# DATA-236 Sec 11 - Distributed Systems

**Lab 1 Assignment**: Using Django and ReactJS
**Due**: October 21th, 2024, 11:59 PM

This lab assignment covers developing REST services using Django and ReactJS. It is worth **30 points** and is an **individual effort** (teamwork is not allowed).

---

**Prerequisites**

- You should be able to run the basic Django and React applications discussed in class.
- You must be familiar with JavaScript programming.

---

## Grading Breakdown

**UberEats Prototype Application** – 30 marks

**Note**: Late assignments will be accepted but will incur a penalty of **-5 points per day**. Submissions received **before or on the due date** will be eligible for maximum points.

---

## UberEats Prototype Requirements

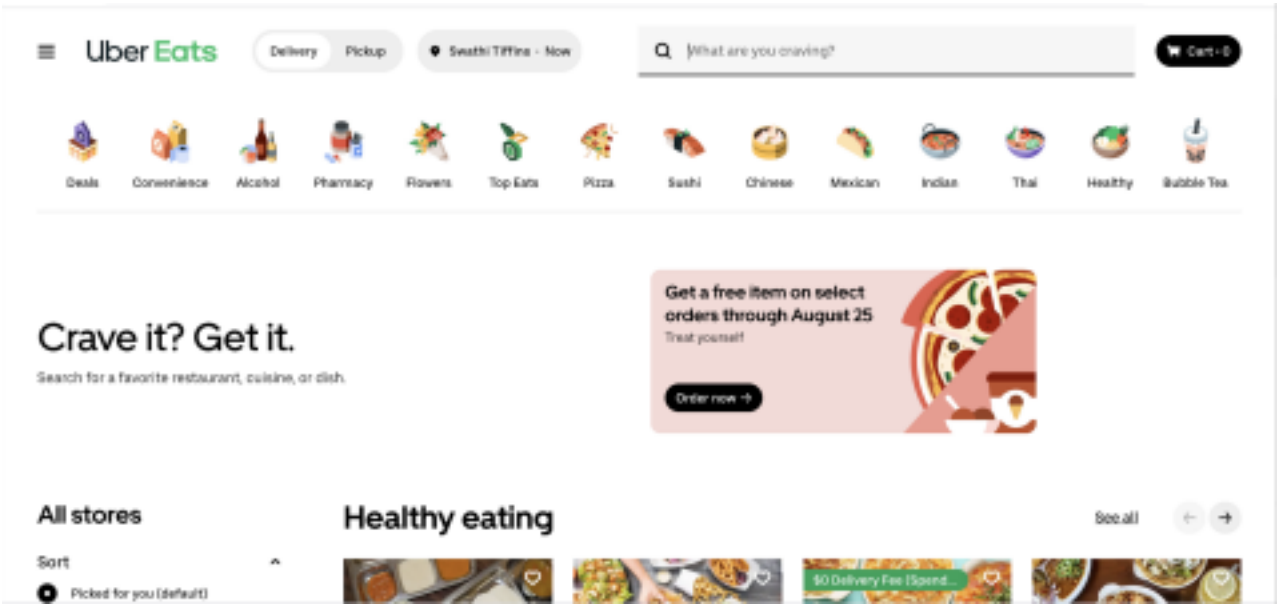
You are required to develop a **Prototype of UberEats** using React and Django. Refer to the UberEats website to understand its core functionalities. The application must support two main personas:

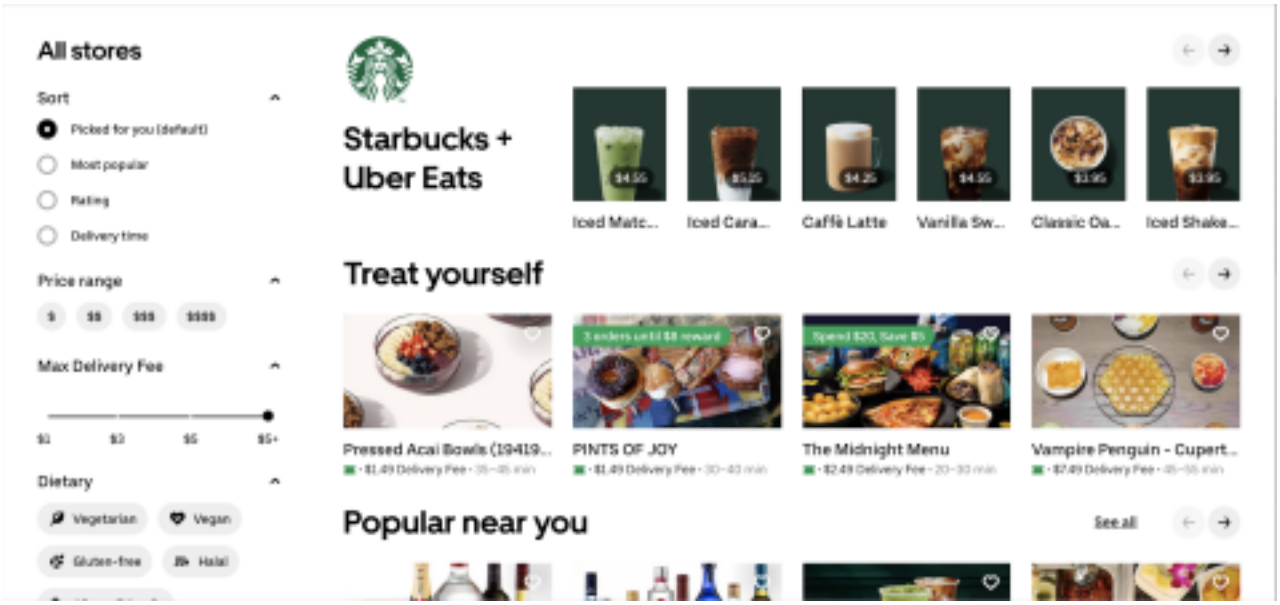
1. **Customer**
2. **Restaurant**

**Required Features:**

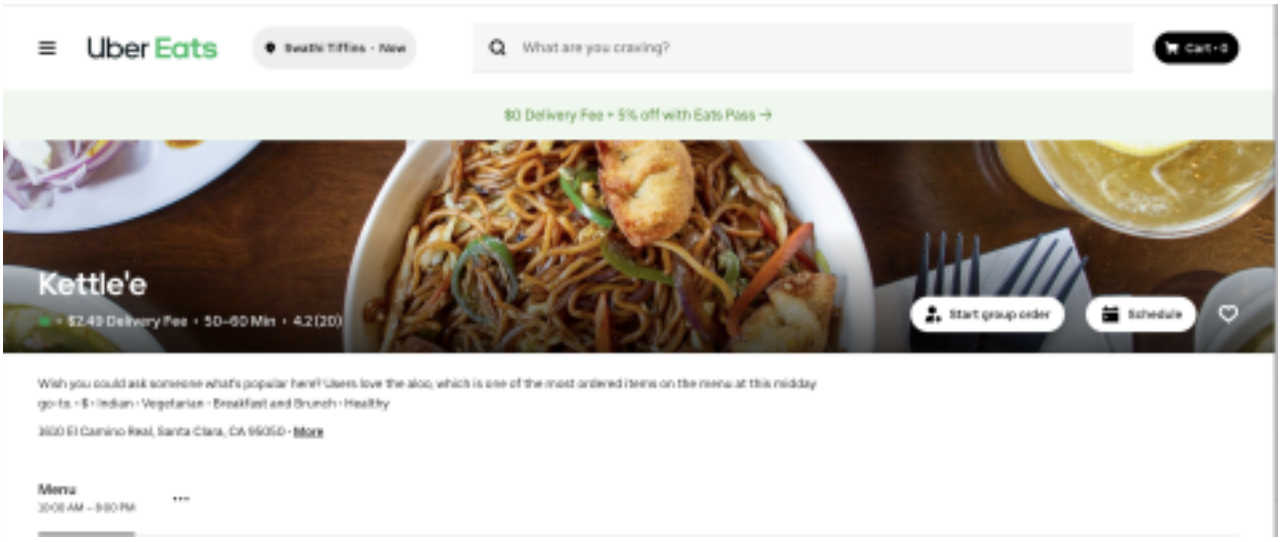
**Customer Features:**

1. **Signup** – Customer sign-up with name, email ID, and password.
2. **Sign in/Sign out** – Implement customer login and logout functionality.
3. **Profile Page**
    - Display customer details (basic info, favorites, profile picture).
    - Update profile information (name, date of birth, city, state, country, nickname, etc.).
    - Upload profile picture.
    - Update contact information (email ID, phone number).
    - **Note**: The country field should be a dropdown with a predefined list of countries (you may also use any external API endpoints to populate this information)

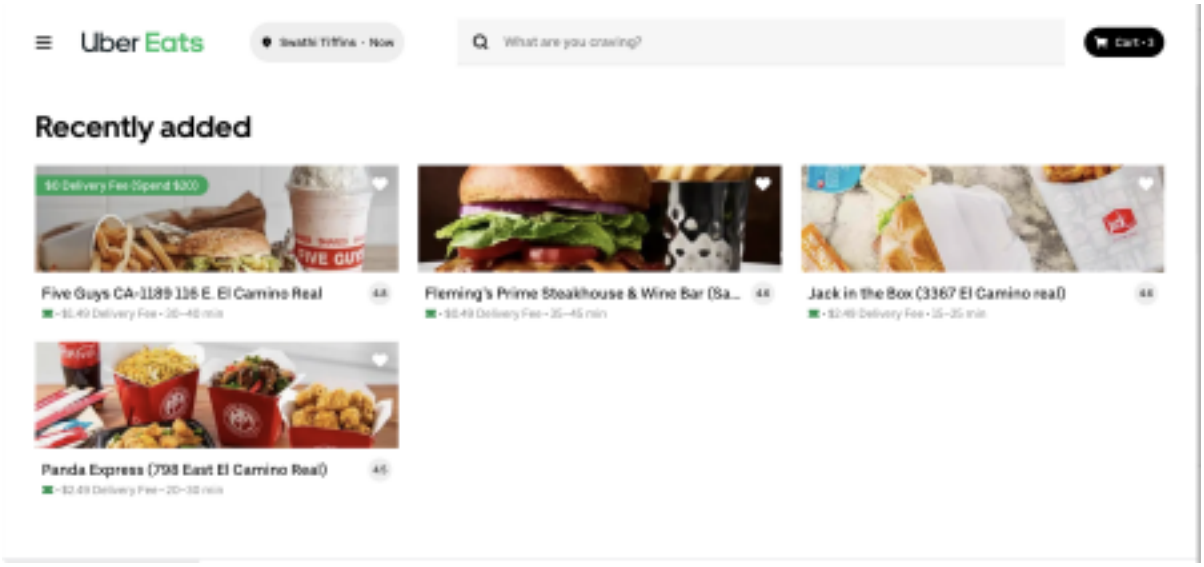


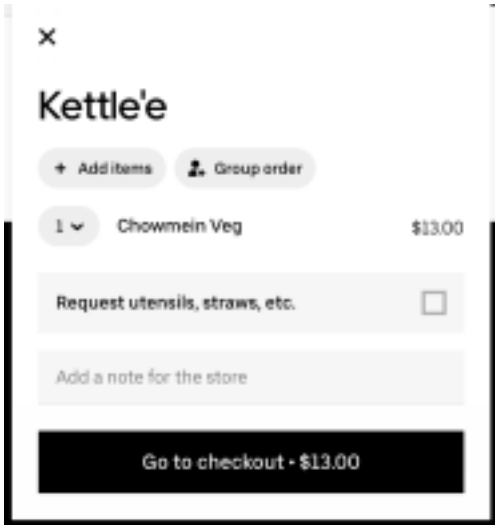

4. **Restaurant Tab**
    - View restaurant details, including a brief description and menu.
    - Select a dish to add to the cart.
    - Finalize the order in the view cart section.



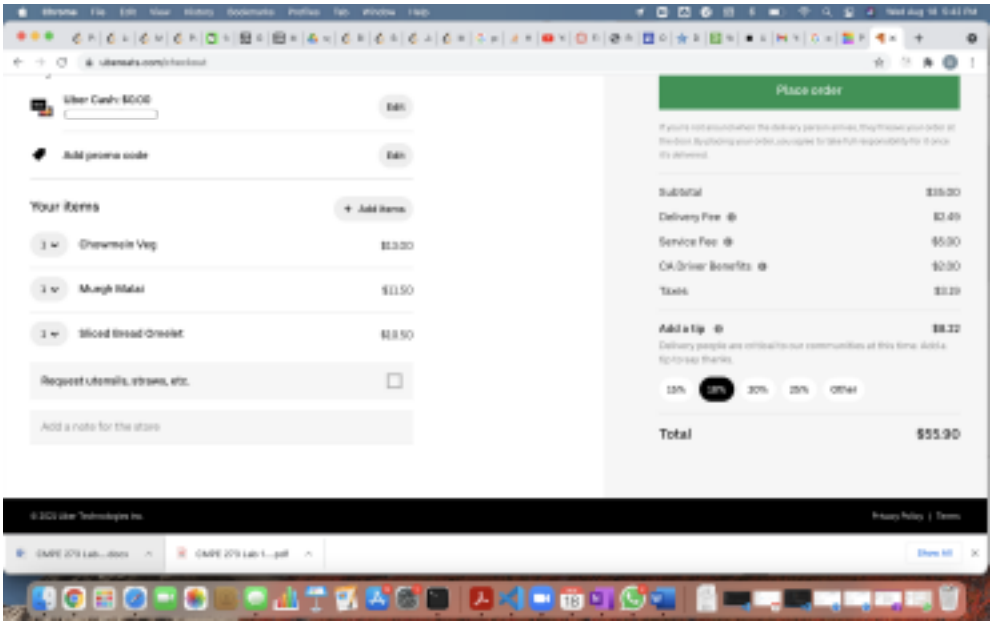

5. **Favorites**
    - Mark restaurants as favorites.
    - Display a list of favorite restaurants in a “Favorite” tab.



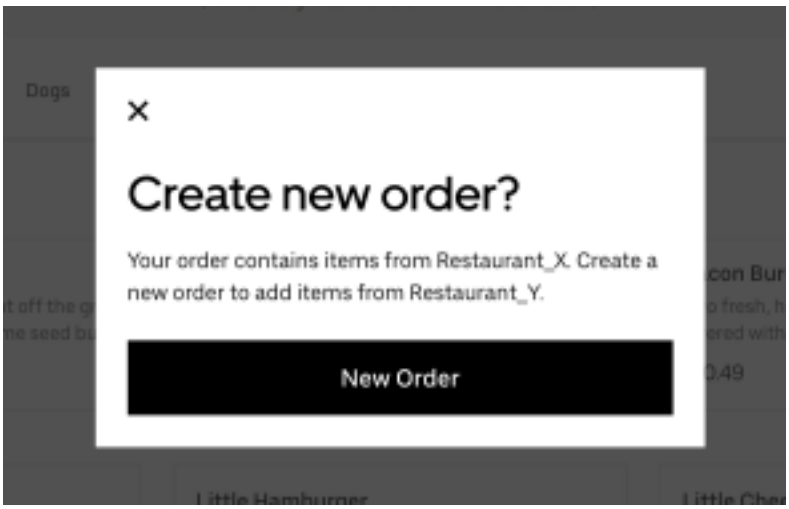

6. **Place an Order**
    - View cart with a list of items selected for the order.




- Proceed to checkout and provide a delivery address or select an existing one.




- Confirm and place the order with a “Order placed successfully” notification.
- **Note**: If the customer adds items from another restaurant, a confirmation dialog should be displayed.

- 


**Restaurant Features:**

1. **Signup** – Restaurant sign-up with restaurant name, email ID, password, and location.
2. **Sign in/Sign out** – Implement restaurant login and logout functionality.
3. **Restaurant Dashboard**
    - **Profile Management**
        - View and update restaurant profile (name, location, description, contact info, images, timings, etc.).
        - Add or edit dishes with details (dish name, ingredients, image, price, description, category – Appetizer, Salad, Main Course, etc.).
        - View a list of added dishes.
    - **Orders Management**
        - View and filter customer orders by status: New, Delivered, Cancelled.
        - Update order delivery status: Order Received, Preparing, On the Way (for delivery), Pick up Ready (for pickup), Delivered, Picked Up.
        - View customer profiles for each order.

---

## Validation and Security Requirements:

- Ensure proper **exception handling** and **validation** for all input fields.
- **Passwords** must be securely encrypted.
- Provide **appropriate feedback messages** for successful and unsuccessful actions.

---

## Deployment:

- The application must be deployed to the cloud (e.g., **Heroku**, **AWS EC2**).
- Your frontend should be **simple, attractive**, and **fully responsive** across all devices. Good design will earn additional marks.

---

## API Documentation:

You are required to document your API endpoints using **Swagger** or a **Postman collection**.

1. **Swagger**:
    - Use Swagger to automatically generate API documentation for your Django REST Framework code. Ensure all API routes, methods (GET, POST, etc.), and inputs/outputs are properly documented.
    - Tools such as `drf-yasg` can help generate Swagger documentation.
    - The Swagger UI should allow for testing the API directly.
2. **Postman Collection**:
    - Alternatively, create and export a **Postman collection** with descriptions for each endpoint, request parameters, headers, and sample responses.
    - Submit the Postman collection along with your project.

**Why This Is Important**: Proper API documentation simplifies the understanding, maintenance, and testing of your application. It allows other developers to interact with your API without diving into the code.

---

## Non-Functional Requirements:

In addition to functional features, your application should meet the following **Non-Functional Requirements**:

1. **Responsiveness Across Devices**:
    - Your web application must be responsive, adapting to different screen sizes (mobile, tablet, desktop). Use CSS frameworks like **Bootstrap** or **media queries** to ensure this.
    - Provide screenshots to demonstrate how the application looks on mobile, tablet, and desktop.
2. **Accessibility**:
    - Implement basic accessibility features such as semantic HTML tags (`<header>`, `<nav>`, `<footer>`), `alt` text for images, and support for keyboard navigation.
3. **Scalability**:
    - Consider how your application would scale as user load increases. Optimize database queries and avoid loading unnecessary data to ensure smooth performance with a growing number of users.

---

## **Git Repository:** Create a private repository for this lab project.

1. **Folder Structure**:
    - Organize your repository into two sub-folders: **Frontend** and **Backend**.
    - Place all source code in the respective folders.
2. **Commit History**:
    - Add detailed commit messages describing changes made.
    - Regular commits are required. (A penalty of 3 marks will be applied if this is missed.)
3. **Dependencies**:
    - Do not submit dependency files (this will result in a 2-mark deduction). Instead, include

    - them in `requirements.txt` (Django) or `package.json` (React).
4. **Readme**:
    - Your repository's **README** file must include instructions to run the application.
5. Invite shreenithi-sivakumar-20(shreenithi.sivakumar@sjsu.edu) to your private project repository.

---

## Project Report:

Your report should contain the following sections:

1. **Introduction**: State the purpose and goals of the system.
2. **System Design**: Describe your chosen system architecture and design.
3. **Results**: Include screenshots of key application screens and test results. (Include detailed screenshots in your git repository)
4. **Performance**: Analyze the performance of the application and discuss the results.

---

## Submission Guidelines:

- Submit your project report (named `John_Lab1_Report.doc`) on **Canvas** before the deadline.
- The report should be **10 pages or fewer**.