RAJALAKSHMI
ENGINEERING COLLEGE
An AUTONOMOUS Institution
Affiliated to ANNA UNIVERSITY, Chennai

ONLINE FOOD ORDERING SYSTEM

MINI PROJECT REPORT

SUBMITTED BY:

MANIKANDAN M - 230701174

MATHAN KUMAR M - 230701181

In partial fulfilment for the award of the Degree of

BACHELOR OF ENGINEERING
IN
COMPUTER SCIENCE

RAJALAKSHMI ENGINEERING COLLEGE
THANDALAM,CHENNAI-602105

2024-2025

BONAFIDE CERTIFICATE

Certified that this project report " **MOVIE TICKET BOOKING SYSTEM**" is the Bonafide work of **"MANIKANDAN M (230701174)** AND **MATHAN KUMAR M (230701181)"** who carried out the project work under my supervision.

Submitted for the Practical Examination held on _____

SIGNATURE

Mrs.K.Maheshmeena
Assistant Professor
Computer Science and Engineering,
Rajalakshmi Engineering College
(Autonomous), Thandalam, Chennai-602105

ABSTRACT:

The Food Ordering System is a software application designed to streamline the process of ordering food online. With the rise of digital platforms, traditional methods of food ordering, such as phone calls and in-person visits, have become inefficient and time-consuming. This system provides a convenient and user-friendly interface for customers to place food orders from various restaurants, manage personal details, and track their order history.

The system is powered by a relational database built with MySQL, which manages customer, restaurant, menu item, and order data. The system allows customers to browse menus, select items, specify quantities, and calculate the total price of their orders. It also provides functionality for customers to view their previous orders, including item names, quantities, total prices, and order dates.

The Food Ordering System employs Java for backend logic and a graphical user interface (GUI) created using Swing. The database is designed with four main tables: Customer, Restaurant, MenuItem, and CustomerOrder, ensuring data integrity through the use of foreign keys and relationships between entities.

This system benefits both customers and restaurants by improving the efficiency of food ordering and order management processes. It reduces errors in order processing, enhances the customer experience with an intuitive interface, and helps restaurants manage orders effectively. Ultimately, this project demonstrates the practical application of database management systems and software development in modern food service environments.

TABLE OF CONTENTS

1. INTRODUCTION

1.1.Background

In the digital age, online food ordering has become increasingly popular. Customers prefer the convenience of ordering food online, and restaurants benefit from the streamlined process of receiving and managing orders. The Food Ordering System aims to provide a platform for customers to place food orders easily and for restaurants to manage those orders efficiently**.**

1.2.ProblemStatement

Traditional food ordering methods, such as phone calls or in-person visits, are often prone to errors, delays, and miscommunication. This system addresses these challenges by automating the ordering process and improving efficiency for both customers and restaurants. The goal is to provide an online interface where customers can browse menus, place orders, and track their order history.

1.3.Objectives

The main objectives of the Food Ordering System are:

- To create a user-friendly interface for customers to select restaurants, menu items, and place orders.
- To design a relational database that manages customer, restaurant, and order information.
- To automate the process of order placement, calculating prices, and order history retrieval.
- To ensure data integrity and seamless interaction between the frontend (user interface) and backend (database).

2. System Design

2.1.DatabaseDesign

The database is the backbone of the system, storing critical information such as customer details, restaurant information, menu items, and orders. The schema is as follows:

- **Customer Table**: Stores customer details like name, email, and phone number.
- **Restaurant Table**: Stores information about restaurants such as name, address, and phone number.
- **MenuItem Table**: Contains menu items with their respective prices, linked to a specific restaurant.
- **CustomerOrder Table**: Records the customer's order, linking customer and menu items along with quantity, total price, and order date.

Entity-RelationshipDiagram(ERD)

An ERD is created to define the relationships between entities in the system. The primary relationships are:

- A **Customer** can place multiple **Orders** (one-to-many relationship).
- A **Restaurant** can have multiple **MenuItems** (one-to-many relationship).
- An **Order** contains one or more **MenuItems**, forming a many-to-many relationship between **CustomerOrder** and **MenuItem**.
-

2.2.UserInterfaceDesign

The user interface is designed using **Java Swing** components. The layout includes:

- A combo box for selecting the restaurant.
- A dynamic combo box for selecting menu items, which updates based on the selected restaurant.

- Input fields for entering quantity, customer name, email, and phone number.
- Buttons for placing an order and viewing order history.

UI Flow

1. The user selects a restaurant from a dropdown.
2. The menu items for the selected restaurant are displayed.
3. The user selects a menu item and enters the quantity.
4. The system calculates the total price, and the user places the order.
5. The order details are stored in the database and can be viewed later by the customer.

3. Implementation

3.1.DatabaseSetupConfiguration:

The MySQL database is used for storing data. The tables are created as per the schema defined in the design phase. Foreign keys are used to link tables, ensuring data integrity between customers, menu items, and orders.

3.2. Backend Logic

- JDBC (Java Database Connectivity) is used to connect the Java application with the MySQL database.
- SQL queries are executed to insert new customers, retrieve menu items, place orders, and fetch order history.

3.3. User Interface Development

- Swing is used for developing the graphical user interface.
- Components such as JComboBox, JTextField, JButton, and JTextArea are used for interaction.
- Event listeners are implemented to handle user interactions, such as selecting a restaurant, placing an order, and viewing orders.

```
CREATE TABLE Customer (
   id INT AUTO_INCREMENT PRIMARY KEY,
   name VARCHAR(255),
   email VARCHAR(255),
   phone VARCHAR(20)
);

CREATE TABLE Restaurant (
   id INT AUTO_INCREMENT PRIMARY KEY,
   name VARCHAR(255),
   address VARCHAR(255),
   phone VARCHAR(20)
);

CREATE TABLE MenuItem (
   id INT AUTO_INCREMENT PRIMARY KEY,
   name VARCHAR(255),
```

```
    price DOUBLE,
    restaurant_id INT,
    FOREIGN KEY (restaurant_id) REFERENCES Restaurant(id)
);

CREATE TABLE CustomerOrder (
    id INT AUTO_INCREMENT PRIMARY KEY,
    customer_id INT,
    item_id INT,
    quantity INT,
    total_price DOUBLE,
    order_date DATE,
    FOREIGN KEY (customer_id) REFERENCES Customer(id),
    FOREIGN KEY (item_id) REFERENCES MenuItem(id)
);
```

- **JDBC (Java Database Connectivity)** is used to establish a connection between the Java application and the MySQL database.
- The system handles the following functionalities:
  - **Insert new customers** into the Customer table.
  - **Select menu items** based on the restaurant chosen.
  - **Calculate total price** based on selected menu items and quantity.
  - **Insert orders** into the CustomerOrder table, linking customers and menu items.
  - **Retrieve order history** for customers from the database.

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;
public class DBConnection {
        public static Connection connect() {
       try {
          Class.forName("com.mysql.cj.jdbc.Driver");
          return
DriverManager.getConnection("jdbc:mysql://localhost:3306/ordering",
"root", "Changeme@54"); // Replace with your credentials
```

```
        } catch (ClassNotFoundException | SQLException e) {
            e.printStackTrace();
        }
        return null;
    }
}
```

3.4 User Interface Development

The **Swing** library was used to create the graphical user interface. Key components include:

- **JComboBox** for selecting restaurants and menu items.
- **JTextField** for entering quantity and customer information.
- **JButton** for submitting orders and viewing order history.
- **JTextArea** for displaying order confirmation or error messages.

Event handling is implemented to respond to user actions such as selecting a restaurant, placing an order, and viewing past orders.

Java Frontend:

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import java.sql.*;
import java.util.*;

public class FoodOrderingSystem extends JFrame {

    private JComboBox<String> restaurantComboBox;
    private JComboBox<String> menuComboBox;
    private JTextField quantityField;
    private JTextField nameField, emailField, phoneField; // New input fields
    private JButton orderButton;
    private JButton viewOrdersButton;
    private JTextArea outputArea;

    private int selectedRestaurantId;

    public FoodOrderingSystem() {
```

```
        setTitle("Online Food Ordering System");
        setSize(500, 500); // Adjust the window size to fit new fields
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setLocationRelativeTo(null);

        JPanel panel = new JPanel();
        panel.setLayout(new GridLayout(9, 2));  // Adjust grid layout to fit new
fields

        // Restaurant ComboBox
        JLabel restaurantLabel = new JLabel("Select Restaurant:");
        restaurantComboBox = new JComboBox<>();
        panel.add(restaurantLabel);
        panel.add(restaurantComboBox);

        // Menu ComboBox
        JLabel menuLabel = new JLabel("Select Menu Item:");
        menuComboBox = new JComboBox<>();
        panel.add(menuLabel);
        panel.add(menuComboBox);

        // Quantity TextField
        JLabel quantityLabel = new JLabel("Quantity:");
        quantityField = new JTextField();
        panel.add(quantityLabel);
        panel.add(quantityField);

        // Customer Name Input
        JLabel nameLabel = new JLabel("Your Name:");
        nameField = new JTextField();
        panel.add(nameLabel);
        panel.add(nameField);

        // Customer Email Input
        JLabel emailLabel = new JLabel("Email:");
        emailField = new JTextField();
        panel.add(emailLabel);
        panel.add(emailField);
```

```
        // Customer Phone Input
        JLabel phoneLabel = new JLabel("Phone Number:");
        phoneField = new JTextField();
        panel.add(phoneLabel);
        panel.add(phoneField);

        // Order Button
        orderButton = new JButton("Place Order");
        panel.add(orderButton);

        // View Orders Button
        viewOrdersButton = new JButton("View Orders");
        panel.add(viewOrdersButton);

        // Output Area for messages or orders
        outputArea = new JTextArea();
        outputArea.setEditable(false);
        panel.add(new JScrollPane(outputArea));

        add(panel);

        // Load Restaurants
        loadRestaurants();

        // Add action listener for restaurant selection
        restaurantComboBox.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                selectedRestaurantId = getRestaurantIdFromName((String)
restaurantComboBox.getSelectedItem());
                loadMenuItems(selectedRestaurantId);
            }
        });

        // Add action listener for order button
        orderButton.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                placeOrder();
```

```
            }
        });

        // Add action listener for view orders button
        viewOrdersButton.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                viewOrders();
            }
        });
    }

    private void loadRestaurants() {
        try (Connection conn = DBConnection.connect()) {
            String sql = "SELECT id, name FROM Restaurant";
            try (Statement stmt = conn.createStatement(); ResultSet rs = stmt.executeQuery(sql)) {
                while (rs.next()) {
                    restaurantComboBox.addItem(rs.getString("name"));
                }
            }
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }

    private void loadMenuItems(int restaurantId) {
        try (Connection conn = DBConnection.connect()) {
            String sql = "SELECT id, name, price FROM MenuItem WHERE restaurant_id = ?";
            try (PreparedStatement pstmt = conn.prepareStatement(sql)) {
                pstmt.setInt(1, restaurantId);
                try (ResultSet rs = pstmt.executeQuery()) {
                    menuComboBox.removeAllItems();
                    while (rs.next()) {
                        menuComboBox.addItem(rs.getString("name"));
                    }
                }
            }
```

```
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }

    private int getRestaurantIdFromName(String restaurantName) {
        try (Connection conn = DBConnection.connect()) {
            String sql = "SELECT id FROM Restaurant WHERE name = ?";
            try (PreparedStatement pstmt = conn.prepareStatement(sql)) {
                pstmt.setString(1, restaurantName);
                try (ResultSet rs = pstmt.executeQuery()) {
                    if (rs.next()) {
                        return rs.getInt("id");
                    }
                }
            }
        } catch (SQLException e) {
            e.printStackTrace();
        }
        return -1; // Invalid ID
    }

    private void placeOrder() {
        String selectedItem = (String) menuComboBox.getSelectedItem();
        int quantity = Integer.parseInt(quantityField.getText());
        int customerId = insertCustomer();  // Insert new customer and get the ID
        int itemId = getMenuItemIdFromName(selectedItem);
        double totalPrice = calculateTotalPrice(itemId, quantity);

        String orderDate = "2024-11-11"; // Hardcoded for simplicity

        try (Connection conn = DBConnection.connect()) {
            String sql = "INSERT INTO CustomerOrder (customer_id, item_id,
quantity, total_price, order_date) VALUES (?, ?, ?, ?, ?)";
            try (PreparedStatement pstmt = conn.prepareStatement(sql)) {
                pstmt.setInt(1, customerId);
                pstmt.setInt(2, itemId);
                pstmt.setInt(3, quantity);
```

```
            pstmt.setDouble(4, totalPrice);
            pstmt.setString(5, orderDate);
            pstmt.executeUpdate();
            outputArea.append("Order placed successfully!\n");
        }
    } catch (SQLException e) {
        e.printStackTrace();
        outputArea.append("Error placing the order.\n");
    }
}

private int insertCustomer() {
    String name = nameField.getText();
    String email = emailField.getText();
    String phone = phoneField.getText();

    try (Connection conn = DBConnection.connect()) {
        String sql = "INSERT INTO Customer (name, email, phone) VALUES
(?, ?, ?)";
        try (PreparedStatement pstmt = conn.prepareStatement(sql,
Statement.RETURN_GENERATED_KEYS)) {
            pstmt.setString(1, name);
            pstmt.setString(2, email);
            pstmt.setString(3, phone);
            pstmt.executeUpdate();

            // Get the generated customer ID
            try (ResultSet generatedKeys = pstmt.getGeneratedKeys()) {
                if (generatedKeys.next()) {
                    return generatedKeys.getInt(1);
                }
            }
        }
    } catch (SQLException e) {
        e.printStackTrace();
    }
    return -1; // Invalid ID
}
```

```
private int getMenuItemIdFromName(String itemName) {
    try (Connection conn = DBConnection.connect()) {
        String sql = "SELECT id FROM MenuItem WHERE name = ?";
        try (PreparedStatement pstmt = conn.prepareStatement(sql)) {
            pstmt.setString(1, itemName);
            try (ResultSet rs = pstmt.executeQuery()) {
                if (rs.next()) {
                    return rs.getInt("id");
                }
            }
        }
    } catch (SQLException e) {
        e.printStackTrace();
    }
    return -1; // Invalid ID
}

private double calculateTotalPrice(int itemId, int quantity) {
    double price = 0;
    try (Connection conn = DBConnection.connect()) {
        String sql = "SELECT price FROM MenuItem WHERE id = ?";
        try (PreparedStatement pstmt = conn.prepareStatement(sql)) {
            pstmt.setInt(1, itemId);
            try (ResultSet rs = pstmt.executeQuery()) {
                if (rs.next()) {
                    price = rs.getDouble("price");
                }
            }
        }
    } catch (SQLException e) {
        e.printStackTrace();
    }
    return price * quantity;
}

// Method to fetch and display the list of orders
private void viewOrders() {
```

```
        try (Connection conn = DBConnection.connect()) {
            String sql = "SELECT c.name AS customer_name, m.name AS item_name, o.quantity, o.total_price, o.order_date " +
                         "FROM CustomerOrder o " +
                         "JOIN Customer c ON o.customer_id = c.id " +
                         "JOIN MenuItem m ON o.item_id = m.id " +
                         "ORDER BY o.order_date DESC";

            try (Statement stmt = conn.createStatement(); ResultSet rs = stmt.executeQuery(sql)) {
                outputArea.setText(""); // Clear the output area
                while (rs.next()) {
                    String orderInfo = String.format("Customer: %s\nItem: %s\nQuantity: %d\nTotal Price: %.2f\nDate: %s\n\n",
                            rs.getString("customer_name"),
                            rs.getString("item_name"),
                            rs.getInt("quantity"),
                            rs.getDouble("total_price"),
                            rs.getString("order_date"));
                    outputArea.append(orderInfo);
                }
            }
        } catch (SQLException e) {
            e.printStackTrace();
            outputArea.append("Error fetching orders.\n");
        }
    }

    public static void main(String[] args) {
        SwingUtilities.invokeLater(new Runnable() {
            public void run() {
                new FoodOrderingSystem().setVisible(true);
            }
        });
    }
}
```

4. Testing and Validation

4.1.TestCasesandResults

The system is tested using a series of test cases to ensure that:

- Customers can successfully place orders.
- The system calculates the total price correctly.
- Orders are correctly stored in the database.
- The system retrieves order history accurately.

Example Test Cases:

- Test Case 1: Place an order for "Chikken Pizza" with quantity 2.
- Test Case 2: View the list of all previous orders for a customer.

4.2.SystemandUserTesting

System testing was conducted to ensure the backend logic (database interactions) works smoothly. User acceptance testing (UAT) involved a small group of users to ensure the interface was intuitive and easy to use. Feedback from users was positive, highlighting the simplicity and efficiency of the ordering process.

4.3 Entity-Relationship Diagram (ERD) Description
Entities and Their Attributes

Sure! Below is a description of the Entity-Relationship Diagram (ERD) for the Food Ordering System that we discussed. I'll also explain how each entity is related to others. While I can't generate actual images directly here, you can easily create an ERD using drawing tools like draw.io, Lucidchart, or even a simple pencil-and-paper diagram**.**

1. **Customer**

   Attributes:
   - id (Primary Key): Unique identifier for each customer.
   - name: Name of the customer.
   - email: Email address of the customer.
   - phone: Contact phone number of the customer**.**

2. **Restaurant**

   Attributes:
   - id (Primary Key): Unique identifier for each restaurant.
   - name: Name of the restaurant.
   - address: Address of the restaurant.
   - phone: Contact phone number of the restaurant.

3. **MenuItem**

   Attributes:
   - id (Primary Key): Unique identifier for each menu item.
   - name: Name of the menu item (e.g., "Chikken Pizza").
   - price: Price of the menu item.
   - restaurant_id (Foreign Key): Links to the restaurant where this menu item is available.

4. **CustomerOrder**

   Attributes:
   - id (Primary Key): Unique identifier for each order.
   - customer_id (Foreign Key): Links to the Customer table.
   - item_id (Foreign Key): Links to the MenuItem table.
   - quantity: Number of items ordered.
   - total_price: Total cost for the order (calculated based on the

quantity and item price).
    - order_date: Date the order was placed.Relationships Between Entities

- **Customer to CustomerOrder (1:M):**

A Customer can place multiple CustomerOrders, but each CustomerOrder is linked to only one Customer.
Relationship: One-to-Many (One customer can have many orders, but each order belongs to only one customer).

- **Restaurant to MenuItem (1:M):**

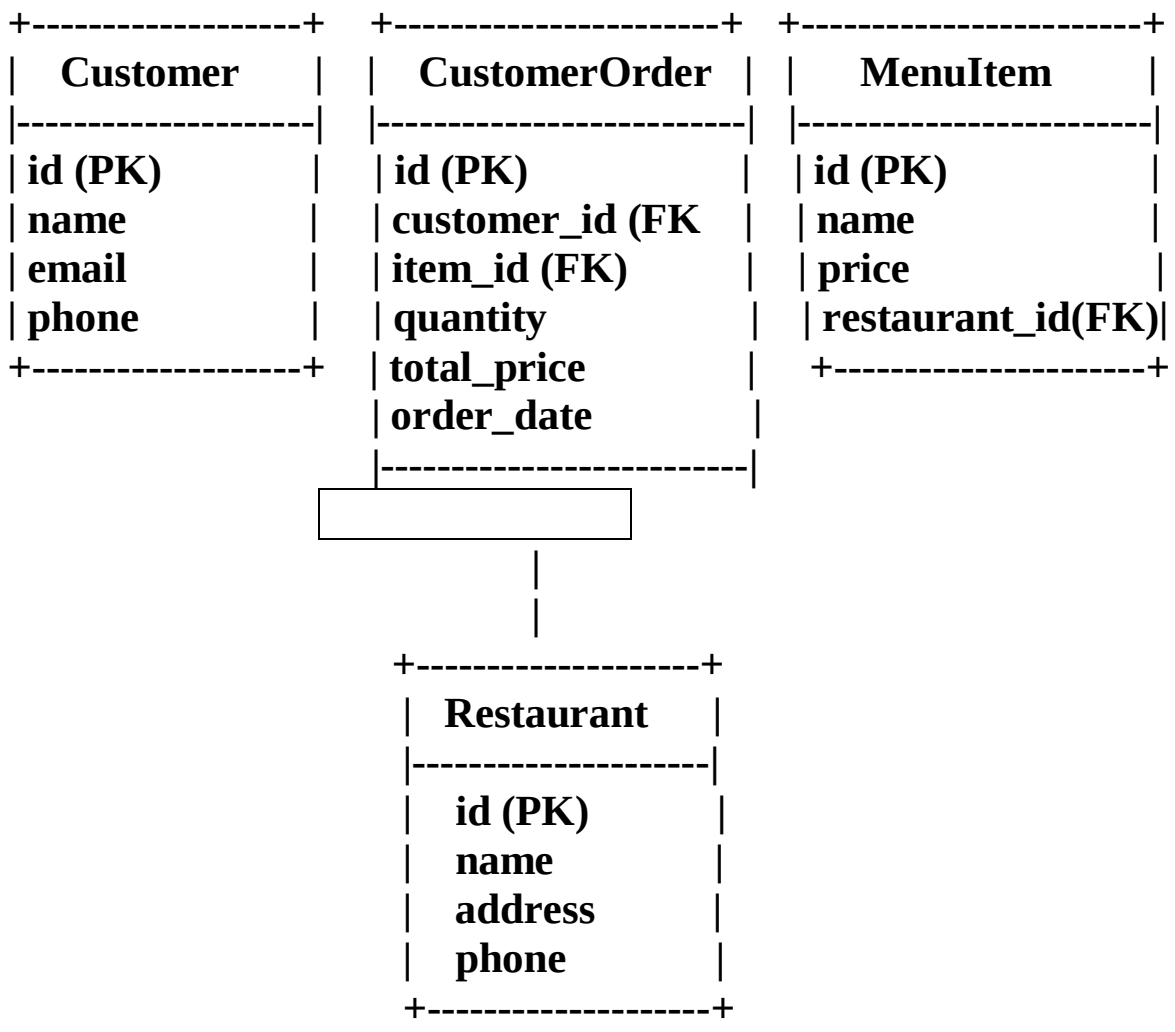
A Restaurant can have many MenuItems, but each MenuItem belongs to only one Restaurant.
Relationship: One-to-Many (One restaurant can have many menu items, but each item belongs to a single restaurant).

- **MenuItem to CustomerOrder (M:N):**

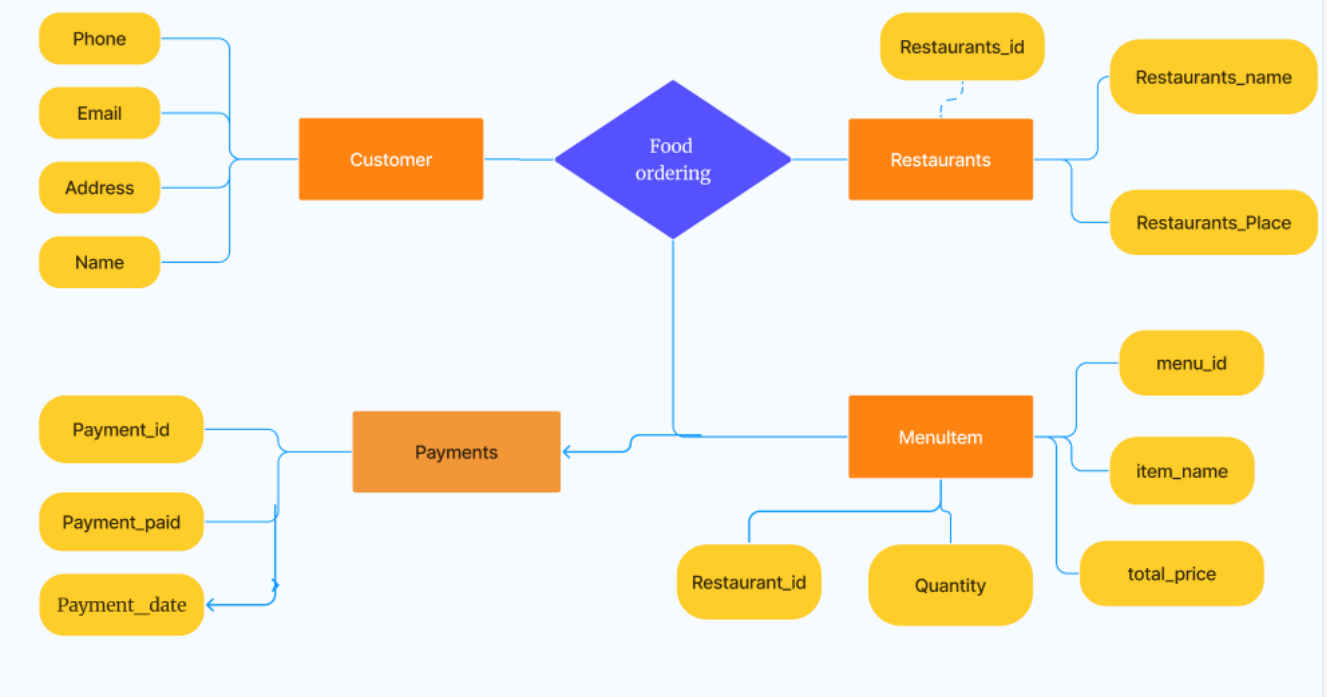
A MenuItem can appear in many CustomerOrders, and a CustomerOrder can contain multiple MenuItems.
This is a Many-to-Many relationship, which is resolved by using a foreign key in the CustomerOrder table that references MenuItem.

ERD Representation

You can visualize the following ERD schema based on the above details:



Entity Relationships



5. Results and Analysis

5.1 System Functionality

The system performs as expected. It allows customers to:

- Select a restaurant and browse its menu.
- Place an order with the chosen item and quantity.
- View the order confirmation and history.

5.2 Performance Evaluation

The system works efficiently with a small to medium-sized database. Future improvements could include optimizing the database for larger datasets or integrating payment systems for online payments.

5. Appendix:

5.1.CodeListings

Sample code for placing an order:

Java:

```
// Example of placing an order
String sql = "INSERT INTO CustomerOrder (customer_id, item_id, quantity, total_price, order_date) VALUES (?, ?, ?, ?, ?)";
PreparedStatement pstmt = conn.prepareStatement(sql);
pstmt.setInt(1, customerId);
pstmt.setInt(2, itemId);
pstmt.setInt(3, quantity);
pstmt.setDouble(4, totalPrice);
pstmt.setString(5, orderDate);
pstmt.executeUpdate();
```

5.2.SQLQueries:

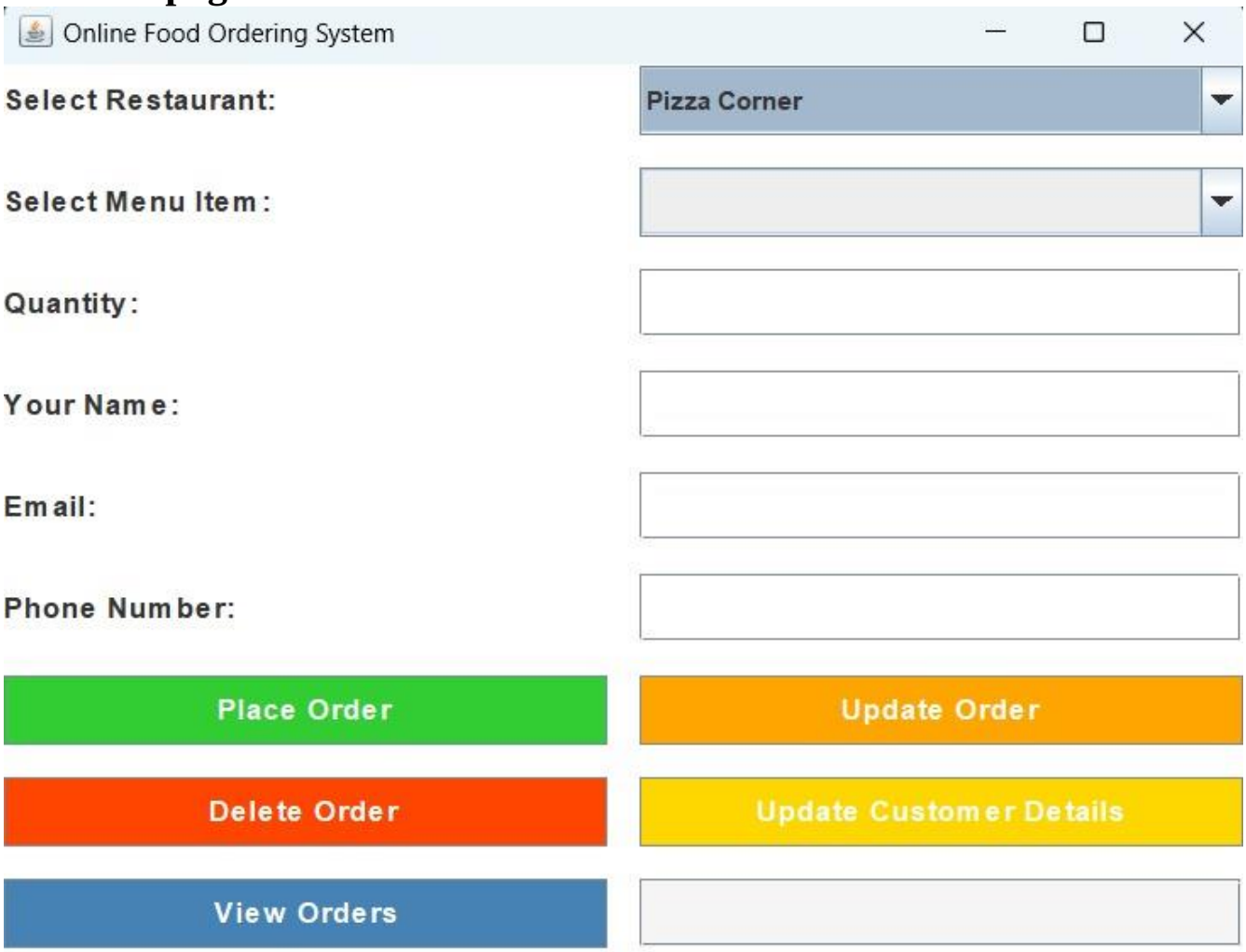
Example of SQL query to retrieve order history:

Sql:

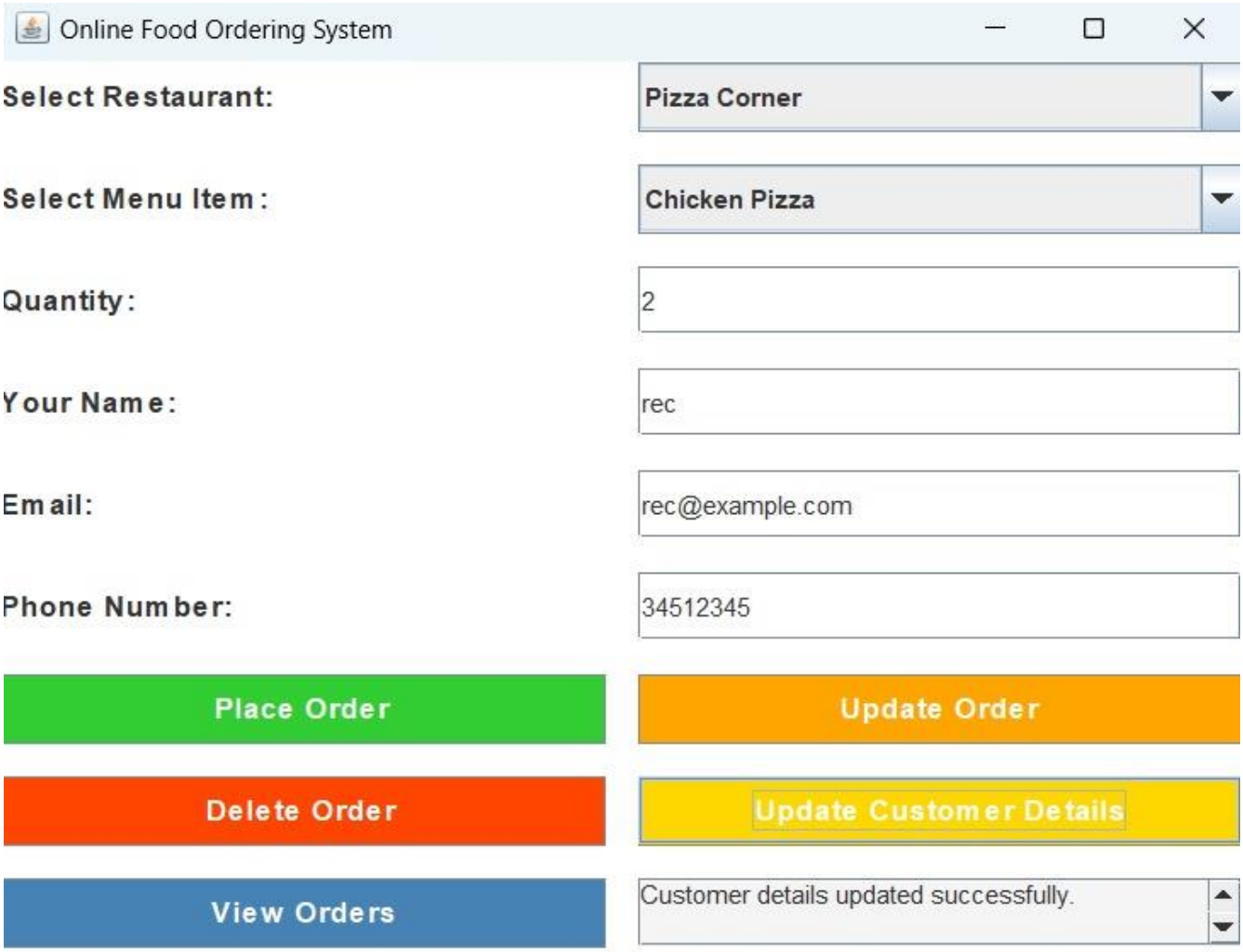
```
SELECT c.name AS customer_name, m.name AS item_name, o.quantity, o.total_price, o.order_date
FROM CustomerOrder o
JOIN Customer c ON o.customer_id = c.id
JOIN MenuItem m ON o.item_id = m.id
ORDER BY o.order_date DESC;
```

7.SNAPSHOTS

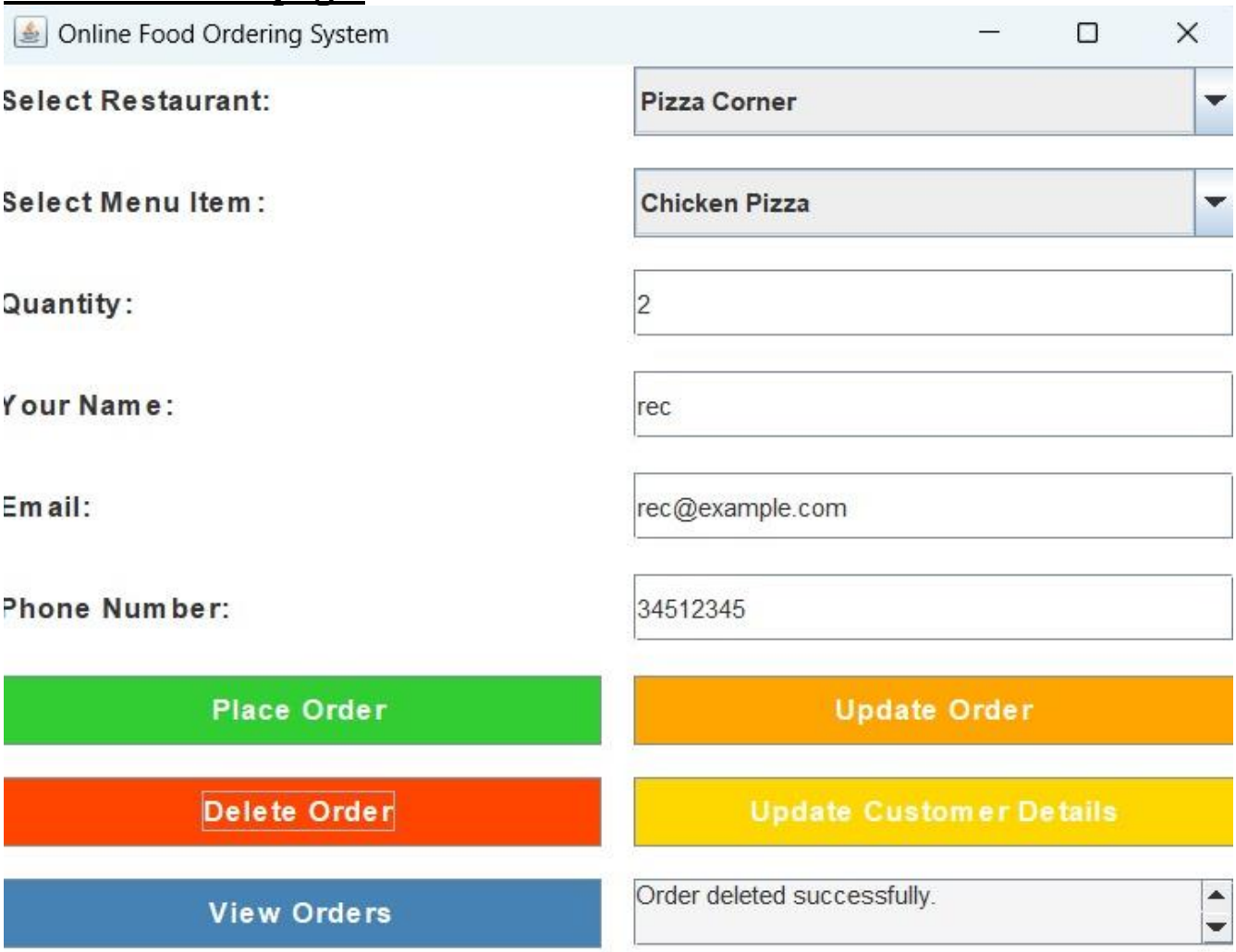
7.1Home page:



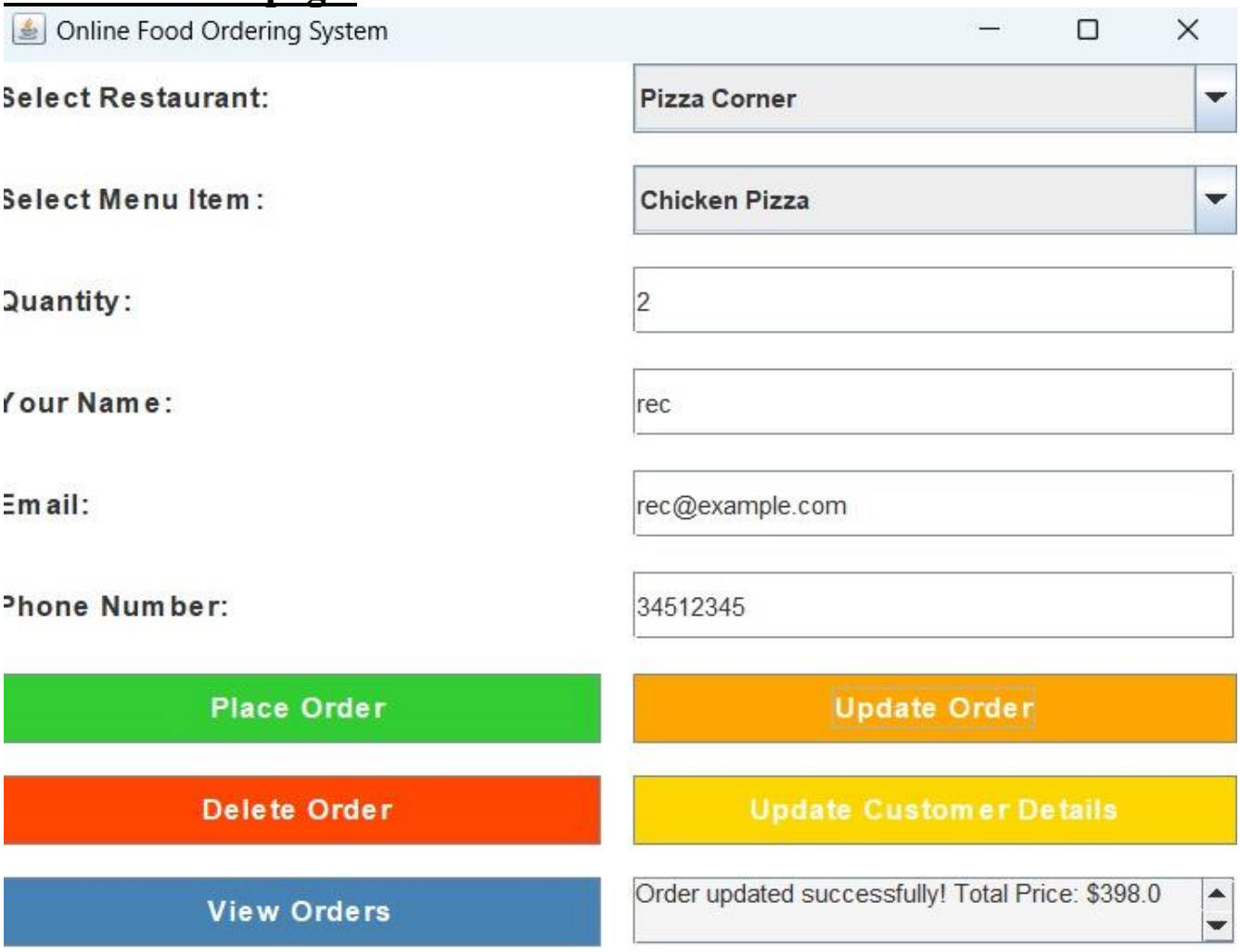
7.2Update Customer Detail page:



7.3Delete order page:



7.4View Order page:



8.Conclusion

8.1.Summary

The Food Ordering System successfully automates the food ordering process. Customers can browse restaurant menus, place orders, and view their previous orders through an intuitive interface. The system is backed by a robust database that ensures data integrity and smooth order processing.

8.2.ChallengesandFutureWork

Challenges faced during implementation included managing database connections efficiently and ensuring that the user interface is responsive. Future improvements could include adding payment integration, incorporating customer reviews, and implementing advanced filtering options for menu items**.**

9. References

- Java Documentation: https://docs.oracle.com/javase/8/docs/
- MySQL Documentation: https://dev.mysql.com/doc/
- SwingDocumentation: https://docs.oracle.com/javase/8/docs/api/javax/swing/package-summary.html

****