Project Report: Uber Eats Prototype

Group:
Shobhita Agrawal-**017552795**
Sushma Tacholi kudai-**017519060**
Kush Bindal-**017441359**

Backend - https://github.com/sushma1311/uberats_backend/tree/main
Frontend - https://github.com/sushma1311/ubereats_frontend
Django site admin - https://ubereats-backend-4199107521a9.herokuapp.com/admin/
Netlify-app - https://profound-lollipop-db5787.netlify.app/

1. Introduction

This project aimed to create a functional prototype of the Uber Eats platform using Django for backend development and ReactJS for the frontend interface. The objective was to implement a system that enables two types of user interactions — customers and restaurant owners. This system allows users to sign up, view restaurants, manage profiles, place orders, and track them, simulating real-world food delivery scenarios.

Project Goals

1. **Build a Robust and Scalable Platform**: Design a system that securely supports various user interactions and can handle high traffic with minimal latency.
2. **Enhance User Experience Across Devices**: Create a responsive user interface that adapts to different screen sizes, allowing users to interact with the platform seamlessly on any device.
3. **Document API Interactions**: Using Postman to provide comprehensive API documentation, simplifying backend access for developers.

This project demonstrates how modern frameworks like Django and ReactJS can be effectively combined to create a full-stack web application with a rich, user-friendly interface.

2. System Design

2.1 Architecture and Design Choices

The project follows a client-server architecture with clear segregation between the frontend and backend. This approach ensures that each component can scale independently, enhancing maintainability and deployment flexibility.

Backend (Django REST Framework)

- **Framework Selection**: Django was chosen for its comprehensive features, including Django REST Framework (DRF), which streamlines REST API creation.
- **Role-Based Access Control**: We used JWT authentication to manage access rights, with separate roles for customers and restaurant owners. This allows for tailored user experiences based on login credentials.
- **Data Models and Relationships**: The backend database in sqlite includes models for users, restaurants, dishes, orders, and favorites. Relationships between tables, such as foreign keys linking customers to orders and restaurants to dishes, were established to maintain data integrity.

Frontend (ReactJS)

- **Single-Page Application (SPA)**: ReactJS was selected for its efficient component-based architecture, enabling dynamic content rendering without frequent page reloads.
- **Routing and Session Management**: React Router manages navigation, while Axios facilitates API calls with token-based session management.
- **State Management**: Application states, such as user authentication status and cart items, are managed locally to ensure quick responses to user actions.

2.2 Deployment Strategy

To ensure smooth access and scalability, the project is deployed across the following platforms:

- **Frontend on Netlify**: Enables global content delivery with minimal setup, providing a fast and responsive user experience.
- **Backend on Heroku**: Heroku offers a reliable environment for Django applications and simplifies database management and scaling.

2.3 Security Measures and Validation

The application includes several security mechanisms to safeguard user data:

- **JWT Authentication**: Ensures each session is unique and authorized by generating tokens for each login.
- **Data Validation**: Input fields across the application undergo both client- and server-side validation, ensuring only correctly formatted data is processed.
- **Password Encryption**: Passwords are encrypted using Django's built-in hashing, protecting sensitive information.

2.4 Data Flow and Interaction

Upon user interaction, the frontend sends HTTP requests to the backend, which processes requests, accesses data, and returns responses. This interaction cycle is optimized to handle high volumes, ensuring real-time data processing for smooth user experiences.

3. Results

3.1 Customer Features

1. **Signup**: New customers register using basic details. Input validation ensures all required fields are filled out correctly.

2. **Sign in/Sign out**: Users securely log in and log out, with JWT tokens managing session persistence.

3. **Profile Page**:

View Profile: Displays user information, including name, profile picture, and favorite restaurants.

Edit Profile: Customers can update various personal details such as name, date of birth, and contact information. The profile picture upload feature includes image validation.

4. **Restaurant Tab**:

Browse Restaurants: Users can view restaurants, descriptions, and menus. Dynamic rendering displays each restaurant's unique menu.

Add to Cart: Customers select dishes to add to the cart. If items from a different restaurant are added, a confirmation modal ensures user clarity.

5. **Favorites**:

Favorite Management: Customers mark preferred restaurants as favorites for quick access.

6. **Place an Order**:

Cart Review: Displays items, quantities, and prices. Users can modify cart contents before proceeding.

Checkout: During checkout, users either add a new delivery address or select an existing one.

Order Confirmation: After completing checkout, users receive a success notification. This process includes real-time validation for a smooth transaction.

3.2 Restaurant Features

1. **Signup**: Restaurants register with their name, email, password, and location.

2. **Sign in/Sign out**: JWT-based login and logout functions provide secure access.

3. **Restaurant Dashboard**:

Profile Management: Restaurants can update business details such as location, contact information, timings, and add images.

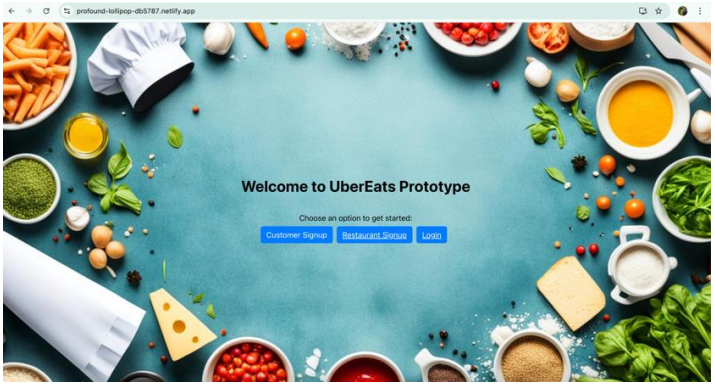
Menu Management: Restaurant owners can add or edit dishes, including images, ingredients, prices, and categories (e.g., appetizers, main courses).

Order Management: Restaurants view and manage orders, filtering them by status and updating the delivery status as needed.

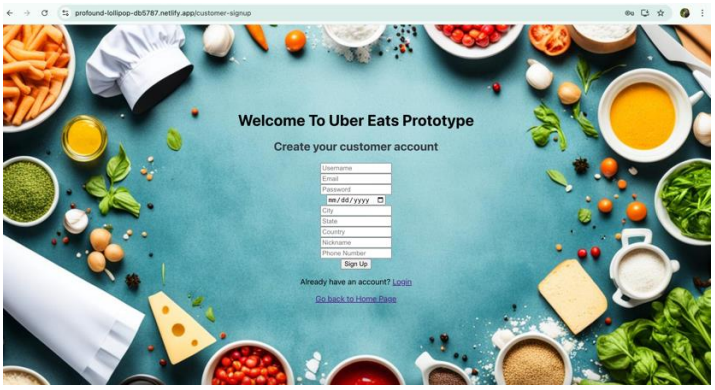
Screenshots, including profile pages, restaurant views, cart views, and order confirmations, are stored in the Git repository to document application workflows visually.

Few Sample pictures of ubereats prototype.

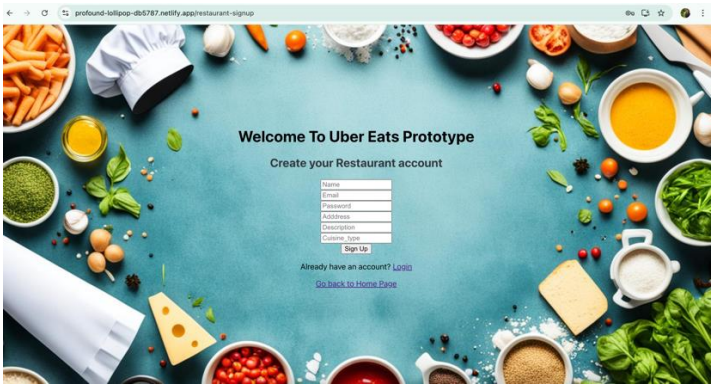
Home Page



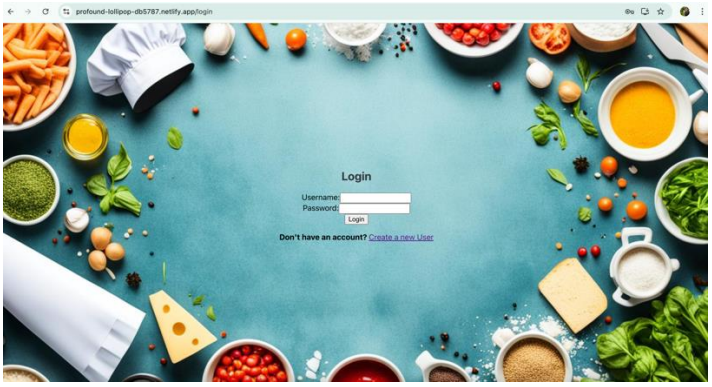
Customer signup page



Restaurant signup page

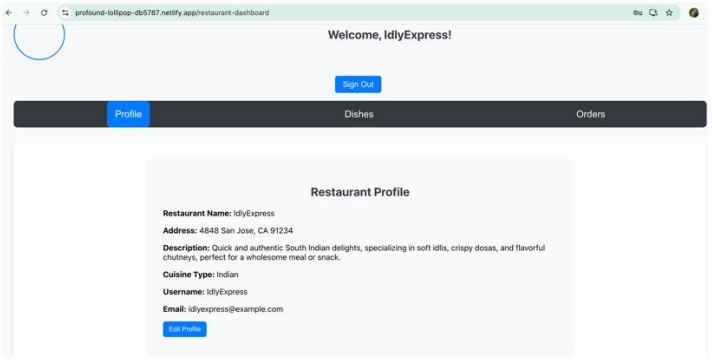


Login page

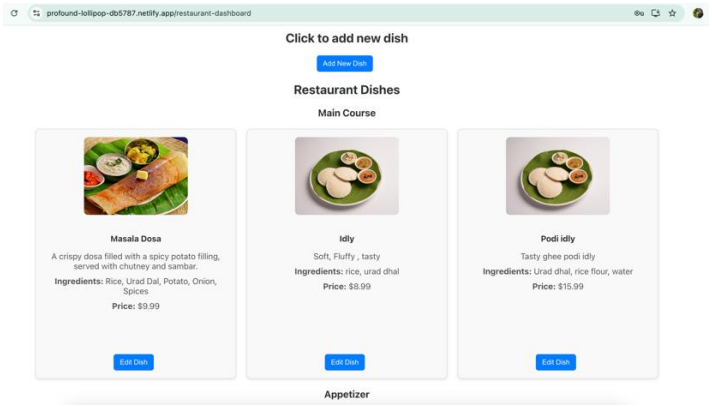


Restaurant Dashboard-

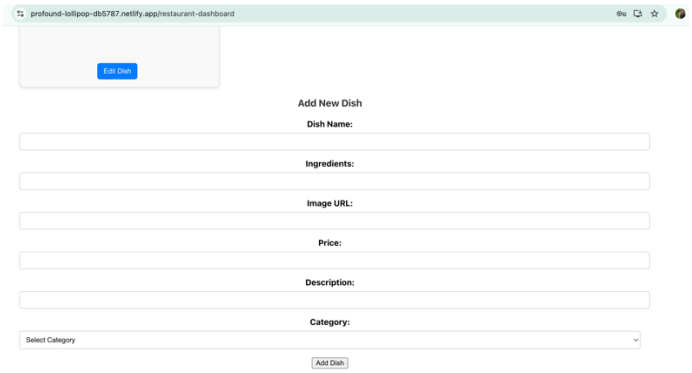
Restaurant Profile



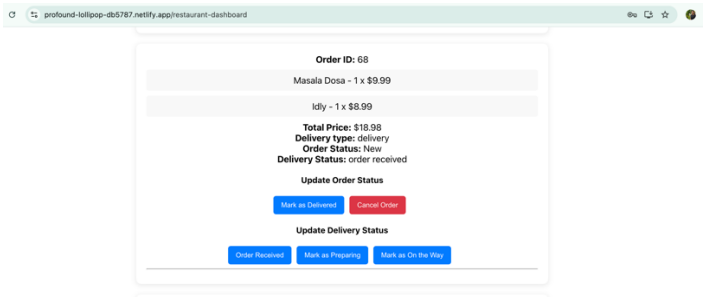
Restaurants Dishes page



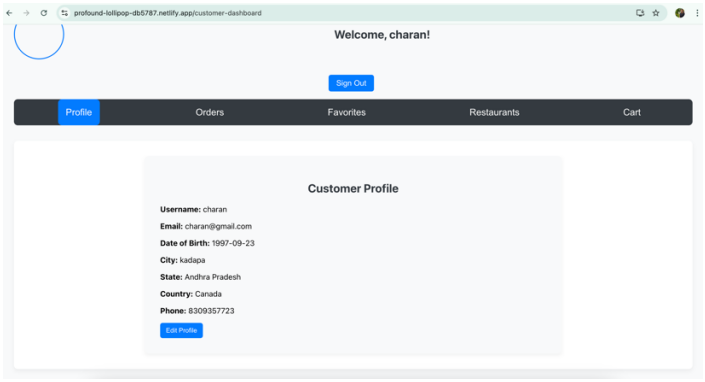
Restaurant can add / edit dishes



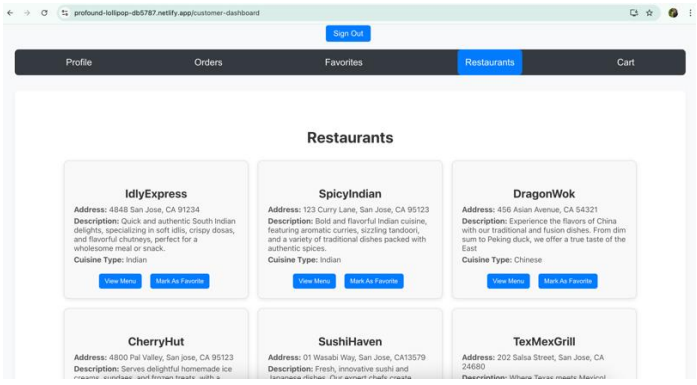
Restaurant orders



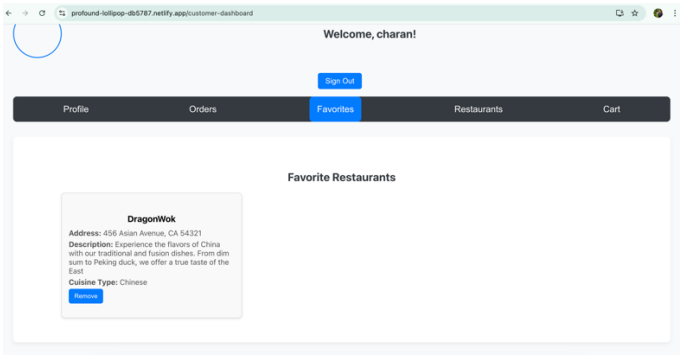
Customer dashboard –
Customer profile



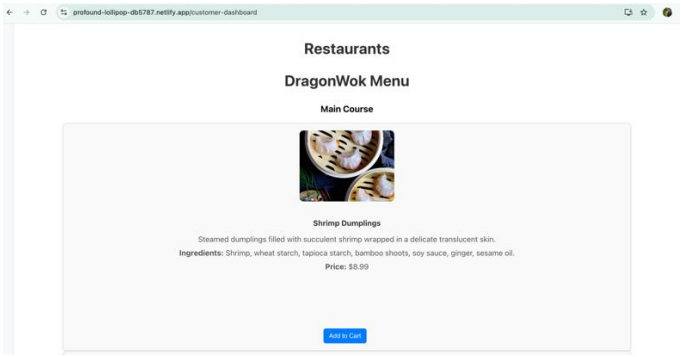
Restaurants List



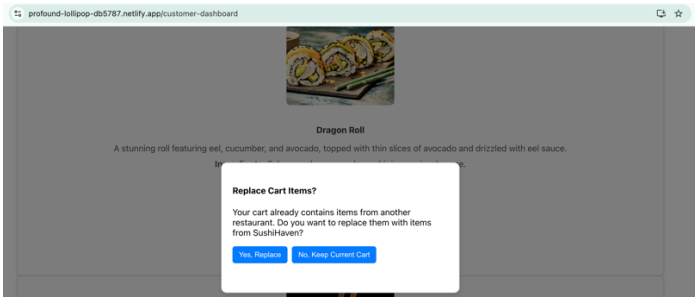
Favorites list page



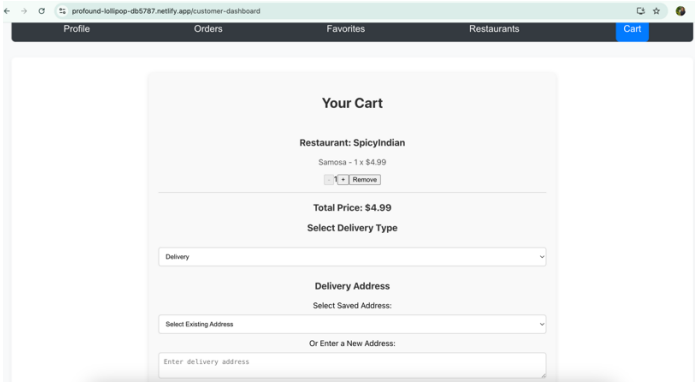
View menu page



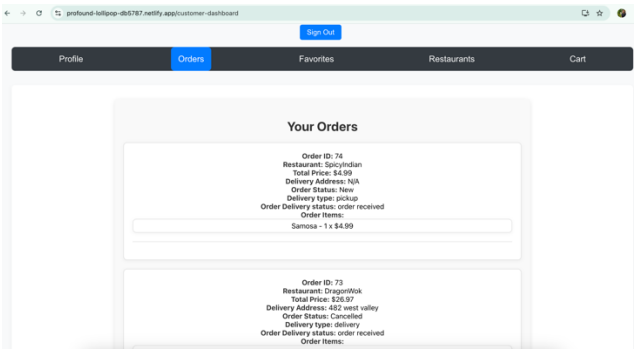
Notification, if we add dishes from multiple restaurants



Cart page



Customer order history page



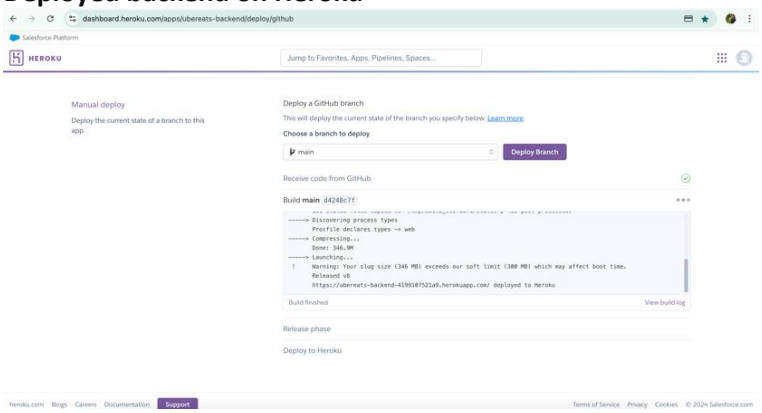
4. Performance

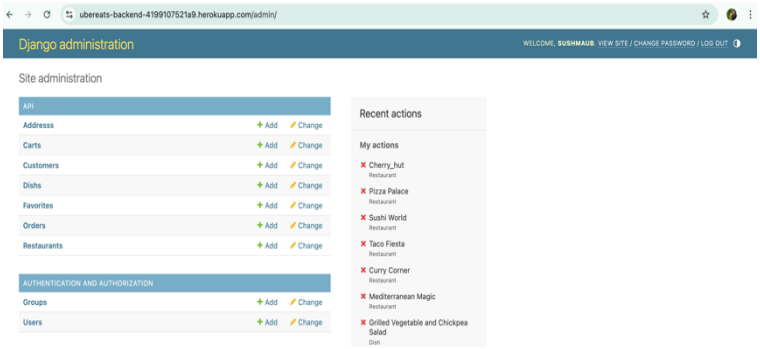
The frontend is developed using responsive design principles, ensuring compatibility across devices (mobile, tablet, and desktop). CSS frameworks, including Bootstrap, enhance adaptability and maintain consistency in visual elements. Each page layout dynamically adjusts to fit varying screen sizes, providing a smooth user experience.

The application underwent thorough testing across multiple devices and browsers to ensure compatibility and functionality. Postman was used to verify each API endpoint, checking parameters, response accuracy, and handling edge cases. Each feature was validated to meet expected outcomes, with debug logging enabling efficient identification of errors.

API Documentation was created using the Postman Collection.
Postman Collection was created, detailing headers, body parameters, and response examples for each endpoint. This format enables developers to quickly understand endpoint structures and integrate them into their applications.

Deployed backend on Heroku





Deployed frontend on Netlify