CRAVIO - FOOD ORDERING APP

TEAM MEMBERS	ROLES
JACKLIN SIBIYAL L	BACKEND
KAVYA M	FRONTEND
ANUSHA A	UI/UX DESIGNER
MANIKANDAN N	BACKEND
DIVYASRI	FRONTEND

CONTENTS

CRAVIO - FOOD ORDERING APP

1. Project Overview

- **Purpose**:
  Cravio is a modern and dynamic food ordering platform tailored to meet the evolving needs of both customers and restaurant partners. Its primary aim is to simplify the food ordering process through a seamless and intuitive interface. The project provides a centralized digital solution for:
  - Customers to explore diverse menu options, place orders, and track their meals.
  - Restaurant owners to manage their listings and orders efficiently.
  - Administrators to oversee the platform's operations, ensuring quality and security.
- **Goals**:
  - To create a scalable platform capable of handling multiple user roles and activities.
  - To integrate secure payment processing for hassle-free transactions.
  - To maintain a responsive design for accessibility across devices like smartphones, tablets, and desktops.
  - To offer tools that empower restaurants to expand their customer base online.
- **Features**:
  - **User Management**:
    - Registration and login functionalities.
    - Profile updates and order history tracking.
  - **Menu and Product Catalog**:
    - Comprehensive menu browsing with detailed descriptions, customer reviews, pricing, and discounts.
    - Filtering options for categories and promotions.
  - **Order Management**:
    - Cart management with item additions, modifications, and deletions.
    - Order placement with secure payment integration.
    - Real-time order tracking for users.
  - **Restaurant Dashboard**:
    - Menu management tools for adding, editing, or removing items.
    - Order status updates to reflect progress like "In Progress," "Out for Delivery," or "Completed."
  - **Admin Panel**:
    - Platform-wide monitoring and management of users, restaurants, and products.
    - Tools for resolving disputes and handling promotional content like banners.

2. Architecture

- **Frontend**:
  The frontend of Cravio is built using **React.js**, ensuring a dynamic and responsive user interface. Key architectural aspects include:
  - **Component-Based Structure**:
    - Reusable components like Navbar, Footer, Login Form, and Cart ensure consistency and scalability.

- o **UI Libraries**:
  - Material UI and Bootstrap enhance the aesthetic appeal and usability of the interface.
- o **State Management**:
  - Context API is utilized for managing global states like user authentication and cart contents without excessive prop drilling.
- o **Routing**:
  - React Router facilitates seamless navigation between different pages such as restaurants, cart, and profile.
- **Backend**:
  The backend, powered by **Node.js** and **Express.js**, serves as the core of Cravio's operations. Key elements include:
  - o **API Development**:
    - RESTful APIs handle requests for user registration, menu browsing, order placement, and more.
  - o **Middleware**:
    - Middleware ensures smooth data processing, error handling, and cross-origin compatibility (CORS).
  - o **Business Logic**:
    - Controllers process requests and manage database interactions for seamless functionality.
- **Database**:
  Cravio uses **MongoDB**, a NoSQL database, for efficient data storage and retrieval. Features include:
  - o **Schema Management**:
    - Mongoose is employed for defining schemas and relationships between entities like Users, Restaurants, Food Items, and Orders.
  - o **Entity Overview**:
    - **User**: Manages user-specific data, including login credentials, order history, and roles (Customer, Restaurant Owner, Admin).
    - **Restaurant**: Stores restaurant details like name, address, menu, and order tracking.
    - **Food Items**: Catalogs details about dishes, including descriptions, prices, and promotional discounts.
    - **Orders**: Tracks order statuses, payment methods, and delivery timelines.
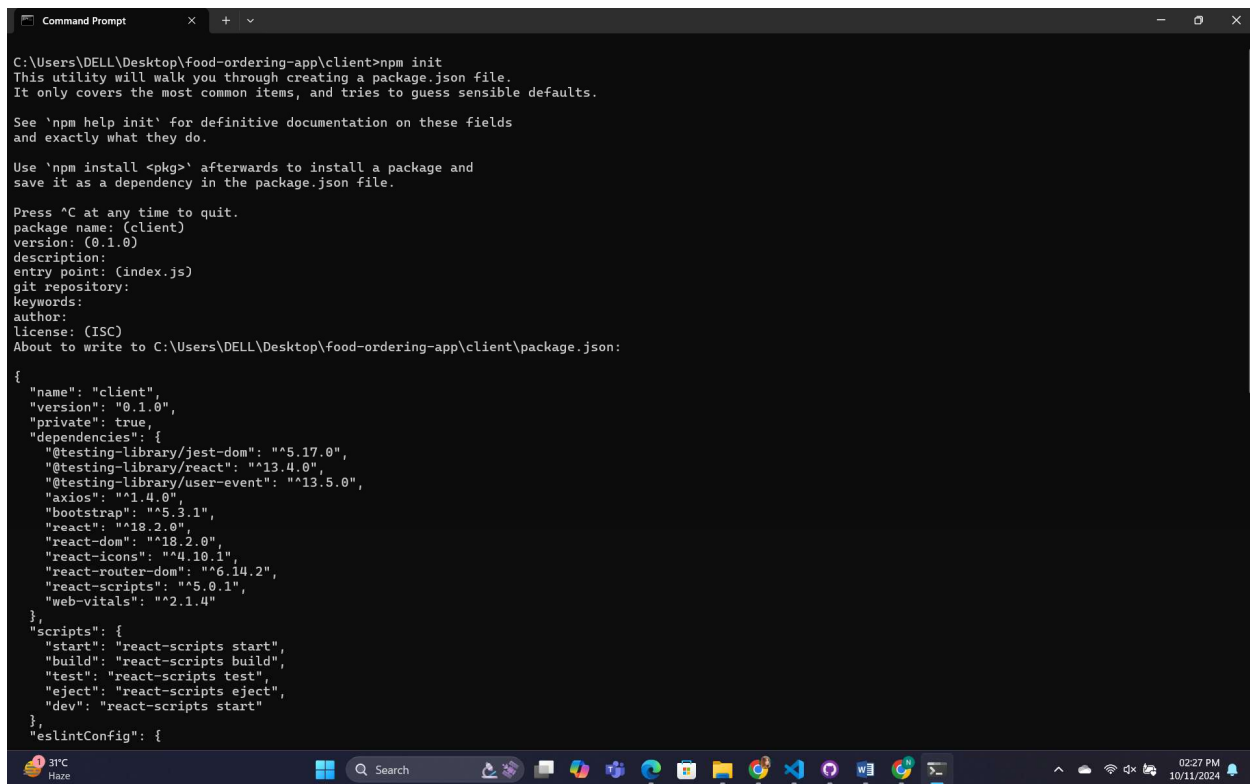
3. Setup Instructions
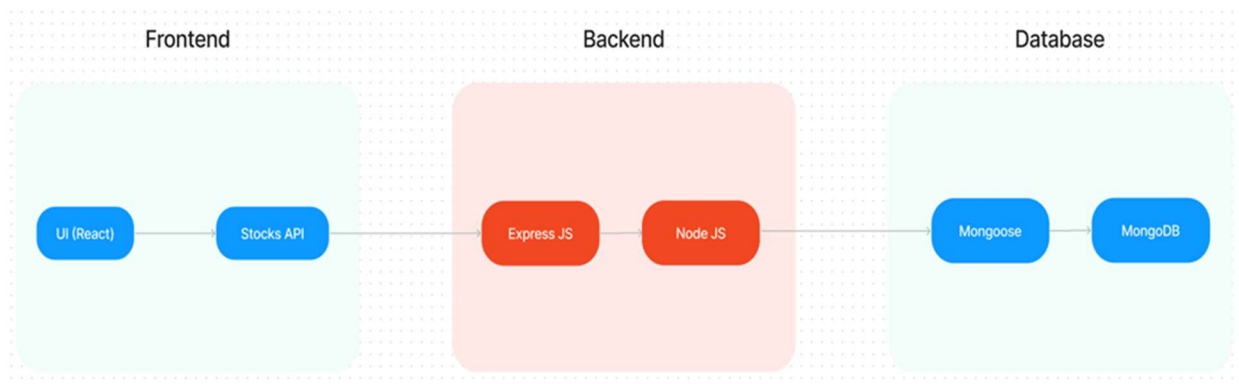
- **Prerequisites**:

To develop the Cravio Food Ordering App using the MERN stack, here are the essential prerequisites and setup instructions:

- **Node.js and npm**:
  - **Node.js** is a powerful runtime that enables server-side JavaScript execution, providing a scalable platform for network applications.
  - Install Node.js and npm (Node Package Manager) as they are required to run JavaScript on the server.
  - **Download**: Node.js Download
  - **Installation Guide**: Node.js Installation Instructions
  - Initialize npm: `npm init` to set up package dependencies.

```
C:\Users\DELL\Desktop\food-ordering-app\client>npm init
This utility will walk you through creating a package.json file.
It only covers the most common items, and tries to guess sensible defaults.

See `npm help init` for definitive documentation on these fields
and exactly what they do.

Use `npm install <pkg>` afterwards to install a package and
save it as a dependency in the package.json file.

Press ^C at any time to quit.
package name: (client)
version: (0.1.0)
description:
entry point: (index.js)
git repository:
keywords:
author:
license: (ISC)
About to write to C:\Users\DELL\Desktop\food-ordering-app\client\package.json:

{
  "name": "client",
  "version": "0.1.0",
  "private": true,
  "dependencies": {
    "@testing-library/jest-dom": "^5.17.0",
    "@testing-library/react": "^13.4.0",
    "@testing-library/user-event": "^13.5.0",
    "axios": "^1.4.0",
    "bootstrap": "^5.3.1",
    "react": "^18.2.0",
    "react-dom": "^18.2.0",
    "react-icons": "^4.10.1",
    "react-router-dom": "^6.14.2",
    "react-scripts": "^5.0.1",
    "web-vitals": "^2.1.4"
  },
  "scripts": {
    "start": "react-scripts start",
    "build": "react-scripts build",
    "test": "react-scripts test",
    "eject": "react-scripts eject",
    "dev": "react-scripts start"
  },
  "eslintConfig": {
```

- **Express.js**:
  - **Express.js** is a lightweight framework for Node.js, simplifying API development and handling server-side routing, middleware, and modular architecture.
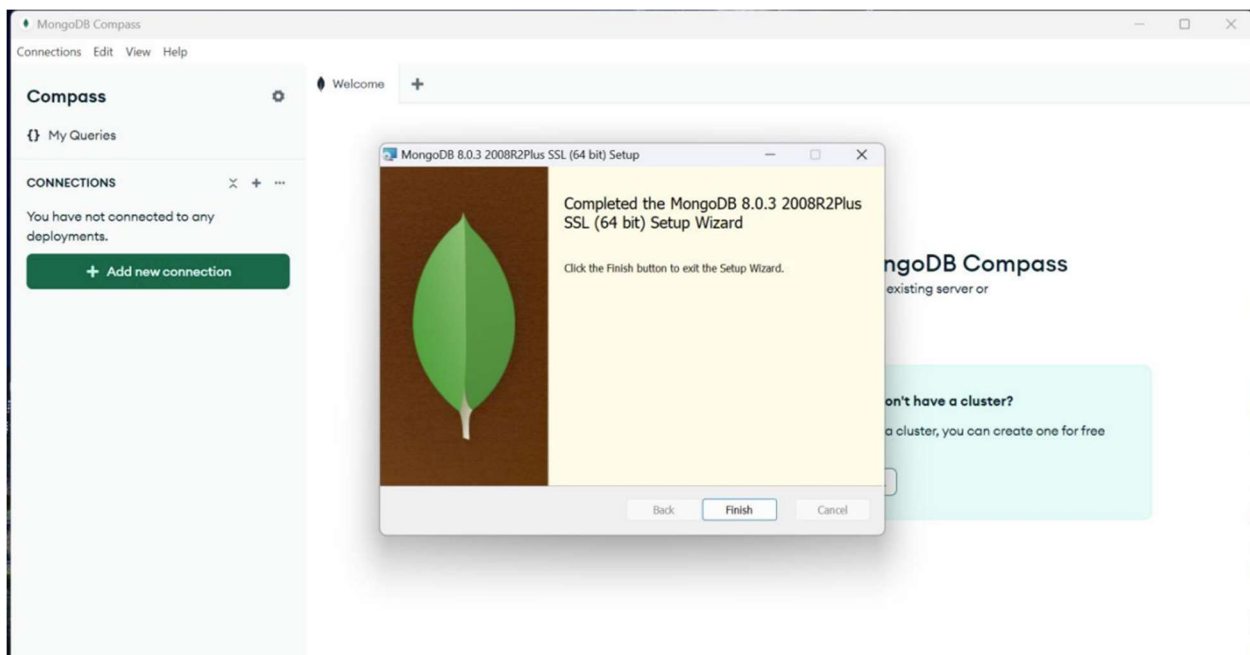 - Install Express with: `npm install express`

- **MongoDB**:
  - MongoDB is a NoSQL database ideal for handling large volumes of structured and unstructured data.
  - Set up MongoDB to store application data.
  - **Download**: MongoDB Download
  - **Installation Guide**: MongoDB Installation Instructions

- **React.js**:
  - React.js is a JavaScript library for building dynamic and responsive UIs with reusable components.
  - **Install React.js**: Follow the setup guide at React.js Setup Guide
- **HTML, CSS, and JavaScript**:
  - Knowledge of HTML for structuring pages, CSS for styling, and JavaScript for client-side interactivity is essential.
- **Database Connectivity**:

o   Use Mongoose, an ODM library, or a MongoDB driver to connect the Node.js server with the MongoDB database for CRUD operations.
o   Install Mongoose: `npm install mongoose`
- **Front-end Libraries**:
  o   **React.js** serves as the main framework for the front end.
  o   Install UI libraries like Material UI and Bootstrap for improved design: `npm install @mui/material bootstrap`

Installation Commands

1. Navigate to the client folder and install dependencies:

```
cd client
npm install
```

2. Navigate to the backend folder and install dependencies:

```
cd ../server
npm install
```

3. Start the Development Server:
o   Run the application with: `npm start`
o   The Cravio app will be accessible at `http://localhost:3000` by default, ready for further development, customization, and testing.

5. Folder Structure

The **Cravio Food Ordering App** project is divided into separate folders for front-end (client) and back-end (server) code, each organized to facilitate a scalable, maintainable, and modular development process.

Front-end (Client) Structure

The front-end code is built using **React.js** and organized into folders and files to separate reusable components, pages, and styles. This modular structure allows for easy maintenance and further extension of the application.

- **Root Folders and Files**:
  o   **node_modules**:
    ▪   Contains all the dependencies and libraries installed via npm (Node Package Manager).
    ▪   Automatically generated when `npm install` is run in the front-end directory.
  o   **public**:
    ▪   Contains static assets that can be directly served by the web server.
    ▪   **index.html**: The main HTML file of the application where the React app is rendered. It contains a root `<div>` element into which the entire React application is injected.

- Other assets like images and icons may be stored here for direct access by the application.
    - o **src**:
        - This is the main source folder where all front-end components, pages, and logic are stored. It is the core of the front-end part of the application.
- **src Folder Structure**:
    - o **components**:
        - Houses reusable components that appear across different parts of the app, ensuring consistency and reducing redundancy.
        - **Footer.jsx**: Component for the website footer, displayed on all pages.
        - **Login.jsx**: Component for the login form, allowing users to enter their credentials and access their accounts.
        - **Navbar.jsx**: A navigation bar component with links to different sections of the site, such as Home, Profile, and Logout.
        - **PopularRestaurants.jsx**: Displays a list of popular restaurants, helping users discover popular dining options.
        - **Register.jsx**: Component for the user registration form, allowing new users to create an account.
        - **Restaurants.jsx**: Shows a list of all available restaurants on the platform.
    - o **context**:
        - Stores global state and context, allowing components to share data without the need for prop drilling.
        - Used for managing application-wide state, such as user authentication status and cart contents.
    - o **images**:
        - Contains media assets like icons, banners, and restaurant images used in the UI.
    - o **pages**:
        - Organizes pages based on user roles (admin, customer, restaurant), allowing for easy access and modular development.
        - **admin**:
            - Contains pages specific to admin functionalities, allowing admins to manage various aspects of the application.
            - **Admin.jsx**: Main dashboard for admin users, displaying an overview of platform activities.
            - **AllOrders.jsx**: Page displaying all orders placed on the platform, with filtering and sorting options for admin review.
            - **AllProducts.jsx**: Displays all products available on the platform, allowing admins to oversee the menu items.
            - **AllRestaurants.jsx**: Shows all registered restaurants on the platform, with options for managing or approving listings.
            - **AllUsers.jsx**: Displays all registered users on the platform, allowing the admin to manage user roles and permissions.
        - **customer**:
            - Pages for customer functionalities, focusing on the ordering experience and user profile management.
            - **Cart.jsx**: Displays items added to the user's cart, with options to adjust quantities or remove items.

- **CategoryProducts.jsx**: Shows food items within specific categories, enhancing browsing for customers based on preferences.
- **IndividualRestaurant.jsx**: Displays the menu and details of a single restaurant, including ratings and reviews.
- **Profile.jsx**: User profile page where customers can view and update their information, and check order history.
  - **restaurant**:
    - Pages for restaurant owners, enabling them to manage menu items, orders, and profile details.
    - **EditProduct.jsx**: Allows restaurant owners to edit details of existing menu items, such as price, description, and availability.
    - **NewProduct.jsx**: Provides a form for adding new products to the menu.
    - **RestaurantHome.jsx**: Home dashboard for restaurant users, displaying a summary of orders and recent activity.
    - **RestaurantMenu.jsx**: Displays the restaurant's menu, with options to update or delete items.
    - **RestaurantOrders.jsx**: Shows all orders placed with the restaurant, allowing the owner to track and update order status.
    - **Authentication.jsx**: Manages restaurant-specific authentication, verifying owner credentials and access rights.
  - o **styles**:
    - Contains CSS files for defining styles used throughout the application.
    - **App.css**: The main stylesheet, applying global styles and theming for the entire front-end.
  - o **App.js**:
    - The root component of the application, containing the main routes for navigating between pages.
    - It imports and renders primary components, managing the flow between different sections of the app.
  - o **index.js**:
    - The main entry point for the React application. It renders the root component (App) into the `index.html` file found in the `public` folder.

Back-end (Server) Structure

The back-end, built with **Node.js** and **Express.js**, handles API requests, manages database interactions, and enforces business logic.

- **node_modules**:
  - o Contains all the dependencies required for the back-end, such as Express, Mongoose, and JWT libraries.
- **server Folder Structure**:
  - o **index.js**:
    - The main entry point for the server, responsible for initializing the Express server, connecting to the MongoDB database, and setting up middleware, routes, and error handling.

- Configures the server to listen on a specified port and logs successful startup messages.
  - o **Schema.js**:
    - Defines Mongoose schemas and models for different entities in the application, such as users, restaurants, food items, carts, and orders.
    - Each schema specifies the data structure, validation rules, and relationships, ensuring consistency across database entries.
  - o **package.json**:
    - Contains metadata about the project, such as the name, version, scripts, and dependencies required for the server.
    - Manages backend libraries and tools, making it easy to install or update dependencies as needed.
  - o **package-lock.json**:
    - Locks dependency versions, ensuring that installations produce consistent results across different environments.Additional Project Configuration Files

- **.gitignore**:
  - o Specifies files and directories that should not be tracked by Git, such as `node_modules`, `.env`, and other sensitive files.
  - o Helps keep the repository clean and prevents unnecessary files from being pushed to version control.
- **README.md**:
  - o Contains documentation for the project, including an overview, setup instructions, and usage details.
  - o Often serves as the first reference for developers working on the project or new contributors.

6. Running the Application

The Cravio Food Ordering App can be run locally by following these steps:

Frontend:

1. Navigate to the `client` directory:

   `cd client`

2. Start the React development server:

   `npm start`

   o The application will be available at `http://localhost:3000`.

Backend:

1. Navigate to the `server` directory:

   `cd server`

2. Start the Express.js server:

   `npm start`

   o The server will be available at `http://localhost:5000`.
3. **Additional Notes**:
   o Ensure MongoDB is running either locally or through a MongoDB Atlas connection.
   o Use `.env` to configure the database connection string and other environment-specific settings.

7. API Documentation

The Cravio Food Ordering App backend exposes a series of RESTful APIs to facilitate communication with the frontend and manage user interactions. Below is a summary of key endpoints:

Authentication:

- **POST** `/api/auth/register`
   o **Description**: Registers a new user.
   o **Parameters**:
      ▪ `username` (String)
      ▪ `email` (String)
      ▪ `password` (String)
   o **Response**:

```json
Copy code
{
  "message": "User registered successfully",
  "userId": "12345"
}
```

- **POST** /api/auth/login
    - o **Description**: Authenticates a user and creates a session.
    - o **Parameters**:
        - ▪ email (String)
        - ▪ password (String)
    - o **Response**:

```json
Copy code
{
  "message": "Login successful",
  "userId": "12345",
  "role": "customer"
}
```

Products:

- **GET** /api/products
    - o **Description**: Retrieves a list of all food items.
    - o **Response**:

```json
Copy code
[
  {
    "id": "1",
    "name": "Pizza",
    "price": 12.99,
    "description": "Delicious cheese pizza",
    "category": "Main Course"
  }
]
```

- **POST** /api/products
    - o **Description**: Adds a new product (admin or restaurant role required).
    - o **Parameters**:
        - ▪ name (String)
        - ▪ price (Number)
        - ▪ description (String)
        - ▪ category (String)
    - o **Response**:

```json
Copy code
{
```

```
        "message": "Product added successfully",
        "productId": "67890"
      }
```

Orders:

- **GET** `/api/orders`
  - o **Description**: Retrieves all orders for admin review.
  - o **Response**:

    ```json
    json
    Copy code
    [
      {
        "id": "1",
        "userId": "12345",
        "items": [
          { "productId": "1", "quantity": 2 }
        ],
        "totalPrice": 25.98,
        "status": "Pending"
      }
    ]
    ```

- **POST** `/api/orders`
  - o **Description**: Places a new order for a user.
  - o **Parameters**:
    - `items` (Array of objects)
    - `address` (String)
    - `paymentMethod` (String)
  - o **Response**:

    ```json
    json
    Copy code
    {
      "message": "Order placed successfully",
      "orderId": "123456"
    }
    ```

This API documentation provides a clear guide for developers to integrate and interact with the backend.

8. Authentication

The Cravio Food Ordering App employs **session-based authentication** to manage user access and roles securely. Here's a detailed explanation of how authentication and authorization are implemented:

Methodology:

- **Session Management**:
  - Upon login, a session is created for the user and maintained through server-side logic.
  - Session information is stored securely on the server, preventing unauthorized access.
- **Role-Based Access Control (RBAC)**:
  - **Customers**: Can browse menus, add items to their cart, place orders, and view their order history.
  - **Restaurant Owners**: Can manage their menus, view orders, and update order statuses.
  - **Admins**: Have full access to monitor user activities, manage platform content, and resolve disputes.

Flow:

1. **Registration**:
   - A user provides details like name, email, and password to register.
   - Passwords are hashed before storage to enhance security.
2. **Login**:
   - User credentials are verified, and a session is created on successful authentication.
   - The session is linked to the user's role (customer, restaurant owner, or admin).
3. **Session Validation**:
   - For every authenticated request, the server checks the session for validity.
   - If the session is invalid or expired, the user is logged out.
4. **Protected Routes**:
   - Access to routes such as order management or admin dashboard is restricted based on user roles.
   - Middleware ensures only authorized users can access sensitive data or perform specific actions.

Security Measures:

- Password hashing using **bcrypt.js** to securely store user credentials.
- Enforcing HTTPS in production environments to secure data transmission.
- Timeout for sessions to prevent indefinite access.

9. User Interface

The Cravio Food Ordering App delivers an intuitive and visually appealing user experience, tailored for different roles such as customers, restaurant owners, and administrators. Each interface is designed to simplify workflows, provide clarity, and enhance the overall experience. Here's an overview of the key UI features, along with their descriptions:

Landing Page

- **Purpose**:
  The initial page users see when they visit the application, featuring an overview of the platform and highlights of popular restaurants.

- **Key Elements**:
  - A prominent search bar for quick restaurant or menu item searches.
 - Featured sections showcasing popular dishes, ongoing discounts, and top-rated restaurants.
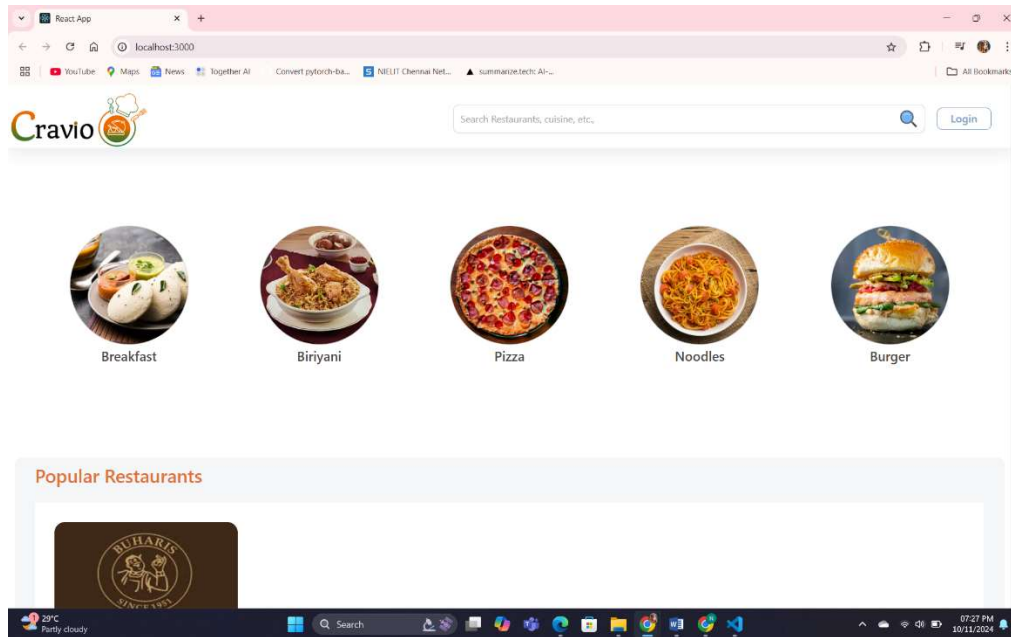
Restaurants Page

- **Purpose**:
  Displays a list of all available restaurants on the platform, allowing users to browse and select a restaurant.
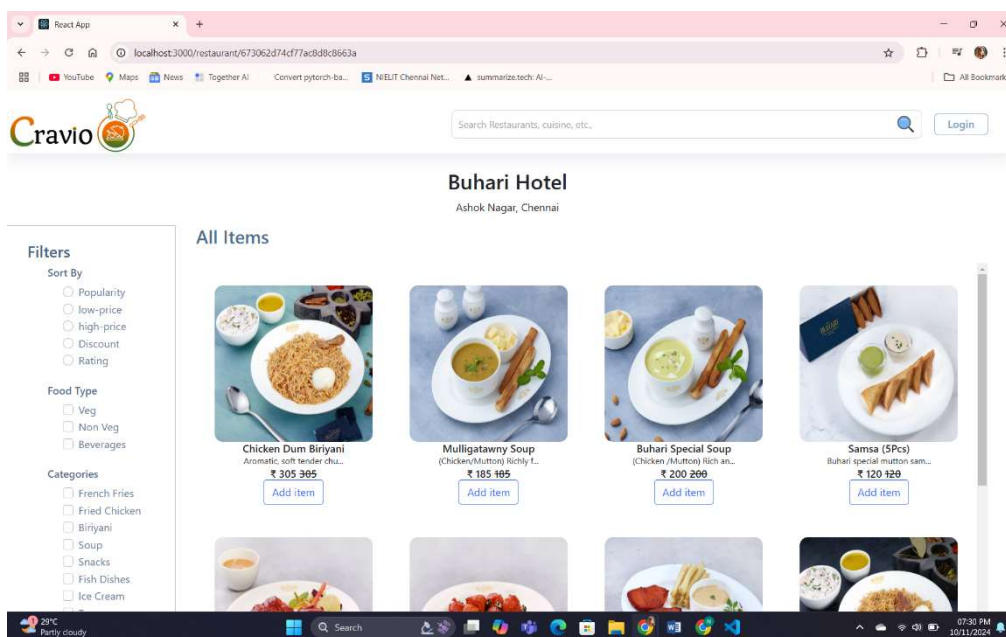- **Key Elements**:
  - Restaurant cards displaying names, ratings, and basic descriptions.
 - Filtering and sorting options by cuisine, price range, or location.
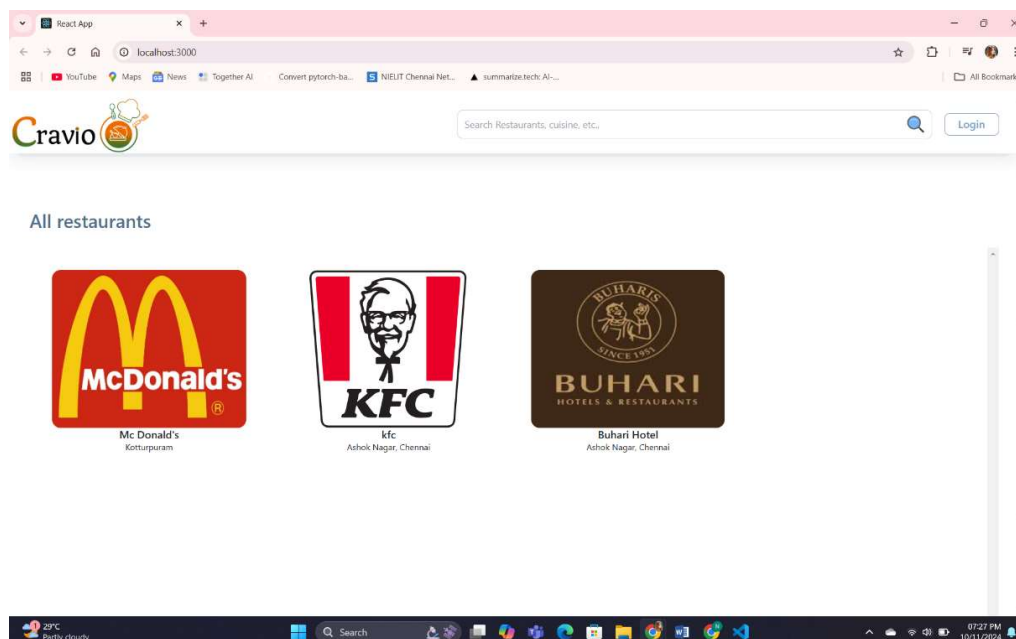
Restaurant Menu

- **Purpose**:
  Shows the menu for a specific restaurant, including details for each food item such as description, price, and availability.
- **Key Elements**:
  - Food item cards with detailed descriptions, images, prices, and add-to-cart buttons.
  - Categories for menu navigation, such as appetizers, main courses, and desserts.

Authentication Page

- **Purpose**:
  Contains forms for user authentication, including login and registration functionalities.
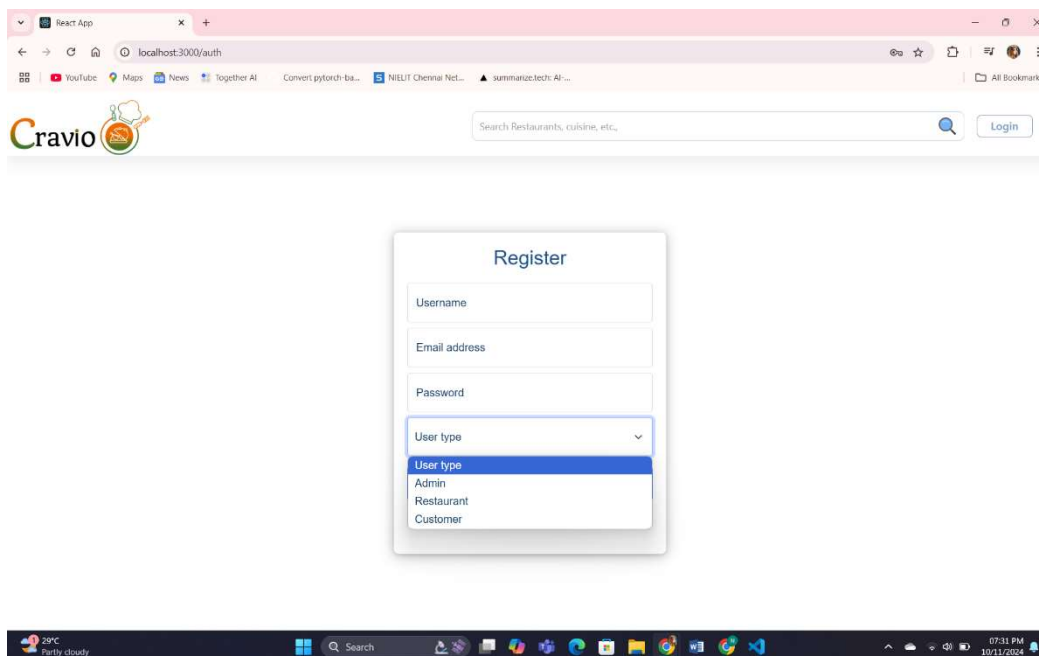
- **Key Elements**:
  - Clean and minimalistic forms for easy input of email, password, and other credentials.
  - Error messages and success alerts for better user feedback.

User Profile

- **Purpose**:
  Displays the user's profile information, order history, and settings for account management.
- **Key Elements**:
  - Editable personal details like name, email, and address.
 - A history tab listing past orders with statuses and total amounts.
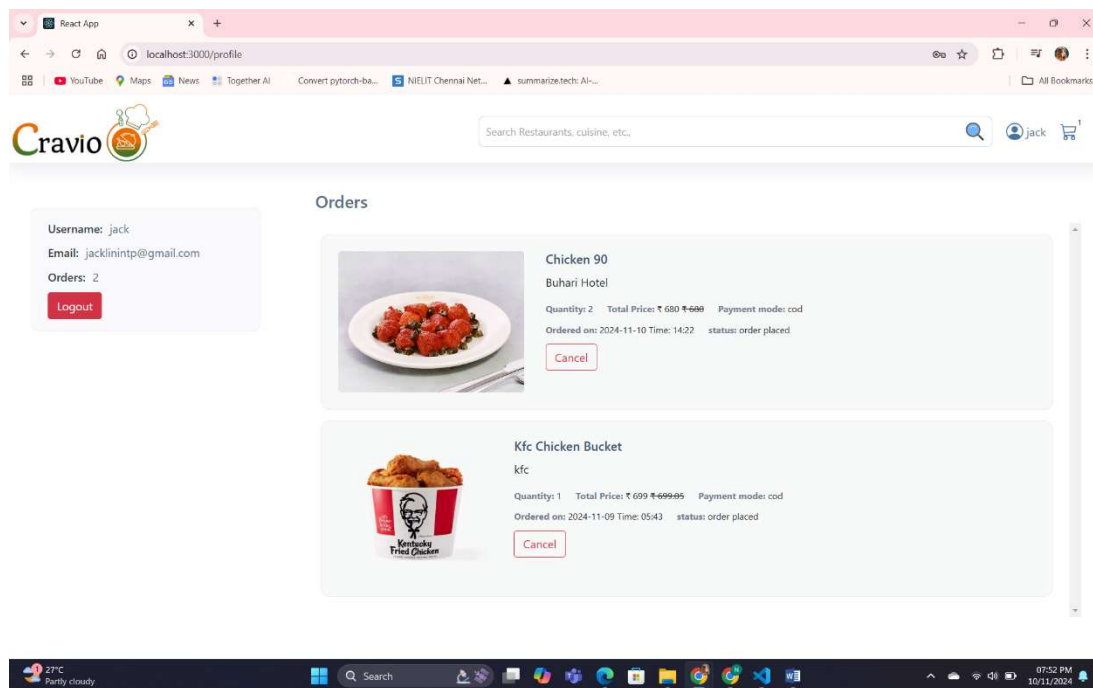
Cart

- **Purpose**:
  Shows items added to the cart, along with options to update quantities, remove items, and proceed to checkout.
- **Key Elements**:
  - A summary of items in the cart, with item-specific prices and overall totals.
  - Checkout button directing users to payment and shipping details.
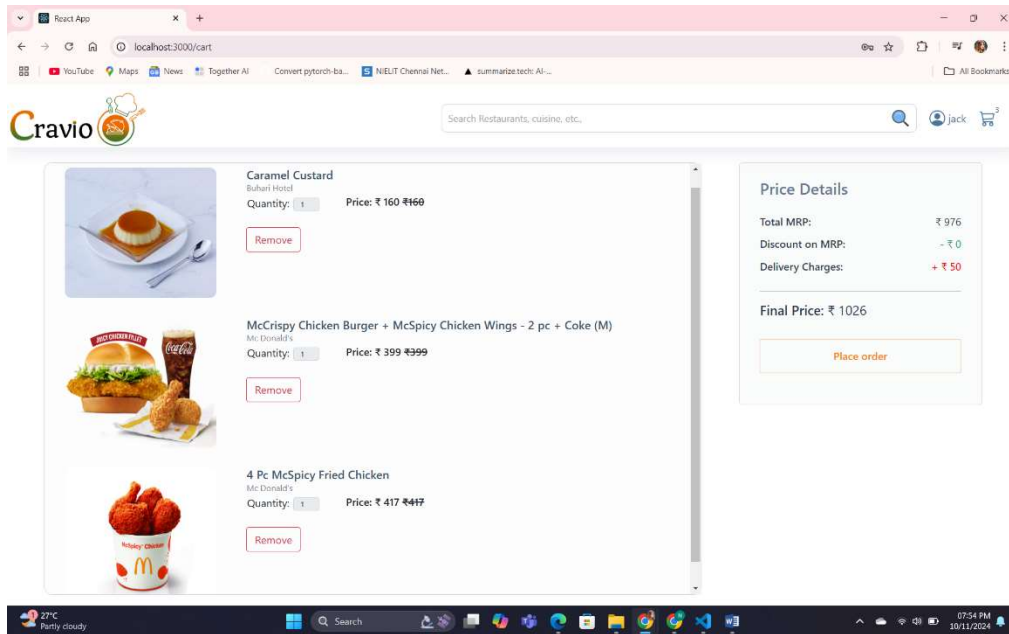
Admin Dashboard

- **Purpose**:
  Main dashboard for the admin user, providing access to all admin-related functionalities and controls.
- **Key Elements**:
  - Graphical metrics showcasing platform activity such as user sign-ups and total orders.
 - Management tools for users, restaurants, and product listings.
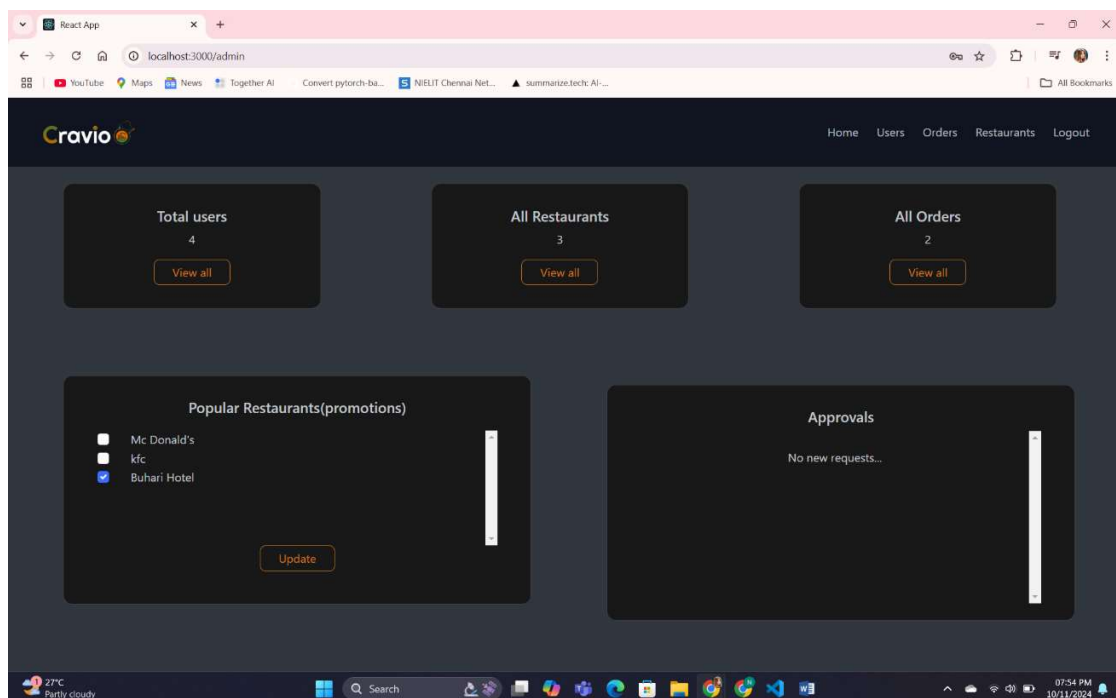
All Orders

- **Purpose**:
  Displays a comprehensive list of all orders placed on the platform, with filtering and sorting options for admin review.
- **Key Elements**:
  - Order ID, customer details, and current status displayed in a table format.
 - Search and filter options to locate specific orders or view orders by status.

All
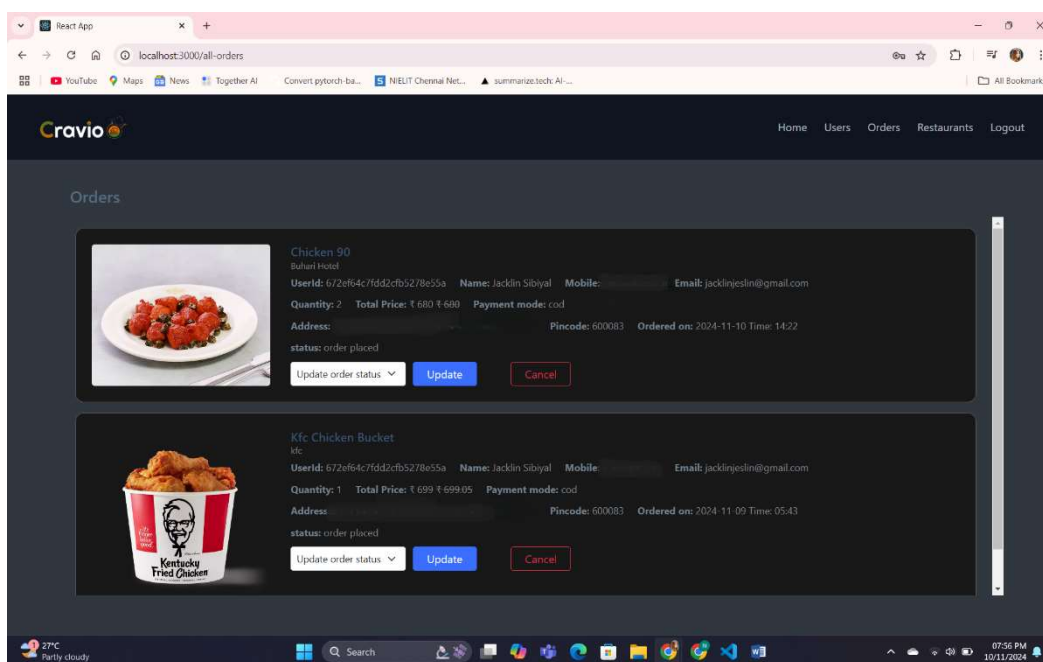
Restaurants

- **Purpose**:
  Lists all registered restaurants on the platform, allowing the admin to manage and review each restaurant's details.
- **Key Elements**:
  - Restaurant profiles with basic details and status indicators.
  - Options to approve or suspend restaurant accounts.
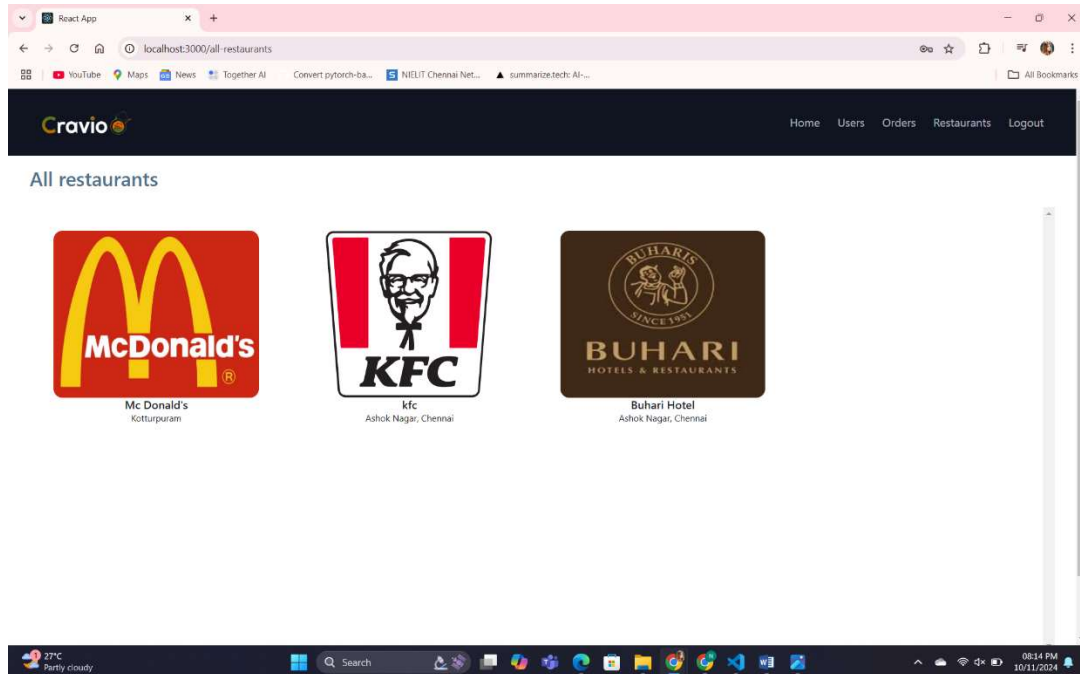
Restaurant Dashboard

- **Purpose**:
  The main dashboard for restaurant owners, allowing them to track orders, manage their menu, and view performance metrics.
- **Key Elements**:
  - A summary of pending and completed orders.
  - Menu management tools to edit, delete, or add new items.
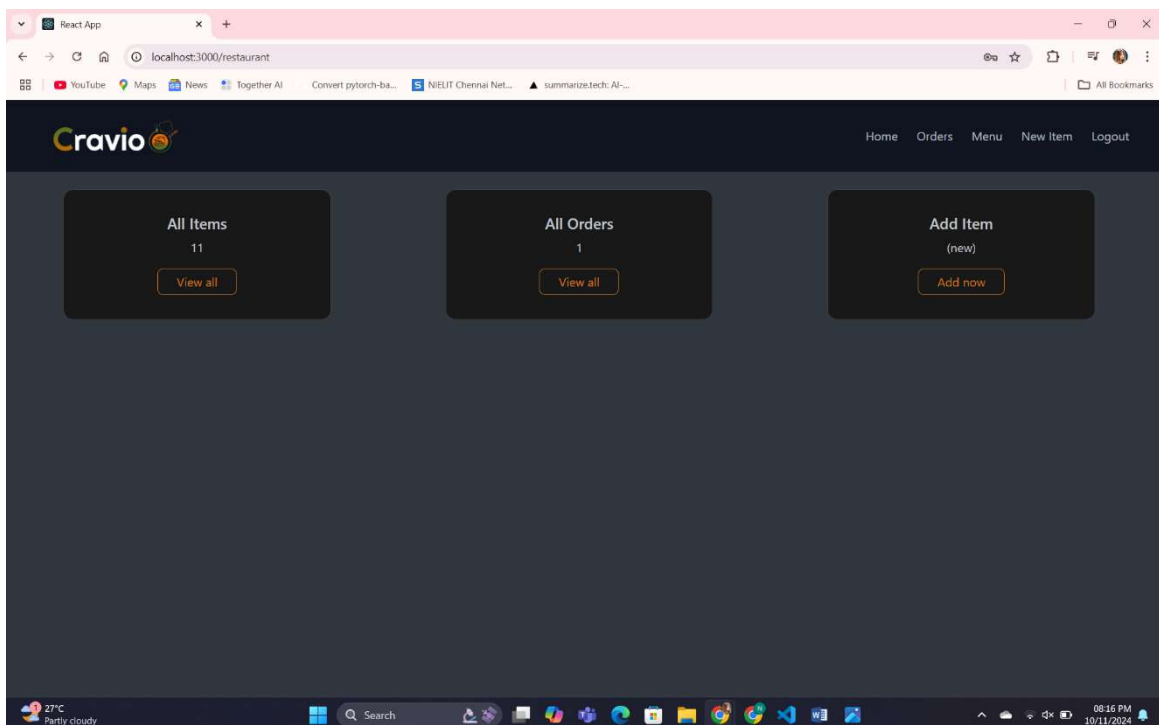
- **Purpose**:
  A form that allows restaurant owners to add new items to their menu, including fields for item name, description, price, and availability.
- **Key Elements**:
  - Upload options for food images.
  - Fields for price, category, and availability toggle.

10. Testing

The Cravio Food Ordering App underwent limited testing to ensure core functionality and basic usability. Below is an overview of the testing scope and plans for future enhancements.

Testing Conducted:

1. **Manual Testing**:
   - Verified key workflows, including user registration, login, and order placement.
   - Checked responsiveness across devices (mobile, tablet, and desktop).
   - Ensured data flow between frontend and backend, such as cart updates and order processing.
2. **API Validation**:

- o Tested critical API endpoints using **Postman** to ensure data retrieval and updates were functioning correctly.
3. **Core Features Tested**:
   - o User authentication and profile updates.
   - o Product browsing, cart management, and order placement.
   - o Basic admin functionalities like user and restaurant approvals.

Future Testing Plans:

- Implement **Jest** for unit testing React components.
- Use **Cypress** for end-to-end testing of user journeys.
- Conduct stress testing to evaluate the app's performance under heavy load.
- Expand cross-browser and device compatibility testing.

11. Screenshots or Demo

Demo Video:

A video demonstration showcasing the Cravio Food Ordering App's features and user interface is available at the following link:

Demo Video Link

What the Demo Includes:

1. **Landing Page Overview**:
   - o Walkthrough of the homepage, including the search bar, featured restaurants, and promotions.
2. **Restaurant Interaction**:
   - o Browsing restaurant menus with details like descriptions, prices, and reviews.
3. **Cart Management**:
   - o Adding items to the cart, updating quantities, and proceeding to checkout.
4. **Order Placement**:
   - o Secure payment process and real-time order tracking.
5. **Admin and Restaurant Dashboards**:
   - o Managing platform activities, menus, and order statuses.

12. Known Issues

The Cravio Food Ordering App is functional but has some known limitations and areas for improvement:

Current Issues:

1. **Limited Payment Integration**:

- o The payment gateway integration supports basic functionality but lacks multiple payment methods and advanced features like refunds or EMI options.

2. **Cross-Browser Compatibility**:
   - o While the app functions well on Google Chrome and Mozilla Firefox, testing for compatibility with Safari, Edge, and other browsers is pending.
3. **Session Timeout**:
   - o There is no automatic session timeout or inactivity detection, which could lead to prolonged access for logged-in users.
4. **Error Handling**:
   - o Some backend routes lack detailed error messages, making it harder for users to understand what went wrong.
5. **Data Validation**:
   - o Input validation for forms (e.g., password strength, email format) could be improved to enhance user experience and data security.

Impact on Users:

- These issues do not prevent the app from functioning but may affect user satisfaction or operational efficiency in certain scenarios.

13. Future Enhancements

The Cravio Food Ordering App has the potential to evolve into a more robust and feature-rich platform. Below are suggested enhancements to address current limitations and introduce new functionalities:

Planned Features:

1. **Enhanced Payment Options**:
   - o Integrate multiple payment gateways and support advanced features like wallet payments, refunds, and installments.
2. **Real-Time Order Updates**:
   - o Implement real-time notifications for order status changes using WebSocket or similar technologies.
3. **Improved Admin Tools**:
   - o Add detailed analytics for order trends, user activity, and restaurant performance to the admin dashboard.
4. **User Recommendations**:
   - o Use AI algorithms to recommend dishes based on user preferences, order history, and popular trends.
5. **Chat Support**:
   - o Integrate real-time chat for customer support and restaurant communication.
6. **Mobile App Development**:
   - o Expand to native mobile applications for Android and iOS to enhance accessibility and convenience.