the app available at http://localhost:3000.

**### Start the Backend Server**

**In a separate terminal, navigate to the server directory and run:**

cd server

npm start

This will start the Node.js backend server, likely running on a port specified in the .env file (like http://localhost:3001).

**## Next Steps**

**- Access the application:** Open a browser and go to http://localhost:3000 to view the Order On The Go frontend.

**- Test functionality:** Try logging in, browsing products, or other features to ensure both frontend and backend are communicating correctly.

**## API Endpoints**

**Authentication**

**- POST /api/auth/login**

- Parameters: email, password

- Example Response: { "token": "jwt-token", "user": { "id": "user-id", "email": "user@example.com" } }

**- POST /api/auth/register**

- Parameters: email, password

- Example Response: { "message": "User registered successfully", "user": { "id": "user-id", "email": "user@example.com" } }

**### Products**

**- GET /api/products**

- Parameters: None

- Example Response: [ { "id": "item-id", "name": "Item Name", "cost": 350 ,”discount”:5%}, ... ]

- GET /api/items/:id

- Parameters: id (item ID)

- Example Response: { "id": "item-id", "name": "Item Name", "cost": 350,”discount”:5% }

**### Orders**

**- POST /api/orders**

- Parameters: products (array of product IDs and quantities), userId

- Example Response: { "message": "Order placed successfully", "order": { "id": "order-id", "total": 332.5 } }

**- GET /api/orders**

- Parameters: None

- Example Response: [ { "id": "order-id", "userId": "user-id", "total": 332.5 }, ... ]

**### User Profile**

- GET /api/users/:id

- Parameters: id (user ID)

- Example Response: { "id": "user-id", "email": "user@example.com" }

**## Notes**

**- Authentication:** Some endpoints might require a valid JWT token in the Authorization header.

**- Error Handling:** Endpoints typically return error responses with a message indicating the issue (e.g., { "message": "Item not found" }).

**## Authentication and Authorization**

In the **Order On The Go** project, authentication and authorization are likely handled using JSON Web Tokens (JWT). This is a common approach for securing RESTful APIs.

**### Authentication**

**- Login:** When a user logs in with valid credentials (email and password), the backend verifies the credentials.

**- Token Generation:** Upon successful verification, the backend generates a JWT token containing user details (like userId , email) and sends it back to the client.

**- Client-Side Storage:** The client (React frontend) typically stores this token in local storage or cookies.

**### Authorization**

**- Token Validation:** For protected routes (like placing an order or accessing user-specific data), the client includes the JWT token in the Authorization header of requests.

**- Backend Verification:** The backend verifies the token on each request to protected endpoints. If valid, it grants access; otherwise, it returns an error (like 401 Unauthorized).

**### Methods Used**

**- JWT Tokens:** Used for stateless authentication, allowing the backend to verify requests without needing server-side sessions.

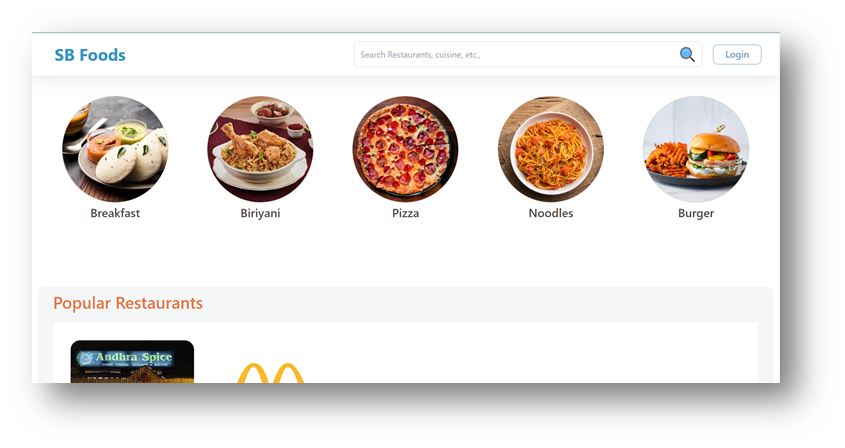
**- No Sessions:** Since JWT is stateless, there's no need for traditional session management on the server.

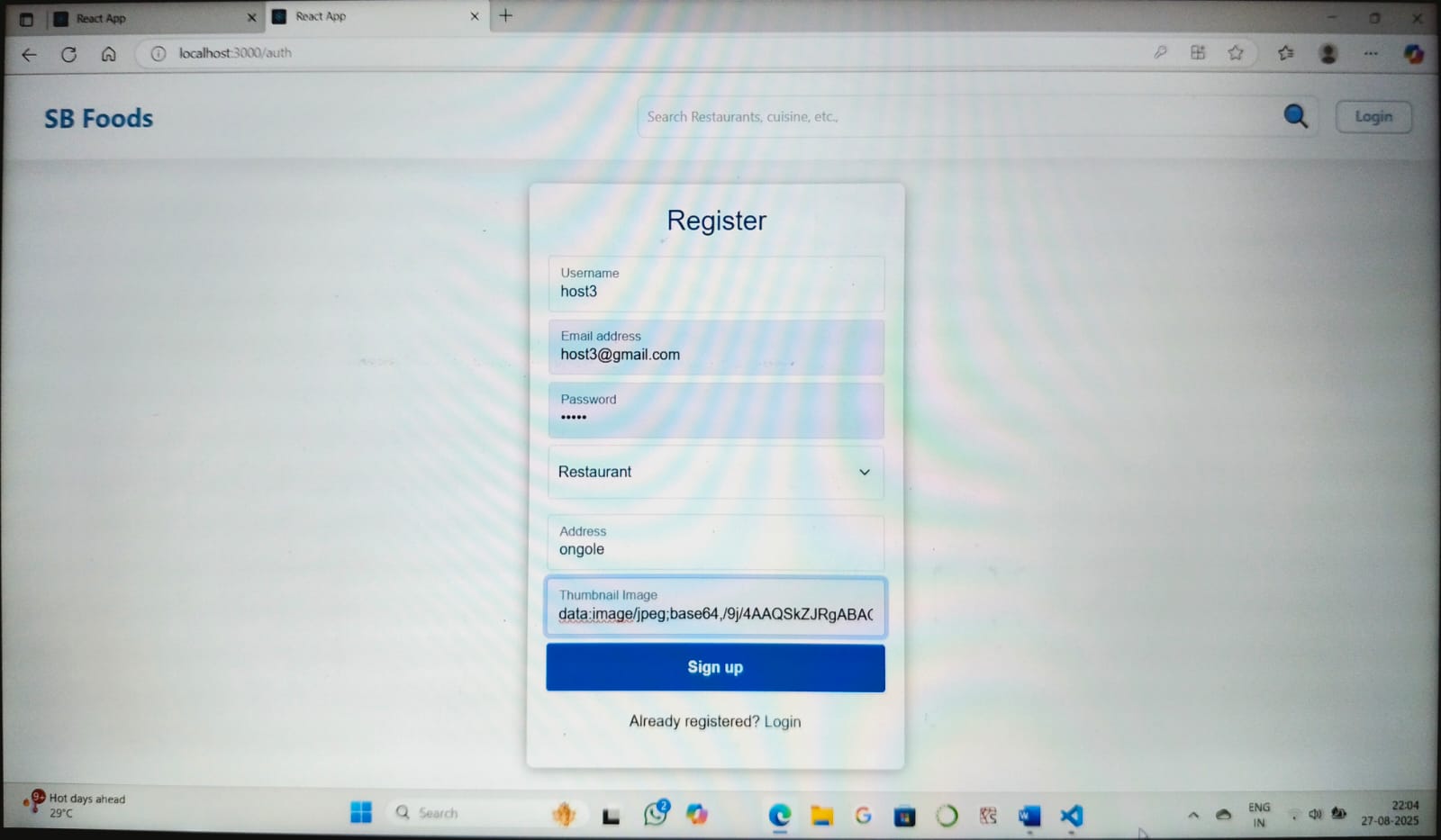
**## Security Considerations**

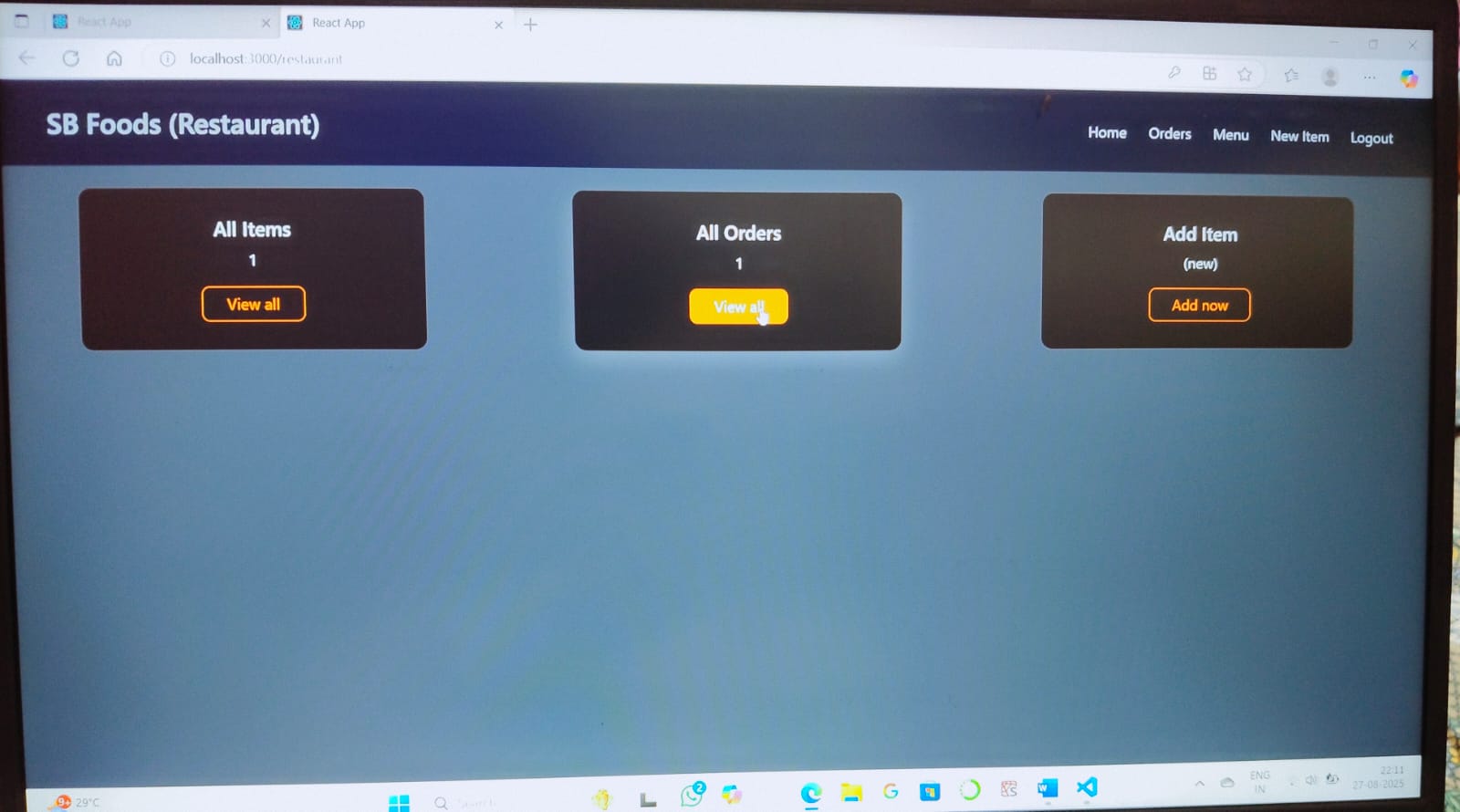
**- Token Expiration:** JWT tokens can have an expiration time for security.

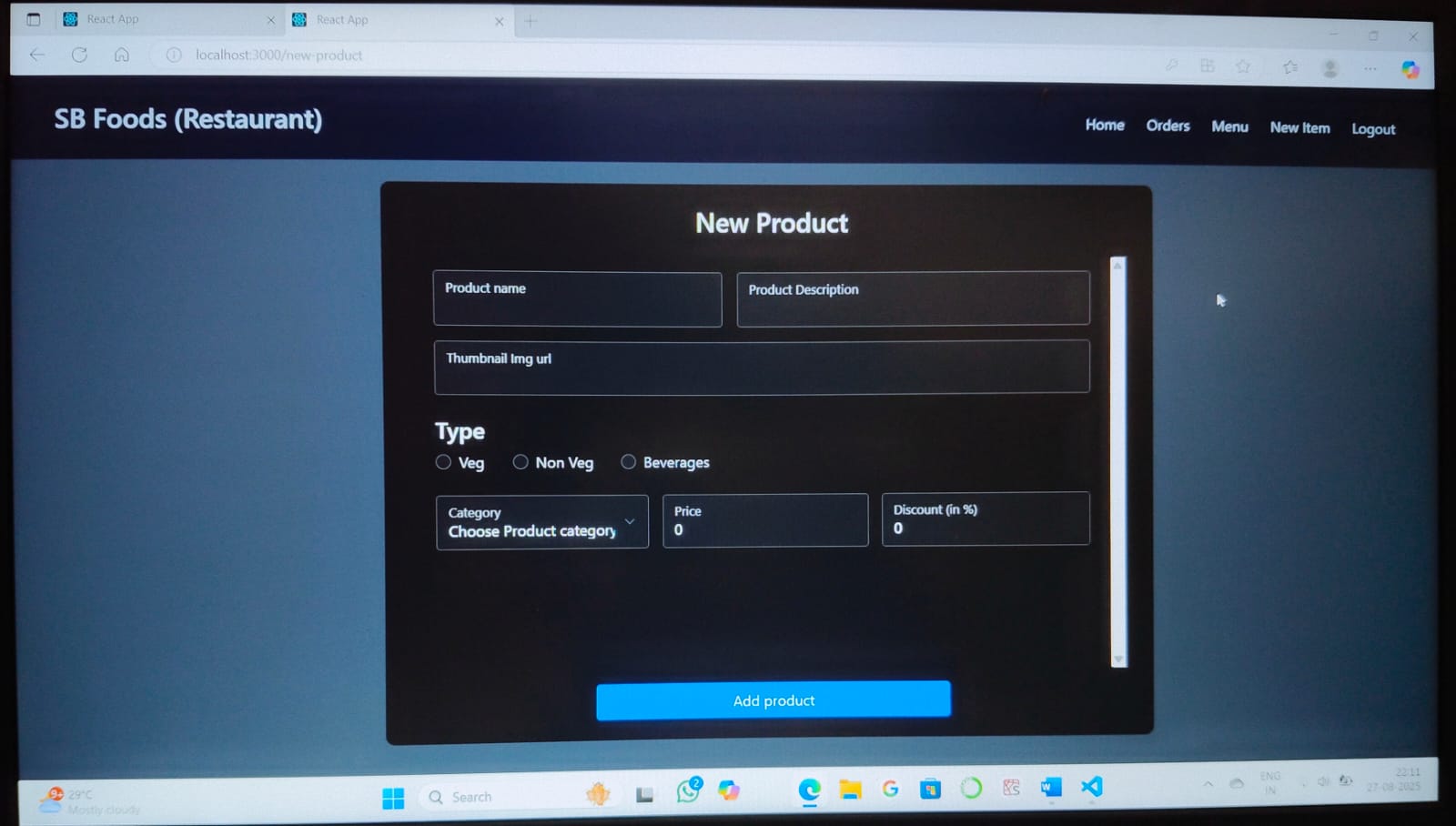
**- Secure Storage:** On the client, tokens should be stored securely (like using HTTP-only cookies for better security against XSS).

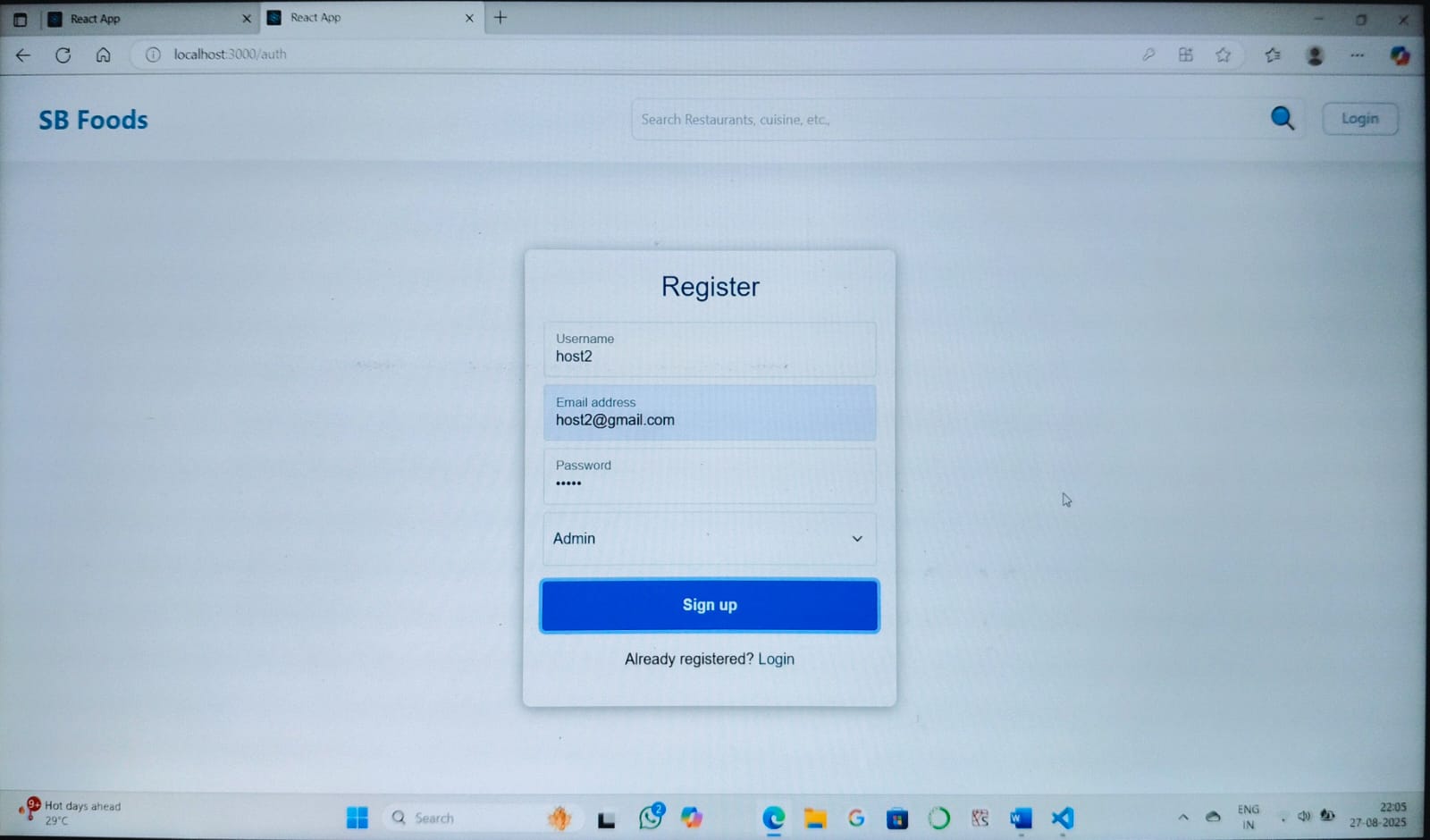
**User Interface:**

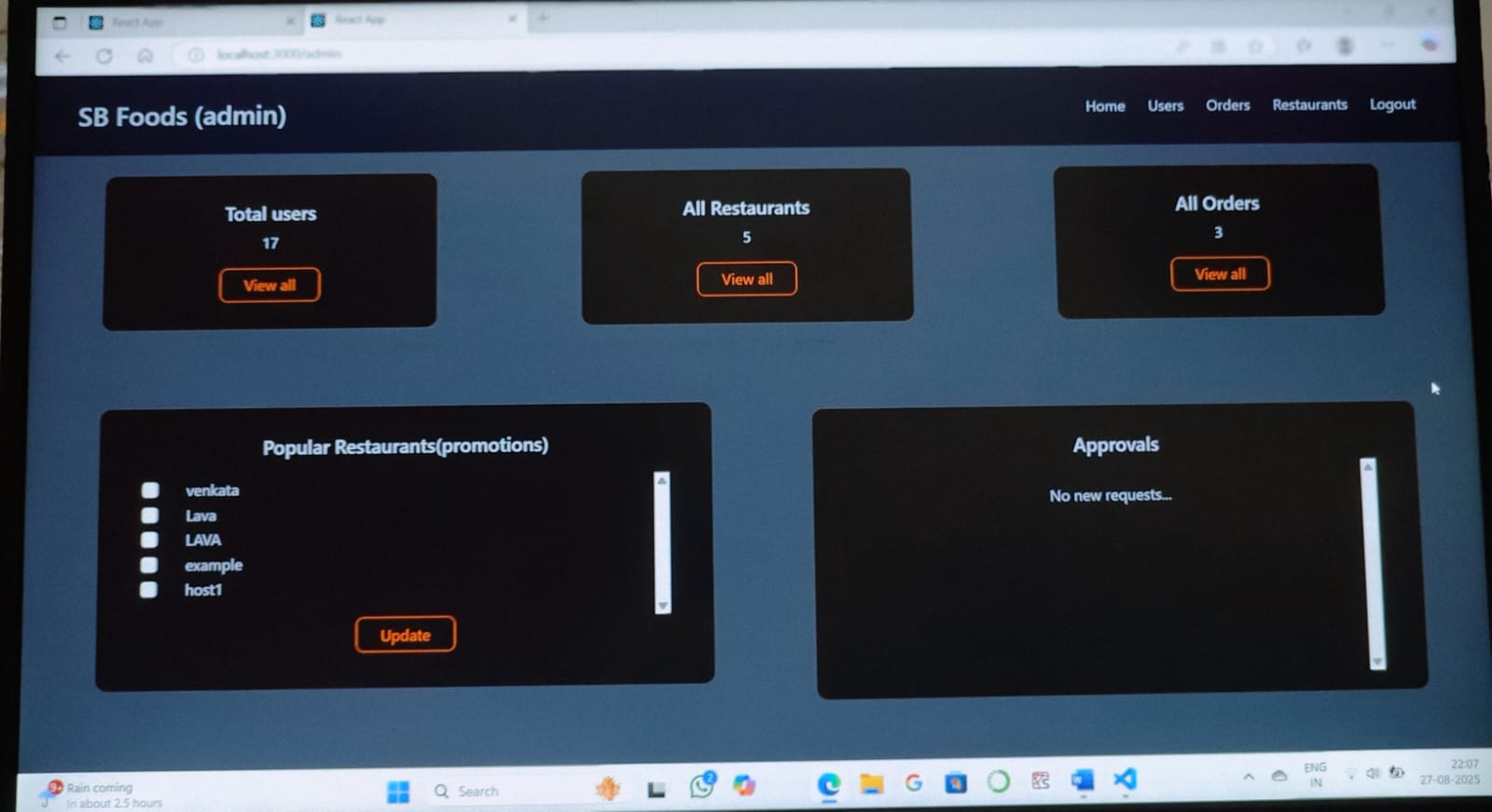


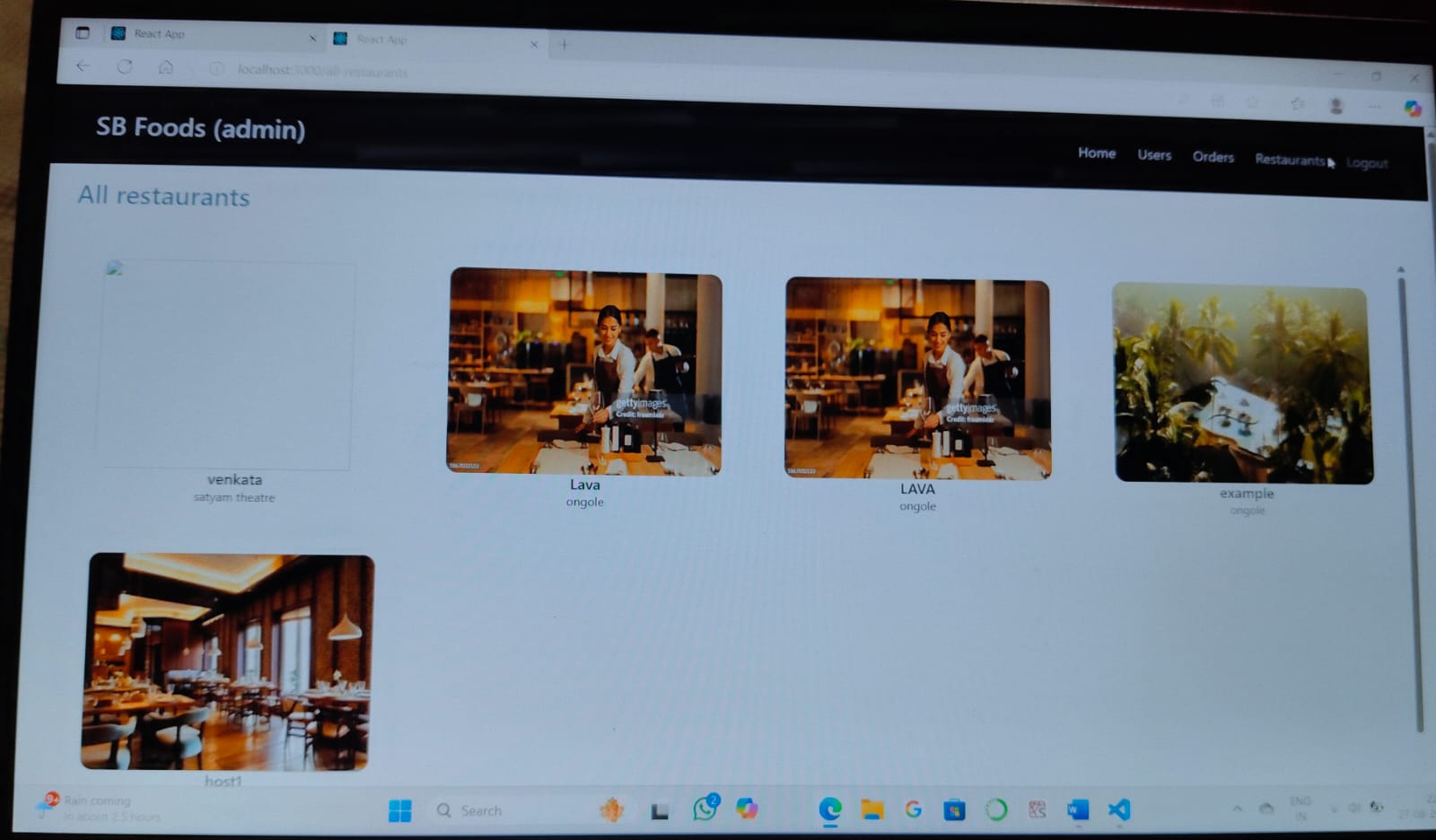


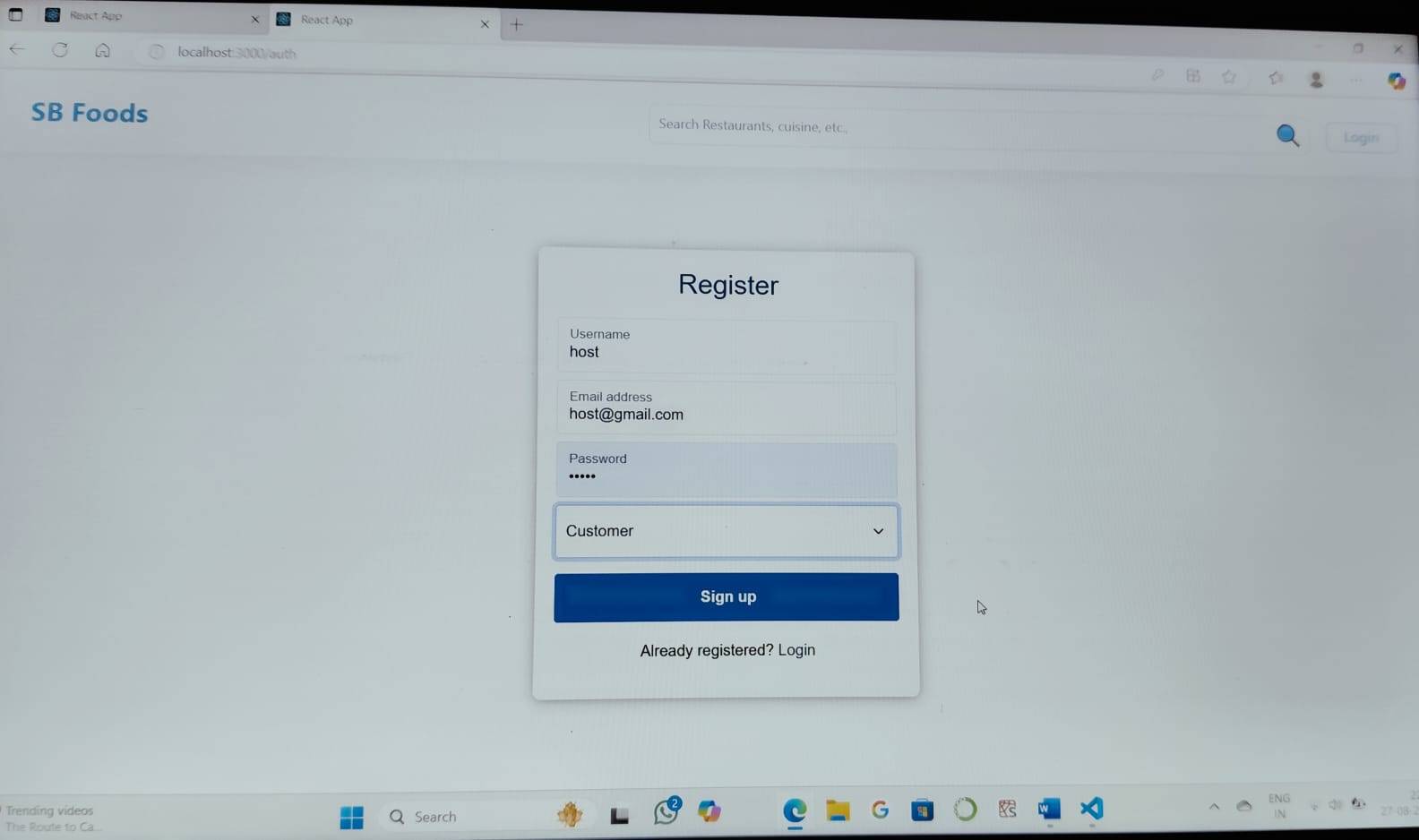












**## Testing Strategy and Tools**

For the Order On The Go application, a typical testing strategy involves a combination of unit tests, integration tests, and potentially end-to-end (E2E) tests to ensure the application works as expected.

**### Testing Tools**

**- Backend (Node.js):**

- Jest: A popular testing framework for unit and integration tests.

- Supertest: For testing API endpoints.

**- Frontend (React):**

- Jest: For unit testing components and logic.

- React Testing Library: For testing React components ’behaviour’.

**### Testing Strategy**

- Unit Tests: Test individual functions, models, or components in isolation.

- Integration Tests: Test how parts of the application work together (like API routes interacting with the database).

- E2E Tests: Optionally, test the entire application flow (like user login, adding Food items to cart, checkout) using tools like Cypress.

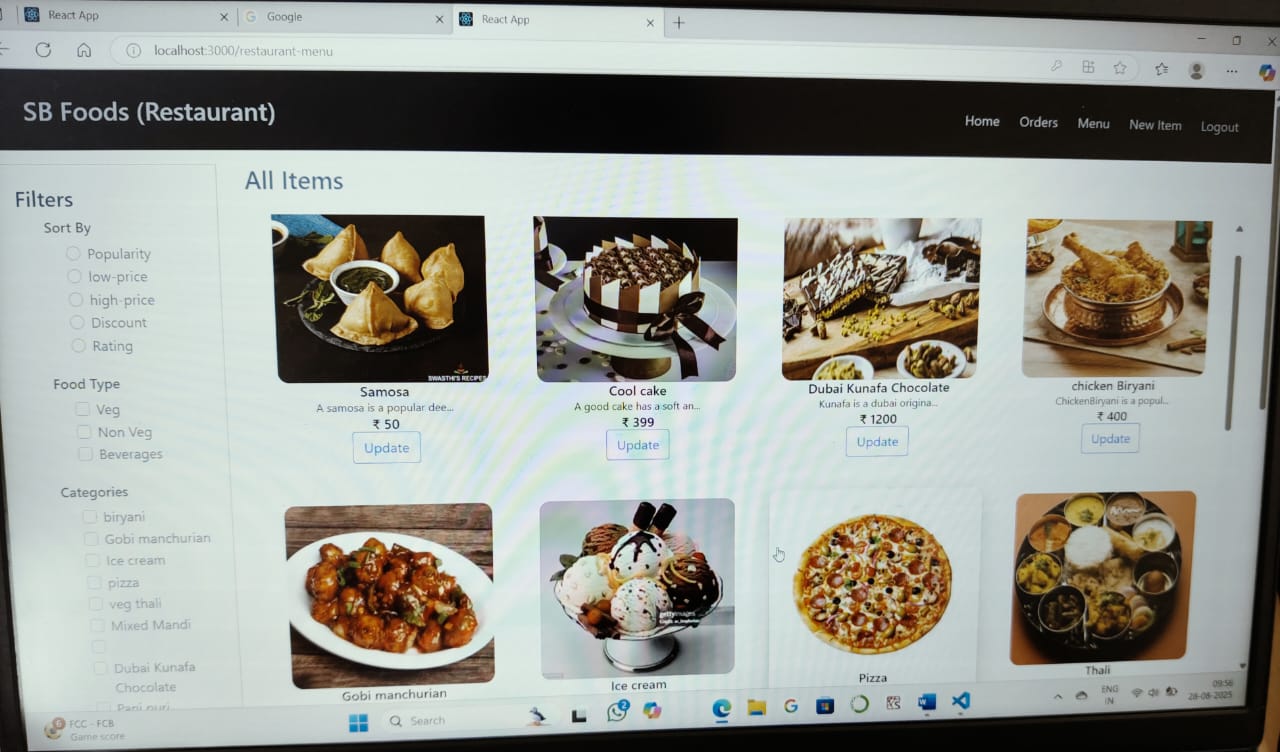
**## Example Testing Scenarios**

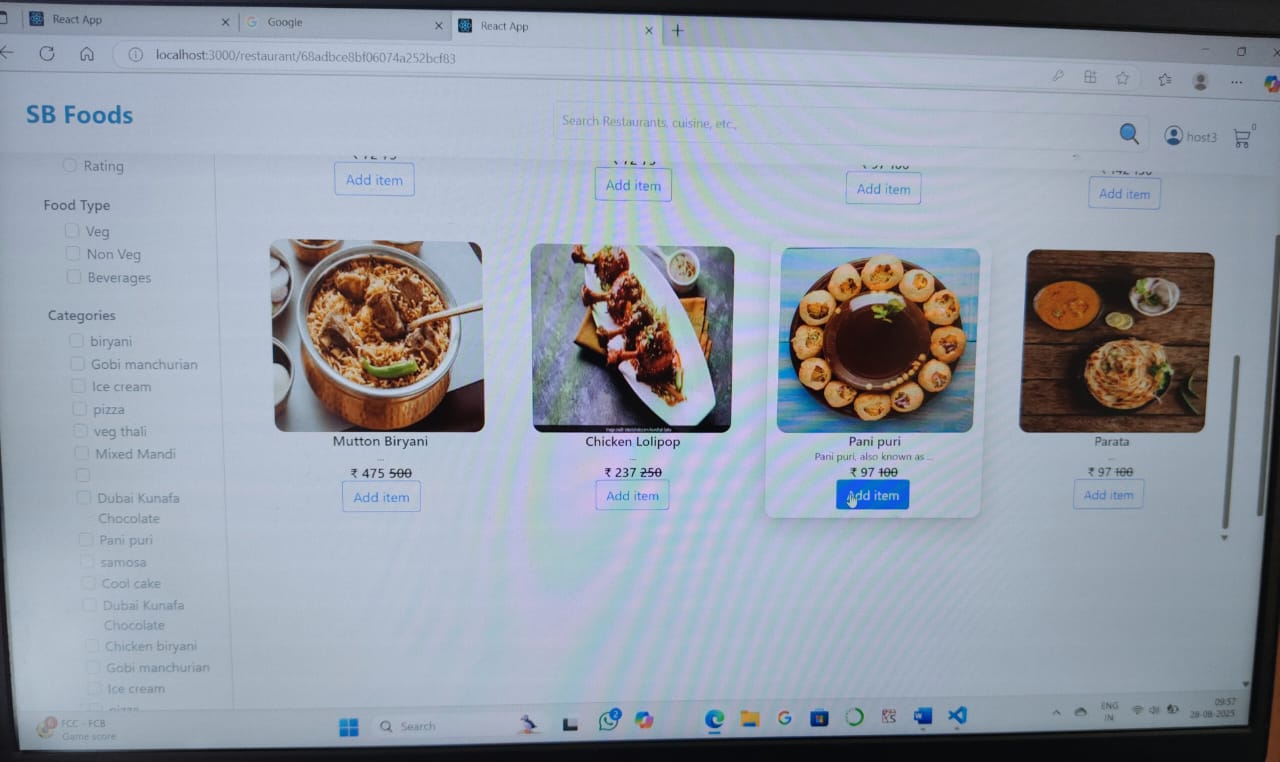
- Test if a user can successfully log in via the /api/auth/login endpoint.

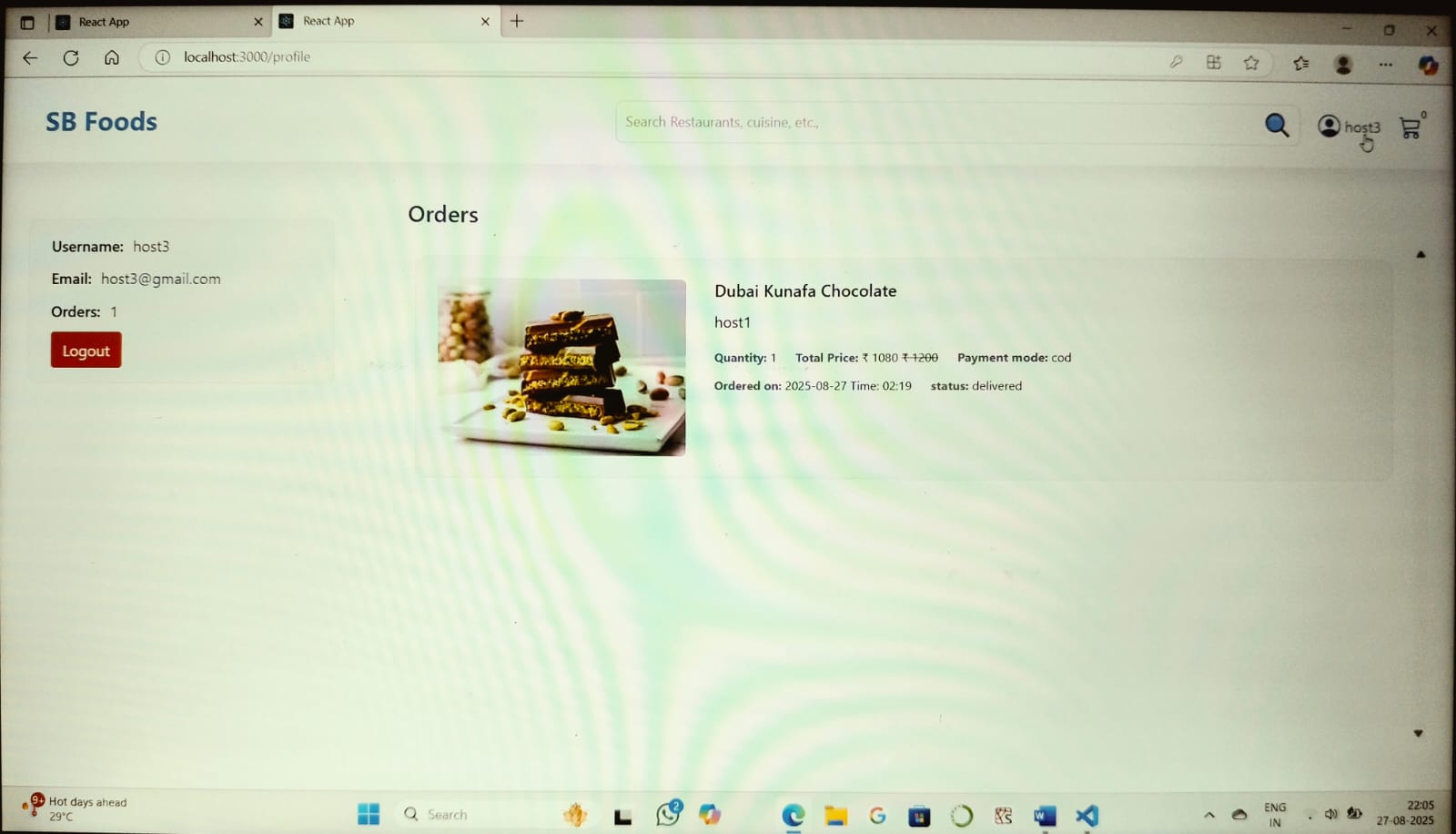
- Test if a product is correctly added to the cart in the React frontend.

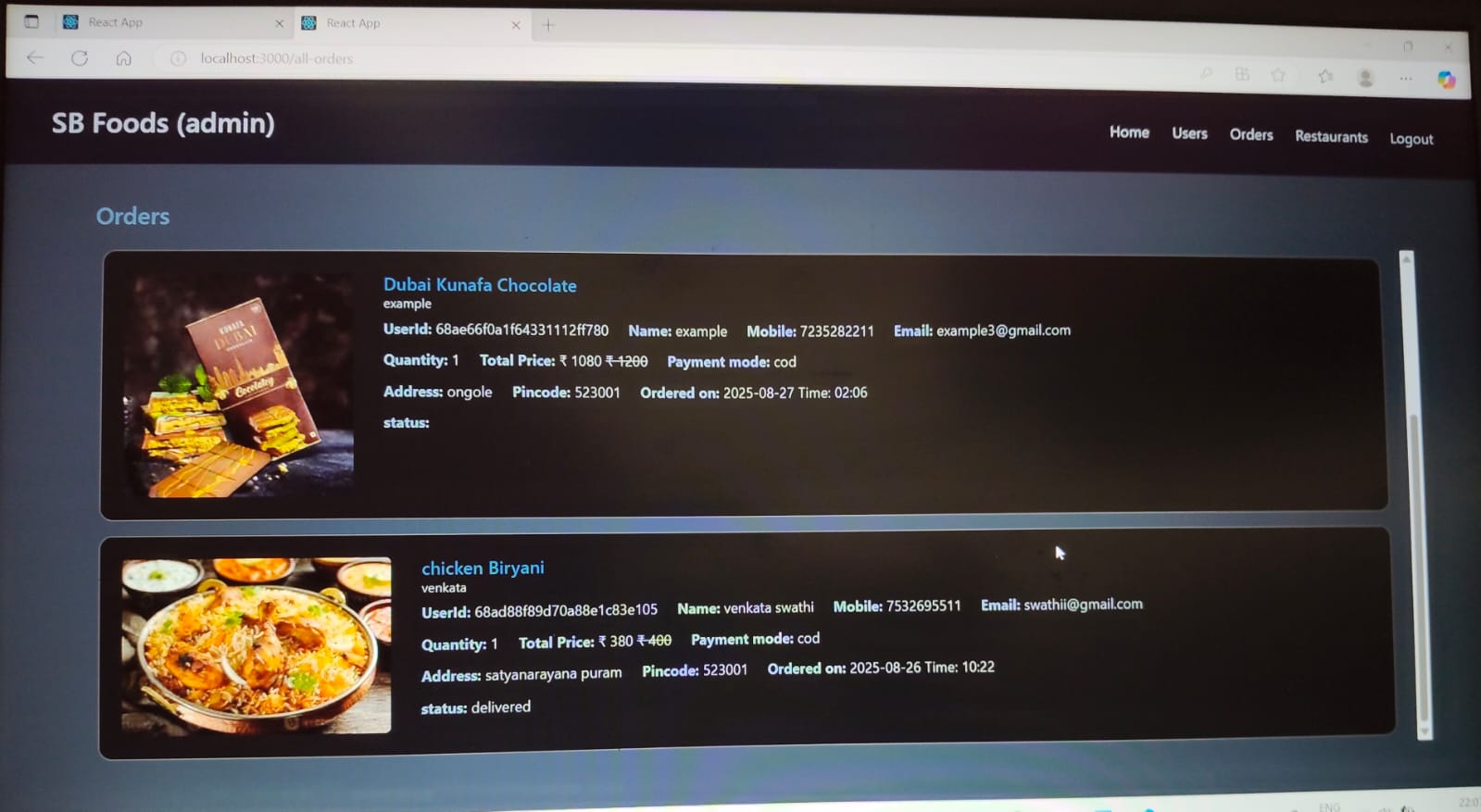
- Test API endpoints for products to ensure they return data as expected.

**Screenshots or Demo**









**## Known Issues**

For the Order On The Go application, here are some potential known issues that users or developers might need to be aware of. Since Order On The Go is a hypothetical application for this conversation, I'll outline common issues that could apply to an e-commerce app like Order On The Go:

**- Cart Sync Issue:** Potential issue with cart data syncing between logged-in and guest states.

**- Payment Gateway Integration Bugs:** Possible issues with payment processing in live mode.

**- Product Image Upload Bugs:** Issues might occur when uploading product images in certain formats or sizes.

**-Slow or Unreliable Data Sync:**

Delays or errors in synchronization can trigger inaccurate inventory or shipping data.

-**Shipping & Label Generation Problems:**

Manual or disconnected label generation can cause delays or mismatched shipping methods, affecting delivery expectations.

-**Lack of Real-Time Tracking & Notifications:**

* Without live updates, customers are left in the dark, increasing support queries and reducing trust.

**## Workarounds or Fixes**

- For cart sync issues, ensure users log in before adding items to the cart.

- For payment gateway issues, double-check API keys and configurations.

- For image upload bugs, validate image formats and sizes on upload.

**## Future Enhancements**

For the Order On The Go application, here are some potential future features or improvements that could enhance the project:

* 1. **AI-Powered Personalization & Intelligent Recommendations:**
* Leverage AI and machine learning to analyse user behaviour, past orders, and dietary preferences to deliver highly tailored menu suggestions.
* Include features like personalized promotions or loyalty rewards that adapt based on user habits.

**2. Voice Ordering & Virtual Assistants:**

* Integrate voice recognition so users can place orders hands-free using voice commands—ideal for on-the-go usage or accessibility.

3.**Real-Time Tracking & IoT Integration:**

* Enable live tracking of orders using GPS, improved with IoT devices like temperature sensors to ensure food freshness.
* Support customizable push notifications to keep users updated and engaged.

**4.Augmented Reality (AR) Menu & Immersive Experiences:**

* Implement AR so users can preview dishes in 3D before ordering, helping manage expectations and reduce disappointment.
* Offer virtual restaurant tours through AR/VR for immersive restaurant discovery.

**5. Smart Order Scheduling & Dynamic Pricing:**

* Allow users to schedule orders in advance—for family meals, business lunches, or meeting deliveries.
* Consider dynamic pricing models where rates adjust based on supply, demand, or delivery times.

**6.Advanced Payments & Contactless Experience:**

* Support multiple secure payment options including UPI, digital wallets, cards—and even crypto if desired.
* Offer contactless deliveries, with flexible delivery instructions such as “leave at door” for safety and convenience.

**7.Autonomous Delivery Solutions:**

* Consider integrating drone or autonomous vehicle options for ultra-fast, cost-effective delivery—especially to hard-to-reach spots.

**## Potential Improvements**

**- UI/UX Enhancements:** Improve the user interface and experience for better engagement.

**- Security Enhancements:** Implement additional security measures like two-factor authentication.

**- Analytics Integration:** Add analytics to track user behaviour and improve marketing strategies.

**- Smart Search & Filtering:**

* Add filters for cuisine, price, ratings, dietary restrictions.
* Use typeahead search that autocompletes restaurant or food item names.

- **Dark Mode and Accessibility:**

* Add support for dark mode for user comfort.
* Include larger fonts, screen reader support, and high-contrast themes for accessibility.

**-Secure Payments & OTP Confirmation**

* Integrate multiple secure payment methods: UPI, cards, wallets.
* Add OTP confirmation for high-value orders or account recovery.

**- Crash Reporting & Error Handling**

* Use tools like Firebase Crashlytics or Sentry to catch app errors.
* Implement graceful failure handling with fallback UIs.