**Mini Project Report: Online Food Ordering System**

**INTRODUCTION**

The Online Food Ordering System is a software application designed to streamline the process of ordering food online. This project aims to provide a user-friendly interface for customers to view a menu, select items, and place an order. The system also includes features such as order tracking and order history.

**ABOUT THE SYSTEM**

**EXISTING SYSTEM**

Traditional food ordering involves phone calls or visiting the restaurant in person. This process can be time-consuming and may lead to miscommunication. The existing system lacks the efficiency and convenience provided by an online platform.

**NEED FOR NEW SYSTEM**

The need for a new system arises from the increasing demand for online services. The new system offers advantages such as:

Convenience: Customers can browse the menu and place orders from the comfort of their homes.

Efficiency: Orders are processed electronically, reducing the chances of errors in communication.

Order History: Customers can track their order history and easily reorder their favorite items.

**PROPOSED SYSTEM**

The proposed system is an Online Food Ordering System that addresses the shortcomings of the existing system. It provides a user-friendly interface for customers to view the menu, place orders, and track their order status. The system also includes features for registering users and maintaining their order history.

**HARDWARE AND SOFTWARE REQUIREMENTS**

Hardware Requirements:

A computer or server to host the application.

Internet connectivity for online order processing.

Software Requirements:

Java Development Kit (JDK)

Java Swing Library

Integrated Development Environment (IDE) such as Eclipse or IntelliJ

Database system (optional for storing order history)

**MODULE DESCRIPTION**

**USER MODULE**

The User Module allows customers to interact with the system. It includes features such as viewing the menu, placing orders, and tracking order status.

**REGISTER MODULE**

The Register Module enables users to create accounts. Registered users can benefit from order history tracking and personalized services.

**RANKLIST GENERATION MODULE**

This module is not explicitly implemented in the current code. It could be extended to generate a ranklist of popular food items based on user preferences and order frequency.

**CHOICE FILLING MODULE**

The Choice Filling Module is responsible for handling the selection of food items by customers during the order placement process.

**AVAILABLE SEAT MODULE**

This module is not applicable to the current system, as it is specific to restaurant reservations rather than food ordering.

**SAMPLE CODING**

The provided Java code demonstrates a simple implementation of an online food ordering system with a graphical user interface (GUI) using Java Swing.

import javax.swing.\*;

import java.awt.\*;

import java.awt.event.ActionEvent;

import java.awt.event.ActionListener;

import java.util.ArrayList;

import java.util.HashMap;

import java.util.List;

import java.util.Map;

class Food {

private String name;

private double price;

public Food(String name, double price) {

this.name = name;

this.price = price;

}

public String getName() {

return name;

}

public double getPrice() {

return price;

}

}

class Order {

private Map<Food, Integer> items;

public Order() {

this.items = new HashMap<>();

}

public void addItem(Food food, int quantity) {

items.put(food, items.getOrDefault(food, 0) + quantity);

}

public Map<Food, Integer>getItems() {

return items;

}

public double calculateTotal() {

return items.entrySet().stream()

.mapToDouble(entry ->entry.getKey().getPrice() \* entry.getValue())

.sum();

}

}

class Customer {

private String name;

private Order currentOrder;

public Customer(String name) {

this.name = name;

this.currentOrder = new Order();

}

public Order getCurrentOrder() {

return currentOrder;

}

public String getName() {

return name;

}

public void placeOrder(Order currentOrder2) {

}

}

class FoodOrderingWindow extends JFrame {

private List<Food> menu;

private Customer customer;

private JTextAreaorderTextArea;

public FoodOrderingWindow() {

this.menu = new ArrayList<>();

this.customer = new Customer("John Doe");

// Adding items to the menu

menu.add(new Food("Burger", 5.99));

menu.add(new Food("Pizza", 8.99));

menu.add(new Food("Salad", 4.99));

// Initialize GUI components

setTitle("Food Ordering System");

setSize(400, 300);

setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

createComponents();

// Set the window to be visible

setVisible(true);

}

private void createComponents() {

// Create a panel for the menu

JPanelmenuPanel = new JPanel();

menuPanel.setLayout(new GridLayout(menu.size(), 1));

for (Food food : menu) {

JButtonaddButton = new JButton(food.getName() + " - $" + food.getPrice());

addButton.addActionListener(new ActionListener() {

@Override

public void actionPerformed(ActionEvent e) {

// Add the selected food item to the current order

customer.getCurrentOrder().addItem(food, 1);

updateOrderTextArea();

}

});

menuPanel.add(addButton);

}

// Create a panel for the order and total

JPanelorderPanel = new JPanel();

orderPanel.setLayout(new BorderLayout());

orderTextArea = new JTextArea();

orderTextArea.setEditable(false);

orderPanel.add(new JScrollPane(orderTextArea), BorderLayout.CENTER);

JButtonplaceOrderButton = new JButton("Place Order");

placeOrderButton.addActionListener(new ActionListener() {

@Override

public void actionPerformed(ActionEvent e) {

// Place the current order

customer.placeOrder(customer.getCurrentOrder());

updateOrderTextArea();

JOptionPane.showMessageDialog(FoodOrderingWindow.this, "Order placed successfully!");

}

});

orderPanel.add(placeOrderButton, BorderLayout.SOUTH);

// Create a main panel to hold the menu and order panels

JPanelmainPanel = new JPanel();

mainPanel.setLayout(new GridLayout(1, 2));

mainPanel.add(menuPanel);

mainPanel.add(orderPanel);

// Add the main panel to the content pane

getContentPane().add(mainPanel);

}

private void updateOrderTextArea() {

// Update the text area with the current order details

Order currentOrder = customer.getCurrentOrder();

StringBuilder orderText = new StringBuilder("Current Order:\n");

for (Map.Entry<Food, Integer>entry :currentOrder.getItems().entrySet()) {

orderText.append(entry.getKey().getName()).append(" x").append(entry.getValue()).append("\n");

}

orderText.append("Total: $").append(currentOrder.calculateTotal());

orderTextArea.setText(orderText.toString());

}

}

public class FoodOrderingApp {

public static void main(String[] args) {

SwingUtilities.invokeLater(new Runnable() {

@Override

public void run() {

new FoodOrderingWindow();

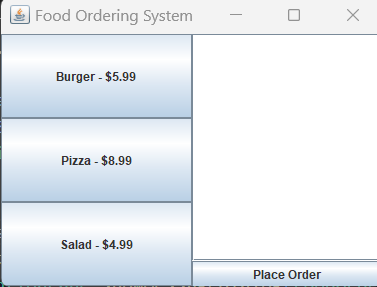
}

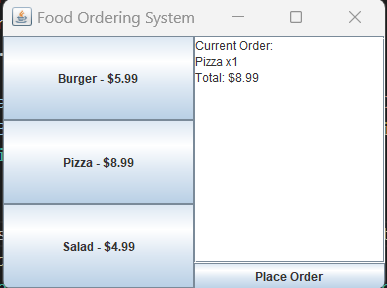
});

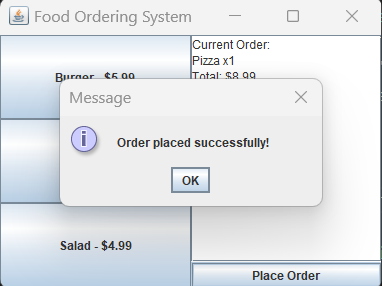
}

}

**OUTPUT SCREEN:**







**CONCLUSION**

The Online Food Ordering System provides an efficient and convenient solution for customers to order food online. It enhances the overall user experience, improves order accuracy, and provides a platform for personalized services.

**REFERENCES**

Java Swing Documentation

Java Platform, Standard Edition Documentation

https://github.com/VICKY-0017/JAVALAB/commit/a36c2af31f699f97984d0c48a14fb6a70ed606c2

**RESULT**

This project report provides an overview of the Online Food Ordering System, including its objectives, features, and technical requirements. The sample code demonstrates a basic implementation of the system, and further enhancements can be made to meet specific project requirements.