



# ONLINE PIZZA DELIVERY



## INTERNSHIP PROJECT REPORT

*Submitted by*

VINOTH P

(Register No.: 95192201118)

*Third year student of*

BACHELOR OF ENGINEERING  
IN  
COMPUTER SCIENCE AND ENGINEERING

P.S.R. ENGINEERING COLLEGE, SIVAKASI – 626 140  
(An Autonomous Institution, Affiliated to Anna University, Chennai)

MARCH 2025

## BONAFIDE CERTIFICATE

Certified that this project report "ONLINE PIZZA DELIVERY " is the bonafide work of VINOTH P (95192201118), " who carried out the project work under my supervision.

SIGNATURE

Mrs.Arthi Venkatesh  
EXTERNAL SUPERVISOR  
Corporate Trainer,  
Evoria Infotech Private Limited  
Bangalore – 560076.

SIGNATURE

Mr. Mohamed Nawfal A  
TEAM LEADER  
Corporate Trainer – Head,  
Evoria Infotech Private Limited  
Bangalore – 560076.

Submitted to the department for the internship report evaluation on  
\_\_\_\_\_.

PROJECT COORDINATOR

HOD/CSE

## ACKNOWLEDGEMENT

I would like to express my sincere gratitude to the following individuals and institutions who have contributed to the successful completion of the Online Pizza Delivery project.

Evoriea Infotech Private Limited, Bengaluru:

Mrs.Arthi Venkatesh, Corporate Trainer, for providing valuable guidance, mentorship, and constructive feedback throughout the project development process. Your expertise has been instrumental in shaping the project.

P.S.R Engineering College, Sivakasi:

We unreservedly express our indebtedness to our Managing Trustee and Correspondent Thiru. R. SOLAISAMY and Director Er. S. VIGNESWARI ARUNKUMAR for providing the needed resources and facilities.

It is our privilege to convey my sincere thanks to our respected Principal Dr. J.S. SENTHIL KUMAAR M.E., Ph.D., for giving us permission to do this project in our organization.

We wish to extend our sincere thanks to our adored Head of the Department Dr. A. RAMATHILAGAM M.E., Ph.D., Professor for her motivation during this period of this internship project work.

Their contributions have significantly enhanced the overall quality and success of the "Online Pizza Delivery" project.

## ABSTRACT

The Online Pizza Delivery System is a web-based platform designed to facilitate seamless ordering, tracking, and delivery of pizzas from various restaurants to customers. The system provides a user-friendly interface that allows customers to browse menus, customize their pizzas, place orders, and make secure online payments. It leverages Spring Boot and MySQL for efficient backend operations and HTML, CSS, and JavaScript for an interactive frontend. The platform streamlines the ordering process by integrating essential features such as real-time order tracking, estimated delivery time, and multiple payment options. Customers can create accounts, save favorite orders, and receive personalized discounts and offers. Restaurant owners can manage menus, track orders, and optimize delivery routes, ensuring efficient service. The key objectives of the system are to enhance customer convenience, reduce order processing time, and improve business efficiency. By utilizing Spring Boot for backend development and MySQL for database management, the system ensures high performance, scalability, and security. Additional features like push notifications, GPS-based tracking, and AI-powered recommendations can be incorporated to enhance the user experience further. This project is ideal for businesses looking to modernize their food delivery operations and provide an efficient, technology-driven solution for pizza lovers.

## TABLE OF CONTENTS

CHAPTER No	TITLE	PAGE NO
	ABSTRACT	4
1	INTRODUCTION	7
	1.1 OVERVIEW OF ONLINE PIZZA DELIVERY	
	1.2 PURPOSE OF REPORT	
	1.3 TECHNOLOGIES USED	
2	SYSTEM ARCHITECTURE	8
	2.1 OVERVIEW OF THE SYSTEM	
	2.2 SPRING BOOT ARCHITECTURE	
	2.3 DATABASE ARCHITECTURE	
3	BACKEND DEVELOPMENT WITH SPRING BOOT	10
	3.1 SETTING UP SPRING BOOT APPLICATION	
	3.2 REST API DEVELOPMENT	
	3.3 INTEGRATING MYSQL WITH SPRING DATA JPA	
4	FRONTEND INTEGRATION	12
	4.1 USING THYMELESF OR REACT /ANGULAR	
	4.2 API COMMUNICATION BETWEEN BACKEND AND FRONT END	
5	FEATURES	13
	5.1 USER FEATURES	

## 5.2 ADMIN FEATURES

6	IMPLEMENTATION &CHALLENGES	15
7	CONCLUSION	57
8	REFERENCE	58

# CHAPTER 1

## INTRODUCTION

### 1. Introduction to Java

Java is a high-level, object-oriented programming language developed by Sun Microsystems (now owned by Oracle) in 1995. It is widely used for building platform-independent applications due to its "Write Once, Run Anywhere" (WORA) capability, enabled by the Java Virtual Machine (JVM).

#### 1.1 Overview Of Online Pizza Delivery

The ONLINE PIZZA DELIVERY SYSTEM is a web-based platform designed to streamline the process of ordering and delivering pizzas from various restaurants to customers. It allows users to browse menus, customize orders, make secure payments, and track deliveries in real-time. By leveraging modern web technologies, the system enhances efficiency, customer convenience, and business profitability.

#### 1.2 Purpose Of The Report

The purpose of this report is to document the development, implementation, and features of the ONLINE PIZZA DELIVERY SYSTEM. It provides insights into the system's architecture, backend and frontend development, user functionalities, and challenges faced during implementation.

#### 1.3 Technologies Used

- BACKEND: SPRING BOOT, JAVA
- FRONTEND: HTML, CSS, JAVASCRIPT (WITH THYMELEAF/REACT/ANGULAR)
- DATABASE: MYSQL
- ADDITIONAL TOOLS: SPRING DATA JPA, REST APIS, HIBERNATE, BOOTSTRAP

## CHAPTER 2

### SYSTEM ARCHITECTURE

#### 2.1 Overview Of The System

The system follows a THREE-TIER ARCHITECTURE consisting of a FRONTEND (UI), BACKEND (BUSINESS LOGIC), AND DATABASE (DATA STORAGE). This structure ensures scalability, maintainability, and efficient data handling.

#### 2.2 Spring Boot Architecture

##### Key Components of Spring Boot Architecture

##### Layered Architecture Of Spring Boot

Spring Boot applications are structured in a LAYERED ARCHITECTURE, which consists of the following layers

##### Presentation Layer(Controller Layer)

- Handles HTTP requests and responses.
- Uses Spring MVC to map requests to appropriate methods.
- Implements RESTful APIs using `@RestController`, `@RequestMapping`, and `@GetMapping` annotations.

##### Service Layer(Business Logic)

- Contains the core business logic and services.
- Uses `@Service` annotation to define service classes.
- Ensures separation of concerns between controllers and repositories

##### Data Access Layer(Repository Layer)

- Manages database operations using Spring Data JPA.
- Uses `@Repository` annotation for data persistence.
- Supports CRUD operations with minimal boilerplate code using `JpaRepository` or `CrudRepository`



## DataBase Layer

- Stores product details, categories, pricing, and availability.
- Uses Spring Data JPA with MySQL/PostgreSQL.
- Supports search and filtering.

## Key Components Of Spring Boot Architecture

### Spring Boot Starters

Predefined dependency configurations that help quickly bootstrap a Spring Boot project. Order Management Service

### Spring Boot Auto-Configuration

- Automatically configures Spring beans and dependencies based on the classpath and properties.
- Removes the need for complex XML configurations.

### Spring Boot Embedded Server

- Supports Tomcat, Jetty, and Undertow as embedded servers.
- No need to deploy WAR files separately; applications run as standalone JARs. Notification Service

### Spring Boot Actuator

- Provides real-time monitoring and management of the application.
- Exposes endpoints (/actuator/health, /actuator/info) to check system health and performance.

### Spring Boot Properties Configuration

- Uses application.properties or application.yml for configuration.
- Supports externalized configuration for port settings, database connections, and logging.

### Spring Security Integration

- Provides built-in authentication and authorization mechanisms.
- Supports JWT (JSON Web Token) and OAuth2 for securing APIs.

## 2.3 Database Architecture

The database is designed using MYSQL, storing user details, orders, restaurant data, and delivery statuses. It follows a RELATIONAL MODEL with tables such as:

- USERS TABLE: STORES CUSTOMER AND ADMIN INFORMATION
- ORDERS TABLE: MAINTAINS ORDER HISTORY AND STATUS
- MENU TABLE: STORES PIZZA DETAILS, PRICING, AND CUSTOMIZATION OPTIONS

## CHAPTER 3

### BACKEND DEVELOPMENT WITH SPRING BOOT

Backend development for the Online Pizza Delivery System is implemented using Spring Boot, which provides a robust and scalable framework for building enterprise-level applications. It simplifies development by offering built-in features such as Spring MVC for web handling, Spring Data JPA for database operations, Spring Security for authentication, and RESTful API support.

#### Key Components of Spring Boot Backend

The backend architecture follows the MVC (Model-View-Controller) pattern, ensuring a modular and maintainable structure.

### 3.1 Setting Up Spring Boot Application

#### Controller Layer(API EndPoints)

- Manages HTTP requests and responses.
- Uses Spring MVC to expose RESTful APIs.
- Handles user actions such as placing orders, managing accounts, and retrieving order history.

#### Service Layer(Business Logic)

- Implements the core functionality of the system.
- Contains methods for order processing, calculating prices, applying discounts, and validating user inputs.
- Ensures separation of concerns by keeping business logic separate from controllers and repositories.

#### Repository Layer(Data Access)

- Uses Spring Data JPA to interact with the database.
- Provides CRUD operations for managing users, orders, menu items, and transactions.
- Ensures optimized queries and indexing for faster data retrieval.

#### Security Layer

- Implements Spring Security to handle user authentication and authorization.
- Supports role-based access control (RBAC) to distinguish between customers, admins, and delivery personnel.

- Uses JWT (JSON Web Token) for securing API requests and user sessions.

#### DataBase Layer

- Manages data storage using MySQL.
- Defines structured relational tables for users, orders, menus, and payments.
- Ensures data consistency, normalization, and referential integrity.

#### Spring MVC(Model-View-Controller)

- Organizes the backend into controllers (handling requests), services (business logic), and repositories (data access).
- Supports REST API development for seamless frontend-backend interaction.

### 3.2 Rest API Development

#### Spring Data JPA(DataBase Management)

- Simplifies database operations through repositories and entity management.
- Reduces boilerplate code by offering built-in methods for database queries.
- Ensures efficient transaction management and data persistence.

#### RestFul API Development

- Provides stateless communication between frontend and backend using JSON-based API responses.
- Allows users to perform CRUD operations (Create, Read, Update, Delete) on orders, menu items, and user data.
- Enables smooth integration with third-party services like payment gateways and delivery tracking APIs.

### 3.3 Integrating MySql With Spring Data JPA

#### DataBase Structure

- users Table: Stores customer details (ID, name, email, password).
- orders Table: Stores order details (ID, pizza type, quantity, total price, status).
- menu Table: Contains pizza items (ID, name, ingredients, price).
- payments Table: Records payment transactions (ID, order ID, amount, status).

#### DataBase Optimization Strategies

- Indexing: Improves search queries for faster order retrieval.
- Query Optimization: Uses optimized SQL queries to reduce response time.
- Caching: Implements caching mechanisms like Redis for frequently accessed data.
- Data Backup & Recovery: Ensures periodic automated database backups to prevent data loss.

## CHAPTER 4

### FRONTEND INTEGRATION

Frontend integration plays a crucial role in creating a seamless user experience for the Online Pizza Delivery System. The frontend is responsible for displaying menus, handling user interactions, and communicating with the backend to process orders, payments, and real-time tracking. For frontend development, Spring Boot can be integrated with various frontend technologies, including Thymeleaf (for server-side rendering) and modern JavaScript frameworks like React and Angular (for a dynamic single-page application experience).

#### 4.1 Using Thymeleaf (Optional) or React/Angular for Frontend

Thymeleaf is a templating engine that integrates seamlessly with Spring Boot for generating dynamic HTML pages.

- Works well with Spring MVC for rendering dynamic data.
- Suitable for simpler applications with minimal frontend complexity.
- Reduces dependency on frontend frameworks like React/Angular.

##### *How Thymeleaf Works?*

- Backend fetches data from the database (Spring Boot + JPA).
- The data is embedded into HTML pages using Thymeleaf tags.
- The rendered HTML is sent to the client (browser).

#### 4.2 API Communication Between Backend and Frontend

The frontend communicates with the Spring Boot backend using REST APIs, which enable secure and efficient data exchange.

##### API Call Mechanism

The backend exposes REST endpoints like:

- GET /api/menu/all → Fetch all available pizzas.
- POST /api/orders/place → Place an order.
- GET /api/orders/{id} → Retrieve order details.
- POST /api/users/login → Authenticate users.

The frontend (Thymeleaf, React, or Angular) calls these APIs using:

- Fetch API (JavaScript)
- Axios (React)

## CHAPTER 5

### 5.Features

#### 5.1 User Features

Customers can use the platform to browse pizzas, customize orders, make payments, and track deliveries efficiently.

##### 5.1.1 User Registration & Login

- New customers can sign up using their email, password, and contact details.
- Secure login with password encryption and authentication.

##### 5.1.2 Browse Menu & Customize Orders

- View available pizza varieties, prices, and ingredients.
- Customize pizza size, toppings, crust type, and add-ons.

##### 5.1.3 Add to Cart & Place Orders

- Select multiple pizzas and add them to the cart.
- Modify cart items before finalizing the order.

##### 5.1.4 Online Payment Integration

- Multiple payment options: Credit/Debit Card, UPI, Net Banking, Wallets, or Cash on Delivery.
- Secure payment processing with encryption and transaction validation.

##### 5.1.5 Real-Time Order Tracking

- Track the order status: Order Placed → Processing → Out for Delivery → Delivered.
- Estimated delivery time updates based on restaurant processing.

##### 5.1.6 Order History & Reordering

- View past orders and repeat previous purchases with one click.
- Download invoice receipts for past transactions.

##### 5.1.7 Ratings & Reviews:

- Customers can rate pizzas and delivery service.
- Leave feedback on taste, delivery speed, and packaging.

##### 5.1.8 Notification & Alerts

- Receive email/SMS notifications for order confirmations and delivery updates.
- Special discounts and personalized promotions for frequent customers.

## 5.2 ADMIN FEATURES

The admin panel provides restaurant owners and managers with the necessary tools to manage orders, customers, menu items, and reports efficiently.

### 5.2.1 Admin Dashboard

- Overview of total orders, revenue, pending deliveries, and customer activity.
- Real-time sales analytics and order trends.

### 5.2.2 Menu Management

- Add, update, or remove pizzas from the menu.
- Set pricing, discounts, and limited-time offers.
- Manage availability status (in stock, out of stock).

### 5.2.3 Order Management

- View all customer orders with statuses (Pending, Processing, Out for Delivery, Delivered).
- Assign delivery personnel and update delivery times.
- Cancel or modify orders in case of customer requests.

### 5.2.4 User & Customer Management

- View registered customers and their order history.
- Block or deactivate suspicious users.
- Respond to customer complaints and queries.

### 5.2.5 Payment & Refund Handling

- Track successful payments, pending transactions, and failed payments.
- Process refunds for canceled orders.

### 5.2.6 Reports & Analytics

- Daily, weekly, and monthly sales reports for business insights.
- Track best-selling pizzas and customer preferences.
- Generate financial reports for profit calculation and tax filing.

### 5.2.7 Promotions & Discounts Management

- Create and manage promo codes and discount offers.
- Set seasonal deals and loyalty rewards for returning customers.

## CHAPTER 6

### IMPLEMENTATION & CHALLENGES

The Online Pizza Delivery System was implemented using Spring Boot for the backend, MySQL for the database, and Thymeleaf/React/Angular for the frontend. This section outlines the key implementation steps and challenges faced during development.

#### Implementation of Online Shoe Mart

The system was built using a modular approach, ensuring scalability and maintainability. The main steps in implementation were:

1. Backend: Spring Boot (REST API, Spring Data JPA, Spring Security)
2. Frontend: React, Angular, or Thymeleaf
3. Database: MySQL (Spring Data JPA, Hibernate)
4. Authentication: JWT for secure login
5. Payment Integration: Stripe, PayPal, Razorpay
6. Hosting: AWS, Heroku, or DigitalOcean

#### Step-by-Step Implementation

##### Step 1: Setting Up the Spring Boot Backend

- Created a Spring Boot project using Spring Initializr.
- Configured MySQL database and established connections using Spring Data JPA.
- Developed REST APIs for user authentication, menu retrieval, and order management.
- Implemented Spring Security with JWT authentication for secure access control.

##### Step 2: Developing the Database Structure

- Designed database schema with tables for users, orders, menu, payments, and reviews.
- Used Hibernate ORM for managing data persistence.
- Optimized queries for fast data retrieval and transaction handling.

##### Step 3: Creating the User & Admin Dashboards

- Used Thymeleaf for server-side rendering (or React/Angular for SPA).
- Implemented dynamic menu display and cart management.
- Developed real-time order tracking and notifications.

## Step 4: Integrating Online Payment Gateway

- Configured third-party payment APIs (Razorpay, Stripe, or PayPal).
- Ensured secure transactions with encrypted payment processing.

## Step 5: Deployment & Optimization

- Deployed backend services on AWS/GCP for scalability.
- Used Docker for containerized deployment.
- Optimized API performance using caching techniques (Redis).

## CHALLENGES :

### Technical Challenges

#### Challenge 1: Ensuring Real-Time Order Tracking

- Problem: Customers need live updates on their order status (Processing → Out for Delivery → Delivered).
- Solution: Implemented WebSockets & polling mechanisms to send live updates to customers.

#### Challenge 2: Handling High Traffic & API Performance Optimization

- Problem: As the number of users increases, database queries slow down the system.
- Solution: Used Redis caching to store frequently accessed data (menu, user details) and optimized SQL queries with indexing and pagination.

#### Challenge 3: Secure Authentication & User Data Protection

- Problem: Unauthorized access and data breaches could compromise user data.
- Solution:
  - Implemented JWT-based authentication for secure login.
  - Used Spring Security & role-based access control (RBAC) to restrict admin features.
  - Enforced SSL encryption for secure API communication.

#### Challenge 4: Payment Integration Issues

- Problem: Handling payment failures, refunds, and order cancellations was complex.
- Solution:
  - Integrated payment gateway APIs (Razorpay/Stripe/PayPal) with proper transaction validation.
  - Implemented automatic refund processing for canceled orders.



## Challenge 5: Frontend-Backend API Communication Errors

- Problem: Inconsistent data synchronization between frontend and backend.
- Solution:
- Standardized RESTful API responses with proper status codes (200 OK, 400 Bad Request).
- Implemented CORS policies to allow cross-origin frontend-backend communication.

## Challenge 6: Deploying the Application in a Production Environment

- Problem: Managing server uptime, scalability, and handling multiple concurrent orders.
- Solution:
- Used Docker containers for easy deployment and scalability.
- Implemented auto-scaling on cloud platforms like AWS/GCP.
- Monitored system performance using Spring Boot Actuator & ELK Stack (Elasticsearch, Logstash, Kibana).

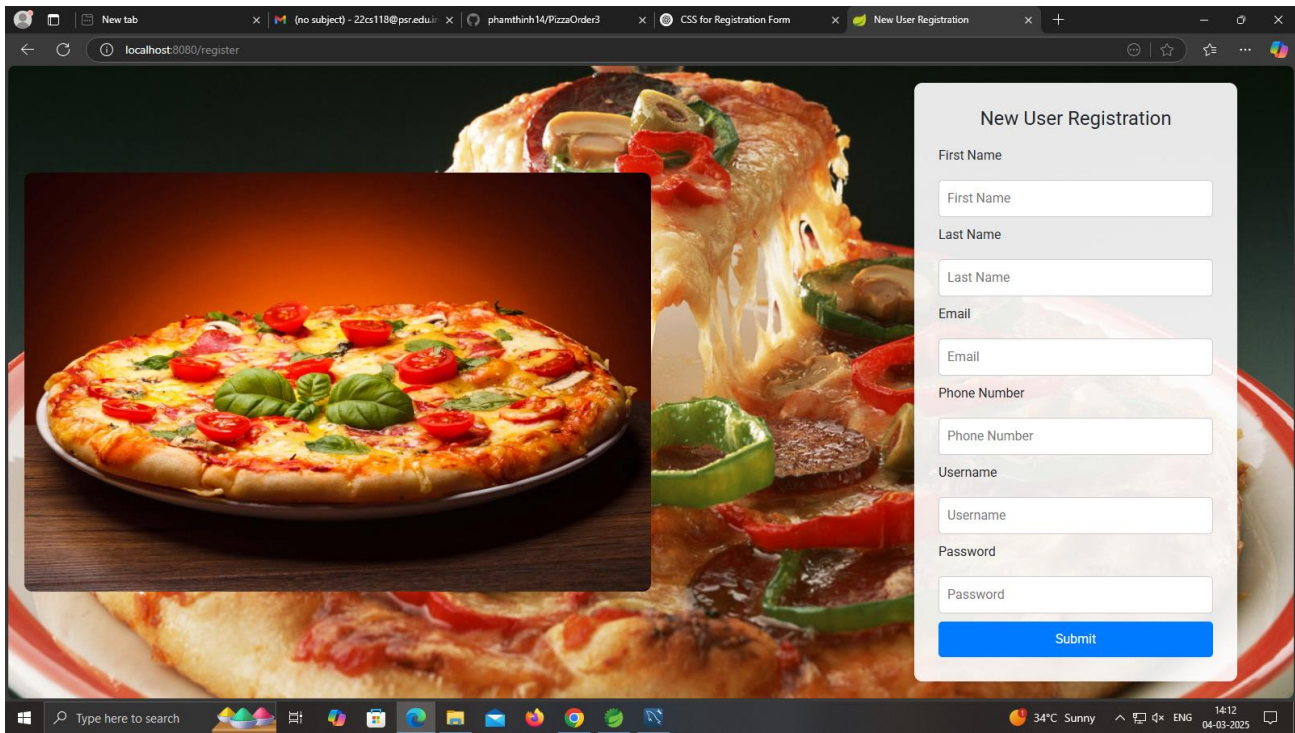
## FUTURE IMPROVEMENTS

- To enhance system performance and user experience, the following upgrades can be implemented:
- AI-based order recommendations to suggest popular pizzas based on customer history.
- Voice-activated ordering system for seamless customer experience.
- GPS-based delivery tracking to provide accurate real-time location updates.
- Machine learning for sales forecasting and menu optimization based on demand.

## CHAPTER 5

## RESULTS

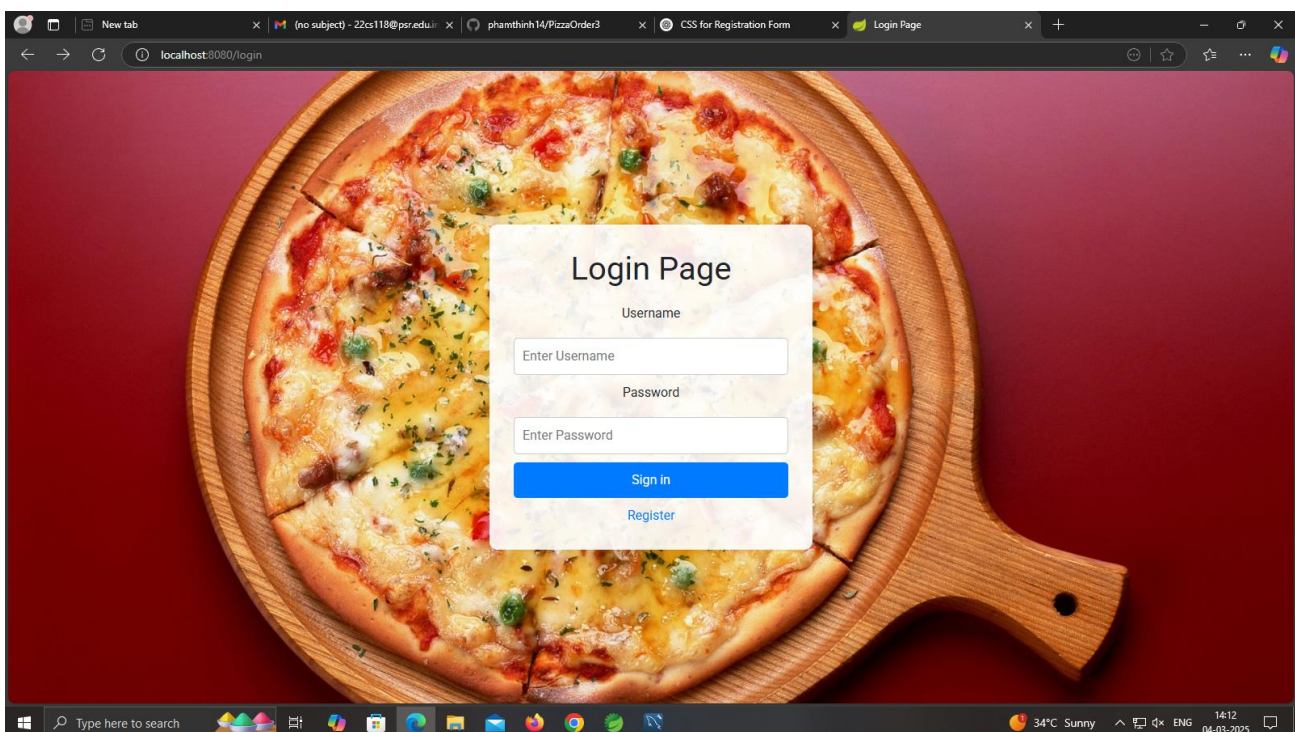
### SIGNUP PAGE:



The screenshot displays a web browser window with the address bar showing 'localhost:8080/register'. The page has a dark background with a large, appetizing image of a pizza. On the right side, there is a white 'New User Registration' form. The form includes the following fields and a submit button:

- First Name:
- Last Name:
- Email:
- Phone Number:
- Username:
- Password:
- Submit:

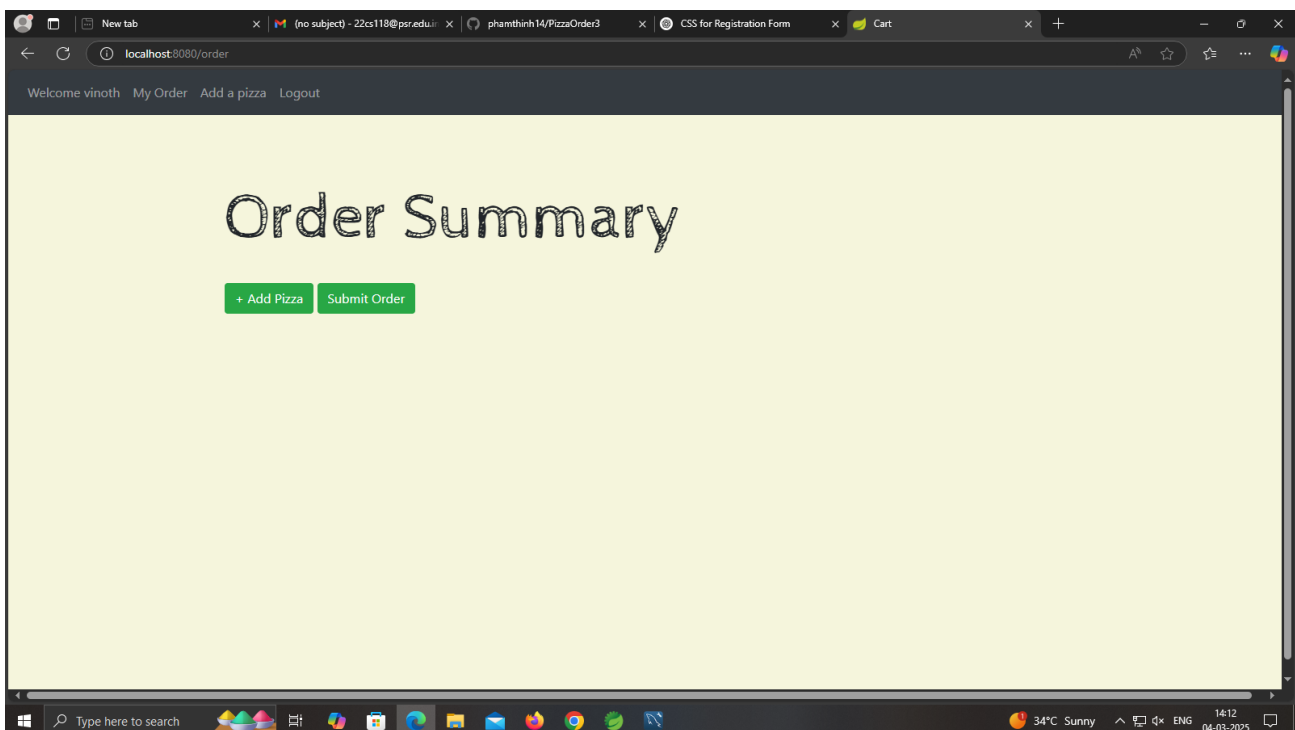
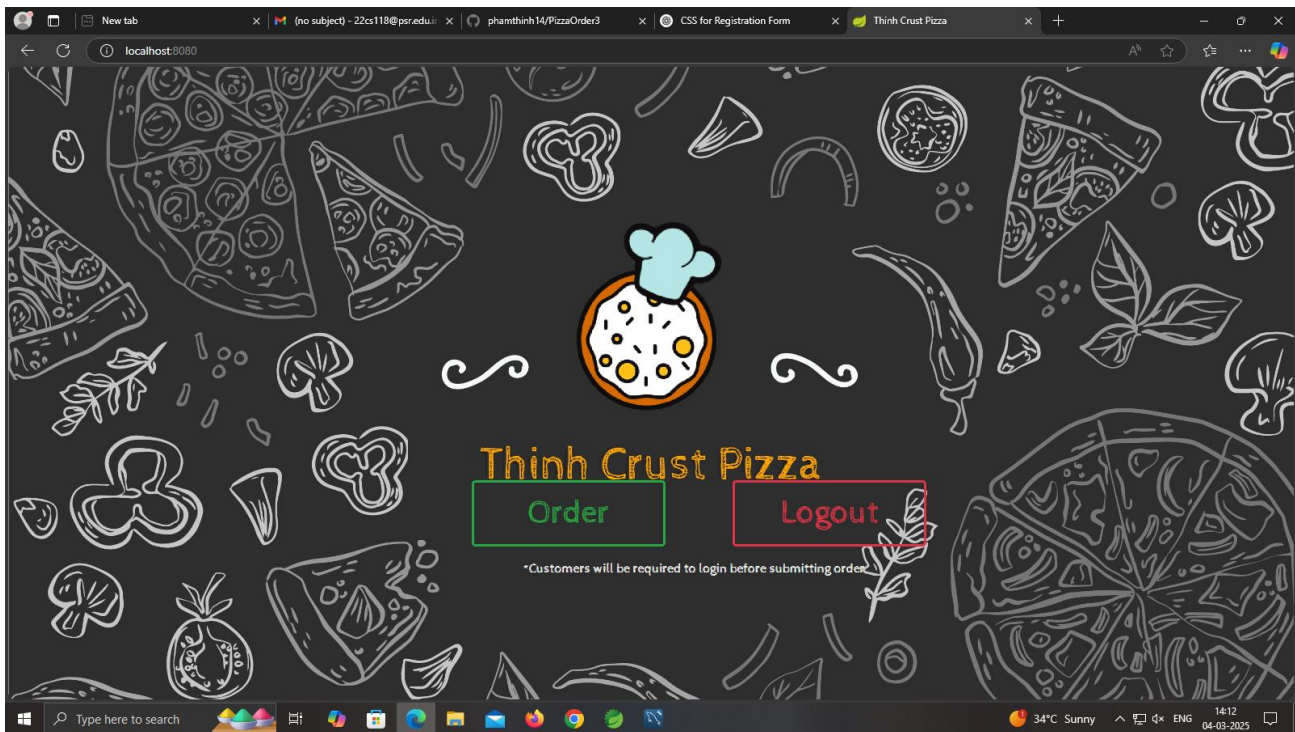
### LOGIN PAGE:



The screenshot displays a web browser window with the address bar showing 'localhost:8080/login'. The page has a dark red background with a large image of a pizza on a wooden board. In the center, there is a white 'Login Page' form. The form includes the following fields and buttons:


- Username:
- Enter Username:
- Password:
- Enter Password:
- Sign in:
- Register: [Register](#)

## HOME PAGE:



## ORDER PAGE:

Build Your Own Pizza!



**-Size-**  
Small ☒ Medium ☐ Large ☐

**-Dough-**  
Normal ☒ Gluten Free ☐

**-Sauce-**  
Red Sauce: ☒  
White Sauce: ☐  
No Sauce: ☐


**-Cheese-**  
Cheese: ☒

**-Veggies-**  
Mushroom ☐  
Onion ☐  
Green Peppers ☐

**-Meat-**  
Bacon: ☐  
Pepperoni: ☐  
Sausage: ☐

Submit

Build Your Own Pizza!



**-Size-**  
Small ☒ Medium ☐ Large ☐

**-Dough-**  
Normal ☒ Gluten Free ☐

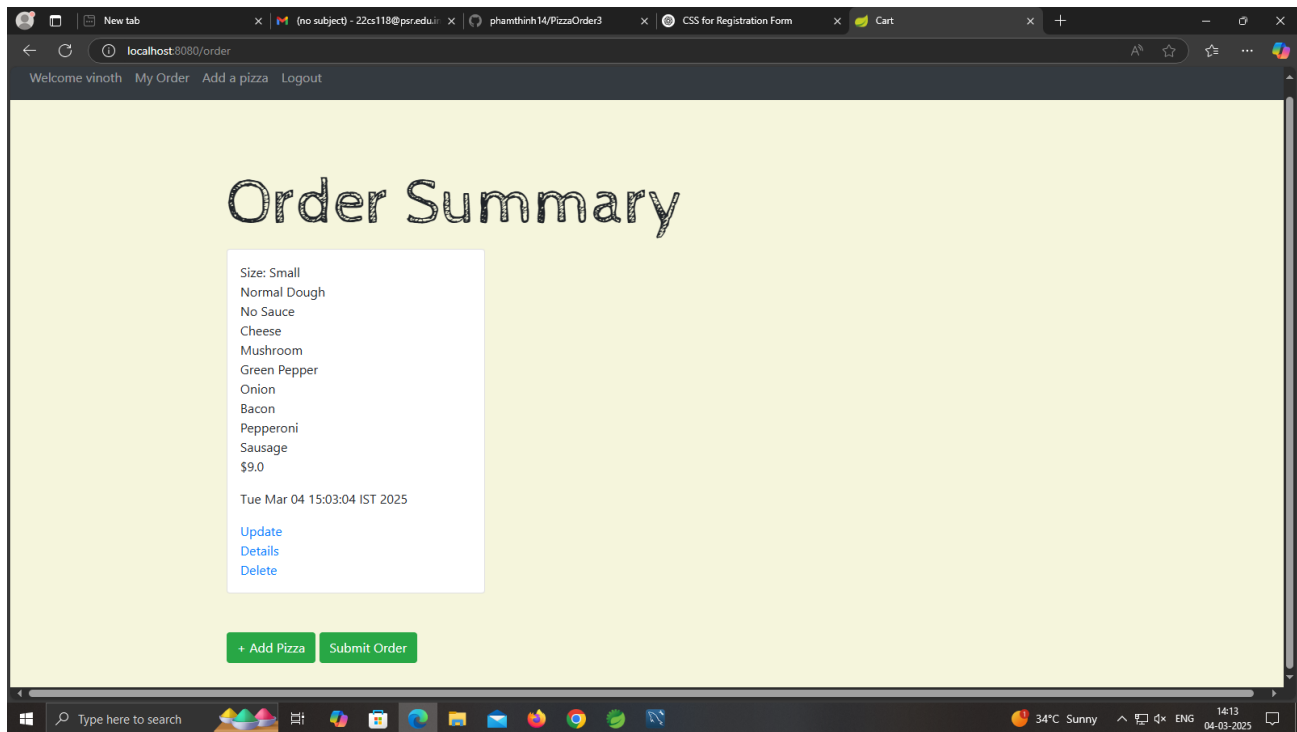
**-Sauce-**  
Red Sauce: ☐  
White Sauce: ☐  
No Sauce: ☒

**-Cheese-**  
Cheese: ☒

**-Veggies-**  
Mushroom ☒

**-Meat-**  
Bacon: ☒

## ORDER DETIALS PAGE:





# MYSQL :

The screenshot shows the MySQL Workbench interface. The left sidebar displays a tree of databases, with 'pizzas' expanded. The main window shows a query: `SELECT * FROM pizzas.user_data;`. The 'Result Grid' displays the following data:

id	email	enabled	first_name	last_name	password	phone_number	username
3	jim@jim.com	1	Jim	Jimmerson	\$2a\$10\$9HhUlpG6y6XicJ.u156SSQjdK6Ta...		jim
4	admin@admin.com	1	Admin	User	\$2a\$10\$6a0kx38aFQv2x95YQl.vupXHyA/...		admin
5	sfnfj	1	S4u	torh	\$2a\$10\$im8KmOQnccl.tnuFLZ/NOR5gAH8ZaA...		sfn

The bottom panel shows the 'Action Output' with a list of SQL statements and their execution times. The status bar at the bottom indicates the system time as 13:20 on 09-03-2025.

The screenshot shows the MySQL Workbench interface. The left sidebar displays a tree of databases, with 'pizzas' expanded. The main window shows a query: `SELECT * FROM pizzas.user_data_roles;`. The 'Result Grid' displays the following data:

user_id	role_id
3	1
4	2
5	1

The bottom panel shows the 'Action Output' with a list of SQL statements and their execution times. The status bar at the bottom indicates the system time as 13:21 on 09-03-2025.

## SOURCE CODE:

### HTML PROGRAM:

#### Index.html:

```
<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Pizza Delivery</title>

<link rel="stylesheet" href="styles.css">

</head>

<body>

<header>

<h1>Online Pizza Delivery</h1>

<nav>

<a href="cart.html">View Cart</a>

</nav>

</header>

<section class="menu">

<h2>Pizza Menu</h2>

<div id="pizza-list"></div>

</section>
```

```

<script>

fetch('http://localhost:8080/api/pizzas')

.then(response => response.json())

.then(data => {

let pizzaList = document.getElementById('pizza-list');

data.forEach(pizza => {

pizzaList.innerHTML += `

<div class="pizza-card">



<h3>${pizza.name}</h3>

<p>${pizza.description}</p>

<p>Price: $$${pizza.price}</p>

<button onclick="addToCart(${pizza.id}, '${pizza.name}', ${pizza.price})">Add to Cart</button>

</div>

`;

});

});

function addToCart(id, name, price) {

let cart = JSON.parse(localStorage.getItem("cart")) || [];

cart.push({ id, name, price });

localStorage.setItem("cart", JSON.stringify(cart));

alert(name + " added to cart!");

}

</script>

</body>

```



</html>

## Cart.html:

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Cart</title>

<link rel="stylesheet" href="styles.css">

</head>

<body>

<header>

<h1>Shopping Cart</h1>

<nav>

<a href="index.html">Back to Menu</a>

</nav>

</header>

<section class="cart">

<h2>Your Cart</h2>

<ul id="cart-list"></ul>

<button onclick="placeOrder()">Place Order</button>

</section>

```

<script>

let cart = JSON.parse(localStorage.getItem("cart")) || [];

let cartList = document.getElementById('cart-list');


cart.forEach(item => {

let li = document.createElement('li');

li.textContent = `${item.name} - ${item.price}`;

cartList.appendChild(li);

});


function placeOrder() {

fetch('http://localhost:8080/api/orders', {

method: 'POST',

headers: { 'Content-Type': 'application/json' },

body: JSON.stringify(cart)

}).then(response => {

if (response.ok) {

alert("Order placed successfully!");

localStorage.removeItem("cart");

window.location.href = "index.html";

}

});

}

</script>

</body>

</html>

```

## Style.css:

```
body {  
  
font-family: Arial, sans-serif;  
  
margin: 0;  
  
padding: 0;  
  
background-color: #f8f8f8;  
  
}
```

```
header {  
  
background-color: #d32f2f;  
  
color: white;  
  
padding: 15px;  
  
text-align: center;  
  
}
```

```
nav {  
  
text-align: center;  
  
margin-top: 10px;  
  
}
```

```
nav a {  
  
color: white;  
  
text-decoration: none;  
  
margin: 0 15px;  
  
font-size: 18px;
```

```
}
```

```
.menu, .cart {  
width: 80%;  
margin: 20px auto;  
text-align: center;  
}
```

```
h2 {  
color: #d32f2f;  
}
```

```
.pizza-card {  
background-color: white;  
padding: 15px;  
margin: 10px;  
display: inline-block;  
width: 250px;  
box-shadow: 0px 0px 10px rgba(0, 0, 0, 0.1);  
}
```

```
.pizza-card img {  
width: 100%;  
height: 150px;  
}
```

```
button {  
  
background-color: #d32f2f;  
  
color: white;  
  
border: none;  
  
padding: 10px;  
  
cursor: pointer;  
  
}
```

```
button:hover {  
  
background-color: #b71c1c;  
  
}
```

Login.html:

```
<!DOCTYPE html>  
  
<html lang="en">  
  
<head>  
  
<meta charset="UTF-8">  
  
<meta name="viewport" content="width=device-width, initial-scale=1.0">  
  
<title>Login</title>  
  
<link rel="stylesheet" href="styles.css">  
  
</head>  
  
<body>  
  
<div class="container">  
  
<h2>Login</h2>  
  
<form id="loginForm">  
  
<input type="text" id="loginUsername" placeholder="Username" required>
```

```
<input type="password" id="loginPassword" placeholder="Password" required>
```

```
<button type="submit">Login</button>
```

```
</form>
```

```
<p>Don't have an account? <a href="signup.html">Sign Up</a></p>
```

```
</div>
```

```
<script>
```

```
document.getElementById("loginForm").addEventListener("submit", function(event) {
```

```
event.preventDefault();
```

```
let username = document.getElementById("loginUsername").value;
```

```
let password = document.getElementById("loginPassword").value;
```

```
fetch("http://localhost:8080/api/auth/login", {
```

```
method: "POST",
```

```
headers: { "Content-Type": "application/json" },
```

```
body: JSON.stringify({ username, password })
```

```
})
```

```
.then(response => response.json())
```

```
.then(data => {
```

```
if (data.success) {
```

```
alert("Login successful!");
```

```
window.location.href = "index.html";
```

```
} else {
```

```
alert("Invalid credentials!");
```

```
}  
  
});  
  
});  
  
</script>  
  
</body>  
  
</html>
```

### Signup.html:

```
<!DOCTYPE html>  
  
<html lang="en">  
  
<head>  
  
<meta charset="UTF-8">  
  
<meta name="viewport" content="width=device-width, initial-scale=1.0">  
  
<title>Sign Up</title>  
  
<link rel="stylesheet" href="styles.css">  
  
</head>  
  
<body>  
  
<div class="container">  
  
<h2>Sign Up</h2>  
  
<form id="signupForm">  
  
<input type="text" id="signupUsername" placeholder="Username" required>  
  
<input type="email" id="signupEmail" placeholder="Email" required>  
  
<input type="password" id="signupPassword" placeholder="Password" required>  
  
<button type="submit">Sign Up</button>  
  
</form>
```

```
<p>Already have an account? <a href="login.html">Login</a></p>
```

```
</div>
```

```
<script>
```

```
document.getElementById("signupForm").addEventListener("submit", function(event) {
```

```
event.preventDefault();
```

```
let username = document.getElementById("signupUsername").value;
```

```
let email = document.getElementById("signupEmail").value;
```

```
let password = document.getElementById("signupPassword").value;
```

```
fetch("http://localhost:8080/api/auth/signup", {
```

```
method: "POST",
```

```
headers: { "Content-Type": "application/json" },
```

```
body: JSON.stringify({ username, email, password })
```

```
})
```

```
.then(response => response.json())
```

```
.then(data => {
```

```
if (data.success) {
```

```
alert("Signup successful! Please login.");
```

```
window.location.href = "login.html";
```

```
} else {
```

```
alert("Signup failed! Try again.");
```

```
}
```

```
});
```



```
});
```

```
</script>
```

```
</body>
```

```
</html>
```

## Style.css:

```
body {
```

```
font-family: Arial, sans-serif;
```

```
background-color: #f4f4f4;
```

```
display: flex;
```

```
justify-content: center;
```

```
align-items: center;
```

```
height: 100vh;
```

```
margin: 0;
```

```
}
```

```
.container {
```

```
background: white;
```

```
padding: 20px;
```

```
width: 300px;
```

```
text-align: center;
```

```
box-shadow: 0px 0px 10px rgba(0, 0, 0, 0.1);
```

```
border-radius: 10px;
```

```
}
```

```
h2 {  
  
color: #d32f2f;  
  
}
```

```
input {  
  
width: 90%;  
  
padding: 10px;  
  
margin: 10px 0;  
  
border: 1px solid #ccc;  
  
border-radius: 5px;  
  
}
```

```
button {  
  
width: 100%;  
  
padding: 10px;  
  
background-color: #d32f2f;  
  
color: white;  
  
border: none;  
  
cursor: pointer;  
  
border-radius: 5px;  
  
}
```

```
button:hover {  
  
background-color: #b71c1c;
```

```
}
```

```
p {
```

```
margin-top: 15px;
```

```
}
```

**Menu.html:**

```
<!DOCTYPE html>
```

```
<html lang="en">
```

```
<head>
```

```
<meta charset="UTF-8">
```

```
<meta name="viewport" content="width=device-width, initial-scale=1.0">
```

```
<title>Pizza Menu</title>
```

```
<link rel="stylesheet" href="styles.css">
```

```
</head>
```

```
<body>
```

```
<header>
```

```
<h1>Our Pizza Menu</h1>
```

```
<nav>
```

```
<a href="index.html">Home</a>
```

```
<a href="cart.html">Cart</a>
```

```
</nav>
```

```
</header>
```

```
<section class="menu">
```

```
<div class="pizza-card">



<h3>Margherita Pizza</h3>

<p>Classic cheese & tomato pizza.</p>

<p>Price: $8.99</p>

<button onclick="addToCart('Margherita Pizza', 8.99)">Add to Cart</button>

</div>
```

```
<div class="pizza-card">



<h3>Pepperoni Pizza</h3>

<p>Loaded with spicy pepperoni.</p>

<p>Price: $10.99</p>

<button onclick="addToCart('Pepperoni Pizza', 10.99)">Add to Cart</button>

</div>

</section>
```

```
<script>

function addToCart(name, price) {

let cart = JSON.parse(localStorage.getItem("cart")) || [];

cart.push({ name, price });

localStorage.setItem("cart", JSON.stringify(cart));

alert(name + " added to cart!");

}
```

```
</script>
```

```
</body>
```

```
</html>
```

## Style.css:

```
body {
```

```
font-family: Arial, sans-serif;
```

```
margin: 0;
```

```
padding: 0;
```

```
background-color: #f8f8f8;
```

```
text-align: center;
```

```
}
```

```
header {
```

```
background-color: #d32f2f;
```

```
color: white;
```

```
padding: 15px;
```

```
}
```

```
nav a {
```

```
color: white;
```

```
text-decoration: none;
```

```
margin: 10px;
```

```
font-size: 18px;
```

```
}
```

```
.hero {  
  
padding: 50px;  
  
background-image: url('pizza-bg.jpg');  
  
background-size: cover;  
  
color: white;  
  
}
```

```
.hero h2 {  
  
font-size: 2em;  
  
}
```

```
.btn {  
  
display: inline-block;  
  
padding: 10px 20px;  
  
background-color: #ff5722;  
  
color: white;  
  
text-decoration: none;  
  
border-radius: 5px;  
  
}
```

```
.menu {  
  
display: flex;  
  
justify-content: center;
```

```
flex-wrap: wrap;

padding: 20px;

}
```

```
.pizza-card {

background-color: white;

padding: 15px;

margin: 10px;

width: 250px;

text-align: center;

box-shadow: 0px 0px 10px rgba(0, 0, 0, 0.1);

}
```

```
.pizza-card img {

width: 100%;

height: 150px;

}
```

```
button {

background-color: #d32f2f;

color: white;

padding: 10px;

border: none;

cursor: pointer;
```

```
border-radius: 5px;

}
```

```
button:hover {

background-color: #b71c1c;

}
```

## Show.html:

```
<!DOCTYPE html>

<html lang="en" xmlns:th="http://www.thymeleaf.org">

<head>

<meta charset="UTF-8">

<th:block th:replace="base :: header"></th:block>

<title>Detail</title>

</head>

<body>

<div th:replace="base :: nav"></div>

<a href="/">Home</a>


<div th:if="${pizzaorder.smallSize == true}">

<p>Size: Small</p>

</div>

<div th:if="${pizzaorder.mediumSize == true}">

<p>Size: Medium</p>

</div>
```



```
<div th:if="${pizzaorder.largeSize == true}">

<p>Size: Large</p>

</div>

<div th:if="${pizzaorder.normalDough == true}">

<p>Normal Dough</p>

</div>

<div th:if="${pizzaorder.glutenFreeDough == true}">

<p>Gluten Free Dough</p>

</div>

<div th:if="${pizzaorder.redSauce == true}">

<p>Red Sauce</p>

</div>

<div th:if="${pizzaorder.whiteSauce == true}">

<p>Red Sauce</p>

</div>

<div th:if="${pizzaorder.noSauce == true}">

<p>No Sauce</p>

</div>

<div th:if="${pizzaorder.cheese == true}">

<p>Cheese</p>

</div>

<div th:if="${pizzaorder.noCheese == true}">

<p>No Cheese</p>

</div>
```

```
<div th:if="{pizzaorder.mushroom == true}">
```

```
<p>Mushroom</p>
```

```
</div>
```

```
<div th:if="{pizzaorder.greenPepper == true}">
```

```
<p>Green Pepper</p>
```

```
</div>
```

```
<div th:if="{pizzaorder.onions == true}">
```

```
<p>Onion</p>
```

```
</div>
```

```
<div th:if="{pizzaorder.bacon == true}">
```

```
<p>Bacon</p>
```

```
</div>
```

```
<div th:if="{pizzaorder.pepperoni == true}">
```

```
<p>Pepperoni</p>
```

```
</div>
```

```
<div th:if="{pizzaorder.sausage == true}">
```

```
<p>Sausage</p>
```

```
</div>
```

```
<p th:text="{pizzaorder.price}"></p>
```

```
<span>Your Pizza will arrive at <p th:text="{pizzaorder.time}"></p></span>
```

```
</div>
```

```
<th:block th:replace="base :: jslinks"></th:block>
```

</body>

</html>

## Updateorder.html:

<!DOCTYPE html>

<html lang="en" xmlns:th="http://www.thymeleaf.org">

<head>

<meta charset="UTF-8">

<title>Order Form</title>

<link rel="stylesheet"  
href="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/css/bootstrap.min.css"

integrity="sha384-  
ggOyR0iXCbMQv3Xipma34MD+dH/1fQ784/j6cY/iJTQUOhcWr7x9JvoRxT2MZw1T"  
crossorigin="anonymous">

<link  
href="https://fonts.googleapis.com/css?family=Roboto|Cabin+Sketch:400,700&display=swap"  
rel="stylesheet">

<!--https://stackpath.bootstrapcdn.com/font-awesome/4.7.0/css/font-awesome.min.css-->

<!--<script type="text/javascript"> (function() { var css = document.createElement('link'); css.href =  
' https://stackpath.bootstrapcdn.com/font-awesome/4.7.0/css/font-awesome.min.css'; css.rel =  
'stylesheet'; css.type = 'text/css'; document.getElementsByTagName('head')[0].appendChild(css);  
})(); </script>-->

<link rel="stylesheet" type="text/css" th:href="@ { css/orderform\_style.css }">

<link rel="stylesheet" type="text/css" href="../../static/css/orderform\_style.css"/>

<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.4.1/jquery.min.js"></script>

</head>

<style>

/\*#crust{\*/

```

/*position:absolute;*/

/*height:300px;*/

/*width:300px;*/

/*background-color: orange;*/

/*border-radius:50%;*/

/*z-index:5;*/

/* } */

/*#sauce{ */

/*position:absolute;*/

/*height:280px;*/

/*width:280px;*/

/*background-color: red;*/

/*border-radius:50%;*/

/*z-index:6;*/

/*left:10px;*/

/*top:10px;*/

/* } */

/*!*Changing Sauce with radio button *!*/

/#7:checked ~ div #sauce{ */

/*background-color:#fcf0d6;*/

/* } */

/*#8:checked ~ div #sauce{ */

/*background:transparent;*/

/* } */

```

```
/*!* Changing Size with button *!*/
```

```
/*#2:checked ~ div #sauce{*/
```

```
/*height:350px;*/
```

```
/*width:350px;*/
```

```
/*}*/
```

```
/*#2:checked ~ div #crust{*/
```

```
/*height:370px;*/
```

```
/*width:370px;*/
```

```
/*}*/
```

```
h1,h3{
```

```
font-family:"Cabin Sketch";
```

```
}
```

```
</style>
```

```
<body>
```

```
<div t:replace="base :: nav"></div>
```

```
<div class="row">
```

```
<div class="col-2"></div>
```

```
<div class="col-8">
```

```
<h1><del>Build</del> Edit Your <del>Own</del> Pizza!</h1>
```

```
<!--<div class="toppings_row row shadow-lg p-3 mb-5 bg-white rounded">-->
```

```
<!--&lt;!&ndash;<div class="row shadow-lg">&ndash;&gt;-->
```

```
<!--<div class="toppings_div">-->
```

```

<!--<div style="position:relative;">-->

<!--&lt;!&ndash;<div id="crust"></div>&ndash;&gt;-->

<!--&lt;!&ndash;<div id="sauce"></div>&ndash;&gt;-->

<!--</div>-->

<!--&lt;!&ndash;FOR LOCAL&ndash;&gt;-->

<!--&lt;!&ndash;&ndash;&gt;-->

<!--&lt;!&ndash;&ndash;&gt;-->

<!--&lt;!&ndash;&ndash;&gt;-->

<!--&lt;!&ndash;&ndash;&gt;-->

<!--&lt;!&ndash;&ndash;&gt;-->

<!--&lt;!&ndash;&ndash;&gt;-->

<!--&lt;!&ndash;&ndash;&gt;-->


<!--&lt;!&ndash;FOR WHEN RUNNING THROUGH SPRINGBOOT&ndash;&gt;-->

<!---->

<!---->

<!---->

<!---->

<!---->

<!---->

```

<!---->

<!--</div>-->

<!--</div>-->

<form action="#"

th:action="@{/process}"

th:object="\${pizzaorder}"

method="post"

onsubmit="return checkSelectAtLeastOne(this);">

<!--These are the contents-->

<input type="hidden" th:field="\*{id}">

<!--Size Section-->

<div class="row">

<div class="col">

<!--border-right: 1px solid black; margin-right: 10rem-->

<h3>-Size-</h3>

<div class="form-check form-check-inline">

<div class="group">

<label class="form-check-label" for="1">Small</label>

```
<input id="1" onclick="selectOneSizeOnly(this.id)" type="checkbox"
th:checked="{pizzaorder.smallSize == true}" name="smallSize" />
```

```
</div>
```

```
<div class="group">
```

```
<label class="form-check-label" for="2">Medium</label>
```

```
<input id="2" onclick="selectOneSizeOnly(this.id)" type="checkbox"
th:checked="{pizzaorder.mediumSize == true}" name="mediumSize">
```

```
</div>
```

```
<div class="group">
```

```
<label class="form-check-label" for="3">Large</label>
```

```
<input id="3" onclick="selectOneSizeOnly(this.id)" type="checkbox"
th:checked="{pizzaorder.largeSize == true}" name="largeSize">
```

```
</div>
```

```
<!--<div style="position:relative;">-->
```

```
<!--<div id="crust"></div>-->
```

```
<!--<div id="sauce"></div>-->
```

```
<!--</div>-->
```

```
</div>
```

```
<!--End Size Section-->
```

```
</div>
```

```
<!--Dough Section-->
```

```
<div class="col">
```

```
<h3>-Dough-</h3>
```



```

<div class="row">

<div class="group">

<label class="form-check-label" for="4">Normal</label>

<input id="4" onclick="selectOneDoughOnly(this.id)" type="checkbox"
th:checked="{pizzaorder.normalDough == true}" name="normalDough" />

</div>

<div class="group">

<label class="form-check-label" for="5">Gluten Free</label>

<input id="5" onclick="selectOneDoughOnly(this.id)" type="checkbox"
th:checked="{pizzaorder.glutenFreeDough == true}" name="glutenFreeDough">

</div>

</div>

</div>

</div>

</div>

<!--End Dough Section-->

<br/>

<!--Second Row-->

<div class="row">

<!--Sauce Section-->

<div class="col">

<h3>-Sauce-</h3>

<div class="group">

<label class="form-check-label" for="6">Red Sauce:</label>

<input id="6" onclick="selectOneSauceOnly(this.id)" type="checkbox"
th:checked="{pizzaorder.redSauce == true}" name="redSauce" />

```

</div>

<div class="group">

<label class="form-check-label" for="7">White Sauce:</label>

<input id="7" onclick="selectOneSauceOnly(this.id)" type="checkbox"  
th:checked="{pizzaorder.whiteSauce == true}" name="whiteSauce">

</div>

<div class="group">

<label class="form-check-label" for="8">No Sauce:</label>

<input id="8" onclick="selectOneSauceOnly(this.id)" type="checkbox"  
th:checked="{pizzaorder.noSauce == true}" name="noSauce">

</div>

<!--End Sauce Section-->

</div>

<!--Cheese Section-->

<div class="col">

<h3>-Cheese-</h3>

<div class="group">

<label class="form-check-label" for="9">Cheese:</label>

<input id="9" type="checkbox" th:checked="{pizzaorder.cheese == true}" name="cheese"  
checked/>

<!--<input id="9" onclick="cheeseOrNoCheese(this.id)" type="checkbox" name="cheese"  
checked/-->

</div>

<!--<div class="group">-->

<!--<label class="form-check-label" for="10">No Cheese:</label>-->

```

<!--<input id="10" onclick="cheeseOrNoCheese(this.id)" type="checkbox" name="noCheese"/>-->

<!--</div>-->

<br/>

<!--End Cheese Section-->

</div>

</div>

<!--3rd row-->

<div class="row">

<!--Vegetable-->

<div class="col">

<h3>-Veggies-</h3>

<div class="group">

<label class="form-check-label" for="11">Mushroom</label>

<input id="11" type="checkbox" th:checked="${pizzaorder.mushroom == true}"
name="mushroom">

</div>

<div class="group">

<label class="form-check-label" for="12">Onion</label>

<input id="12" type="checkbox" th:checked="${pizzaorder.onions == true}" name="onions">

</div>

<div class="group">

<label class="form-check-label" for="13">Green Peppers</label>

```

```
<input id="13" type="checkbox" th:checked="${pizzaorder.greenPepper == true}"
name="greenPepper">
```

```
</div>
```

```
<!--End Vegetable-->
```

```
</div>
```

```
<div class="col">
```

```
<!--Meat Section-->
```

```
<h3>-Meat-</h3>
```

```
<div class="group">
```

```
<label class="form-check-label" for="14">Bacon:</label>
```

```
<input id="14" type="checkbox" th:checked="${pizzaorder.bacon == true}" name="bacon">
```

```
</div>
```

```
<div class="group">
```

```
<label class="form-check-label" for="15">Pepperoni:</label>
```

```
<input id="15" type="checkbox" th:checked="${pizzaorder.pepperoni == true}"
name="pepperoni">
```

```
</div>
```

```
<div class="group">
```

```
<label class="form-check-label" for="16">Sausage:</label>
```

```
<input id="16" type="checkbox" th:checked="${pizzaorder.sausage == true}" name="sausage">
```

```
</div>
```

```
<!--End of Meat Section-->
```

```
</div>
```

```
</div>
```

```

<input type="hidden" th:field="*{time}">
<input type="hidden" th:field="*{user}">
<!--<input type="submit" value="Submit">-->

<br/>

<button type="submit" class="btn btn-lg btn-success">Submit</button>

<!--END-->

</form>

<br/>

```

```

</div>

<div class="col-2"></div>

</div>

</body>

<script>

function selectOneSizeOnly(id) {

for (var i = 1; i <= 3; i++) {

document.getElementById(i).checked = false;

}

document.getElementById(id).checked = true;

}

```

```

function selectOneDoughOnly(id) {

for (var i = 4; i <= 5; i++) {

document.getElementById(i).checked = false;

}

document.getElementById(id).checked = true;

}

```

```

function selectOneSauceOnly(id) {

for (var i = 6; i <= 8; i++) {

document.getElementById(i).checked = false;

}

document.getElementById(id).checked = true;

}

```

```

function cheeseOrNoCheese(id) {

for (var i = 9; i <= 10; i++) {

document.getElementById(i).checked = false;

}

document.getElementById(id).checked = true;

```

//Still a bug here

```

if($(this).is(":checked")){

$(".cheese-img").show();

}else{

```

```
$(".cheese-img").hide();  
  
}  
  
}
```

```
</script>
```

```
<script>
```

```
// $(".cheese-img").hide();  
  
$("#9").click(function(){  
  
if($(this).is(":checked")){  
  
$(".cheese-img").show();  
  
}else{  
  
$(".cheese-img").hide();  
  
}  
  
});
```

```
$(".mushroom-img").hide();  
  
$("#11").click(function(){  
  
if($(this).is(":checked")){  
  
$(".mushroom-img").show();  
  
}else{  
  
$(".mushroom-img").hide();
```

```
}
```

```
});
```

```
$(".onion-img").hide();
```

```
$("#12").click(function(){
```

```
if($(this).is(":checked")){
```

```
$(".onion-img").show();
```

```
}else{
```

```
$(".onion-img").hide();
```

```
}
```

```
});
```

```
$(".green-pepper-img").hide();
```

```
$("#13").click(function(){
```

```
if($(this).is(":checked")){
```

```
$(".green-pepper-img").show();
```

```
}else{
```

```
$(".green-pepper-img").hide();
```

```
}
```

```
});
```

```
$(".bacon-img").hide();
```

```
$("#14").click(function(){
```

```
if($(this).is(":checked")){
```



```
$(".bacon-img").show();

}else{

$(".bacon-img").hide();

}

});

$(".pepperoni-img").hide();

$("#15").click(function(){

if($(this).is(":checked")){

$(".pepperoni-img").show();

}else{

$(".pepperoni-img").hide();

}

});

$(".sausage-img").hide();

$("#16").click(function(){

if($(this).is(":checked")){

$(".sausage-img").show();

}else{

$(".sausage-img").hide();

}

});

</script>
```

<script src="https://code.jquery.com/jquery-3.2.1.slim.min.js" integrity="sha384-KJ3o2DKtIkvYIK3UENzmM7KCkRr/rE9/Qpg6aAZGJwFDMVNA/GpGFF93hXpG5KkN" crossorigin="anonymous"></script>

</html>

## CHAPTER 7

### CONCLUSION

The Online Pizza Delivery System streamlines the process of ordering and delivering pizzas efficiently. By integrating Spring Boot, Java, MySQL, and a user-friendly frontend, the system ensures smooth order placement, real-time tracking, and secure transactions. It enhances customer convenience, reduces manual workload for restaurant staff, and optimizes delivery operations. With features like user authentication, menu management, and automated billing, the project successfully improves the traditional pizza ordering experience. Future enhancements can include AI-driven recommendations, multiple payment integrations, and delivery partner tracking for an even better user experience.

#### Key Takeaways:

- **Automation & Efficiency:** Streamlines the pizza ordering and delivery process, reducing manual effort.
- **User Convenience:** Provides an easy-to-use interface for customers to browse menus, place orders, and track deliveries.
- **Secure Transactions:** Ensures data security with authentication and safe payment options.
- **Real-time Order Management:** Enables restaurants to manage orders, inventory, and deliveries efficiently.
- **Scalability:** Can be expanded with AI-based recommendations, multiple payment integrations, and GPS tracking for delivery optimization.

## CHAPTER 8

### REFERENCE

- E. D. Madyatmadja, D. Pristinella, A. S. Nirvani, I. Hilmansyah, A. R. Rajendra and R. Aulia, "The Effectiveness of the Online Food Delivery Application on the Person Who Lives in Boarding House," *2023 International Conference on Information Management and Technology (ICIMTech)*, Malang, Indonesia, 2023, pp. 30-35, doi: 10.1109/ICIMTech59029.2023.10277800.  
keywords: {Costs;Data analysis;Writing;Mathematical models;Information management;effectivity;restaurant;online food delivery;structural equation modelling},
- C. -E. Domokos, B. Séra, K. Simon, L. Kovács and T. -B. Szakács, "Netfood: A Software System for Food Ordering and Delivery," *2018 IEEE 16th International Symposium on Intelligent Systems and Informatics (SISY)*, Subotica, Serbia, 2018, pp. 000143-000148, doi: 10.1109/SISY.2018.8524854.  
keywords: {Servers;Databases;Companies;Data models;Software systems;Personnel;Computer architecture},
- T. Sahoo, S. Sonwani, R. Pandya and N. Rane, "Advanced Food Delivery: Efficiency, Safety, and Satisfaction Innovations," *2024 2nd World Conference on Communication & Computing (WCONF)*, RAIPUR, India, 2024, pp. 1-6, doi: 10.1109/WCONF61366.2024.10692311.  
keywords: {Strips;Technological innovation;Ergonomics;Customer satisfaction;Materials reliability;Safety;Maintenance;Personnel;Standards;Logistics;Food delivery;Delivery Bag;Safety;Temperature Maintenance;Lifting Technique;GPS},