# PROJECT DOCUMENTATION

## 1. Introduction

***Purpose:***

The purpose of this system is to create an online food ordering and delivery platform which is similar to UberEats that allows customers to browse restaurants, view menus, add items to their cart, place orders, and manage their profiles. For restaurant owners, the platform helps manage their profiles, dishes, and orders. This system also supports user authentication, delivery address management, and order tracking to both customers and restaurants.

***Goals:***

- The system is designed to achieve the following goals:
- Better experience for users to browse, order, and track food deliveries.
- Enable restaurant owners to manage menus and orders effectively.
- Support scalability for a potentially large number of users and restaurants.
- Ensure secure, reliable data handling, especially for user profiles, orders, and sensitive information.

## 2. System Design

***Architecture Overview****:*
This system is built on client-server architecture using **Django REST Framework** for the backend and **React** for the frontend. The backend handles core functionalities such as user management, order processing, restaurant management, and cart handling, while the frontend provides user-friendly interface for user interactions. The system leverages the following architectural components:

- **Django REST API** for backend services
- **React** for the client-side interface
- **SQLite** (for development) as the primary database
- **Token-based Authentication** for secure login sessions
- **AWS EC2** for scaling and deployment.

***Components and Data Flow****:*

- **Authentication**: Utilizes Django's User model for both customers and restaurants, with token-based authentication ensuring secure access to different user routes.
- **Order Processing**: Orders are stored in the Order model, with each order containing multiple OrderItems representing individual dishes. Customers add dishes to their cart, which can later be converted into orders. These orders can be viewed on the Order Management dashboard of the
- **Profile Management**: Customers and restaurants manage their profiles with separate models, allowing each user type to view or update their details securely.

- **Dish and Menu Management**: Restaurant owners manage dishes through Dish models associated with Restaurant, with options to add and edit dishes from their menus.
- **Favorites and Cart**: Customers can save favorite restaurants and manage cart items, which are linked to orders upon checkout.
- **Order Tracking**: Customers and restaurant owners track orders using a series of status updates (new, preparing, on the way, etc.). Once there is an update from Restaurant, eventually the order status gets updated at the user end.

Uber eats prototype - Process chart (User)

**User Login** ( **React-** Authenticate User, **Django-** Issue token, **Database-** Store credentials)

↓

**Browse Restaurants** (**React** – Display list, **Django** – API call)

↓

**Select Dish** (**React** – Show menu, **Django-** Fetch data)

↓

**Add to Cart** (**React-** Update cart, **Django-** Fetch data)

↓

**Place Confirmation** (**React** – Submit order, **Django**-Record order, **Database** – Save order)

↓

**Restaurant Prepares Order** (**Django**-Update order / status, **Database**-Store order and status)

↓

**Order Update** (**React-** Show final order status, **Database**- Record final order status)

Uber eats prototype - Process chart (Restaurant)

**Restaurant Login** ( **React-** Authenticate User, **Django-** Issue token, **Database-** Store credentials)

↓

**Manage Profile** (**React** – Update Profile, **Django** – API call, **Database** – Store Profile data)

↓

**Manage Menu** (**React** – Display / Edit menu, **Django-** Fetch data and update data, **Database** – Store details)

↓

**Receive Orders** (**React-** View Orders , **Django-** Fetch orders)

↓

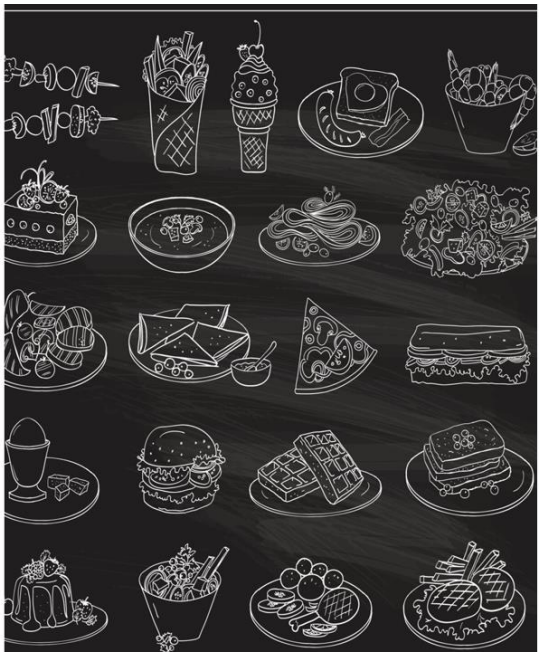
**Place Confirmation** (**React** – Confirm order, **Django**-Record order, **Database** – Save order)

↓

**Update order status** (**Django**-Update status, **Django-** Process status change, **Database**- Record status)

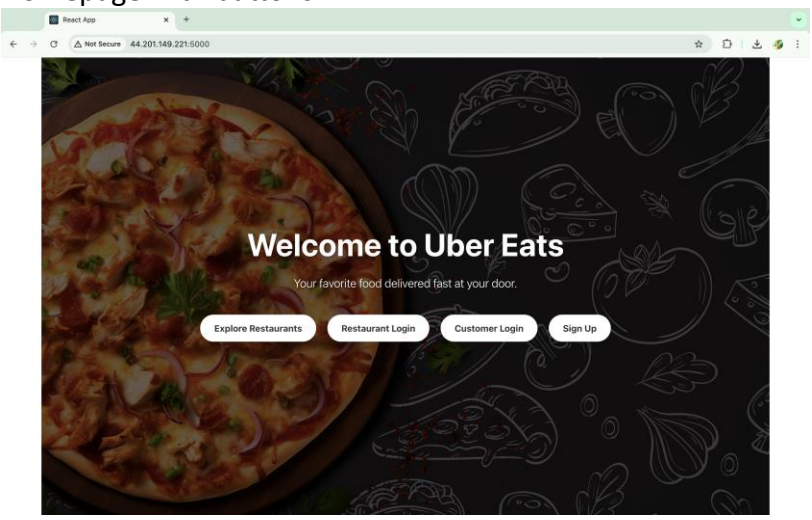
↓

**Order Fulfillment** (**React-** Show final order status, **Database**- Record final order status)

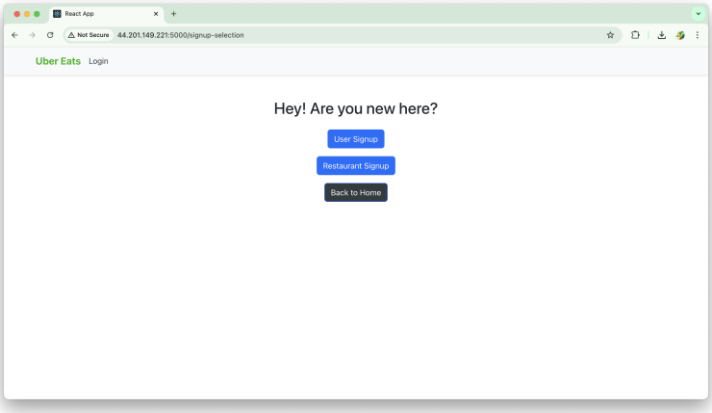


## 3. Results:

1. Homepage with buttons

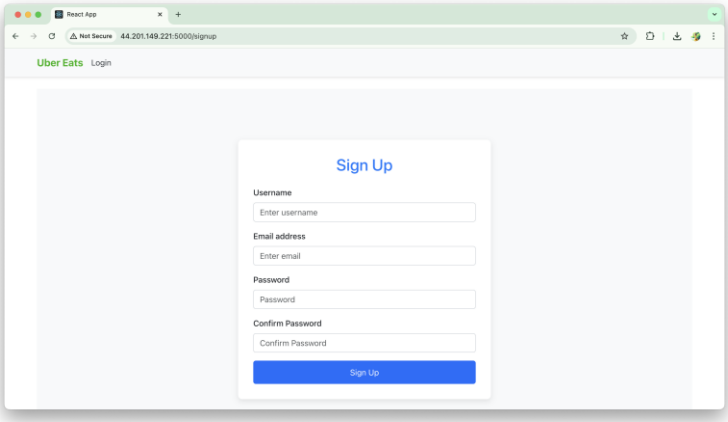



2. Signup Selection

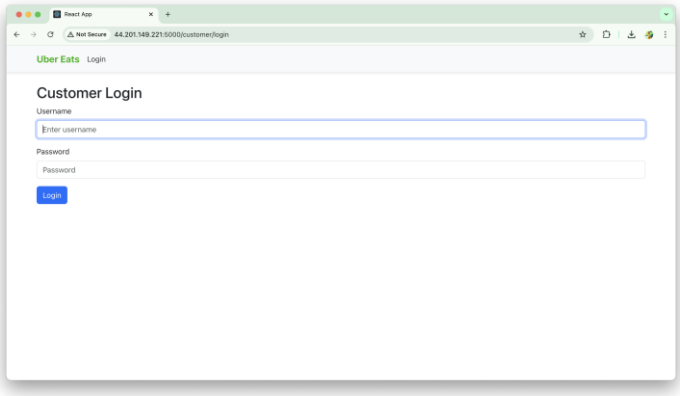



### PART 1: On the Customer / User side:

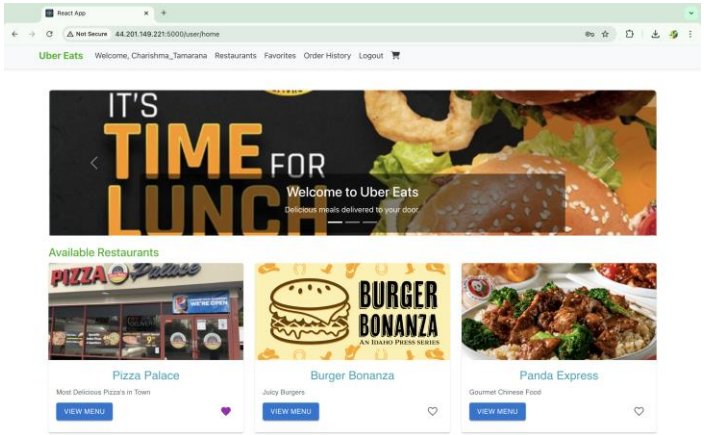
1. Customer Signup



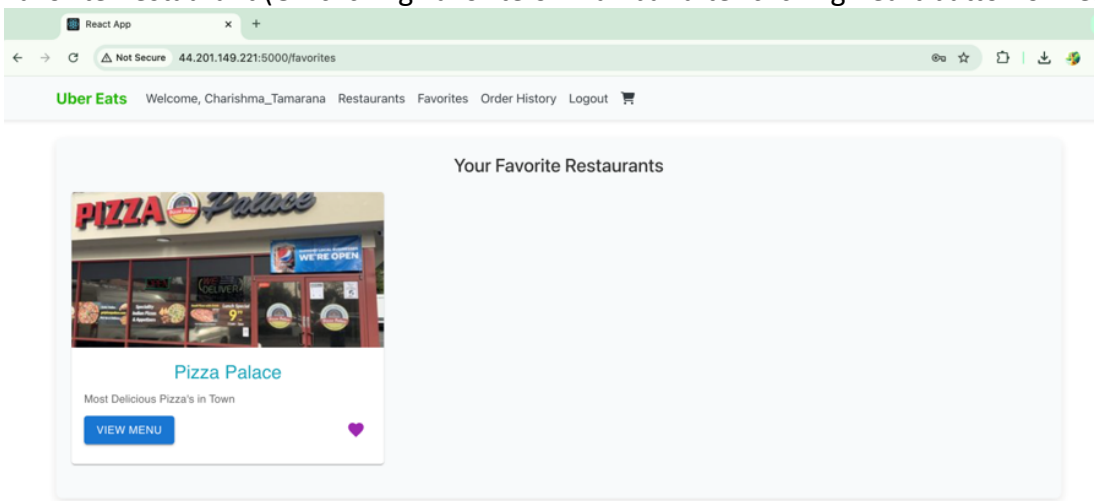
2. Customer Login



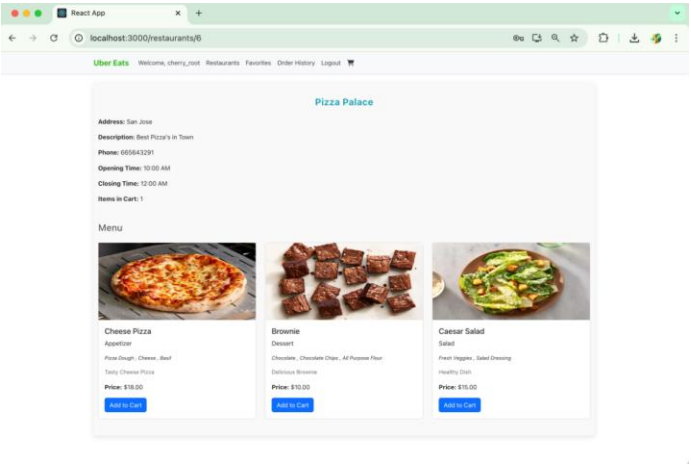

3. Restaurants Details



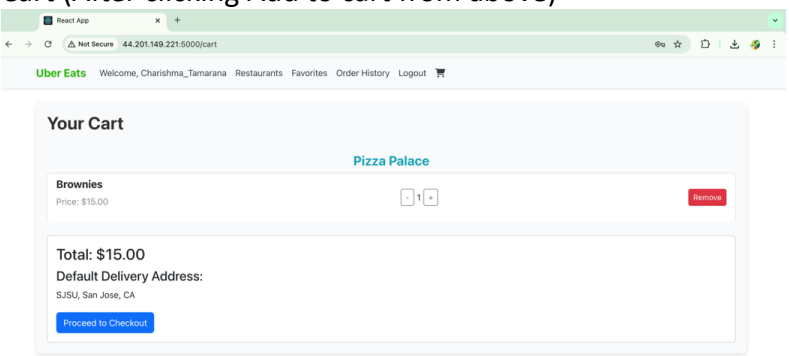

4. Favorite Restaurant (On clicking Favorite on Nav bar after clicking heart button on restaurant card)



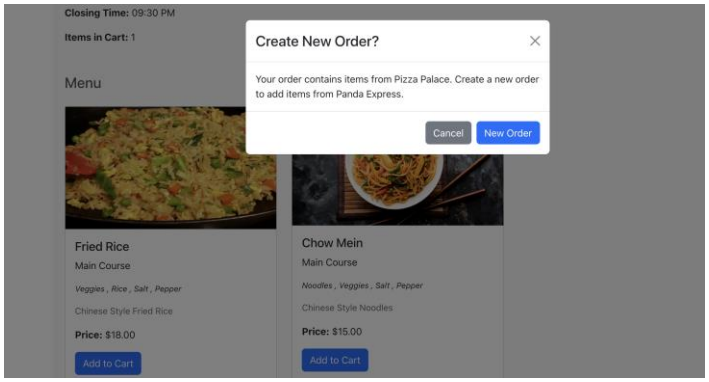
5. Restaurant Menu with dishes



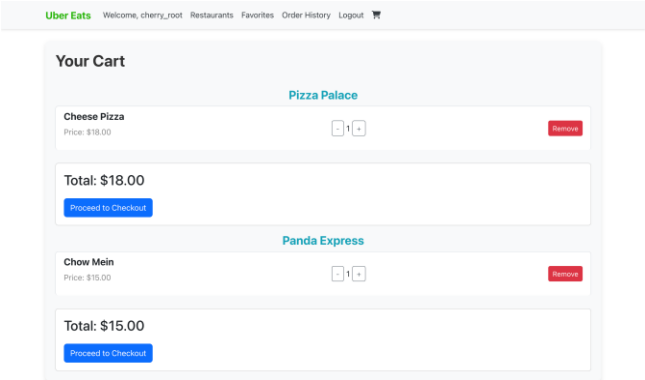

6. Cart (After clicking Add to cart from above)



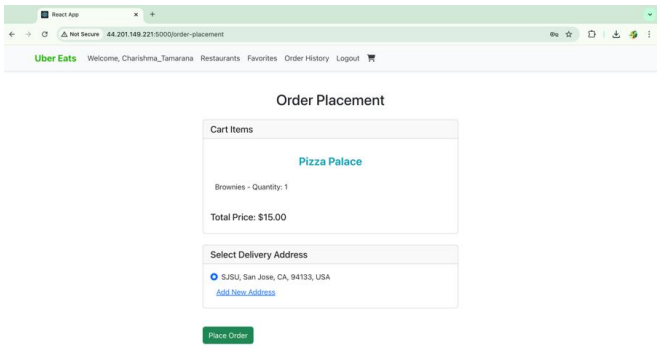

Dialog confirmation if order placed from another restaurant ( when add to cart is selected )



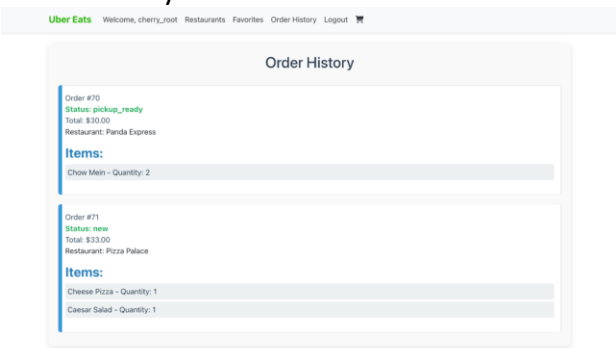

In Cart when New Order is Selected, It groups the new order with the restaurant and dish details but user can place order (proceed to checkout ) only from one restaurant :




7. Order Placement

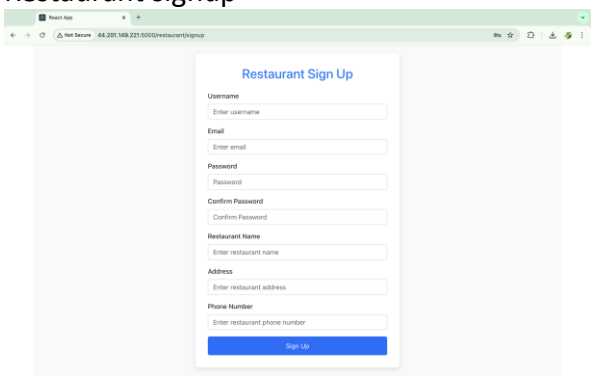



8. Order History

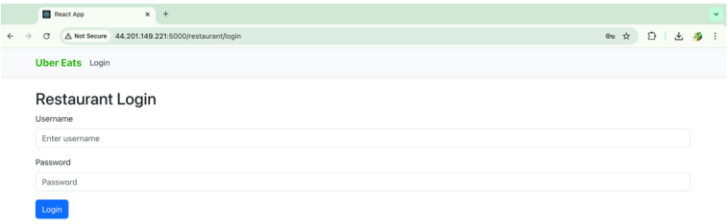



## Part 2: Restaurant Side:

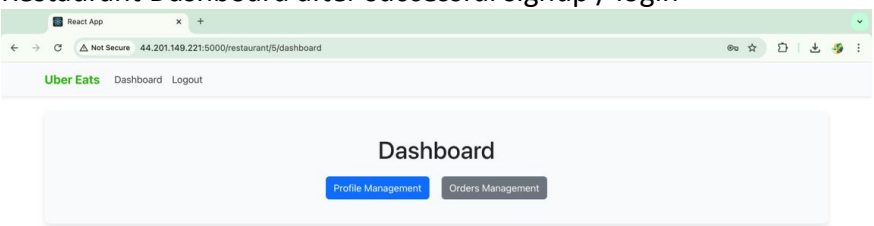
1. Restaurant signup



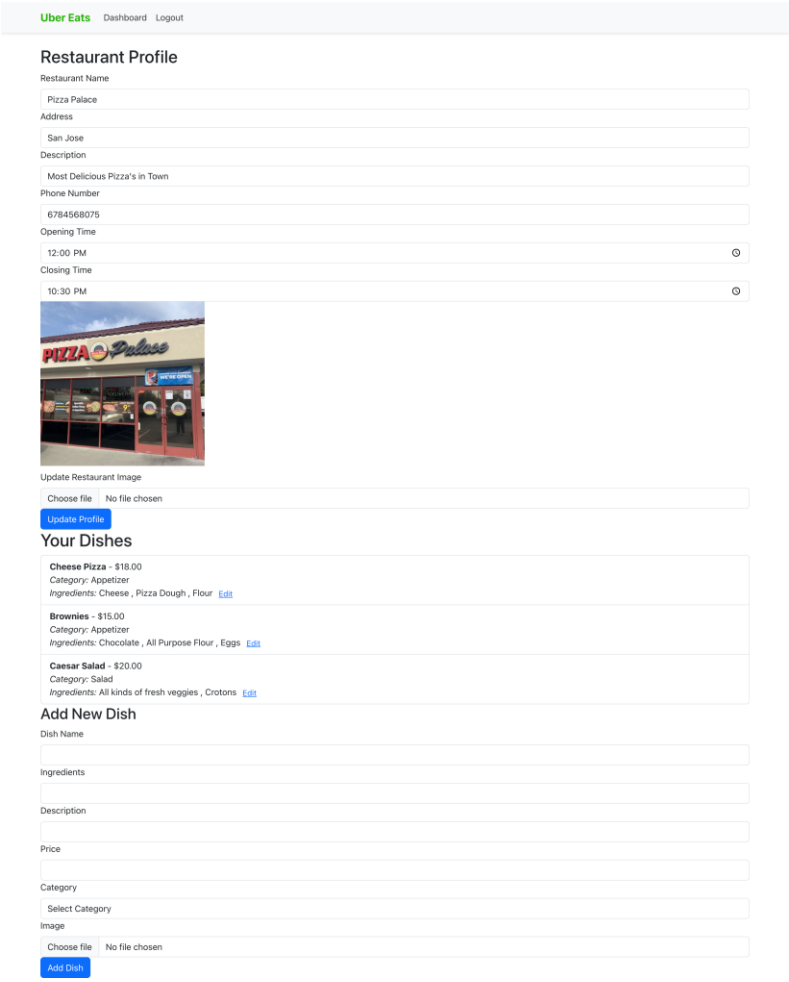
2. Restaurant Login



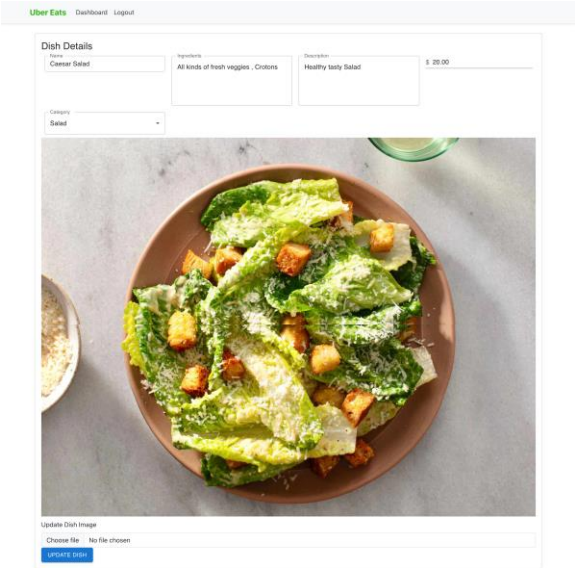

3. Restaurant Dashboard after successful signup / login



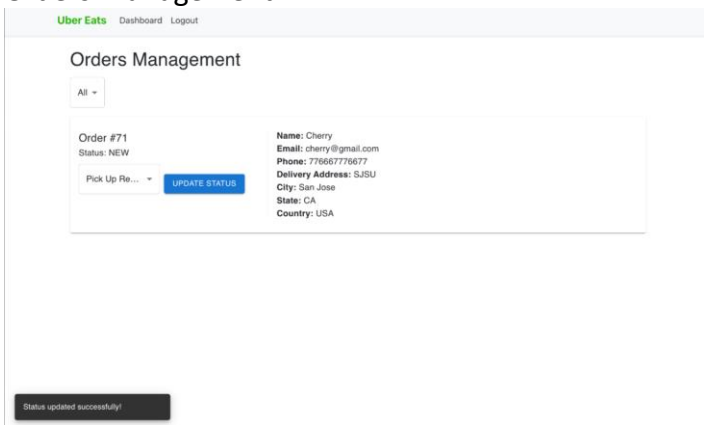

4. Profile Management



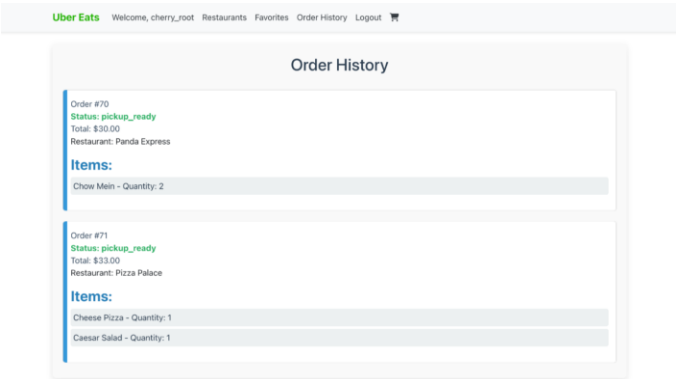
5. Edit Dishes (After clicking Edit from your Dishes )




6. Orders Management



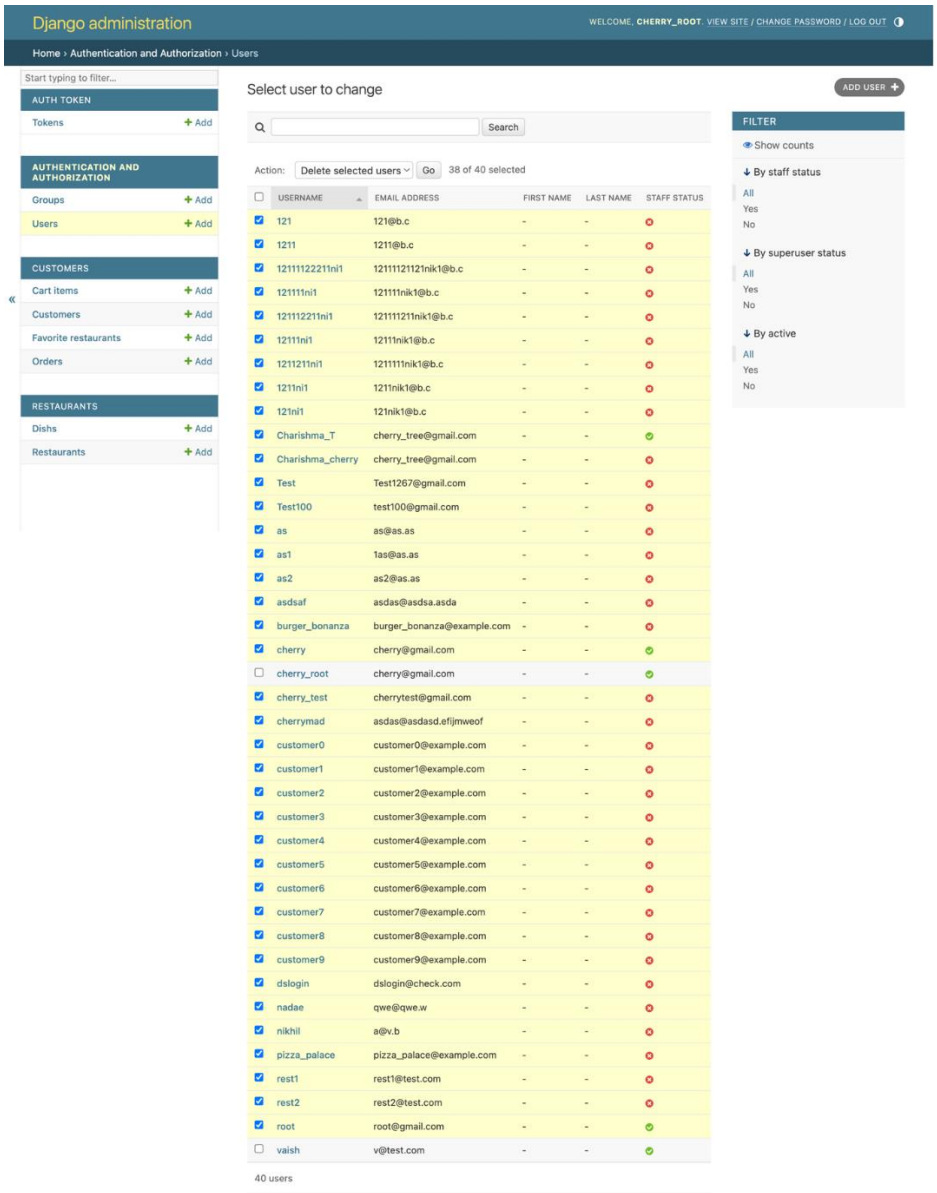

On user side, status updated (pickup_ready On Pizza Palace)



## 4.Performance:

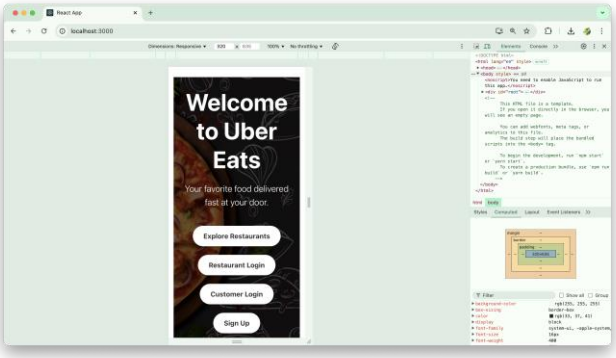
In the Django Admin panel, a total of 40 users are managed, few associated with specific restaurant and dish information. This setup organizes data for user-specific operations, such as managing individual restaurants, updating menu items, and viewing orders, all through a centralized admin interface.

While the current setup has 40 users, Django Admin's pagination and filtering features make the application scale further if the user base or associated data grows.

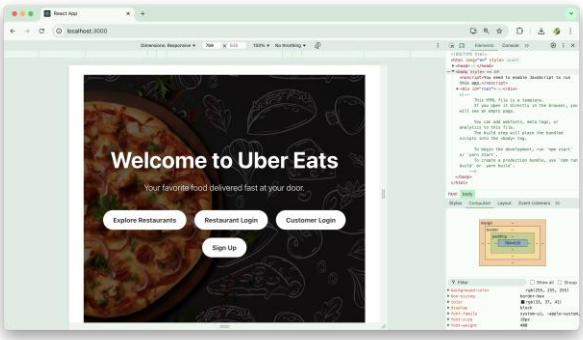


**Responsive Design:**

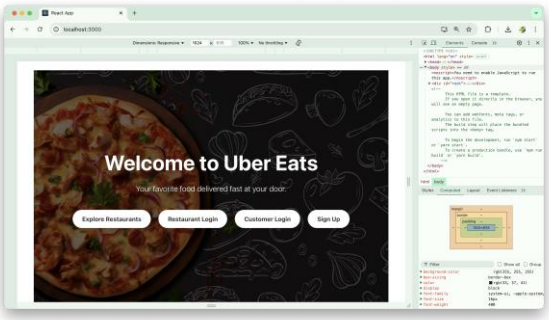
On Mobile view:




On Tablet view:




On Laptop view:




**Paired Programming (Group 7)**

Vaishnavi Nanduri (017506489)

Charishma Tamarana (017549181)