Food Delivery Website

A Project Report

Submitted in partial fulfilment of the

Requirements for the award of the Degree of

BACHELOR OF SCIENCE (INFORMATION TECHNOLOGY)

By

Shubham Lakhendra Thakur (9552)

Under the esteemed guidance of

Mr. Omkar Sherkhane

Designation: Assistant Professor

DEPARTMENT OF INFORMATION TECHNOLOGY

MAHATMA EDUCATION SOCIETY'S

PILLAI COLLEGE OF ARTS, COMMERCE AND SCIENCE (AUTONOMOUS)

(Affiliated to University of Mumbai)

NEW PANVEL, 410206

MAHARASHTRA

2024-2025

MAHATMA EDUCATION SOCIETY'S

Pillai College of Arts, Commerce and Science (Autonomous), New Panvel

(Affiliated to University of Mumbai)

CERTIFICATE

This is to certify that the project entitled "Food Delivery Website" is the Bonafide work of "Mr. Shubham Lakhendra Thakur", bearing Seat No. 9552, submitted in partial fulfilment for the completion of the B.Sc. degree in Information Technology at the University of Mumbai.

Internal Guide

External Examiner

College seal

Date:

Co-Ordinator

ACKNOWLEDGEMENT

I, **Mr. Shubham Lakhendra Thakur** student of **Pillai College Of Arts, Commerce & Science (Autonomous), New Panvel** would like to express my sincere gratitude towards our college's Information Technology Department.

I would like to thank Mrs. Deepika Sharma (Vice Principal) for granting me the opportunity to build a project for the college. Last but not least I thank our guide Prof. Omkar for his constant support during this project. The project would have not been completed without the dedication, creativity and the enthusiasm my family provided me.

Yours faithfully,

SHUBHAM LAKHENDRA THAKUR

(Final Year Information Technology)

DECLARATION

I hereby declare that the project entitled, **"Food Delivery Website"** done at **Pillai College Of Arts, Commerce & Science (Autonomous), New Panvel**, has not been in any case duplicated to submit to any other university for the award of any degree. To the best of my knowledge other than me, no one has submitted to any other university.

The project is done in partial fulfilment of the requirements for the award of degree of **BACHELOR OF SCIENCE (INFORMATION TECHNOLOGY)** to be submitted as final semester project as part of our curriculum.

Name and Signature of the Student

MR.SHUBHAM LAKHENDRA THAKUR

TABLE OF CONTENTS

Chapter 1

INTRODUCTION TO FOOD DELIVERY WEBSITE

1.1Background

Traditionally, ordering food involved direct phone calls to restaurants, which were often cumbersome and limited in scope. Such systems were prone to errors, lacked real-time tracking capabilities, and offered limited payment options, resulting in suboptimal user experiences.

The emergence of online food delivery platforms revolutionized this landscape by introducing web-based interfaces that allowed users to browse restaurant menus, place orders online, and have food delivered to their doorstep. These platforms addressed many of the inefficiencies of traditional ordering systems, offering greater convenience and a wider range of restaurant choices.

The proposed food delivery service website aims to build upon the foundation laid by existing platforms by integrating advanced functionalities and enhancing user engagement. This project seeks to develop a comprehensive platform that not only facilitates seamless order placement and secure transactions but also enhances user interaction through personalized recommendations, group ordering options, and intuitive order scheduling capabilities.

Despite these advancements, challenges persisted. Existing online food delivery systems often lacked robust personalization features, real-time order tracking, and comprehensive user engagement tools. Moreover, security concerns related to payment processing and data protection remained critical considerations for both users and service providers.

Current Challenges in Food Delivery:

As online food delivery gained popularity, new challenges emerged for both consumers and service providers. For consumers, concerns included the reliability of delivery times, the accuracy of orders, and the quality of customer support. Meanwhile, restaurants faced issues such as managing fluctuating demand, optimizing delivery routes, and maintaining consistent service standards.

1.2 Objective:

In response to the growing demand for convenient food delivery services, this project aims to develop a comprehensive food delivery website. The objective is to create a platform that enhances user experience through intuitive design, advanced features, and seamless functionality. By leveraging modern web technologies, the website will address existing inefficiencies in traditional food ordering systems and offer significant improvements in terms of convenience, personalization, efficiency, and security.

Objectives:

1) Enhanced User Experience: The primary objective of this project is to provide users with a seamless and enjoyable food ordering experience. By developing a user-friendly interface, intuitive navigation, and responsive design, the website aims to simplify the process of browsing restaurants, viewing menus, placing orders, and tracking deliveries. Key features such as personalized recommendations, order scheduling, group ordering, and secure payment options will be implemented to cater to diverse user preferences and requirements.

2) Improving Operational Efficiency: Another core objective is to streamline operations for both customers and restaurants. The proposed system will replace traditional phone-based orders and manual tracking with an automated process that ensures accuracy and efficiency. For restaurants, the website will facilitate menu management, order processing, and real-time updates on order statuses. Customers will benefit from a transparent and reliable system that provides instant confirmations, accurate delivery times, and options for feedback and reviews.

3) Security and Trustworthiness: Ensuring the security of transactions and protecting user data are critical objectives of this project. By implementing secure payment processing mechanisms and adhering to best practices in data protection, the website will establish trust and credibility among users. Robust security measures will be integrated at every stage of the ordering and payment processes to safeguard sensitive information and mitigate potential risks.

4) Scalability and Future Growth: The project aims to develop a scalable solution capable of handling a large volume of users and orders concurrently. By adopting flexible architecture and efficient database management practices, the website will accommodate future growth and expansion. Potential enhancements such as developing a mobile application, integrating AI for advanced recommendations, enabling voice-based ordering, enhancing analytics capabilities,

and optimizing delivery routes will be considered to further enhance the platform's scalability and competitive edge in the market.

5) Setting Industry Standards: Ultimately, the objective is to set a new standard in the food delivery industry by delivering superior service and user satisfaction. Through continuous innovation and responsiveness to user feedback, the website will strive to become a preferred choice for customers seeking convenience, reliability, and quality in food delivery services. By exceeding expectations and surpassing existing benchmarks, the project aims to establish a strong market presence and achieve sustainable growth in the competitive food delivery market.

1.3 Purpose :

Enhancing User Experience:

The proposed website aims to revolutionize the user experience by making the food ordering process as smooth and enjoyable as possible. The platform will feature an aesthetically pleasing design and intuitive navigation that allows users to find their desired restaurants and menu items quickly. Personalized recommendations based on user preferences and past orders will make it easier for users to discover new dishes and restaurants, enhancing their overall experience.

Streamlining Order Management:

Restaurants often face challenges with traditional order-taking methods, which can lead to errors and inefficiencies. This project aims to provide restaurants with a robust system for managing orders digitally. Features such as real-time order status updates, efficient menu management, and streamlined order processing will help restaurants reduce errors and improve service speed.

Ensuring Security:

In the digital age, data security is of paramount importance. The proposed website will incorporate state-of-the-art security measures to protect users' personal and financial information. Secure payment gateways and encryption protocols will be employed to ensure that transactions are safe and reliable, thereby fostering trust among users.

Promoting Convenience:

Convenience is a significant driving factor for the success of any online service. By allowing users to place orders from multiple restaurants, schedule deliveries, and even create group orders for events or gatherings, the proposed website aims to offer unmatched convenience. Real-time order tracking will provide users with updates on their order status, enhancing the overall service experience.

1.4) Scope:

User Module:

The user module will facilitate user registration and login, enabling users to create accounts and manage their profiles. Users will be able to view their order history, mark favorite restaurants and menu items, and receive personalized recommendations. This module will also include features for password recovery and account management.

Order Management Module:

This module will handle the entire order lifecycle, from adding items to the cart to placing and confirming orders. Users will be able to modify their carts, select delivery options, and review order details before finalizing their purchases. The system will ensure that orders are processed efficiently and accurately.

Payment Module:

A critical component of the website, the payment module will facilitate secure transaction processing. Users will be able to pay for their orders using various payment methods, including credit/debit cards, digital wallets, and other online payment systems. The module will also generate invoices for users' records.

Customer Support Module:

To ensure a high level of customer satisfaction, the website will include a comprehensive customer support module. Users will have access to a help center with FAQs to address common queries. Additionally, they will be able to provide feedback and reviews, helping the platform improve its services continuously.

1.5) Applicability

For Customers:

The website will cater to a diverse user base, including busy professionals, students, and families looking for convenient meal options. By providing a wide range of restaurants and cuisines, the platform will meet the varied tastes and preferences of its users. The personalized recommendation system will further enhance the user experience by suggesting dishes and restaurants based on individual preferences.

For Delivery Personnel:

Delivery personnel will benefit from an organized system that provides real-time updates on order statuses and delivery routes. The route optimization feature will help couriers deliver orders more efficiently, reducing delivery times and improving service quality.

For Administrators:

Platform administrators will have access to a suite of tools for managing user accounts, monitoring transactions, and ensuring the smooth operation of the website. They will be able to address issues promptly, manage platform security, and implement new features based on user feedback and industry trends.

Chapter 2

SURVEY OF TECHNOLOGIES

System Planning

2.1 Survey of Technologies

Frontend Technologies:

1. **HTML (HyperText Markup Language):** HTML is the standard language for creating web pages. It provides the structure for content, such as headings, paragraphs, links, images, and other elements. For the food delivery website, HTML will be used to structure the layout of pages like the homepage, menu details, and checkout pages.

2. **CSS (Cascading Style Sheets):** CSS is used to style the HTML content, controlling the layout, colors, fonts, and overall design of the website. It allows for a consistent look and feel across different pages and devices.

3. **JavaScript:** JavaScript adds interactivity to the website, enabling dynamic updates and interactions without requiring page reloads. It is used for features such as real-time search, form validation, and live notifications.

4. **Java** is a class-based, object-oriented programming language that is designed to have as few implementation dependencies as possible. It is a general-purpose programming language intended to let application developers write once, run anywhere (WORA), meaning that compiled Java code can run on all platforms that support Java without the need for recompilation. Java applications are typically compiled to bytecode that can run on any Java virtual machine (JVM) regardless of the underlying computer architecture. The syntax of Java is similar to C and C++, but has fewer low-level facilities than either of them.

Backend Technologies:

SQL (Structured Query Language): SQL is used for managing and querying relational databases. It handles tasks such as storing user information, restaurant data, order details, and payment transactions.

2.2 Fact and Finding Techniques

Fact finding is process of collection of data and information based on techniques which contain sampling of existing documents, research, observation, questionnaires, interviews, prototyping and joint requirements planning. System analyst uses suitable fact-finding techniques to develop and implement the current existing system. Collecting required facts are very important to apply tools in System Development Life Cycle (SDLC) because tools cannot be used efficiently and effectively without proper extracting from facts. Fact-finding techniques are used in the early stage of System Development Life Cycle including system analysis phase, design and post implementation review. Facts included in any information system can be tested based on three steps: data- facts used to create useful information, process- functions to perform the objectives and interface- designs to interact with users.

2.3 Feasibility Study

An important outcome of preliminary investigation is to determine whether the proposed system is feasible or not. A feasibility study is a test of a system proposal according to its work ability, ability to meet users need an effective use of resources. It is conducted to select the best system that meets the performance standards. It reviews the alternative solution and its objective is not solving the problem.

Three key considerations are involved in feasibility analysis.

• Economic feasibility

• Technical feasibility

• Operational feasibility

Economic feasibility:

Economic Feasibility is considered as the cost/benefit analysis of the proposed project. It is also helpful to find out if the system is economically feasible. Economic or financial feasibility is the second part of the resource determination. The effectiveness of the system. Commonly known as cost benefit analysis, it determining the benefits and savings that are expected from the candidate's system. This is an ongoing process that improves accuracy at each phase of the system development life cycle. The basic resources to be considered are as follows:

● Cost of doing entire system study

● Estimate cost of hardware

● Estimated cost of software

Expenditure may be incurred at the administrator's end as the website will be required to be hosted on the Internet. Plus, asthe data being transacted increasesin volume, extra expenditure may incur to lease or buy memory in order to accommodate the data. On the other hand, the user will use the website at no extra cost.

Technical Feasibility:

Technical feasibility centers on existing computer system for hardware and software and to what extent it can support the proposed system. The current technical resource available in the organization is capable of handling user requirements. The proposed website is technically feasible, as the users will access it directly via the Web Browsers in their systems. The website is also technically feasible to the administrators as they require no extra software or hardware, other than the backend dashboard provided by the server software.

Operational Feasibility:

It considered the acceptability of the system and checks whether

• System will be used if it is developed and implemented.

• The user will be able to handle the system easily.

• Whether the proposed system will cause any trouble. The website is user-friendly, not only to the users, but also to the administrators handling its workings. The website gives full control over every aspect to the administrators, which in turn allows them to manage and monitor the content being posted on the website.

2.4 Stakeholders

Stakeholders are individuals or groups who have an interest in the project and its outcomes. Identifying and engaging with stakeholders is crucial for ensuring that the project meets their needs and expectations.

1. **Customers:** The primary users of the food delivery website who will browse restaurants, place orders, and provide feedback. Their needs include a user-friendly interface, accurate order tracking, and secure payment options.
2. **Developers:** The developers are the ones who design, test and maintain the project.
3. **Investor:** Investors are those who invest their money in our website for their profits.
4. **Advertiser:** These are the group of people who promotes their products or services in our website.

Chapter 3

REQUIREMENT AND ANALYSIS

3.1 Problem Definition:

In the modern world, the demand for convenience in food delivery has surged, yet existing traditional and online food ordering systems are fraught with limitations. Traditional methods, such as phone orders, are prone to errors, inefficiencies, and limited payment options, leading to subpar customer experiences. While current online food delivery platforms have mitigated some of these issues, they still fall short in areas like personalization, real-time tracking, and user engagement.

The primary problem is the lack of a comprehensive, user-friendly, and secure food delivery platform that caters to both customers and restaurants, offering a seamless experience from order placement to delivery. Existing systems do not fully utilize modern web technologies and advanced features, resulting in a gap between user expectations and the services provided. This project aims to address these challenges by developing an intuitive, feature-rich food delivery website that enhances user convenience, security, and overall satisfaction.

3.2 Requirement Specification:

Functional Requirements

- User Registration & Login: Users can create accounts via email, phone, or social media, and log in securely.
- Profile Management: Users can update personal details, change passwords, and set delivery preferences.
- Order History & Favorites: Users can view past orders and save favorites.
- Restaurant Module:
    - Restaurants can register, manage profiles, update menus, and handle orders.
- Order Management Module:
    - Users can manage items in the cart, place orders, and receive order confirmations.
- Payment Module:
    - Secure payment processing and automatic invoice generation.
- Customer Support Module:
    - Help center, FAQs, and feedback/reviews system.

2. Non-Functional Requirements

- Performance: The system should handle many users and orders with minimal latency.
- Usability: Intuitive, responsive design across devices with efficient task completion.
- Security: Data encryption, robust authentication, and regular security audits.
- Reliability: High availability, minimal downtime, and graceful degradation during failures.
- Scalability: The system must scale easily to accommodate growth.
- Maintainability: Modular codebase with documentation for easy updates and maintenance.

3.3 Planning and Scheduling:

Sr No	Task	Month	June		July				August					September		
		Date	20	27	4	11	18	25	1	8	22	29	5	12	19	26
1	INTRODUCTION															
	Background	Planned														
		Actual														
	Objectives	Planned														
		Actual														
	Purpose, scope & Applicability	Planned														
		Actual														
2	SURVEY OF TECHNOLOGY															
	Survey	Planned														
		Actual														
	Fact-Finding	Planned														
		Actual														
	Feasibility study	Planned														
		Actual														
	Stake Holder	Planned														
		Actual														
3	Requirement and Analysis															
	Problem Definition	Planned														
		Actual														
	Requirement Specification	Planned														
		Actual														
	Planning & scheduling	Planned														
		Actual														
	Software &	Planned														

	Hardware req.	**Actual**															
	Conceptual Model	**Planned**															
		Actual															
4	**System Design**																
	Basic Model	**Planned**															
		Actual															
	Data Design	**Planned**															
		Actual															
	Data integrity & Constraints	**Planned**															
		Actual															
	User Interface & Design	**Planned**															
		Actual															
	Security Issues	**Planned**															
		Actual															
	Test Case Design	**Planned**															
		Actual															

Sr. No.	Task	Month	Dec			Jan				Feb		
		Date	20	27	29	4	18	20	27	8	16	
5	System Coding, Implementation and Testing											
	Implementation and approaches	Planned										
		Actual										
	Coding Details and Coding Efficiency	Planned										
		Actual										
	Testing Approach	Planned										
		Actual										
6	Conclusion and Future Works											
	Conclusion and Future Scope	Planned										
		Actual										
	Limitations	Planned										
		Actual										
	Feasibility Study	Planned										
		Actual										
	Stakeholder	Planned										
		Actual										
7	References											
	References	Planned										
		Actual										

3.4 Software and Hardware Requirements:

1. Software Requirements

Frontend Development:

- HTML, CSS, JavaScript:
    - Used for creating the structure, styling, and interactivity of the website.

Backend Development:

- Programming Languages:
    - Node.js (JavaScript) or Python (Django/Flask): For server-side development.

Database:

- SQL-based: MySQL for storing user, order, and restaurant data.

2. Hardware Requirements

2.1 Development Environment

- Development Machines:
    - Laptops or Desktops: Each with at least:
        - Processor: Intel i5 or AMD Ryzen 5 equivalent or higher.
        - RAM: 16 GB minimum.
        - Storage: 512 GB SSD minimum.
        - OS: Windows 10/11, macOS, or Linux (Ubuntu or similar distributions).

2.2 Testing Environment

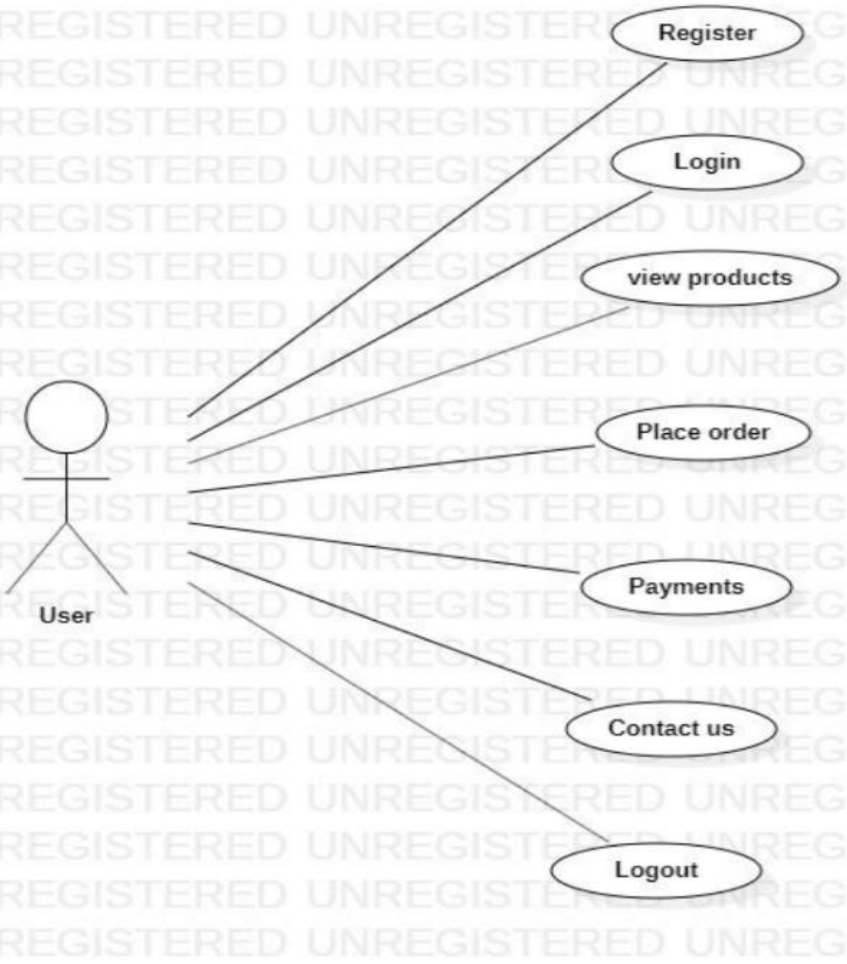
- Test Devices:
    - Desktop: Different screen resolutions and browsers (e.g., Chrome, Firefox, Safari).
    - Mobile Devices: Various smartphones and tablets (iOS and Android) to test responsiveness and mobile usability.

3.5 Conceptual model:

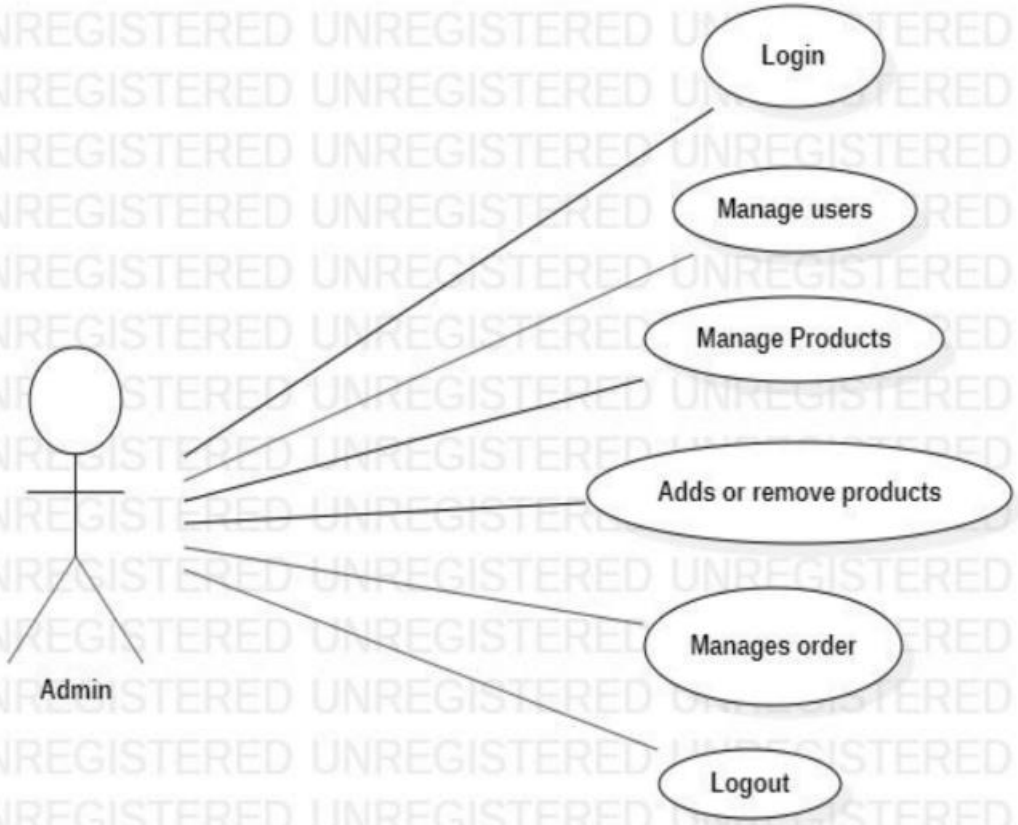
3.5.1) USECASE DIAGRAM:

A use case diagram at its simplest is a representation of a user's interaction with the system that shows the relationship between the user and the different use case in which the user is involved. A use case diagram can identify the different types of users of a system and the different use cases and will often be accompanied by other types of diagrams as well. The use cases are represented by either circles or ellipses.

User/Customer:



Admin:

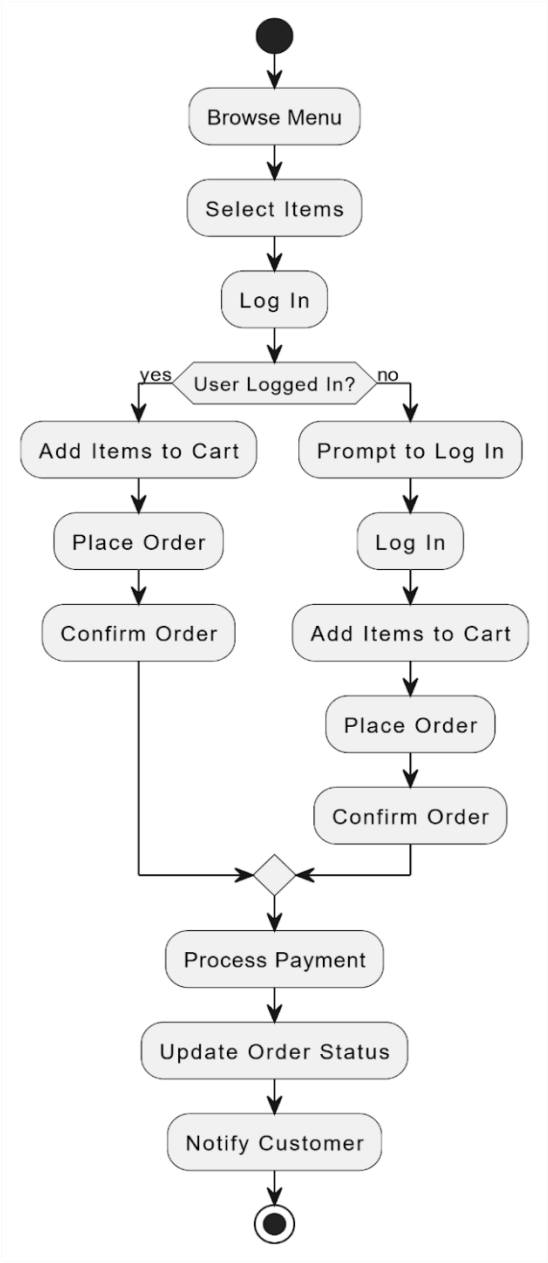


3.5.2) Activity diagram:

- Browse Menu: The customer starts by browsing the menu.
- Select Items: The customer selects items they want to order.
- Log In: The system checks if the user is logged in.

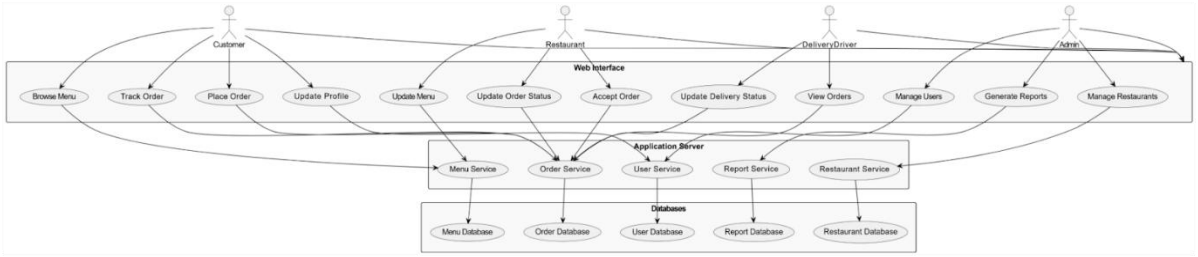
1. If yes, the user can add items to the cart, place the order, and confirm it.
2. If no, the user is prompted to log in before proceeding.

- Add Items to Cart: The selected items are added to the cart.
- Place Order: The customer places the order.
- Confirm Order: The system confirms the order.
- Process Payment: The payment is processed.
- Update Order Status: The status of the order is updated.
- Notify Customer: The customer is notified of the order status.


Browse Menu
Select Items
Log In
yes
User Logged In?
no
Add Items to Cart
Prompt to Log In
Place Order
Log In
Confirm Order
Add Items to Cart
Place Order
Confirm Order
Process Payment
Update Order Status
Notify Customer

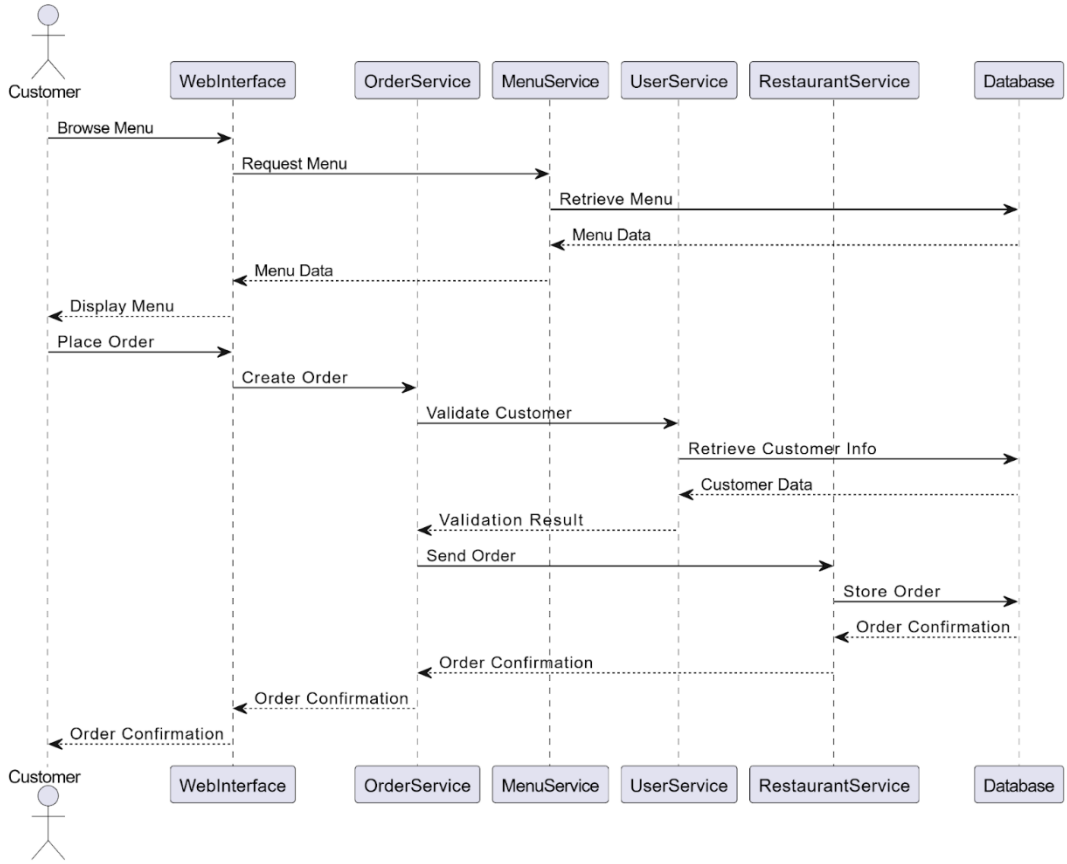
3.5.3) System flow diagram:

A System Flow Diagram (SFD) visualizes the flow of data through a system and its components. For a food delivery website, the SFD would typically include various components like users, databases, and services.



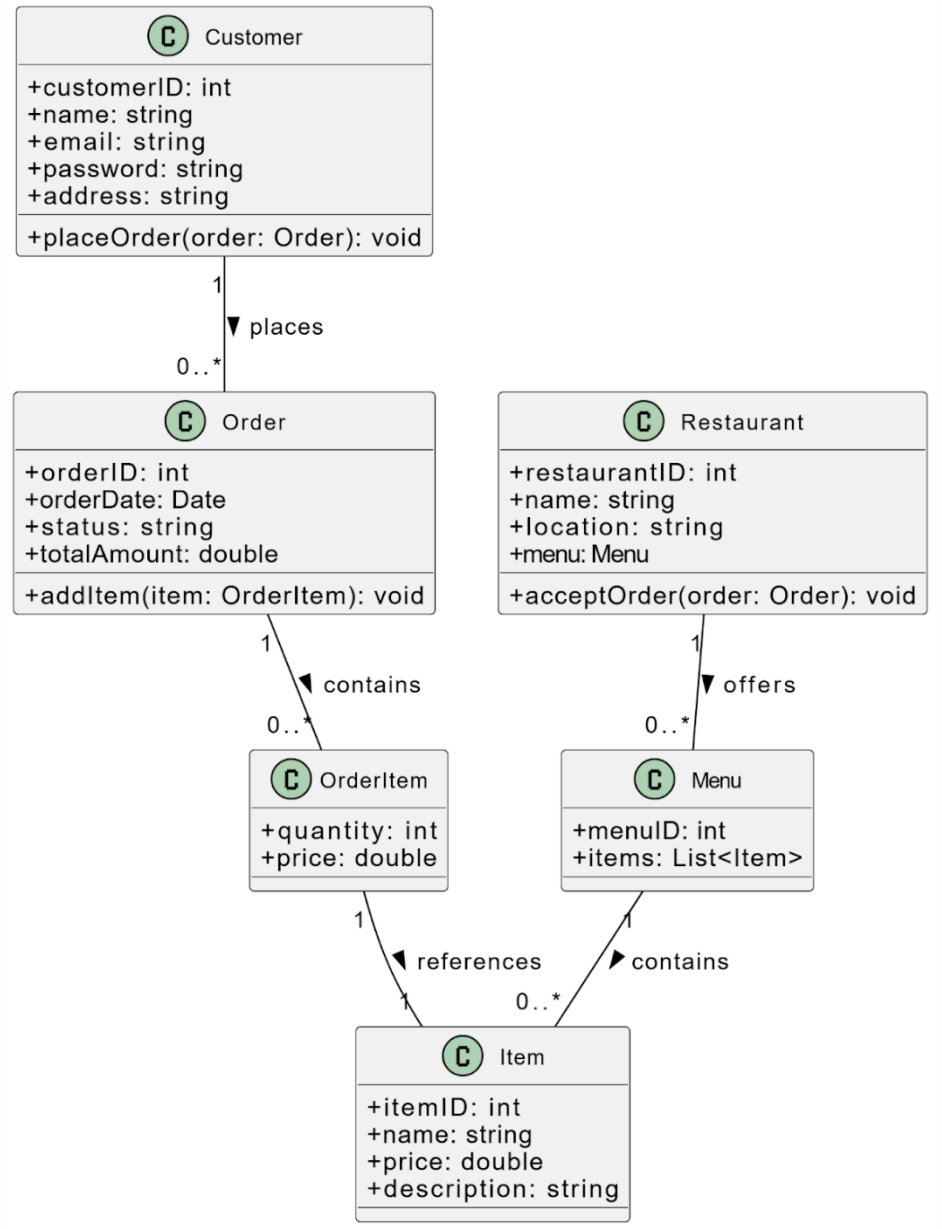
3.5.4) SEQUENCE DIAGRAM

A sequence diagram shows object interactions arranged in time sequence. It depicts the objects and classes involved in the scenario and the sequence of messages exchanged between the objects needed to carry out the functionality of the scenario. Sequence diagrams are typically associated with use case realizations in the Logical View of the system under development. Sequence diagrams are sometimes called event diagrams or event scenarios. A sequence diagram shows, as parallel vertical lines (lifelines), different processes or object that live simultaneously, and, as horizontal arrows, the messages exchanged between them, in the order in which they occur. This allows the specification of simple runtime scenarios in a graphical manner.



3.5.5) Class Diagram:

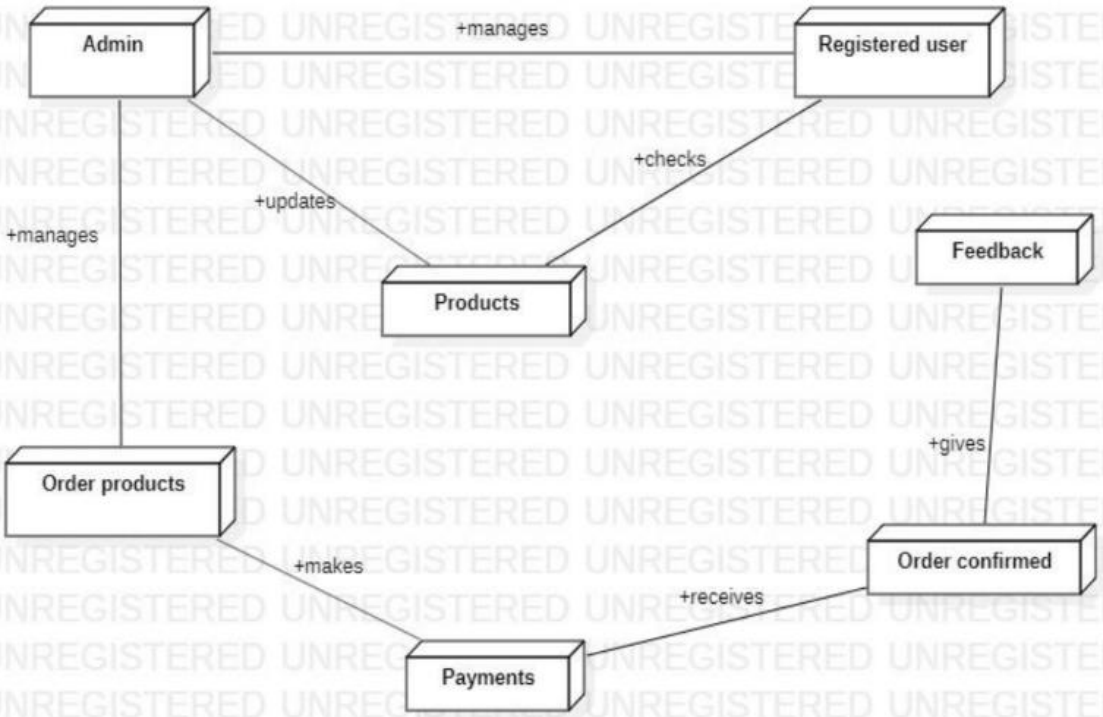
The class diagram represents the static structure of the system by showing its classes ,attributes, methods and relationship.



3.5.6) Deployment Diagram:

The Deployment diagram illustrates how the software is deployed on hardware components.

A deployment diagram is a UML diagram type that shows the execution architecture of a system, including nodes such as hardware or software execution environments, and the middleware connecting them. Deployment diagrams are typically used to visualize the physical hardware and software of a system. Using it you can understand how the system will be physically deployed on the hardware. Deployment diagrams help model the hardware topology of a system compared to other UML diagram types which mostly outline the logical components of a system.



Chapter 4

SYSTEM DESIGN

4.1 BASIC MODULES:

Registration Page

In this module, the users / customers will be asked to enter their personal details such as Name, Email ID, Password, OTP. This information will be stored in the database in an encrypted format. If the user or customer has an account already, he will need to click on the existing user's link and will be redirected to the Log In page.

Log In

In this module, the customer / user will be asked to enter Email ID or username and password. These details will be then verified from the database and the user or customer will be Logged in. If the verification fails then the user will need to sign up through the registration page.

Order

In this module the customer has to enter the details such as name, phone number, email id, total quantity, product price for ordering a particular product.

Payment

In this module, the customer has to enter the payment details asked such as total amount, address for delivery.

4.2 Data Design:

Data design is essential for ensuring that the Food Delivery Website operates efficiently, accurately, and securely. It involves structuring the database to handle critical data like users, restaurants, orders, menus, and deliveries. This section outlines the database schema and key constraints to maintain data integrity and performance.

4.2.1 Schema Design

The schema design defines the structure of the database, focusing on how different entities interact within the system. For the Food Delivery Website, the database schema can include the following key tables:

1. User Table

Table Name: Users

This table stores information about all users (customers, restaurant owners, delivery personnel).

- UserID (Primary Key, Integer, Auto-increment)
- Name (VARCHAR, 255)
- Email (VARCHAR, 255, Unique)
- PasswordHash (VARCHAR, 255)
- Role (ENUM: 'Customer', 'RestaurantOwner', 'DeliveryPersonnel', 'Admin')
- ContactNumber (VARCHAR, 20)
- Address (TEXT)
- ProfilePictureURL (VARCHAR, 255, Nullable)

2. Menu Table

Table Name: MenuItems

This table contains information about the dishes offered by restaurants.

- MenuItemID (Primary Key, Integer, Auto-increment)
- RestaurantID (Foreign Key, references Restaurants(RestaurantID))
- Name (VARCHAR, 255)

- Description (TEXT)
- Price (DECIMAL(10,2))
- ImageURL (VARCHAR, 255)
- AvailabilityStatus (ENUM: 'Available', 'Out of Stock')

4. Order Table

Table Name: Orders

This table stores data related to customer orders.

- OrderID (Primary Key, Integer, Auto-increment)
- UserID (Foreign Key, references Users(UserID))
- RestaurantID (Foreign Key, references Restaurants(RestaurantID))
- TotalAmount (DECIMAL(10,2))
- OrderDate (DATETIME)
- Status (ENUM: 'Pending', 'Confirmed', 'Preparing', 'Out for Delivery', 'Delivered', 'Canceled')

5. Order Item Table

Table Name: OrderItems

Each order can consist of multiple items, stored here.

- OrderItemID (Primary Key, Integer, Auto-increment)
- OrderID (Foreign Key, references Orders(OrderID))
- MenuItemID (Foreign Key, references MenuItems(MenuItemID))
- Quantity (INTEGER)
- Price (DECIMAL(10,2))

6. Delivery Table

Table Name: Deliveries

This table tracks delivery details for orders.

- DeliveryID (Primary Key, Integer, Auto-increment)
- OrderID (Foreign Key, references Orders(OrderID))
- DeliveryPersonnelID (Foreign Key, references Users(UserID))
- DeliveryStatus (ENUM: 'Assigned', 'Picked Up', 'In Transit', 'Delivered', 'Failed')
- DeliveryTime (DATETIME)

7. Payment Table

Table Name: Payments

This table holds information about payment transactions.

- PaymentID (Primary Key, Integer, Auto-increment)
- OrderID (Foreign Key, references Orders(OrderID))
- PaymentMethod (ENUM: 'PayPal', 'CashOnDelivery')
- PaymentStatus (ENUM: 'Pending', 'Completed', 'Failed')
- TransactionID (VARCHAR, 255, Nullable)
- Amount (DECIMAL(10,2))

4.2.2 Data Integrity and Constraints

1. Primary Keys

- Every table must have a unique identifier to distinguish each record. E.g., UserID, OrderID, and RestaurantID.

Example:

- OrderID in the Orders table is the primary key, ensuring that each order is uniquely identified.

```
CREATE TABLE Orders (
    OrderID INT PRIMARY KEY AUTO_INCREMENT,
    UserID INT,
    RestaurantID INT,
```

```
    TotalAmount DECIMAL(10, 2),
    Status ENUM('Pending', 'Confirmed', 'Delivered', 'Cancelled')
);
```

2. Foreign Keys

- Foreign Key Constraints maintain relationships between tables, ensuring referential integrity. For example:
    - OrderID in OrderItems references Orders(OrderID) to link each item to its respective order.
    - RestaurantID in MenuItems links dishes to their restaurant.
    - DeliveryPersonnelID in Deliveries ensures that the assigned delivery personnel exists in the Users table.

Example:

- UserID in the Orders table is a foreign key that references the Users table, ensuring that every order is linked to an existing user.

```
CREATE TABLE Orders (
    OrderID INT PRIMARY KEY AUTO_INCREMENT,
    UserID INT,
    FOREIGN KEY (UserID) REFERENCES Users(UserID),
    TotalAmount DECIMAL(10, 2)
);
```

3. Unique Constraints

- Enforce unique values for specific fields where duplication is not allowed. E.g., Email in the Users table and TransactionID in the Payments table must be unique.

Example:

- Email in the Users table must be unique to ensure no duplicate accounts.

```
CREATE TABLE Users (
```

```
    UserID INT PRIMARY KEY AUTO_INCREMENT,
    Email VARCHAR(255) UNIQUE,
    Name VARCHAR(255),
    PasswordHash VARCHAR(255)
);
```

4. Not Null Constraints

- Ensure certain fields cannot be left empty. E.g., the Name and Email fields in the Users table must not be null.

Example:

- The Name and Email fields in the Users table cannot be left blank (null).

```
CREATE TABLE Users (
    UserID INT PRIMARY KEY AUTO_INCREMENT,
    Name VARCHAR(255) NOT NULL,
    Email VARCHAR(255) NOT NULL
);
```

5. Check Constraints

- Enforce rules on specific fields to maintain data quality. For example:
    - Price in MenuItems should be greater than zero.
    - TotalAmount in Orders must be greater than zero.

Example:

- Price in the MenuItems table must be greater than 0.

```
CREATE TABLE MenuItems (
    MenuItemID INT PRIMARY KEY AUTO_INCREMENT,
    Name VARCHAR(255),
    Price DECIMAL(10, 2) CHECK (Price > 0)
```

);

6. Default Values

- Provide default values for fields when no value is specified. For example:
    - OrderDate in Orders can default to the current date and time.
    - Status in Orders can default to 'Pending' when a new order is placed.

Example:

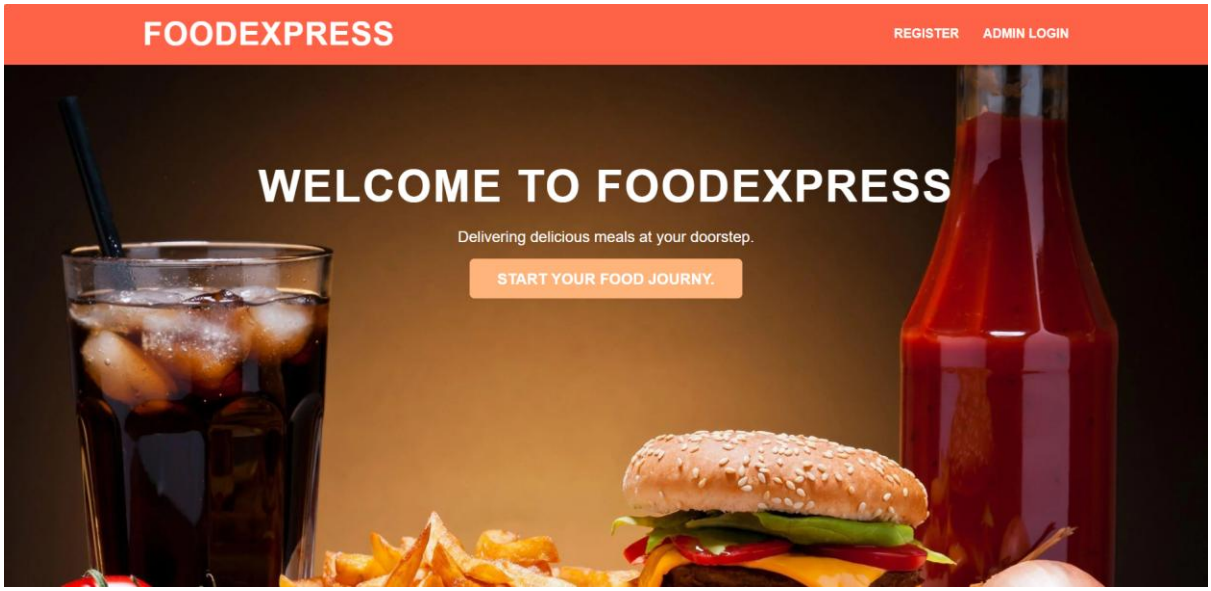
- The OrderDate field in the Orders table defaults to the current timestamp if not provided.

```
CREATE TABLE Orders (
  OrderID INT PRIMARY KEY AUTO_INCREMENT,
  OrderDate TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

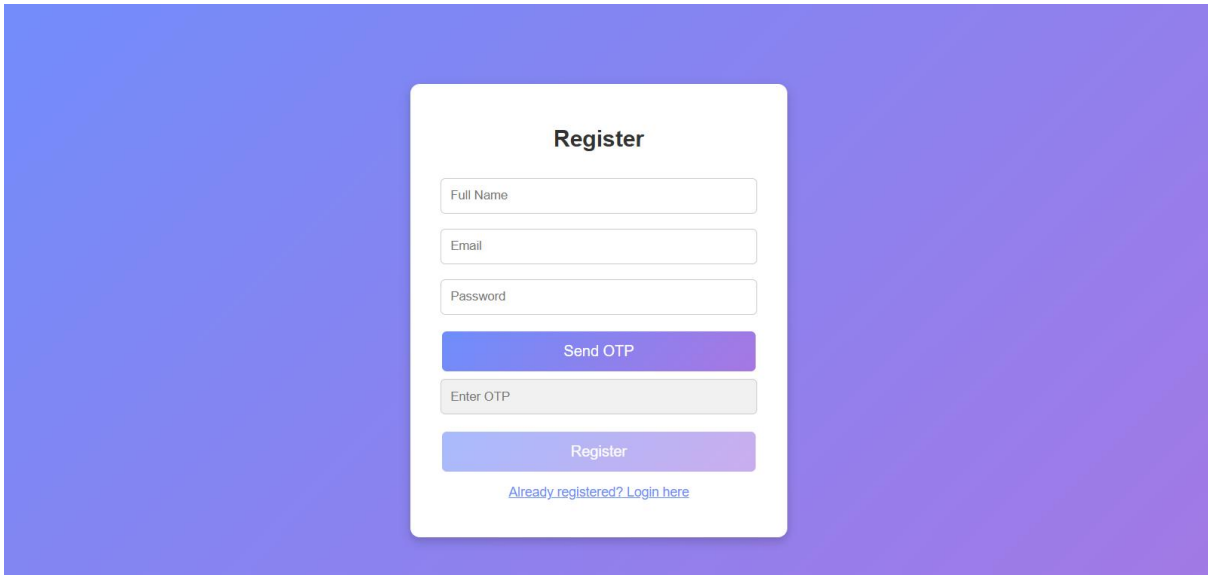
4.3 User interface design:

Client View:

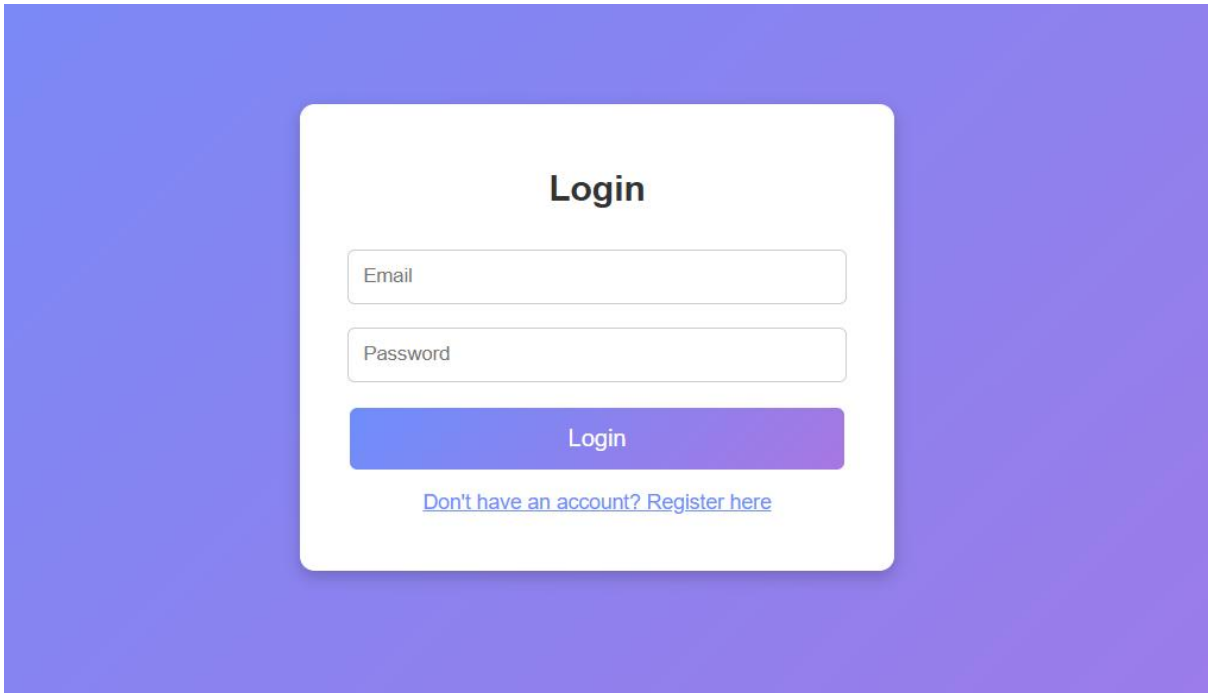
Home page:



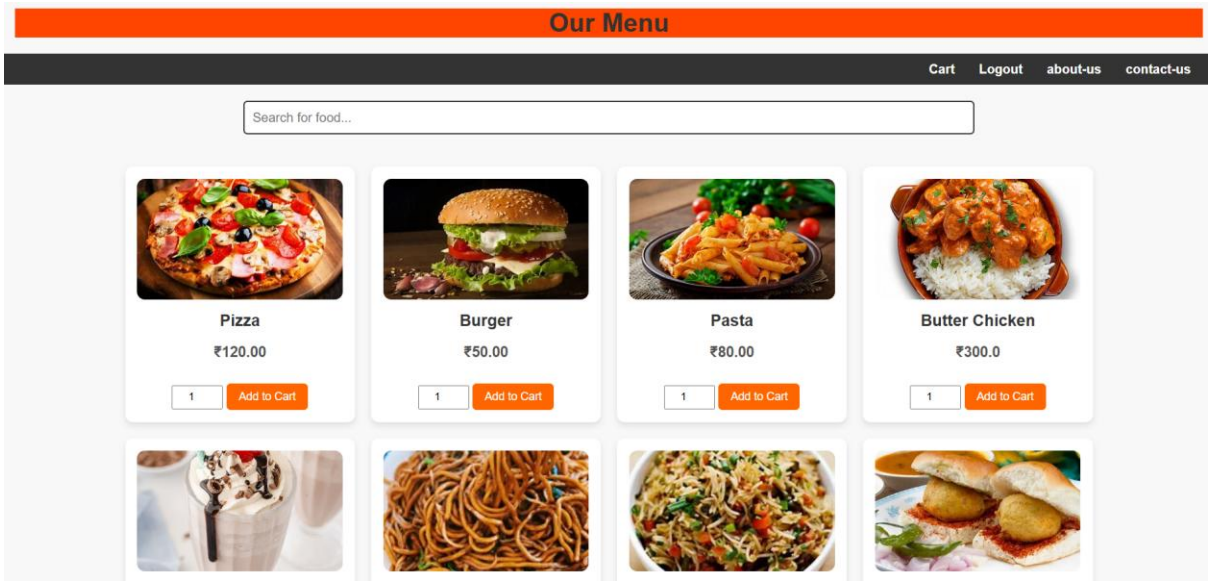
Register page:



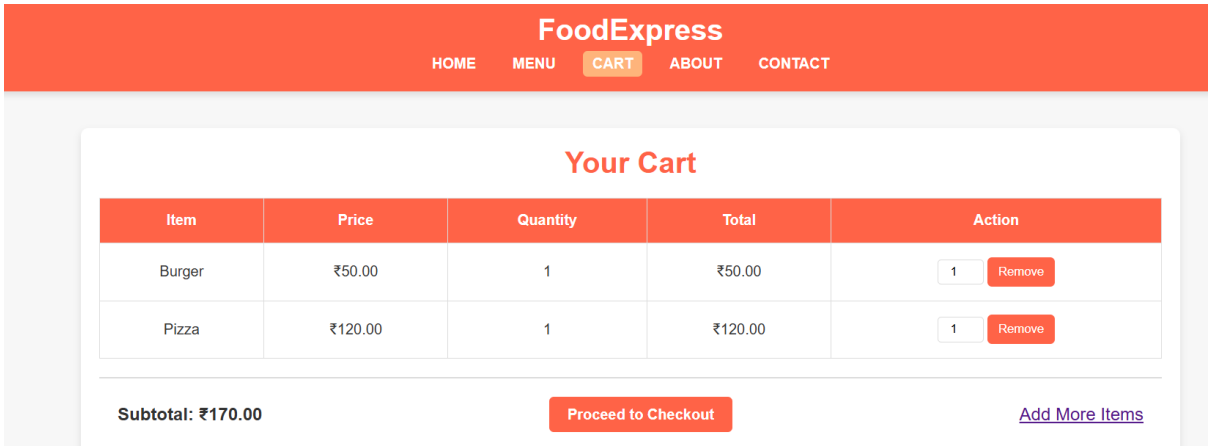
Login Page:



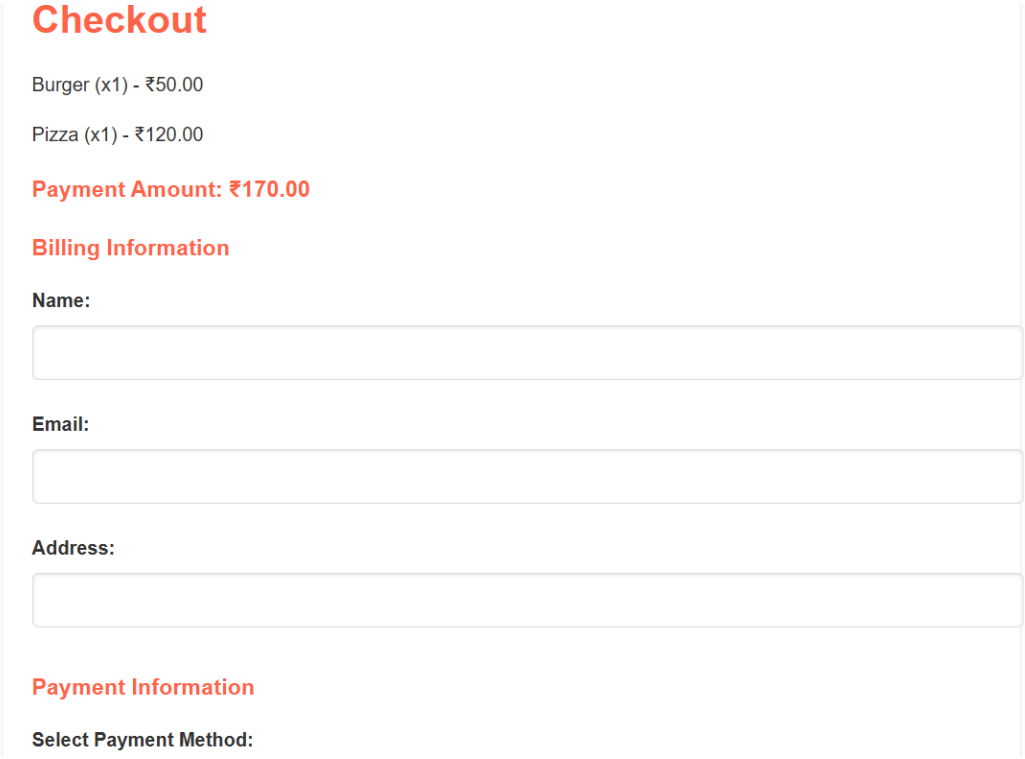
Menu page:



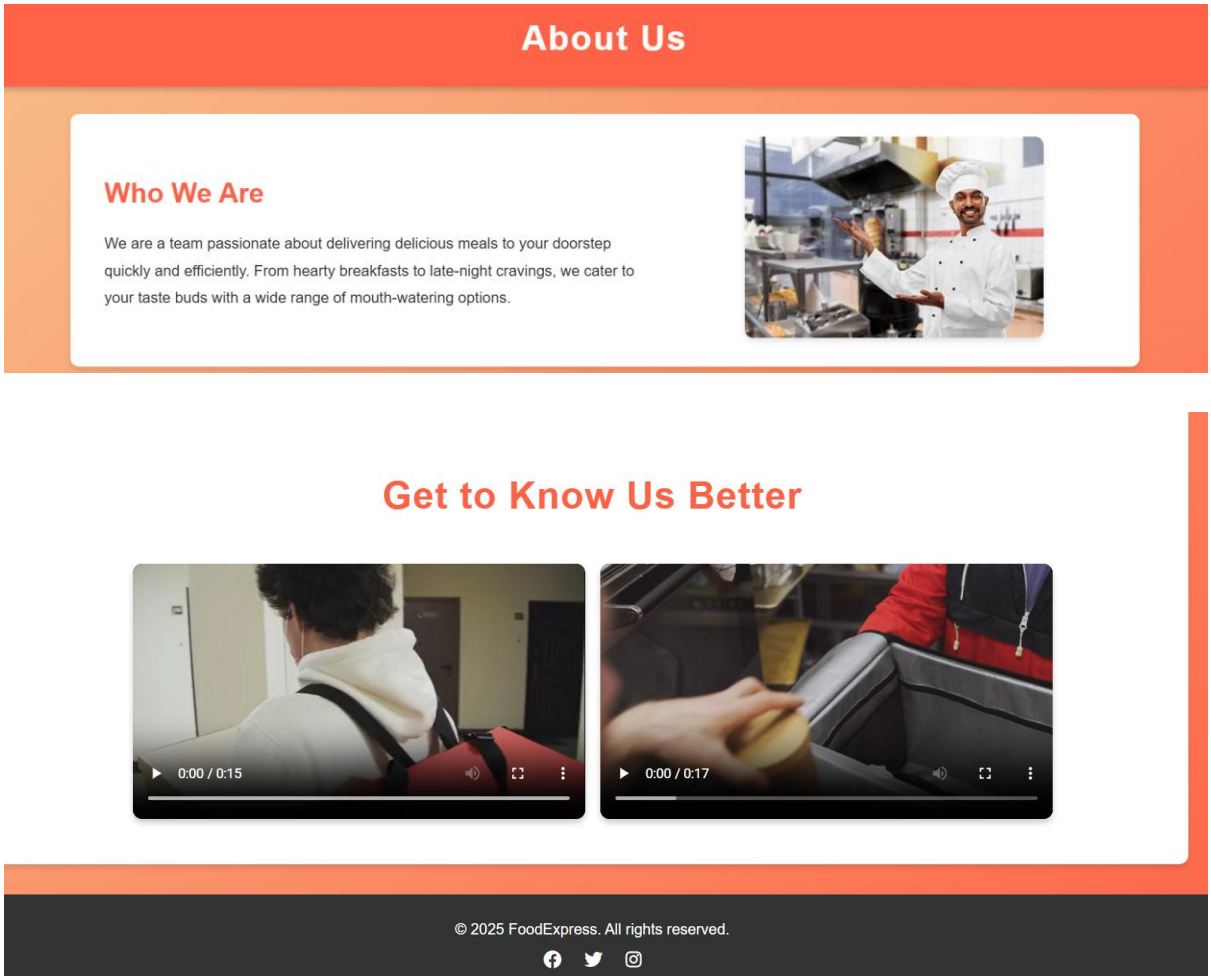
Cart page:



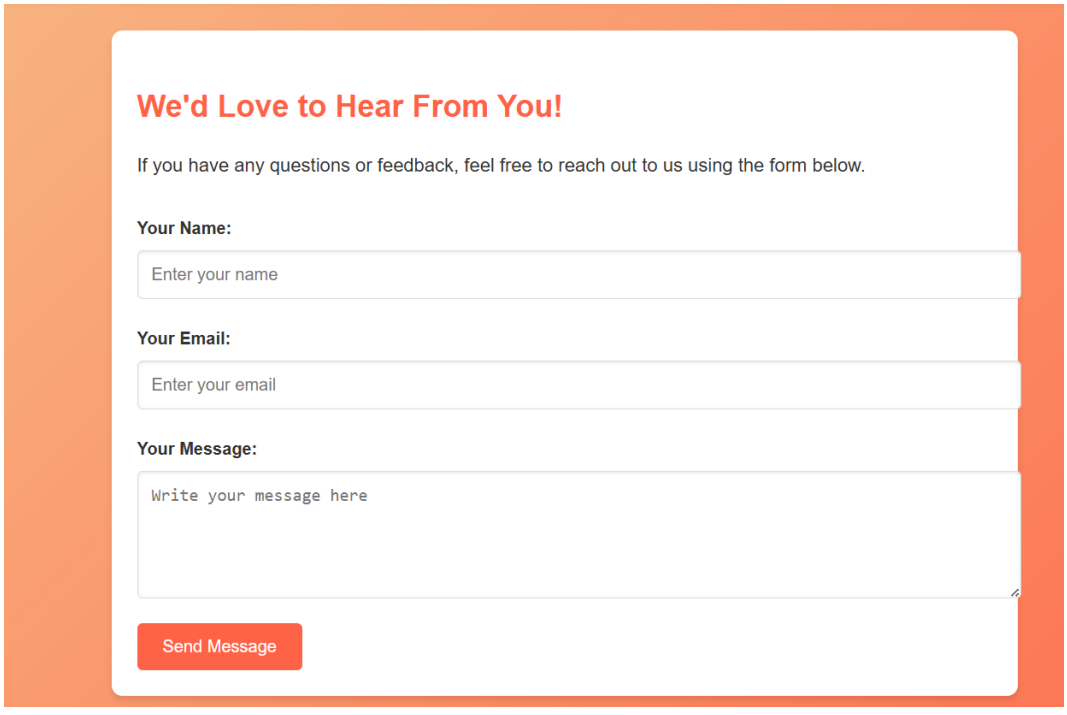
Checkout page:



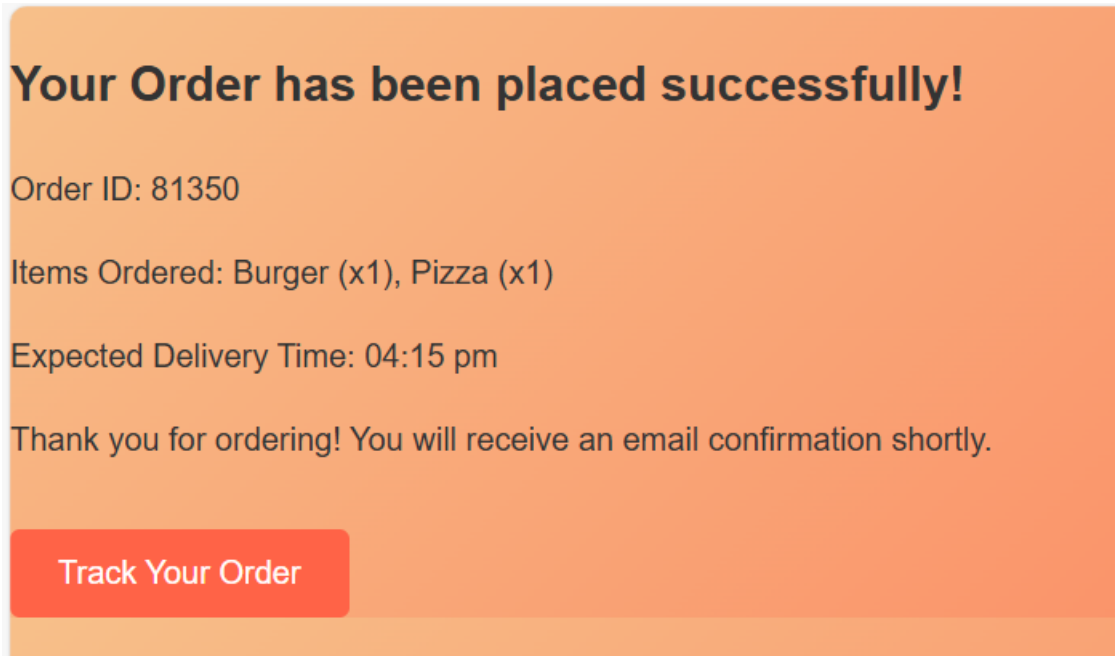
About us:



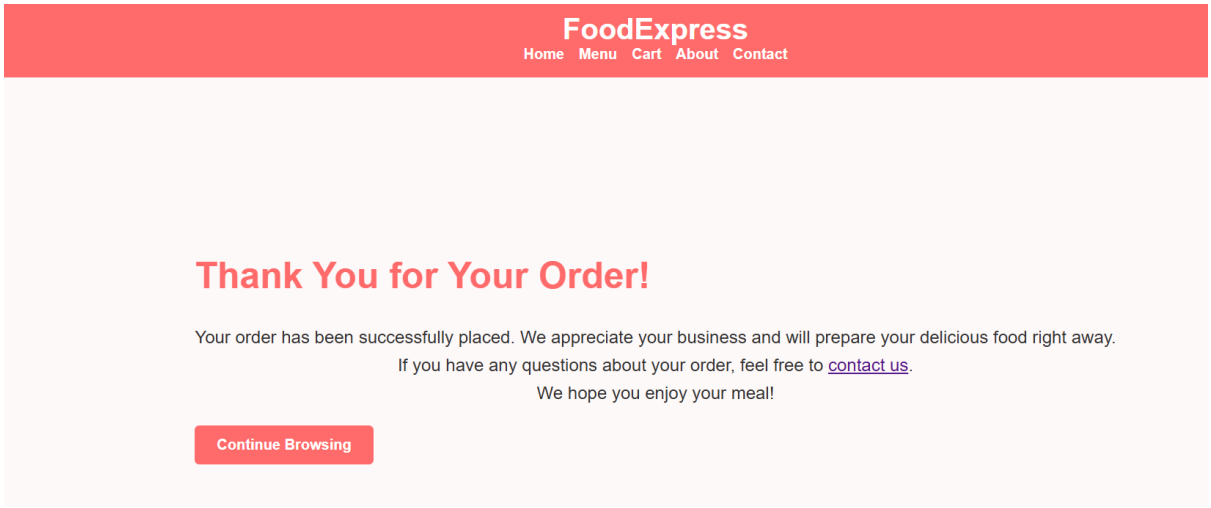
Feedbackpage:



Order confirmation:

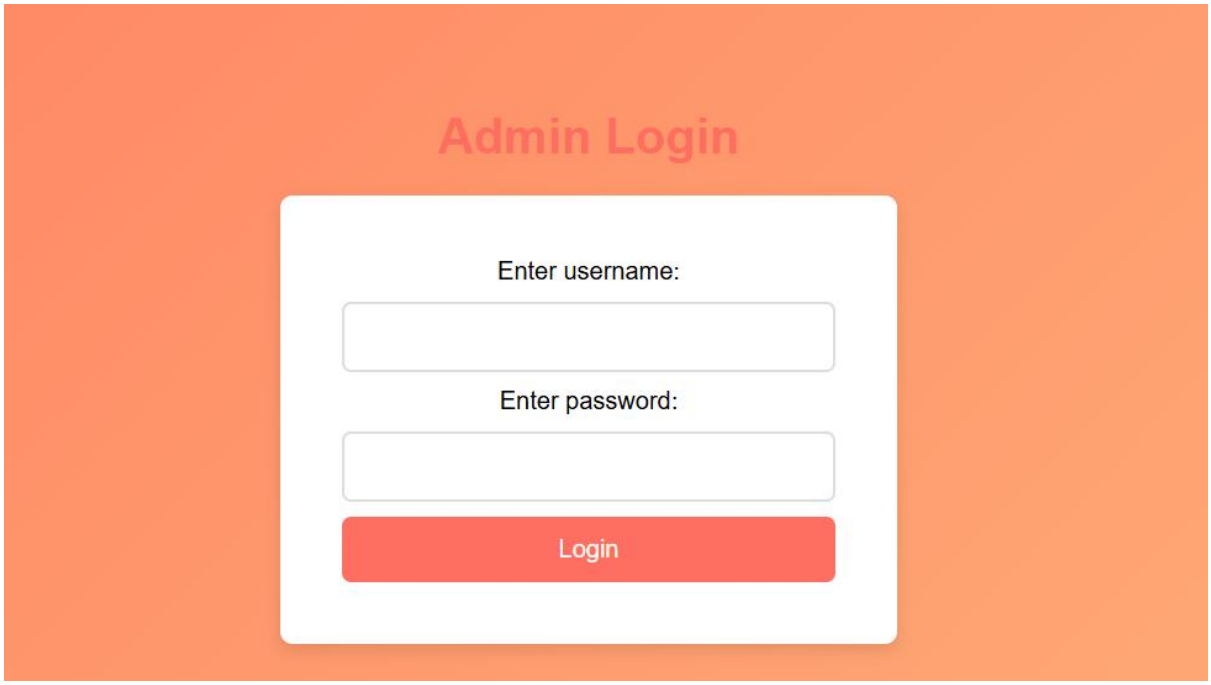


Thankyou page:

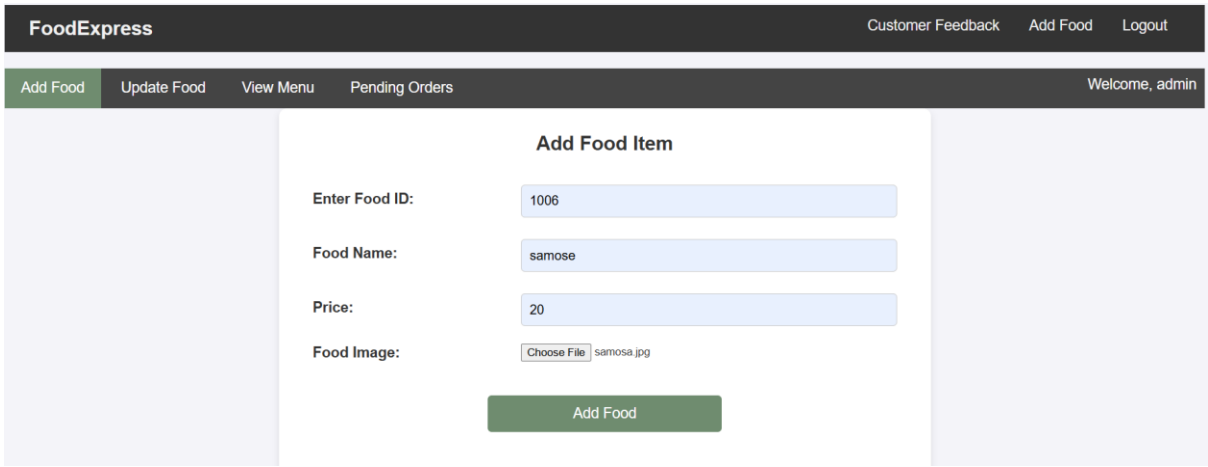


Admin side:

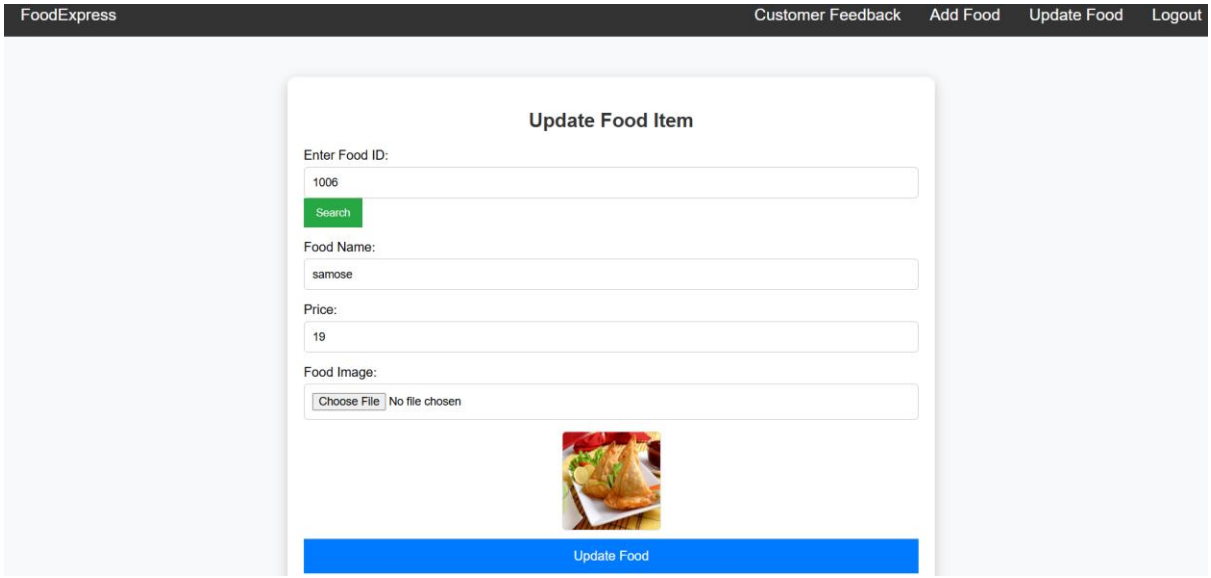
Admin login:



Add food:

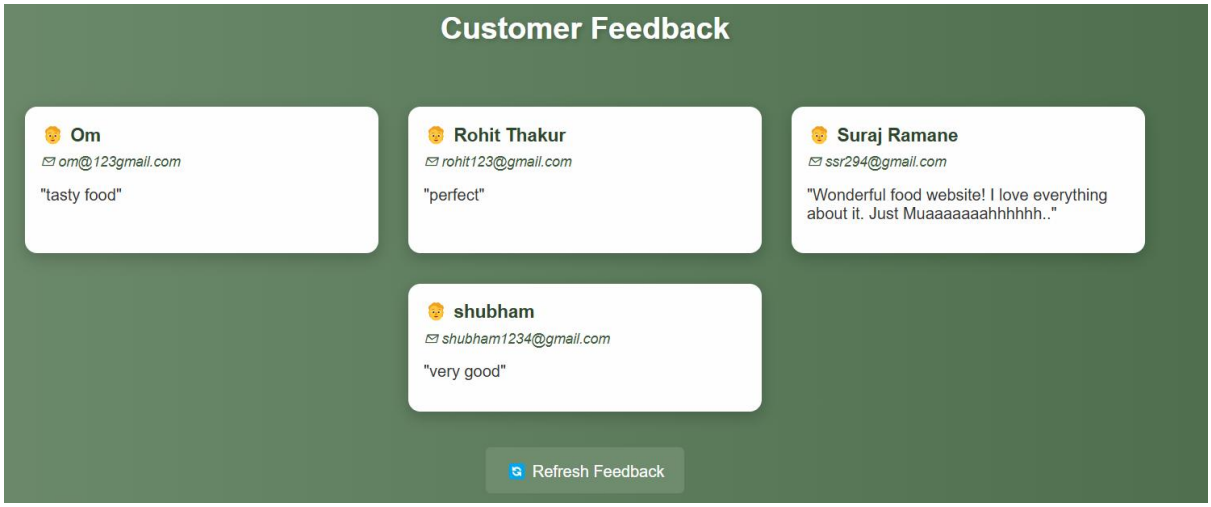


Update Food:



View menu:

Customer feedback:



4.4 Security Issues:

Authentication and Authorization: Use strong login methods like multi-factor authentication and role-based access to control user permissions.

Data Encryption: Encrypt all sensitive data, including user information and payment details, during transmission and storage to protect against interception.

Secure Payment Handling: Ensure compliance with payment standards (PCI-DSS) and use trusted payment gateways to prevent fraud.

Input Validation: Validate and sanitize all user input to prevent SQL injection, cross-site scripting (XSS), and cross-site request forgery (CSRF) attacks.

Session Management: Implement secure session handling, including secure cookies, session expiration, and token-based authentication to prevent hijacking.

Secure File Uploads: Restrict file types, limit file size, and scan uploaded files for malware to avoid malicious file uploads.

Monitoring and Intrusion Detection: Use intrusion detection systems (IDS) and monitor logs to detect suspicious activities.

API Security: Secure APIs with authentication, rate limiting, and input validation to prevent exploitation.

Regular Updates: Keep software, libraries, and dependencies updated to patch known vulnerabilities.

Privacy Compliance: Follow data protection laws (like GDPR) and encrypt user data to ensure privacy.

4.5 Test Case Design

1. User Authentication Test Cases

Test Case 1: User Registration with Valid Data

- Test Objective: Ensure users can successfully register with valid inputs.
- Test Steps:
    1. Navigate to the registration page.
    2. Enter a valid email, password, and other required information.
    3. Submit the form.
- Expected Result: The user is successfully registered and redirected to the login page.
- Status: Pass/Fail

Test Case 2: Login with Invalid Credentials

- Test Objective: Verify that login fails when invalid credentials are used.
- Test Steps:
    1. Navigate to the login page.
    2. Enter an invalid email or password.
    3. Click "Login".
- Expected Result: An error message appears, and login is denied.
- Status: Pass/Fail

2. Menu Management Test Cases (Restaurant Owner)

Test Case 1: Add New Menu Item

- Test Objective: Ensure restaurant owners can add new menu items.
- Test Steps:
    1. Log in as a restaurant owner.

2. Navigate to the menu management section.
3. Add a new menu item (name, price, description).
4. Save the changes.

- Expected Result: The new menu item is displayed in the menu list.
- Status: Pass/Fail

Test Case 2: Update Menu Item

- Test Objective: Verify that restaurant owners can successfully update existing menu items.
- Test Steps:
    1. Log in as a restaurant owner.
    2. Select an existing menu item.
    3. Modify the price or description.
    4. Save the changes.
- Expected Result: The updated details are reflected in the restaurant menu.
- Status: Pass/Fail

3. Order Placement Test Cases (Customer)

Test Case 1: Place Order with Valid Cart

- Test Objective: Verify that users can place orders with valid items in the cart.
- Test Steps:
    1. Browse the restaurant's menu.
    2. Add items to the cart.
    3. Proceed to checkout and confirm the order.
- Expected Result: The order is successfully placed, and the user receives a confirmation message.

- Status: Pass/Fail

Test Case 2: Attempt Order Placement with Empty Cart

- Test Objective: Ensure users cannot place orders with an empty cart.
- Test Steps:
    1. Go to the checkout page without adding items to the cart.
    2. Try to place the order.
- Expected Result: An error message appears, and the order is not placed.
- Status: Pass/Fail

4. Security Test Cases

Test Case 1: CSRF Attack Prevention

- Test Objective: Ensure the website is protected from Cross-Site Request Forgery (CSRF) attacks.
- Test Steps:
    1. Attempt to submit a form without the CSRF token.
- Expected Result: The form submission fails, and an error message is displayed.
- Status: Pass/Fail

Test Case 2: SQL Injection Prevention

- Test Objective: Verify that input fields are protected from SQL injection attacks.
- Test Steps:
    1. Enter SQL code into login or search fields (e.g., SELECT * FROM Users;).
- Expected Result: The system does not execute the SQL code, and the input is safely handled.
- Status: Pass/Fail

Chapter 5

5.1 Coding Details.

5.2 Coding Efficiency

Client Side:

1)Login.html:

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<title>Registration & Login Form</title>
<style>
body {
font-family: 'Poppins', sans-serif;
background: linear-gradient(135deg, #6e8efb, #a777e3);
display: flex;
justify-content: center;
align-items: center;
height: 100vh;
margin: 0;
}

.container {
background: white;
padding: 25px;
width: 350px;
```

```
border-radius: 10px;
box-shadow: 0px 4px 10px rgba(0, 0, 0, 0.2);
text-align: center;
}

h2 {
margin-bottom: 20px;
color: #333;
}

input {
width: 90%;
padding: 10px;
margin: 8px 0;
border: 1px solid #ccc;
border-radius: 5px;
outline: none;
transition: 0.3s;
}

input:focus {
border-color: #6e8efb;
box-shadow: 0 0 8px rgba(110, 142, 251, 0.6);
}

button {
```

```
width: 95%;
padding: 12px;
border: none;
border-radius: 5px;
background: linear-gradient(135deg, #6e8efb, #a777e3);
color: white;
font-size: 16px;
cursor: pointer;
transition: 0.3s;
margin-top: 10px;
}

button:hover {
transform: scale(1.05);
box-shadow: 0px 4px 10px rgba(110, 142, 251, 0.6);
}

#otp {
background: #f1f1f1;
cursor: not-allowed;
}

.disabled {
opacity: 0.6;
cursor: not-allowed;
}
```

```
.form-container {
display: flex;
justify-content: space-between;
}

.form-container div {
width: 48%;
}

.switch-form {
font-size: 14px;
color: #6e8efb;
cursor: pointer;
text-decoration: underline;
}

</style>
</head>
<body>

<div class="container">
<h2 id="formTitle">Login</h2>

<!-- Login Form (Visible by default) -->
<form id="loginForm" method="POST" action="LoginServlet">
```

```
<input type="email" id="loginEmail" placeholder="Email" required>
<input type="password" id="loginPassword" placeholder="Password" required>
<button type="button" onclick="loginUser()">Login</button>
</form>

<!-- Registration Form (hidden by default) -->
<form id="registrationForm" style="display: none;">
<input type="text" id="name" placeholder="Full Name" required>
<input type="email" id="email" placeholder="Email" required>
<input type="password" id="password" placeholder="Password" required>
<button type="button" onclick="sendOTP()">Send OTP</button>
<input type="text" id="otp" placeholder="Enter OTP" disabled required>
<button type="button" id="verifyButton" onclick="registerUser()" class="disabled" disabled>Register</button>
</form>

<p id="switchFormLink" class="switch-form" onclick="switchForm()">Don't have an account? Register here</p>
</div>

<script>
// Switch between login and registration forms
function switchForm() {
var registrationForm = document.getElementById("registrationForm");
var loginForm = document.getElementById("loginForm");
var formTitle = document.getElementById("formTitle");
```

```
var switchFormLink = document.getElementById("switchFormLink");


if (loginForm.style.display === "none") {
registrationForm.style.display = "none";
loginForm.style.display = "block";
formTitle.textContent = "Login";
switchFormLink.textContent = "Don't have an account? Register here";
} else {
registrationForm.style.display = "block";
loginForm.style.display = "none";
formTitle.textContent = "Register";
switchFormLink.textContent = "Already registered? Login here";
}
}


// Send OTP to the email address
function sendOTP() {
let email = document.getElementById("email").value;
if (email === "") {
alert("Enter your email!");
return;
}


fetch("SendOTPServlet", {
method: "POST",
headers: { "Content-Type": "application/x-www-form-urlencoded" },
```

```
body: "email=" + email
})
.then(response => response.text())
.then(data => {
alert(data);
if (data.includes("OTP sent")) {
document.getElementById("otp").disabled = false;
document.getElementById("otp").style.background = "#fff";
document.getElementById("otp").style.cursor = "text";
document.getElementById("verifyButton").classList.remove("disabled");
document.getElementById("verifyButton").disabled = false;
}
});
}

// Register user with OTP
function registerUser() {
let name = document.getElementById("name").value;
let email = document.getElementById("email").value;
let password = document.getElementById("password").value;
let otp = document.getElementById("otp").value;

fetch("RegisterServlet", {
method: "POST",
headers: { "Content-Type": "application/x-www-form-urlencoded" },
body: `name=${name}&email=${email}&password=${password}&otp=${otp}`
```

```
})
.then(response => response.text())
.then(data => {
alert(data);
});
}

// Login user with email and password
function loginUser() {
let email = document.getElementById("loginEmail").value;
let password = document.getElementById("loginPassword").value;

fetch("LoginServlet", {
method: "POST",
headers: { "Content-Type": "application/x-www-form-urlencoded" },
body: `email=${email}&password=${password}`
})
.then(response => response.text())
.then(data => {
if (data === "Invalid email or password.") {
alert(data);  // Show login failure message
} else {
window.location.href = "menu.jsp";  // Redirect to home page on success
}
})
.catch(error => {
```

```
console.error("Error during login:", error);
alert("An error occurred while trying to log in. Please try again later.");
});
}
</script>
</body>
</html>
```

2)SendOTPServlet.java:

```
import java.io.IOException;
import java.io.PrintWriter;
import java.util.Properties;
import java.util.Random;
import javax.mail.*;
import javax.mail.internet.*;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

@WebServlet("/SendOTPServlet")
public class SendOTPServlet extends HttpServlet {
public static String otpCode;

protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
```

```
String recipientEmail = request.getParameter("email");
otpCode = generateOTP();

final String senderEmail = "shubhamthakurr31@gmail.com";
final String senderPassword = "kyei retu qjoj qlag"; // Use App Password

Properties props = new Properties();
props.put("mail.smtp.host", "smtp.gmail.com");
props.put("mail.smtp.port", "587");
props.put("mail.smtp.auth", "true");
props.put("mail.smtp.starttls.enable", "true");

Session session = Session.getInstance(props, new Authenticator() {
protected PasswordAuthentication getPasswordAuthentication() {
return new PasswordAuthentication(senderEmail, senderPassword);
}
});
try {
Message message = new MimeMessage(session);
message.setFrom(new InternetAddress(senderEmail));
message.setRecipients(Message.RecipientType.TO, InternetAddress.parse(recipientEmail));
message.setSubject("Your OTP Code");
message.setText("Your OTP for registration is: " + otpCode);
Transport.send(message);

PrintWriter out = response.getWriter();
```

```
out.print("OTP sent successfully to " + recipientEmail);
} catch (MessagingException e) {
e.printStackTrace();
}
}
private String generateOTP() {
return String.valueOf(100000 + new Random().nextInt(900000));
}
}
```

3) menu.jsp:

```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8" %>
<%@ page import="java.sql.*" %>

<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<title>Menu - Food Delivery</title>
<style>
/* General Styles */
body {
font-family: Arial, sans-serif;
margin: 0;
padding: 0;
background-color: #f8f8f8;
```

```
text-align: center;
}
h1 {
margin: 20px;
color: #333;
background-color: orangered;
}

/* Navigation Bar */
.cart-button {
margin-bottom: 20px;
text-align: right;
}

.cart-button nav {
display: flex;
justify-content: flex-end;
align-items: center;
background-color: #333;
padding: 10px 20px;
border-radius: 5px;
}

.cart-button nav a {
color: white;
text-decoration: none;
```

```
font-size: 16px;
font-weight: bold;
margin: 0 15px;
transition: color 0.3s ease-in-out;
}
.cart-button nav a:hover {
color: #f8b400;
}
/* Responsive Grid Layout for Menu Items */
.menu-items {
display: grid;
grid-template-columns: repeat(auto-fit, minmax(250px, 1fr)); /* Auto adjust grid */
gap: 20px;
padding: 20px;
max-width: 1200px;
margin: 0 auto;
}

.item {
background-color: white;
border-radius: 10px;
padding: 15px;
box-shadow: 0px 4px 8px rgba(0, 0, 0, 0.1);
text-align: center;
transition: transform 0.2s ease-in-out;
}
```

```
.item:hover {
transform: scale(1.05);
}
.item img {
width: 100%;
max-height: 150px;
object-fit: cover;
border-radius: 10px;
}
.item h3 {
color: #333;
font-size: 20px;
margin: 10px 0;
}
.item p {
color: #555;
font-size: 18px;
font-weight: bold;
}

.item input {
width: 50px;
padding: 5px;
margin-top: 5px;
text-align: center;
}
```

```
.item button {
background-color: #ff6600;
color: white;
border: none;
padding: 8px 15px;
margin-top: 10px;
cursor: pointer;
border-radius: 5px;
font-size: 14px;
transition: background 0.3s ease;
}

.item button:hover {
background-color: #cc5500;
}

/* Search Bar */
.search-bar {
margin: 20px;
}
.search-bar input {
width: 60%;
padding: 10px;
font-size: 16px;
border: 2px solid #333;
```

```
border-radius: 5px;
}
</style>
</head>
<body>
<header>
<h1>Our Menu</h1>
</header>

<div class="cart-button">
<nav>
<a href="cart.html">Cart</a>
<a href="logouts.jsp">Logout</a>
<a href="aboutus.html">about-us</a>
<a href="contact.html">contact-us</a>
</nav>
</div>
<div class="search-bar">
<input type="text" id="searchInput" placeholder="Search for food..." onkeyup="filterMenu()">
</div>
<div class="menu-items" id="menuItems">
<%-- Static Menu Items --%>
<div class="item" data-name="pizza">
<img src="pizza.jpg" alt="Pizza">
<h3>Pizza</h3>
<p>₹120.00</p>
```

<input type="number" id="pizzaQty" value="1" min="1" max="10">

<button onclick="addToCart('Pizza', 120.00, document.getElementById('pizzaQty').value)">Add to Cart</button>

<img src="burger.jpg" alt="Burger">

<h3>Burger</h3>

₹50.00

<input type="number" id="burgerQty" value="1" min="1" max="10">

<button onclick="addToCart('Burger', 50.00, document.getElementById('burgerQty').value)">Add to Cart</button>

<img src="pasta.jpg" alt="Pasta">

<h3>Pasta</h3>

₹80.00

<input type="number" id="pastaQty" value="1" min="1" max="10">

<button onclick="addToCart('Pasta', 80.00, document.getElementById('pastaQty').value)">Add to Cart</button>

<%

Connection con = null;

Statement stmt = null;

ResultSet rs = null;

try {

Class.forName("org.apache.derby.jdbc.ClientDriver");

con = DriverManager.getConnection("jdbc:derby://localhost:1527/fooddelivery","app","app");

```
stmt = con.createStatement();

rs = stmt.executeQuery("SELECT * FROM menu");

while (rs.next()) {

String name = rs.getString("food_name");

String image = rs.getString("food_image");

double price = rs.getDouble("price");

%>

<div class="item" data-name="<%= name.toLowerCase() %>">

<img src="<%= image %>" alt="<%= name %>">

<h3><%= name %></h3>

<p>₹<%= price %></p>

<input type="number" id="<%= name.toLowerCase() %>Qty" value="1" min="1" max="10">

<button onclick="addToCart('<%= name %>', <%= price %>, document.getElementById('<%= name.toLowerCase() %>Qty').value)">Add to Cart</button>

</div>

<%

}

} catch (Exception e) {

e.printStackTrace();

} finally {

if (rs != null) rs.close();

if (stmt != null) stmt.close();

if (con != null) con.close();

}

%>

</div>
```

```
<script>
function addToCart(itemName, itemPrice, quantity) {
let cart = JSON.parse(localStorage.getItem('cart')) || [];
const item = { name: itemName, price: itemPrice, quantity: parseInt(quantity) };
const existingItemIndex = cart.findIndex(cartItem => cartItem.name === itemName);
if (existingItemIndex > -1) {
cart[existingItemIndex].quantity += item.quantity;
} else {
cart.push(item);
}
localStorage.setItem('cart', JSON.stringify(cart));
alert(`${itemName} (x${quantity}) has been added to your cart!`);
}
function filterMenu() {
const input = document.getElementById('searchInput').value.toLowerCase();
const items = document.querySelectorAll('.menu-items .item');
items.forEach(item => {
const name = item.getAttribute('data-name');
item.style.display = name.toLowerCase().includes(input) ? '' : 'none';
});
}
</script>
</body>
</html>
```

4) About us.html:

```
<!DOCTYPE html>
```

```
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>About Us - Food Delivery</title>
    <link rel="stylesheet" href="newcss2.css">
    <link rel="stylesheet" href="video.css">
    <!-- Add Font Awesome for social icons -->
    <link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/6.0.0-beta3/css/all.min.css">
</head>
<body>
    <header>
        <h1>About Us</h1>
    </header>
    <section class="about-us">
        <div class="about-content">
            <h2>Who We Are</h2>
            <p>
                We are a team passionate about delivering delicious meals to your doorstep quickly and efficiently.
                From hearty breakfasts to late-night cravings, we cater to your taste buds with a wide range of mouth-watering options.
            </p>
        </div>
        <div class="team-image">
            <img src="about.jpg" alt="Our Team" />
```

```
        </div>
    </section>
    <section class="about-video">
        <h2>Get to Know Us Better</h2>
        <div class="video-container">
            <video controls>
                <source src="video1.mp4" type="video/mp4">
                <source src="video2.mp4" type="video/mp4">
                Your browser does not support the video tag.
            </video>
            <video controls>
                <source src="video2.mp4" type="video/mp4">
                Your browser does not support the video tag.
            </video>
        </div>
    </section>
    <footer>
        <p>&copy; 2025 FoodExpress. All rights reserved.</p>
        <div class="social-icons">
            <!-- Font Awesome social media icons -->
            <a href="#" class="fab fa-facebook"></a>
            <a href="#" class="fab fa-twitter"></a>
            <a href="#" class="fab fa-instagram"></a>
        </div>
    </footer>
</body>
```

```
</html>
```

5) Contact.jsp:(Feedback)

```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<%@page import="java.sql.*"%>
<%
    String name = request.getParameter("t1");
    String email = request.getParameter("t2");
    String message = request.getParameter("t3");
    try {
        // Load the JDBC driver
        Class.forName("org.apache.derby.jdbc.ClientDriver");
        // Establish a database connection
        Connection con =
DriverManager.getConnection("jdbc:derby://localhost:1527/fooddelivery","app","app");

        // Prepare the SQL query
        PreparedStatement pst = con.prepareStatement("INSERT INTO contact_us (name, email,
message) VALUES (?, ?, ?)");
        pst.setString(1, name);
        pst.setString(2, email);
        pst.setString(3, message);
        // Execute the query
        pst.executeUpdate();
        // Close the connection
        pst.close();
        con.close();
```

```
        // Display success message
        out.println("<script>alert('Thank you for reaching out! We will get back to you soon.');
window.location.href='contact.html';</script>");
    } catch (Exception e) {
        out.println("Error: " + e.getMessage());
    }
%>
```

6) cart:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Cart - FoodExpress</title>
    <link rel="stylesheet" href="cart.css">
</head>
<body>
    <header>
        <nav>
            <h1>FoodExpress</h1>
            <ul>
                <li><a href="index.html">Home</a></li>
                <li><a href="menu.jsp">Menu</a></li>
                <li><a href="cart.html" class="active">Cart</a></li>
                <li><a href="aboutus.html">About</a></li>
                <li><a href="contact.html">Contact</a></li>
```

```
        </ul>
    </nav>
</header>
<main>
    <section class="cart">
        <h2>Your Cart</h2>
        <table>
            <thead>
                <tr>
                    <th>Item</th>
                    <th>Price</th>
                    <th>Quantity</th>
                    <th>Total</th>
                    <th>Action</th>
                </tr>
            </thead>
            <tbody id="cart-items">
                <!-- Cart items will be dynamically populated here -->
            </tbody>
        </table>
        <div class="cart-summary">
            <p>Subtotal: <span id="subtotal">₹0.00</span></p>
            <a href="checkout.html" class="checkout-btn">Proceed to Checkout</a>
            <a href="menu.jsp" class="add-more-btn">Add More Items</a>
        </div>
    </section>
```

```
</main>
<footer>
    <p>&copy; 2025 FoodExpress. All rights reserved.</p>
</footer>
<script>
    document.addEventListener('DOMContentLoaded', function () {
        displayCart();
        const checkoutBtn = document.querySelector('.checkout-btn');
        // Add event listener for "Proceed to Checkout" button
        checkoutBtn.addEventListener('click', function (event) {
            const cart = JSON.parse(localStorage.getItem('cart')) || [];

            if (cart.length === 0) {
                // Prevent redirection and show message
                event.preventDefault();
                alert('Your cart is empty. Add items before proceeding to checkout.');
            }
        });
    });
    // Function to display items in the cart
    function displayCart() {
        const cart = JSON.parse(localStorage.getItem('cart')) || [];
        const cartItemsContainer = document.getElementById('cart-items');
        const subtotalContainer = document.getElementById('subtotal');
        let subtotal = 0;
        cartItemsContainer.innerHTML = ''; // Clear previous cart items
```

```
        if (cart.length === 0) {
            cartItemsContainer.innerHTML = '<tr><td colspan="5">Your cart is empty.</td></tr>';
            return;
        }
        cart.forEach((item, index) => {
            const row = document.createElement('tr');
            const totalPrice = item.price * item.quantity;
            row.innerHTML = `
                <td>${item.name}</td>
                <td>₹${item.price.toFixed(2)}</td>
                <td>${item.quantity}</td>
                <td>₹${totalPrice.toFixed(2)}</td>
                <td>
                    <input type="number" min="1" max="${item.quantity}" value="1" id="remove-quantity-${index}" class="remove-quantity-input">
                    <button class="remove-btn" onclick="removeItem(${index})">Remove</button>
                </td>
            `;
            cartItemsContainer.appendChild(row);
            subtotal += totalPrice;
        });

        // Update subtotal
        subtotalContainer.textContent = `₹${subtotal.toFixed(2)}`;
    }
    // Function to remove a specified number of items from the cart
```

```
        function removeItem(index) {
            const cart = JSON.parse(localStorage.getItem('cart')) || [];
            const removeQuantityInput = document.getElementById(`remove-quantity-${index}`);
            const removeQuantity = parseInt(removeQuantityInput.value, 10);
            if (removeQuantity <= 0 || isNaN(removeQuantity)) {
                alert('Please enter a valid number of items to remove.');
                return;
            }
            if (removeQuantity >= cart[index].quantity) {
                // Remove the entire item if removeQuantity is greater than or equal to current quantity
                cart.splice(index, 1);
            } else {
                // Reduce the quantity of the item
                cart[index].quantity -= removeQuantity;
            }
            localStorage.setItem('cart', JSON.stringify(cart));
            displayCart();
        }
    </script>
</body>
</html>
```

7) checkout.html:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
```

```
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Checkout - FoodExpress</title>
    <link rel="stylesheet" href="checkout.css">
</head>
<body>
    <header>
        <nav>
            <h1>FoodExpress</h1>
            <ul>
                <li><a href="index.html">Home</a></li>
                <li><a href="menu.jsp">Menu</a></li>
                <li><a href="cart.html">Cart</a></li>
                <li><a href="aboutus.html">About</a></li>
                <li><a href="contact.html">Contact</a></li>
            </ul>
        </nav>
    </header>
    <main>
        <section class="checkout">
            <h2>Checkout</h2>
            <div id="order-summary">
                <!-- Order summary will be populated here -->
            </div>
            <h3>Payment Amount: <span id="payment-amount">₹0.00</span></h3>
            <form action="pending_orders.jsp" method="POST" id="checkout-form">
                <h3>Billing Information</h3>
```

```
            <label for="name">Name:</label>
            <input type="text" id="name" name="name" required>
            <label for="email">Email:</label>
            <input type="email" id="email" name="email" required>
            <label for="address">Address:</label>
            <input type="text" id="address" name="address" required>

            <h3>Payment Information</h3>

            <label for="payment-method">Select Payment Method:</label>
            <select id="payment-method" name="payment-method" required>
                <option value="" disabled selected>Select Payment Mode</option>
                <option value="cod">Offline</option>
                <option value="paypal">Online</option>
            </select>  <br><br>
            <button type="submit">Submit Order</button>
        </form>
    </section>
</main>
<footer>
    <p>&copy; 2024 FoodExpress. All rights reserved.</p>
</footer>
<script>
  document.addEventListener('DOMContentLoaded', function () {
        const cartItems = JSON.parse(localStorage.getItem('cart')) || [];
        const orderSummary = document.getElementById('order-summary');
```

```
        const paymentAmount = document.getElementById('payment-amount');
        let total = 0;
        // Display order summary and calculate total
        if (cartItems.length > 0) {
            cartItems.forEach(item => {
                const itemElement = document.createElement('div');
                itemElement.classList.add('order-item');
                itemElement.innerHTML = `
                    <p>${item.name} (x${item.quantity}) - ₹${(item.price *
item.quantity).toFixed(2)}</p>
                `;
                orderSummary.appendChild(itemElement);

                // Update total
                total += item.price * item.quantity;
            });
        } else {
            orderSummary.innerHTML = '<p>Your cart is empty.</p>';
        }
        // Update the payment amount display
        paymentAmount.textContent = `₹${total.toFixed(2)}`;
    });
    document.getElementById('checkout-form').onsubmit = function (event) {
        event.preventDefault();
        const paymentMethod = document.getElementById('payment-method').value;
        if (paymentMethod === 'paypal') {
```

```
            // Redirect to the online payment page for PayPal
            window.location.href = 'onlinepayment.html';
            return;
        }
        const name = document.getElementById('name').value;
        const email = document.getElementById('email').value;
        const address = document.getElementById('address').value;
        const paymentAmount = document.getElementById('payment-amount').textContent;
        const orderId = Math.floor(Math.random() * 100000);
        const orderDetails = {
            orderId: orderId,
            items: JSON.parse(localStorage.getItem('cart')),
            totalAmount: paymentAmount,
            deliveryTime: '45 minutes'
        };
        sessionStorage.setItem('orderDetails', JSON.stringify(orderDetails));
        // Clear cart and redirect to confirmation page
        localStorage.removeItem('cart');
        window.location.href = 'order-confirmation.html';
    };
  </script>
</body>
</html>
```

8) home.html:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Food Delivery Website</title>
    <link rel="stylesheet" href="newcss.css">
</head>
    <header>
        <nav>
            <h1>FoodExpress</h1>
            <ul>
                <li><a href="logreg.html">Register</a></li>
                <li><a href="adminlogin.html">Admin Login</a></li>
            </ul>
        </nav>
    </header>
    <section id="home">
        <h2>Welcome to FoodExpress</h2>
        <p>Delivering delicious meals at your doorstep.</p>
        <a href="logreg.html" class="button">Start Your Food Journy.</a>
    </section>
    <footer>
        <p>&copy; 2025 FoodExpress. All rights reserved.</p>
    </footer>
```

```
    <script src="script.js">
    </script>
</body>
</html>
```

Admin side:

9) adminlogin.html:

```
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title> Admin Login</title>
    <link rel="stylesheet" href="admin.css">
</head>
<body>
    <h2>Admin Login</h2>
    <form action="admin_loginform.jsp" method="post">
        Enter username:<input type="text" name="t1"><br>
        Enter password:<input type="text" name="t2"><br>
        <input type="submit" name="b1" value="Login">
    </form>
</body>
</html>
```

10) addfood.jsp:

```
<%@page import="java.sql.*"%>
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<script>
var openFile = function(event) {
    var input = event.target;
    var reader = new FileReader();
    reader.onload = function() {
        var dataURL = reader.result;
        var output = document.getElementById('img1');
        output.src = dataURL;
        document.getElementById("bd1").value = dataURL;
    };
    reader.readAsDataURL(input.files[0]);
};
</script>
<head>
    <link rel="stylesheet" href="addfood.css">
</head>
<body>
    <div class="header">
        <a href="index.html" class="logo">FoodExpress</a>
        <div class="header-right">
            <a href="admin_feedback.jsp">Customer Feedback</a>
            <a class="active" href="add_food.jsp">Add Food</a>
            <a href="admin_logout.jsp">Logout</a>
```

```
        </div>
    </div>
    <div class="topnav">
        <a class="active" href="add_food.jsp">Add Food</a>
        <a href="admin_update.jsp">Update Food</a>
        <a href="admin_view.jsp">View Menu</a>
        <a href="pending_orders.jsp">Pending Orders</a>
        <ul style="position:relative; padding:10px; color:#fefefe"> Welcome,
<%=session.getAttribute("a")%></ul>
    </div>
    <div class="update_p">
        <h2 align="center">Add Food Item</h2>
        <form action="add_food_server.jsp" method="POST">
            <table border="0" align="center">
                <tr>
                    <td><b>Enter Food ID:</b></td>
                    <td><input type="number" name="food_id" id="food_id" required></td>
                </tr>
                <tr>
                    <td><b>Food Name:</b></td>
                    <td><input type="text" name="food_name" id="food_name" required></td>
                </tr>
                <tr>
                    <td><b>Price:</b></td>
                    <td><input type="number" name="price" id="price" required></td>
```

```
            </tr>
            <tr>
                <td><b>Food Image:</b></td>
                <td><input type="file" name="food_image" id="food_image"
onchange="openFile(event)"></td>
            </tr>
            <tr>
                <td colspan="4" align="center">
                    <input type="submit" value="Add Food" style="background-color: #6f8c6f;color:
white;padding: 12px 20px;margin: 20px 0;border:none;cursor: pointer;width: 40%;">
                </td>
            </tr>
        </table>
        <img src="" id="img1" width="100" height="100"
style="position:absolute;top:900px;right:150px">
        <input type="hidden" name="bd" id="bd1">
      </form>
  </div>
</body>
</html>
```

11) add_food_server.jsp:

```
<%@page import="java.sql.*"%>
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<%
   int food_id = Integer.parseInt(request.getParameter("food_id"));
   String food_name = request.getParameter("food_name");
   double price = Double.parseDouble(request.getParameter("price"));
```

```
    String food_image = request.getParameter("food_image");  // Image in base64 format
    Connection con = null;
    PreparedStatement ps = null;
    try {
        Class.forName("org.apache.derby.jdbc.ClientDriver");
        con = DriverManager.getConnection("jdbc:derby://localhost:1527/fooddelivery","app","app");
       String sql = "INSERT INTO menu(food_id, food_name, price, food_image) VALUES ( ?, ?, ?,
?)";
       ps = con.prepareStatement(sql);
       ps.setInt(1, food_id);
       ps.setString(2, food_name);
       ps.setDouble(3, price);
       ps.setString(4, food_image);
       int result = ps.executeUpdate();
       if (result > 0) {
          response.sendRedirect("admin_view.jsp");
       } else {
          out.println("<script>alert('Failed to add food item.'); window.history.back();</script>");
       }
    } catch (Exception e) {
       e.printStackTrace();
    } finally {
       if (ps != null) ps.close();
       if (con != null) con.close();
    }
%>
```

12) admin_update.jsp:

```
<%@page import="java.sql.*"%>
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<html>
<head>
    <title>Update Food Item</title>
    <link rel="stylesheet" href="update_food.css">
    <script>
        // Function to preview image before updating
        function previewImage(event) {
            var input = event.target;
            var reader = new FileReader();
            reader.onload = function () {
                var dataURL = reader.result;
                document.getElementById("imgPreview").src = dataURL;
                document.getElementById("hiddenImage").value = dataURL;
            };
            reader.readAsDataURL(input.files[0]);
        }
        // Function to fetch existing food item details
        function fetchFoodDetails() {
            var foodId = document.getElementById("food_id").value;
            if (foodId === "") {
                alert("Please enter a Food ID.");
                return;
            }
```

```
            var xhr = new XMLHttpRequest();
            xhr.open("GET", "fetch_food_details.jsp?food_id=" + foodId, true);
            xhr.onreadystatechange = function () {
                if (xhr.readyState == 4 && xhr.status == 200) {
                    var response = JSON.parse(xhr.responseText);
                    if (response.status === "success") {
                        document.getElementById("food_name").value = response.food_name;
                        document.getElementById("price").value = response.price;
                        document.getElementById("imgPreview").src = response.food_image;
                        document.getElementById("hiddenImage").value = response.food_image;
                    } else {
                        alert("Food ID not found!");
                    }
                }
            };
            xhr.send();
        }
    </script>
</head>
<body>
    <div class="header">
        <a href="index.html" class="logo">FoodExpress</a>
        <div class="header-right">
            <a href="admin_feedback.jsp">Customer Feedback</a>
            <a href="add_food.jsp">Add Food</a>
            <a class="active" href="admin_update.jsp">Update Food</a>
```

```
            <a href="admin_logout.jsp">Logout</a>
        </div>
    </div>
    <div class="container">
        <h2>Update Food Item</h2>
        <form action="update_food_server.jsp" method="POST">
            <div class="input-group">
                <label>Enter Food ID:</label>
                <input type="number" name="food_id" id="food_id" required>
                <button type="button" onclick="fetchFoodDetails()">Search</button>
            </div>
            <div class="input-group">
                <label>Food Name:</label>
                <input type="text" name="food_name" id="food_name" required>
            </div>
            <div class="input-group">
                <label>Price:</label>
                <input type="number" name="price" id="price" required>
            </div>
            <div class="input-group">
                <label>Food Image:</label>
                <input type="file" name="food_image" id="food_image"
onchange="previewImage(event)">
            </div>
            <div class="preview">
                <img src="" id="imgPreview" width="120" height="120">
```

```
            </div>
            <input type="hidden" name="hiddenImage" id="hiddenImage">
            <button type="submit" class="update-btn">Update Food</button>
        </form>
    </div>
</body>
</html>
```

13) update_food_server.jsp:

```
<%@page import="java.sql.*"%>
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<html>
<head>
    <title>Update Food - Server</title>
    <style>
        body {
            font-family: Arial, sans-serif;
            background: linear-gradient(to right, #ff9966, #ff5e62);
            margin: 0;
            padding: 0;
            display: flex;
            justify-content: center;
            align-items: center;
            height: 100vh;
        }
        .container {
```

```
    background: white;
    padding: 20px;
    width: 50%;
    text-align: center;
    border-radius: 10px;
    box-shadow: 0 5px 15px rgba(0, 0, 0, 0.2);
    animation: fadeIn 0.8s ease-in-out;
}
h2 {
    color: #333;
}
.status-box {
    margin: 20px 0;
    padding: 15px;
    border-radius: 5px;
    font-size: 16px;
}
.success {
    background-color: #28a745;
    color: white;
    padding: 10px;
    border-radius: 5px;
    animation: slideIn 0.5s ease-in-out;
}
```

```
.error {
    background-color: #dc3545;
    color: white;
    padding: 10px;
    border-radius: 5px;
    animation: slideIn 0.5s ease-in-out;
}
.back-btn {
    display: inline-block;
    margin-top: 15px;
    padding: 10px 20px;
    background-color: #007bff;
    color: white;
    text-decoration: none;
    border-radius: 5px;
    transition: background 0.3s ease;
}
.back-btn:hover {
    background-color: #0056b3;
}
@keyframes fadeIn {
    from { opacity: 0; transform: scale(0.9); }
    to { opacity: 1; transform: scale(1); }
}
```

```
        @keyframes slideIn {
            from { transform: translateY(-20px); opacity: 0; }
            to { transform: translateY(0); opacity: 1; }
        }
    </style>
</head>
<body>
    <div class="container">
        <h2>Update Food Status</h2>
        <div class="status-box">
            <%
                Connection con = null;
                PreparedStatement ps = null;
                String foodId = request.getParameter("food_id");
                String foodName = request.getParameter("food_name");
                String price = request.getParameter("price");
                String foodImage = request.getParameter("food_image");
                try {
                    Class.forName("org.apache.derby.jdbc.ClientDriver");
                    con = DriverManager.getConnection("jdbc:derby://localhost:1527/fooddelivery",
"app", "app");

                    String query = "UPDATE menu SET food_name=?, price=?, food_image=?
WHERE food_id=?";
                    ps = con.prepareStatement(query);
```

```
            ps.setString(1, foodName);
            ps.setDouble(2, Double.parseDouble(price));
            ps.setString(3, foodImage);
            ps.setInt(4, Integer.parseInt(foodId));
            int updated = ps.executeUpdate();
            if (updated > 0) {
%>
                <div class="success">
                    ✅ Food item updated successfully!
                </div>
<%
            } else {
%>
                <div class="error">
                    ❌ Update failed! Food ID not found.
                </div>
<%
            }
        } catch (Exception e) {
%>
            <div class="error">
                ❌ Error: <%= e.getMessage() %>
            </div>
<%
```

```
            } finally {
                if (ps != null) ps.close();
                if (con != null) con.close();
            }
        %>
    </div>
    <a href="admin_update.jsp" class="back-btn">Back to Update Page</a>
  </div>
</body>
</html>
```

14) admin_view.jsp:

```
<%@ page import="java.sql.*" %>
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<html>
<head>
  <title>Admin View - Food Items</title>
  <style>
    body { font-family: Arial, sans-serif; margin: 20px; }
    table { width: 100%; border-collapse: collapse; margin-top: 20px; }
    th, td { border: 1px solid black; padding: 10px; text-align: left; }
    th { background-color: #f2f2f2; }
    img { max-width: 100px; max-height: 100px; border-radius: 5px; }
    .delete-btn {
      background-color: red;
      color: white;
```

```
            padding: 5px 10px;
            border: none;
            border-radius: 5px;
            cursor: pointer;
        }
        .delete-btn:hover {
            background-color: darkred;
        }
    </style>
</head>
<body>
    <h2>Admin View - Food Items</h2>
    <table>
        <tr>
            <th>ID</th>
            <th>Food Name</th>
            <th>Price</th>
            <th>Food Image</th>
            <th>Action</th>
        </tr>
        <%
        Connection con = null;
        PreparedStatement ps = null;
        ResultSet rs = null;
        try {
            Class.forName("org.apache.derby.jdbc.ClientDriver");
```

```
            con = DriverManager.getConnection("jdbc:derby://localhost:1527/fooddelivery", "app",
"app");
            String query = "SELECT * FROM menu";
            ps = con.prepareStatement(query);
            rs = ps.executeQuery();
            while (rs.next()) {
        %>
            <tr>
                <td><%= rs.getInt("food_id") %></td>
                <td><%= rs.getString("food_name") %></td>
                <td>₹<%= rs.getDouble("price") %></td>
                <td>
                    <img src="<%= rs.getString("food_image") %>" alt="Food Image">
                </td>
                <td>
                    <form action="delete_food.jsp" method="POST" onsubmit="return confirm('Are you
sure you want to delete this item?');">
                        <input type="hidden" name="food_id" value="<%= rs.getInt("food_id") %>">
                        <button type="submit" class="delete-btn">Remove</button>
                    </form>
                </td>
            </tr>
        <%
            }
        } catch (Exception e) {
            out.println("<p style='color:red;'>Error: " + e.getMessage() + "</p>");
```

```
        } finally {
            if (rs != null) rs.close();
            if (ps != null) ps.close();
            if (con != null) con.close();
        }
        %>
    </table><br>
    <center><a href="add_food.jsp">Back to Admin Page</a></center>
</body>
</html>
```

15)admin_feedback.jsp:

```
<%@page import="java.sql.*"%>
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<html>
<head>
    <title>Customer Feedback</title>
    <style>
        body {
            font-family: Arial, sans-serif;
            background: linear-gradient(to right, #6f8c6f, #4b6d4b); /* Matching add_food.jsp */
            margin: 0;
            padding: 0;
            display: flex;
            flex-direction: column;
            align-items: center;
```

```
}
h2 {
    margin-top: 20px;
    color: white;
    text-shadow: 2px 2px 5px rgba(0, 0, 0, 0.2);
    font-size: 30px;
    animation: fadeIn 1s ease-in-out;
}
.feedback-container {
    display: flex;
    flex-wrap: wrap;
    justify-content: center;
    margin-top: 20px;
    width: 80%;
}
.feedback-card {
    background: #fefefe; /* Light background for contrast */
    padding: 15px;
    width: 320px;
    margin: 15px;
    border-radius: 12px;
    box-shadow: 0 5px 15px rgba(0, 0, 0, 0.2);
    transition: transform 0.3s ease, box-shadow 0.3s ease;
    animation: slideIn 0.8s ease-in-out;
    position: relative;
    overflow: hidden;
```

```
}
.feedback-card:hover {
    transform: translateY(-5px);
    box-shadow: 0 8px 20px rgba(0, 0, 0, 0.3);
}
.feedback-header {
    font-size: 18px;
    font-weight: bold;
    color: #2e472e; /* Darker shade from add_food.jsp */
}

.feedback-email {
    font-size: 14px;
    color: #3e5b3e; /* Muted green */
    font-style: italic;
    margin-top: 5px;
}
.feedback-message {
    font-size: 16px;
    color: #333;
    margin: 15px 0;
}
.refresh-btn {
    background: #6f8c6f; /* Matching primary button color */
    color: white;
    padding: 12px 20px;
```

```
            border: none;
            cursor: pointer;
            font-size: 16px;
            border-radius: 5px;
            transition: background 0.3s ease, transform 0.2s ease;
            margin-top: 20px;
        }
        .refresh-btn:hover {
            background: #4b6d4b; /* Darker green for hover */
            transform: scale(1.05);
        }

        @keyframes fadeIn {
            from { opacity: 0; transform: scale(0.9); }
            to { opacity: 1; transform: scale(1); }
        }
        @keyframes slideIn {
            from { transform: translateY(20px); opacity: 0; }
            to { transform: translateY(0); opacity: 1; }
        }
    </style>
</head>
<body>
    <h2>Customer Feedback</h2>
    <div class="feedback-container">
        <%
```

```
        Connection con = null;
        PreparedStatement ps = null;
        ResultSet rs = null;
        try {
            Class.forName("org.apache.derby.jdbc.ClientDriver");
            con = DriverManager.getConnection("jdbc:derby://localhost:1527/fooddelivery", "app",
"app");

            String query = "SELECT * FROM Contact_us ORDER BY name ASC";
            ps = con.prepareStatement(query);
            rs = ps.executeQuery();

            while (rs.next()) {
    %>
        <div class="feedback-card">
            <div class="feedback-header">
                👴 <%= rs.getString("name") %>
            </div>
            <div class="feedback-email">
                ✉ <%= rs.getString("email") %>
            </div>
            <div class="feedback-message">
                "<%= rs.getString("message") %>"
            </div>
        </div>
    <%
```

```
            }
        } catch (Exception e) {
            out.println("<p style='color: red;'>Error: " + e.getMessage() + "</p>");
        } finally {
            if (rs != null) rs.close();
            if (ps != null) ps.close();
            if (con != null) con.close();
        }
    %>
  </div>

  <button class="refresh-btn" onclick="location.reload();">🔄 Refresh Feedback</button>
</body>
</html>
```

16) admin_logout.jsp:

```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<html>
 <body>
 <%
 session.invalidate();
 response.sendRedirect("adminlogin.html");
 %>
 </body>
</html>
```

17) admin_loginform.jsp

```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
    <title>JSP Page</title>
</head>
<body>
<%
String user = request.getParameter("t1");
String pass = request.getParameter("t2");

if ((user != null && pass != null) &&
    ((user.equals("shubham") && pass.equals("shubham")) ||
     (user.equals("admin") && pass.equals("admin")))) {

    pageContext.setAttribute("a", user, PageContext.SESSION_SCOPE);
    response.sendRedirect("add_food.jsp");
} else {
    out.println("<script>alert('Invalid User Name or Password..');");
    out.println("window.location.assign('adminlogin.html');</script>");
}
%>
</body>
</html>
```

Chapter 6

Conclusion And Future Work:

Conclusion

The FoodExpress food delivery website successfully provides an efficient and user-friendly platform for ordering food online. It integrates essential features such as a menu system, shopping cart, checkout process, and payment options (Cash on Delivery & Online Payment). The system is built using JSP, Java, and Apache Derby, ensuring smooth database connectivity and order management.

By implementing a structured database model (PENDING_ORDERS table) and a seamless user experience, the website meets the basic requirements of a food ordering system. The project demonstrates effective use of front-end and back-end technologies to create a functional web application.

Future Work:

The FoodExpress food delivery website successfully implements core features, but there is still scope for future enhancements to improve user experience, security, and scalability. The following improvements can be considered:

1. Advanced Order Management & Tracking

- Develop an Admin Panel to manage orders, update the menu, and handle customer queries.
- Implement real-time order tracking so users can monitor their food delivery progress.

2. Enhanced Payment & Security

- Integrate online payment gateways

3. Mobile App Development

- Develop a mobile app for Android & iOS to improve accessibility.

4.Exapnd admin feature and migrate to a scalable, cloud based infrastructure

Chapter 7

Reference:

1. Guidance from Sir Omkar Sherkhane

2. www.tutorialspoint.com

3. www.youtube.com

4. www.w3schools.com

5. www.stackoverflow.com