INTERNSHIP REPORT

***Submitted by***

**VANAMALA PURUSHOTHAM [RA2111026040048]**

***Under the guidance of***

# Mr. SHATHISH KHANNA KUMAR INCEDO

(Duration: January 2025 – June 2025)

***In partial fulfillment for the award of the degree***

***of***

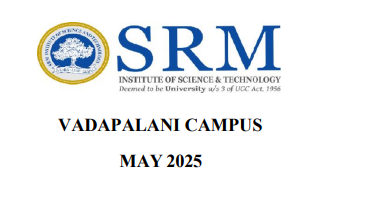
**BACHELOR OF TECHNOLOGY**

***in***

**COMPUTER SCIENCE AND ENGINEERING**

***of***

**FACULTY OF ENGINEERING AND TECHNOLOGY**



**VADAPALANI CAMPUS**

**MAY 2025**

# SRM INSTITUTE OF SCIENCE AND TECHNOLOGY

(Deemed to be University Under Section 3 of UGC Act, 1956)

**BONAFIDE CERTIFICATE**

This is to certify that the “**INTERNSHIP REPORT**” on the topic “**ONLINE FOOD DELIVERY APPLICATION – KORPORATE KITCHEN**” submitted

by **VANAMALA PURUSHOTHAM (Reg. No. RA2111026040048)** is the work done by her and submitted during 2024 – 2025 academic year, in partial fulfillment of the requirements for the award of the degree of BACHELOR OF TECHNOLOGY in COMPUTER SCIENCE AND ENGINEERING, SRM Institute of Science and Technology, Vadapalani at Incedo Inc., Chennai

|  |  |
| --- | --- |
| **INTERNSHIP COORDINATOR** | **HEAD OF THE DEPARTMENT** |
| **Mr. Shathish Khanna Kumar** | **Dr. P. Chitra** |
| Senior Director | Dept. of CSE (Emerging Technologies) |
| Incedo Inc,  RMZ MILLENIA BUSINESS PARK II, Block 3B, 07th Floor, 11, MGR Main Rd, Kandhanchavadi, Perungudi, Chennai, India, 600096 | SRM INSTITUTE OF SCIENCE AND TECHNOLOGY, VADAPALANI, CHENNAI |

**Incedo Technology Solutions Limited CIN: U72200DL2011PLC220071**

**Corporate Office** : 248,Udyog Vihar, Phase-IV, Gurugram-122015

**Registered Office**: A-47, Lower Ground Floor, Hauz Khas, New Delhi-110016 [inquiries@incedoinc.com](mailto:inquiries@incedoinc.com) | [www.incedoinc.com](http://www.incedoinc.com/)

**December 16, 2024**

**Purushotham Vanamala**

1-198, Cherlo Palli, Cherlopalle, Chittoor**,** AP

Chittoor, 517505

**LETTER FOR INTERNSHIP**

**Dear Purushotham Vanamala**,

**Congratulations!** With reference to your application and subsequent interview(s) you had with us, we are pleased to offer you internship with Incedo starting **January 8, 2025** as per the below terms and conditions

**Designation:** Intern

**Location of Training:** Chennai

**Department:** RMZ

**Duration:** 8th Jan 2025 - 8th July 2025

**Stipend:** INR 15,000 per month

1. Please note this is the internship confirmation letter, internship completion letter will be issued only after successful completion of the training.
2. This internship is convertible into an employment with the company subject to the terms mentioned in your appointment letter and submission of below self-attested documents.

 Educational documents: 10th, 12th, Graduation Degree, Post-Graduation Degree (if applicable)  Address proof: Passport/Driving license/ Voter ID card/Aadhaar card

 PAN card copy (Mandatory)

 Aadhaar card copy (Mandatory)

 Work experience letter of previous organizations (if applicable)

 One (1) coloured photograph (passport size, with white background)  Resume





1. We understand that your final semester results are still awaited. Please note, that management reserves the right to terminate your employment with or without notice in case your final results are not as per our expectations.
2. The Company may, at its sole and absolute discretion, conduct background checks prior to or after joining or at any time in future, to check but not limited to your identity, the address provided by you, your education background and past work experience, past antecedents, drug tests and/or any other test or verification. You expressly consent to the Company conducting above checks. You are required to furnish the documents listed in the "Appointment Letter", "Background Verification Form" or any other document as may be required. If the Company, is not satisfied, in its sole and absolute subjective discretion, with the outcome of the aforesaid checks, the Company may (I) Reserve the right to withdraw the Offer made to you without any notice and Compensation (II) Or may treat your appointment as null and void ab-initio (III) Or it may take such other appropriate action as may be advised.
3. Attending and completing all trainings scheduled for your post your joining, and qualifying the assessment is a critical requirement for the employment. If the Company, is not satisfied, in its sole and absolute discretion, with the outcome of the aforesaid assessment, the Company (i) Reserves the right to withdraw the Offer made to you without any notice and compensation ; OR (ii) may treat your appointment as null and void ab-initio; OR (iii) may take such other appropriate action as may be advised.
4. In case, you decide to leave Incedo during your internship period under any circumstances, we recommend providing an advance notice of 4 weeks for a smooth transition.
5. You shall always comply with the Company’s policies/guidelines. Management reserves the rights to amend or modify the existing policies/guidelines as required, at its sole discretion, any time during the year, with or without notice.

Wishing you a good learning experience at Incedo!

**FOR INCEDO TECHNOLOGY SOLUTIONS LIMITED**



**(ESHA GULATI) AUTHORIZED SIGNATORY HUMAN RESOURCES**



# ACKNOWLEDGEMENT

I place on record my deep sense of gratitude to our lionized Chairman **Dr. R. SHIVAKUMAR, MBBS., MD.,** for providing the requisite infrastructure throughout the course.

I take the opportunity to extend my hearty and sincere thanks to our Dean

**Dr.C.V Jayakumar** for manoeuvring into accomplishing the internship.

I take the privilege to extend my hearty and sincere gratitude to the Professor **Dr.Bharathi** for her suggestions, support and encouragement towards the completion of this internship with perfection.

I take the privilege to extend my hearty and sincere gratitude to our Head of the Department **Dr. P.Chitra** for her suggestions, support and encouragement throughout this internship.

I express my hearty and sincere thanks to **Mr. SHATHISH KHANNA KUMAR, SENIOR DIRECTOR(TELECOM), INCEDO** for the constant guidance throughout this internship.

My thanks to the teaching and non-teaching staff of the Computer Science and Engineering department of SRM Institute of Science and Technology,Vadapalani campus, for providing necessary resources and facilities for my work..

VANAMALA PURUSHOTHAM [RA21110260240048]

# ABSTRACT

Korporate Kitchen is an innovative online food delivery system developed during my internship at Incedo Inc., a leading technology firm renowned for its expertise in digital transformation. This project represents a significant step forward in the evolution of the food delivery industry, aiming to streamline and enhance the experience for all stakeholders involved. The primary goal of Korporate Kitchen is to provide a seamless, end-to-end solution that integrates all components of the food delivery ecosystem. This includes connecting users, restaurants, delivery agents, and administrators into one cohesive platform. By leveraging cutting-edge technology, the system ensures a smooth and efficient process from the moment a customer places an order to the final delivery at their doorstep. Key features of the platform include a User-Friendly interface where customers can effortlessly browse menus, place orders, and track deliveries in real time. Next is Restaurant Integration where the system allows restaurants to manage orders, update menus, and optimize delivery schedules with ease. The process after Restaurant integration is Efficient Delivery Management where Delivery agents can quickly receive, process, and complete orders with enhanced route optimization. Then, the Admin Dashboard where Administrators have full control over the platform, including managing users, restaurants, and deliveries, ensuring a seamless operation. Korporate Kitchen not only aims to simplify the food ordering process but also to provide a reliable, fast, and personalized service for both customers and service providers. Through this project, I had the opportunity to apply my technical skills in a real- world setting, contributing to the digital transformation of the food service industry.

## Organization Information

Incedo’s emphasis on innovation and rapid technological advancement were likely involved in projects where we had to analyse complex problems, devise scalable solutions, and ensure that these solutions aligned with industry best practices. Encountering real-world challenges helped to think critically and adapt to fast-paced environments and to develop a strong ability to troubleshoot and optimize systems efficiently. Given Incedo’s focus on staying at the forefront of technology, I have worked with AI/ML models, big data analytics, cloud computing, IoT, or blockchain technologies. Incedo’s emphasis on fostering a culture of innovation and valuing creative solutions helped to think outside the box and experiment with new ideas.

## Programs and Opportunities

During my tenure at Incedo Inc., the Korporate Kitchen project provided you with an exceptional opportunity to not only enhance my technical expertise but also develop a strong foundation in agile methodologies and collaborative problem-solving. This experience gave us a well-rounded exposure to full-stack development, integrating back-end and front-end technologies seamlessly, while reinforcing essential software engineering practices. Being assigned to a full-stack application like Korporate Kitchen meant that you had to work on multiple layers of the application, including Backend Development with Spring Boot, Leveraging Spring Boot, we built robust and scalable RESTful APIs, ensuring efficient communication between the client and the server. We worked with features such as dependency injection, aspect-oriented programming (AOP), and Spring Security to enhance the security and modularity of the application. Developing APIs using Spring Boot taught us how to manage complex business logic, implement exception handling, and optimize response times. Database Management with Hibernate ORM, we effectively mapped Java objects to database tables, allowing seamless interaction between the application and the relational database. Developing dynamic web interfaces with JSP allowed me to create interactive and user-friendly interfaces that communicated with the backend seamlessly.Our core values focus on delivering exceptional service and building lasting relationships with our clients. By maintaining a commitment to delivering projects on time and within budget, we ensure that every solution we offer is not just a service, but a partnership aimed at long-term success. We take pride in our ability to collaborate, communicate transparently, and offer ongoing support, ensuring that our clients always feel confident and empowered in their digital transformation journey.

## Methodologies

The project was developed using the Model-View-Controller (MVC) architecture, ensuring a clear separation of concerns between business logic, the presentation layer, and the data access layer, enhancing maintainability and scalability. Spring Boot streamlined application development by eliminating boilerplate code and enabling seamless deployment with its embedded server. Hibernate and Spring Data JPA simplified database management through efficient object-relational mapping (ORM) and reduced SQL complexity. JSP and JSTL facilitated dynamic content rendering in the front end, while Bootstrap ensured a responsive, mobile-friendly design. Agile methodologies, including Scrum, guided the development process through iterative sprints, regular reviews, and continuous feedback. User stories and backlog management ensured that the most valuable features aligned with business objectives. Continuous Integration and Deployment (CI/CD) pipelines automated build, testing, and deployment processes, ensuring high code quality. Git was used for version control, following a branching strategy to manage development and releases effectively. Automated testing, along with deployment tools like Jenkins or GitHub Actions, minimized downtime and ensured smooth production releases.

## Key Benefits Achieved

* Scalability and Maintainability: Clean separation of concerns made it easier to extend and modify individual application components.
* Reduced Development Time: Spring Boot’s auto-configuration and dependency management significantly reduced manual effort.
* Improved Collaboration: Agile practices ensured that business needs were met through continuous feedback and iterative releases.
* Enhanced User Experience: Bootstrap’s responsive design ensured that the application adapted well to various devices and screen sizes.

## Key Parts Of The Report

The project report is structured to provide a comprehensive and detailed analysis of the development process, highlighting the project's purpose, technical architecture, and overall impact. It begins with an introduction that outlines the primary objective of the project, emphasizing its goal to create a seamless and efficient platform that caters to different stakeholders, including users, restaurants, delivery personnel, and administrators. The requirement analysis section delves into the meticulous process of gathering both functional and non-functional requirements. This phase involved extensive interaction with stakeholders through interviews and surveys to define user stories and use cases that reflected real-world needs. During this phase, critical design considerations, including scalability, security, and performance optimization, were also taken into account to ensure that the system was capable of handling increasing user loads and adapting to future business requirements.

## Benefits Of The Company

Incedo specializes in delivering industry-specific solutions across multiple domains using cutting-edge technologies. The company focuses on building strong client relationships by ensuring timely delivery of projects within budget. With expertise in emerging technologies like AI, ML, and Cloud Computing, Incedo provides scalable and future-ready solutions. Its agile approach ensures flexibility, allowing clients to adapt to changing business needs efficiently.

.

# TABLE OF CONTENTS

|  |  |  |
| --- | --- | --- |
| **INTERNSHIP OFFER LETTER** | | **iii** |
| **ACKNOWLEDGEMENT** | | **v** |
| **ABSTRACT** |  | **vi** |
| **LIST OF FIGURES** | | **xii** |
| **LIST OF TABLES** |  | **xiii** |
| **LIST OF ACRONYMS AND ABBREVIATIONS** | | **xiv** |
| **CHAPTER NO.** | **TITLE** | **PAGE NO.** |
| **1** | **INTRODUCTION** | **1** |
|  | 1.1 Overview | 1 |
|  | 1.2 Objectives Of The System | 1 |
|  | 1.3 Scope Of The Project | 2 |
|  | 1.4 Technologies Used | 2 |
|  | 1.4.1 Backend Technologies | 2 |
|  | 1.4.2 Frontend Technologies | 3 |
|  | 1.4.3 Database and Infrastructure | 3 |
| **2** | **LITERATURE SURVEY** | **4** |
|  | 2.1 Existing Food Delivery Systems | 4 |
|  | 2.2 Challenges in Online Food Delivery | 5 |
|  | 2.3 Proposed System | 5 |
| **3** | **SYSTEM REQUIREMENTS** | **6** |
|  | 3.1 Functional Requirements | 6 |
|  | 3.2 Software Requirements | 6 |
|  | 3.3 Hardware Requirements | 6 |
| **4** | **SYSTEM ARCHITECTURE AND DESIGN** | **7** |
|  | 4.1 System Overview | 7 |
|  | 4.2 Class Diagram | 7 |

|  |  |  |
| --- | --- | --- |
|  | 4.3 Sequence Diagram | 8 |
|  | 4.4 Data Flow Diagram | 9 |
| **5** | **METHODOLOGY AND IMPLEMENTATION** | **11** |
|  | 5.1 User Roles and Functionalities | 11 |
|  | 5.1.1 Admin | 11 |
|  | 5.1.2 User | 11 |
|  | 5.1.3 Restaurant | 11 |
|  | 5.1.4 Delivery Agent | 11 |
|  | 5.2 Database Design and Implementation | 12 |
|  | 5.2.1 Admin Schema | 12 |
|  | 5.2.2 User Schema | 13 |
|  | 5.2.3 Restaurant Schema | 13 |
|  | 5.2.4 Delivery Agent Schema | 14 |
|  | 5.3 Admin Dashboard and User Managment | 14 |
|  | 5.4 Customer Portal | 25 |
|  | 5.5 Restaurant Management Portal | 37 |
|  | 5.6 Delivery Agent Portal | 44 |
| **6** | **CONCLUSION AND FUTURE ENHANCEMENTS** | **52** |
|  | 6.1 Conclusion | 52 |
|  | 6.2 Future Scope | 52 |
|  | **REFERENCES** | **53** |
|  | **BIBILIOGRAPHY** | **54** |
| **APPENDIX** | | |
| **A** | **PLAGIARISM REPORT** | **55** |

**LIST OF FIGURES**

|  |  |  |
| --- | --- | --- |
| **FIG NO.** | **FIGURE NAME** | **PAGE NO.** |
| 4.1 | Class Diagram Of Korporate Kitchen | 7 |
| 4.2 | Sequence Diagram Of Korporate Kitchen | 8 |
| 4.3 | Data Flow Diagram Of Korporate Kitchen | 9 |
| 5.1 | Admin Schema | 12 |
| 5.2 | User Schema | 13 |
| 5.3 | Restaurant Schema | 13 |
| 5.4 | Delivery Agent Schema | 14 |
| 5.5 | Admin Portal | 20 |
| 5.6 | Admin Dashboard | 21 |
| 5.7 | Admin Profile | 22 |
| 5.8 | User Management | 23 |
| 5.9 | Restaurant Management | 24 |
| 5.10 | Delivery Agents Management | 25 |
| 5.11 | Customer Home Page | 30 |
| 5.12 | Login/Registration Page | 31 |
| 5.13 | User Dashboard | 34 |
| 5.14 | User Cart | 35 |
| 5.15 | User Remarks | 36 |
| 5.16 | Restaurant Portal | 42 |
| 5.17 | Restaurant Dashboard | 44 |
| 5.18 | Delivery Partner Portal | 49 |
| 5.19 | Delivery Partner Registration | 50 |
| 5.20 | Delivery Partner Dashboard | 51 |

**LIST OF TABLES**

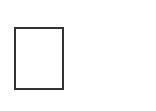
**WEEKLY OVERVIEW OF INTERNSHIP ACTIVITIES**

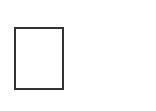
|  |  |
| --- | --- |
| **WEEK NO.** | **PAGE NO.** |
| WEEK-1 | xv |
| WEEK-2 | xv |
| WEEK-3 | xvi |
| WEEK-4 | xvi |

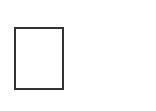
**LIST OF ACRONYMS AND ABBREVIATIONS**

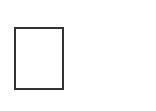
|  |  |
| --- | --- |
| JSP | Java Server Pages |
| JSTL | Java Server Pages Standard Tag Library |
| JDK | Java Development Kit |
| IDE | Integrated Development Environment |
| MVC | Model-View-Controller |
| REST | Representational State Transfer |
| UI/UX | User Interface/User Experience |
| API | Application Programming Interface |
| CRUD | Create, Read, Update, Delete |
| JPA | Java Persistence API |
| CSS | Cascading Style Sheets |
| SQL | Structured Query Language |
| VRPTW | Vehicle Routing Problem with Time Windows |
| UML | Unified Markup Language |
| ORM | Object-Relational Mapping |
| GPS | Global Positioning System |
| UPI | Unified Payments Interface |

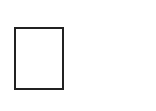
Learning Objectives/Internship Objectives

Internships are generally thought of to be reserved for college students looking to gain experience in a particular field. However, a wide array of people can benefit from Training Internships in order to receive real world experience and develop their skills.

An objective for this position should emphasize the skills you already possess in the area and your interest in learning more

Internships are utilized in a number of different career fields, including architecture, engineering, healthcare, economics, advertising and many more.

Some internship is used to allow individuals to perform scientific research while others are specifically designed to allow people to gain first-hand experience working.

Utilizing internships is a great way to build your resume and develop skills that can be emphasized in your resume for future jobs. When you are applying for a Training Internship, make sure to highlight any special skills or talents that can make you stand apart from the rest of the applicants so that you have an improved chance of landing the position.

**WEEKLY OVERVIEW OF INTERNSHIP ACTIVITIES**

|  |  |  |  |
| --- | --- | --- | --- |
| 1st WEEK | DATE | DAY | NAME OF THE TOPIC/MODULE COMPLETED |
| 08/01/25 | Wednesday | Introduction to Korporate Kitchen |
| 09/01/25 | Thursday | Understanding Business Requirements |
| 10/01/25 | Friday | Setting up Development Environment |

|  |  |  |  |
| --- | --- | --- | --- |
| 2nd WEEK | DATE | DAY | NAME OF THE TOPIC/MODULE COMPLETED |
| 13/01/25 | Monday | Database Schema and Planning |
| 14/01/25 | Tuesday | Setting up Spring Boot & Data Base Connection |
| 15/01/25 | Wednesday | Implementing User Authentication & Authorization |
| 16/01/25 | Thursday | Designing User Management Module |
| 17/01/25 | Friday | Continuing windows services |

|  |  |  |  |
| --- | --- | --- | --- |
| 3rd WEEK | DATE | DAY | NAME OF THE TOPIC/MODULE COMPLETED |
| 20/01/25 | Monday | Integrating User Profiles & Settings |
| 21/01/25 | Tuesday | Implementing API Security & JWT Authentication |
| 22/01/25 | Wednesday | Developing order Management Module |
| 23/01/25 | Thursday | Implementing Order Placement |
| 24/01/25 | Friday | Integrating Payment Gateway |

|  |  |  |  |
| --- | --- | --- | --- |
| 4th WEEK | DATE | DAY | NAME OF THE TOPIC/MODULE COMPLETED |
| 27/01/25 | Monday | Handling Order Status and Notifications |
| 28/01/25 | Tuesday | Implementing Order History & Tracking |
| 29/01/25 | Wednesday | Conducting System – Wide Testing |
| 30/01/25 | Thursday | Implementing Performance Enhancements |
| 31/01/25 | Friday | User Acceptance Testing |

* 1. **Overview**

# CHAPTER 1 INTRODUCTION

Korporate Kitchen is an innovative online food delivery system designed specifically for corporate professionals, providing them with healthy and customizable meal plans. The platform bridges the gap between restaurants, cloud kitchens, and office-goers, offering seamless meal ordering, subscription-based services, and real-time tracking. This solution addresses the common challenge faced by employees in finding healthy, affordable, and timely meal options.

Unlike existing platforms, Korporate Kitchen focuses exclusively on corporate employees who require timely and well-balanced meals during work hours. The system integrates advanced technologies, ensuring efficiency and reliability in food delivery services. By leveraging a robust microservices architecture, Korporate Kitchen supports multiple payment gateways, user role management, and an intuitive UI/UX for a smooth ordering experience.

Key features of Korporate Kitchen include personalized meal recommendations, flexible subscription plans, corporate partnerships, real-time order tracking, and analytics for optimizing restaurant operations. This project aims to redefine corporate food delivery through a technology-driven approach, prioritizing user convenience, health, and efficiency. By doing so, Korporate Kitchen seeks to enhance the overall dining experience for corporate professionals.

## Objectives Of The System

The primary objective of Korporate Kitchen is to streamline and enhance the corporate food delivery experience by integrating all stakeholders—employees, restaurants, delivery agents, and administrators—into a single cohesive platform. The system aims to:

* + 1. Seamless Online Ordering:

Hassle-free ordering experience for employees to browse menus and place orders.

* + 1. Streamlined Restaurant Operations:

Efficient menu management, order optimization, and kitchen workflows.

* + 1. Robust Security:

Role-based access control and data privacy measures.

* + 1. Secure Transactions:

Multiple secure payment gateways for reliable transactions.

* + 1. Intuitive User Interface:

Convenient, accessible, and high-performance UI/UX for users.

## Scope Of The Project:

The scope of this project encompasses an online food delivery platform that unites customers, restaurants, delivery personnel, and administrators for streamlined order fulfillment. Some of the key features include User registration and browsing of restaurant menus, Secure payment processing for online orders, Restaurant management of menus and order preparation, Delivery agent assignment and status updates, Automated notification system for users, restaurants, and agents, Administrative oversight of user, restaurant, and delivery management.

## Technologies Used:

* + 1. Backend Technologies
       - Spring Boot is used to build RESTful APIs for user management, order processing, and payment handling. It simplifies deployment with an embedded Tomcat server and integrates PostgreSQL using Spring Data JPA. Auto-configuration and security features streamline authentication, role management, and database connectivity.
       - Hibernate (JPA) manages database interactions for users, orders, restaurants, and delivery agents. It maps Java entities to PostgreSQL tables, enabling seamless CRUD operations using Spring Data JPA.
       - Spring Data JPA simplifies database operations for entities like users, orders, and restaurants. It enables CRUD operations through JpaRepository, reducing boilerplate code. Additionally, it supports custom queries and pagination, making data retrieval efficient and scalable.
    2. Frontend Technologies
       - JSP and JSTL are used for creating dynamic web pages like user dashboards, restaurant menus, and order history. JSP helps generate dynamic content by interacting with Servlets, while JSTL simplifies looping, conditionals, and formatting in the UI. This ensures a cleaner, more maintainable front-end.
       - Bootstrap is used to design a responsive and visually appealing UI for user dashboards, restaurant listings, menus, and order pages. It ensures mobile- friendliness, provides pre-styled components like buttons and forms, and helps maintain a consistent layout across different devices.
       - Font Awesome is used to add icons for navigation menus, buttons, cart items, and order status indicators. It enhances user experience by providing visual cues, making the interface more intuitive and interactive. Icons are easily customized for size, color, and placement using CSS.
    3. Database and Infrastructure
       - MySQL is used as the primary database to store user details, restaurant data, menu items, orders, and delivery information. It ensures data integrity, fast queries, and secure storage for all transactions. Spring Boot + Hibernate (JPA) seamlessly integrates with MySQL to perform CRUD operations efficiently.
       - MySQL Workbench is used to design the database schema, write and execute SQL queries, manage user access, and monitor database performance.

# CHAPTER 2 LITERATURE SURVEY

## Existing Food Delivery Systems:

Daman Deep Singh et al. discusses on the topic “Towards Fairness in Online Service with k Servers and its Application on Fair Food Delivery" where the k-FOOD problem, a realistic generalization of the k-SERVER problem, to model real-world applications like food delivery and ride sharing. The k-FOOD problem is strongly NP-hard, and an optimal offline algorithm is developed using a time-expanded flow network. An online algorithm, DOC4FOOD, is proposed and shown to be effective through experiments on real-world and synthetic datasets. [1]

Yang Liu et al. explores a hybrid delivery model on "Joint Infrastructure Planning and Order Assignment for On-Demand Food-Delivery Services with Coordinated Drones and Human Couriers" where optimal infrastructure planning and order assignment for an on-demand food delivery platform using a mixed fleet of drones and human couriers. A novel neural network- assisted optimization method is developed to solve the mixed-integer nonlinear program, minimizing operational costs and maximizing order bundling opportunities. The approach is validated through a case study in Hong Kong, revealing benefits and trade-offs of drone delivery. [2]

Manas Joshi et al. designed an algorithm on “Batching and Matching for Food Delivery in Dynamic Road Networks” to minimize delivery times by effectively assigning orders to vehicles and forming optimal batches in dynamic road conditions. The FoodMatch algorithm is developed to minimize food delivery time by assigning orders to vehicles, grouping orders into batches, and adapting to dynamic vehicle positions. FoodMatch maps the vehicle assignment problem to minimum weight perfect matching on a bipartite graph, reducing construction costs through best- first search. Experiments on real-world food-delivery data show FoodMatch outperforms baseline strategies while handling large workloads efficiently.[3]

Slavomír Švancár et al. discusses the Cloud Kitchen platform on "Cloud Kitchen: Using Planning-based Composite AI to Optimize Food Delivery Processes". The Cloud Kitchen platform utilizes AI to optimize food delivery, providing a decision-making tool for restaurants and a simulator to evaluate decision impact. The platform employs a Vehicle Routing Problem

with Time Windows (VRPTW) to allocate orders to vehicles and prioritize customer service. Results show improved customer satisfaction through reduced delayed deliveries using a real- world dataset. [4]

Ashman Mehra et al. introduces DeliverAI, "DeliverAI: Reinforcement Learning Based Distributed Path-Sharing Network for Food Deliveries". The existing food delivery model is sub- optimal, with each delivery optimized individually, resulting in high costs. DeliverAI, a reinforcement learning-based path-sharing algorithm, is proposed to optimize consumer satisfaction and delivery costs. [5]. [6] concludes that online food delivery services are working as a bridge between hotel business and consumers. They make people life’s easy and people also depending on online food delivery services.

X.Liu, X-J. Lim investigated the different attributes (i.e. convenience, online reviews, online ratings, visual appeal and various food choices) of food delivery applications (FDAs) that impacted customers' perceived benefits, satisfaction and loyalty. [[7]](https://paperpile.com/c/jRSMUt/XOv2)

## Challenges in Online Food Delivery:

The challenges in existing food delivery systems include inefficient route optimization, leading to delayed deliveries and higher operational costs. Fairness in order assignments remains a concern, as certain users or areas may experience service delays. The integration of drones and human couriers faces infrastructure and regulatory hurdles. Batching and matching of orders require real-time adaptability to dynamic traffic conditions. Lastly, high fleet costs and underutilization reduce efficiency, necessitating AI-driven solutions for better resource allocation.

## Proposed System:

The proposed system ensures a seamless online ordering experience, optimizing restaurant operations with efficient menu management and order processing. It enhances security through role-based access control and secure transactions via multiple payment gateways. With an intuitive user interface, it offers a fast, secure, and efficient platform for both users and restaurants.

# CHAPTER 3 SYSTEM REQUIREMENTS

## Functional Requirements

The software requirement specification can produce at the culmination of the analysis task. The function and performance allocated to software as part of system engineering are refined by established a complete information description, a detailed functional description, a representation of system behavior, and indication of performance and design constrain, appropriate validate criteria, and other information pertinent to requirements.

## Software Requirements

* + - Operating system: Windows 11 (64-bit).
    - Coding Language: Java 11
    - IDE: Spring Tool Suite (STS) 4
    - Application Server**:** Apache Tomcat 10.1.39
    - Database management tool: MySQL Workbench 8.0.41
    - Build Tool**:** Maven 3.9.9

## Hardware Requirements

* + - Processor**:** Intel Core i5
    - Hard Disk**:** Minimum 500 GB
    - RAM**:** 16 GB

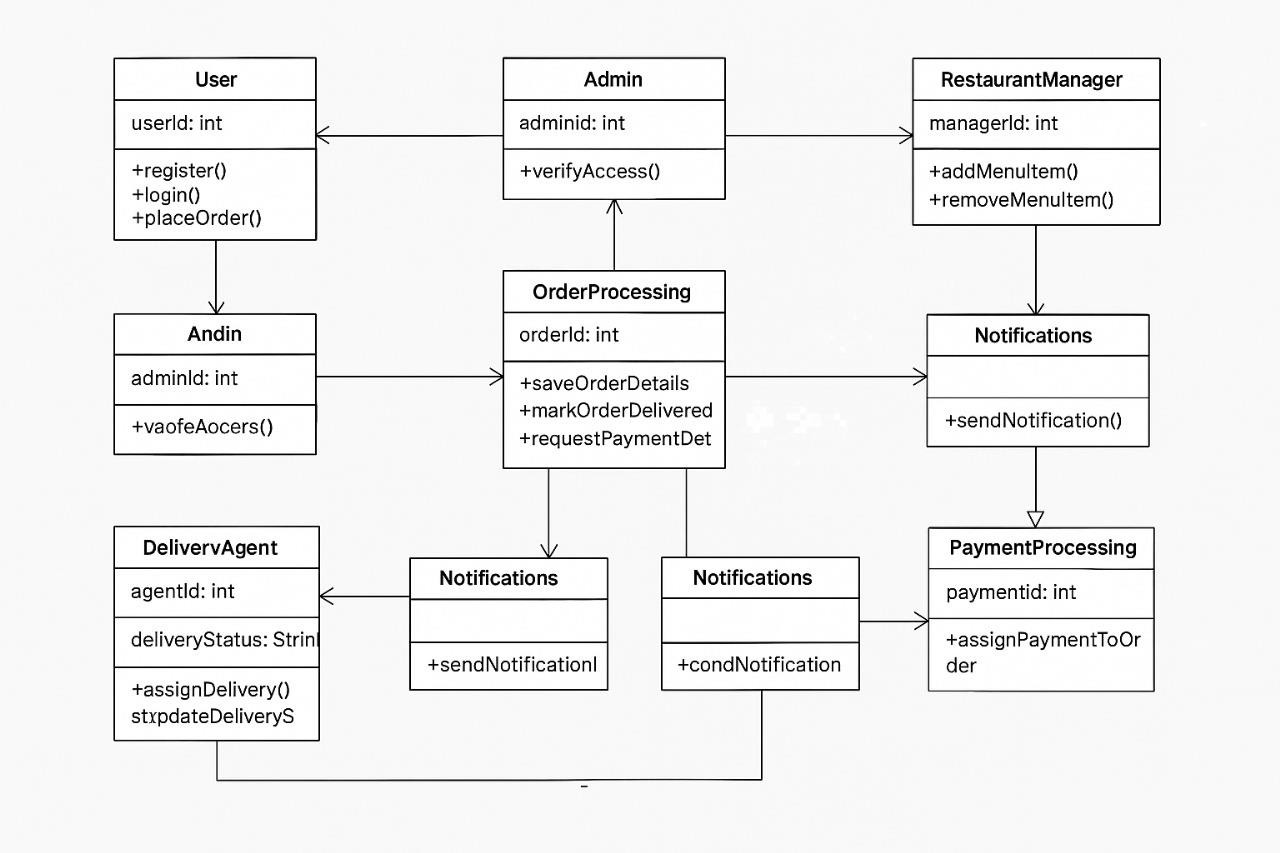
# CHAPTER 4

**SYSTEM ARCHITECTURE AND DESIGN**

## System Overview

Korporate Kitchen is a centralized food ordering and delivery system designed for corporate environments. It streamlines the process of ordering food, managing restaurant operations, and ensuring secure transactions. The system follows a modular architecture where users, restaurants, and administrators interact through a well-defined workflow. The system incorporates various UML Diagrams including Use Case diagram, Class diagram, Sequence diagram and Data Flow Diagrams to ensure a seamless, secure, and efficient food ordering experience for corporate employees while optimizing restaurant operations.

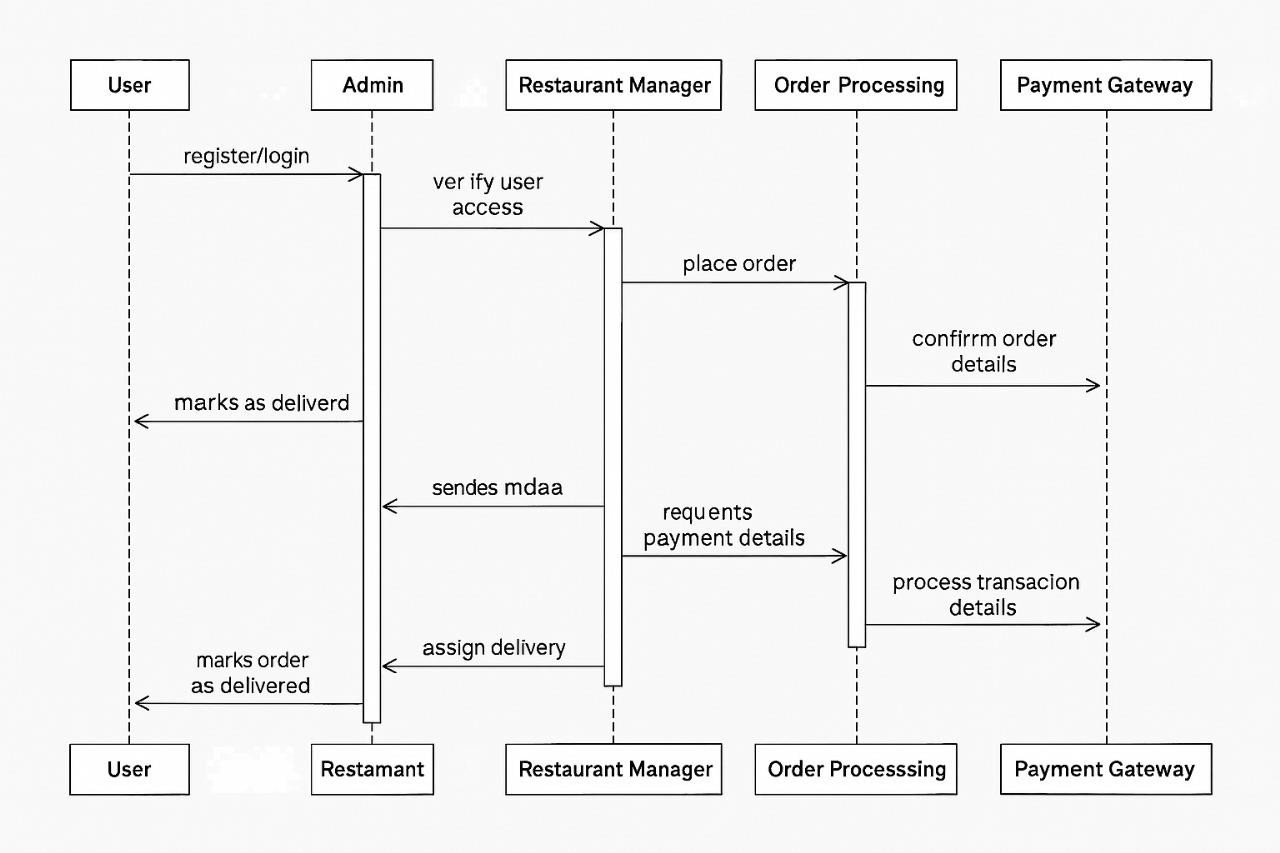
## Class diagram



**Fig 4.1 Class Diagram Of Korporate Kitchen**

Fig 4.1 follows a structured design with interconnected entities. The MasterUser and Role classes handle user authentication, registration, and role-based access. The Orders and Cart modules manage food ordering, item selection, and the checkout process, while Payments and Payment Gateway ensure secure transactions through multiple payment methods. The Delivery and Order History components track order status, delivery progress, and user purchase history. Restaurants manage their offerings through MasterMenuItems, and users can provide feedback using the Remarks module. Additionally, the Notification system sends real-time updates, and the Bank entity verifies transactions, ensuring a seamless and secure food ordering experience.

## Sequence diagram

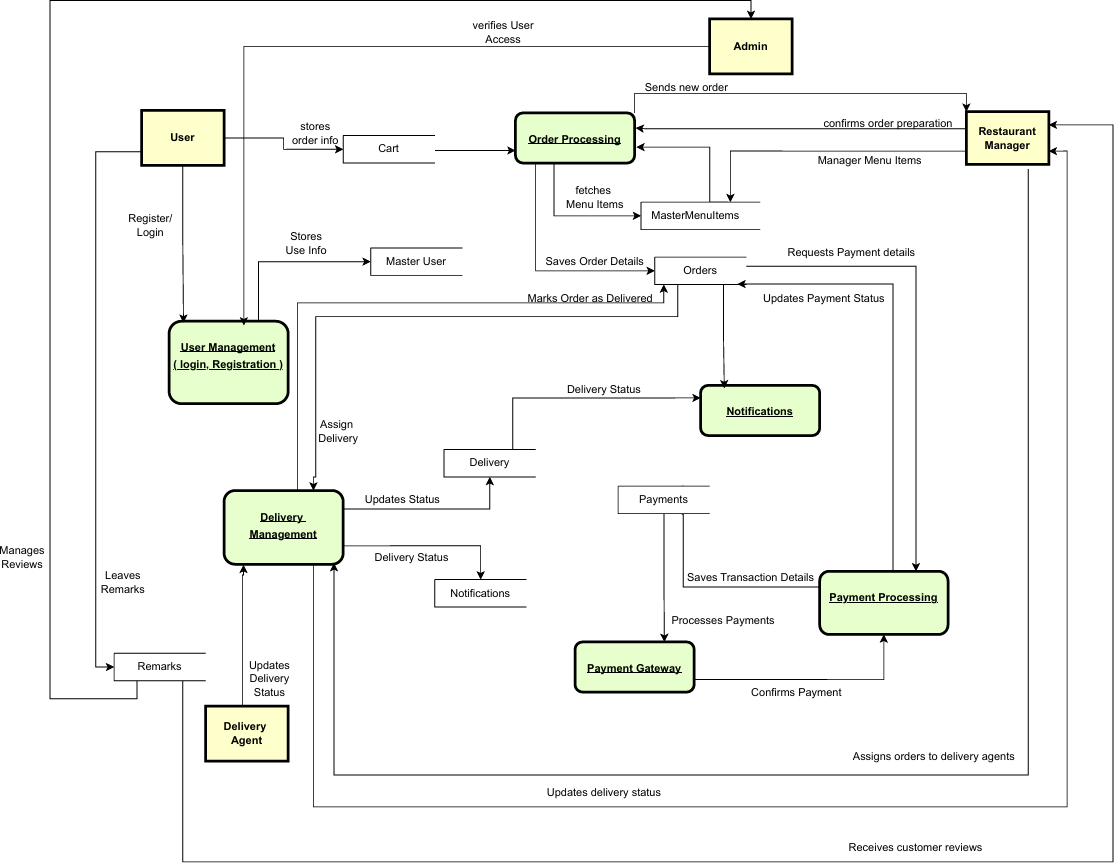


**Fi g 4.2 Sequence Diagram Of Korporate Kitchen**

Fig 4.2 represents the interactions between different actors such as User, Admin, DeliveryAgent, RestaurantManager, and PaymentGateway with system components like MasterUser, Orders, Payments, PaymentGateway, Delivery, Remarks, UserFavorites, Cart, OrderHistory, Bank, Notifications, and MasterMenuItems. The process starts with User actions like Login(), Register(), and ForgotPassword(), followed by order-related operations such as

PlaceOrder(), UpdateOrderStatus(), CalculateTotalCost(), and CancelOrder(). The PaymentGateway handles ProcessPaymentRequest() and PaymentSuccessful(), while the DeliveryAgent performs UpdateDeliveryStatus() and ViewDeliveryHistory().The Admin manages operations like AddNewMenuItem(), RemoveItem(), and UpdateItemDetails(), while the Bank verifies transactions through CheckTransactionStatus().

## Data Flow Diagram



**Fig 4.3 Data Flow Diagram Of Korporate Kitchen**

Fig 4.3 illustrates the flow of data between various system components, ensuring seamless order processing and delivery. The process begins with the User, who registers or logs in through the User Management module, which stores user information in the Master User database. When a user places an order, the Cart stores the order information before sending it to the Order Processing module, which fetches menu items from MasterMenuItems and saves order details. The Restaurant Manager confirms order preparation, while the system requests payment details. The Payment Processing module communicates with the Payment Gateway,

which processes payments and updates the transaction status. Upon successful payment confirmation, the system assigns delivery to Delivery Management, which in turn updates the Delivery Agent on the status. Delivery updates are sent back to Notifications for real-time tracking. The Admin oversees user access and manages new orders, while users can leave remarks and manage reviews.

# CHAPTER 5 METHODOLOGY AND IMPLEMENTATION

## User Roles and Functionalities

* + 1. **Admin:**
       - User & Restaurant Management: Admin manages users, assigns roles, approves restaurants, and monitors their activities.
       - Order & Delivery Oversight: Assigns delivery agents, tracks orders, and resolves issues related to delays or incorrect deliveries.
       - Payment & Dispute Resolution: Oversees transactions, handles refunds, and addresses payment-related disputes.

## User:

* + - * Account & Profile Management: Users can register, log in, update their profiles, and manage account details.
      * Order & Payment Processing: Browse restaurant menus, place orders, make secure payments, and track order status.
      * Feedback & Support: Rate orders, provide feedback, and raise complaints for issue resolution.

## Restaurant:

* + - * Menu & Order Management: Add, update, and manage menu items, process incoming orders, and track order status.
      * Payment & Financial Management: Handle payment transactions, view earnings, and manage bank details for settlements.
      * Restaurant Operations: Update restaurant details, manage working hours, and monitor customer feedback for service improvement.

## Delivery Agent:

* + - * Order Handling: Accept delivery requests, pick up orders from restaurants, and ensure timely delivery to customers.
      * Delivery Tracking: Update order status in real-time, including pickup, transit, and successful delivery.

## Database Design and Implementation

The database is designed using MySQL, with tables for Users, Orders, Restaurants, Payments, and Delivery Agents. The entity relationships ensure seamless order processing and tracking.

* + 1. Admin Schema

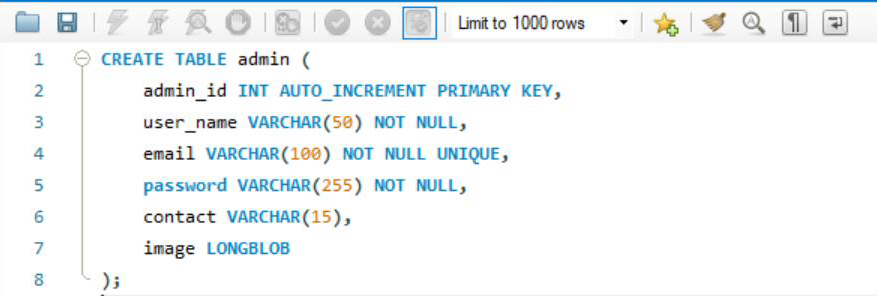


Fig 5.1 Admin Schema Fig 5.1 shows the following:

* + - * admin\_id is the primary key and auto-increments.
      * user\_name is required and limited to 50 characters.
      * email is required, unique, and limited to 100 characters.
      * password is required and can hold up to 255 characters.
      * contact is optional and limited to 15 characters.
      * image is stored as a large binary object (LONGBLOB)
    1. User Schema

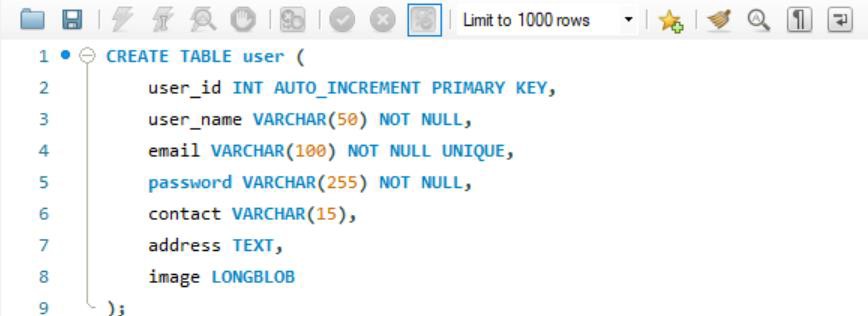


Fig 5.2 User Schema

Fig 5.2 shows the following:

* + - * user\_id is the primary key with auto-increment.
      * user\_name is required (NOT NULL) and has a 50-character limit.
      * email is unique and required, with a 100-character limit.
      * password is required and supports secure storage with a 255-character limit.
      * contact is optional and can store up to 15 characters.
      * address is stored as TEXT to handle different address lengths.
      * image is stored as a large binary object (LONGBLOB) for profile pictures.
    1. Restaurant Schema:

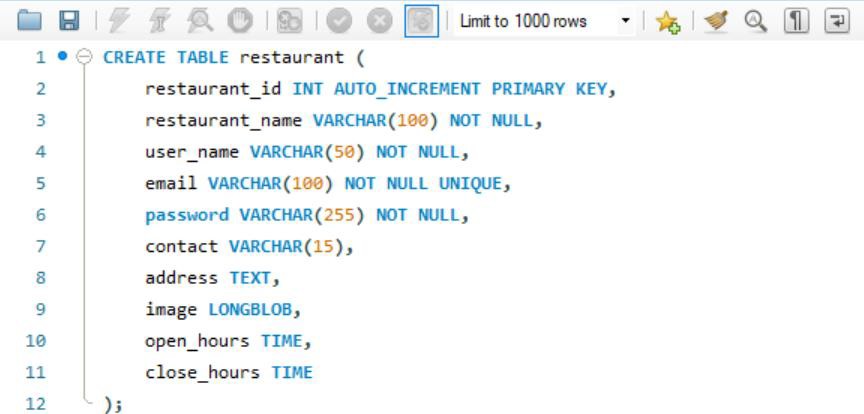


Fig 5.3 Restaurant Schema Fig 5.3 shows the following:

* + - * restaurant\_id is the primary key with auto-increment.
      * restaurant\_name is required (NOT NULL) and can store up to 100 characters.
      * user\_name is required and limited to 50 characters.
      * email is unique and required, allowing 100 characters.
      * password is required and stored securely with a 255-character limit.
      * contact is optional and can store up to 15 characters.
      * address is stored as TEXT to support various lengths.
      * image is stored as a large binary object (LONGBLOB).
      * open\_hours and close\_hours store the opening and closing times as TIME format.
    1. Delivery Agent Schema:

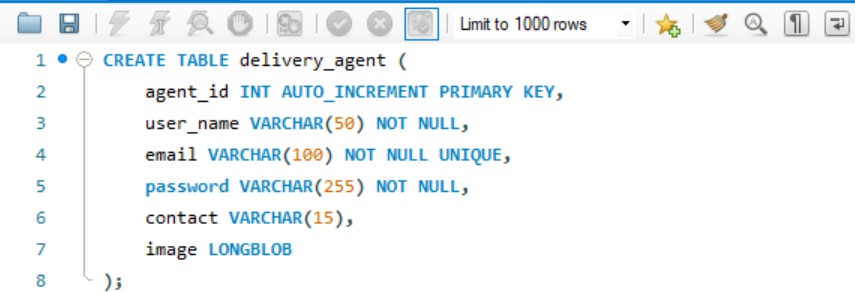


Fig 5.4 Delivery Agent Schema Fig 5.4 shows the following:

* + - * agent\_id is the primary key with auto-increment.
      * user\_name is required (NOT NULL) and can store up to 50 characters.
      * email is unique and required, allowing 100 characters.
      * password is required and stored securely with a 255-character limit.
      * contact is optional and can store up to 15 characters.
      * image is stored as a large binary object (LONGBLOB) for profile pictures.
  1. Admin Dashboard and User Management

The admin dashboard allows user management, order tracking, and delivery assignments.

## Admin.java

package com.example.demo.model; import jakarta.persistence.\*;

@Entity

@Table(name = "admin") public class Admin {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY) private int admin\_id;

@Column(name = "user\_name", nullable = false, length = 50) private String userName;

@Column(nullable = false, unique = true, length = 100) private String email;

@Column(nullable = false, length = 255) private String password;

@Column(length = 15) private String contact;

@Lob

private byte[] image;

// Getters and Setters public int getAdmin\_id() {

return admin\_id;

}

public void setAdmin\_id(int admin\_id) { this.admin\_id = admin\_id;

}

public String getUserName() { return userName;

}

public void setUserName(String userName) { this.userName = userName;

}

public String getEmail() { return email;

}

public void setEmail(String email) { this.email = email;

}

public String getPassword() { return password;

}

public void setPassword(String password) { this.password = password;

}

public String getContact() { return contact;

}

public void setContact(String contact) { this.contact = contact;

}

public byte[] getImage() { return image;

}

public void setImage(byte[] image) { this.image = image;

}

}

**Admin Service.java**

package com.example.demo.service; import com.example.demo.model.Admin; import java.util.List;

public interface AdminService { Admin saveAdmin(Admin admin); Admin getAdminById(int id);

Admin getAdminByEmail(String email); List<Admin> getAllAdmins();

void deleteAdmin(int id);

}

## Admin Service Impl.java

package com.example.demo.service; import com.example.demo.model.Admin;

import com.example.demo.repository.AdminRepository;

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

import java.util.List; @Service

public class AdminServiceImpl implements AdminService { @Autowired

private AdminRepository adminRepository;

@Override

public Admin saveAdmin(Admin admin) { return adminRepository.save(admin);

}

@Override

public Admin getAdminById(int id) {

return adminRepository.findById(id).orElse(null);}

@Override

public Admin getAdminByEmail(String email) { return adminRepository.findByEmail(email);

}

@Override

public List<Admin> getAllAdmins() { return adminRepository.findAll();

}

@Override

public void deleteAdmin(int id) { adminRepository.deleteById(id);

}

}

## Admin Controller.java

**// --- Portal ----** @GetMapping("/portal") public String adminPortal() {

return "adminPortal";

}

## // --- Registration Form ---

@GetMapping("/register")

public String showRegisterForm() { return "adminRegister";

}

## // ---- Process Registration ---

@PostMapping("/register")

public String registerAdmin(@ModelAttribute Admin admin, Model model) { Admin savedAdmin = adminService.saveAdmin(admin); model.addAttribute("savedAdmin", savedAdmin);

return "adminRegisterSuccess";

}

## // --- Login Form ---

@GetMapping("/login")

public String showLoginForm() { return "adminLogin";

}

## // --- Process Login ---

@PostMapping("/login")

public String loginAdmin(@RequestParam String email, @RequestParam String password, HttpSession session,

Model model) {

Admin admin = adminService.getAdminByEmail(email);

if (admin != null && admin.getPassword().equals(password)) { session.setAttribute("admin", admin);

return "redirect:/admin/dashboard";

} else {

model.addAttribute("error", "Invalid email or password"); return "adminLogin";

}

}

## // --- Logout ---

@GetMapping("/logout")

public String logout(HttpSession session) { session.invalidate();

return "redirect:/admin/portal";}

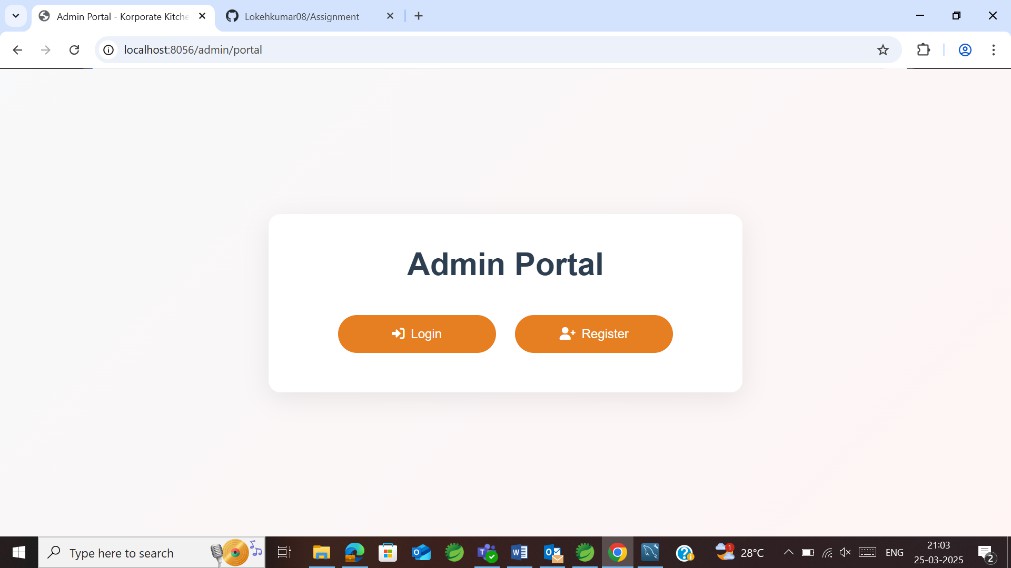


Fig 5.5 Admin Portal

Fig 5.5 allows administrators to register and securely log in using their credentials. After login, admins can access the dashboard to manage users, restaurants, orders, deliveries, and payments. Role-based authentication ensures only authorized admins can control system operations.

## // ---Dashboard & Management ---

**// --- Admin Dashboard – shows counts and a slide-out menu ---**

@GetMapping("/dashboard")

public String dashboard(Model model) {

long totalUsers = userService.countUsers();

long totalRestaurants = restaurantService.countRestaurants();

long totalDeliveryAgents = deliveryAgentService.countDeliveryAgents(); model.addAttribute("totalUsers", totalUsers); model.addAttribute("totalRestaurants", totalRestaurants); model.addAttribute("totalDeliveryAgents", totalDeliveryAgents);

return "adminDashboard";

}

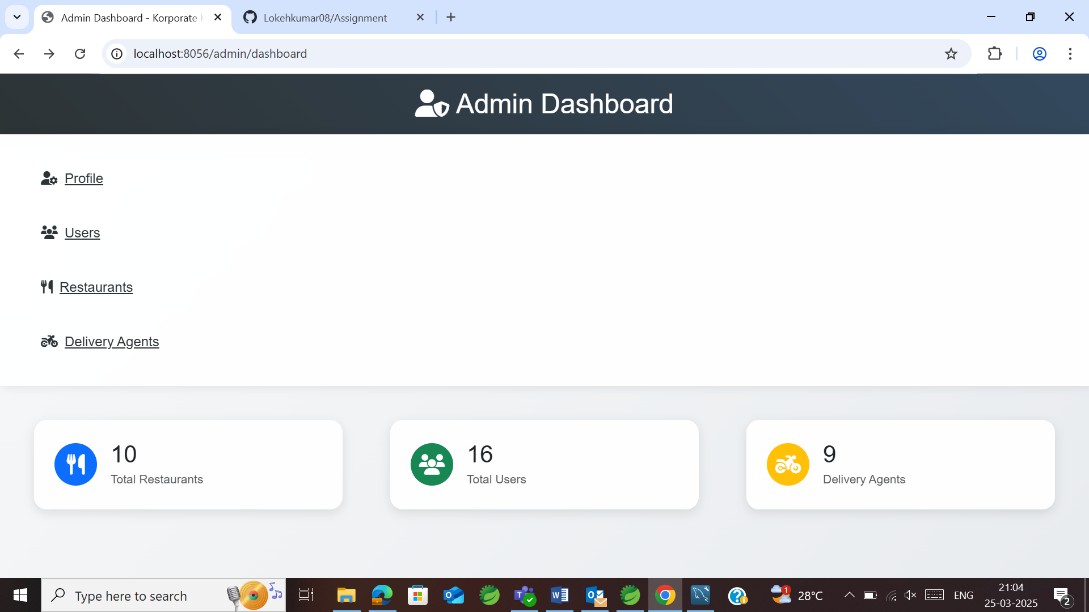


Fig 5.6 Admin Dashboard

Fig 5.6 shows the admin dashboard which allows the admin to view the total number of restaurants, registered users and delivery agents in partnership with them. In addition, the admin can also view their profile and update information accordingly.

## //--- Admin Profile Page---

@GetMapping("/profile")

public String adminProfile(HttpSession session, Model model) { Admin admin = (Admin) session.getAttribute("admin");

if (admin == null) {

return "redirect:/admin/login";

}

model.addAttribute("admin", admin); return "adminProfile";

}

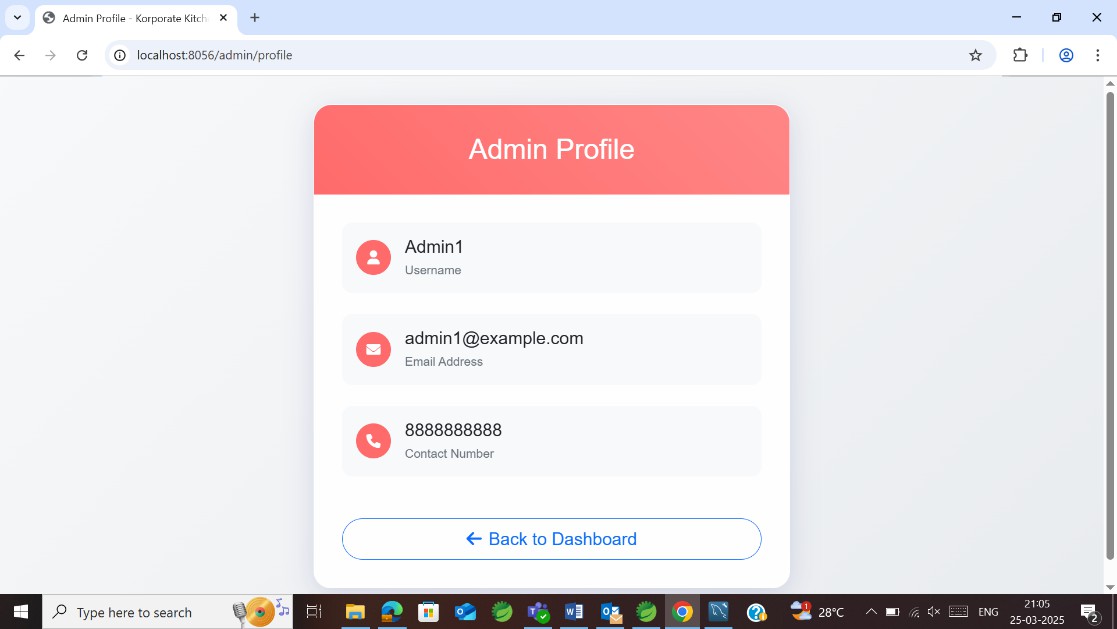


Fig 5.7 Admin Profile

Fig 5.7 shows the admin profile containing admin username,email address and contact number . In addition, the back to dashboard button is also present which directs the admin back to the admin dashboard.

## // ---User Management – list all users with delete option----

@GetMapping("/users")

public String userManagement(Model model) { List<User> users = userService.getAllUsers(); model.addAttribute("users", users);

return "userManagement";

}

@PostMapping("/deleteUser")

public String deleteUser(@RequestParam int userId) { userService.deleteUser(userId);

return "redirect:/admin/users";

}

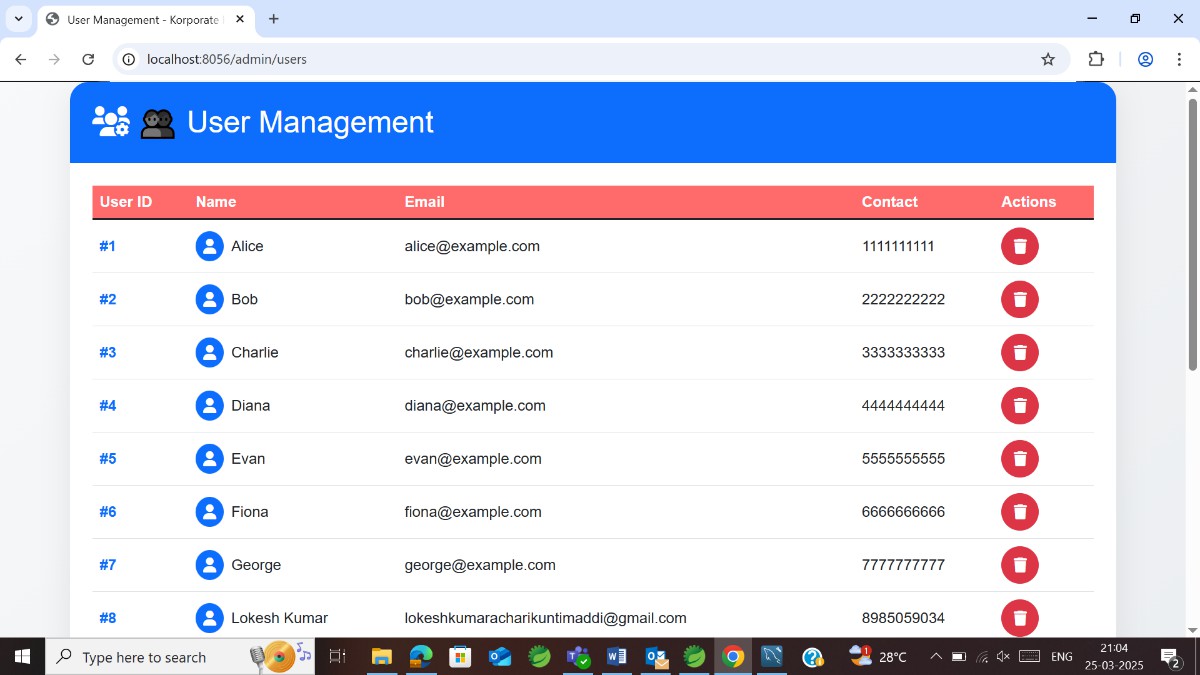


Fig 5.8 User Management

Figure 5.8 displays the list of users, including their user\_id, name, email, and contact details. The admin interface provides action buttons to manage users efficiently. The delete button allows the admin to remove a user from the system when necessary.

## // ---Restaurant Management – list all restaurants with delete option---

@GetMapping("/restaurants")

public String restaurantManagement(Model model) {

List<Restaurant> restaurants = restaurantService.getAllRestaurants(); model.addAttribute("restaurants", restaurants);

return "restaurantManagement";

}

@PostMapping("/deleteRestaurant")

public String deleteRestaurant(@RequestParam int restaurantId) { restaurantService.deleteRestaurant(restaurantId);

return "redirect:/admin/restaurants";

}

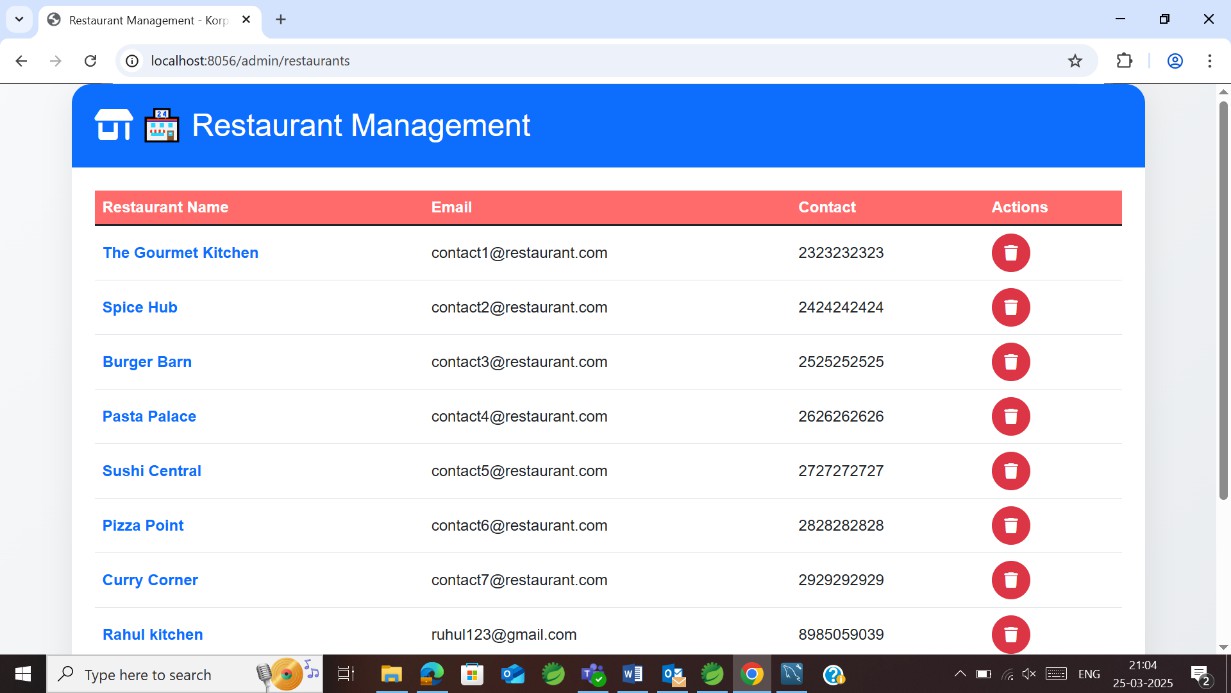


Fig 5.9 Restaurant Management

Figure 5.9 displays the list of restaurants, including restaurant\_name, email, and contact details. The admin interface provides action buttons for managing restaurant records. The delete button enables the admin to remove a restaurant from the system when needed.

## // ---Delivery Agent Management – list all delivery agents with delete option---

@GetMapping("/delivery")

public String deliveryAgentManagement(Model model) {

List<DeliveryAgent> agents = deliveryAgentService.getAllDeliveryAgents(); model.addAttribute("agents", agents);

return "deliveryAgentManagement";

}

@PostMapping("/deleteDeliveryAgent")

public String deleteDeliveryAgent(@RequestParam int agentId) { deliveryAgentService.deleteDeliveryAgent(agentId);

return "redirect:/admin/delivery";

}

}

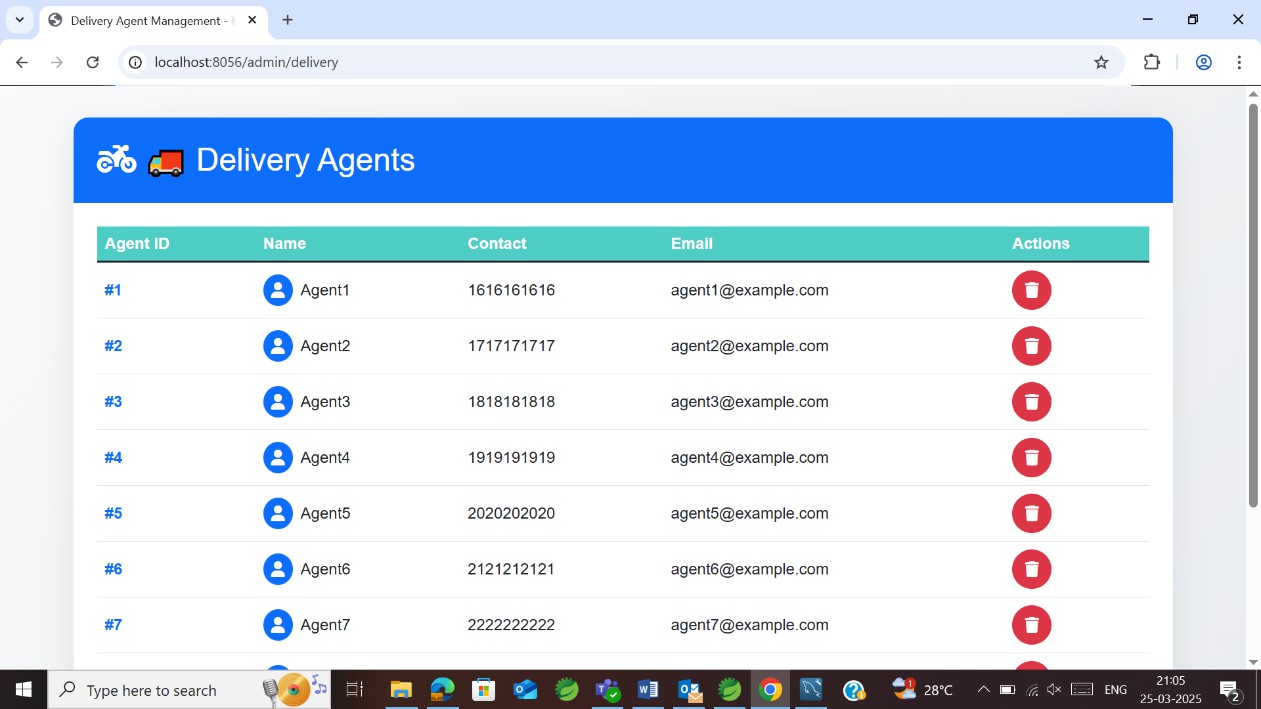


Fig 5.10 Delivery Agents Management

Figure 5.10 displays the list of delivery agents, including agent\_id,name, email, and contact details. The admin interface provides action buttons for managing agent records. The delete button enables the admin to remove the agent from the system when needed.

## Customer Portal

The User (Customer) registers, places and tracks orders, and provides feedback.

## User.java

package com.example.demo.model; import jakarta.persistence.\*; @Entity

@Table(name = "user") public class User {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY) @Column(name = "user\_id") // Database column remains "user\_id" private int userId; // Updated property name (camelCase)

@Column(name = "user\_name", nullable = false, length = 50)

private String userName;

@Column(nullable = false, unique = true, length = 100) private String email;

@Column(nullable = false, length = 255) private String password;

@Column(length = 15) private String contact;

@Column(columnDefinition = "TEXT") private String address;

@Lob

private byte[] image;

// Getters and Setters public int getUserId() {

return userId;

}

public void setUserId(int userId) { this.userId = userId;

}

public String getUserName() { return userName;

}

public void setUserName(String userName) { this.userName = userName;

}

public String getEmail() { return email;

}

public void setEmail(String email) { this.email = email;

}

public String getPassword() { return password;

}

public void setPassword(String password) { this.password = password;

}

public String getContact() { return contact;

}

public void setContact(String contact) { this.contact = contact;

}

public String getAddress() { return address;

}

public void setAddress(String address) { this.address = address;

}

public byte[] getImage() { return image;

}

public void setImage(byte[] image) { this.image = image;

}

}

## User Service.java

package com.example.demo.service; import com.example.demo.model.User; import java.util.List;

public interface UserService { User saveUser(User user);

User getUserByEmail(String email); List<User> getAllUsers();

void deleteUser(int id); long countUsers();

}

## User Service Impl.java

package com.example.demo.service; import com.example.demo.model.User;

import com.example.demo.repository.UserRepository;

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

import java.util.List; @Service

public class UserServiceImpl implements UserService { @Autowired

private UserRepository userRepository;

@Override

public User saveUser(User user) { return userRepository.save(user);

}

@Override

public User getUserByEmail(String email) { return userRepository.findByEmail(email);

}

@Override

public List<User> getAllUsers() { return userRepository.findAll();

}

@Override

public void deleteUser(int id) { userRepository.deleteById(id);

}

@Override

public long countUsers() { return userRepository.count();

}

}

## User Controller.java

@GetMapping("/home")

public String userHome(HttpSession session, Model model) { User user = (User) session.getAttribute("user");

if (user == null) {

*logger*.error("User not found in session for home"); return "redirect:/login";

}

model.addAttribute("user", user);

List<Restaurant> restaurants = restaurantService.getAllRestaurants(); model.addAttribute("restaurants", restaurants);

return "homeDashboard"; // Renders /WEB-INF/jsp/homeDashboard.jsp

}

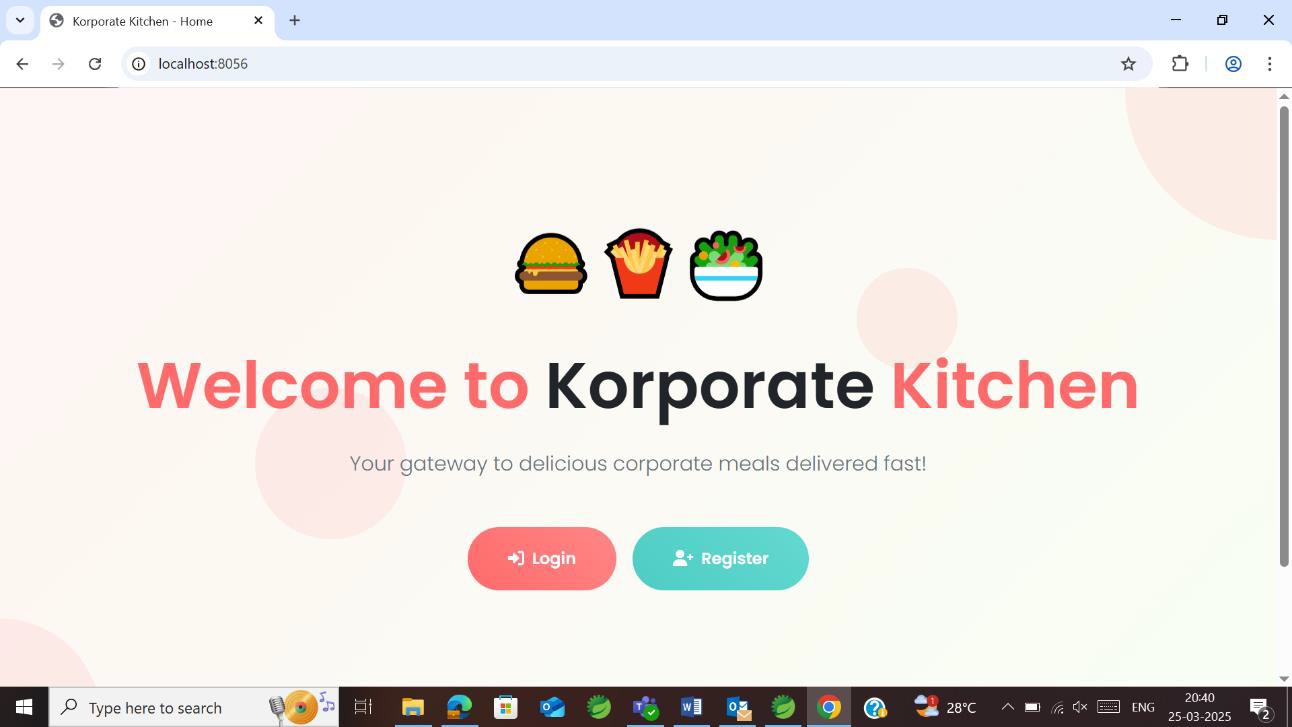


Fig 5.11 Customer Home page

Fig 5.11 displays the customer home page featuring a user-friendly interface. It includes login and register buttons for easy access. The design ensures a seamless navigation experience for users.

## // ---POST mapping for user login/registration---

@PostMapping("/login")

public String loginUser(@RequestParam String email, @RequestParam String password, HttpSession session,

Model model) {

User user = userService.getUserByEmail(email);

if (user != null && user.getPassword().equals(password)) { session.setAttribute("user", user);

*logger*.info("User logged in: {}", user.getUserName()); return "redirect:/user/home";

} else {

model.addAttribute("error", "Invalid email or password");

*logger*.error("Login failed for email: {}", email); return "login";

}}

## // --- POST mapping for user registration ---

@PostMapping("/register")

public String registerUser(@ModelAttribute User user, Model model) { User savedUser = userService.saveUser(user); model.addAttribute("registeredUser", savedUser);

*logger*.info("User registered: {}", savedUser.getUserName());

return "registerSuccess"; // Renders /WEB-INF/jsp/registerSuccess.jsp

}

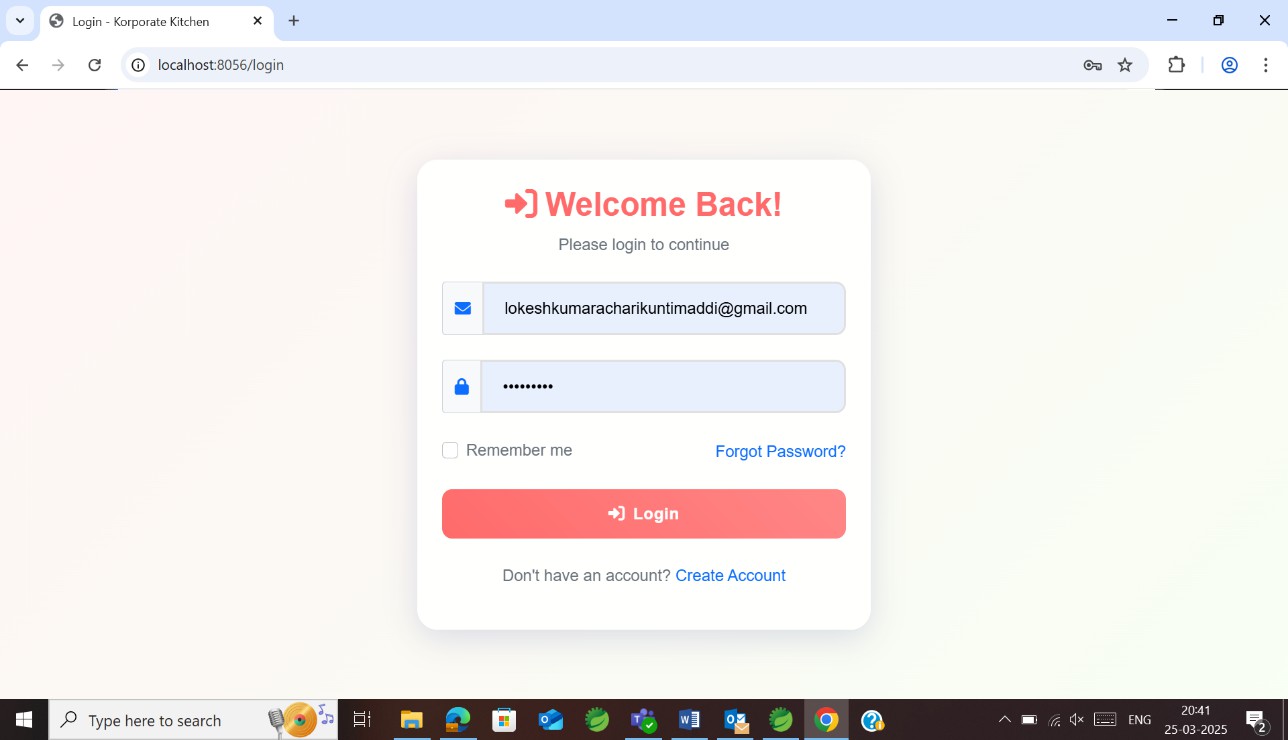


Fig 5.12 Login/Registration page

Fig 5.12 shows the login/registration page which allows users to sign in using their email ID and password. New users can register by providing their details, including email and a secure password. The interface ensures a smooth authentication process with validation checks.

## //--- Display Restaurant Menu – expects ?restaurantId= parameter---

@GetMapping("/restaurantMenu")

public String restaurantMenu(@RequestParam int restaurantId, Model model, HttpSession session)

{

Restaurant restaurant = restaurantService.getRestaurantById(restaurantId); if (restaurant == null) {

model.addAttribute("error", "Restaurant not found"); *logger*.error("Restaurant not found for id: {}", restaurantId); return "homeDashboard";

}

## // ---User Profile Page---

@GetMapping("/profile")

public String userProfile(HttpSession session, Model model) { User user = (User) session.getAttribute("user");

if (user == null) {

*logger*.error("User not found in session for profile"); return "redirect:/login";

}

model.addAttribute("user", user);

return "userProfile"; // Renders /WEB-INF/jsp/userProfile.jsp

}

## // ---Set currentRestaurant in session for order placement---

session.setAttribute("currentRestaurant", restaurant);

*logger*.info("Set currentRestaurant in session: {}", restaurant.getRestaurantName()); List<MasterMenuItem> menuItems =

masterMenuItemService.getMenuItemsByRestaurantId(restaurantId);

model.addAttribute("restaurant", restaurant); model.addAttribute("menuItems", menuItems);

return "restaurantMenu"; // Renders /WEB-INF/jsp/restaurantMenu.jsp

}

## // --- Display Cart Dashboard ---

@GetMapping("/cart")

public String viewCart(HttpSession session, Model model) { User user = (User) session.getAttribute("user");

if (user == null) {

*logger*.error("User not found in session for cart"); return "redirect:/login";

}

List<Cart> cartItems = cartService.getCartItemsByUserId(user.getUserId()); model.addAttribute("cartItems", cartItems);

BigDecimal total = BigDecimal.*ZERO*; for (Cart c : cartItems) {

total = total.add(c.getTotalPrice());

}

model.addAttribute("totalPrice", total);

return "cartDashboard"; // Renders /WEB-INF/jsp/cartDashboard.jsp

}

@GetMapping("/remarks")

**public** String showRemarks(HttpSession session, Model model) { User user = (User) session.getAttribute("user");

**if**(user == **null**) {

**return** "redirect:/login";

}

model.addAttribute("remarks", remarkService.getRemarksByUserId(user.getUserId()));

**return** "remarks"; // Renders /WEB-INF/jsp/remarks.jsp

}

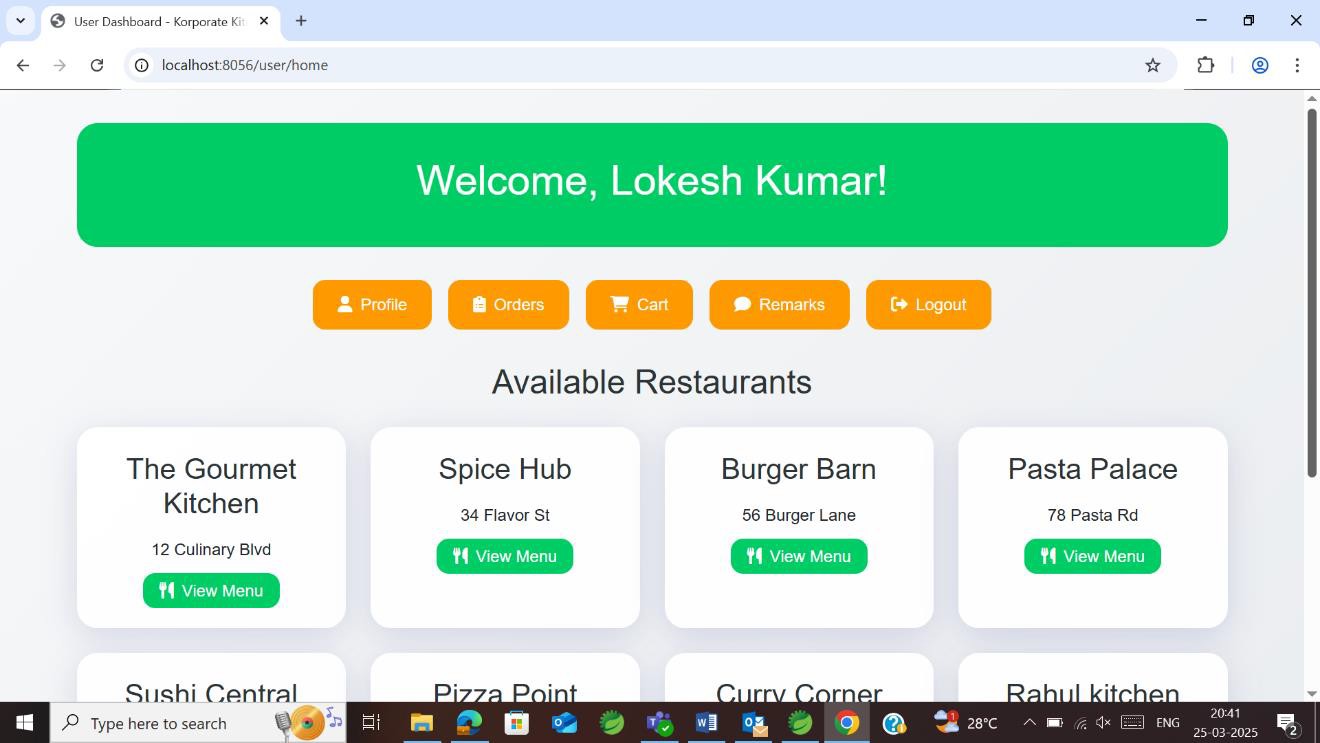


Fig 5.13 User Dashboard

Fig 5.13 shows the user dashboard page which contains the user profile, their orders, cart containing items to buy, remarks of the restaurant and a logout option. It also contains many restaurant options for the users to choose from.

## // --- Remove an item from the cart ---

@PostMapping("/removeFromCart")

public String removeFromCart(@RequestParam int cartId) { cartService.deleteCartItem(cartId);

*logger*.info("Removed cart item with id: {}", cartId); return "redirect:/user/cart";

}

## // --- Increase quantity endpoint ---

@PostMapping("/increaseQuantity")

public String increaseQuantity(@RequestParam int cartId) { cartService.incrementQuantity(cartId); *logger*.info("Increased quantity for cart id: {}", cartId); return "redirect:/user/cart";

}

## //--- Decrease quantity endpoint ---

@PostMapping("/decreaseQuantity")

public String decreaseQuantity(@RequestParam int cartId) { cartService.decrementQuantity(cartId); *logger*.info("Decreased quantity for cart id: {}", cartId); return "redirect:/user/cart";

}

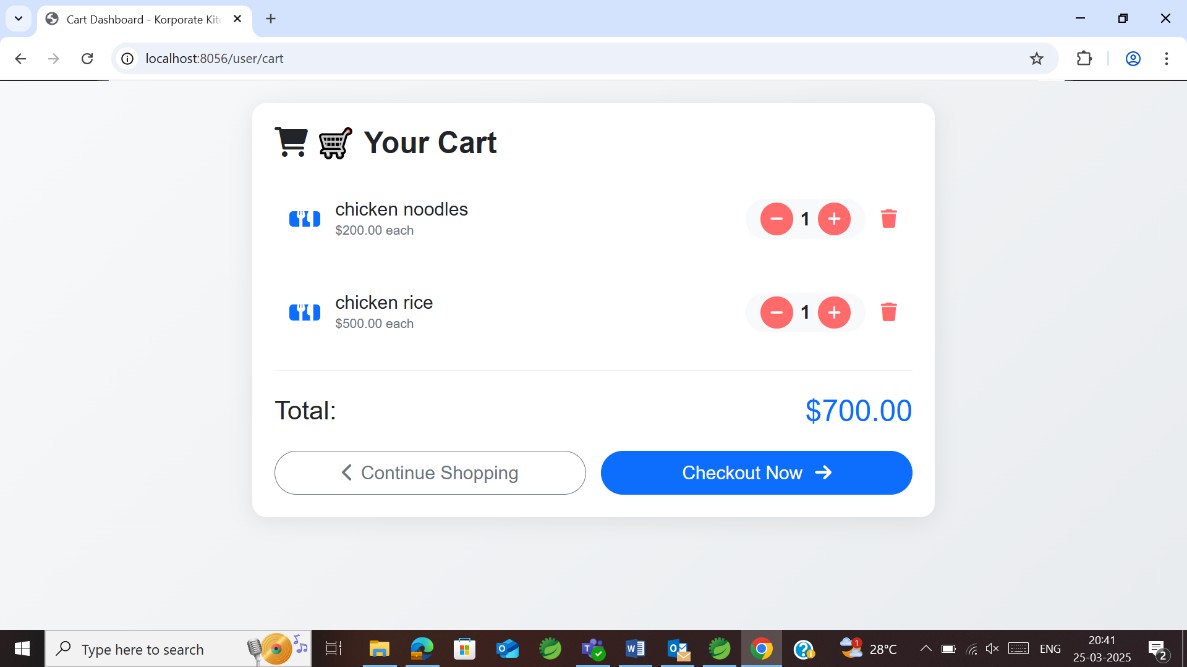


Fig 5.14 User Cart

Fig 5.14 displays selected items before placing an order. It shows item details, quantity, and total price, allowing users to review their selections. Users can modify the cart by adding or removing items before proceeding to checkout.

## // --- New: Show form to add a new remark ---

@GetMapping("/addRemark")

public String showAddRemarkForm(HttpSession session, Model model) { User user = (User) session.getAttribute("user");

if(user == null) {

return "redirect:/login";

}

return "addRemark"; // Renders /WEB-INF/jsp/addRemark.jsp

}

## // --- New: Process new remark submission ---

@PostMapping("/addRemark")

public String addRemark(@RequestParam String userRemark, HttpSession session) { User user = (User) session.getAttribute("user");

if(user == null) {

return "redirect:/login";

}

Remark remark = new Remark(); remark.setUser(user); remark.setUserRemark(userRemark); remarkService.saveRemark(remark); return "redirect:/user/remarks";

}

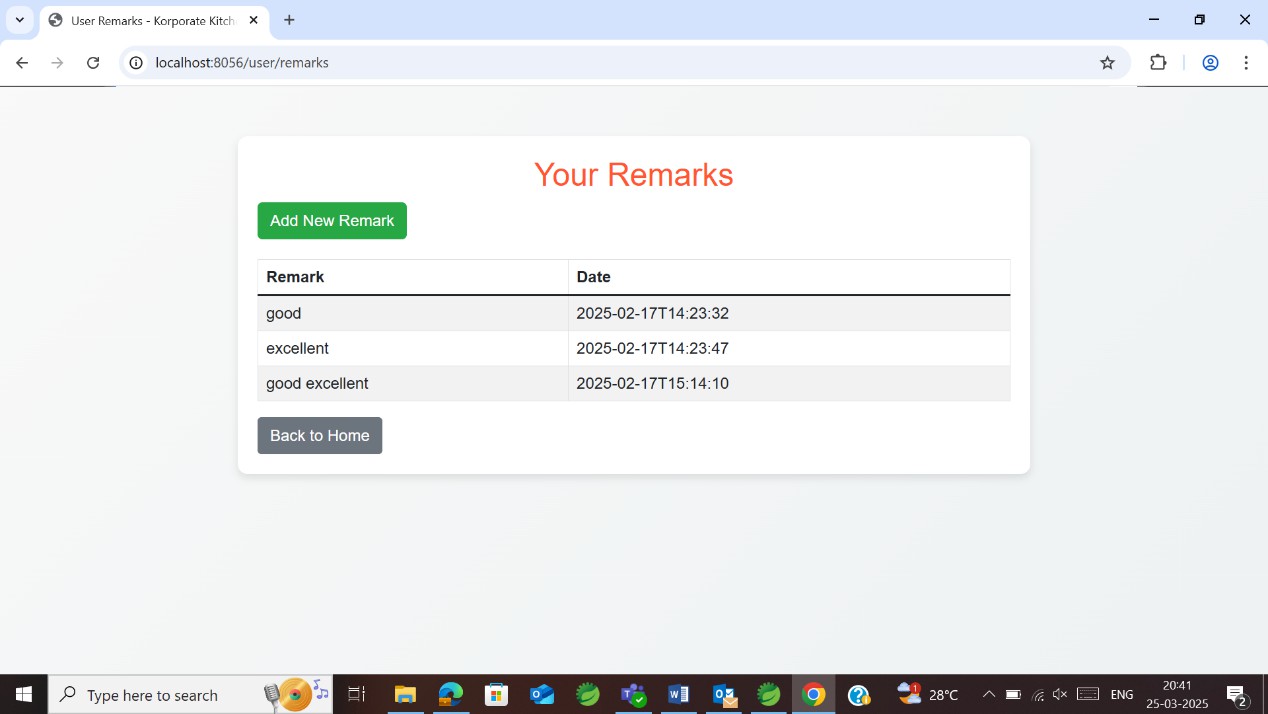


Fig 5.15 User Remarks

Fig 5.15 shows the user remarks page which records the remarks of the user along with the date. It also allows the user to add a new remark using the Add New Remark button.

## Restaurant Management Portal Restaurant.java

package com.example.demo.model; import jakarta.persistence.\*;

import java.sql.Time; @Entity

@Table(name = "restaurant") public class Restaurant {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

@Column(name = "restaurant\_id") // Database column remains "restaurant\_id" private int restaurantId; // Updated property name (camelCase)

@Column(name = "restaurant\_name", nullable = false, length = 100) private String restaurantName;

@Column(name = "user\_name", nullable = false, length = 50) private String userName;

@Column(nullable = false, unique = true, length = 100) private String email;

@Column(nullable = false, length = 255) private String password;

@Column(length = 15) private String contact;

@Column(columnDefinition = "TEXT") private String address;

@Lob

private byte[] image;

private Time open\_hours; private Time close\_hours;

// Getters and Setters

public int getRestaurantId() { return restaurantId;

}

public void setRestaurantId(int restaurantId) { this.restaurantId = restaurantId;

}

public String getRestaurantName() { return restaurantName;

}

public void setRestaurantName(String restaurantName) { this.restaurantName = restaurantName;

}

public String getUserName() { return userName;

}

public void setUserName(String userName) { this.userName = userName;

}

public String getEmail() { return email;

}

public void setEmail(String email) { this.email = email;

}

public String getPassword() { return password;

}

public void setPassword(String password) { this.password = password;

}

public String getContact() { return contact;

}

public void setContact(String contact) { this.contact = contact;

}

public String getAddress() { return address;

}

public void setAddress(String address) { this.address = address;

}

public byte[] getImage() { return image;

}

public void setImage(byte[] image) { this.image = image;

}

public Time getOpen\_hours() { return open\_hours;

}

public void setOpen\_hours(Time open\_hours) { this.open\_hours = open\_hours;

}

public Time getClose\_hours() { return close\_hours;

}

public void setClose\_hours(Time close\_hours) { this.close\_hours = close\_hours;

}

}

## Restaurant Service.java

package com.example.demo.service;

import com.example.demo.model.Restaurant; import java.util.List;

public interface RestaurantService {

Restaurant saveRestaurant(Restaurant restaurant); Restaurant getRestaurantById(int id);

Restaurant getRestaurantByEmail(String email); List<Restaurant> getAllRestaurants();

void deleteRestaurant(int id); long countRestaurants();

}

## Restaurant Service Impl.java

package com.example.demo.service; import com.example.demo.model.Remark;

import com.example.demo.repository.RemarkRepository;

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

import java.util.List; @Service

public class RemarkServiceImpl implements RemarkService {

@Autowired

private RemarkRepository remarkRepository;

@Override

public Remark saveRemark(Remark remark) { return remarkRepository.save(remark);

}

@Override

public List<Remark> getRemarksByUserId(int userId) { return remarkRepository.findByUser\_UserId(userId);

}

}

## Restaurant Controller.java

**//--- Portal page for restaurant login/register options---**

@GetMapping("/portal")

public String restaurantPortal() { return "restaurantPortal";

}

## // --- Restaurant Registration Form ---

@GetMapping("/register")

public String showRegistrationForm() { return "restaurantRegister";

}

## // --- Process Restaurant Registration ---

@PostMapping("/register")

public String registerRestaurant(@ModelAttribute Restaurant restaurant, Model model) { Restaurant savedRestaurant = restaurantService.saveRestaurant(restaurant); model.addAttribute("savedRestaurant", savedRestaurant);

return "restaurantRegisterSuccess";

}

## // --- Restaurant Login Form ---

@GetMapping("/login")

public String showLoginForm() { return "restaurantLogin";

}

## // Process Restaurant Login

@PostMapping("/login")

public String loginRestaurant(@RequestParam String email,

@RequestParam String password, HttpSession session,

Model model) {

Restaurant restaurant = restaurantService.getRestaurantByEmail(email); if (restaurant != null && restaurant.getPassword().equals(password)) {

session.setAttribute("restaurant", restaurant); return "redirect:/restaurant/dashboard";

} else {

model.addAttribute("error", "Invalid email or password"); return "restaurantLogin"; } }

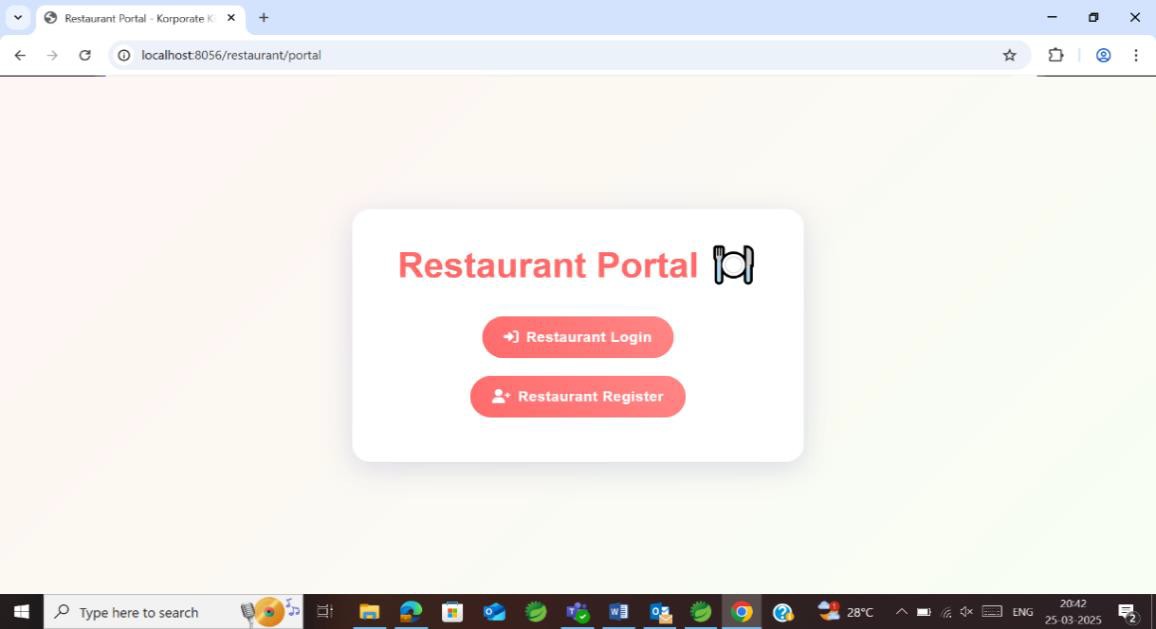


Fig 5.16 Restaurant Portal

Fig 5.16 displays the customer home page featuring a restaurant-friendly interface. It includes login and register buttons for easy access. The design ensures a seamless navigation experience for restaurants.

## // --- Restaurant Dashboard ---

@GetMapping("/dashboard")

public String dashboard(HttpSession session) {

Restaurant restaurant = (Restaurant) session.getAttribute("restaurant"); if (restaurant == null) {

return "redirect:/restaurant/login";

}

return "restaurantDashboard";

}

## // Manage Menu – displays all menu items for the logged-in restaurant

@GetMapping("/manageMenu")

public String manageMenu(HttpSession session, Model model) { Restaurant restaurant = (Restaurant) session.getAttribute("restaurant"); if (restaurant == null) {

return "redirect:/restaurant/login";

}

List<MasterMenuItem> menuItems=masterMenuItemService.getMenuItemsByRestaurantId( restaurant.getRestaurantId());

model.addAttribute("menuItems", menuItems); return "manageMenu";

}

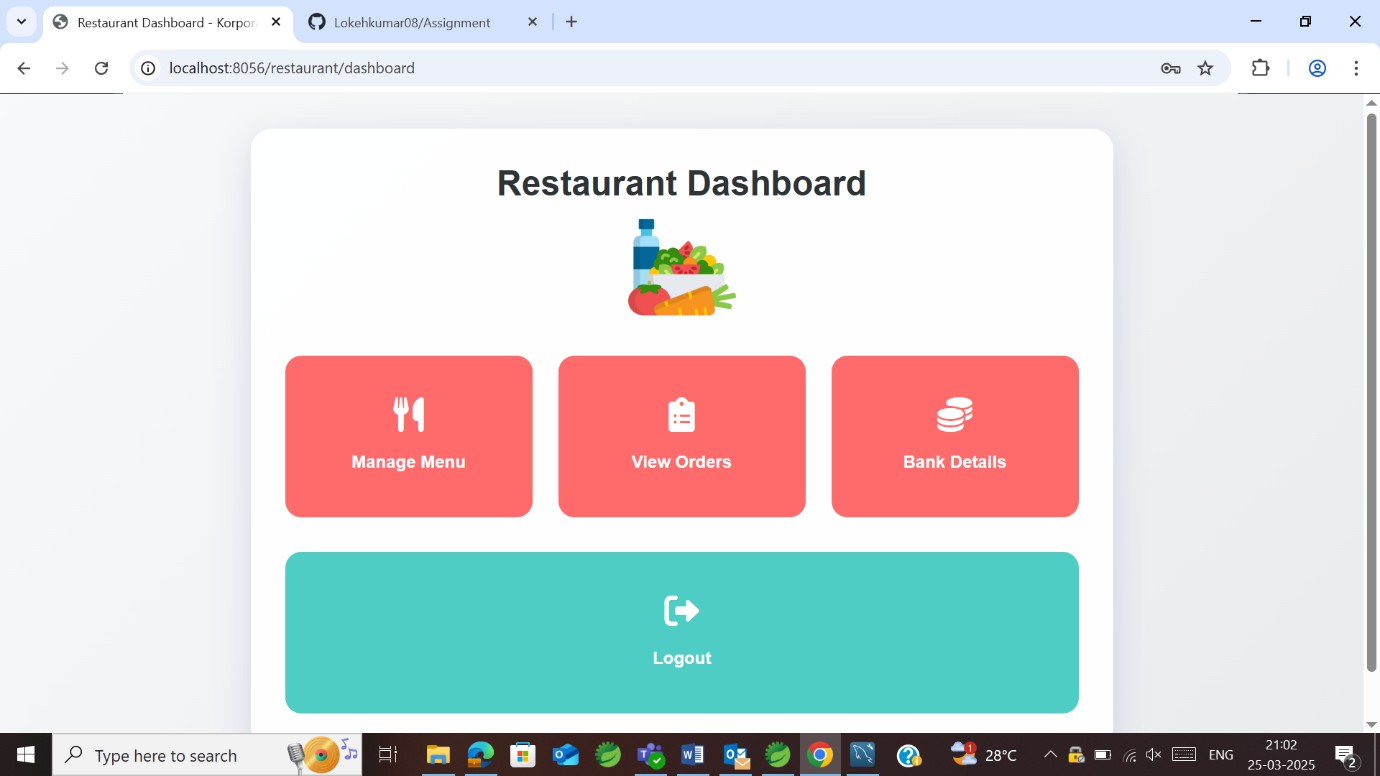


Fig 5.17 Restaurant Dashboard

Fig 5.17 shows the restaurant dashboard which contains options for managing menu, viewing orders and bank details. There is also an option to logout of the website.

## Delivery Agent Portal Delivery Agent.java

package com.example.demo.model; import jakarta.persistence.\*; @Entity

@Table(name = "delivery\_agent") public class DeliveryAgent {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY) private int agentId;

@Column(name = "user\_name", nullable = false, length = 50) private String userName;

@Column(nullable = false, unique = true, length = 100) private String email;

@Column(nullable = false, length = 255) private String password;

@Column(length = 15) private String contact;

@Lob

private byte[] image;

// Getters and Setters public int getAgentId() {

return agentId;

}

public void setAgentId(int agentId) { this.agentId = agentId;

}

public String getUserName() { return userName;

}

public void setUserName(String userName) { this.userName = userName;

}

public String getEmail() { return email;

}

public void setEmail(String email) { this.email = email;

}

public String getPassword() {

return password;

}

public void setPassword(String password) { this.password = password;

}

public String getContact() { return contact;

}

public void setContact(String contact) { this.contact = contact;

}

public byte[] getImage() { return image;

}

public void setImage(byte[] image) { this.image = image;

}

}

## Delivery Agent Service.java

package com.example.demo.service;

import com.example.demo.model.DeliveryAgent; import java.util.List;

public interface DeliveryAgentService {

DeliveryAgent saveDeliveryAgent(DeliveryAgent agent); DeliveryAgent getDeliveryAgentById(int id); DeliveryAgent getDeliveryAgentByEmail(String email); List<DeliveryAgent> getAllDeliveryAgents();

void deleteDeliveryAgent(int id); long countDeliveryAgents();

}

Delivery Agent Service Impl.java package com.example.demo.service;

import com.example.demo.model.DeliveryAgent;

import com.example.demo.repository.DeliveryAgentRepository; import org.springframework.beans.factory.annotation.Autowired; import org.springframework.stereotype.Service;

import java.util.List; @Service

public class DeliveryAgentServiceImpl implements DeliveryAgentService { @Autowired

private DeliveryAgentRepository deliveryAgentRepository;

@Override

public DeliveryAgent saveDeliveryAgent(DeliveryAgent agent) { return deliveryAgentRepository.save(agent);

}

@Override

public DeliveryAgent getDeliveryAgentById(int id) {

return deliveryAgentRepository.findById(id).orElse(null);

}

@Override

public DeliveryAgent getDeliveryAgentByEmail(String email) { return deliveryAgentRepository.findByEmail(email);

}

@Override

public List<DeliveryAgent> getAllDeliveryAgents() { return deliveryAgentRepository.findAll();

}

@Override

public void deleteDeliveryAgent(int id) { deliveryAgentRepository.deleteById(id);

}

@Override

public long countDeliveryAgents() { return deliveryAgentRepository.count();

}

}

## Delivery Agent Controller.java

**//--- Delivery Portal: Offers options to register or log in---**

@GetMapping("/portal") public String deliveryPortal() {

return "deliveryPortal"; // JSP: /WEB-INF/jsp/deliveryPortal.jsp

}

## // --- Registration form for delivery agent ---

@GetMapping("/register")

public String showRegisterForm() {

return "deliveryRegister"; // JSP: /WEB-INF/jsp/deliveryRegister.jsp

}

## // --- Process registration for delivery agent ---

@PostMapping("/register")

public String registerDeliveryAgent(@ModelAttribute DeliveryAgent agent, Model model) { DeliveryAgent savedAgent = deliveryAgentService.saveDeliveryAgent(agent); model.addAttribute("savedAgent", savedAgent);

return "deliveryRegisterSuccess"; // JSP: /WEB-INF/jsp/deliveryRegisterSuccess.jsp

}

## //--- Login form for delivery agent ---

@GetMapping("/login")

public String showLoginForm() {

return "deliveryLogin"; // JSP: /WEB-INF/jsp/deliveryLogin.jsp

}

## // --- Process login for delivery agent ---

@PostMapping("/login")

public String loginDeliveryAgent(@RequestParam String email,

@RequestParam String password, HttpSession session,

Model model) {

DeliveryAgent agent = deliveryAgentService.getDeliveryAgentByEmail(email); if (agent != null && agent.getPassword().equals(password)) {

session.setAttribute("deliveryAgent", agent); return "redirect:/delivery/dashboard";

} else {

model.addAttribute("error", "Invalid email or password"); return "deliveryLogin";}}

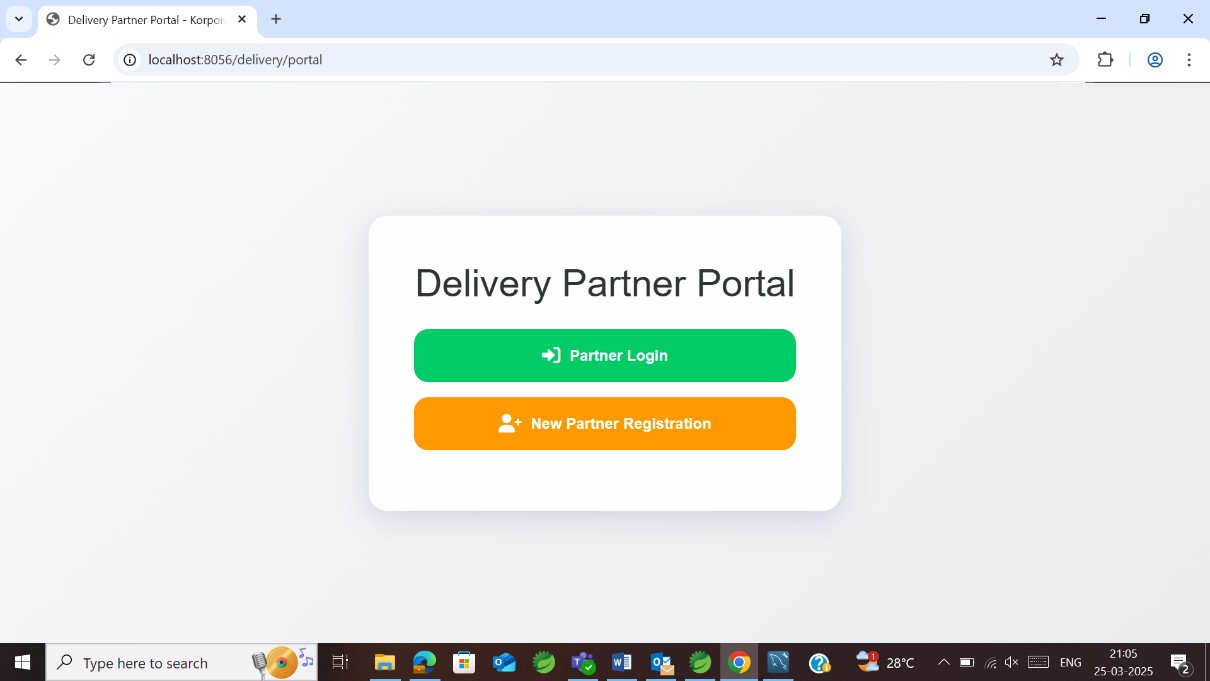


Fig 5.18 Delivery Partner Portal

Fig 5.18 displays the delivery partner home page featuring a restaurant-friendly interface. It includes login and register buttons for easy access. The design ensures a seamless navigation experience for delivery partners.

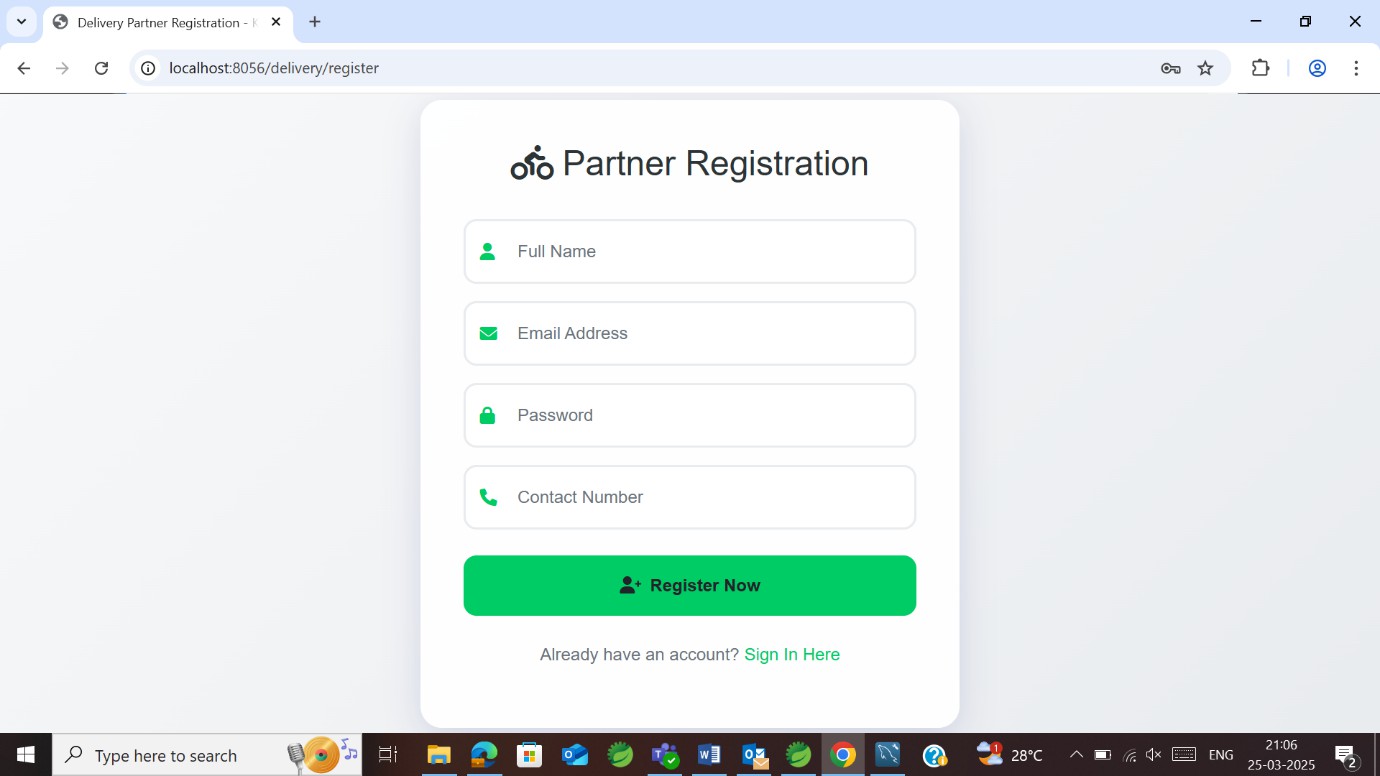


Fig 5.19 Delivery Partner Registration

Fig 5.18 shows the details such as full name, email address, password, contact number for the partner to make a registration

## // --- Delivery dashboard ---

@GetMapping("/dashboard")

**public** String dashboard(HttpSession session) {

DeliveryAgent agent = (DeliveryAgent) session.getAttribute("deliveryAgent");

**if** (agent == **null**) {

**return** "redirect:/delivery/login";

}

**return** "deliveryDashboard";

}

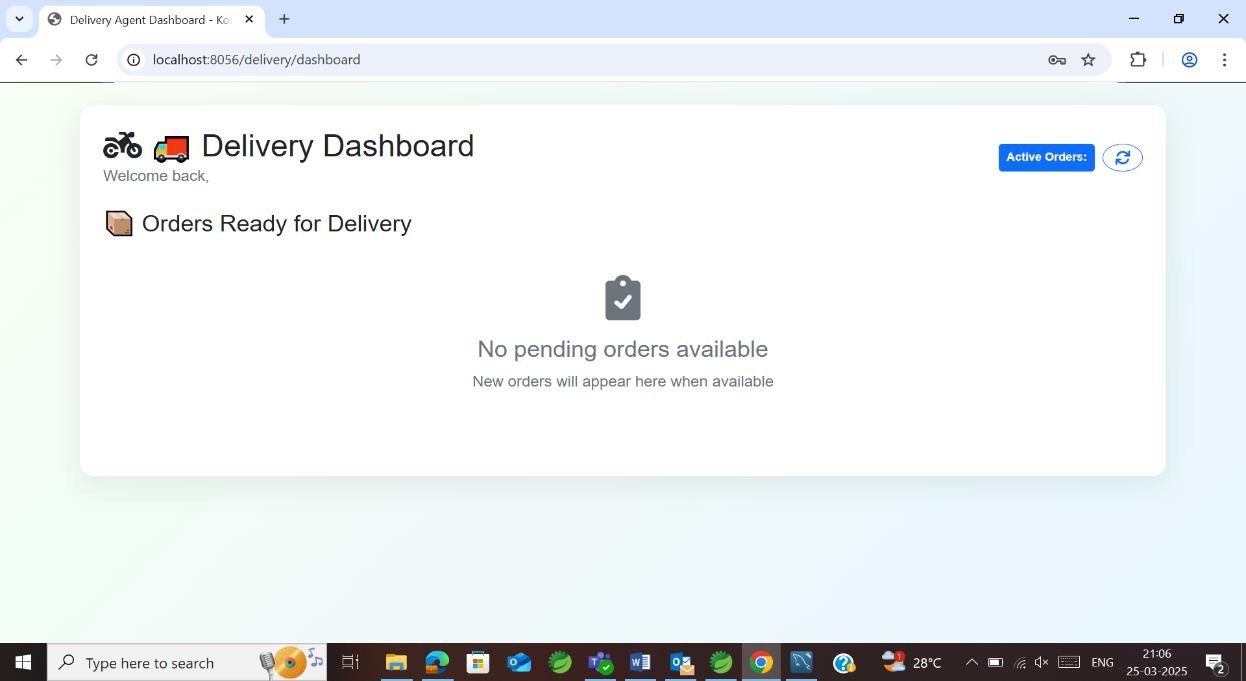


Fig 5.20 Delivery Partner Dashboard

Fig 5.19 shows the partner’s delivery dashboard page which contains pending orders(if any) to be delivered across customers. It also contains a space to see active orders.

# CHAPTER 6

**CONCLUSION AND FUTURE ENHANCEMENTS**

## Conclusion

The Korporate Kitchen project successfully demonstrates the development of a comprehensive online food delivery system using modern web technologies. Through this project, I gained valuable experience in full-stack development, including the use of Spring Boot for backend services, Hibernate for ORM, and JSP with Bootstrap for building dynamic and responsive user interfaces. The modular design, following the MVC architecture, ensures that the application is scalable, maintainable, and easy to enhance. The project not only meets the functional requirements of an online food delivery system but also incorporates robust security, performance, and scalability measures. Overall, this internship project has been a significant learning experience, bridging the gap between academic theory and real-world application. The skills and knowledge I acquired during this project will undoubtedly serve as a strong foundation for my future career in software development.

## Future Scope

In the future, the platform can be enhanced with AI-powered recommendations to provide personalized food suggestions based on user preferences and order history. Real-time order tracking with improved GPS integration will allow customers to monitor their deliveries more accurately. Restaurants can benefit from automated inventory management, which will track stock levels and notify them when supplies are running low. The integration of multiple payment options, including digital wallets, UPI, and even cryptocurrency, will make transactions more seamless. Additionally, loyalty and subscription plans can be introduced to encourage customer retention by offering exclusive discounts and rewards for frequent users.

# REFERENCES

1. [D. D. Singh, A. Kumar, and A. Chakraborty, “Towards Fairness in Online Service with](http://paperpile.com/b/cSXCNW/B5u8) [k Servers and its Application on Fair Food Delivery,” Dec. 18, 2023. Accessed: Apr. 02, 2025.](http://paperpile.com/b/cSXCNW/B5u8) [[Online]. Available:](http://paperpile.com/b/cSXCNW/B5u8) <http://arxiv.org/abs/2312.11280>
2. [Y. Liu, Y. Shang, and S. Li, “Joint Infrastructure Planning and Order Assignment for](http://paperpile.com/b/cSXCNW/PPCP) [On-Demand Food-Delivery Services with Coordinated Drones and Human Couriers,” Jan. 24,](http://paperpile.com/b/cSXCNW/PPCP) [2025. Accessed: Apr. 02, 2025. [Online]. Available:](http://paperpile.com/b/cSXCNW/PPCP) <http://arxiv.org/abs/2501.14325>
3. [M. Joshi, A. Singh, S. Ranu, A. Bagchi, P. Karia, and P. Kala, “Batching and Matching](http://paperpile.com/b/cSXCNW/2ZJh) [for Food Delivery in Dynamic Road Networks,” Aug. 29, 2020. Accessed: Apr. 02, 2025.](http://paperpile.com/b/cSXCNW/2ZJh) [[Online]. Available:](http://paperpile.com/b/cSXCNW/2ZJh) <http://arxiv.org/abs/2008.12905>
4. [S. Švancár, L. Chrpa, F. Dvořák, and T. Balyo, “Cloud Kitchen: Using Planning-based](http://paperpile.com/b/cSXCNW/sKCz) [Composite AI to Optimize Food Delivery Processes,” Feb. 16, 2024. Accessed: Apr. 02, 2025.](http://paperpile.com/b/cSXCNW/sKCz) [[Online]. Available:](http://paperpile.com/b/cSXCNW/sKCz) <http://arxiv.org/abs/2402.10725>
5. [A. Mehra, S. Saha, V. Raychoudhury, and A. Mathur, “DeliverAI: Reinforcement](http://paperpile.com/b/cSXCNW/kvuh) [Learning Based Distributed Path-Sharing Network for Food Deliveries,” Nov. 03, 2023.](http://paperpile.com/b/cSXCNW/kvuh)

[Accessed: Apr. 02, 2025. [Online]. Available:](http://paperpile.com/b/cSXCNW/kvuh) <http://arxiv.org/abs/2311.02017>

1. [Philomina, “CUSTOMERS PERCEPTION ON ONLINE FOOD DELIVERY: A CASE](http://paperpile.com/b/jRSMUt/AQXm) [STUDY OF KARNATAKA STATE,” *MJACS*, vol. 1, no. 1, pp. 9–16, Dec. 2023.](http://paperpile.com/b/jRSMUt/AQXm)
2. [X. Liu, X.-J. Lim, J.-H. Cheah, S. I. Ng, and N. K. Basha, “Food at your doorstep?](http://paperpile.com/b/jRSMUt/XOv2) [Examining customer loyalty towards online food delivery applications,” *British Food Journal*,](http://paperpile.com/b/jRSMUt/XOv2) [vol. ahead-of-print, no. ahead-of-print, Jul. 2023, doi:](http://paperpile.com/b/jRSMUt/XOv2) [10.1108/BFJ-02-2023-0116.](http://dx.doi.org/10.1108/BFJ-02-2023-0116)

# BIBILIOGRAPHY

* + Spring Boot Reference Documentation: <https://spring.io/projects/spring-boot>
  + Hibernate ORM Documentation: <https://hibernate.org/orm/documentation/>
  + MySQL Documentation: <https://dev.mysql.com/doc/>
  + JSP and JSTL Guides: Various online tutorials and official documentation.
  + Bootstrap Documentation: <https://getbootstrap.com/>
  + Additional Resources:
    - Spring Data JPA Documentation: <https://spring.io/projects/spring-data-jpa>
    - Maven Documentation: <https://maven.apache.org/guides/index.html>