**Full Stack Development with MERN Project Documentation format**

# 1. Introduction

* **Project Title:** OrderOnTheGo: Your On-Demand Food Ordering Solution
* **Team Members:**
  + Naidu Manasa – Database development
  + Miriyala Ramya Sree – Frontend Developer
  + Maguluri Varsha – Backend Developer
  + Makani Ravi – Project Setup & Configuration, Project Implementation & Execution

# 2. Project Overview

* **Purpose:** The primary purpose of this project is to develop a comprehensive web-based food delivery system. It aims to connect customers with local restaurants, providing a seamless platform for food ordering, delivery tracking, and administrative management. The system enhances convenience for customers and streamlines operations for restaurant owners.
* **Features:**
* **User Management:** Role-based registration and authentication for Customers, Restaurant Owners, and Admins.
* **Customer Features:** Browse restaurants and menus, search/filter options, add items to cart, place orders, track order status, manage profile, submit feedback/ratings.
* **Restaurant Owner Features:** Register and await admin approval, manage restaurant details, add/edit/delete menu items, view and update order statuses, submit feedback to admin.
* **Admin Features:** Approve/reject restaurant owners, manage global food categories, view system dashboard (users, restaurants, orders, food items, pending approvals), manage promoted restaurants, send announcements/feedback, and generate analytical reports (order trends, top restaurants, category popularity, rating distribution).
* **Payment:** Cash on Delivery (COD) and Card payment options.

# 3. Architecture

* **Frontend:** The client-side application responsible for the user interface and interactions.
  + **Technology:** React.js for building dynamic user interfaces, complemented by HTML, CSS, and JavaScript for structural layout, styling, and interactivity.
  + **Description:** Provides intuitive dashboards for customers (ordering, tracking), restaurant owners (menu/order management), and admins (system oversight).
* **Backend:** The server-side application that handles business logic, API requests, and database interactions.
  + **Technology:** Node.js runtime environment with Express.js framework.
  + **Description:** Exposes RESTful API endpoints for all CRUD (Create, Read, Update, Delete) operations related to users, restaurants, food items, orders, and feedback. Handles authentication, authorization, and data processing.
* **Database:** The persistence layer for storing all application data.
  + **Technology:** MongoDB (NoSQL database).
  + **Description:** Stores user accounts, restaurant details, menu items, order history, shopping cart contents, administrative configurations (categories, promoted restaurants), and various feedback types. Mongoose is used as an ODM (Object Data Modelling) library for schema definition and interaction.

# 4. Setup Instructions

* **Prerequisites:**
  + Node.js (LTS version recommended)
  + MongoDB (Community Server or a cloud-based service like MongoDB Atlas)
  + Git (for cloning the repository)
* **Installation:**
  1. **Clone the repository:**
  2. git clone <repository\_url>
  3. cd FOOD\_DELIVERY # Navigate to your project root
  4. **Install Node.js Dependencies:**
  5. npm install
  6. **Set up Environment Variables:** Create or edit the .env file in the project root directory and add:
  7. MONGO\_URI=<Your MongoDB Connection String>
  8. JWT\_SECRET=<A\_Strong\_Random\_Secret\_Key>
  9. NODE\_ENV=development
  10. PORT=5000

# 5. Folder Structure

FOOD\_DELIVERY/

├── config/ # Database connection (db.js), other configurations

│ └── db.js

├── middleware/ # Authentication and authorization middleware

│ ├── auth.js

│ ├── customerMiddleware.js

│ └── restaurantMiddleware.js

├── models/ # Mongoose schemas for data models

│ └── schema.js

├── node\_modules/ # Installed Node.js packages

├── public/ # Static assets accessible directly (e.g., login.html, register.html)

│ ├── login.html

│ └── register.html

├── routes/ # API endpoint definitions grouped by functionality

│ ├── adminRoutes.js

│ ├── apiRoutes.js # General API routes

│ ├── customerRoutes.js

│ └── restaurantRoutes.js

├── views/ # HTML template files for dashboards (served by Express)

│ ├── admin-dashboard.html

│ ├── customer-dashboard.html

│ └── restaurant-dashboard.html

├── .env # Environment variables

├── index.js # Main application entry point

├── package-lock.json # Records exact dependency versions

└── package.json # Project metadata and dependencies

# 6. Running the Application

To start the backend server and serve the HTML files locally:

* **From the project root directory (FOOD\_DELIVERY):**
* node index.js

The server will typically run on http://localhost:5000. You can then access the login/registration pages or specific dashboards directly via your browser.

# 7. API Documentation

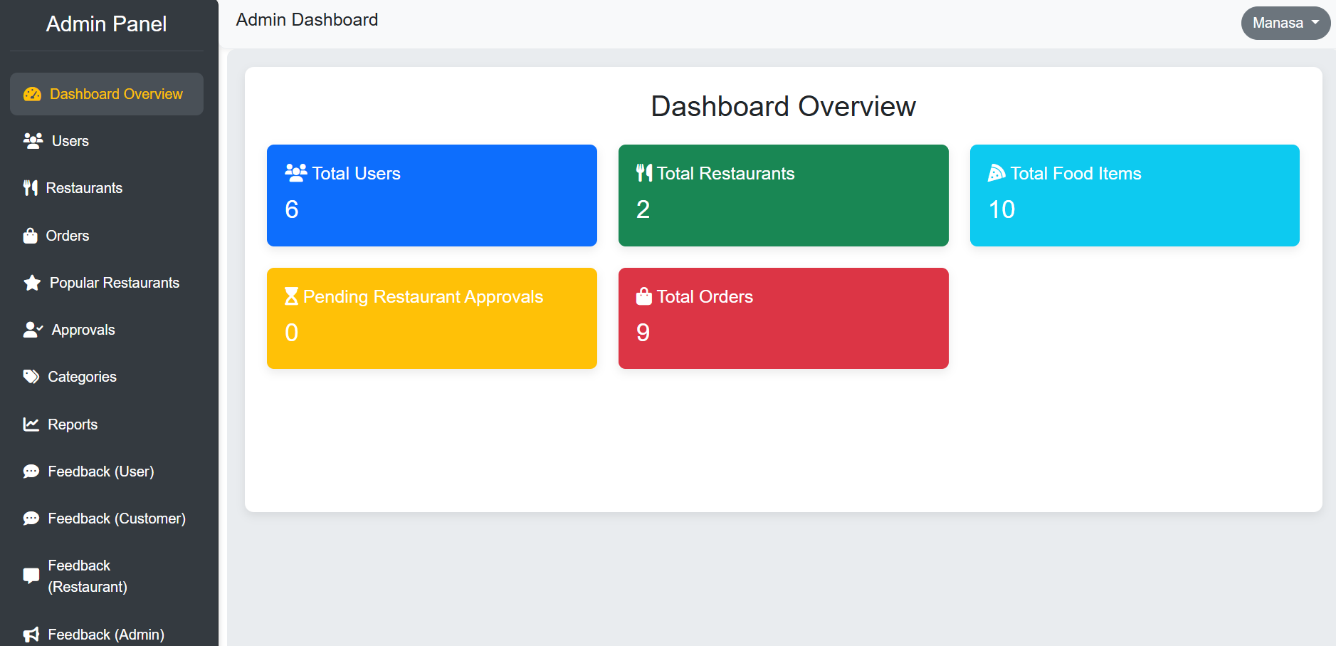
All backend endpoints follow a RESTful convention. Below are examples; a full Postman collection or Swagger documentation would be provided separately.

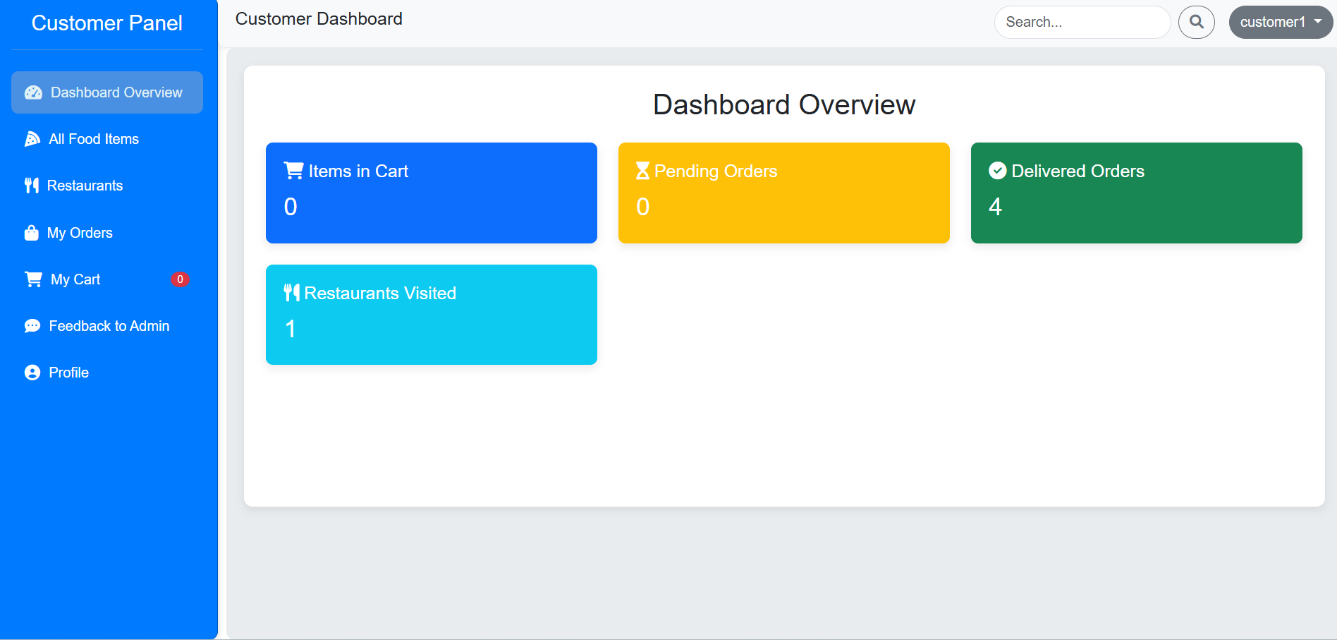
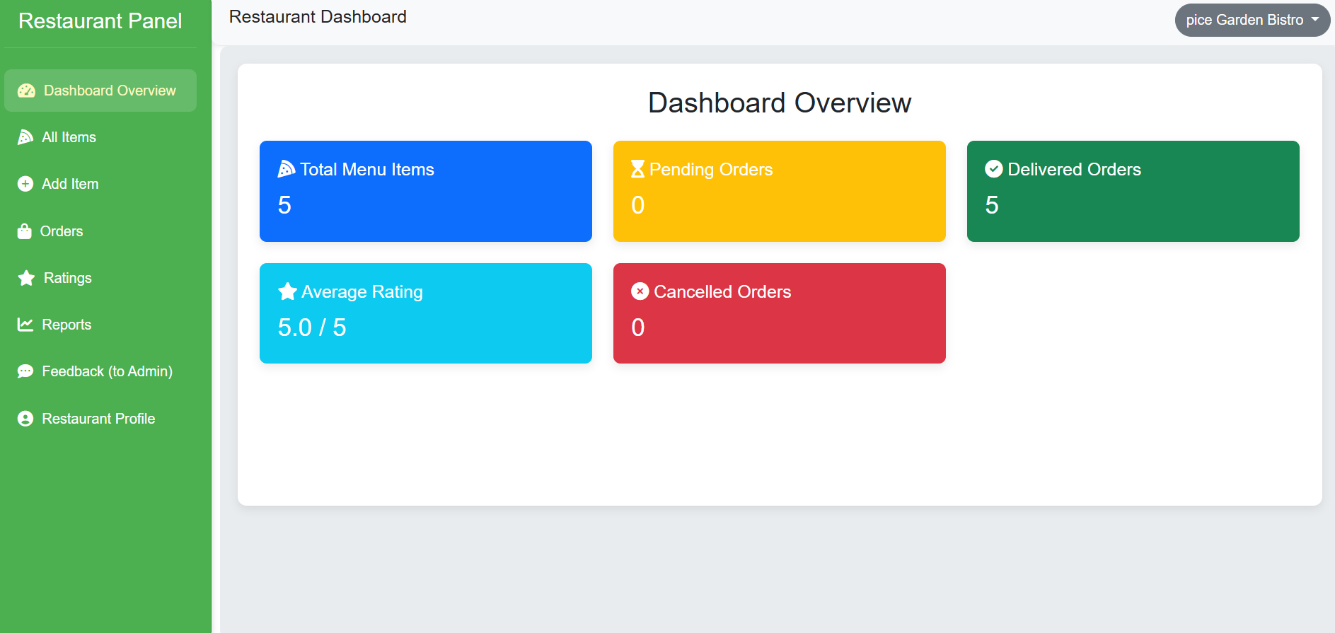
* **User Authentication:**
  + POST /api/auth/register (Register a new user)
  + POST /api/auth/login (Login user and get JWT)
  + GET /api/auth/profile (Get user profile, requires JWT)
  + PUT /api/auth/profile (Update user profile, requires JWT)
  + PUT /api/auth/change-password (Change password, requires JWT)
* **Restaurant Endpoints (Owner & Admin Protected):**
  + POST /api/restaurants (Create new restaurant)
  + GET /api/restaurants/:id (Get single restaurant details)
  + PUT /api/restaurants/:id (Update restaurant details)
  + DELETE /api/restaurants/:id (Delete restaurant - Admin only)
* **Food Item Endpoints (Restaurant Owner Protected):**
  + POST /api/fooditems (Add new food item)
  + PUT /api/fooditems/:id (Update food item)
  + DELETE /api/fooditems/:id (Delete food item)
* **Order Endpoints:**
  + POST /api/orders (Place a new order - Customer protected)
  + GET /api/orders/myorders (Get customer's orders - Customer protected)
  + GET /api/orders/restaurant (Get restaurant's orders - Restaurant owner protected)
  + PUT /api/orders/:id/status (Update order status - Restaurant owner/Admin protected)
* **Admin Specific Endpoints:**
  + GET /api/admin/dashboard-counts
  + GET /api/admin/users
  + PUT /api/admin/users/:id/approve
  + POST /api/admin/categories
  + GET /api/admin/reports/metrics

# 8. Authentication

* Authentication is handled using **JSON Web Tokens (JWT)**.
* Upon successful login, the backend generates a JWT containing the user's ID and userType (admin, restaurant, customer).
* This token is sent back to the client (JavaScript in HTML files) and stored (e.g., in localStorage).
* For all protected routes, the client-side JavaScript includes the JWT in the Authorization header (Bearer YOUR\_TOKEN).
* A backend middleware (middleware/auth.js - protect function) verifies the token's authenticity and expiration. If valid, the user's data (including userType) is attached to req.user.
* Another middleware (middleware/auth.js - authorize function), along with customerMiddleware.js and restaurantMiddleware.js, checks req.user.userType against a required role to ensure role-based access control.
* Passwords are securely stored as hashed values using bcrypt.js.

1. **User Interface**

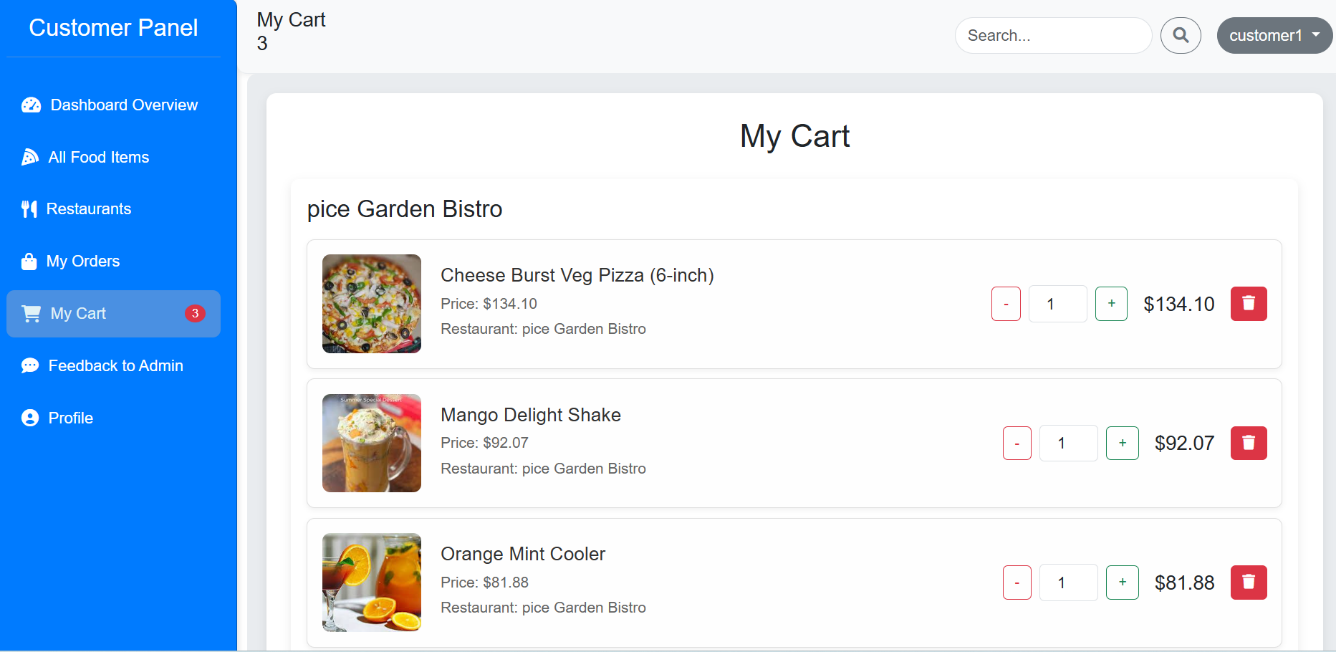
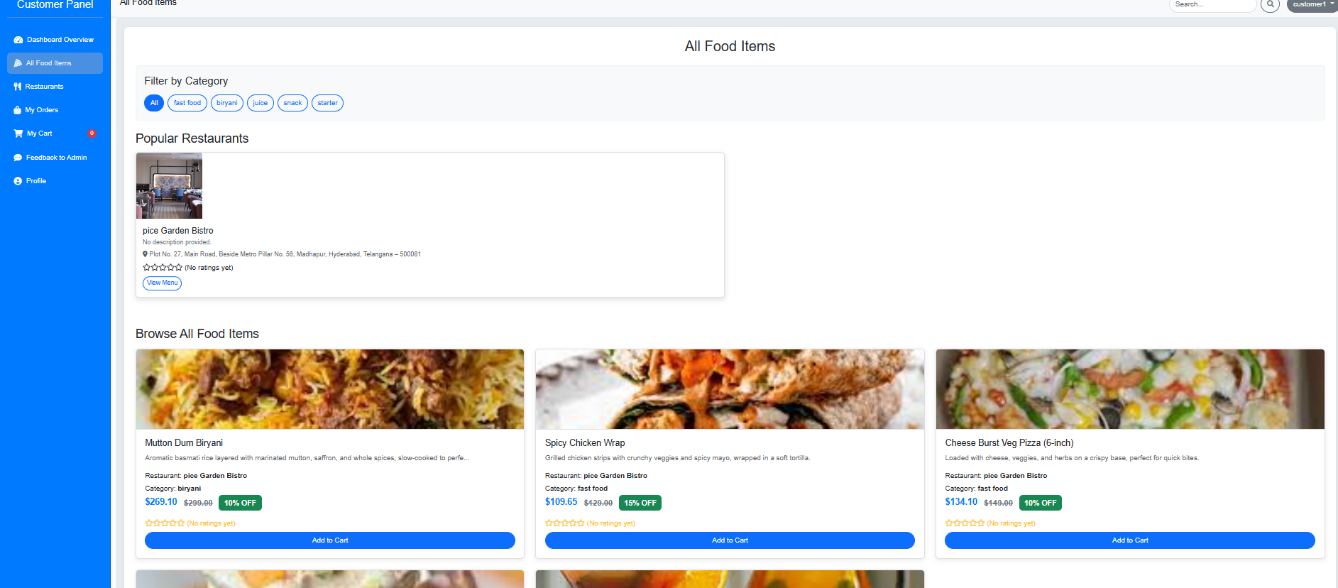
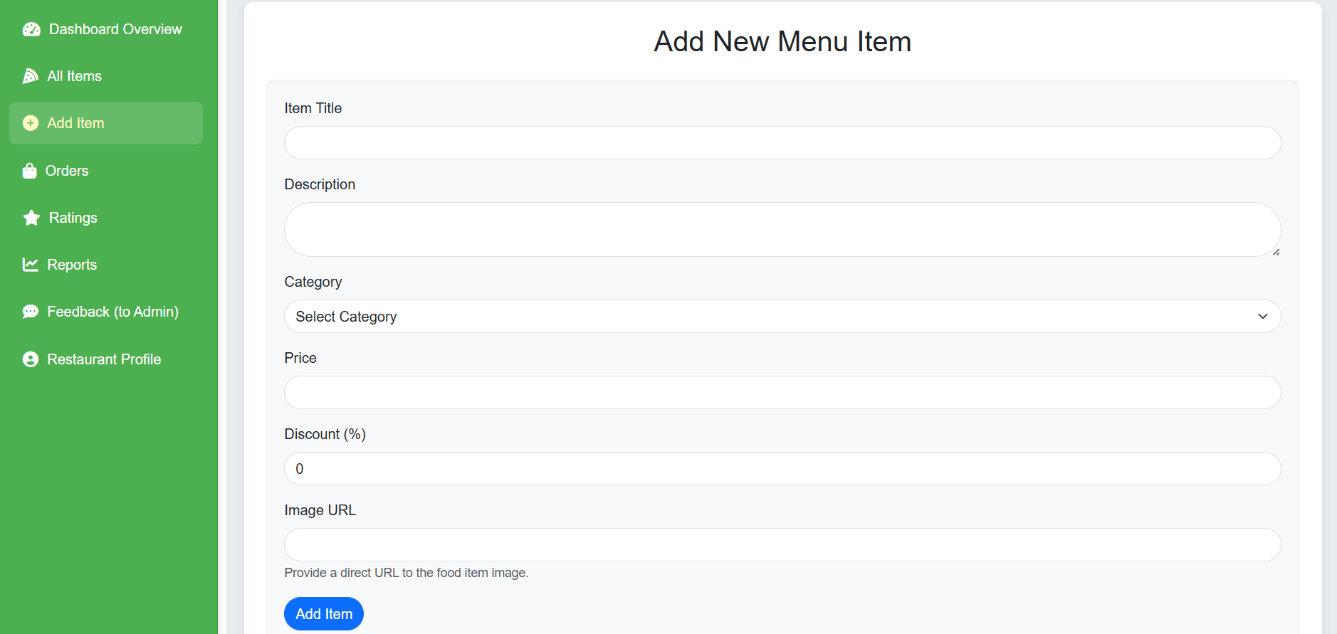
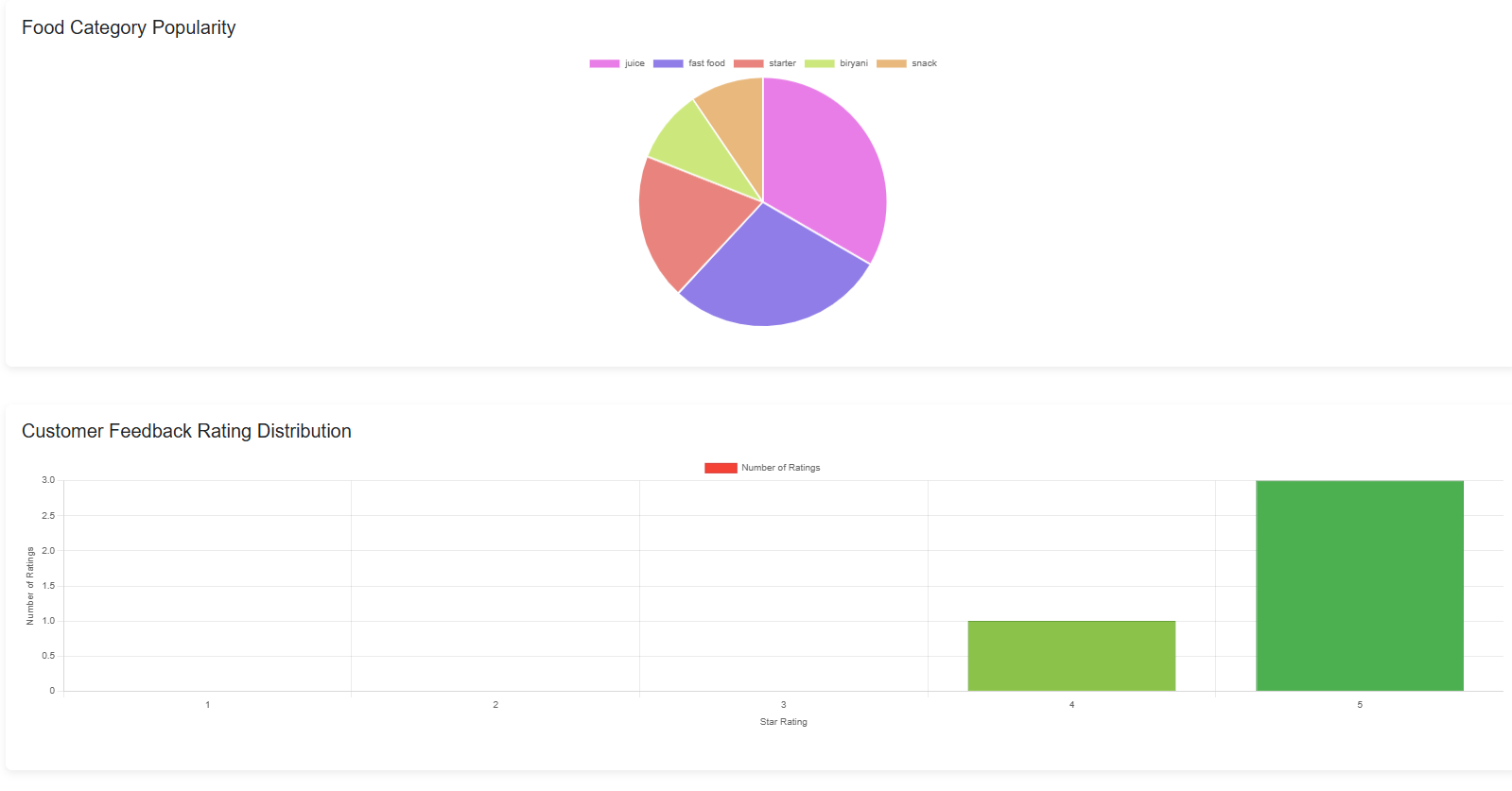
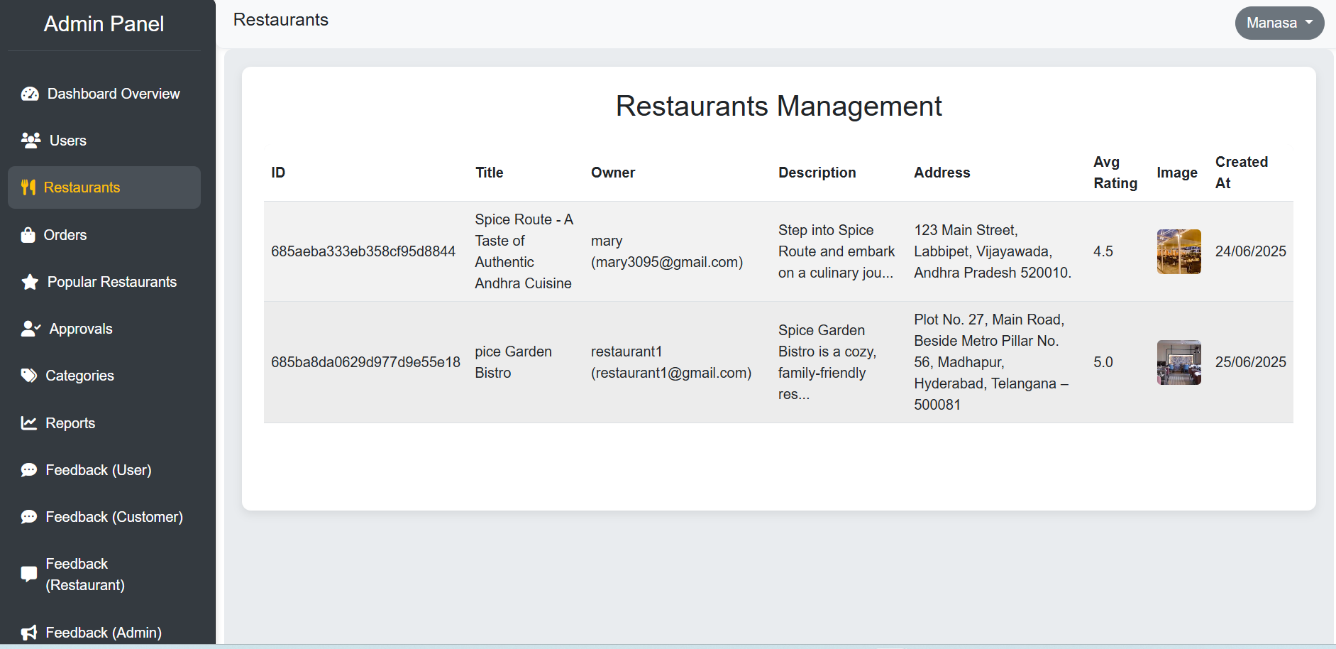
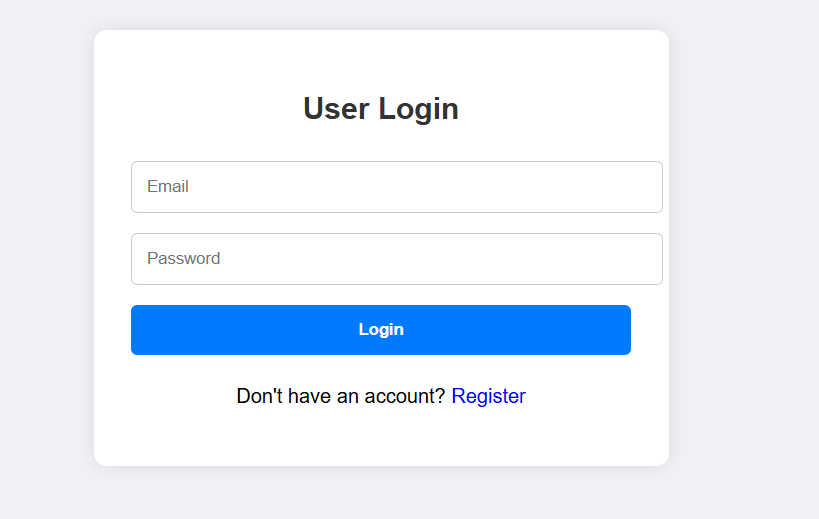
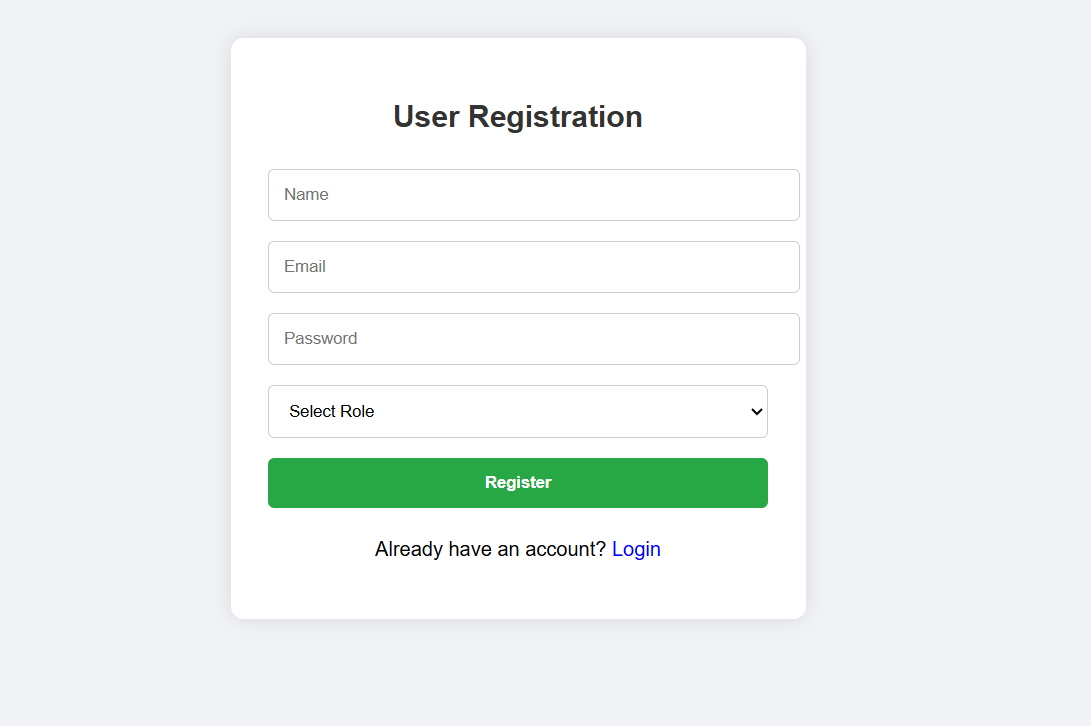




1. **Testing**

* **Testing Strategy:** The project employed a combination of manual testing and unit testing (where applicable) to ensure functionality and robustness. User Acceptance Testing (UAT) was conducted using a structured template to validate features against user stories.
* **Tools Used:** Postman for API testing, browser developer tools for frontend debugging, and potentially unit testing frameworks like Jest (if implemented for specific modules).

1. **Screenshots or Demo**



1. **Known Issues**

* Minor UI inconsistencies on specific mobile resolutions.
* (Any other non-critical bugs discovered during UAT that are pending fix, e.g., "Edge case where deleting a restaurant fails if related feedback exists without a valid user link.")

# 13. Future Enhancements

* **Advanced Search & Filtering:** Implement more robust search capabilities (e.g., by cuisine, dietary restrictions, delivery time).
* **Real-time Delivery Tracking:** Integrate with a mapping service (e.g., Google Maps API) for live tracking of delivery drivers.
* **Notifications:** Implement real-time notifications for order updates (e.g., push notifications or websockets).
* **Online Payment Integration:** Expand payment options beyond COD, potentially integrating with services like Stripe or PayPal for direct card processing.
* **User Reviews for Food Items:** Allow customers to rate individual food items in addition to restaurants.
* **Promotional Campaigns:** Allow admins and restaurant owners to create and manage dynamic discount campaigns.
* **Performance Monitoring:** Implement dedicated logging and monitoring tools (e.g., Prometheus, Grafana) for better system health insights.
* **Container Orchestration:** Deploy with Kubernetes for advanced scaling and management in a production environment.