ONLINE FOOD ORDERING SYSTEM

A MINI-PROJECT BY:

MANIKANDAN M - 230701174
MATHAN KUMAR M - 230701181

in partial fulfillment of the award of the degree

OF

BACHELOR OF ENGINEERING

IN

COMPUTER SCIENCE AND ENGINEERING

RAJALAKSHMI ENGINEERING COLLEGE, CHENNAI

An Autonomous Institute

CHENNAI

NOVEMBER 2024

BONAFIDE CERTIFICATE

Certified that this project **"ONLINE FOOD ORDERING SYSTEM"** is the bonafide work of "**MANIKANDAN M , MATHAN KUMAR M"** who carried out the project work under my supervision.

Submitted for the practical examination held on _____

SIGNATURE
Ms.Dharshini B S
Rajalakshmi Engineering College,
Computer Science and Engineering
(Autonomous),
Thandalam,Chennai-602105

SIGNATURE

INTERNAL EXAMINER / EXTERNAL EXAMINER

ABSTRACT

This report details the design and implementation of an Online Food Ordering System built with Java and MySQL. The project features a dynamic user interface, an efficient backend for managing customer and restaurant data, and allows customers to place and view food orders. It integrates MySQL for data storage and Java Swing for the frontend, providing a seamless food ordering experience. The system includes essential features such as customer registration, restaurant selection, menu item browsing, and order management.

TABLE OF CONTENTS

Introduction

The purpose of this project is to automate the food ordering process through a digital interface. As food ordering becomes more digitized, the need for efficient, scalable, and user-friendly systems has grown. Traditional methods like phone-based ordering are being replaced by online platforms for greater efficiency and customer satisfaction. This system aims to provide a seamless interface for customers to place orders, view their orders, and interact with restaurants.

Objectives

- Provide an intuitive, easy-to-use interface for food ordering.
- Enable dynamic updates of menus based on the selected restaurant.
- Efficiently manage customer and order data in a relational database.
- Provide real-time order tracking and order history.

Literature Review

This section compares major food ordering systems like Swiggy, Uber Eats, and Zomato with the developed system. While these platforms provide similar functionalities, they are highly complex and often have a steep learning curve for new users. The system discussed in this report is simpler, focusing on customer interaction with restaurants and providing essential features like menu browsing, order placement, and tracking, without additional overheads.

Database Design:

```
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", "MANImmk@30"); // Replace with your credentials
        } catch (ClassNotFoundException | SQLException e) {
            e.printStackTrace();
        }
        return null;
    }
}
```

Food Ordering System Java Code

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

public class FoodOrderingSystem extends JFrame {

    private JComboBox<String> restaurantComboBox;
    private JComboBox<String> menuComboBox;
    private JTextField quantityField;
    private JTextField nameField, emailField, phoneField;
    private JButton orderButton;
    private JButton viewOrdersButton;
    private JButton updateOrderButton;
    private JButton deleteOrderButton;
    private JButton updateCustomerButton;
    private JTextArea outputArea;

    private int selectedRestaurantId;

    // Constructor for the Food Ordering System UI
    public FoodOrderingSystem() {
        setTitle("Online Food Ordering System");
        setSize(600, 600);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setLocationRelativeTo(null);

        // Creating the main panel with a grid layout
        JPanel panel = new JPanel();
        panel.setLayout(new GridLayout(12, 2, 15, 15));
        panel.setBackground(new Color(255, 255, 255));

        // Creating and adding components to the panel
        JLabel restaurantLabel = new JLabel("Select Restaurant:");
        restaurantLabel.setFont(new Font("Arial", Font.BOLD, 14));
        restaurantComboBox = new JComboBox<>();
        panel.add(restaurantLabel);
```

```
panel.add(restaurantComboBox);

JLabel menuLabel = new JLabel("Select Menu Item:");
menuLabel.setFont(new Font("Arial", Font.BOLD, 14));
menuComboBox = new JComboBox<>();
panel.add(menuLabel);
panel.add(menuComboBox);

JLabel quantityLabel = new JLabel("Quantity:");
quantityLabel.setFont(new Font("Arial", Font.BOLD, 14));
quantityField = new JTextField();
panel.add(quantityLabel);
panel.add(quantityField);

JLabel nameLabel = new JLabel("Your Name:");
nameLabel.setFont(new Font("Arial", Font.BOLD, 14));
nameField = new JTextField();
panel.add(nameLabel);
panel.add(nameField);

JLabel emailLabel = new JLabel("Email:");
emailLabel.setFont(new Font("Arial", Font.BOLD, 14));
emailField = new JTextField();
panel.add(emailLabel);
panel.add(emailField);

JLabel phoneLabel = new JLabel("Phone Number:");
phoneLabel.setFont(new Font("Arial", Font.BOLD, 14));
phoneField = new JTextField();
panel.add(phoneLabel);
panel.add(phoneField);

// Adding buttons for order and view orders
orderButton = new JButton("Place Order");
orderButton.setBackground(new Color(50, 205, 50)); // Green button
orderButton.setForeground(Color.WHITE);
orderButton.setFont(new Font("Arial", Font.BOLD, 14));
panel.add(orderButton);

updateOrderButton = new JButton("Update Order");
updateOrderButton.setBackground(new Color(255, 165, 0)); // Orange button
```

```
        updateOrderButton.setForeground(Color.WHITE);
        updateOrderButton.setFont(new Font("Arial", Font.BOLD, 14));
        panel.add(updateOrderButton);

        deleteOrderButton = new JButton("Delete Order");
        deleteOrderButton.setBackground(new Color(255, 69, 0)); // Red button
        deleteOrderButton.setForeground(Color.WHITE);
        deleteOrderButton.setFont(new Font("Arial", Font.BOLD, 14));
        panel.add(deleteOrderButton);

        updateCustomerButton = new JButton("Update Customer Details");
        updateCustomerButton.setBackground(new Color(255, 215, 0)); // Gold button
        updateCustomerButton.setForeground(Color.WHITE);
        updateCustomerButton.setFont(new Font("Arial", Font.BOLD, 14));
        panel.add(updateCustomerButton);

        // Button to view orders
        viewOrdersButton = new JButton("View Orders");
        viewOrdersButton.setBackground(new Color(70, 130, 180)); // SteelBlue button
        viewOrdersButton.setForeground(Color.WHITE);
        viewOrdersButton.setFont(new Font("Arial", Font.BOLD, 14));
        panel.add(viewOrdersButton);

        // TextArea to display the output (order confirmation, order details)
        outputArea = new JTextArea();
        outputArea.setEditable(false);
        outputArea.setBackground(new Color(245, 245, 245));
        panel.add(new JScrollPane(outputArea));

        // Adding the panel to the JFrame
        add(panel, BorderLayout.CENTER);

        // Loading restaurants and attaching event listeners
        loadRestaurants();

        restaurantComboBox.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                selectedRestaurantId = getRestaurantIdFromName((String)
restaurantComboBox.getSelectedItem());
                loadMenuItems(selectedRestaurantId);
            }
```

```
    });

    // Handling order placement action
    orderButton.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent e) {
            placeOrder();
        }
    });

    // Handling update order action
    updateOrderButton.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent e) {
            updateOrder();
        }
    });

    // Handling delete order action
    deleteOrderButton.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent e) {
            deleteOrder();
        }
    });

    // Handling update customer details action
    updateCustomerButton.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent e) {
            updateCustomer();
        }
    });

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

// Method to load the list of restaurants from the database
private void loadRestaurants() {
    try (Connection conn = DBConnection.connect()) {
```

```
            String sql = "SELECT id, name FROM Restaurants";
            try (Statement stmt = conn.createStatement(); ResultSet rs = stmt.executeQuery(sql))
{
                while (rs.next()) {
                    restaurantComboBox.addItem(rs.getString("name"));
                }
            }
        } catch (SQLException e) {
            e.printStackTrace();
            outputArea.append("Error loading restaurants.\n");
        }
    }

    // Method to load the menu items of a selected restaurant
    private void loadMenuItems(int restaurantId) {
        try (Connection conn = DBConnection.connect()) {
            String sql = "SELECT id, name, price FROM Menu_Items 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();
            outputArea.append("Error loading menu items.\n");
        }
    }

    // Get restaurant ID from its name
    private int getRestaurantIdFromName(String restaurantName) {
        try (Connection conn = DBConnection.connect()) {
            String sql = "SELECT id FROM Restaurants 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; // Return -1 if restaurant not found
}

// Method to place the order by inserting data into the database
private void placeOrder() {
    String selectedItem = (String) menuComboBox.getSelectedItem();
    String quantityText = quantityField.getText();

    if (selectedItem == null || quantityText.isEmpty() || !quantityText.matches("\\d+")) {
        outputArea.append("Please select a menu item and enter a valid quantity.\n");
        return;
    }

    int quantity = Integer.parseInt(quantityText);
    int customerId = insertCustomer();
    int itemId = getMenuItemIdFromName(selectedItem);
    double totalPrice = calculateTotalPrice(itemId, quantity);

    String orderDate = new java.sql.Date(System.currentTimeMillis()).toString();

    try (Connection conn = DBConnection.connect()) {
        String sql = "INSERT INTO Orders (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! Total Price: $" + totalPrice +
"\n");
        }
    } catch (SQLException e) {
```

```
            e.printStackTrace();
            outputArea.append("Error placing the order.\n");
        }
    }

    // Method to insert customer details and return the generated customer ID
    private int insertCustomer() {
        String name = nameField.getText();
        String email = emailField.getText();
        String phone = phoneField.getText();

        if (name.isEmpty() || email.isEmpty() || phone.isEmpty()) {
            outputArea.append("Please fill in all customer details.\n");
            return -1;
        }

        try (Connection conn = DBConnection.connect()) {
            String sql = "INSERT INTO Customers (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();

                try (ResultSet generatedKeys = pstmt.getGeneratedKeys()) {
                    if (generatedKeys.next()) {
                        return generatedKeys.getInt(1);
                    }
                }
            }
        } catch (SQLException e) {
            e.printStackTrace();
        }
        return -1; // Return -1 if insertion fails
    }

    // Method to get the menu item ID from its name
    private int getMenuItemIdFromName(String itemName) {
        try (Connection conn = DBConnection.connect()) {
            String sql = "SELECT id FROM Menu_Items 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;
    }

    // Method to calculate the total price based on item ID and quantity
    private double calculateTotalPrice(int itemId, int quantity) {
        double price = 0;
        try (Connection conn = DBConnection.connect()) {
            String sql = "SELECT price FROM Menu_Items 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 display all orders placed by customers
    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 Orders o " +
                         "JOIN Customers c ON o.customer_id = c.id " +
                         "JOIN Menu_Items m ON o.item_id = m.id " +
```

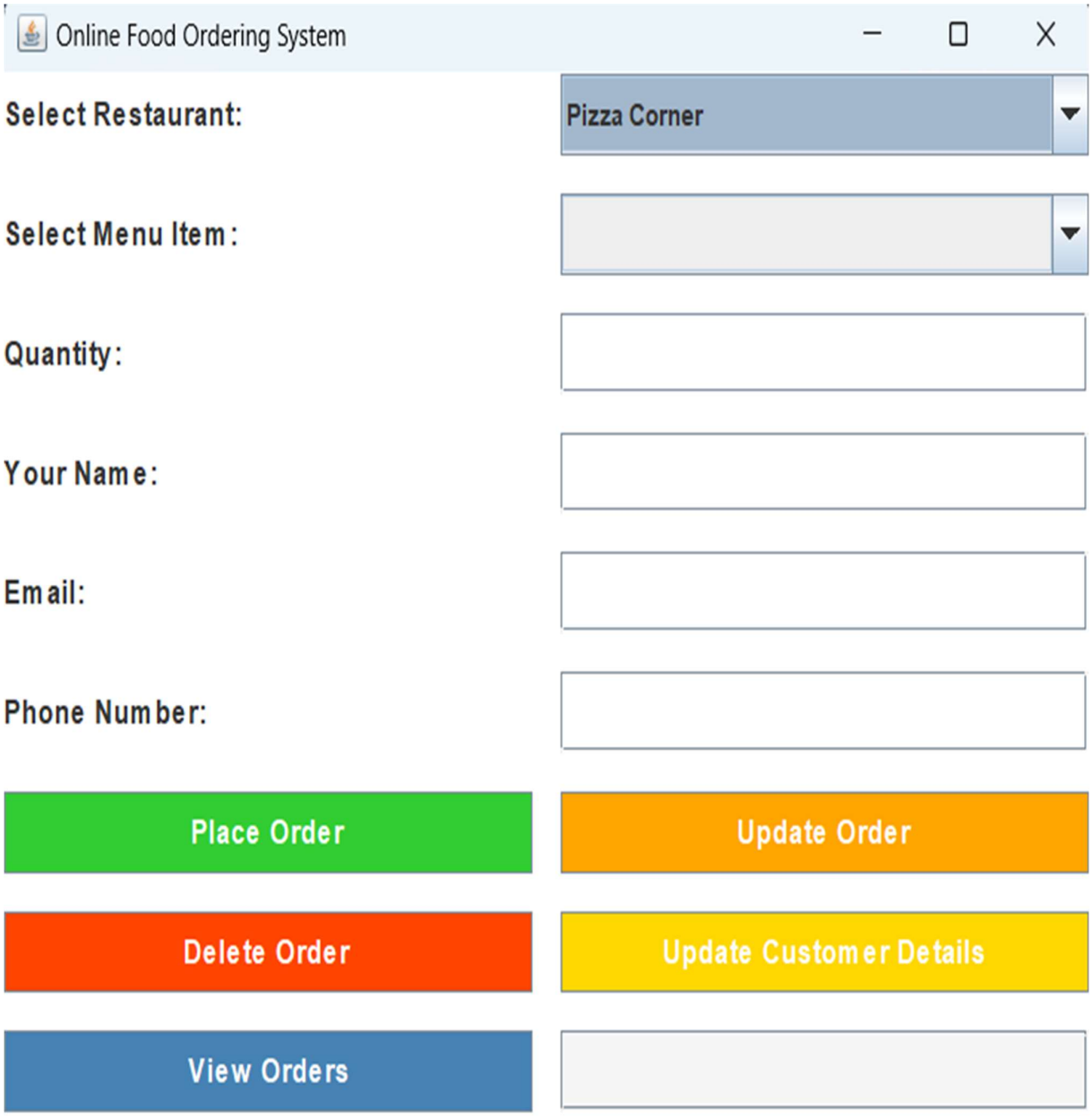
```
                "ORDER BY o.order_date DESC";

        try (Statement stmt = conn.createStatement(); ResultSet rs = stmt.executeQuery(sql))
{
            outputArea.setText(""); // Clear previous output
            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");
    }
  }

  // Method to update an existing order
  private void updateOrder() {
    String selectedItem = (String) menuComboBox.getSelectedItem();
    String quantityText = quantityField.getText();

    if (selectedItem == null || quantityText.isEmpty() || !quantityText.matches("\\d+")) {
        outputArea.append("Please select a menu item and enter a valid quantity to
update.\n");
        return;
    }

    int quantity = Integer.parseInt(quantityText);
    int itemId = getMenuItemIdFromName(selectedItem);
    double totalPrice = calculateTotalPrice(itemId, quantity);

    String orderDate = new java.sql.Date(System.currentTimeMillis()).toString();

    try (Connection conn = DBConnection.connect()) {
```

```
            String sql = "UPDATE Orders SET quantity = ?, total_price = ?, order_date = ?
WHERE item_id = ? AND customer_id = ?";
            try (PreparedStatement pstmt = conn.prepareStatement(sql)) {
                pstmt.setInt(1, quantity);
                pstmt.setDouble(2, totalPrice);
                pstmt.setString(3, orderDate);
                pstmt.setInt(4, itemId);
                pstmt.setInt(5, insertCustomer());  // Assume this updates the customer order by
customer ID
                pstmt.executeUpdate();
                outputArea.append("Order updated successfully! Total Price: $" + totalPrice +
"\n");
            }
        } catch (SQLException e) {
            e.printStackTrace();
            outputArea.append("Error updating the order.\n");
        }
    }

    // Method to delete an order
    private void deleteOrder() {
        String selectedItem = (String) menuComboBox.getSelectedItem();

        if (selectedItem == null) {
            outputArea.append("Please select a menu item to delete the order.\n");
            return;
        }

        int itemId = getMenuItemIdFromName(selectedItem);

        try (Connection conn = DBConnection.connect()) {
            String sql = "DELETE FROM Orders WHERE item_id = ? AND customer_id = ?";
            try (PreparedStatement pstmt = conn.prepareStatement(sql)) {
                pstmt.setInt(1, itemId);
                pstmt.setInt(2, insertCustomer());  // Assume customer ID deletion logic
                pstmt.executeUpdate();
                outputArea.append("Order deleted successfully.\n");
            }
        } catch (SQLException e) {
            e.printStackTrace();
            outputArea.append("Error deleting the order.\n");
```

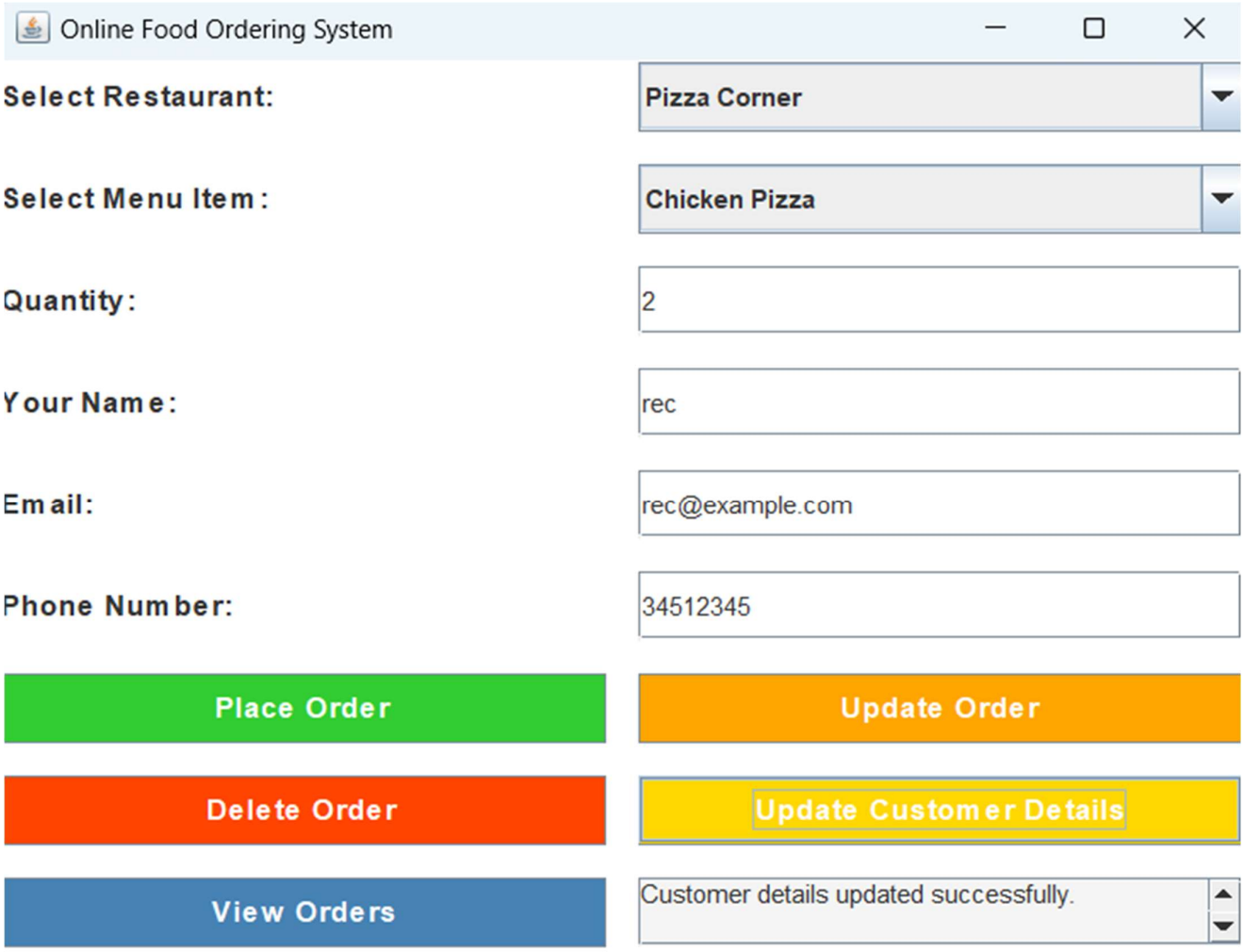
```
        }
    }

    // Method to update customer details
    private void updateCustomer() {
        String name = nameField.getText();
        String email = emailField.getText();
        String phone = phoneField.getText();

        if (name.isEmpty() || email.isEmpty() || phone.isEmpty()) {
            outputArea.append("Please fill in all customer details to update.\n");
            return;
        }

        try (Connection conn = DBConnection.connect()) {
            String sql = "UPDATE Customers SET name = ?, email = ?, phone = ? WHERE
email = ?";
            try (PreparedStatement pstmt = conn.prepareStatement(sql)) {
                pstmt.setString(1, name);
                pstmt.setString(2, email);
                pstmt.setString(3, phone);
                pstmt.setString(4, email);
                pstmt.executeUpdate();
                outputArea.append("Customer details updated successfully.\n");
            }
        } catch (SQLException e) {
            e.printStackTrace();
            outputArea.append("Error updating customer details.\n");
        }
    }

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

SNAPSHOTS

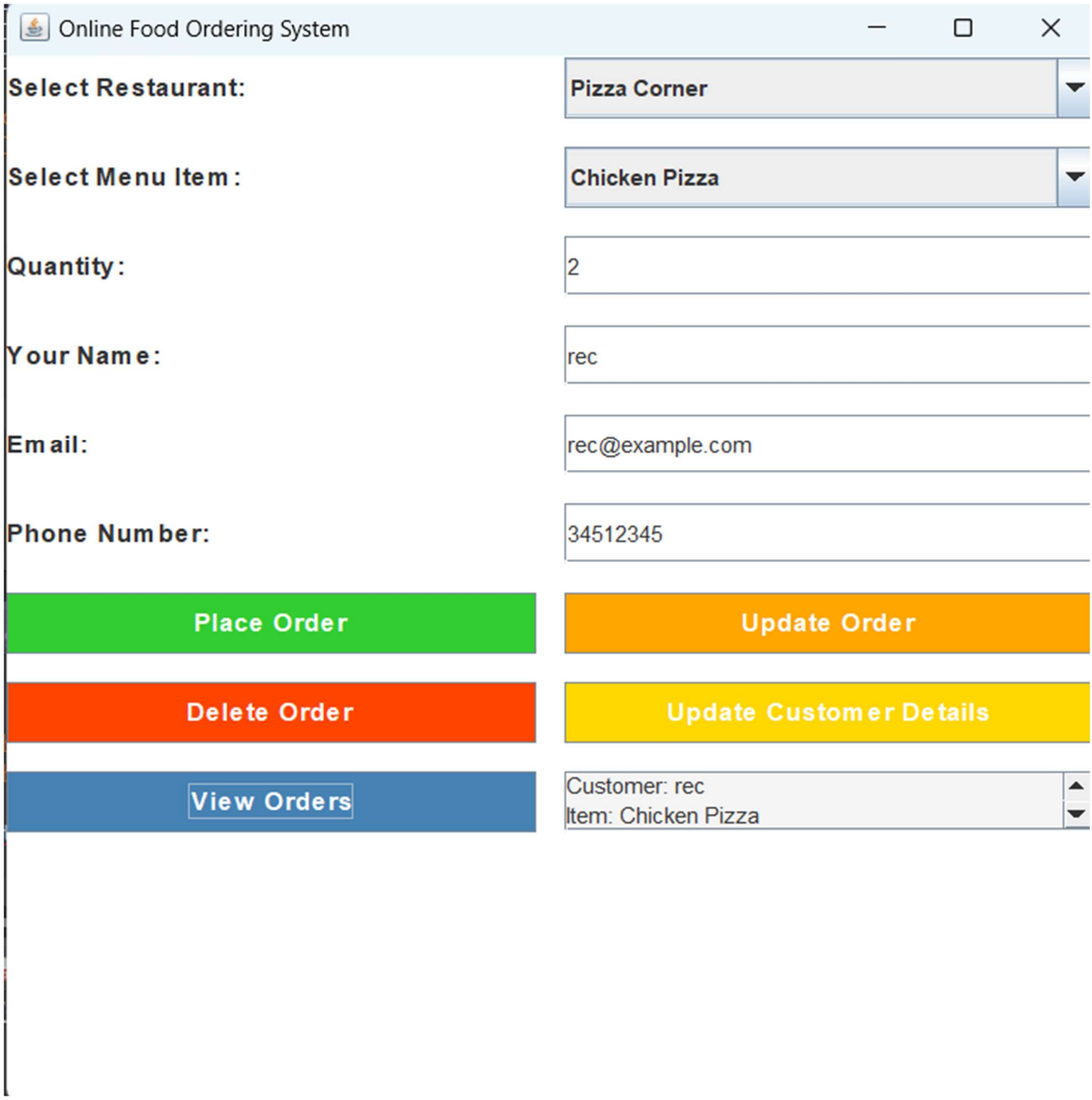
HOME PAGE



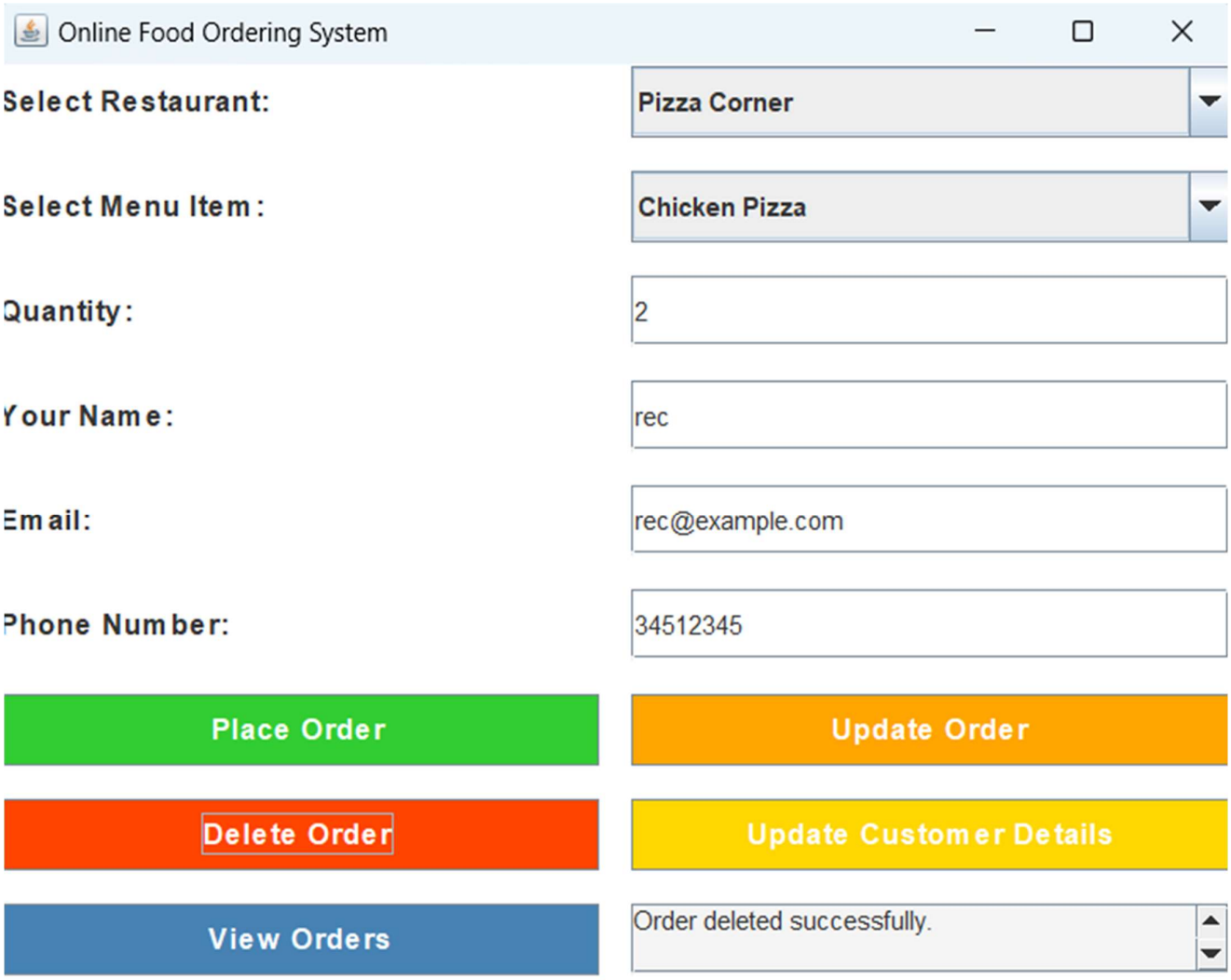
UPDATE CUSTOMER DETAILS PAGE



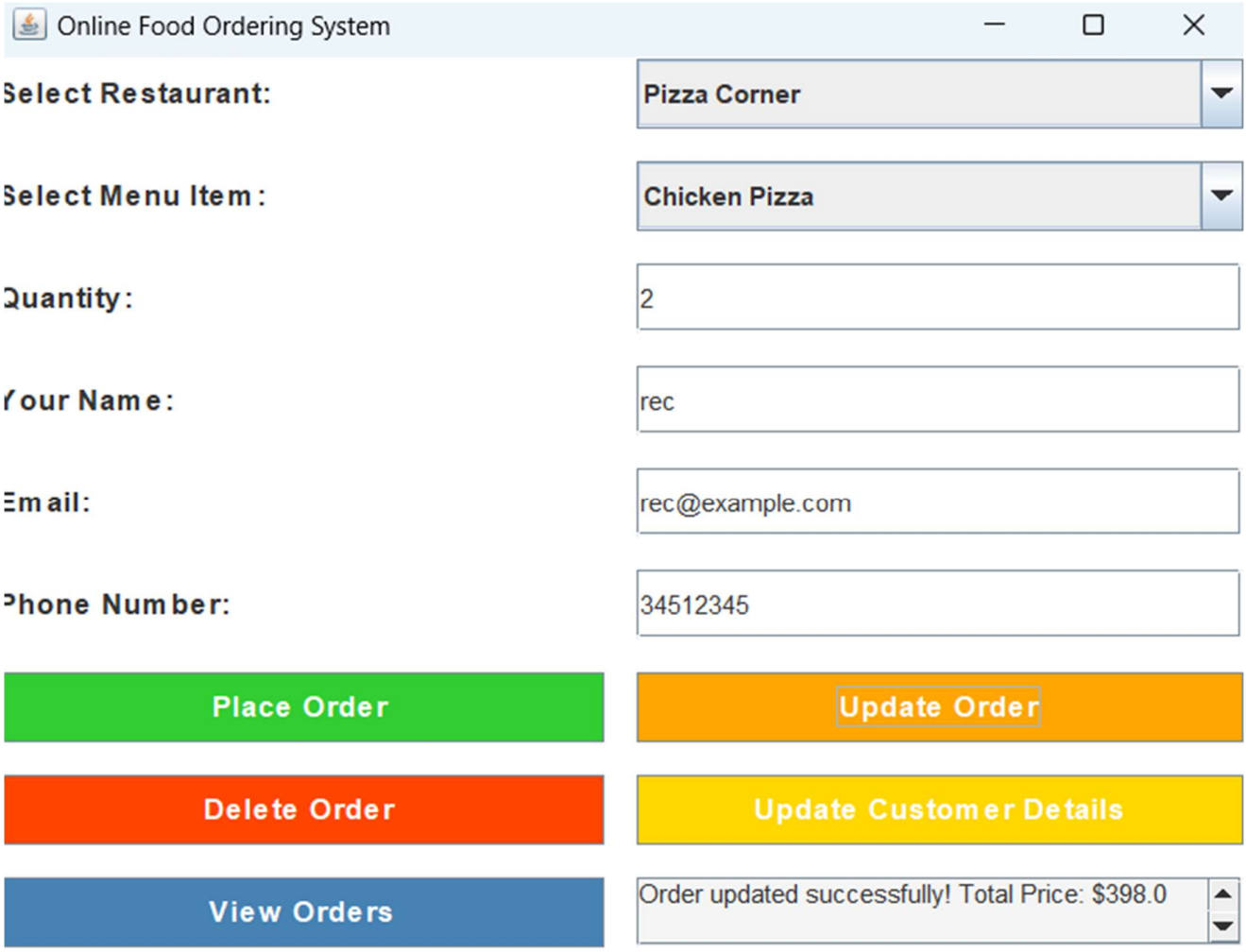
VIEW ORDER PAGE



DELETE ORDER PAGE



UPDATE ORDER PAGE



CONCLUSION

This project successfully implemented a simple and efficient food ordering system using Java and MySQL. It allows customers to place and view orders, stores order and customer data efficiently, and can be expanded with features like payment integration and web-based access.

REFERENCES

1. **https://www.javatpoint.com/java-tutorial**
2. https://www.wikipedia.org/
3. https://www.w3schools.com/sql/
4. SQL | Codecademy

****