# 

# CHAPTER 1

# INTRODUCTION

**INTRODUCTION**

ReactJS facilitates the creation of dynamic and responsive user interfaces, ensuring an engaging experience for customers navigating menus, placing orders, and tracking deliveries. Meanwhile, Spring Boot empowers the backend with scalable and high-performance services, allowing for the efficient management of multiple restaurant operations. With features such as multi-restaurant support, order management, and real-time updates, this system not only enhances the operational efficiency of each establishment but also promises a secure and scalable platform for the evolving landscape of the food service industry.

This system not only supports multiple restaurants, each with its unique menu and parameters, but also prioritizes order management, user authentication, and real-time updates. Scalable and flexible, our Multi-Restaurant Management System is poised to elevate restaurant operations, offering a comprehensive solution that embraces innovation and efficiency in the ever-evolving culinary landscape.

# 1.1 Purpose

The purpose of our Multi-Restaurant Management System, crafted with ReactJS and Spring Boot, is to revolutionize and optimize the complexities of running multiple restaurants seamlessly. By leveraging ReactJS for the frontend, we ensure an intuitive and engaging user interface that facilitates effortless navigation and order placement for customers. Simultaneously, the robust Spring Boot backend ensures reliability and efficiency in managing the diverse operations of multiple establishments. This system streamlines order processing, enhances user experiences, and provides real-time insights into restaurant performance. Through the integration of ReactJS and Spring Boot, our aim is to empower restaurant owners with a scalable, flexible, and technologically advanced solution that not only meets but exceeds the demands of the dynamic and competitive food industry, ultimately leading to enhanced customer satisfaction and operational excellence.

# 1.2 Intended Audience

The multi Restaurant Management System, developed with React.js and Spring Boot, is specifically tailored for restaurant owners, managers, and staff seeking a technologically advanced solution to streamline their operations. This system caters to individuals and teams responsible for order management, menu customization, and overall restaurant administration. Restaurant owners can benefit from the scalability and adaptability of the system, allowing them to efficiently manage their establishments and stay ahead of industry trends. Managers will appreciate the real-time communication and dynamic interfaces, facilitating seamless order tracking and enhancing customer interactions. Meanwhile, staff members involved in day-to-day operations will find the system user-friendly and efficient, ultimately contributing to enhanced productivity and service quality. Overall, this solution targets the diverse needs of the restaurant industry, offering a comprehensive toolset to optimize processes and improve the overall dining experience for customers.

**1.3 Scope**

# The scope of the Online Restaurant Management System developed with React.js and Spring Boot encompasses a wide range of functionalities aimed at revolutionizing the restaurant industry. This system provides a comprehensive platform for managing orders, customizing menus, and overseeing various aspects of restaurant operations. With a dynamic and responsive front-end powered by React.js, the system ensures an intuitive user interface for both restaurant staff and customers. The robust backend capabilities of Spring Boot contribute to scalable, secure, and efficient data processing, enabling real-time communication and streamlined workflows. The scope extends to features such as secure user authentication, adaptable menu management, and responsive design for cross-platform accessibility. This system not only addresses the immediate needs of order processing and menu customization but also positions the restaurant to adapt to evolving industry trends, enhance operational efficiency, and elevate the overall customer experience. Its scalability and flexibility make it a versatile solution for restaurants of varying sizes and operational models.

# CHAPTER 2

# LITERATURE SURVEY

# 2. LITERATURE SURVEY

* Soon Nyean Cheong,et al[1] Orders made by the customers will be updated instantly to a centralized database and subsequently reach the cashier and the kitchen module respectively.
* Deksne,et al[2] Service time is one of the main criteria that can be improved to enhance the speed of the customer service as well as to increase the number of restaurant visitors.
* Jain,et al[3] It will considerably reduce human efforts. Customers will be able to place their orders a automatically with the help of the ordering system and a menu card kept on their table.
* Bankar,et al[4] In most of the restaurant meal ordering is relying on the interaction with waiters to place order into the kitchen. In busy hours of restaurant this coordination is a challenge result in un-satisfaction to the customer.
* Kocaman,et al[5] A large number of RMSs are commercially available in the market with their ever-developing features.

# 

**CHAPTER 3**

**ANALYSIS**

**3.ANALYSIS**

# 

# 3.1 Existing System:-

* + - **Square for Restaurants**
    - **Restaurant Management**
    - **Order and Delivery Management**
    - **OpenTable**
    - **Shop Keep**

**Square for Restaurants:**

Square for Restaurants provides a range of tools for managing orders, payments, and menus. It integrates with various Square hardware and software solutions.

**Restaurant Management:**

* + Menu management: Adding, updating, and removing items.
  + Order management: Receiving and processing orders.
  + Table reservation system (if applicable).

**Order and Delivery Management:**

* + - Placing and tracking orders.
    - Assigning and managing delivery personnel.
    - Real-time order tracking for customers.

**OpenTable:**

OpenTable is primarily known for its reservation system, allowing restaurants to manage bookings efficiently. It also offers guest management and analytics tools.

**Shop Keep:**

Shop Keep is a POS system with features for inventory management, staff tracking, and customer analytics, catering to the needs of small and medium-sized restaurants.

* + 1. **Disadvantages for Existing System**
    - Initial Cost and Implementation Time
    - Require more staff
    - Customization Limitations
    - Maintenance and Support
    - Vendor Reliability
    - Operational costs
    - Space limitations
    - Real estate challenges
    - Limited audience reach

**Initial Cost and Implementation Time:**

Implementing a new online restaurant management system can involve significant upfront costs, including software licenses, hardware, and training. The initial setup and integration may also take time, causing potential disruptions to daily operations.

**Require more staff:**

In this system require more employees they are waiters, and staff.

**Customization Limitations:**

Some off-the-shelf systems might have limitations in terms of customization. Restaurants with unique or specific requirements may find it challenging to adapt the system to their exact needs.

**Ongoing Maintenance and Support:**

Continuous updates, maintenance, and support are necessary for the smooth functioning of the system. Some businesses may find it challenging to keep up with regular updates or may face delays in receiving support from the software provider.

**Vendor Reliability:**

The reliability of the software vendor is crucial. If the vendor faces financial instability or goes out of business, it could result in discontinued support, updates, or service, leaving the restaurant in a difficult position.

**Operational costs:**

Maintaining a physical restaurant involves significant expenses such as rent, utilities, staff wages, inventory, and equipment maintenance. These costs can eat into profits, especially during slow periods or economic downturns.

**Space limitations:**

Physical restaurants have limited seating capacities, which can result in turning away potential customers during peak hours. Expanding the space might not always be feasible due to cost or space constraints.

**Real estate challenges:**

Securing and maintaining a prime location for a restaurant can be costly and competitive. Additionally, changes in the surrounding neighbourhood or property value can affect the restaurant's sustainability.

**Limited audience reach:**

Physical restaurants are constrained by their location, limiting their accessibility to a specific local or regional audience. They might not be able to cater to customers beyond that area without expanding or franchising.

**3.2 PROBLEM STATEMENT**

In the rapidly evolving landscape of the hospitality industry, traditional restaurant management systems often struggle to meet the dynamic needs of modern establishments. Current manual processes, paper-based order management, and disjointed communication channels result in inefficiencies, increased operational costs, and a diminished overall customer experience. The lack of a comprehensive online restaurant management system contributes to challenges such as order processing delays, inventory discrepancies, and an inability to adapt quickly to changing market demands. Additionally, the absence of real-time data analytics impedes strategic decision-making for restaurant owners and managers. There is a critical need for an integrated solution that seamlessly connects all aspects of restaurant operations, from order placement to inventory management, to enhance efficiency, reduce costs, and elevate the overall dining experience for both customers and staff.

This problem statement identifies key issues such as manual processes, communication challenges, operational inefficiencies, and the absence of real-time data analytics. It sets the stage for the development of an online restaurant management system that addresses these challenges and provides a comprehensive solution for streamlining restaurant operations.

### **Proposed System:-**

* + - User Authentication and Authorization
    - Order Management Interface
    - Menu Customization
    - Customer Interaction
    - Responsive Design
    - API Development
    - Database Integration
    - Security Measures
    - Logging and Monitoring

**User Authentication and Authorization:**

Implement a secure user authentication system to ensure that only authorized personnel have access to the system. This may include roles for administrators, managers, and staff.

**Order Management Interface:**

Develop an intuitive order management interface that allows staff to view and process incoming orders efficiently. Implement features for order tracking, status updates, and order history.

**Menu Customization:**

Utilize React.js to create a flexible and dynamic menu management system. Allow restaurant owners to easily customize menus, add new items, update prices, and manage categories.

**Customer Interaction:**

Implement features that enhance customer interaction, such as order confirmation notifications, feedback forms, and the ability for customers to view their order history.

**Responsive Design:**

Ensure a responsive design that provides a consistent and enjoyable experience across various devices, including desktops, tablets, and smartphones.

**API Development:**

Develop RESTful APIs to facilitate communication between the frontend and backend. Define endpoints for user authentication, order management, menu customization, and other key functionalities.

**Database Integration:**

Integrate a relational database (e.g., MySQL) to store data such as user information, orders, menus, and customer feedback. Utilize Spring Data JPA for seamless database interactions.

**Security Measures:**

Implement security measures using Spring Security to protect sensitive data and ensure secure user authentication and authorization. Use encryption for data transmission and storage.

**Real-time Communication:**

Utilize WebSocket technology to enable real-time communication between the frontend and backend. This is particularly useful for instant updates in order processing and status changes.

**Scalability:**

Design the system with scalability in mind, allowing it to handle a growing number of users, orders, and transactions. Use Spring Boot's scalability features to optimize performance.

**Logging and Monitoring:**

Implement logging and monitoring features to track system activities, detect potential issues, and facilitate troubleshooting. Utilize Spring Boot Actuator for monitoring endpoints.

**3.3.1 Advantages of proposed system**

* User-Friendly Interface
* Real-Time Updates
* Scalability and Performance
* Security Measures
* Dynamic Menu Customization
* Responsive Design
* Efficient Order Management

**User-Friendly Interface:**

The use of React.js enables the creation of a highly responsive and user-friendly interface. This ensures an intuitive and engaging experience for both restaurant staff and customers, contributing to increased usability and satisfaction.

**Real-Time Updates:**

Leveraging WebSocket technology in Spring Boot facilitates real-time communication between the frontend and backend. This allows for instant updates on order status, menu changes, and other relevant information, enhancing operational efficiency.

**Scalability and Performance:**

The Spring Boot backend provides a robust and scalable foundation. It is designed to handle growing data loads and increasing transaction volumes, ensuring optimal performance as the restaurant business expands.

**Security Measures:**

Spring Security is utilized to implement robust security measures. This includes secure user authentication, authorization, and encryption for data transmission and storage, ensuring the protection of sensitive information.

**Dynamic Menu Customization:**

React.js is employed to create a dynamic and customizable menu management system. Restaurant owners can easily customize menus, add new items, and update prices, providing flexibility to adapt to changing preferences.

**Responsive Design:**

The system incorporates a responsive design using React.js, ensuring a consistent and enjoyable experience across various devices. This responsiveness is crucial for staff managing orders on desktops and customers placing orders via mobile devices.

**Efficient Order Management:**

The proposed system includes an intuitive order management interface, allowing staff to efficiently process and track orders. Real-time updates and status changes contribute to smoother order processing.

**CHAPTER4**

**SYSTEM ARCHITECTURE**

**4. SYSTEM ARCHITECTURE:**

### **UML DIAGRAMS**

UML is a method for describing the system architecture in detail using the blueprint. UML represents a collection of best engineering practices that have proven successful in the modeling of large and complex systems. UML is a very important part of developing objects-oriented software and the software development process. UML uses mostly graphical notations to express the design of software projects. Using the UML helps project teams communicate, explore potential designs, and validate the architectural design of the software.

**Definition:** UML is a general-purpose visual modeling language that is used to specify, visualize, construct, and document the artifacts of the software system.

**UML is a Language**

It will provide vocabulary and rules for communications and function on conceptual and physical representation. So it is modeling language.

**UML Specifying**

Specifying means building models that are precise, unambiguous and complete. In particular, the UML address the specification of all the important analysis, design and implementation decisions that must be made in developing and displaying a software intensive system.

**UML Visualization**

The UML includes both graphical and textual representation. It makes easy to visualize the system and for better understanding.

### 

**4.1 USE CASE DIAGRAM**

### 

Fig 4.1 USE CASE DIAGRAM

### **4.2 SEQUENCE DIAGRAM**

**4.2.1Customer:**

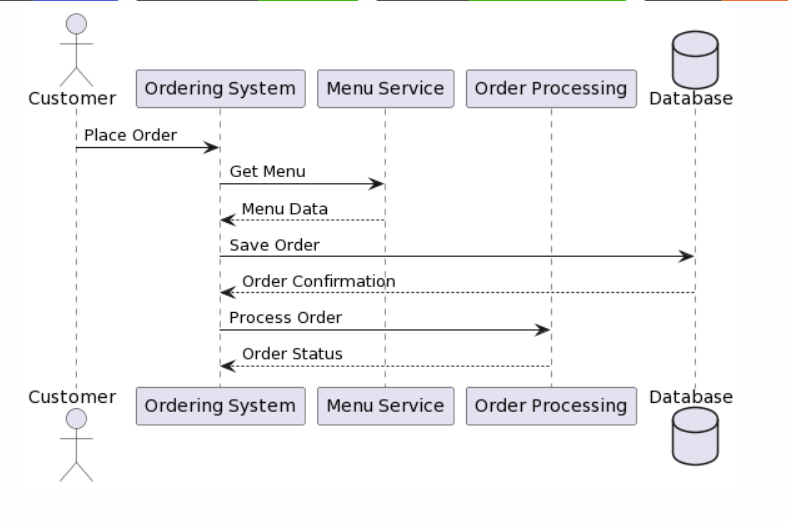


Fig 4.2.1 Customer

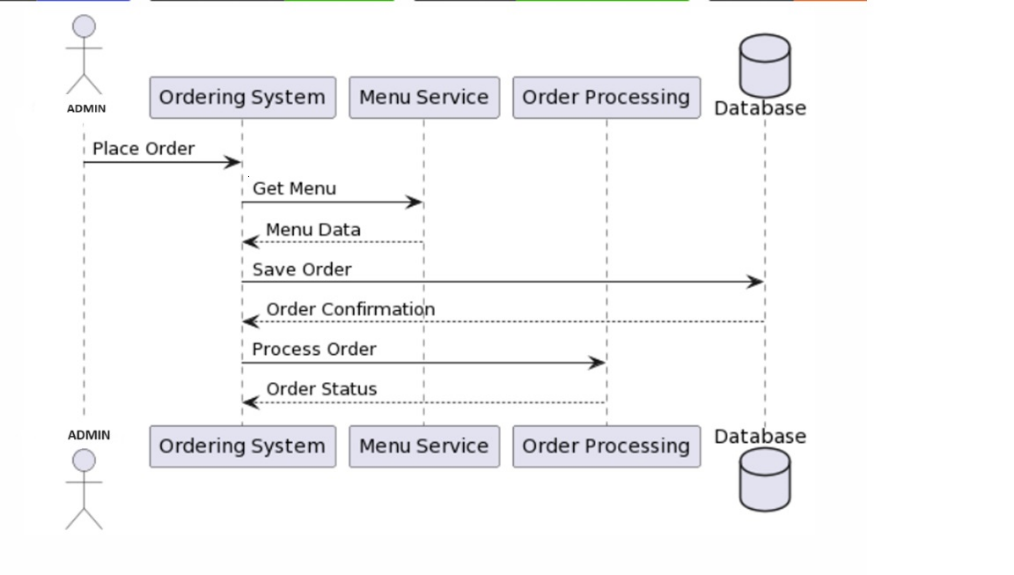
**4.2.2 Admin:**

Fig 4.2.2 Admin

**4.2.3 Restaurant:**

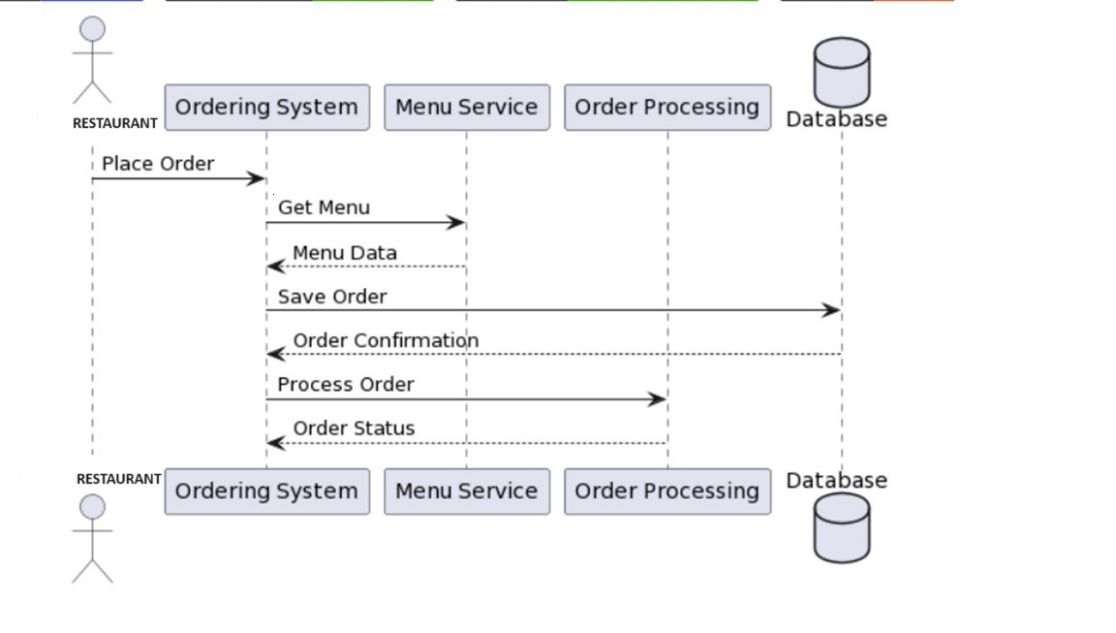


Fig 4.2.3 Restaurant

**4.2.4 Delivery person:**

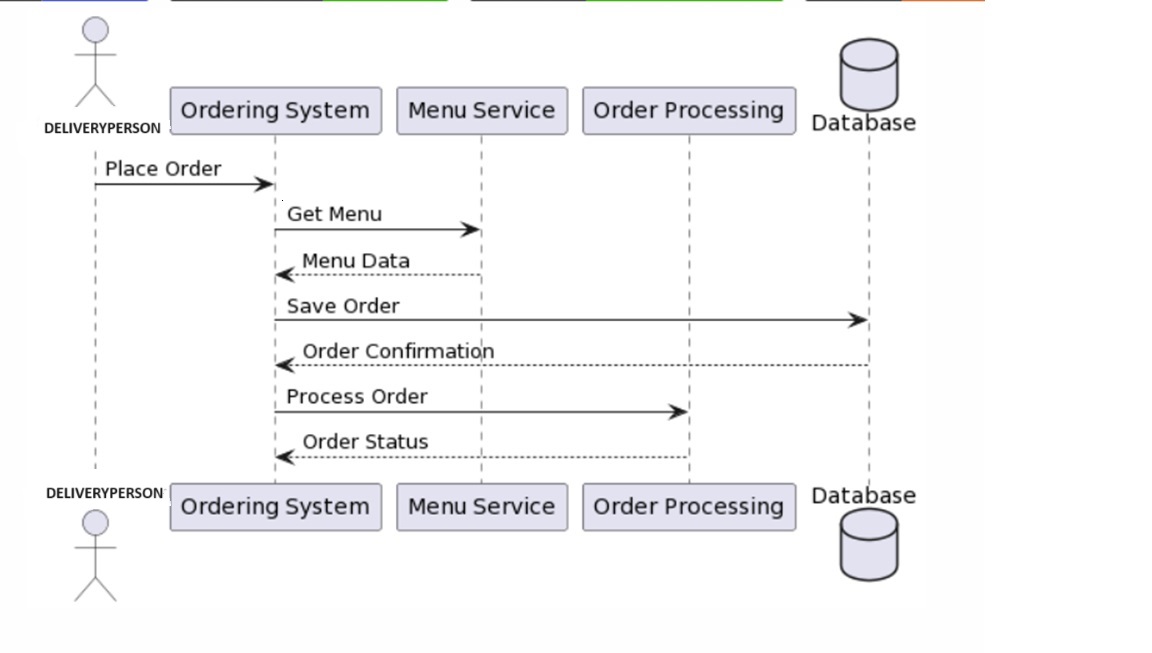
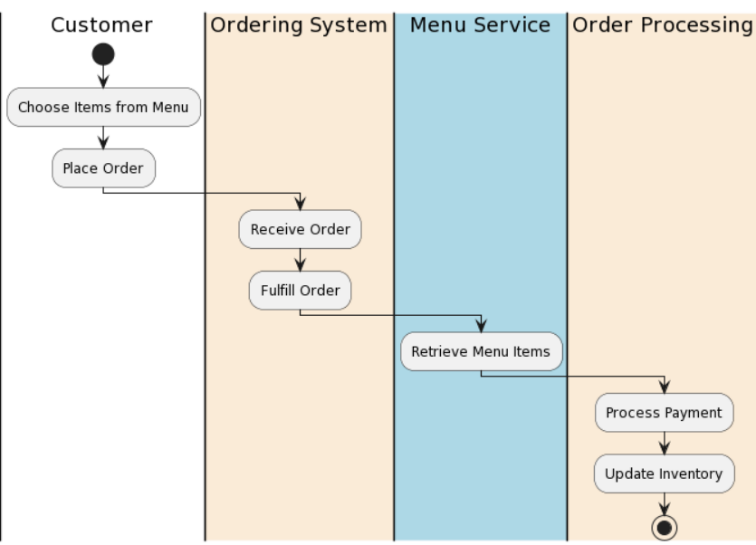


Fig 4.2.4 Delivery person

**4.3 ACTIVITY DIAGRAM:**

**4.3.1Customer:**



## 

## Fig 4.3.1 Customer

## **4.3.2 Admin:**

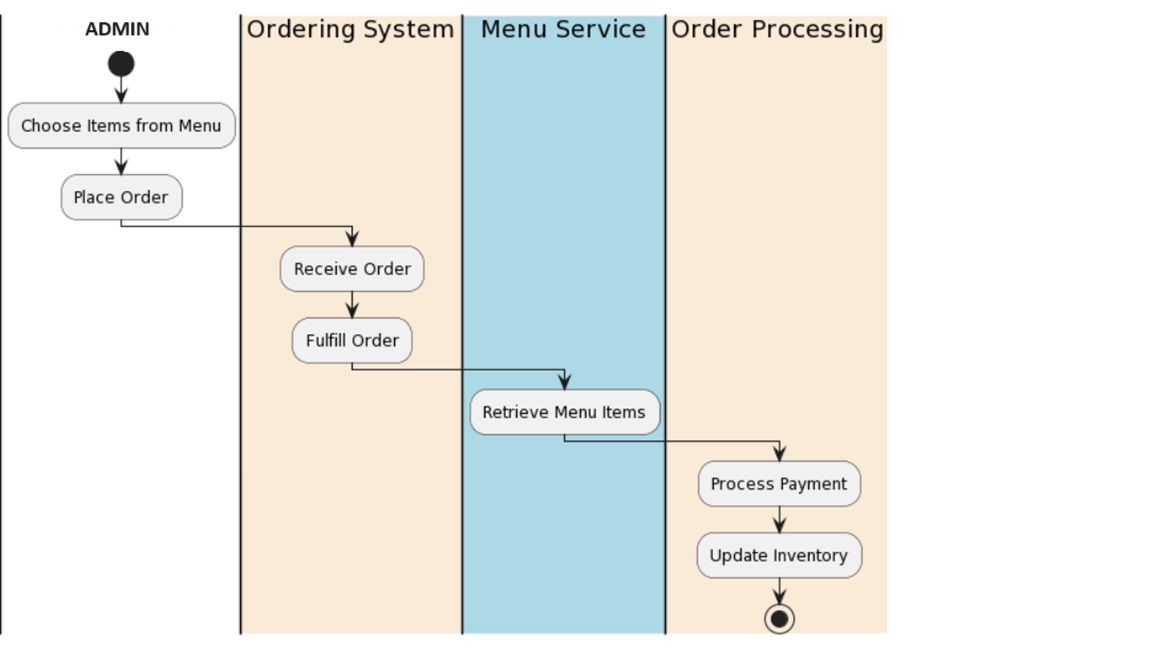


Fig 4.3.2 Admin

## **4.3.3 Restaurant:**

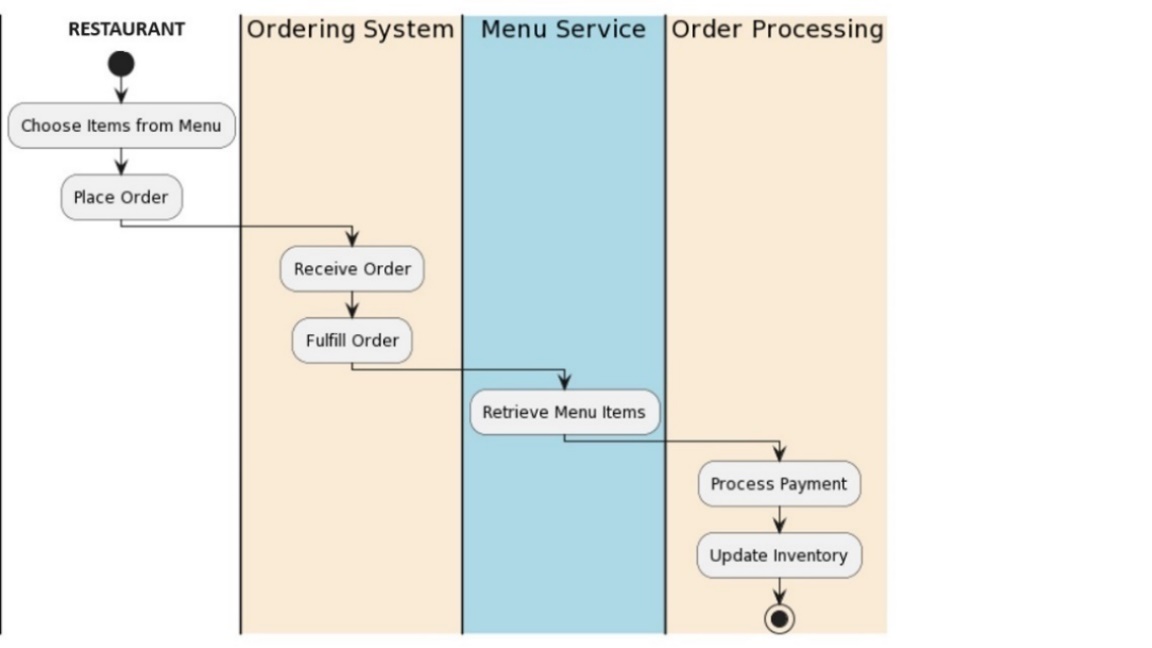


Fig 4.3.3 Restaurant

**4.3.4 Delivery person:**

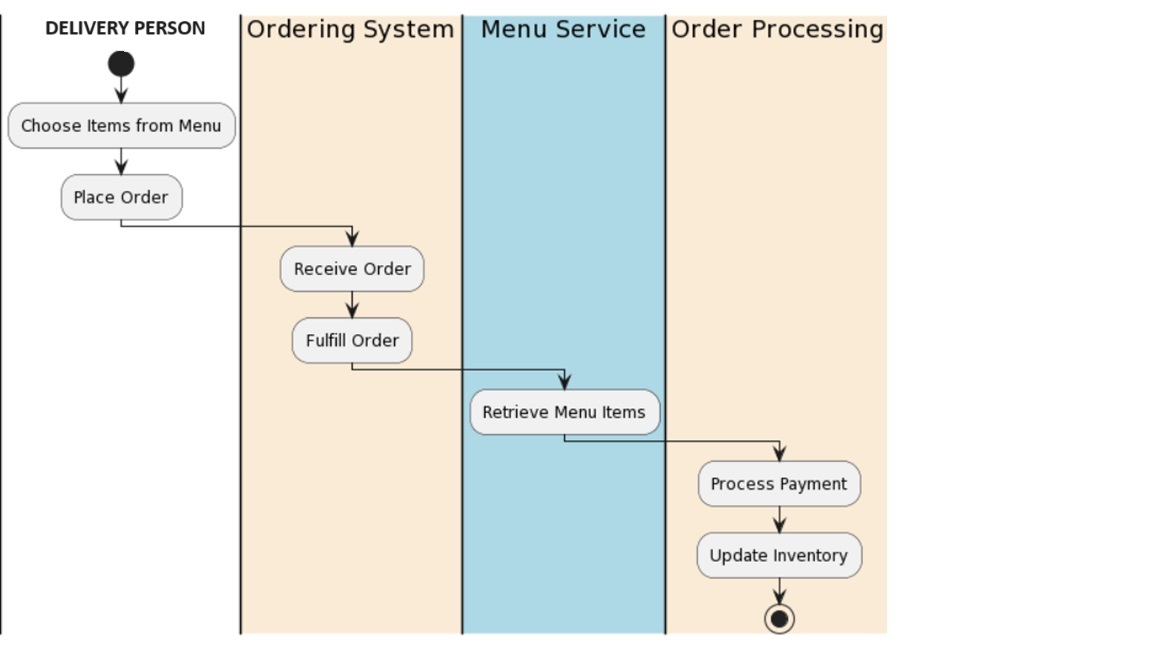


Fig 4.3.4 Delivery person

## **4.4 FLOW CHART:**

# 

# Fig 4.4 flowchart

# 4.5 System configuration:

**4.5.1 Software requirements:**

* + Frontend (React.js)
  + Code Editor
  + Node.js
  + Create React App

**Frontend (React.js):**

Install the React.js library to build the user interface. You can use Node Package Manager (npm) or Yarn for package management.

**Code Editor:**

Choose a code editor such as Visual Studio Code, Atom, or Sublime Text for writing React.js code.

**Node.js:**

React.js requires Node.js for development. Ensure you have Node.js installed to run npm commands.

**Create React App:**

Use Create React App to set up a new React.js project quickly. This tool simplifies the initial project setup.

**Backend (Spring Boot):**

* + Java Development Kit (JDK)
  + Integrated Development Environment (IDE):
  + Spring Boot
  + Database
  + Spring Data JPA

**Java Development Kit (JDK):**

Install the Java Development Kit (JDK) to develop and run Java applications. Spring Boot is a Java-based framework.

**Integrated Development Environment (IDE):**

Choose an IDE for Java development. Popular choices include IntelliJ IDEA, Eclipse, or Spring Tool Suite.

**Spring Boot:**

Install Spring Boot to create and develop the backend of the restaurant management system. You can use the Spring Initializer to bootstrap your project.

**Database:**

Choose a relational database like MySQL for storing data. Install the necessary database server and client.

**Spring Data JPA:**

Spring Data JPA simplifies database access in a Spring application. Include it in your project for efficient data access.

**Communication Between Frontend and Backend:**

**RESTful API Design:**

Define RESTful APIs to facilitate communication between the frontend and backend. Design clear endpoints for data exchange.

**API Documentation:**

Use tools like Swagger or Spring RestDocs to document your RESTful APIs, making it easier for frontend developers to integrate.

**4.5.2 Hardware requirements**

**Backend (Spring Boot):**

* + Server
  + Processor (CPU)
  + Memory (RAM)
  + Storage
  + Network Interface
  + Separate Database Server
  + Processor (CPU) and Memory (RAM):
  + Storage
  + Client Devices
  + Network Connectivity

**Server:**

A dedicated server or virtual machine for hosting the Spring Boot backend. The server specifications should be based on factors such as expected traffic, concurrent users, and computational requirements.

**Processor (CPU):**

A multi-core processor with a decent clock speed is recommended. The number of cores and clock speed will depend on the expected load and complexity of backend operations.

**Memory (RAM):**

Allocate sufficient RAM to the server based on the size of the application and the expected user load. A minimum of 8GB RAM is recommended, but the actual requirements may vary.

**Storage:**

Use SSD storage for faster read and write operations. The storage capacity depends on the size of the database, application code, and any additional files.

**Network Interface:**

A stable and high-speed network connection is crucial, especially if the application interacts with external services, databases, or other components.

**Separate Database Server:**

For larger applications, consider hosting the database on a separate server to distribute the load.

**Processor (CPU) and Memory (RAM):**

Similar to the backend server, allocate appropriate CPU and RAM resources to the database server based on expected usage.

**Storage:**

Use fast and reliable storage for the database server. Consider the type of database system (e.g., MySQL, PostgreSQL) and its storage requirements.

**Client Devices:**

Client devices (computers, tablets, smartphones) that access the React.js frontend should meet basic requirements for modern web applications. These typically include a web browser with JavaScript enabled.

**Network Connectivity:**

A stable internet connection is essential for users accessing the online restaurant management system.

**CHAPTER 5**

**IMPLEMENTATION**

**5. IMPLEMENTATION**

**5.1. Implementation code:**

# App.js

import { Routes, Route } from "react-router-dom";

import Header from "./NavbarComponent/Header";

import AdminRegisterForm from "./UserComponent/AdminRegisterForm";

import UserLoginForm from "./UserComponent/UserLoginForm";

import UserRegister from "./UserComponent/UserRegister";

import AboutUs from "./PageComponent/AboutUs";

import ContactUs from "./PageComponent/ContactUs";

import HomePage from "./PageComponent/HomePage";

import AddCategoryForm from "./CategoryComponent/AddCategoryForm";

import AddFoodForm from "./FoodComponent/AddFoodForm";

import Food from "./FoodComponent/Food";

import AddFoodReview from "./ReviewComponent/AddFoodReview";

import GetFoodReviews from "./ReviewComponent/GetFoodReviews";

import RestaurantFoods from "./FoodComponent/RestaurantFoods";

import ViewRestaurantFoods from "./FoodComponent/ViewRestaurantFoods";

import UpdateFoodForm from "./FoodComponent/UpdateFoodForm";

import ViewAllCategories from "./CategoryComponent/ViewAllCategories";

import UpdateCategoryForm from "./CategoryComponent/UpdateCategoryForm";

import ViewAllFoods from "./FoodComponent/ViewAllFoods";

import AddCardDetails from "./OrderComponent/AddCardDetails";

import ViewMyCart from "./CartComponent/ViewMyCart";

import ViewMyOrders from "./OrderComponent/ViewMyOrders";

import ViewAllOrders from "./OrderComponent/ViewAllOrders";

import ViewRestaurantDeliveryPerson from "./UserComponent/ViewRestaurantDeliveryPerson";

import ViewRestaurantOrders from "./OrderComponent/ViewRestaurantOrders";

import ViewAllRestaurants from "./UserComponent/ViewAllRestaurants";

import ViewAllDeliveryPersons from "./UserComponent/ViewAllDeliveryPersons";

import ViewAllCustomers from "./UserComponent/ViewAllCustomers";

import ViewDeliveryOrders from "./OrderComponent/ViewDeliveryOrders";

function App() {

return (

<div>

<Header />

<Routes>

<Route path="/" element={<HomePage />} />

<Route path="/home" element={<HomePage />} />

<Route path="/user/admin/register" element={<AdminRegisterForm />} />

<Route path="/user/login" element={<UserLoginForm />} />

<Route path="/user/customer/register" element={<UserRegister />} />

<Route path="/user/restaurant/register" element={<UserRegister />} />

<Route path="/restaurant/delivery/register" element={<UserRegister />} />

<Route path="/aboutus" element={<AboutUs />} />

<Route path="/contactus" element={<ContactUs />} />

<Route

path="/food/category/:categoryId/:categoryName"

element={<HomePage />}

/>

<Route path="/category/add" element={<AddCategoryForm />} />

<Route path="/food/add" element={<AddFoodForm />} />

<Route

path="/food/:foodId/category/:categoryId"

element={<Food />}

/>

<Route

path="/food/:foodId/review/add"

element={<AddFoodReview />}

/>

<Route path="/food/review/fetch" element={<GetFoodReviews />} />

<Route

path="/food/restaurant/:restaurantId/:restaurantName"

element={<RestaurantFoods />}

/>

<Route

path="/food/restaurant/:restaurantId/:restaurantName/category/:categoryId/:categoryName"

element={<RestaurantFoods />}

/>

<Route path="/restaurant/food/all" element={<ViewRestaurantFoods />} />

<Route path="/restaurant/food/update" element={<UpdateFoodForm />} />

<Route path="/admin/category/all" element={<ViewAllCategories />} />

<Route path="/admin/category/update" element={<UpdateCategoryForm />} />

<Route path="/admin/food/all" element={<ViewAllFoods />} />

<Route path="/customer/order/payment" element={<AddCardDetails />} />

<Route path="/customer/cart" element={<ViewMyCart />} />

<Route path="/customer/order" element={<ViewMyOrders />} />

<Route path="/admin/order/all" element={<ViewAllOrders />} />

<Route

path="/restaurant/delivery-person/all"

element={<ViewRestaurantDeliveryPerson />}

/>

<Route path="/restaurant/order/all" element={<ViewRestaurantOrders />} />

<Route path="/admin/restaurant/all" element={<ViewAllRestaurants />} />

<Route

path="/admin/delivery-person/all"

element={<ViewAllDeliveryPersons />}

/>

<Route path="/admin/customer/all" element={<ViewAllCustomers />} />

<Route

path="/delivery-person/order/all"

element={<ViewDeliveryOrders />}

/>

</Routes>

</div>

);

}

export default App;

**index.js**

import React from "react";

import ReactDOM from "react-dom/client";

import App from "./App";

import { BrowserRouter } from "react-router-dom";

import "./index.css";

const root = ReactDOM.createRoot(document.getElementById("root"));

root.render(

<BrowserRouter>

<App />

</BrowserRouter>);

**Home page.jsx**

import React, { useState, useEffect } from "react";

import { useParams } from "react-router-dom";

import axios from "axios";

import FoodCard from "../FoodComponent/FoodCard";

import Carousel from "./Carousel";

import Footer from "../NavbarComponent/Footer";

const HomePage = () => {

const { categoryId, categoryName } = useParams();

const [foods, setFoods] = useState([]);

const [searchText, setSearchText] = useState("");

const [tempSearchText, setTempSearchText] = useState("");

useEffect(() => {

const fetchData = async () => {

try {

let response;

if (categoryId == null && searchText === "") {

// Fetch all foods

response = await axios.get(

`http://localhost:8080/api/food/fetch/all`

);

} else if (searchText) {

// Fetch foods by name

response = await axios.get(

`http://localhost:8080/api/food/search?foodName=${searchText}`

);

} else {

// Fetch foods by category

response = await axios.get(

`http://localhost:8080/api/food/fetch/category-wise?categoryId=${categoryId}`

);

}

if (response.data) {

setFoods(response.data.foods);

}

} catch (error) {

console.error("Error fetching data:", error);

}

};

fetchData();

}, [categoryId, searchText]);

const searchFoods = (e) => {

e.preventDefault();

setSearchText(tempSearchText);

};

return (

<div className="container-fluid mb-2">

<Carousel />

<div className="d-flex aligns-items-center justify-content-center mt-5">

<form class="row g-3">

<div class="col-auto">

<input

type="text"

class="form-control"

id="inputPassword2"

placeholder="Enter Food Name..."

onChange={(e) => setTempSearchText(e.target.value)}

style={{

width: "350px",

}}

value={tempSearchText}

required

/>

</div>

<div class="col-auto">

<button

type="submit"

class="btn bg-color custom-bg-text mb-3"

onClick={searchFoods}

>

Search

</button>

</div>

</form>

</div>

<div className="col-md-12 mt-3 mb-5">

<div className="row row-cols-1 row-cols-md-4 g-4">

{foods.map((food) => {

return <FoodCard item={food} key={food.id} />;

})}

</div>

</div>

<hr />

<Footer />

</div>

);

};

export default HomePage;

**Restaurant foods.jsx:**

import React, { useState, useEffect } from "react";

import { useParams } from "react-router-dom";

import axios from "axios";

import FoodCard from "../FoodComponent/FoodCard";

import { useLocation } from "react-router-dom";

const RestaurantFoods = () => {

const location = useLocation();

const restaurant = location.state;

const { categoryId, categoryName, restaurantName } = useParams();

const [foods, setFoods] = useState([]);

useEffect(() => {

const fetchData = async () => {

try {

let response;

if (categoryId == null) {

// Fetch all foods

response = await axios.get(

`http://localhost:8080/api/food/fetch/restaurant-wise?restaurantId=${restaurant.id}`

);

} else {

// Fetch foods by category

response = await axios.get(

`http://localhost:8080/api/food/fetch/restaurant-wise/category-wise?restaurantId=${restaurant.id}&categoryId=${categoryId}`

);

}

if (response.data) {

setFoods(response.data.foods);

}

} catch (error) {

console.error("Error fetching data:", error);

}

};

fetchData();

}, [restaurant, categoryId]);

return (

<div className="container-fluid mb-2">

{/\* <Carousel /> \*/}

<div

className="bg-color custom-bg-text mt-2 d-flex justify-content-center align-items-center"

style={{

borderRadius: "1em",

height: "38px",

}}

>

<h5 class="card-title ms-3">Restaurant Name: {restaurantName}</h5>

</div>

<div className="col-md-12 mt-3">

<div className="row row-cols-1 row-cols-md-4 g-4">

{foods.map((food) => {

return <FoodCard item={food} key={food.id} />;

})}

</div>

</div>

</div>

);

};

export default RestaurantFoods;

**Application. properties**

# MySQL Properties

spring.datasource.url=jdbc:mysql://localhost:3306/food\_multivendor\_store?createDatabaseIfNotExist=true&useUnicode=true

spring.datasource.username=root

#enter the correct MySQL Password below

spring.datasource.password=1234

spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver

spring.jpa.hibernate.ddl-auto=update

spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQL57Dialect

springdoc.packagesToScan=com.onlinefood.controller

springdoc.pathsToMatch=/api/\*\*

# File upload properties

com.onlinefood.image.folder.path=C:/Users/Admin/OneDrive/Desktop/full stack project/images

spring.servlet.multipart.max-file-size=10MB

spring.servlet.multipart.max-request-size=10MB

**onlinemultishopvendorapplication.java**

package com.onlinefood;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.boot.CommandLineRunner;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

import org.springframework.security.crypto.password.PasswordEncoder;

import com.onlinefood.entity.User;

import com.onlinefood.service.UserService;

import com.onlinefood.utility.Constants.UserRole;

import com.onlinefood.utility.Constants.UserStatus;

@SpringBootApplication

public class OnlineFoodMultivendorShopApplication implements CommandLineRunner {

private final Logger LOG = LoggerFactory.getLogger(OnlineFoodMultivendorShopApplication.class);

@Autowired

private UserService userService;

@Autowired

private PasswordEncoder passwordEncoder;

public static void main(String[] args) {

SpringApplication.run(OnlineFoodMultivendorShopApplication.class, args);

}

@Override

public void run(String... args) throws Exception {

User admin = this.userService.getUserByEmailIdAndRoleAndStatus("demo.admin@demo.com",

UserRole.ROLE\_ADMIN.value(), UserStatus.ACTIVE.value());

if (admin == null) {

LOG.info("Admin not found in system, so adding default admin");

User user = new User();

user.setEmailId("demo.admin@demo.com");

user.setPassword(passwordEncoder.encode("123456"));

user.setRole(UserRole.ROLE\_ADMIN.value());

user.setStatus(UserStatus.ACTIVE.value()); this.userService.addUser(user);

}}}

**Pom.xml**

<?xml version="1.0" encoding="UTF-8"?>

<projectxmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<parent>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-parent</artifactId>

<version>3.1.3</version>

<relativePath/> <!-- lookup parent from repository -->

</parent>

<groupId>com.onlinefood</groupId>

<artifactId>online-food-multivendor-shop</artifactId>

<version>0.0.1-SNAPSHOT</version>

<name>online-food-multivendor-shop</name>

<description>Online Food Order Multi-Vendor Shop Rest APIs using Spring Boot</description>

<properties>

<java.version>17</java.version>

</properties>

<dependencies>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-data-jpa</artifactId>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-security</artifactId>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-web</artifactId>

</dependency>

<!-- exclude logback -->

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter</artifactId>

<exclusions>

<exclusion>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-logging</artifactId>

</exclusion>

</exclusions>

</dependency>

<!-- add log4j2 -->

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-log4j2</artifactId>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-devtools</artifactId>

<scope>runtime</scope>

<optional>true</optional>

</dependency>

<dependency>

<groupId>com.mysql</groupId>

<artifactId>mysql-connector-j</artifactId>

<scope>runtime</scope>

</dependency>

<dependency>

<groupId>org.projectlombok</groupId>

<artifactId>lombok</artifactId>

<optional>true</optional>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-test</artifactId>

<scope>test</scope>

</dependency>

<dependency>

<groupId>org.springframework.security</groupId>

<artifactId>spring-security-test</artifactId>

<scope>test</scope>

</dependency>

<dependency>

<groupId>io.jsonwebtoken</groupId>

<artifactId>jjwt-api</artifactId>

<version>0.11.5</version>

</dependency>

<dependency>

<groupId>io.jsonwebtoken</groupId>

<artifactId>jjwt-impl</artifactId>

<version>0.11.5</version>

</dependency>

<dependency>

<groupId>io.jsonwebtoken</groupId>

<artifactId>jjwt-jackson</artifactId>

<version>0.11.5</version>

</dependency>

<dependency>

<groupId>org.springdoc</groupId>

<artifactId>springdoc-openapi-starter-webmvc-ui</artifactId>

<version>2.2.0</version>

</dependency>

<dependency>

<groupId>com.github.librepdf</groupId>

<artifactId>openpdf</artifactId>

<version>1.3.8</version>

</dependency>

<dependency>

<groupId>com.fasterxml.jackson.core</groupId>

<artifactId>jackson-databind</artifactId>

</dependency>

</dependencies>

<build>

<plugins>

<plugin>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-maven-plugin</artifactId>

</plugin>

</plugins>

</build>

</project>

# CHAPTER 6

# RESULT&ANALYSIS

**6. RESULT**

# 6.1 Customer Outputs

**6.1.1 Guest User Home page**

# 

Fig 6.1.1 Guest user home page

This is the home page for the guests those who don’t register account and just see the website and available foods.

**6.1.2 Guest User Menu Items**

# 

Fig Guest user menu items

This is the items available by all the restaurants. Guests can see the items if they want to add to the cart guest must login or sign up to their account.

**6.1.3 Customer Login page**

# 

Fig customer login page

This is the login page for the previous customers those want to login to their account and order the food items

**6.1.4 Add to cart page**

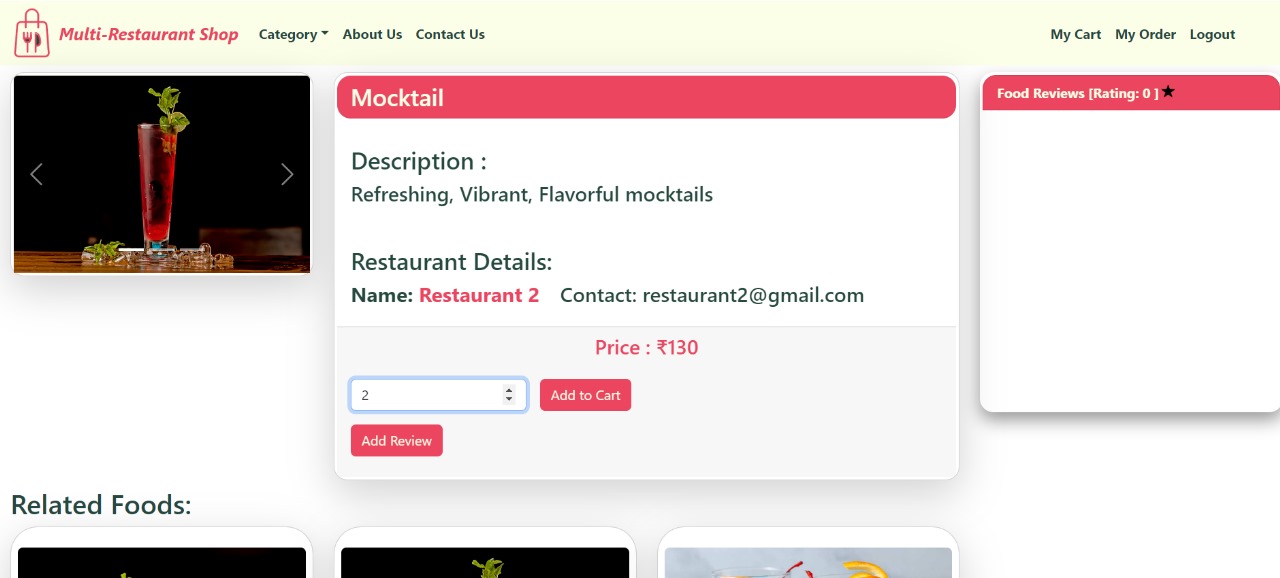


Fig 6.1.4 Add to cart page

This page appears after customer added the item to the cart and they have to mention the quantity of the items they want.

**6.1.5 Related Foods**

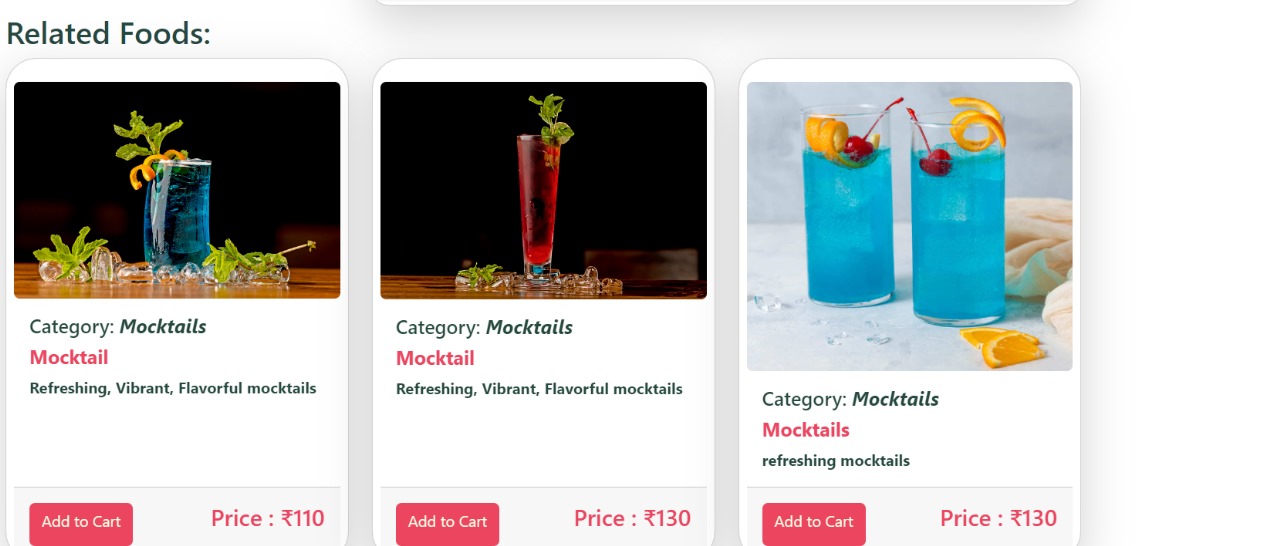


Fig 6.1.5 Releted foods

This page appears just after the add to cart page and they can see the related food items to their cart.

**6.1.6 My cart**

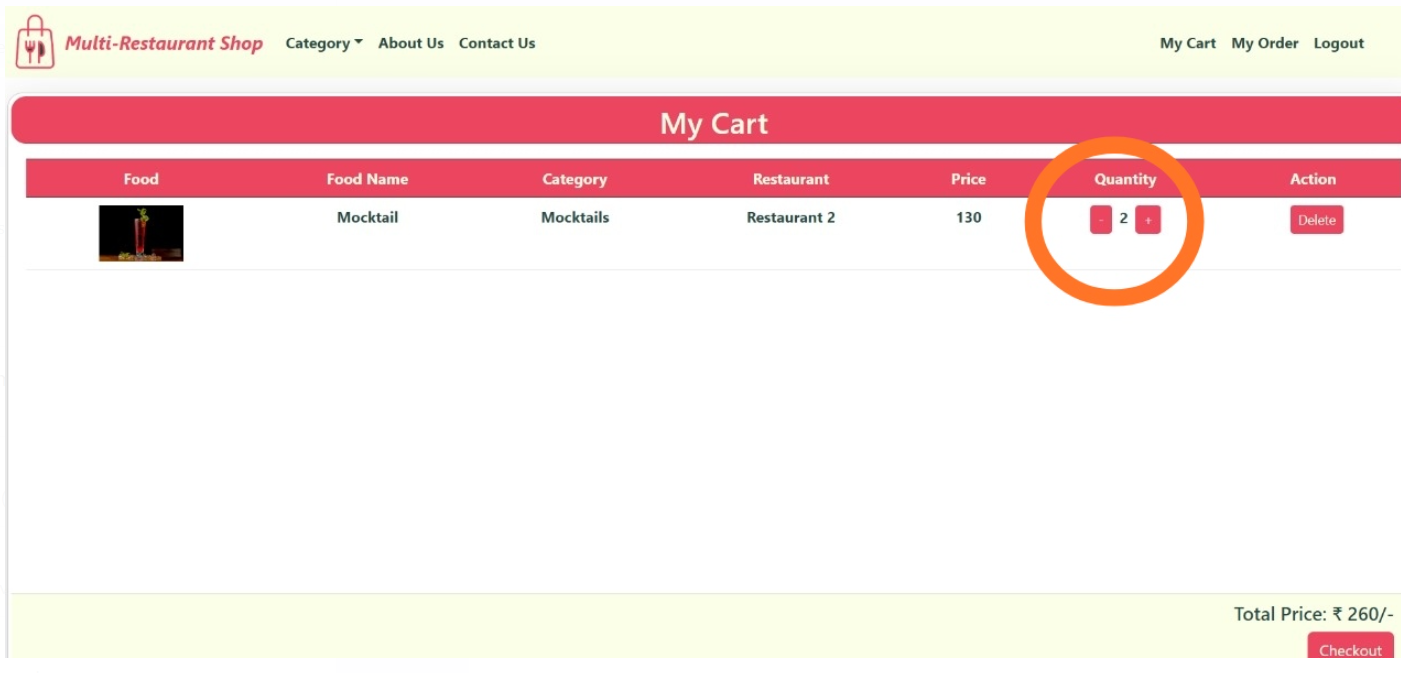
****

Fig 6.1.6 My cart

This page shows the items we added in the cart and which restaurant the product belongs and the total price of the items.

**6.1.7 Payment Details**

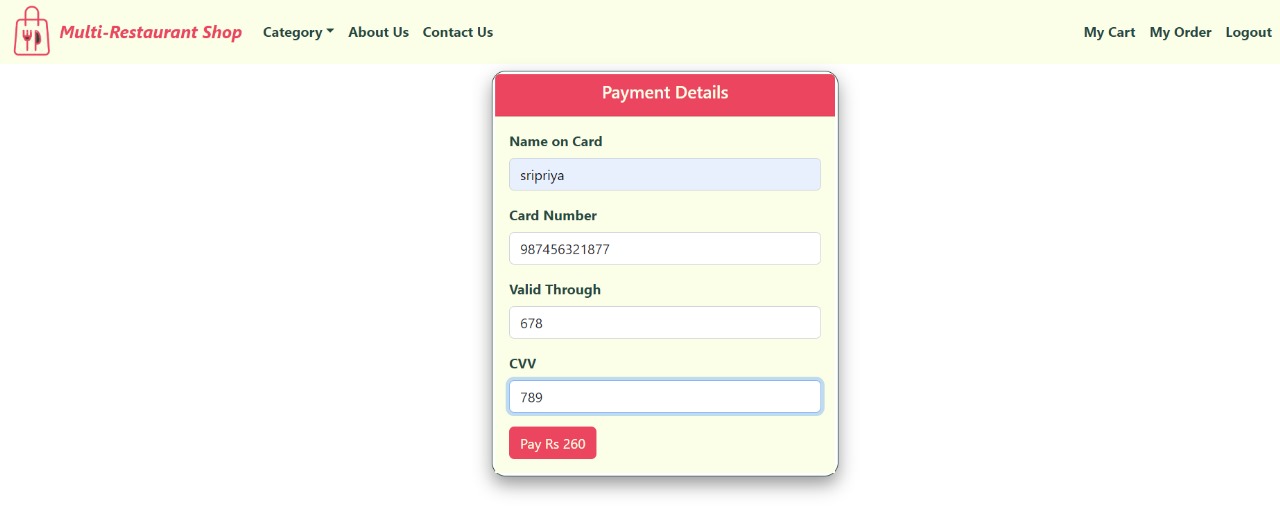
****

Fig 6.1.7 Payment details

This page appears after we click the checkout button and we have to enter the details of the cart and the order will be placed.

**6.1.8 My Orders**

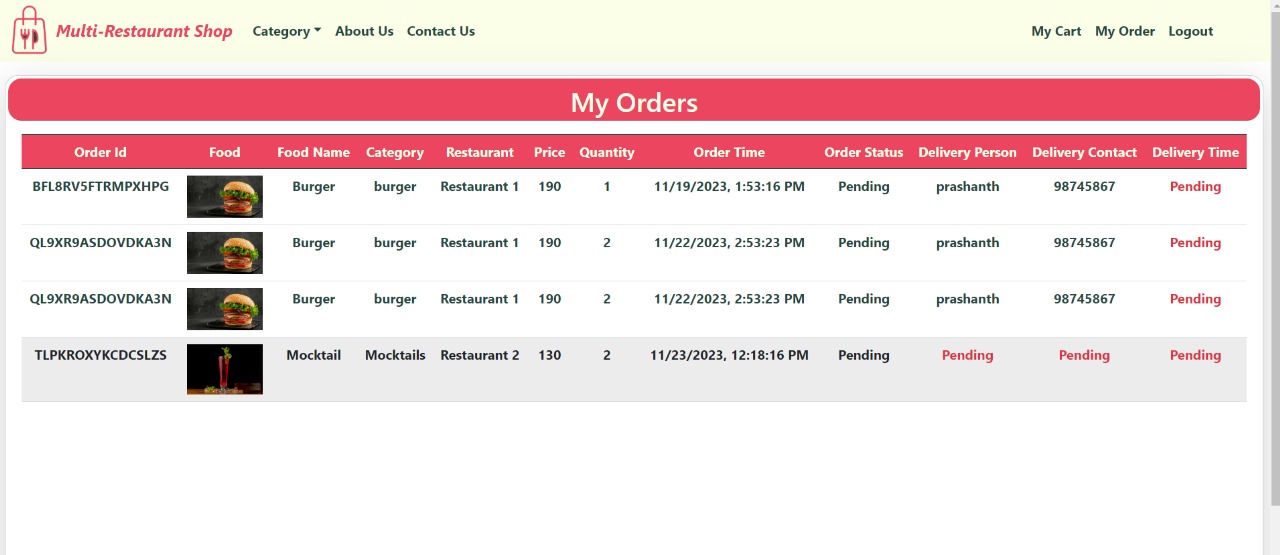


Fig 6.1.8 My orders

This page shows all the orders the customer placed before and the price of the item and delivery person also.

# 6.2 Restaurant Outputs

**6.2.1 Restaurant Login Page**

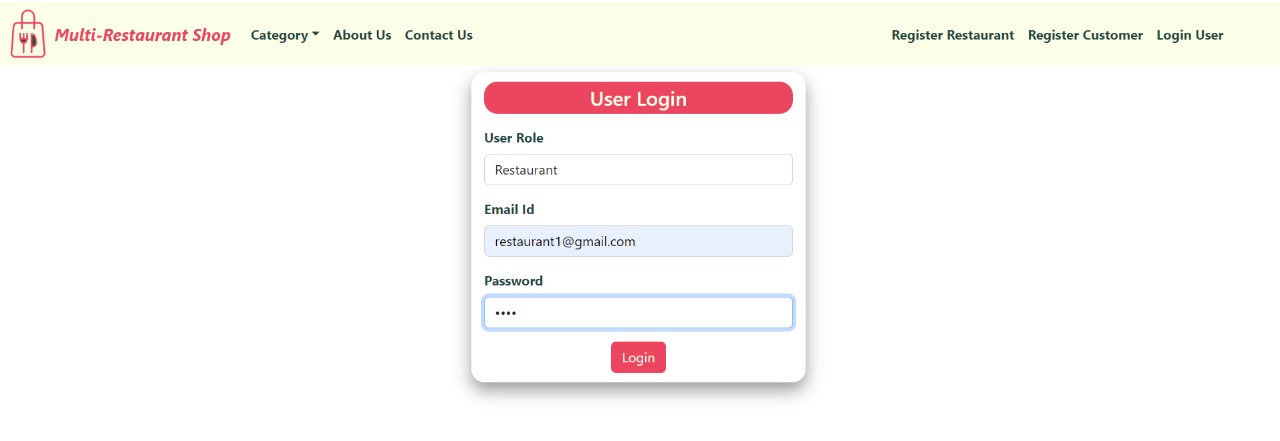


Fig 6.2.1 Restaurant login page

This is the login page for the restaurant. Restaurants can login into their account and add their products

**6.2.2 Restaurant Home page**

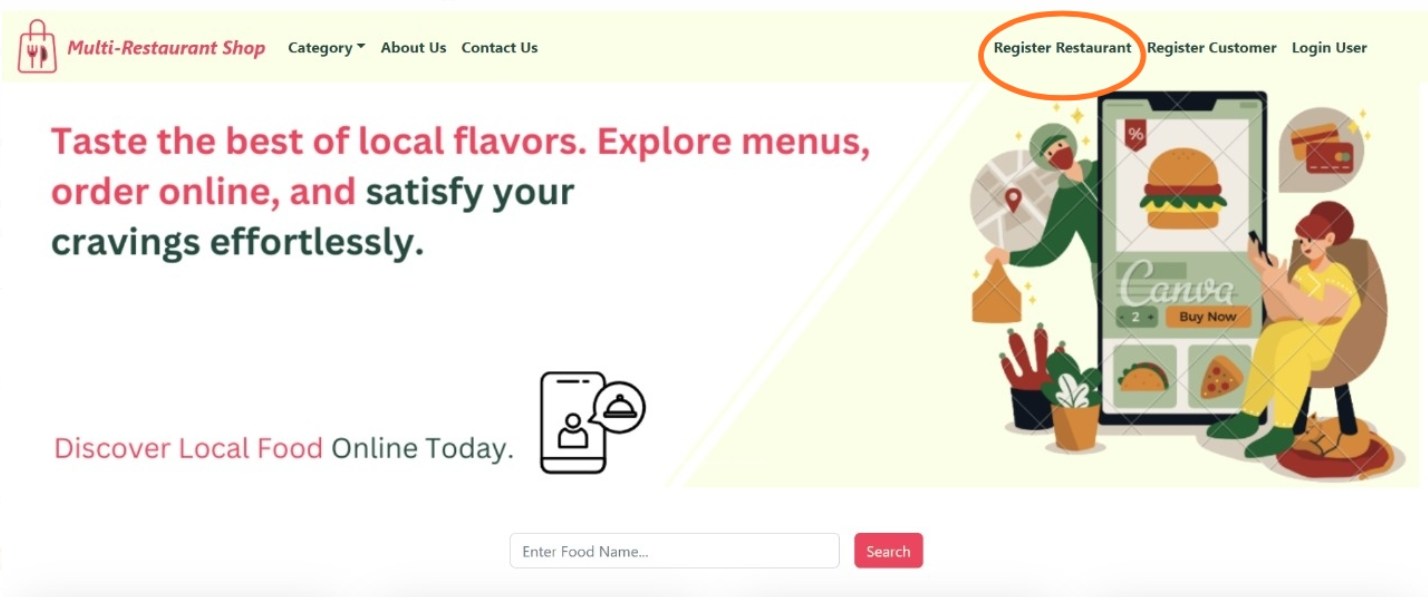


Fig 6.2.2 Restaurant home page

This is the home page for restaurant and they are having access to view restaurants and foods.

**6.2.3 Restaurant Orders**

# 

Fig 6.2.3 Restaurant orders

Restaurant can see the orders placed by the customers and they have to assign the order to the delivery person.

**6.2.4 All Delivery Persons**

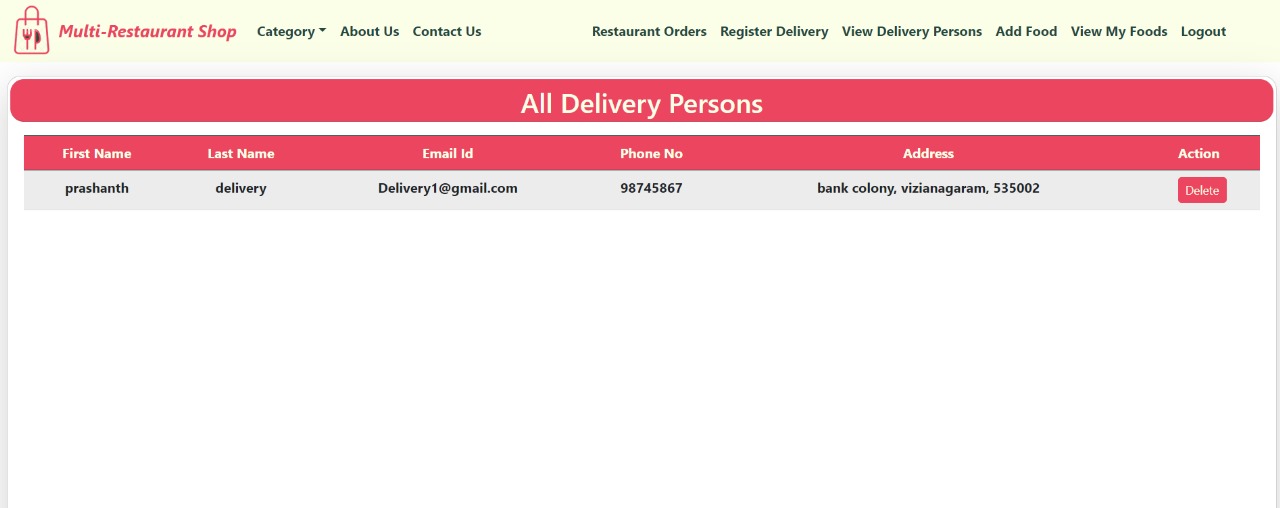


Fig 6.2.4 All Delivery Persons

This page shows all delivery persons available for the restaurant and restaurant people check the delivery person and assign the order to them.

**6.2.5 My Foods**



Fig 6.2.5 My foods

This page shows all the available foods by the restaurant and restaurant can update or delete the foods.

**6.2.6 Add food**

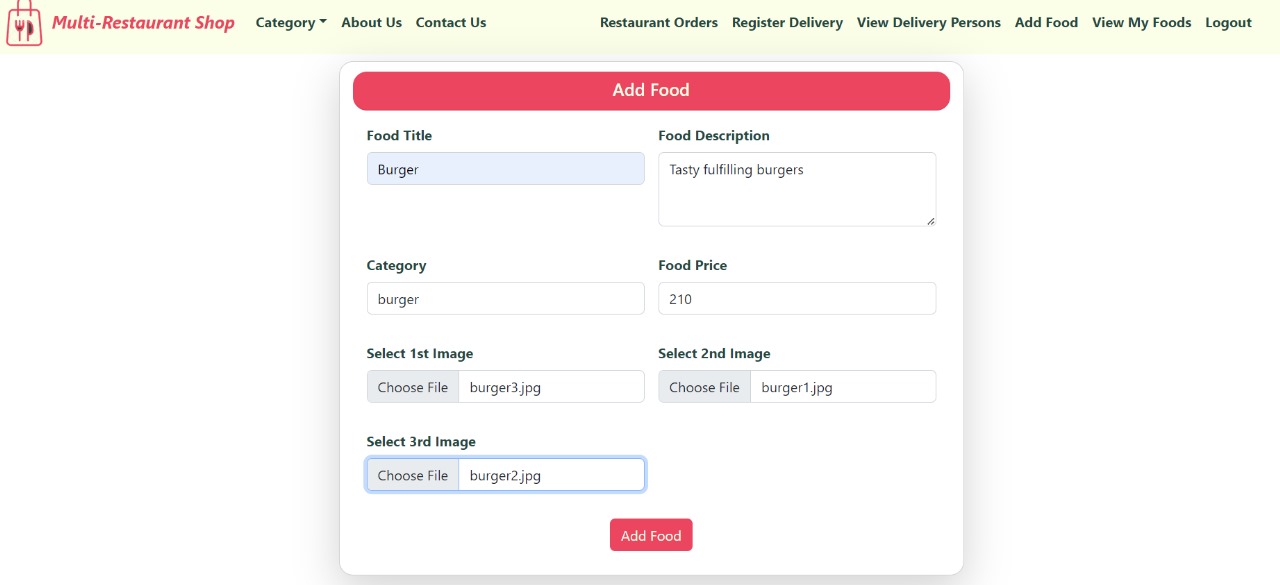


Fig 6.2.6 Add food

Restaurant can add the food by entering food title, description, category and food price.

**6.3 Delivery Person**

**6.3.1 Delivery Login Page**

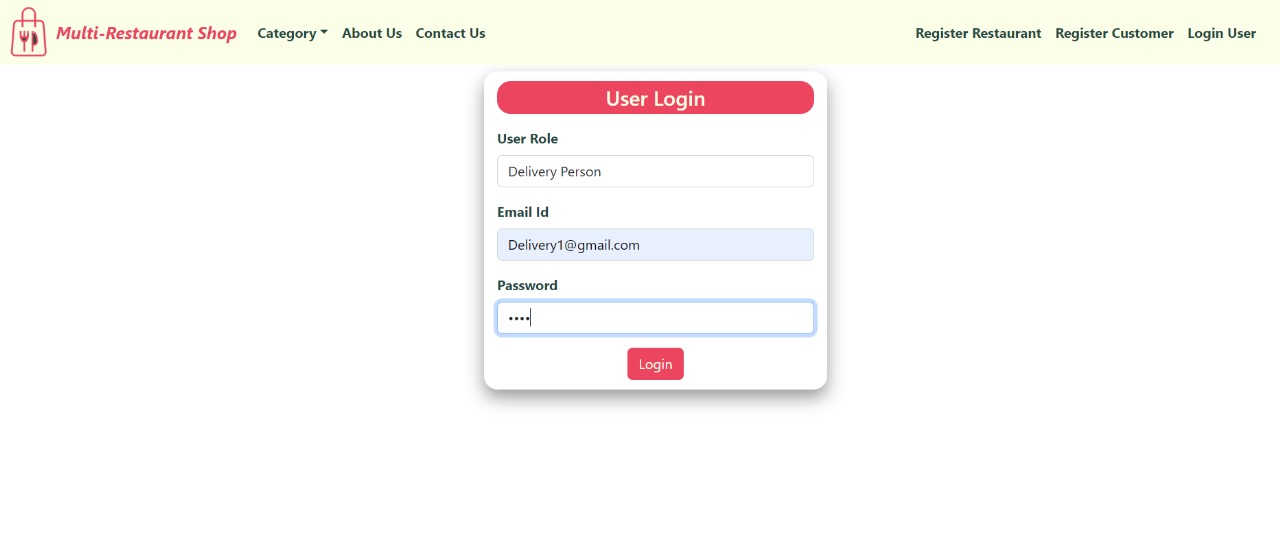


Fig 6.3.1 Delivery login page

This is the login page for delivery persons who were assigned by the restaurants.

**6.3.2 Delivery Home Page**

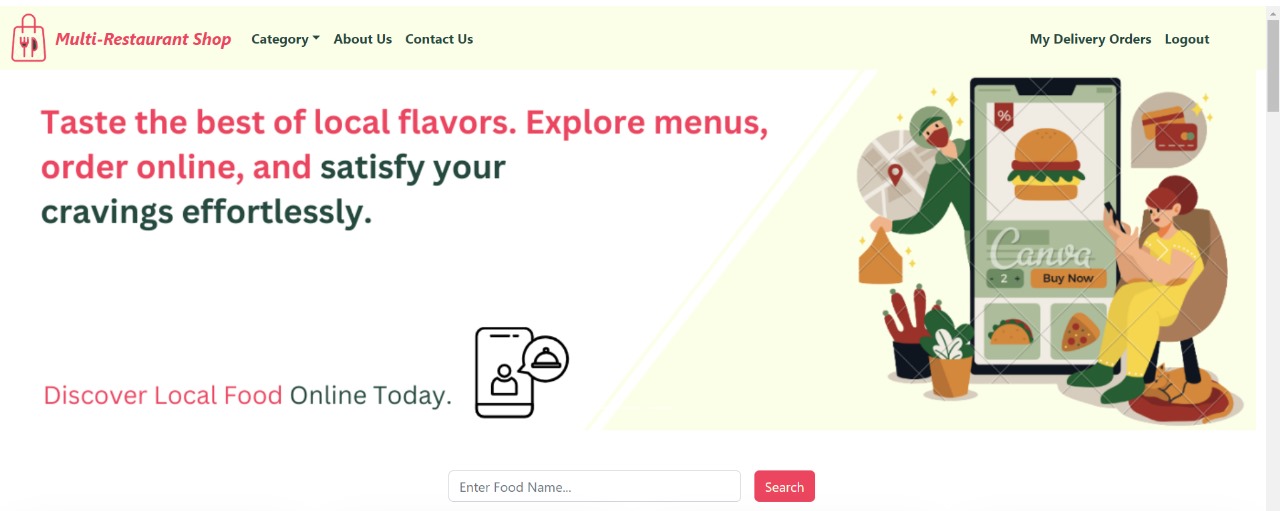


Fig 6.3.2 Delivery home page

This is the home page for delivery persons and they can check the delivery orders which were assigned by the restaurant and update the order.

**6.3.3 My Delivery Orders**

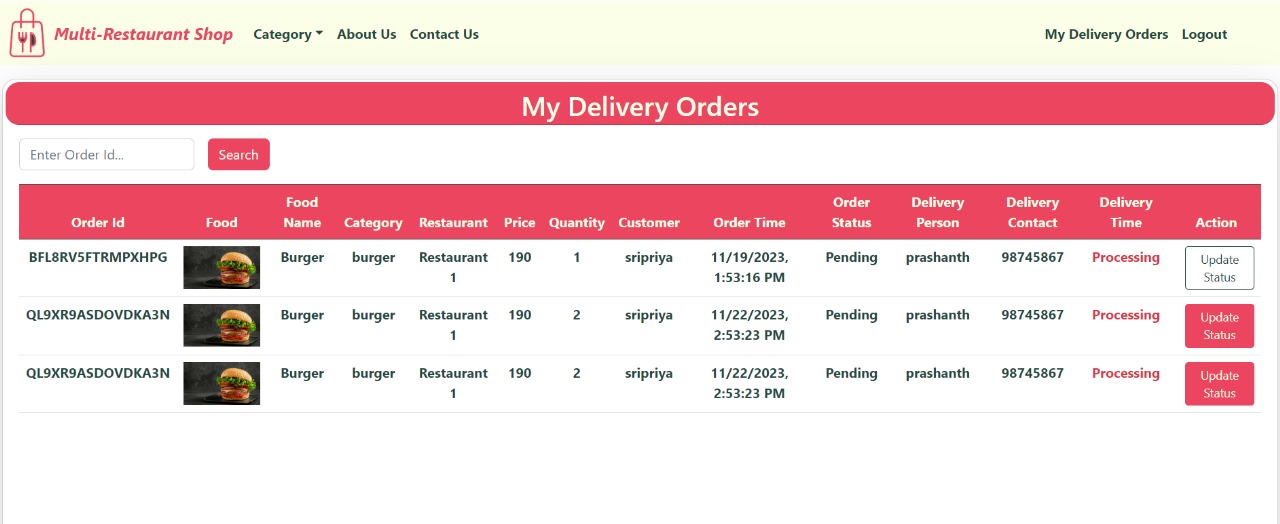


Fig 6.3.3 My Delivery orders

This page displays all the delivery orders for delivery person which were assigned by the restaurants.

**6.3.4 Update Delivery Status**

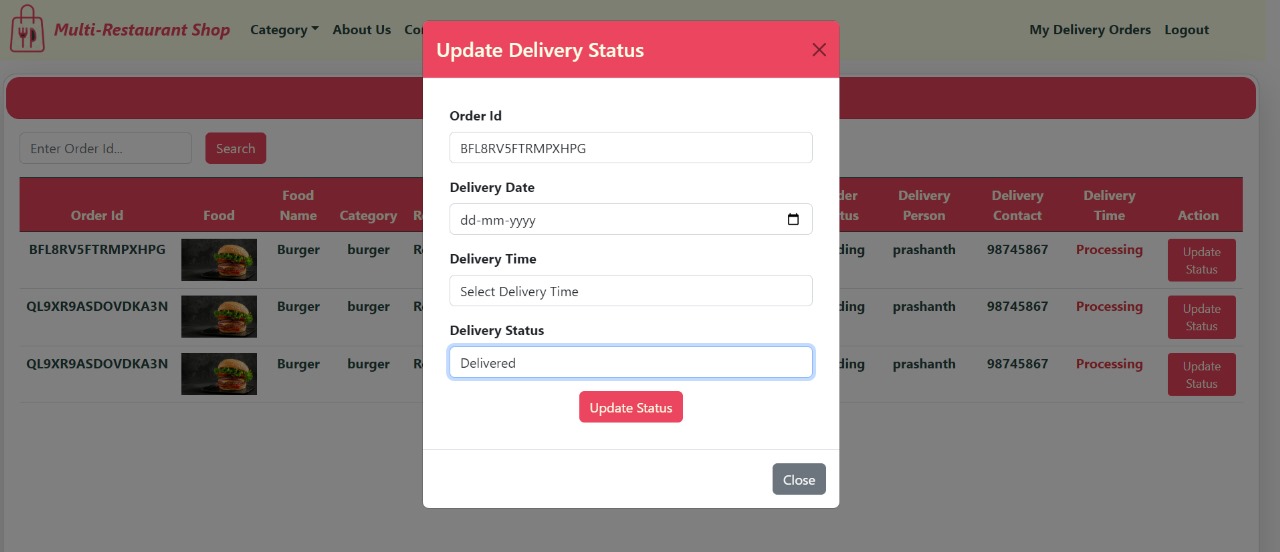


Fig 6.3.4 update delivery status

Delivery person must update the delivery status once it was completely delivered to the customer.

**6.4 Admin**

**6.4.1 Admin Login**

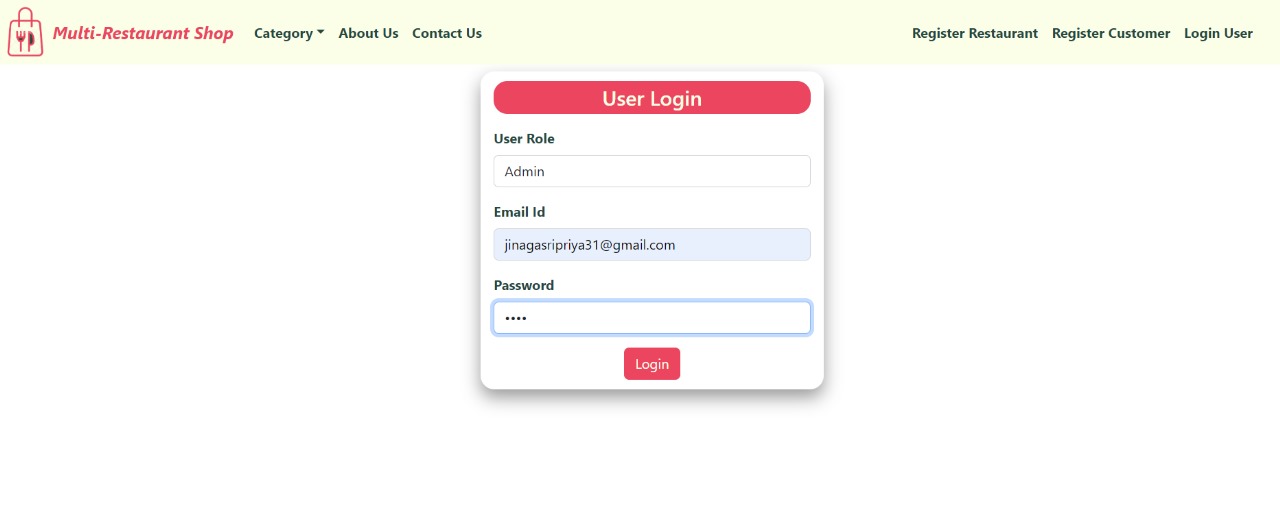


Fig 6.4.1 Admin login

This is the login page for admin where admin has access to whole website.

**6.4.2 Admin Home Page**

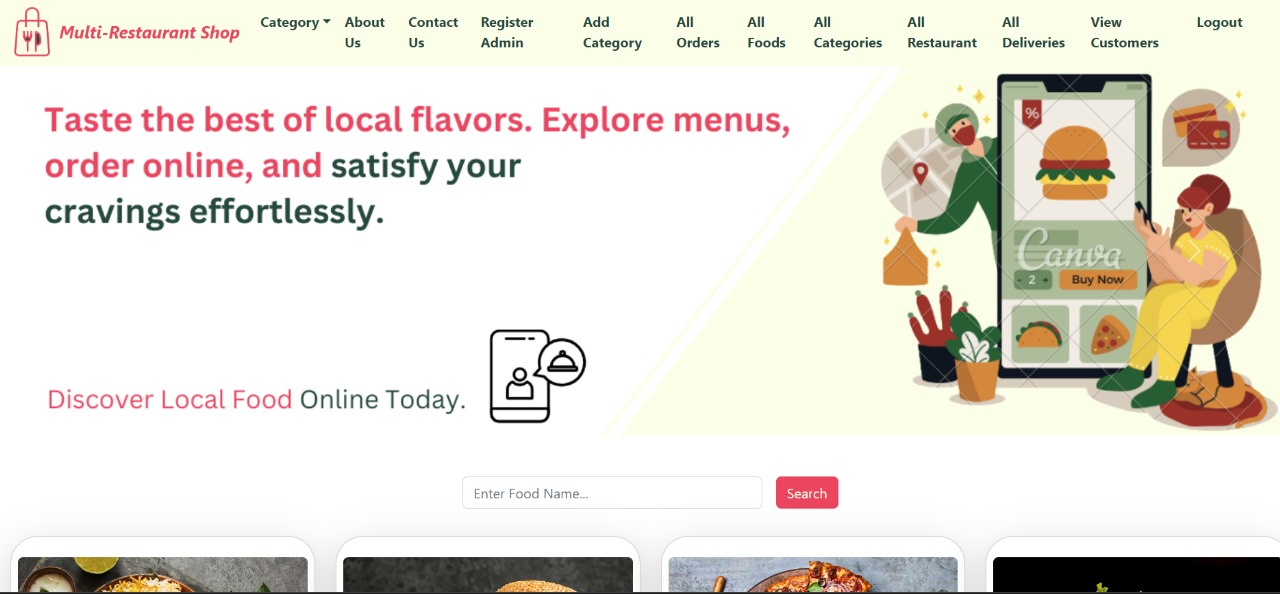


Fig 6.4.2 Admin home page

It is the home page for admin where he is having access to check all orders, restaurants and costumers

**6.4.3 All restaurants**



Fig 6.4.3 All Restaurants

Admin have all the rights to check the restaurants available in the website and delete them.

**6.4.4 All customers**

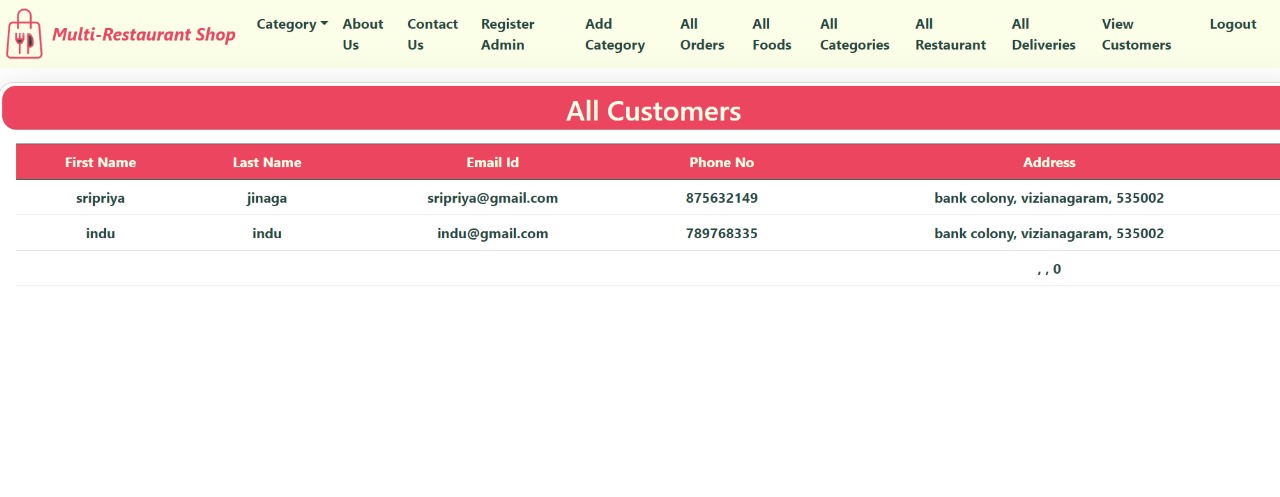


Fig 6.4.4 All customers

Admin have all the rights to check the customers who were registered in this website and their orders

# CHAPTER:7

# CONCLUSION & FUTURE WORK

**7.CONCLUSION &FUTURE WORK**

**7.1 CONCLUSION:**

The development of an online restaurant management system using React.js for the frontend and Spring Boot for the backend represents a significant leap forward in streamlining restaurant operations, enhancing customer experiences, and improving overall efficiency. The integration of a modern and responsive user interface (UI) through React.js ensures a seamless and enjoyable experience for both customers and restaurant staff. Meanwhile, the robust backend powered by Spring Boot provides a scalable and secure foundation for managing orders, menus, and various other aspects of restaurant operations. The system facilitates real-time communication, enabling instantaneous updates on order statuses, menu changes, and customer interactions. Secure user authentication and authorization mechanisms enhance data security, ensuring that sensitive information is protected throughout the system. The utilization of RESTful APIs and a well-designed database schema ensures smooth communication between the frontend and backend components.The online restaurant management system offers features such as dynamic menu customization, efficient order processing, inventory management, and secure payment handling. These functionalities collectively contribute to operational efficiency, reduced costs, and an improved customer experience.

**7.2FUTUREWORK:**

**Mobile Application Development:**

Extend the system by developing dedicated mobile applications for customers and staff. This allows for greater accessibility and convenience, enabling users to manage orders and menus on the go.

**Integration of Machine Learning:**

Implement machine learning algorithms for personalized customer recommendations, predictive inventory management, and dynamic pricing strategies. This could further optimize the system based on historical data and user behaviors.

**Enhanced Analytics and Reporting:**

Incorporate advanced analytics tools to generate detailed reports on sales trends, customer preferences, and operational insights. This can aid decision-making for restaurant managers and owners.

**Integration with Third-Party Services:**

Explore partnerships and integrations with third-party services such as food delivery platforms, reservation systems, or loyalty programs to expand the system's capabilities and reach.

**Multi-Language and Accessibility Support:**

Enhance the system's inclusivity by providing multi-language support for a diverse customer base. Implement accessibility features to ensure the platform is usable by individuals with disabilities.

**Scalability and Performance Optimization:**

Conduct performance testing to identify areas for optimization and ensure the system can seamlessly handle increased user loads as the restaurant grows.

**Feedback and Customer Interaction:**

Implement features for collecting customer feedback and reviews. Utilize this data to continually improve the system and address any pain points in the user experience.

**Blockchain Integration for Security:**

Explore the integration of blockchain technology to enhance data security, traceability, and transparency in transactions and user interactions.

**8.REFERENCES**

**8.REFERENCES:**

[1] Soon Nyean Cheong, Wei Wing Chiew, and Wen Jiun Yap, “Design and development of Multi-touchable E-restaurant Management System,” in *2010 International Conference on Science and Social Research (CSSR 2010)*, IEEE, Dec. 2010, pp. 680–685. doi: 10.1109/CSSR.2010.5773867.

[2] L. Deksne, A. Kempelis, T. Sniedzins, and A. Kozlovskis, “Automated System for Restaurant Services,” *Inf. Technol. Manag. Sci.*, vol. 24, pp. 15–25, Dec. 2021, doi: 10.7250/itms-2021-0003.

[3] A. Jain, S. Chauhan, A. Hirlekar, and S. Sarange, “Automated Restaurant Management System,” *Int. J. Innov. Res. Electr. Electron. Instrum. Control Eng.*, vol. 4, no. 5, pp. 35–38, 2016, doi: 10.17148/IJIREEICE.2016.4508.

[4] A. Bankar and S. S. Suresh, “Intelligent Restaurant -Menu Ordering System,” *IOSR J. VLSI Signal Process.*, vol. 5, no. 5, pp. 47–53, 2015, doi: 10.9790/4200-05514753.

[5] E. M. Kocaman, “Operational effects of using restaurant management system: An assessment according to business features,” *Int. J. Gastron. Food Sci.*, vol. 25, p. 100408, Oct. 2021, doi: 10.1016/j.ijgfs.2021.100408.