

A dark blue vertical bar runs down the left side of the page. A blue arrow points to the right from the bar, containing the date.

6/20/2025

LAB FINAL

SSDD

AYESHA SIDDIQA

SP22-BCT-008

BCT-7

SIR FAISAL SHAHZAD

SECURE SOFTWARE DESIGN AND DEVELOPMENT

Several thin, curved lines in shades of blue and grey originate from the bottom left corner and sweep upwards and to the right, creating a dynamic, abstract design element.

Contents

Secure Food Delivery System	2
1. Project Overview	2
2. Core Features & Functionality	2
2.1 Authentication & Role-Based Authorization	2
2.2 Role-Based Authorization.....	2
2.3 Restaurant Management Checks applied for restaurant are as follows:.....	3
2.4 Order Management	3
2.5 Payments	4
<input type="checkbox"/> User Functionality:.....	4
<input type="checkbox"/> Admin Functionality:	4
<input type="checkbox"/> Backend Enhancements:	4
2.6 User Management	4
2.7 Access Control and Token	5
3. JWT Authentication: Flow & Protection	5
Why we used JWT Instead of Sessions?.....	5
The following reason make use of JWT a better choice than Sessions:	5
Token Security	7
4. Monitoring & Observability.....	7
5. Project Structure.....	8
6. UI	9
7. Conclusion	19

Secure Food Delivery System

Powered by FastAPI, Streamlit, and JWT Authentication

1. Project Overview

This project is a full-stack Food Delivery System developed with a strong focus on security, modular architecture, and role-based access control. The backend is built using FastAPI, the frontend with Streamlit, and JWT is used to ensure secure, stateless authentication.

Admin and User roles are managed distinctly, offering controlled access to sensitive resources like user data, order management, and restaurant operations. Monitoring is future-proofed using Prometheus and Grafana for container-level metrics.

2. Core Features & Functionality

2.1 Authentication & Role-Based Authorization

User Registration:

Firstly the User registers himself keeping in view the following protocols:

- Validates proper email structure.
- Password strength: min 8 characters, includes uppercase, lowercase, number, and special character.
- Username limited to 30 characters.

User Login:

Then using the username and password user registered, fills in Login form and gets successful login after passing the valid credential check.

- Secure credential check.
- Returns a JWT token with encoded user_type (admin/user).
- Token is saved in st.session_state in Streamlit.

2.2 Role-Based Authorization

After successful login, in register the user registers as either admin or user i.e as a customer. This field user_type defines the role and on this basis, it is decided what operations are available for the user. Following below is the access control provided on basis of the user_type.

Access Control

- **Admin:**
 - ❖ Manage users (view, update, delete).
 - ❖ Full CRUD on restaurants.
 - ❖ View and delete all orders.
- **User:**
 - ❖ Can browse restaurants.
 - ❖ Place and track their own orders only.

2.3 Restaurant Management

Checks applied for restaurant are as follows:

- **Admin Capabilities**
 - ❖ Create, update, and delete restaurants.
 - ❖ Validates phone format (e.g., 03XXXXXXXXX for Pakistan).
- **User Access**
 - ❖ Read-only access to restaurant list.

2.4 Order Management

- **User Functionality**
 - ❖ Select restaurant.
 - ❖ Enter food item, quantity, address.
 - ❖ Set order & payment status.
 - ❖ Order is auto-linked with the user's id and username.
- **Admin Features**
 - ❖ View all orders with full metadata.
 - ❖ Delete any order.

- **Security Enhancements**

- ❖ JWT authentication on all endpoints.
- ❖ Orders protected: /orders/ GET is admin-only.
- ❖ ordered_by stored and retrieved from DB for clear traceability.

2.5 Payments

The payment system is fully implemented and functional with clear separation of user/admin roles.

- **User Functionality:**

- ❖ **A user can create a payment** after placing an order.
- ❖ Payment form includes:
 - order_id
 - amount
 - payment_method (e.g., cash, credit card)
 - payment_status (e.g., Paid, Unpaid)
 - Optional: transaction_id

- **Admin Functionality:**

- ❖ **Admin can view all payments**, including:
 - User details (via relationship to order)
 - Payment status and metadata
- ❖ This is useful for reconciling transactions and auditing order-payment linkage.

- **Backend Enhancements:**

- ❖ Payments table is linked via ForeignKey(order_id) to the Orders table.
- ❖ All endpoints are protected with JWT authentication.
- ❖ Role-based access is enforced:
 - Users can only create payments
 - Admins can only view payments

2.6 User Management

- **Admin View**

- ❖ View all users.
- ❖ Update and delete capabilities.

- **Users**

- ❖ Cannot view or modify other users.
- ❖ Can manage their own session via frontend.

2.7 Access Control and Token

Endpoint	Token Required?	Access
/auth/login, /register	No	Everyone
/orders/ (POST)	Yes	User
/orders/ (GET)	Yes	Admin only
/restaurants (CRUD)	Yes	Admin only
/restaurants (GET)	Yes	All users
/users (GET/DELETE)	Yes	Admin only
/payments/ (POST)	Yes	User
/payments/ (GET)	Yes	Admin only

3. JWT Authentication: Flow & Protection

JWT (JSON Web Token) is a compact, URL-safe means of representing claims between two parties. In this project, JWT is used to secure the backend by verifying the identity of users making API requests without maintaining server-side session storage.

It ensures that only authenticated and authorized users (admins or regular users) can access protected endpoints in the system.

Why we used JWT Instead of Sessions?

The following reasons make use of JWT a better choice than Sessions:

- **Stateless:** No session data is stored on the server.
- **Scalable:** Ideal for distributed systems or containerized deployments.
- **Secure:** Tokens are cryptographically signed to prevent tampering.
- **Portable:** Can be passed easily in HTTP headers, making API integration simple.

JWT Authentication Workflow

1. User Logs In

- The user sends a POST request to /auth/login with their username and password.
- FastAPI verifies the credentials.
- If valid, a JWT access token is created, containing:
 - username

- user_id
- user_type (admin/user)
- exp (expiry timestamp)

The token is returned to the frontend.

2. Token Storage in Frontend (Streamlit)

- The token is securely stored in:

```
st.session_state.access_token  
st.session_state.user_type
```

- This state is preserved during the session and used for all API calls.

3. Token Sent with Each Request

- All protected endpoints in FastAPI expect a valid token in the **Authorization header**:

Authorization: Bearer <JWT_ACCESS_TOKEN>

- In Streamlit:

```
headers = {  
  "Authorization": f"Bearer {st.session_state.access_token}"  
}
```

4. Token Decoding & Verification (FastAPI)

- The backend uses Depends(get_current_user) on protected routes.
- This uses a JWT middleware (custom or FastAPI dependency) to:
 - Check if the token is valid.
 - Decode the token using the server's secret key.
 - Extract the user details and permissions from the payload.

If the token is invalid, expired, or missing, a 401 Unauthorized response is returned.

5. Role-Based Access Enforcement

- After decoding, the backend extracts:

```
{
  "username": "ayesha",
  "user_type": "admin",
  "user_id": 1
}
```

- Based on user_type, access is conditionally allowed:
 - Admins can access routes like /users, /restaurants, and /orders (view/delete).
 - Users can only access routes like /orders/ (POST) or /payments/.

Token Security

- JWTs are signed using HS256 (HMAC + SHA-256).
- Expiry (exp) is enforced to prevent token reuse or replay attacks.
- Passwords are securely hashed before JWT creation using passlib with bcrypt.

Security Benefit	How It's Achieved
Authentication	Via token payload (username, user_type)
Authorization	Role-based routing logic
Stateless Security	No server sessions; token-driven access
Reusability	Token used across requests
Protection from Tampering	JWTs are signed with a secret key

4. Monitoring & Observability

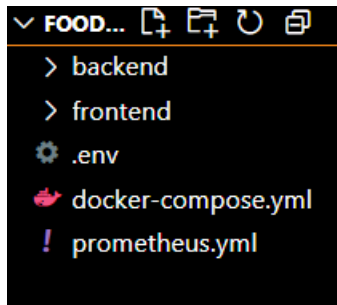
For this purpose, we shall be integrating Prometheus and Grafana along. The will:

- **Prometheus** scrapes backend /metrics endpoint every 15s.
- **Grafana** dashboards to visualize:
 - ❖ Request latency
 - ❖ Active users
 - ❖ Endpoint hit counts

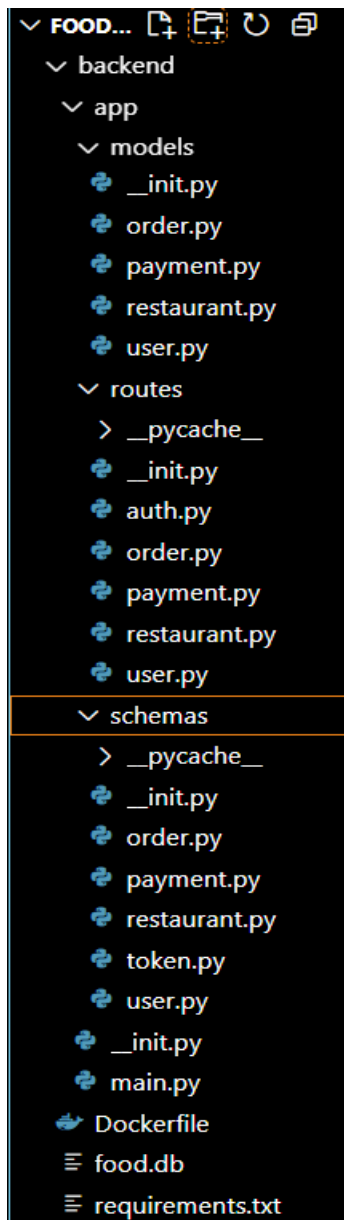
This ensures real-time health tracking and performance tuning for production readiness.

5. Project Structure

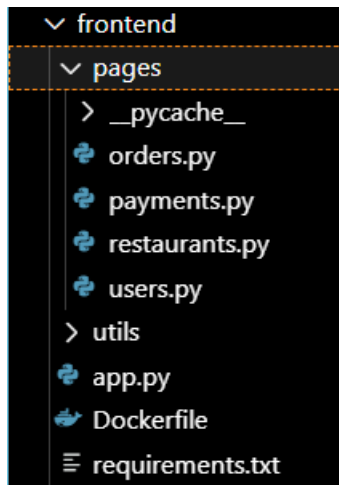
Overall Structure



Backend:



Frontend

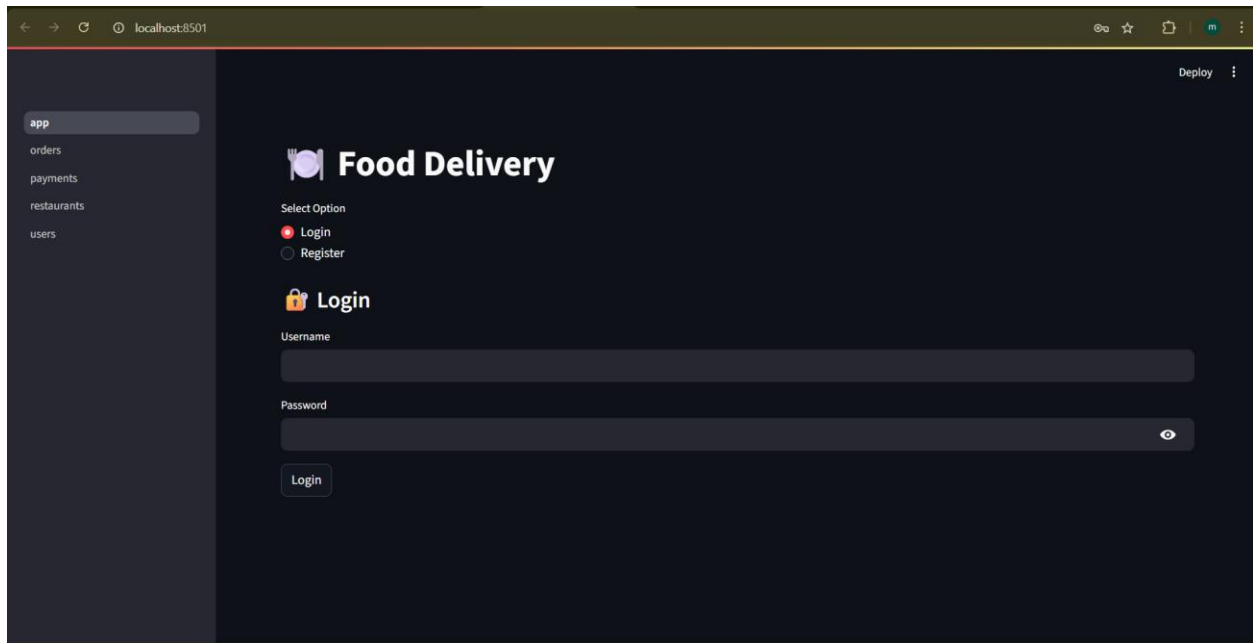


6. UI

Register:

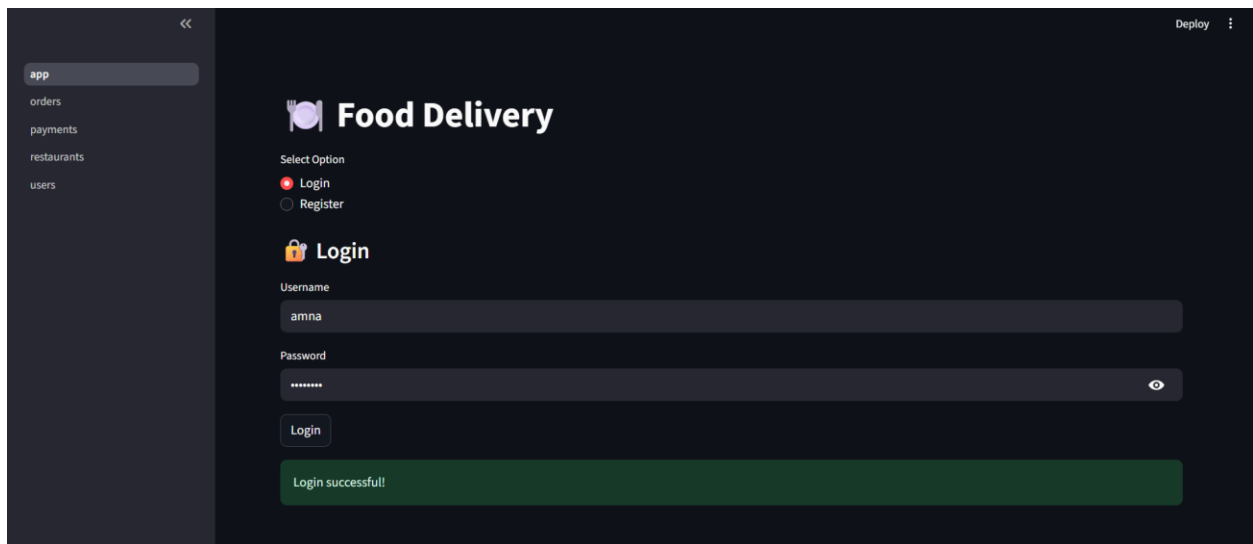
Making an admin account.

The screenshot shows the 'Food Delivery' app interface. On the left is a sidebar with links: 'app', 'orders', 'payments', 'restaurants', and 'users'. The main content area has a 'Select Option' section with 'Login' and 'Register' (selected). Below is the 'Register' form with fields for 'Username' (Ayesha), 'Password' (masked), 'Email' (asiiijj@gmail.com), and 'User Type' (admin). A 'Register' button is at the bottom. A green success message at the bottom states 'Registered successfully'.

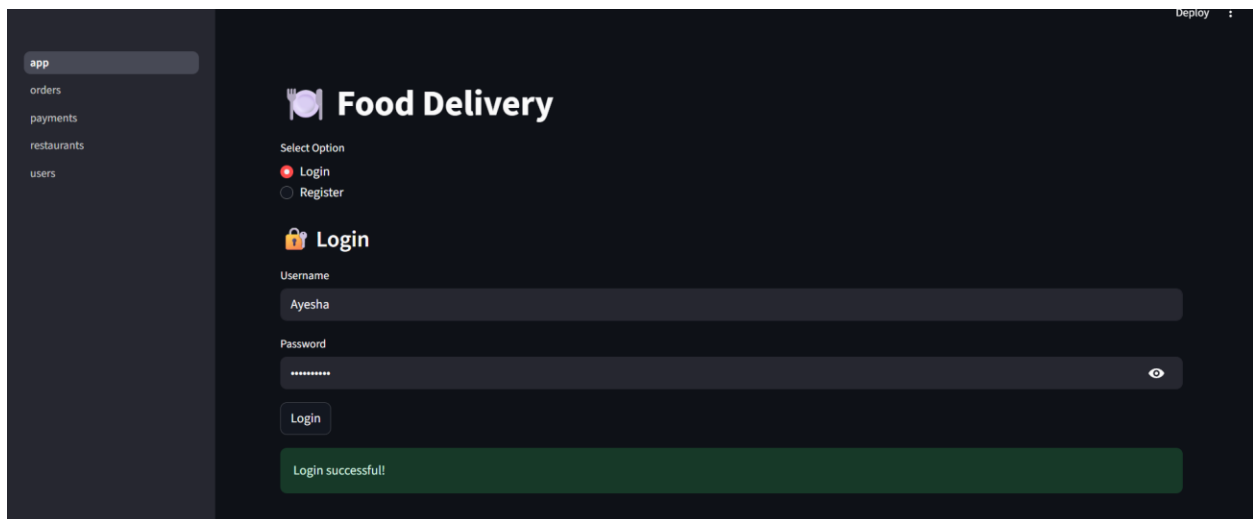


Login as User:

Amna is a registered user. By entering valid credentials we can successfully login.



Login as Admin:



Food Delivery

Select Option

☒ Login

☐ Register

Login

Username

Ayesha

Password

Login

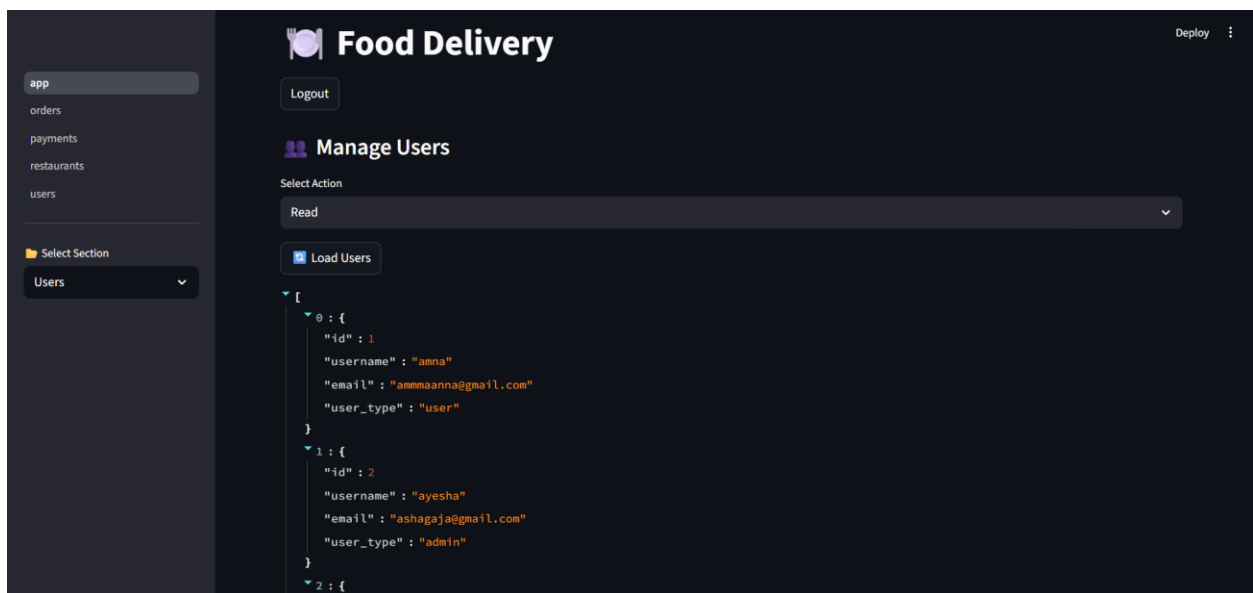
Login successful!

USERS:

In Admin Portal:

a. Read Users:

By clicking on Load User we get all the users registered in database.



Food Delivery

Logout

Manage Users

Select Action

Read

Load Users

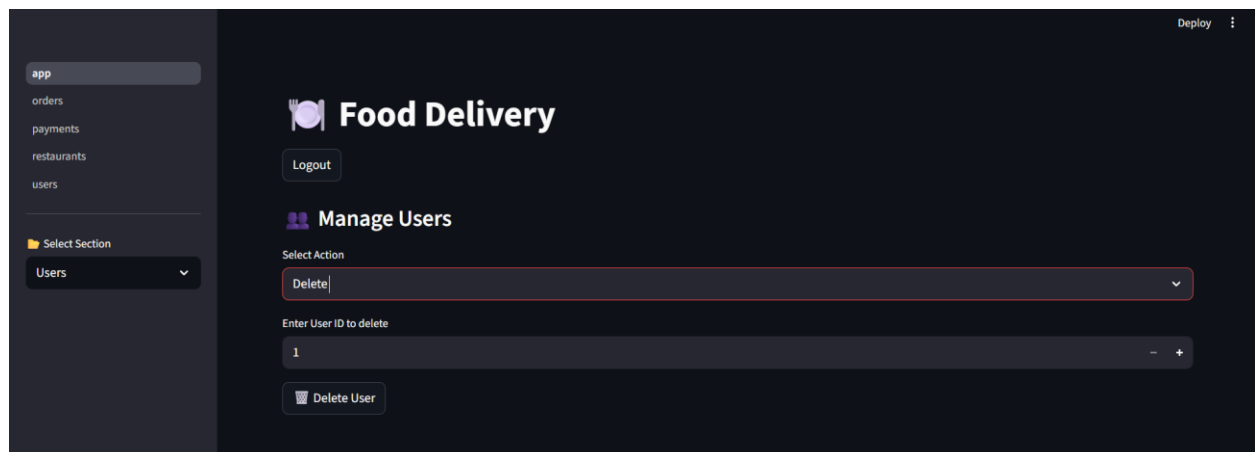
```
[
  {
    "id": 1,
    "username": "amna",
    "email": "ammaanna@gmail.com",
    "user_type": "user"
  },
  {
    "id": 2,
    "username": "ayesha",
    "email": "ashaga@gmail.com",
    "user_type": "admin"
  },
  {
    "id": 3,
    "username": "ayesha",
    "email": "ashaga@gmail.com",
    "user_type": "admin"
  }
]
```

- b. Update User:**
Admin has right to update the registered users.



The screenshot shows the 'Food Delivery' app interface. At the top, there is a logo with a fork and knife icon and the text 'Food Delivery'. A 'Logout' button is in the top right corner. Below the logo, there is a 'Manage Users' section with a user icon. Under 'Manage Users', there is a 'Select Action' dropdown menu with 'Update' selected. Below this, there is a text input field labeled 'Enter User ID to update' with the value '1'. There are also input fields for 'New Username' and 'New Email'. A 'User Type' dropdown menu is set to 'admin'. At the bottom, there is a green button with a checkmark icon and the text 'Update User'.

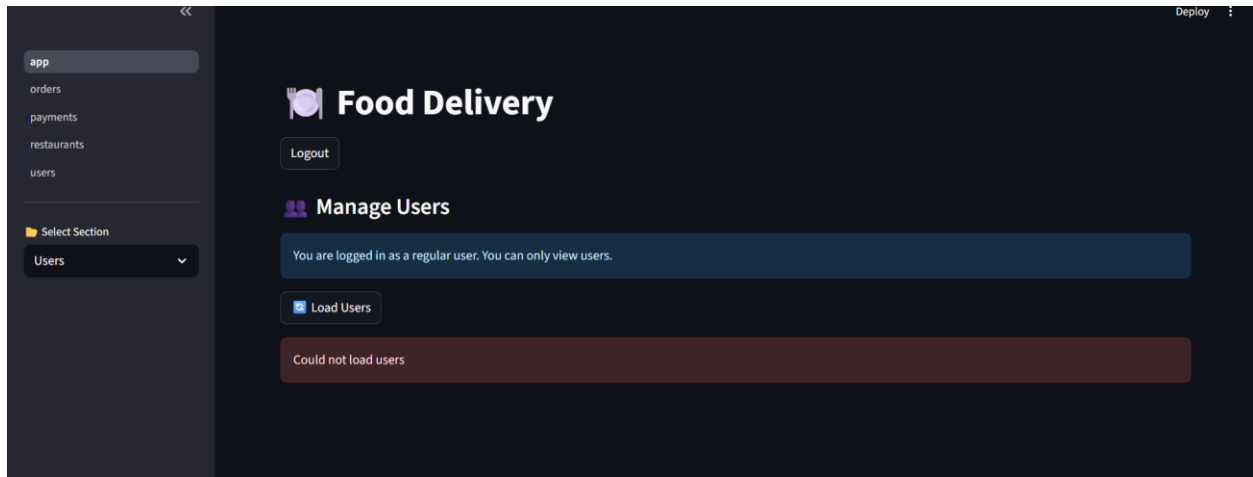
- c. Delete User:**
Admin can delete any user from database by entering the user id.



The screenshot shows the 'Food Delivery' app interface. On the left, there is a sidebar with a list of sections: 'app', 'orders', 'payments', 'restaurants', and 'users'. Below this list is a 'Select Section' dropdown menu with 'Users' selected. The main content area shows the 'Food Delivery' logo and a 'Logout' button. Below the logo, there is a 'Manage Users' section with a user icon. Under 'Manage Users', there is a 'Select Action' dropdown menu with 'Delete' selected. Below this, there is a text input field labeled 'Enter User ID to delete' with the value '1'. At the bottom, there is a button with a trash can icon and the text 'Delete User'.

In User Portal:

User can not access the information of registered users.



RESTAURANT:

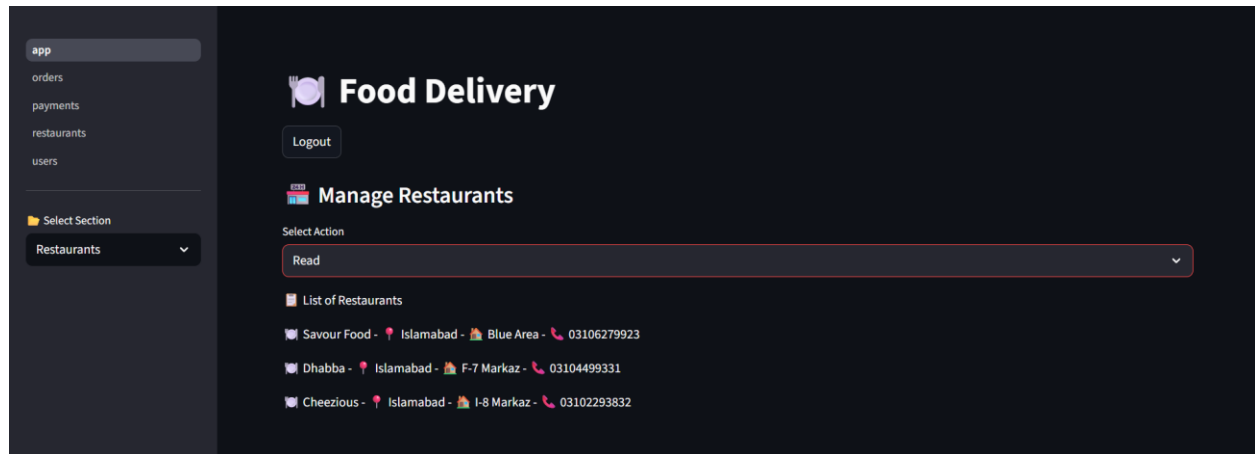
In Admin Portal:

a. Create:

Registering a new restaurant.



- b. View:**
Admin can view all registered Restaurant.

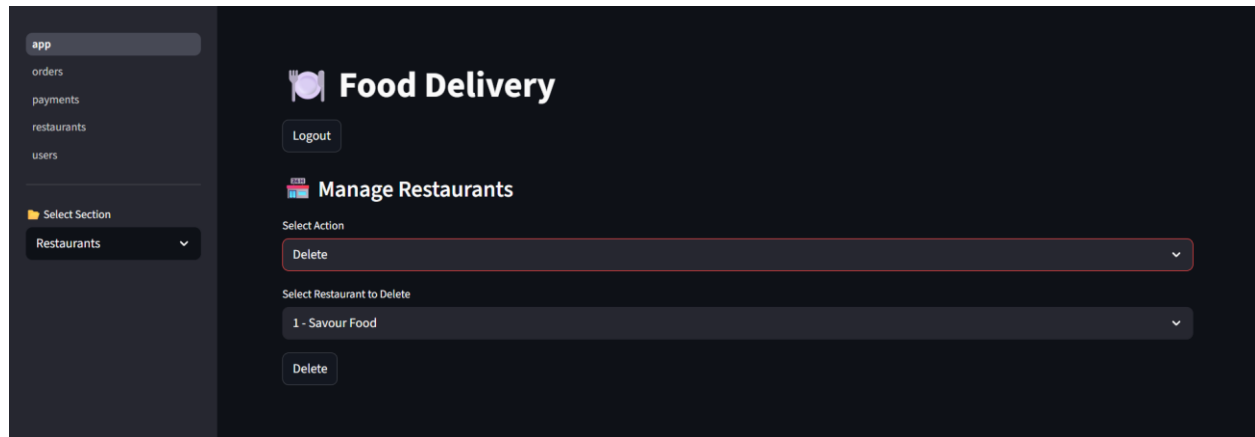


- c. Update:**
Admin can update the information of registered restaurants.



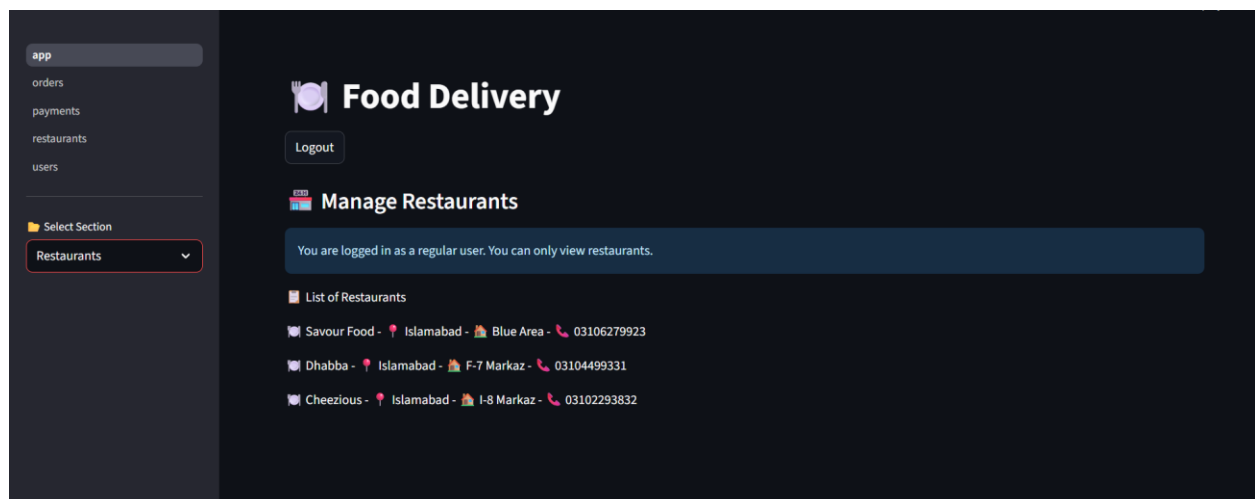
d. Delete:

Admin has right to delete.



In User Portal:

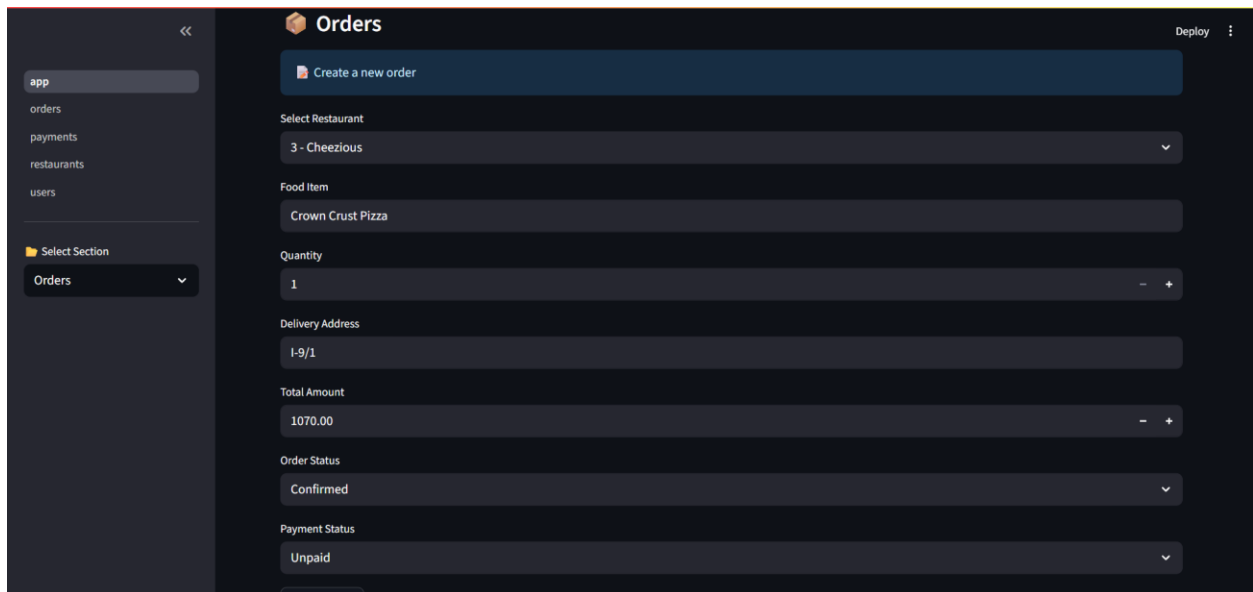
User can only view registered restaurants in the database.



ORDER:

In Admin Portal:

In User Portal:



The screenshot shows the 'Orders' form in the User Portal. The left sidebar contains a menu with 'app', 'orders', 'payments', 'restaurants', and 'users'. The 'Orders' section is selected. The main content area has a 'Deploy' button and a 'Create a new order' button. Below these are several input fields: 'Select Restaurant' (3 - Cheezious), 'Food Item' (Crown Crust Pizza), 'Quantity' (1), 'Delivery Address' (I-9/1), 'Total Amount' (1070.00), 'Order Status' (Confirmed), and 'Payment Status' (Unpaid).

Orders

Create a new order

Select Restaurant

3 - Cheezious

Food Item

Crown Crust Pizza

Quantity

1

Delivery Address

I-9/1

Total Amount

1070.00

Order Status

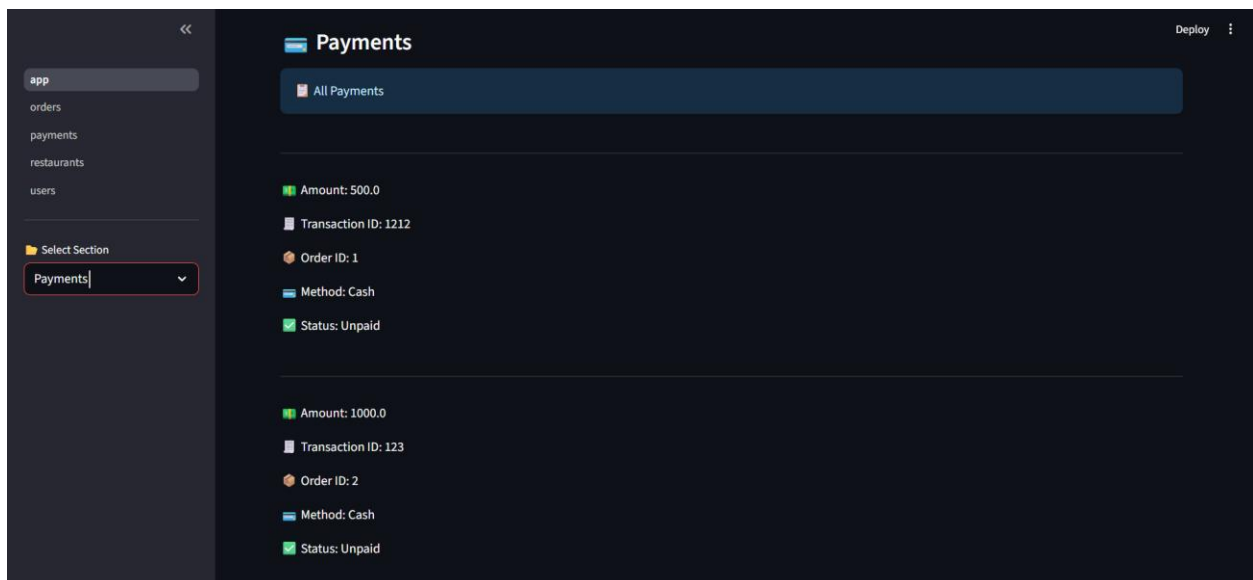
Confirmed

Payment Status

Unpaid

PAYMENT:

In Admin Portal:



The screenshot shows the 'Payments' list in the Admin Portal. The left sidebar contains a menu with 'app', 'orders', 'payments', 'restaurants', and 'users'. The 'Payments' section is selected. The main content area has a 'Deploy' button and an 'All Payments' button. Below these are two payment entries. Each entry shows the amount, transaction ID, order ID, method, and status.

Payments

All Payments

Amount: 500.0

Transaction ID: 1212

Order ID: 1

Method: Cash

Status: Unpaid

Amount: 1000.0

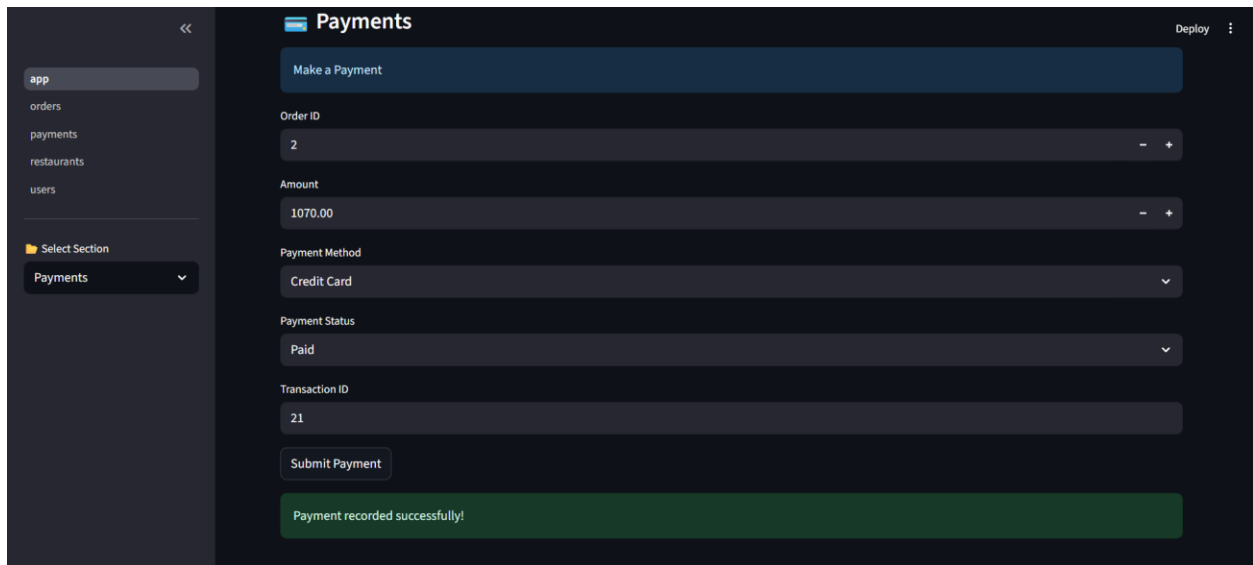
Transaction ID: 123

Order ID: 2

Method: Cash

Status: Unpaid

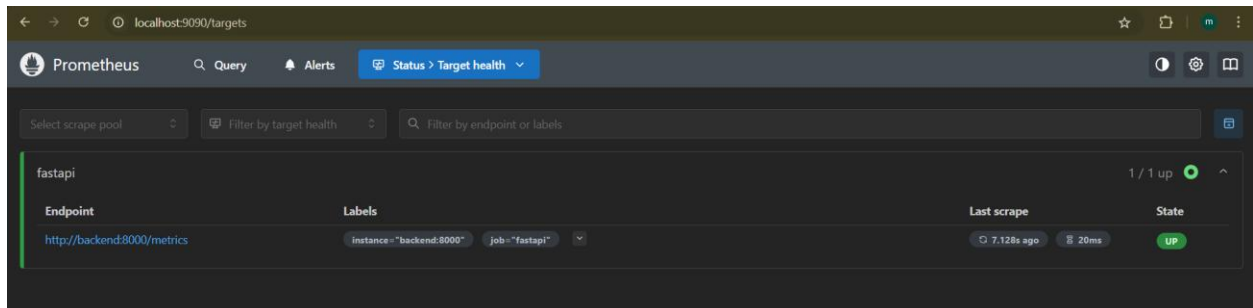
In User Portal:



The screenshot shows a web application interface for a user portal. On the left is a dark sidebar with a menu containing 'app', 'orders', 'payments', 'restaurants', and 'users'. Below the menu is a 'Select Section' dropdown currently set to 'Payments'. The main content area is titled 'Payments' and features a 'Make a Payment' button at the top. Below this are several input fields: 'Order ID' with the value '2', 'Amount' with '1070.00', 'Payment Method' set to 'Credit Card', 'Payment Status' set to 'Paid', and 'Transaction ID' with '21'. Each of these fields has minus and plus icons for adjustment. A 'Submit Payment' button is located below the input fields. At the bottom of the form, a green banner displays the message 'Payment recorded successfully!'.

PROMETHEUS

Status Health:



The screenshot displays the Prometheus web interface at the URL 'localhost:9090/targets'. The page title is 'Prometheus' and the active tab is 'Status > Target health'. The interface includes a search bar and filters for 'Select scrape pool', 'Filter by target health', and 'Filter by endpoint or labels'. The main table lists the targets for the 'fastapi' scrape pool. There is one target shown, which is 'up'.

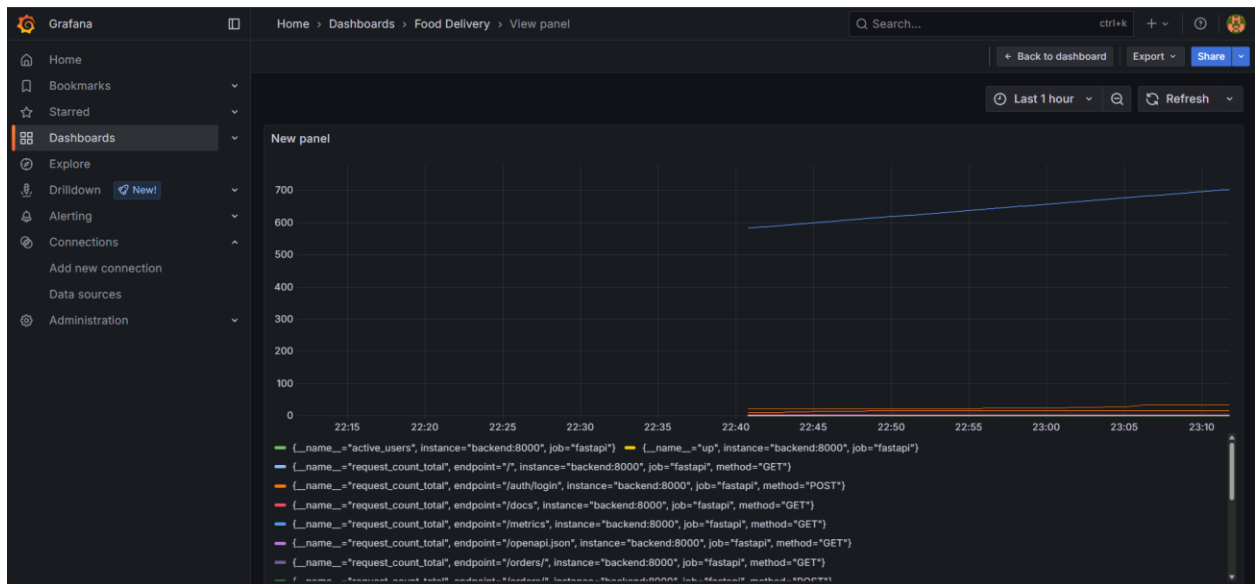
Endpoint	Labels	Last scrape	State
http://backend:8000/metrics	instance="backend:8000" job="fastapi"	7.128s ago 20ms	UP

Grafana:

Setting up Prometheus in grafana.

The screenshot shows the Grafana interface for configuring a Prometheus data source. The breadcrumb navigation at the top reads "Home > Connections > Data sources > prometheus". A search bar is located in the top right corner. The main header area displays the Prometheus logo and name, with a "Type: Prometheus" label. To the right, there are tabs for "Type" (showing "Prometheus") and "Alerting" (showing "Supported"). Below this, there are two tabs: "Settings" (selected) and "Dashboards". The "Settings" tab contains a "Name" field with the value "prometheus", a "Default" toggle switch which is turned on, and a "Prometheus server URL" field with the value "http://prometheus:9090". A note below the fields states: "Before you can use the Prometheus data source, you must configure it below or in the config file. For detailed instructions, [view the documentation](#). Fields marked with * are required".

Dashbaord:



7. Conclusion

This project presents a secure, modular, and production-ready Food Delivery System that integrates robust backend services with a clean, interactive frontend interface. The backend, developed using FastAPI, follows a well-structured, modular architecture that ensures a clear separation of concerns across models, schemas, routes, business logic (CRUD), and authentication utilities. By implementing JWT-based authentication and role-based authorization, the application enforces strict access control, ensuring that only authenticated users can interact with protected resources and that permissions are appropriately enforced based on user roles (admin/user).

The frontend, built using Streamlit, offers an intuitive interface for users and administrators, adapting dynamically based on the logged-in user's role. It leverages session-based JWT tokens to communicate securely with the backend, enabling real-time interactions such as order placement, restaurant management, user actions, and payment creation.

Additionally, the system is designed with scalability and observability in mind. A monitoring stack comprising Prometheus and Grafana is integrated to track backend performance metrics, including request latency, active users, and endpoint hit counts. This prepares the application for real-world deployment scenarios where reliability, security, and visibility are crucial.

Overall, this project demonstrates a complete, end-to-end implementation of a modern web application with a strong focus on security, maintainability, and user experience, making it suitable for both academic submission and professional deployment.