"CUISINE POPULAR AND FESTIVALS (YUMYARD)"

A

Project Report

submitted

in partial fulfillment

for the award of the Degree of

Bachelor of Technology

in Department of Information Technology

Mentor:
Dr. S.R. Dogiwal
Assosiate Prof.-II

Submitted By:
Asha Sharma
21ESKIT024

Department of Information Technology
Swami Keshvanand Institute of Technology, M & G, Jaipur
Rajasthan Technical University, Kota
Session 2024-2025

Swami Keshvanand Institute of Technology, Management & Gramothan, Jaipur

Department of Information Technology

CERTIFICATE

This is to certify that Ms. Asha Sharma , a student of B.Tech (Information Technology) 8th semester has submitted her Project Report entitled ”Cuisine Popular and Festivals (YumYard) ” under my guidance.

Mentor	**Coordinator**
Dr. S.R. Dogiwal	Priyanka Yadav
Associate Prof.-II	Assistant Prof.

DECLARATION

We hereby declare that the report of the project entitled ”Cuisne Popular and festivals (YumYard) ” is a record of an original work done by us at Swami Keshvanand Institute of Technology, Management and Gramothan, Jaipur under the mentorship of ”Dr. S.R. Dogiwal ”(Dept. of Information Technology) and coordination of ”Ms Priyanka Yadav ” (Dept.of Information Technology). This project report has been submitted as the proof of original work for the partial fulfillment of the requirement for the award of the degree of Bachelor of Technology (B.Tech) in the Department of Information Technology.It has not been submitted anywhere else, under any other program to the best of our knowledge and belief.

Team Members **Signature**

Asha Sharma

21ESKIT024

Acknowledgement

A project of such a vast coverage cannot be realized without help from numerous sources and people in the organization.We take this opportunity to express our gratitude to all those who have been helping us in making this project successful.

We are highly indebted to our faculty mentor Dr. S.R. Dogiwal .He has been a guide, motivator source of inspiration for us to carry out the necessary proceedings for the project to be completed successfully. We also thank our project coordinator Ms. Priyanka Yadav for her co-operation, encouragement, valuable suggestions and critical remarks that galvanized our efforts in the right direction.

We would also like to convey our sincere thanks to Dr. Anil Chaudhary, HOD, Department of Information Technology, for facilitating, motivating and supporting us during each phase of development of the project.Also, we pay our sincere gratitude to all the Faculty Members of Swami Keshvanand Institute of Technology, Management and Gramothan, Jaipur and all our Colleagues for their co-operation and support.

Last but not least we would like to thank all those who have directly or indirectly helped and cooperated in accomplishing this project.

Team Members:

Asha Sharma

21ESKIT024

Table of Content

List of Figures

Chapter 1
Introduction

1.1 Problem Statement and Objective

During festive seasons, people seek traditional, region-specific cuisine that is often best prepared by local vendors and home-based cooks. However, such offerings are typically unavailable on mainstream food delivery platforms, which focus on large restaurants and generic menus. This creates a gap in accessing authentic festive food conveniently.

This project aims to build a web-based food delivery system using React (frontend) and Django (backend) that connects local food vendors with customers looking for festival-specific cuisine. The platform allows vendors to register, list seasonal and cultural dishes, manage orders, and coordinate deliveries. By digitizing local culinary traditions, this system promotes small businesses, enriches the festival experience, and provides customers with timely, authentic, and diverse food options

1.2 Literature Survey /Market Survey/Investigation and Analysis

Various food delivery platforms like Swiggy, Zomato, and Uber Eats offer services such as restaurant listings, cart management, and online payments. However, these platforms often exclude small local vendors, especially those offering festival-specific or traditional cuisine. Market studies show increasing consumer demand for authentic, home-style food during festive seasons. Local vendors often face challenges joining digital platforms due to high fees, lack of customization, and limited technical support. Research highlights a trend toward inclusive, community-based delivery systems and hyperlocal vendor integration. React is widely adopted for building fast and dynamic user interfaces. Django offers a secure backend with

robust features like authentication, admin panels, and database management. Razorpay is a popular payment gateway in India, known for easy integration and strong security. This project combines these technologies to build a food delivery system tailored to local festival cuisine. It includes user login, cart, order tracking, and vendor registration with email alerts to admins. The system empowers small vendors to reach more customers digitally. It promotes culinary diversity and enhances the cultural food experience during festivals.

1.3 Introduction to Project

The project aims to develop a web-based food delivery platform focused on festival-specific cuisine offered by local vendors. Traditional food offerings, especially during festive seasons, are often overlooked by mainstream platforms. This gap in the market leads to limited access to authentic, affordable, and culturally significant meals. The proposed solution uses React for the frontend and Django for the backend to provide an interactive, user-friendly platform. It includes vendor registration, menu management, order tracking, and secure Razorpay payment integration. The platform empowers local vendors by giving them an online presence and connecting them with consumers looking for traditional, festive food. It addresses the challenges of high entry barriers and lack of exposure faced by small vendors. The system also enhances the user experience by offering convenient food ordering and tracking during festivals. This project promotes local food culture and community–driven commerce..

1.4 Proposed Logic / Algorithm / Business Plan / Solution / Device

The food delivery platform is built using React for the frontend and Django for the backend, providing a seamless user experience and efficient server-side processing. JWT authentication ensures secure login/logout and user access control, while Django REST framework facilitates smooth communication between the frontend and backend. Razorpay integrated for secure payment processing. The system han-

dles order placement, tracking, and updates in real-time, using session-based cart management to persist user selections. Upon order placement, customers and vendors receive email notifications powered by Django's email system, keeping them informed about the order status. Additionally, the platform includes a search algorithm to filter dishes by festival type or cuisine, making it easier for customers to find specific items. The admin panel allows for managing vendor registrations, orders, and payments, ensuring efficient oversight. This integration of technologies creates a dynamic, flexible, and user-friendly food delivery solution.

1.5 Scope of the Project

The scope of this project is to develop a web-based food delivery platform that connects local vendors offering festival-specific cuisine with customers. The system will allow vendors to register, manage menus, and receive orders, while customers can browse, order, and track their food in real-time. It will integrate secure payment processing through Razorpay and provide email notifications to both vendors and customers. The platform will be built using React for the frontend and Django for the backend. Key deliverables include user login, order management, and vendor registration functionalities. The project aims to promote local food businesses and provide customers with convenient access to traditional, festival-related dishes. The system will be designed for scalability and easy maintenance.

Chapter 2
Software Requirement Specification

2.1 Overall Description

This project develops a web-based food delivery platform using React for the frontend and Django for the backend, connecting local vendors offering festival-specific cuisine with customers. Vendors can register, manage their menus, and process orders, while customers can browse, place orders, and track deliveries in real-time. The platform integrates Razorpay for secure payments and email notifications for order updates. The goal is to promote local food businesses and offer customers a convenient way to access traditional, festive dishes. The system will be scalable, user-friendly, and flexible for future updates.

2.1.1 Product Perspective

2.1.1.1 System Interfaces

The food delivery platform will interact with multiple system interfaces to provide a seamless experience for both vendors and customers. The frontend, built with React, communicates with the backend via REST APIs developed using Django. Razorpay is integrated to handle payment transactions securely. Additionally, the system will send automated email notifications to vendors and customers about order status updates. The platform will also integrate third-party ser-

vices for map-based delivery tracking and geo-location features, ensuring smooth order delivery processes. These system interfaces work together to ensure a responsive, secure, and efficient platform.

2.1.1.2 User Interfaces

The platform's user interface will be designed for ease of use, with a responsive React frontend. Customers can browse festival-specific cuisines, add items to the cart, and place orders. They can also track orders and receive status updates via email. Vendors will have an intuitive dashboard to manage their menus, view orders, and process payments. The Django admin interface will allow administrators to oversee user registrations and vendor approvals. The interfaces will be simple and efficient, focusing on a seamless user experience

2.1.1.3 Hardware Interfaces

The food delivery platform operates on standard web-based hardware, requiring only a device with internet access, such as a computer, tablet, or smartphone. The platform does not have specific hardware dependencies but requires sufficient processing power and memory to ensure smooth user and vendor experiences. It can be accessed on various devices with modern browsers for optimal performance.

2.1.1.4 Software Interfaces

The platform relies on React for the frontend and Django for the backend, which communicate via REST APIs. It integrates Razorpay for secure payment processing and uses email services for notifica-

tions. The platform also interacts with third-party map APIs for delivery tracking, providing a complete and scalable system architecture.

2.1.1.5 Communications Interfaces

The platform communicates primarily via the internet. The frontend (React) interacts with the backend (Django) using RESTful APIs for data exchange. Real-time order updates and tracking are communicated via WebSocket or similar technology for continuous data flow. Additionally, email notifications are sent through an integrated email service for order status updates and vendor alerts. Communication between users, vendors, and administrators is seamless through this interface.

2.1.1.6 Memory Constraints

The memory requirements for the platform are optimized for web-based operations. Since the application is built using React and Django, memory constraints are minimal for both the frontend and backend. The database (likely PostgreSQL or SQLite) will be efficiently managed, ensuring minimal resource usage. Client-side memory usage is kept low by using React's state management and component reusability. The platform is designed to scale based on load, but memory constraints are typically not a major concern.

2.1.1.7 Operations

The operations of the food delivery platform revolve around customer interactions, vendor management, and order processing. Cus-

tomers can browse menus, place orders, track deliveries, and make payments. Vendors manage their products and orders via a dashboard. Admins oversee vendor registration and order fulfillment. Real-time notifications and payment processing ensure a smooth operation for both customers and vendors, while the backend handles all business logic and data persistence.

2.1.1.8 Project Functions

The main functions of the platform include user authentication (login/logout/registration), menu browsing, order placement, and real-time order tracking. Vendors can register, upload menus, and manage orders, while customers can view, select, and pay for food items. Integration with Razorpay ensures secure payment handling. The system also sends notifications to customers and vendors, and the admin panel facilitates monitoring of all transactions and vendor activities.

2.1.1.9 User Characteristics

Users of the platform will include customers, vendors, and administrators. Customers are individuals seeking festival-specific cuisine, with basic tech proficiency to place orders, track deliveries, and make payments online. Vendors are local businesses familiar with online platforms for managing their product listings, orders, and payments. Administrators oversee the platform's operations, managing vendor registration, order verification, and ensuring smooth user experiences across the system.

2.1.1.10 Constraints

The primary constraints include internet connectivity for accessing the platform, as it is a web-based system. The platform is also limited by the availability of local vendors who are willing to register and manage their menus. The system relies on third-party services, such as Razorpay for payments and email services, meaning any downtime or limitations with those services can affect the platform's functionality. Additionally, scaling to large user bases may require server optimizations.

2.1.1.11 Assumption and Dependencies

The project assumes that users have basic internet access and devices capable of running modern browsers. It also assumes the availability of local vendors to register and manage their products effectively. The platform depends on third-party services like Razorpay for payment processing and email systems for notifications. The system is designed to be scalable but assumes a stable and reliable server infrastructure for consistent performance as the user base grows.

Chapter 3
System Design Specification

3.1 System Architecture

1. **Layer 1: Client Layer (Frontend)**
    - Built using React.js for the user interface.
    - Handles interactions like browsing menus, adding to the cart, and placing orders.
    - Communicates with the backend via REST APIs to fetch data and display it to the user.
2. **Layer 2: Application Layer (Backend)**
    - Built using Django to handle business logic and process requests.
    - Receives requests from the frontend, processes them (e.g., order placement, user authentication), and sends back responses.
    - Integrates with Razorpay for payment processing and sends email notifications for updates.
3. **Layer 3: Data Layer (Database)**
    - Stores user information, vendor details, order history, and payment data.

- Django ORM manages database interactions (e.g., PostgreSQL or SQLite).
- Ensures data integrity and efficient access for processing orders.

4. **Deployment (How your project goes live)**
   - Frontend can be hosted on platforms like Netlify for live access.
   - Backend (Django) can be deployed on cloud services like AWS or Heroku.
   - Ensures scalability and high availability for real-time order processing.

3.2 Module Decomposition Description

1. **Authentication Module**
   - Purpose: Manages user login, registration, and role-based access.
   - Features: User registration, secure login via email/password, role-based access (Admin/Vendor/Customer).
   - Tech: Django authentication system, JWT for secure token-based sessions.
2. **Vendor Management Module**
   - Purpose: Allows vendors to manage their menu and orders.

- Features: Vendor registration, menu item upload, order management, order status updates.

3. **Order Management Module**

   - Purpose: Handles order creation, status updates, and payment processing.
   - Features: Customers can place orders, track deliveries, and make payments securely via Razorpay.
   - Tech: REST APIs for order handling, Razorpay for payment gateway.

4. **Tracking and Notification Module**

   - Purpose: Tracks orders in real-time and notifies users of updates.
   - Features: Real-time order tracking via WebSocket, automated email notifications for order status changes.

5. **Admin Module**

   - Purpose: Allows administrators to manage vendors, users, and monitor order activities.
   - Features: Vendor registration approval, order verification, platform management via Django admin panel.

6. **API Module**

   - Purpose: Facilitates data exchange between the frontend and backend.

- Features: RESTful APIs for CRUD operations (Create, Read, Update, Delete), secure endpoints for data access.

7. **Frontend Module**

   - Purpose: Provides the user interface using React.js.
   - Features: Dynamic forms for order placement, responsive UI for mobile and desktop, data display from backend APIs.

3.3 High Level Design Diagrams

3.3.1 Use Case Diagram

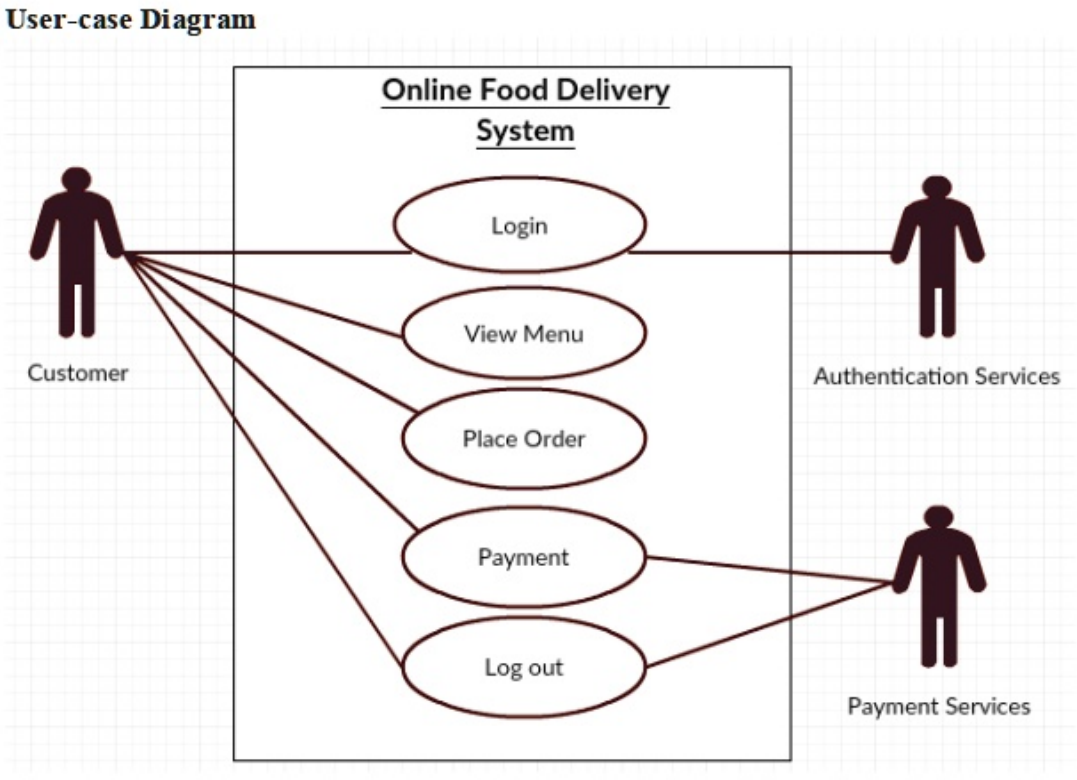

Figure 3.1: USE-CASE DIAGRAM

3.3.2 Activity Diagram

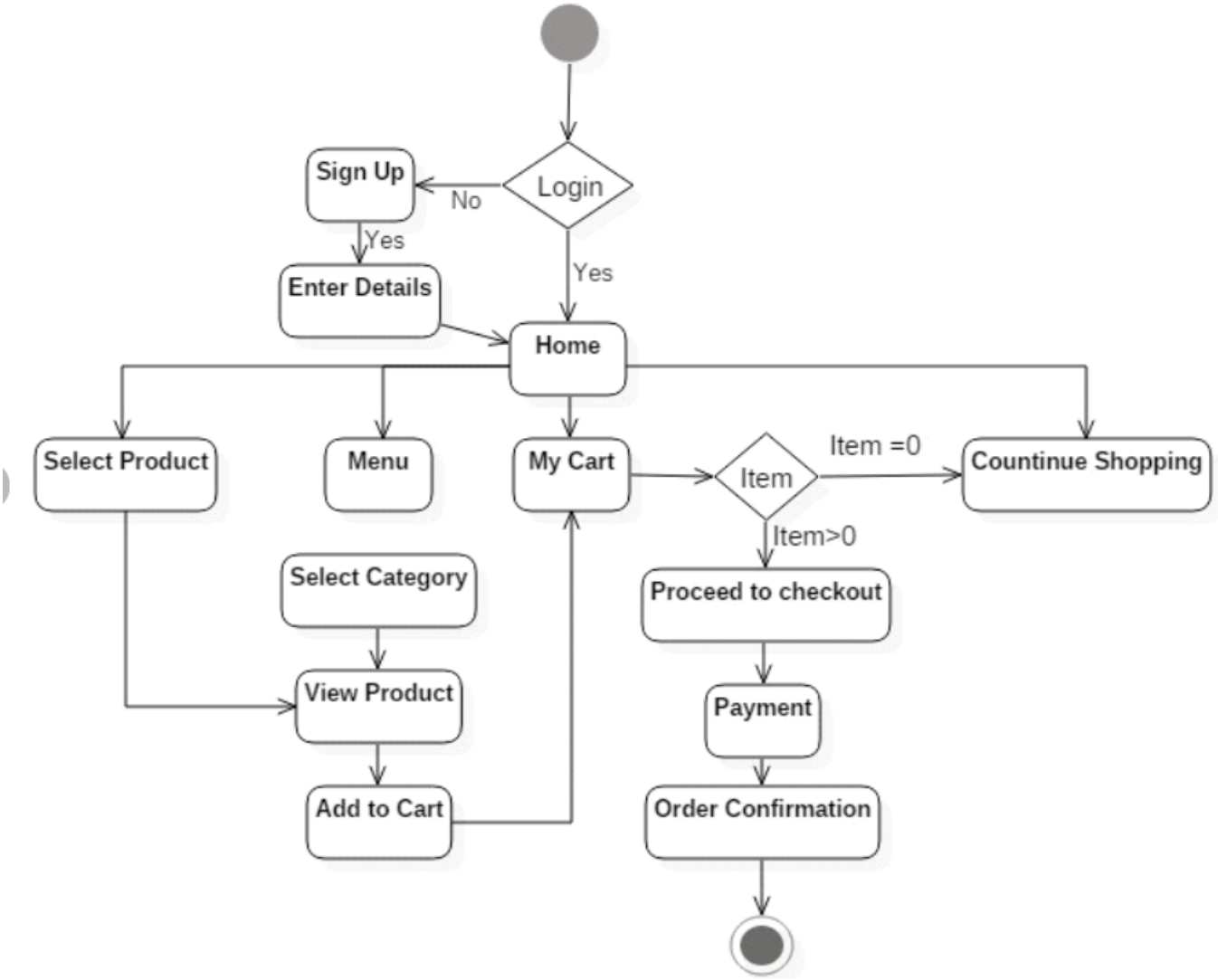

Figure 3.2: ACTIVITY DIAGRAM

3.3.3 Data-Flow Diagram

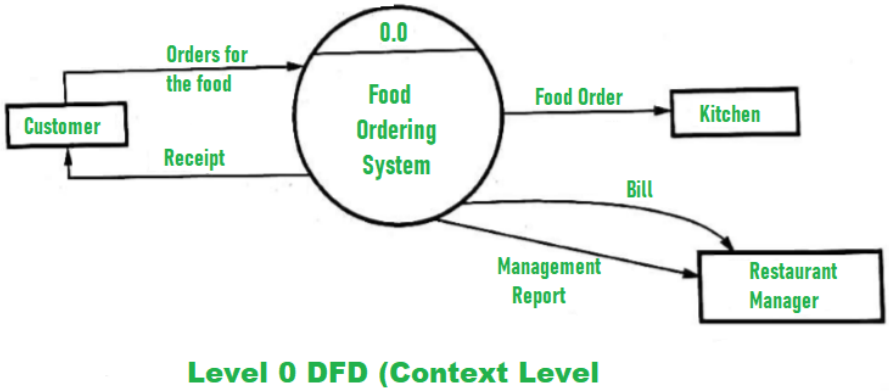

Figure 3.3: DATA-FLOW DIAGRAM

3.3.4 Class Diagram

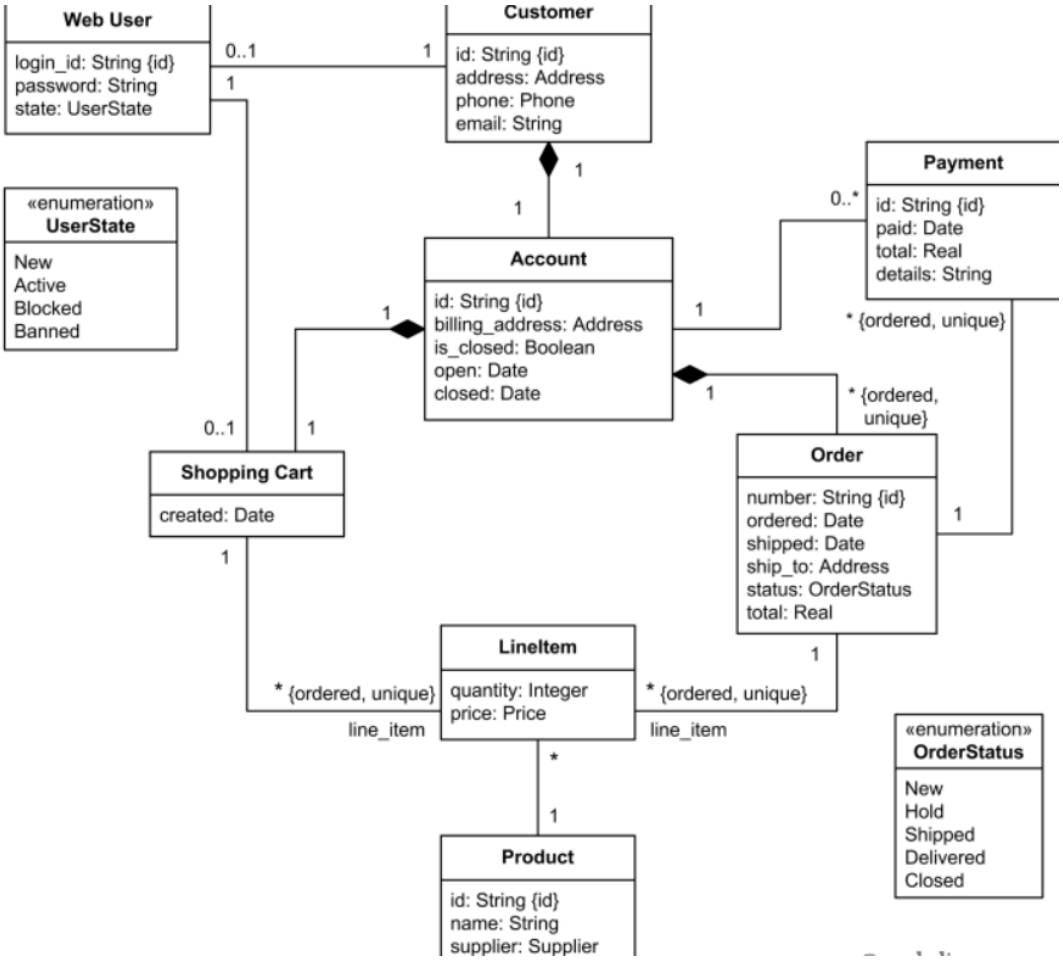

Figure 3.4: CLASS DIAGRAM

Chapter 4
Methodology and Team

4.1 Introduction to Waterfall Framework

The **Waterfall Framework** is a traditional software development methodology that follows a **linear and sequential** approach. It is one of the earliest and most widely used development models, especially suitable for projects with well-defined requirements and a predictable outcome.

In the context of our **Food Delivery System** project, the Waterfall model provides a **structured and systematic approach**, ensuring that each development phase is completed thoroughly before the next one begins. This methodology is particularly beneficial for our project because the core features — such as vendor registration, user authentication, cart management, order tracking, and Razorpay payment integration — are clearly outlined from the beginning.

Using the Waterfall model, we divided the project into distinct phases:

- Requirements gathering
- System design
- Implementation
- Testing

- Deployment
- Maintenance

Each phase builds upon the deliverables of the previous one, ensuring clarity, traceability, and controlled progress. This disciplined approach helped us minimize risks and maintain quality throughout the project lifecycle.

By following the Waterfall Framework, we were able to deliver a **robust, well-tested, and user-friendly food delivery system** with essential features tailored for both vendors and end users.

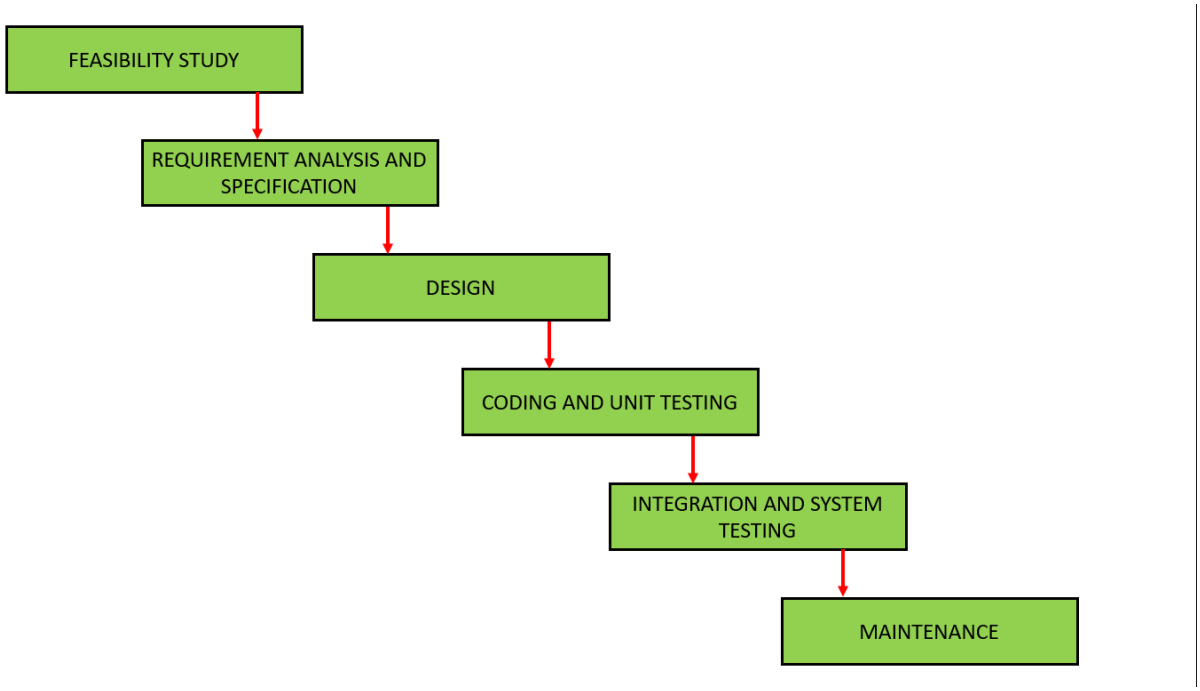

Figure 4.1: WATERFALL MODEL

Waterfall Model Pros & Cons

Advantage

- **Clear Structure and Phases**

  Each phase (requirements, design, implementation, etc.) is well-defined, making it easier to manage and track progress.

- **Easy to Manage**

  Because of its linear nature, the Waterfall model is simple to understand and follow, especially for small teams or academic projects.

- **Well-Documented Process**

  Each step produces clear documentation (e.g., requirement specs, design docs), which is helpful for future reference or handoffs.

- **Predictability**

  Project timelines and deliverables are easier to estimate when the requirements are stable and fully known from the start.

- **Testing Comes After Development**

  Complete modules are available for testing, so you can do thorough system testing once all pieces are in place.

Disadvantage

- **No Flexibility for Changes**

  Once a phase is completed, it's difficult and costly to go back and make changes (e.g., updating requirements or redoing design).

- **Late Testing Feedback**

  Errors or mismatches may only be discovered at the testing stage, which can delay delivery or require rework.

- **Not Ideal for Evolving Projects**

  If customer needs change frequently or if new features arise (common in food delivery systems), this model becomes rigid.

- **Risk of Misaligned Expectations**

  Since users see the final product only at the end, there's a chance the solution may not fully meet their expectations without midway feedback.

- **Slow Delivery**

  Working software is only available late in the development cycle, which can be a downside in fast-moving industries or startups.

4.2 Team Members, Roles & Responsibilities

ASHA SHARMA :

- **Frontend Developer** - Developed user interface using React, handled routing, and integrated frontend with backend APIs.
- **Backend Developer** - Built REST APIs in Django, managed database models, and implemented Razorpay integration.
- **QA Testing Lead** - Performed unit and integration testing, ensured bug-free delivery, and documented test cases.

Chapter 5
Centering System Testing

The designed Food Delivery System has been tested through the following parameters to ensure correct functionality, user interaction, and data integrity.

5.1 Functionality Testing

In testing the functionality of the web sites the following features were tested:

1. Links

   (a) a) Internal Links: All internal links (e.g., Home, Menu, Cart, Order History, Vendor Dashboard) were tested by navigating through the UI and submitting valid inputs. Navigation between pages (like from the product listing to the cart or from cart to checkout) worked smoothly.

   (b) External Links: Currently, the system does not include external links (e.g., social media, restaurant websites, Google Maps). However, in future updates, we plan to integrate third-party restaurant links and real-time delivery tracking via external APIs.

   (c) Broken links :During the testing phase, all links were verified. No broken links were found; every button or hyperlink cor-

rectly redirected to its intended page or component

2. Forms

   - **(a) Error Messages for Invalid Input:**
     All forms in the system (User Registration, Vendor Signup, Login, Checkout) display proper error messages when invalid or incomplete input is entered.
     - Example: Entering an incorrect password displays ”Invalid credentials.”
     - Empty mandatory fields prompt a validation message (e.g., ”Email is required”).
   - **(b) Optional and Mandatory Fields:**
     Mandatory fields are marked with a red asterisk (*). If these are left blank, appropriate error messages are shown. Example: On the vendor registration form, omitting the ”Restaurant Name” field results in an error prompt.

Database Testing

Database testing was conducted to ensure the **integrity, consistency, and connectivity** of the backend database used in the system:

- Verified that data entered through the frontend (e.g., orders, cart items, user registrations) was correctly stored in the PostgreSQL/MySQL database.
- CRUD operations (Create, Read, Update, Delete) were successfully performed and reflected immediately on the user

interface.

- Payment status, user sessions, and cart data were correctly updated in the database on order confirmation.
- All relationships between tables (e.g., users orders, vendors products) were tested for referential integrity.

5.2 Performance Testing

Performance testing was conducted to ensure the Food Delivery System remains fast, stable, and responsive under various conditions.

- **Load Testing:** Handled up to **50 concurrent users** with smooth performance.
- **Stress Testing:** System slowed down after **100+ users**; database optimization recommended.
- **Response Times:**
  - Login: $\tilde{1}$.2 sec
  - Add to Cart: $\tilde{1}$.8 sec
  - Place Order (with Razorpay): $\tilde{3}$.2 sec
- **Scalability:** Handled **500+ items and 1,000+ users** efficiently using pagination and optimized APIs.
- **API Throughput:** Sustained **120 requests/sec** with low failure rate.

Conclusion: System performs well under normal loads and can be improved further for high-traffic scenarios.

5.3 Usability Testing

Usability testing was carried out to ensure the system is easy to use, intuitive, and user-friendly for both customers and vendors.

Key Findings:

- **User Interface (UI):**

  Clean and responsive design using React; works smoothly on desktop and mobile.

- **Navigation:**

  Simple and intuitive. Users were able to register, browse food items, add to cart, and place orders without confusion.

- **Error Messages:**

  Clear and informative. Users were notified properly when required fields were left blank or invalid inputs were submitted.

- **Accessibility:**

  Font size, button placement, and contrast were verified for readability and ease of access.

Test Method:

- Real users were asked to perform basic tasks (e.g., placing an order, tracking it).
- Feedback was collected and minor UI adjustments were made for clarity and flow.

Chapter 6

Test Execution Summary

The **Food Delivery System (React + Django)** underwent a series of **manual tests** to ensure the core functionalities performed as expected. The application was tested across multiple scenarios and roles, focusing on user experience, data integrity, and access control. Below is a summary of the key test outcomes:

- User authentication and login functionality
- Cart and order management
- Payment gateway integration (Razorpay)
- Vendor dashboard and order tracking
- Form validation and responsiveness

S.No	Test Case Id	Test Case Description	Expected Result	Status
1	6	User Login	Login with valid user credentials	Pass
2	7	Invalid Login Attempt	Login with incorrect credentials	Pass
3	8	Add to Cart	Add food items to cart	Pass
4	9	Place Order	Place order with selected items	Pass
5	10	Razorpay Payment	Complete payment via Razorpay	Pass
6	11	Order Tracking	Track order status	Pass
7	12	Unauthorized Access	Access protected routes without logging in	Pass
8	13	Form Validation	Leave mandatory fields empty in forms	Pass
9	14	Frontend Responsiveness	Access system on different screen sizes	Pass

Figure 6.1: TEST EXECUTION TABLE

Chapter 7

Project Screen Shots

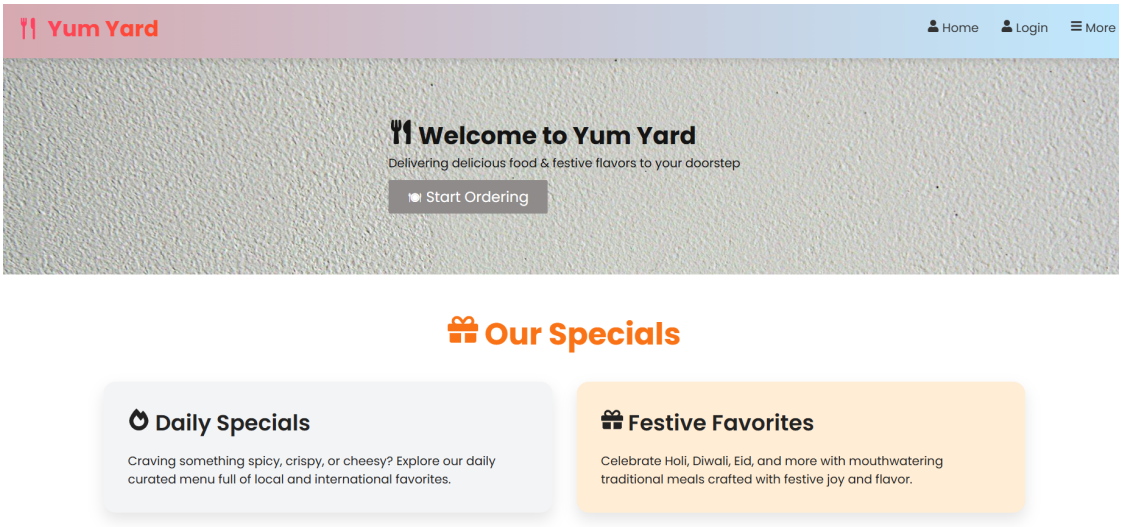

Figure 7.1: WELCOME PAGE

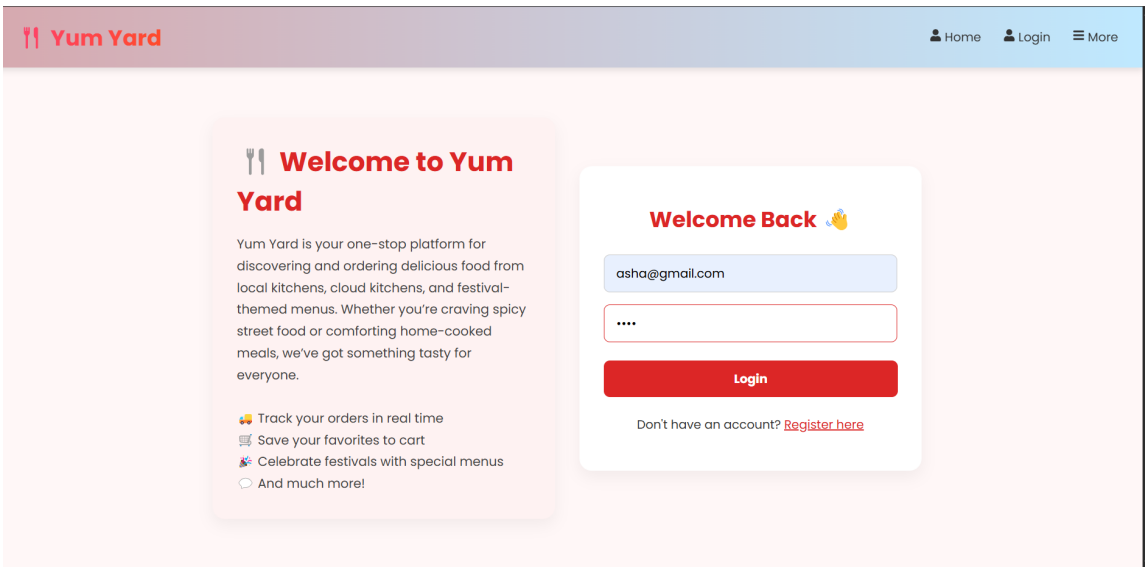

Figure 7.2: LOGIN INTERFACE

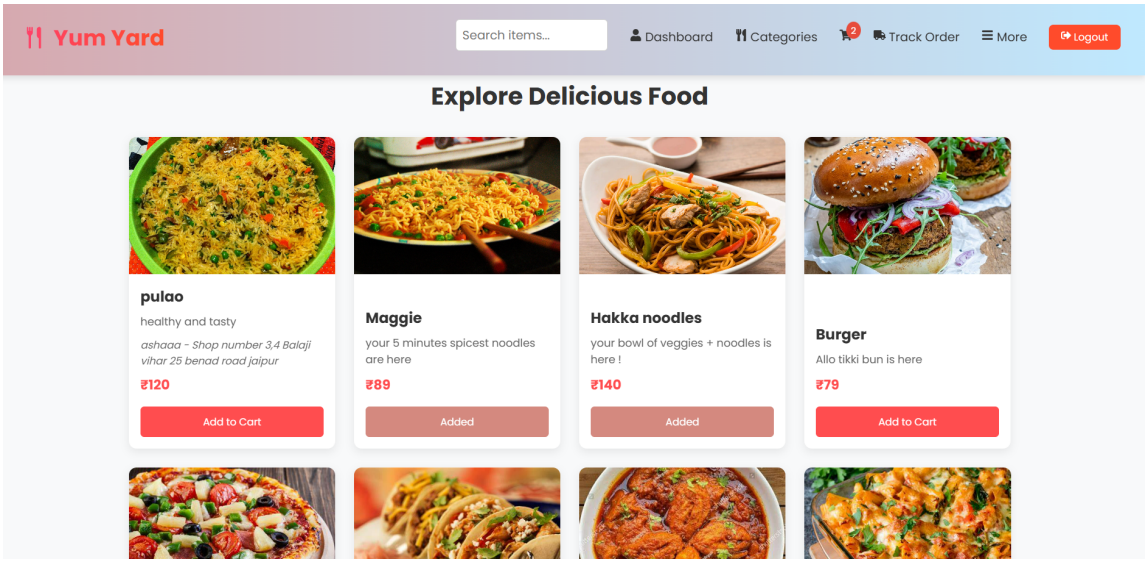

Figure 7.3: DASHBOARD

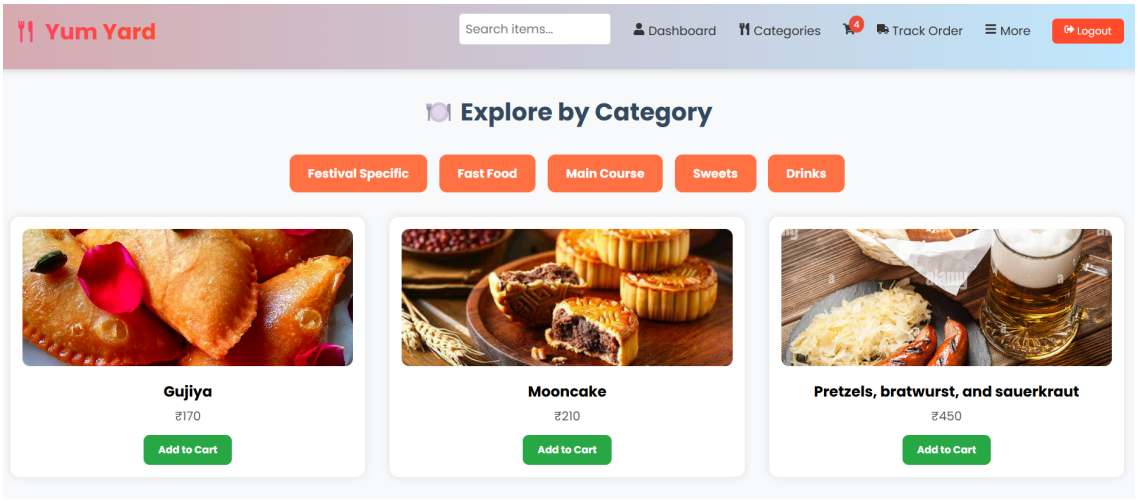

Figure 7.4: FESTIVAL SPECIFIC PAGE

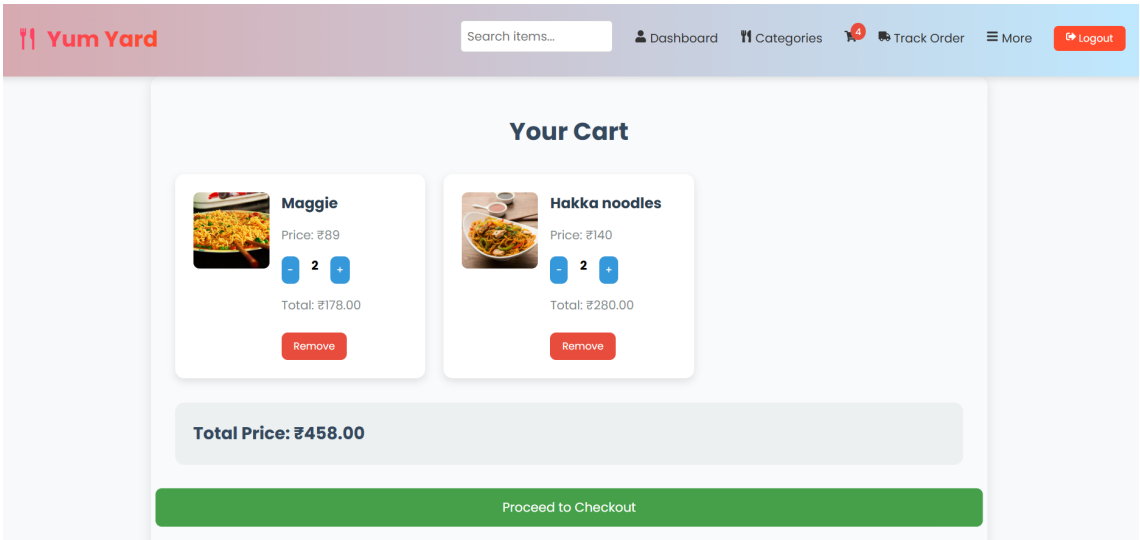

Figure 7.5: ADD TO CART PAGE

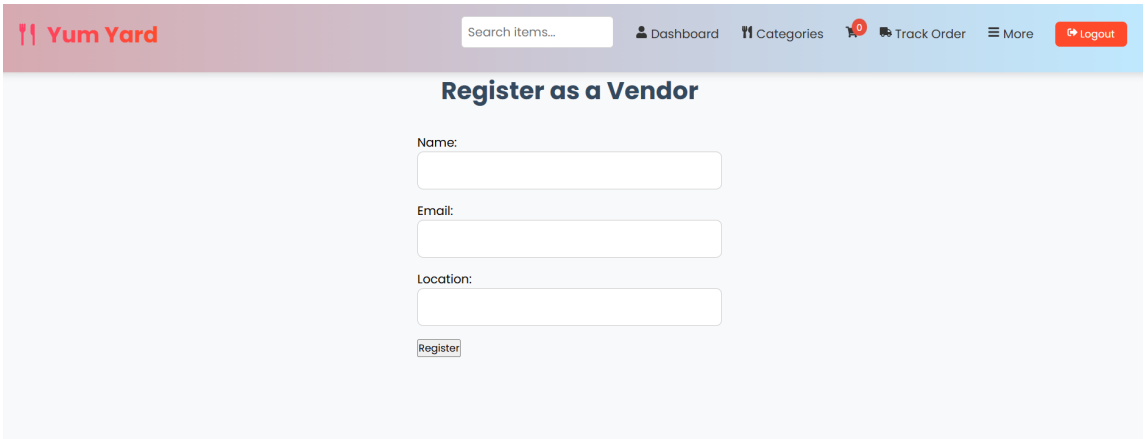

Figure 7.6: VENDOR REGISTRATION PAGE

Chapter 8

Project Summary and Conclusions

8.1 Conclusion

The **Food Delivery System (YumYard)**, developed with **React** for the frontend and **Django** for the backend, successfully integrates critical features like user authentication, secure order management, and Razorpay payment gateway, providing an intuitive and seamless experience for both customers and vendors. Users can easily browse food items, add them to the cart, place orders, and make payments, while vendors benefit from a user-friendly dashboard to manage orders and track performance. The system is responsive, ensuring a smooth experience across different devices, and its core functionalities have been thoroughly tested to meet the expected standards for reliability and security.

Despite the system's robust functionality, there is significant potential for future improvements. Enhancements such as **real-time order tracking**, **mobile app development**, and the addition of multiple payment gateways would further increase its usability and reach. Other future upgrades like **loyalty programs**, **personalized recommendations**, and **third-party delivery service integration** could enhance user engagement and vendor efficiency.

Chapter 9
Future Scope

- **Vendor Analytics:**

  Enhancing the vendor dashboard with deeper analytics and reporting tools can help vendors make better business decisions. This can include sales insights, customer demographics, inventory management, and performance tracking.

- **Internationalization and Localization:**

  Expanding the system to support multiple languages, currencies, and different time zones could open up new market opportunities. Localization would also improve the experience for users in different regions.

- **Internationalization and Localization:**

  Expanding the system to support multiple languages, currencies, and different time zones could open up new market opportunities. Localization would also improve the experience for users in different regions.

References

[1] React. (n.d.). *React – A JavaScript library for building user interfaces.* Retrieved from `https://reactjs.org/`

[2] Django Software Foundation. (n.d.). *Django Documentation.* Retrieved from `https://www.djangoproject.com/`

[3] Razorpay. (n.d.). *Razorpay API Documentation.* Retrieved from `https://razorpay.com/docs/`