UE21CS352B - Object Oriented Analysis & Design using Java

Mini Project Report

"Food Delivery Management"

Submitted by:

Syed Hamza Hussain	**PES1UG21CS655**
Sunidhi S Naik	**PES1UG21CS643**
Stuthi M Udupa	**PES1UG21CS628**
Sujay S Gudur	**PES1UG21CS637**

6th Semester K Section

Prof. Bhargavi Mokashi
Assistant Professor

January - May 2024

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
FACULTY OF ENGINEERING
PES UNIVERSITY
(Established under Karnataka Act No. 16 of 2013)
100ft Ring Road, Bengaluru – 560 085, Karnataka, India

Problem Statement

In today's fast-paced world, online food delivery services have become indispensable for many people. However, existing platforms often lack the simplicity and efficiency needed for a smooth user experience. This project proposes a Java-based Online Food Delivery System designed to enhance the ordering process and cater to a wide range of users.

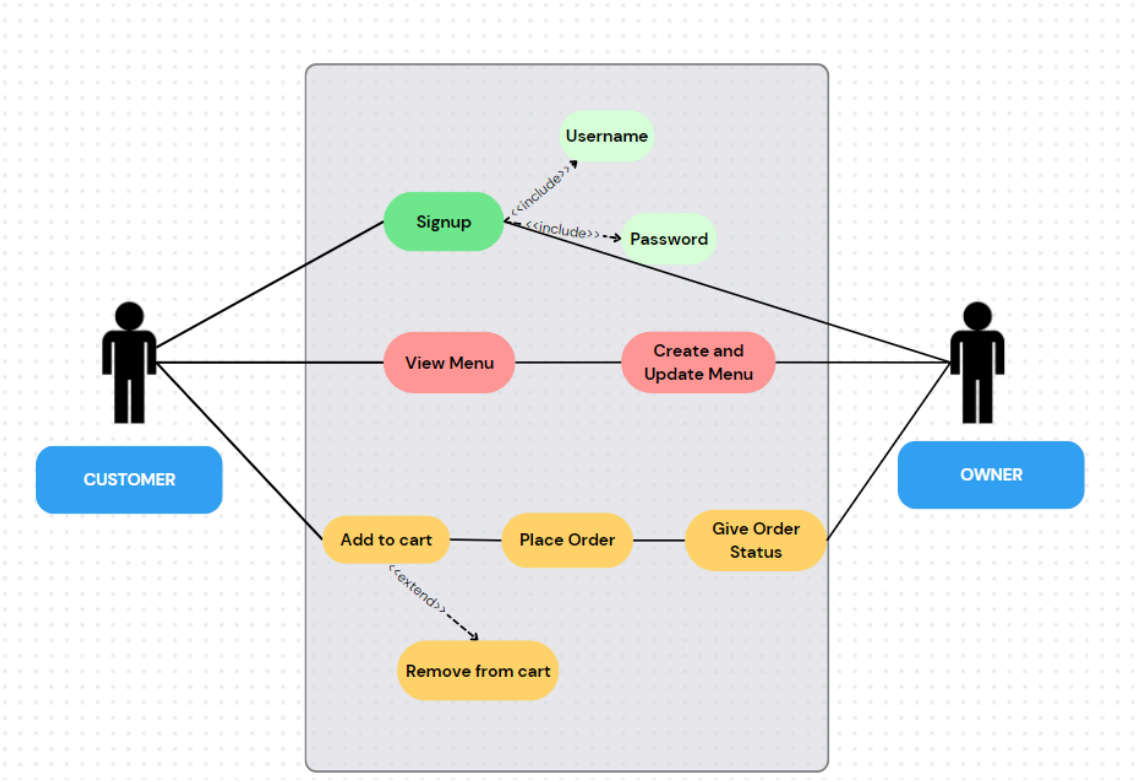
- User-Friendly Interface: The system will prioritize simplicity, allowing users to easily explore available restaurants, view menus, place orders, and track deliveries. The goal is to create an intuitive platform accessible to users of all technological backgrounds.
- Streamlined Ordering Process: The system aims to simplify the ordering process by providing access to nearby restaurants, offering personalized recommendations, and enabling easy reordering of preferred items.
- Support for Restaurants and Delivery Partners: The platform will provide essential tools for restaurants to manage their offerings and process orders efficiently.
- Scalability and Reliability: The system will be designed to handle increasing demand over time, ensuring reliable performance during peak hours. This will involve

implementing a robust backend infrastructure and database management for a smooth user experience.

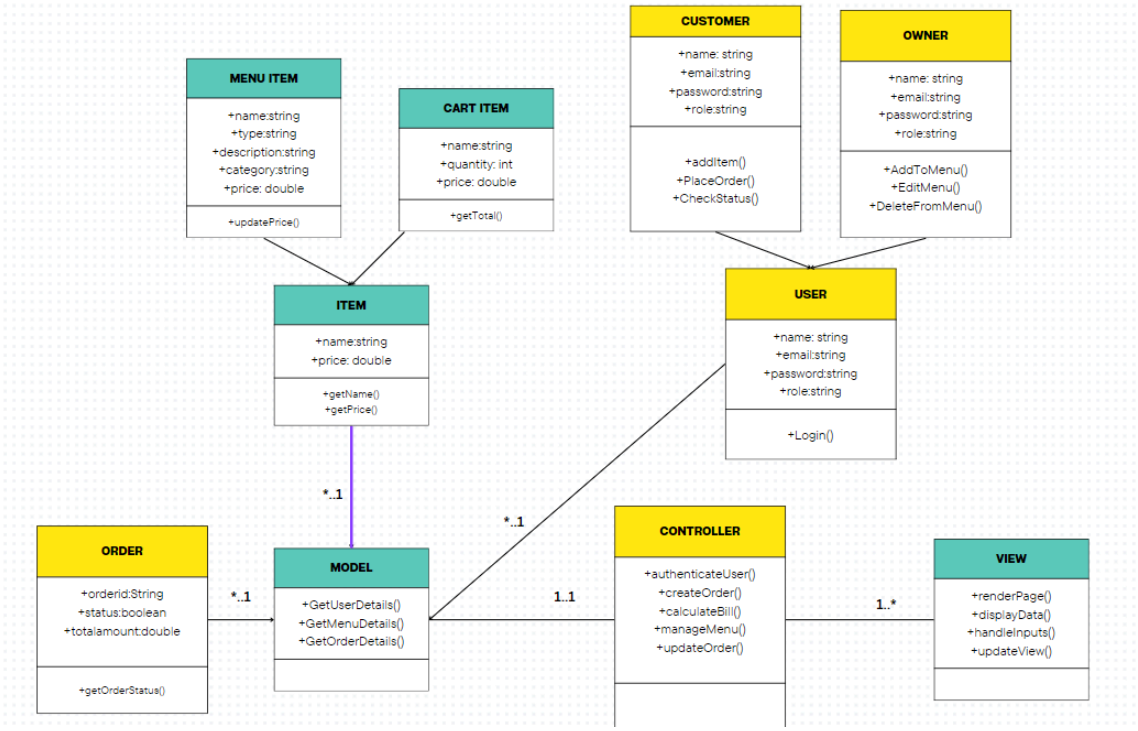
The Online Food Delivery System project aims to address the current shortcomings of existing platforms by providing a user-friendly, efficient, and reliable solution for ordering food online.

UML Models:

Use case Diagram:

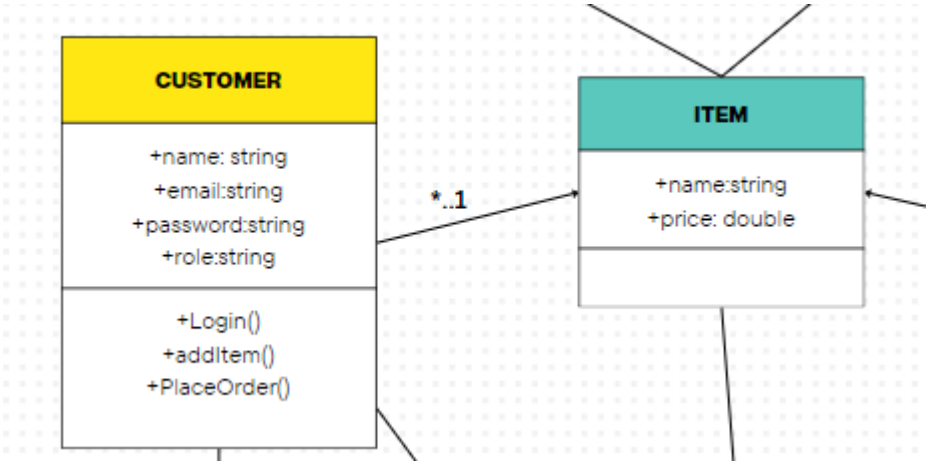


Class Diagram:



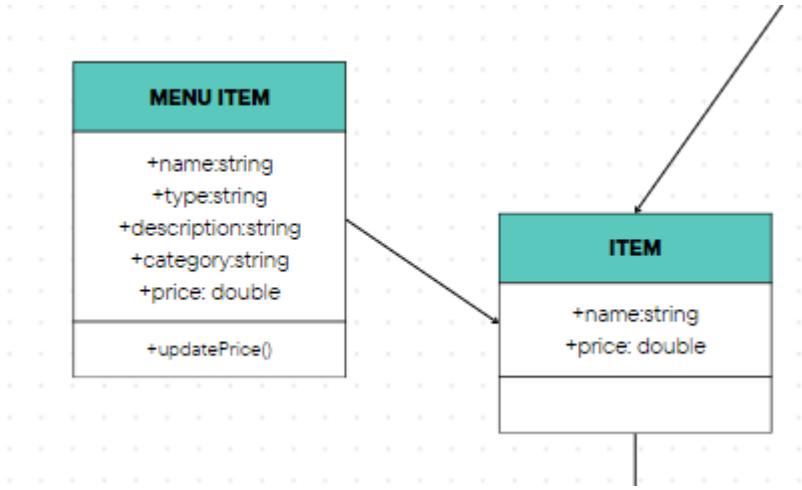
1) Single Responsibility

In this diagram, each class (`Customer` and `Item`) has a single responsibility. The `Customer` class handles customer information and actions, while the `Item` class manages item details.



2) Open/Close Principle:

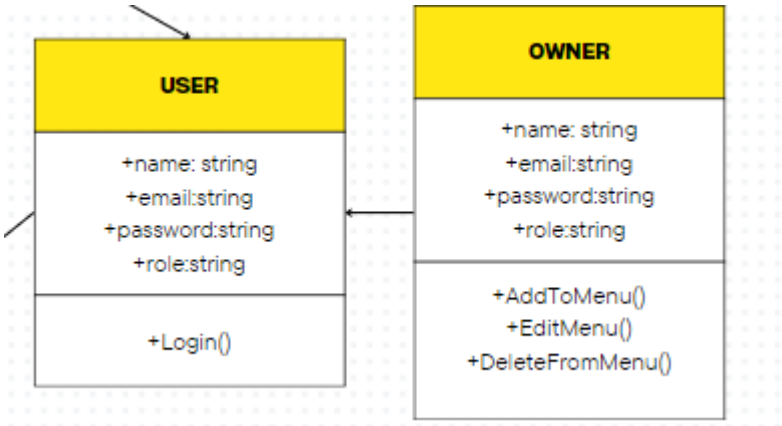
In this diagram, the `Item` class is open for extension but closed for modification. The `MenuItem` class extends the `Item` class and adds a new method `updatePrice()`, demonstrating the principle.



.

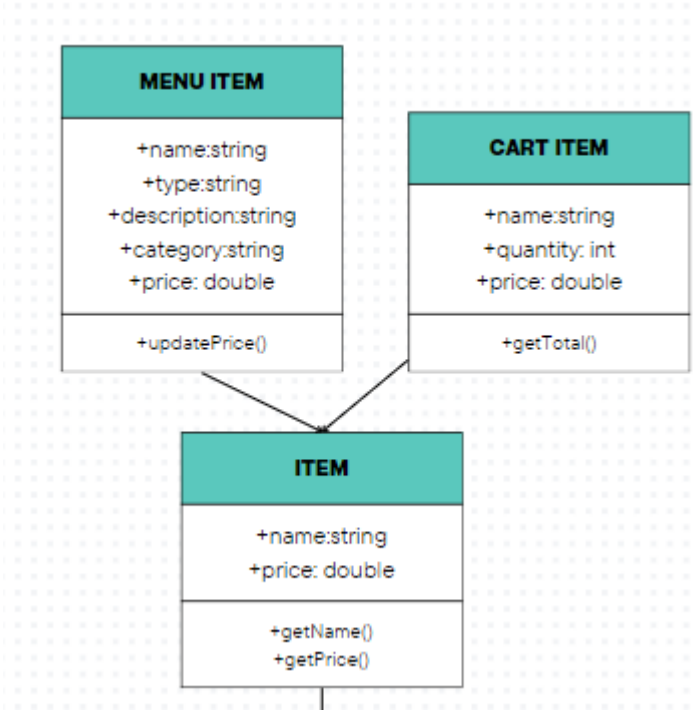
3) Liskov Substitution Principle (LSP)

In this diagram, the `Owner` class is a subtype of the `User` class. The Liskov Substitution Principle holds that objects of the `Owner` class can substitute objects of the `User` class without affecting the correctness of the program.

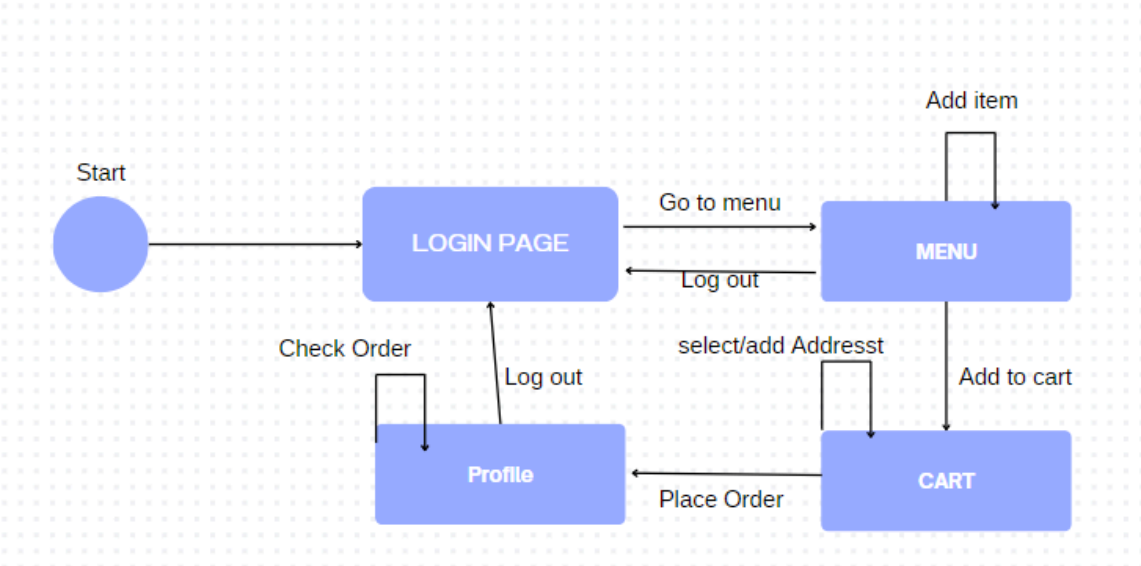


4) Interface Segregation Principle:

Common behaviors among `CartItem` and `MenuItem` are defined in the Item superclass to avoid imposing unnecessary dependencies on clients

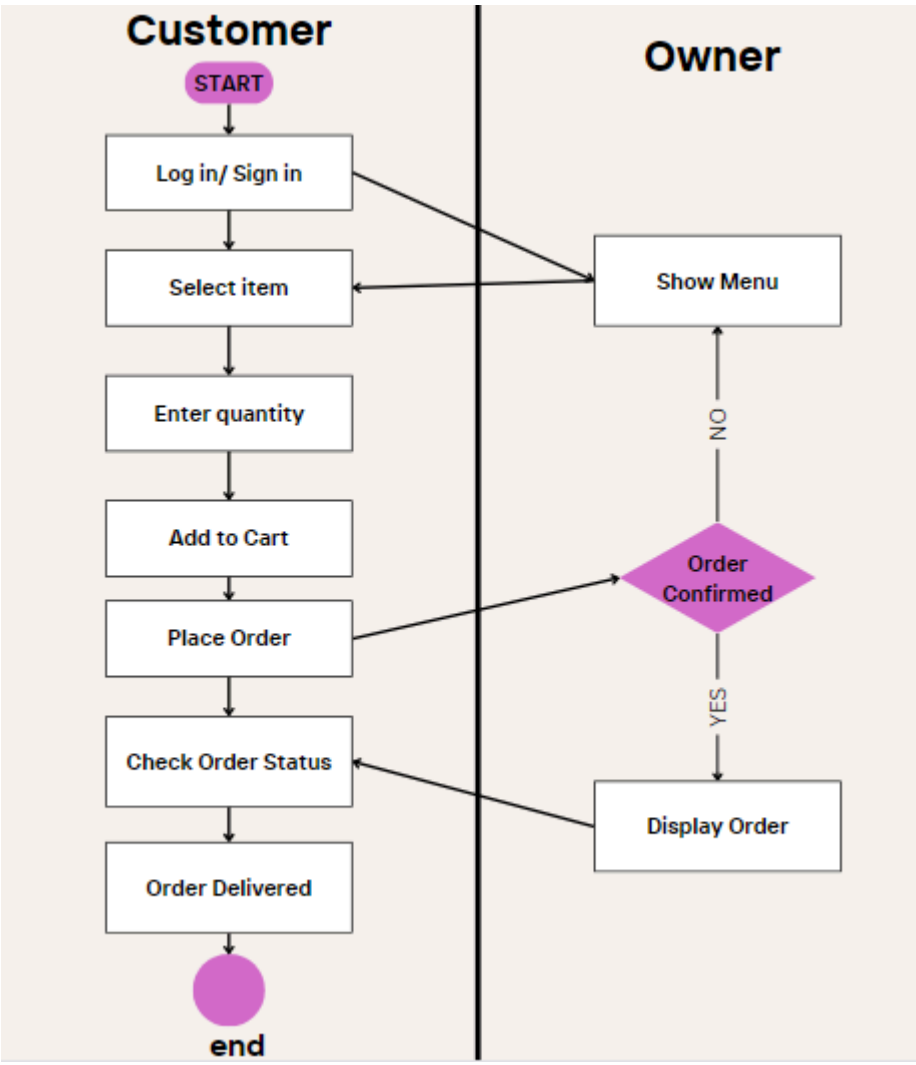


State Diagram:

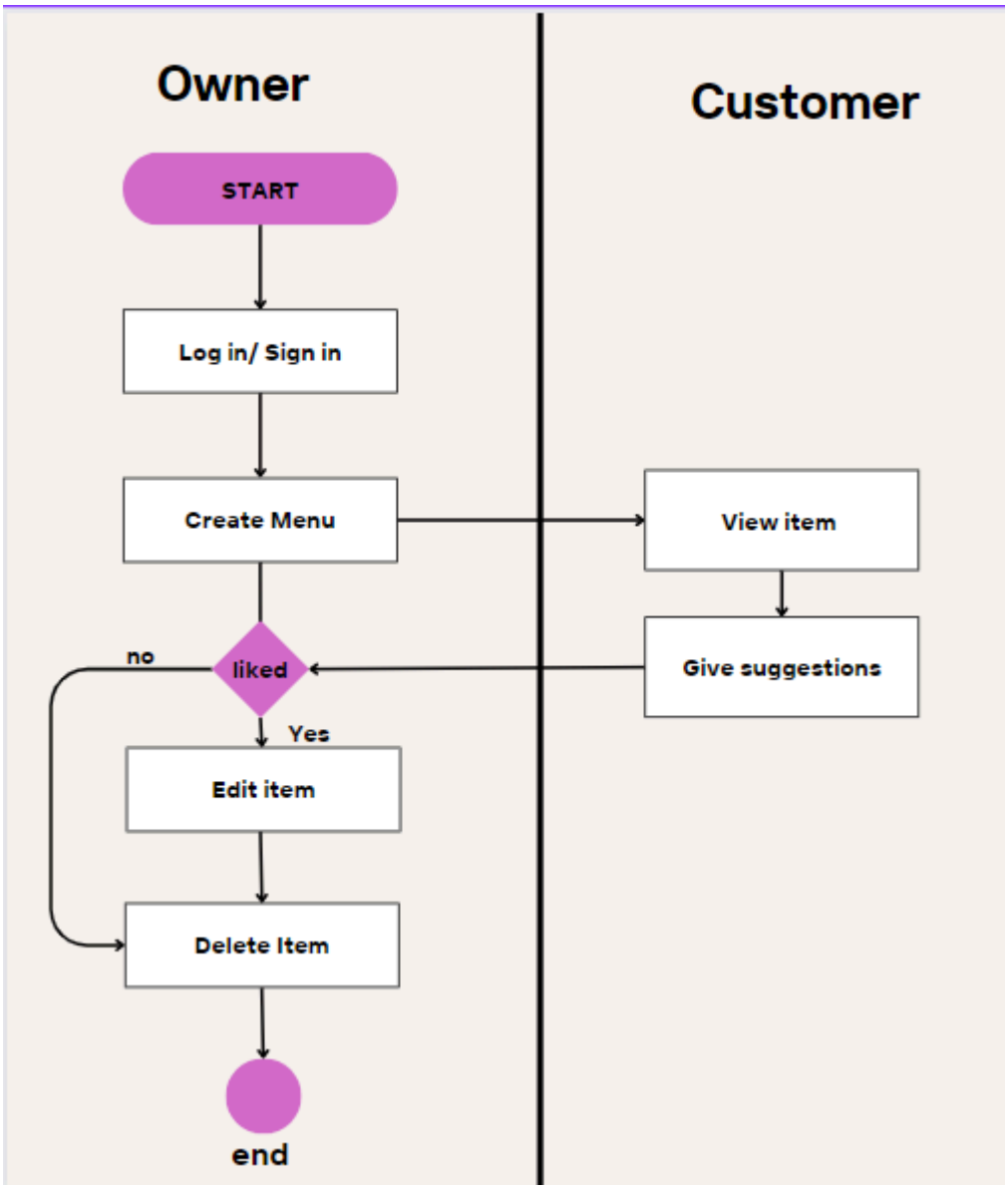


Activity Diagram:

Major function: Place Order



Minor function: Update Menu


Owner
Customer
START
Log in/ Sign in
Create Menu
View item
Give suggestions
liked
no
Yes
Edit item
Delete Item
end

Architecture Patterns:

1. Model:
    - The Model encapsulates the data and behavior of the application and provides an interface to access and manipulate that data.
    - It responds to requests for information, changes in state, and instructions to perform operations.
    - The `Model` class encapsulates methods like `GetUserDetails()`, `GetMenuDetails()`, and `GetOrderDetails()`, providing access to user, menu, and order data.
    -
2. View:
    - The View is responsible for presenting data to the user and handling user interactions. It displays the Model's data to the user and provides means for users to interact with the application.
    - It observes changes in the Model and updates the user interface accordingly.
    - The `View` class contains methods like `renderPage()`, `displayData()`, and `updateView()`, responsible for rendering UI elements and updating the view based on changes in the Model.

3. Controller:

- The Controller acts as an intermediary between the Model and the View. It receives user input, processes it, and invokes appropriate actions on the Model or View.
- It interprets user actions, initiates requests to the Model or View, and updates the View based on changes in the Model.
- The `Controller` class contains methods like `authenticateUser()`, `createOrder()`, `calculateBill()` and `manageMenu()`, handling user interactions and invoking corresponding actions on the Model or View.

Design Principles:

1) Single Responsibility Principle (SRP):
    - This principle states that a class should have only one reason to change.
    - In the given diagram, each class seems to have a clear responsibility:
        - Customer: Handles customer information and actions like login, adding items, and placing orders.
        - Menu: Manages menu items and their details.
        - Model: Handles retrieving user, menu, and order details.
        - Owner: Manages owner information and actions related to menu management.
        - Order: Represents an order and its details, including order status.
        - Controller: Controls the flow of the application, including user authentication, order creation, bill calculation, and menu management.
        - View: Handles rendering and displaying data to the user interface.
        - Cart item: Represents an item in the user's cart and provides functionality to calculate total.
        - Menu Item: Represents a menu item with its details and provides functionality to update its price.

2) Open/Closed Principle (OCP):
    This principle states that classes should be open for extension but closed for modification

- For example, if new types of users are added, the `Customer` and `Owner` classes might need to be extended rather than modified.
- Similarly, if new types of menu items are introduced, the `Menu` and `Menu Item` classes might need extension.

3) Liskov Substitution Principle -
    - The Liskov Substitution Principle states that objects of a superclass should be replaceable with objects of its subclasses without affecting the correctness of the program.
    - Subclasses (`CartItem` and `MenuItem`) are substitutable for their superclass (`Item`) without affecting the correctness of the program.
    - Methods or properties defined in `Item` are appropriately implemented or overridden in its subclasses to maintain substitutability.

4) Interface Segregation Principle -
    - The Interface Segregation Principle states that clients should not be forced to depend on interfaces they don't use.
    - Common behaviors among `CartItem` and `MenuItem` are defined in the `Item` superclass to avoid imposing unnecessary dependencies on clients.
    - If specific behaviors are only relevant to either `CartItem` or `MenuItem`, they are segregated into separate interfaces to prevent clients from depending on irrelevant behaviors.

Design Patterns:

1. Factory Method Pattern:
    - The creation of instances of various classes such as `Customer`, `Owner`, `Order`, and others might benefit from the Factory Method pattern.
    - For example, a factory method could create instances of different user types (`Customer` and `Owner`) based on the provided role.
2. Observer Pattern:
    - The `View` class has methods for rendering, displaying data, and updating the view.

    - This suggests an Observer pattern where the view observes changes in the model and updates itself accordingly to reflect those changes.
3. Strategy Pattern:
    - The `Controller` class implements different strategies for authentication, order creation, bill calculation, and menu management.
    - By encapsulating each of these algorithms into separate classes and allowing the `Controller` to switch between them dynamically, the system becomes more flexible and easier to extend.
4. Decorator Pattern:
    - The `MenuItem` class, with its `updatePrice()` method, might benefit from the Decorator pattern.
    - Different decorators could be applied to `MenuItem` objects to modify their behavior or add additional functionalities dynamically.

Github link :

https://github.com/syed-hamza/java-principles-using-food-ordering-website (main project)

https://github.com/syed-hamza/zoggy_admin (admin panel)

Contribution of team members -

Hamza : Authentication : Log In and SignUp
Cart Page : Remove from cart, Calculate Bill
Address Page : Add Address, Delete Address

Sunidhi : Admin Panel : Display Menu
Create New Food Item
Edit or Update item
Delete food item

Stuthi : Menu Page : Display Menu

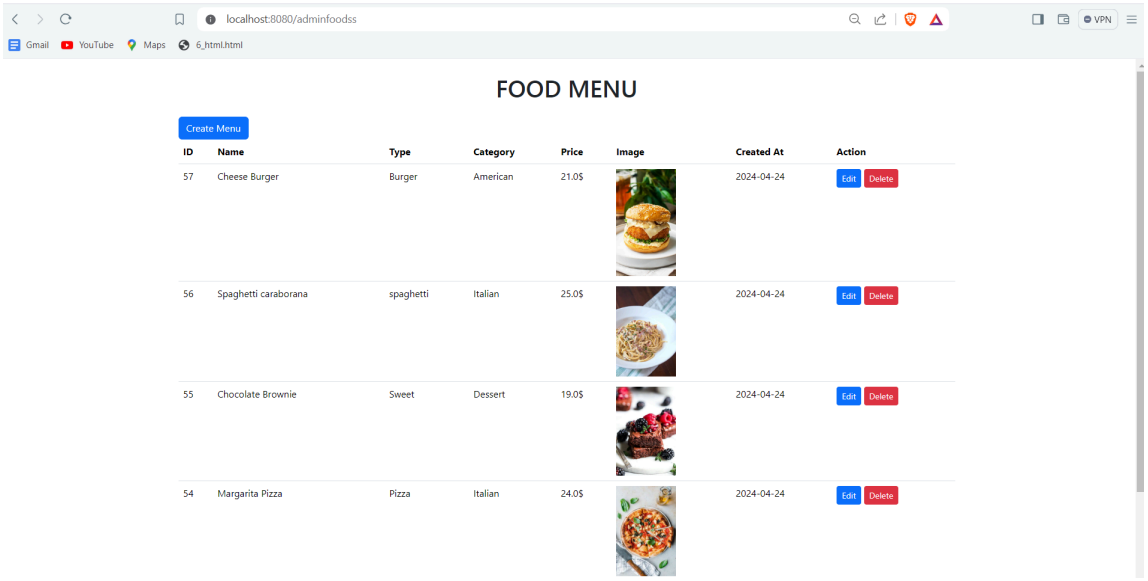
Add To Cart

Sujay : Profile Page: Display Order Status

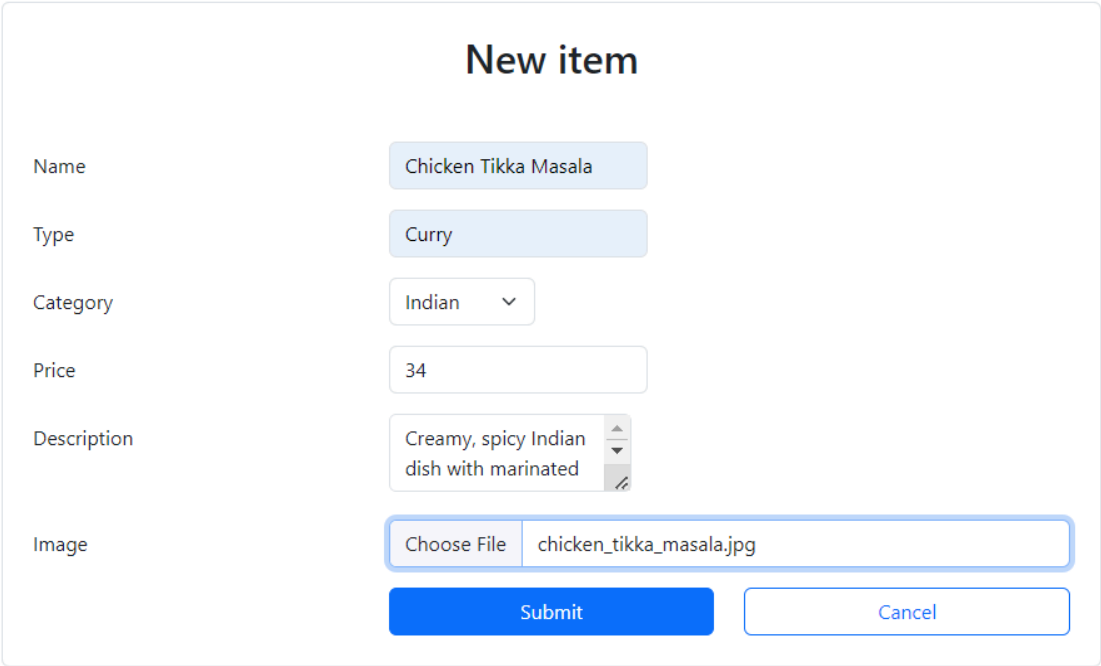
Screenshots of Input and Output -

Admin Panel:

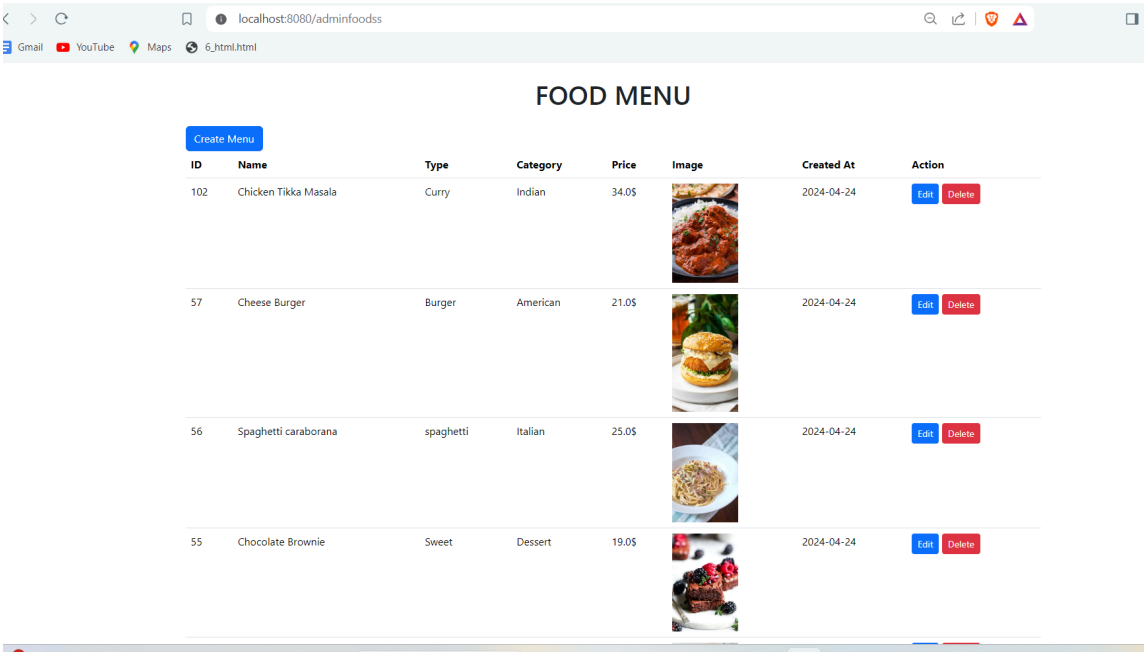
Display Menu:



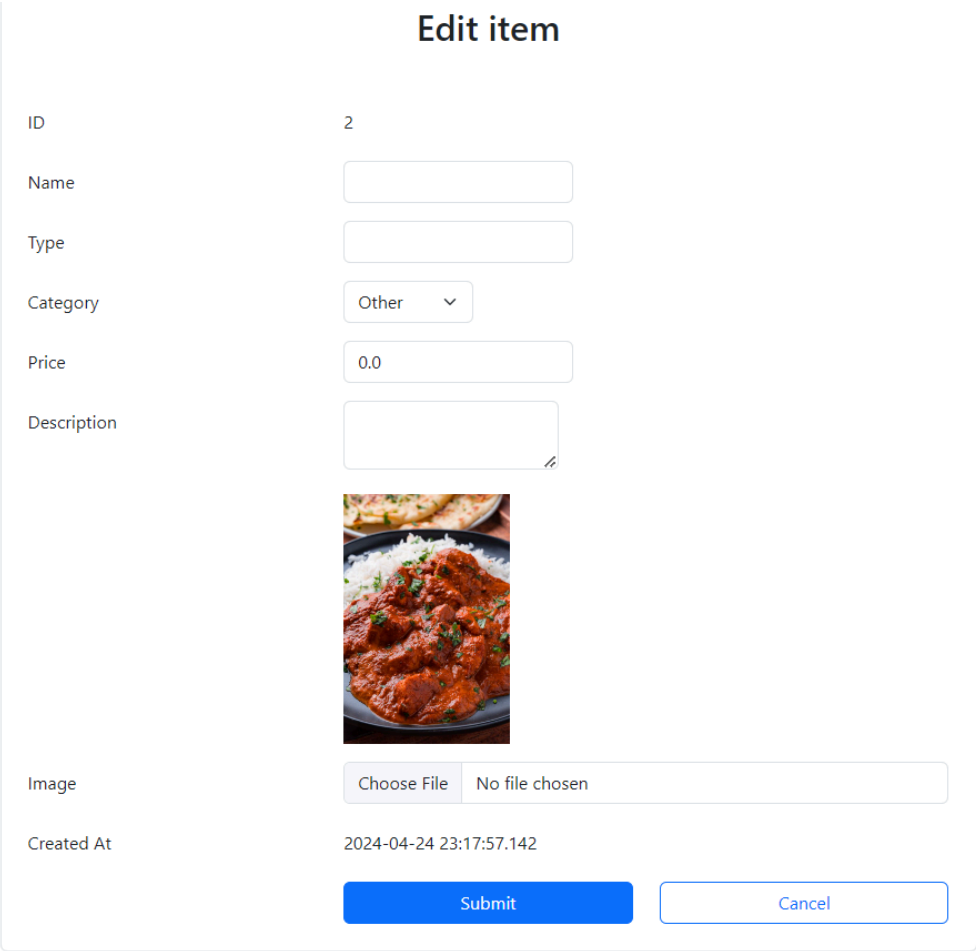
Create a Food Item:



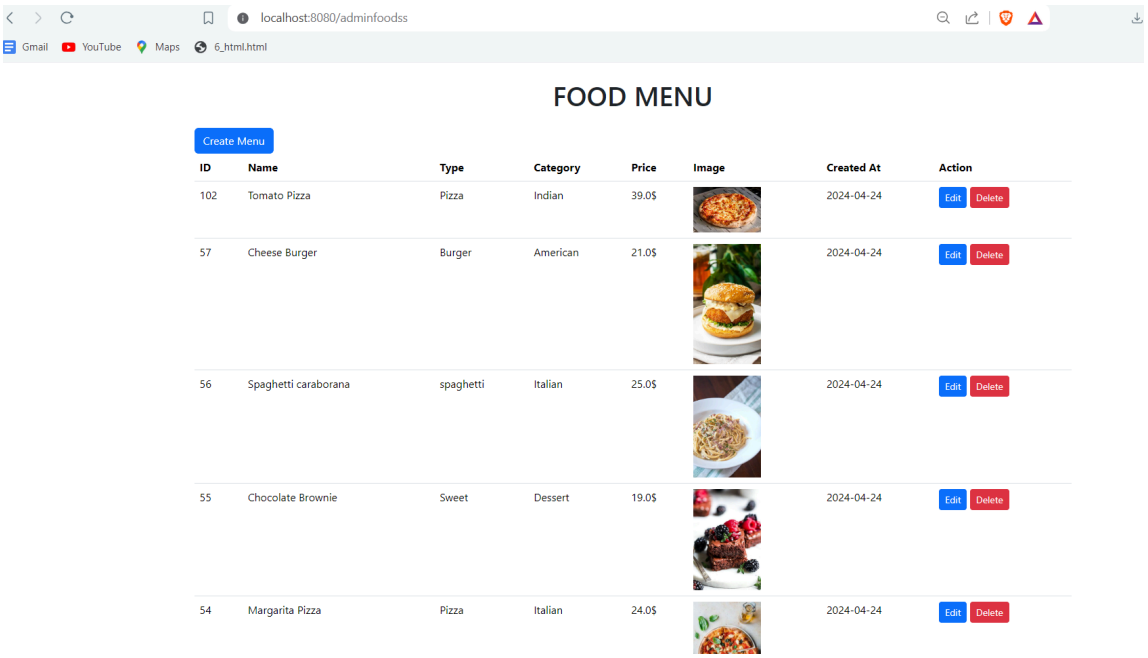
After Adding New Food Item:



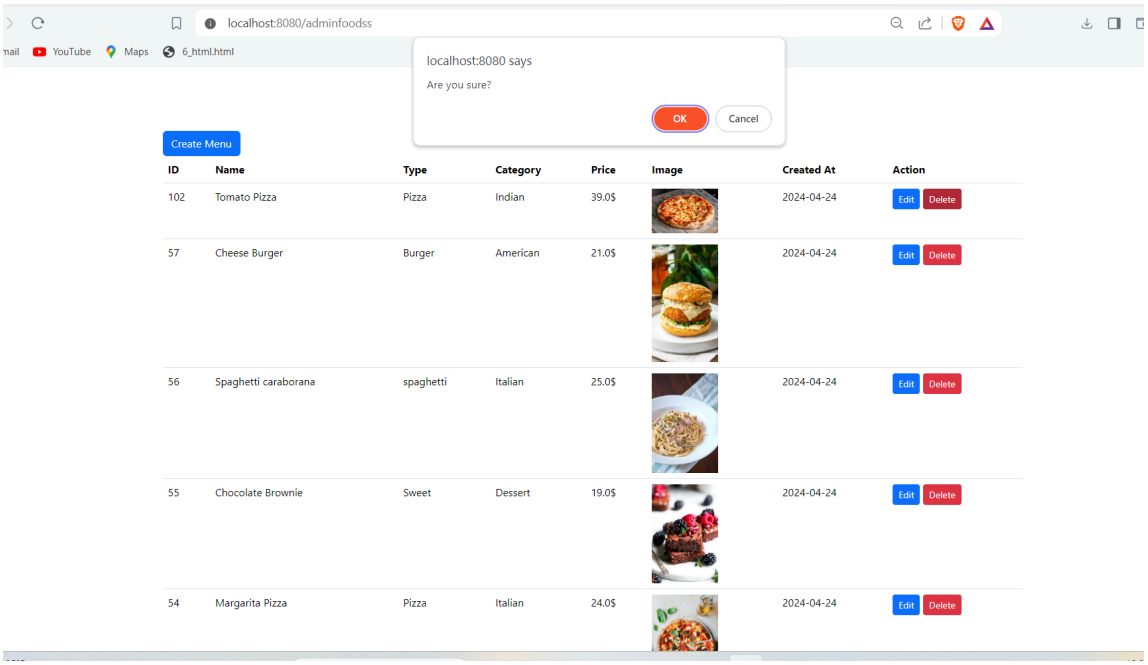
Editing The Food Item:



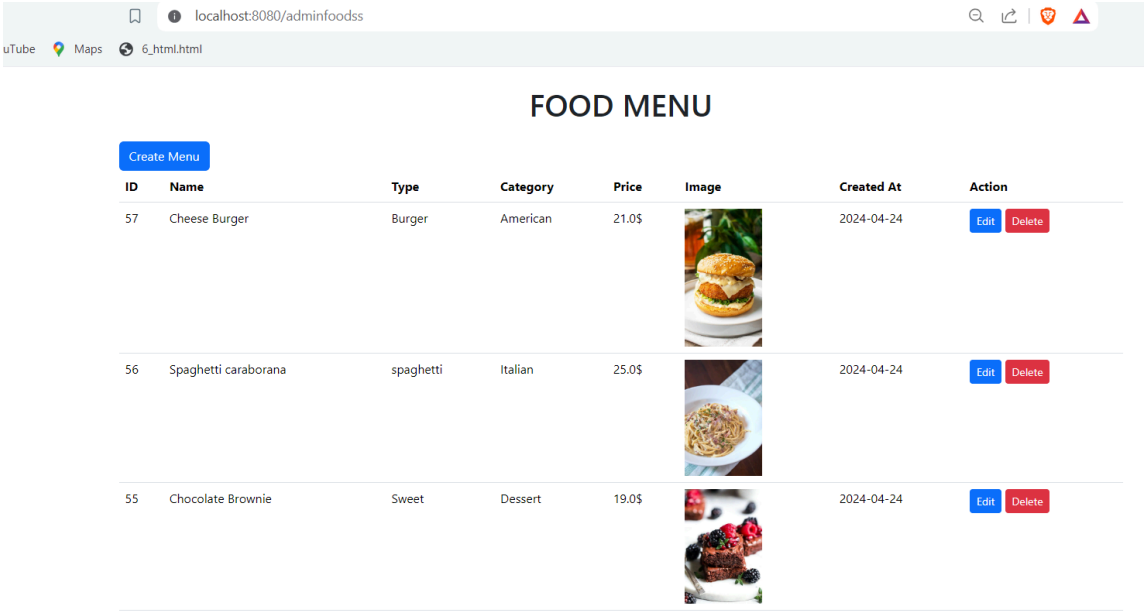
After the Edit:



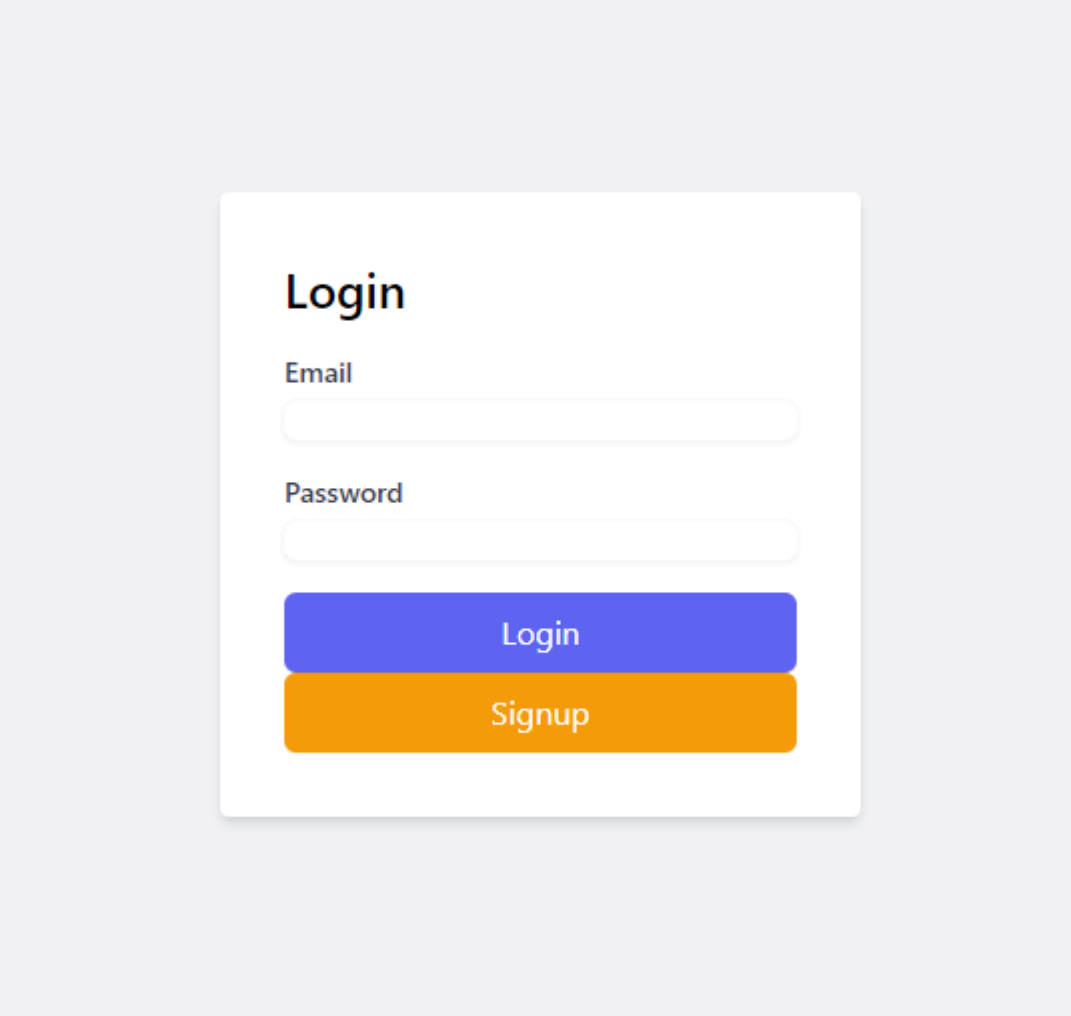
Delete an Item from Menu:



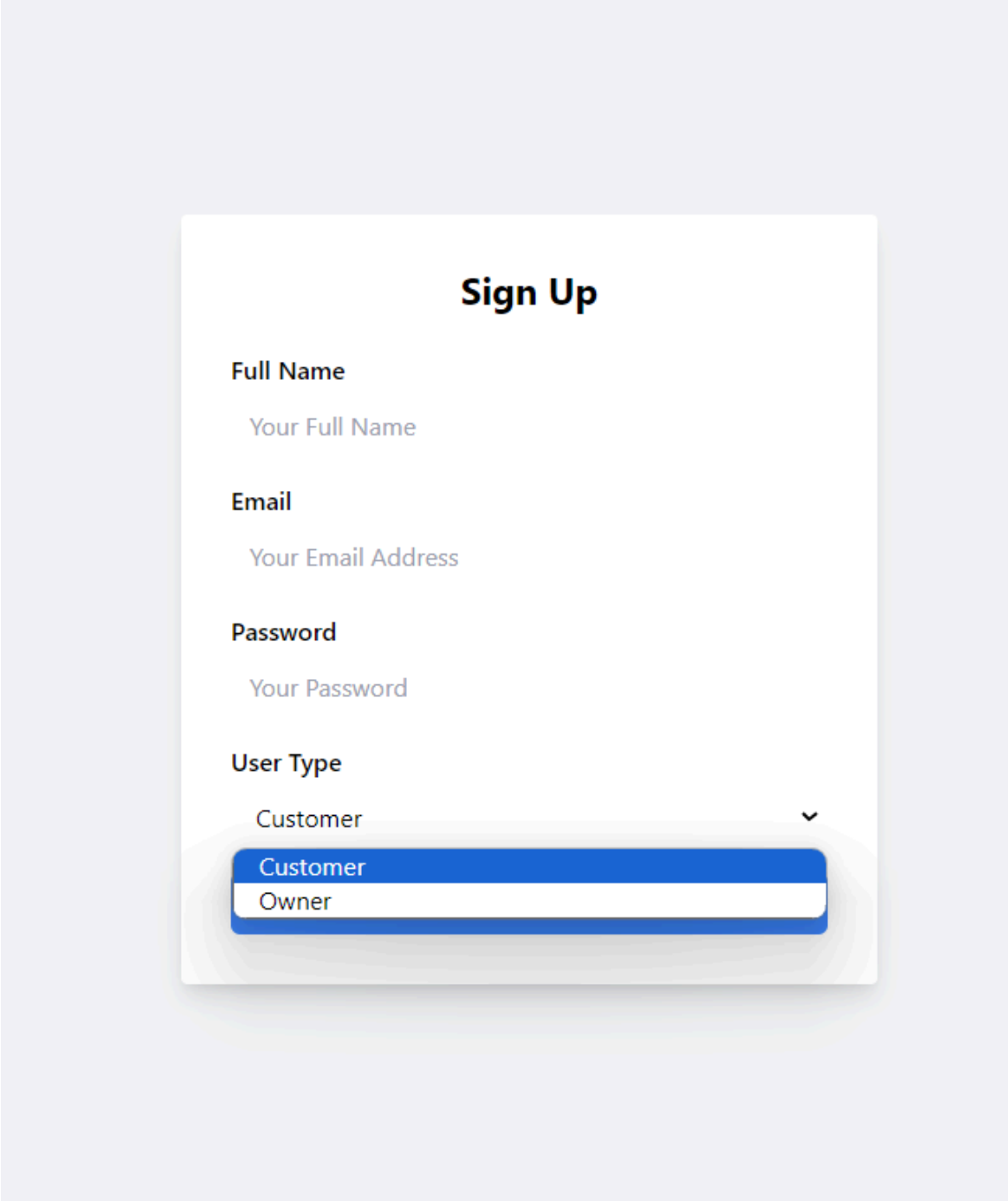
After Deleting:


localhost:8080/adminfoodss
uTube
Maps
6_html.html
FOOD MENU
Create Menu
ID
Name
Type
Category
Price
Image
Created At
Action
57
Cheese Burger
Burger
American
21.0$
2024-04-24
Edit
Delete
56
Spaghetti caraborana
spaghetti
Italian
25.0$
2024-04-24
Edit
Delete
55
Chocolate Brownie
Sweet
Dessert
19.0$
2024-04-24
Edit
Delete

Log In Page:

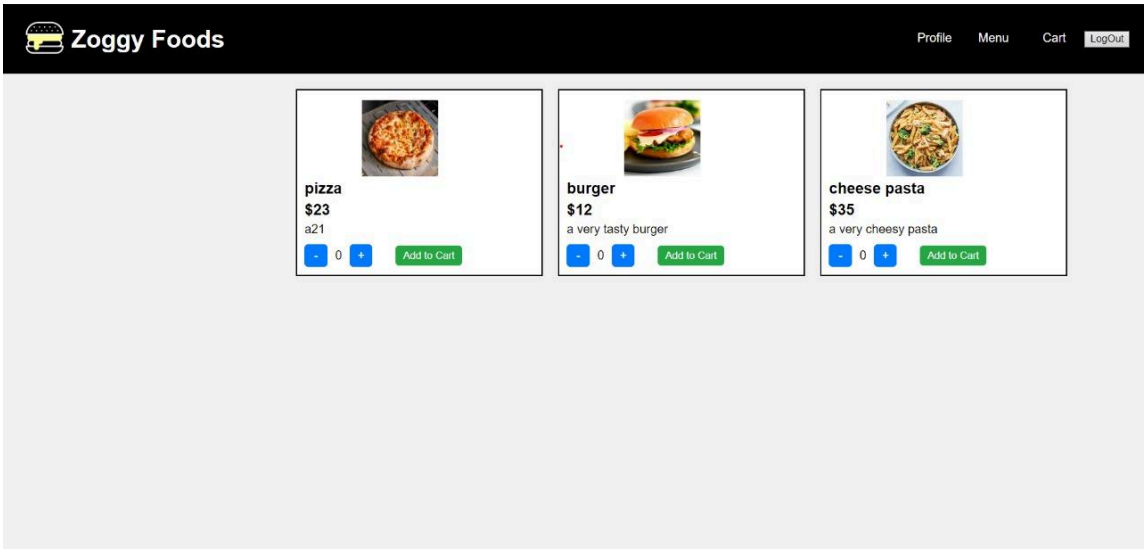


Sign Up Page:

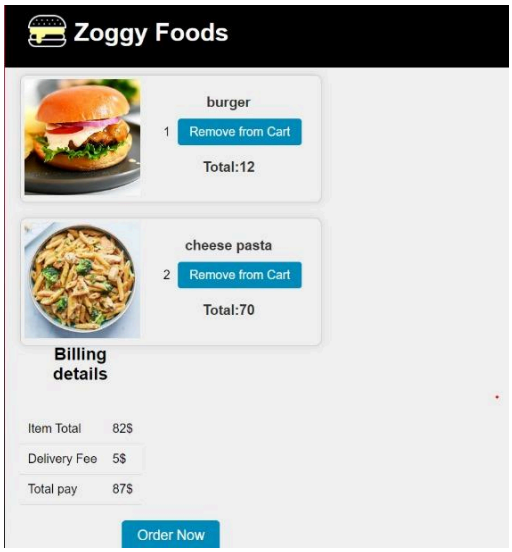

Sign Up
Full Name
Your Full Name
Email
Your Email Address
Password
Your Password
User Type
Customer
Customer
Owner

Home Page:

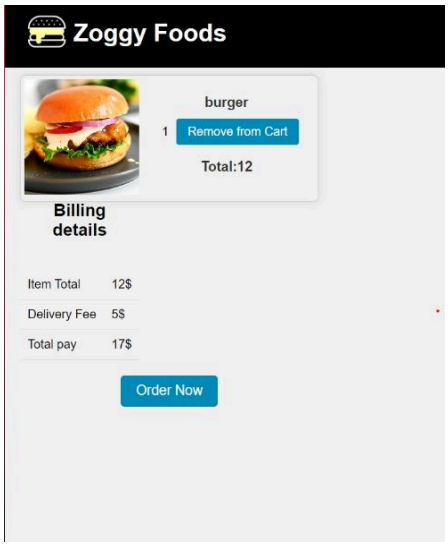
Menu Page:



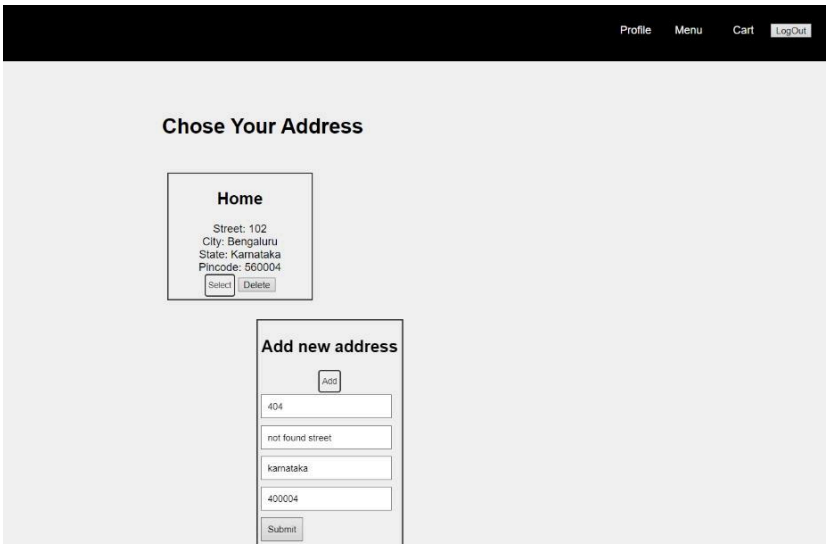
Cart Page (After add to cart):



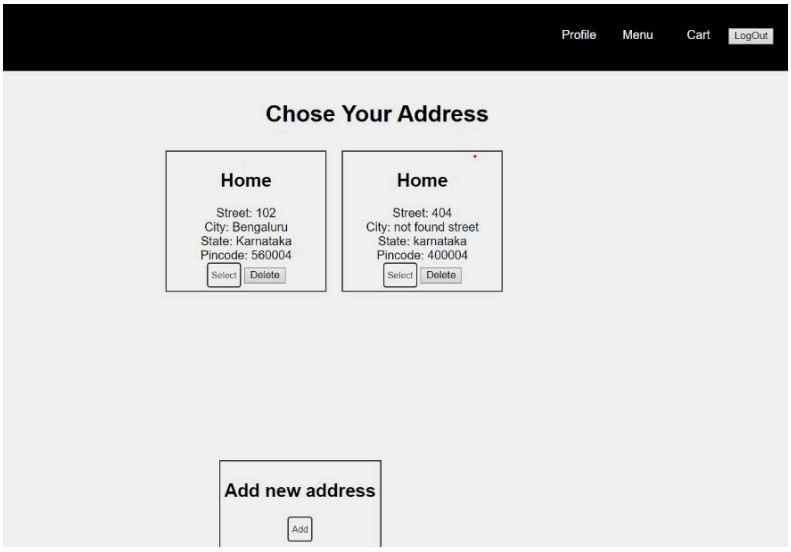
Cart Page (After remove from cart):



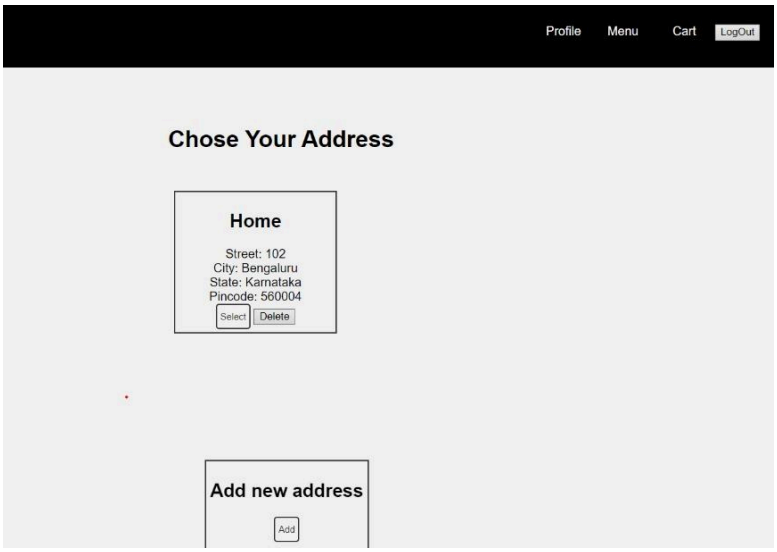
Address Page (Add address):



Display Addresses:



Address Page (Delete Address):


Profile
Menu
Cart
LogOut
Chose Your Address
Home
Street: 102
City: Bengaluru
State: Karnataka
Pincode: 560004
Select
Delete
Add new address
Add