Karachi Institute of Economics and Technology

Course: Software Requirement　　Program: BS-CS, AI, CySec
Faculty: Jaweria Asif　　Student ID:
Class ID: 119005
Assignment 2: FALL-2025　　Date: 23rd-11-2025
Total Marks: 10　　Date of Submission: 30th-11-2025

Software Requirement Engineering Lab Assignment 2　　Total Marks: 10

Instructions:

Diagrams must be created using StarUML only, Coding is required in this assignment.

Case Study: Online Food Delivery & Restaurant Management System

Description:

A system where customers can order food online from multiple restaurants, track delivery in real-time, make payments, and rate food. Restaurants can manage menus, process orders, and assign delivery personnel. Delivery personnel can receive delivery requests and update status. The system should handle multiple users simultaneously, maintain order history, and ensure secure payment processing.

Question # 01: Sequence Diagram & Requirements　　(3 Marks)

a) Write **functional** and **non-functional requirement** for "Process Payment".

b) Draw a **Sequence Diagram** for "Place Order and Process Payment" including Customer, Restaurant, and Payment Gateway.

Question # 02: SRS Document in IEEE Format　　(2 Marks)

Prepare a mini **SRS document** for Online Food Delivery System including:

- Introduction
- Overall Description
- Functional Requirements
- Non-Functional Requirements

Question # 03: Class Diagram　　(2 Marks)

Draw a **Class Diagram** including at least 5 classes (Customer, Order, Restaurant, Menu, Delivery) with attributes and methods.

Question # 04: Coding & Git Update　　(2 Marks)

a) Implement a simple **console-based program** for placing and tracking orders using **Python or C#**.

b) Push all diagrams, SRS document, and code to a **GitHub repository**. Share the repository link

Question # 05: Security Analysis – Vulnerability Assessment　　(1 Mark)

a) Identify **at least 3 potential security vulnerabilities** in an online food delivery system (e.g., data leaks, payment fraud, and weak authentication).

b) Suggest **basic mitigation measures** for each vulnerability.

Submission Requirements:

- StarUML **must be used** for all diagrams (Use Case, Activity, Sequence, Class, and Collaboration).
- Coding **must be in Python or C# only**. No other programming languages allowed.
- Security analysis is **theoretical only**; students should not attempt hacking.
- **GitHub submission is mandatory**, include all diagrams, SRS, and code.
- Assignment 1 must be submitted on the LMS in PDF format by the specified deadline.

Question 01: Sequence Diagram & Requirements

a) Functional and Non-Functional Requirements for “Process Payment”

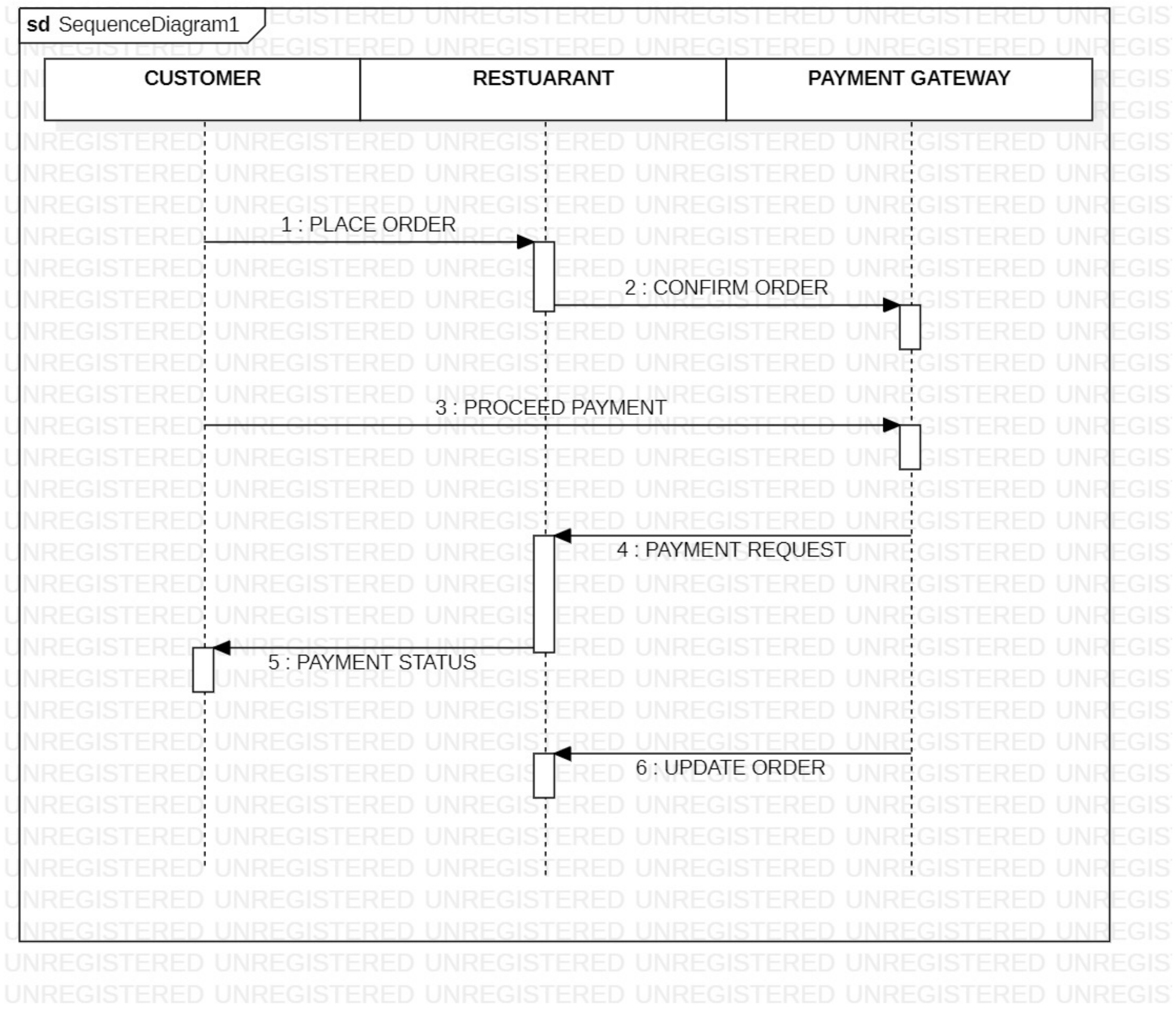
Functional Requirements:

1. System should accept multiple payment methods (credit/debit cards, online banking, e-wallets).
2. System should securely process payments.
3. System should update order status after successful payment confirmation.
4. System should send payment confirmation notifications to the customer.

Non-Functional Requirements:

1. **Security:** Payment data must be encrypted and securely transmitted.
2. **Reliability:** Payment processing must be consistent with minimal errors.
3. **Performance:** Payment process should complete within 5–10 seconds under normal load.

b) Sequence Diagram for “Place Order and Process Payment”



Question 02: SRS Document

1. Introduction

The system allows customers to order food online, track delivery in real-time, make payments, and rate food. Restaurants can manage menus, process orders, and assign delivery personnel. Delivery personnel can accept delivery requests and update order status.

2. Overall Description

- **Users:** Customers, Restaurants, Delivery Personnel.
- **Customer:** Browse menus, place orders, track delivery, make payments.
- **Restaurant:** Manage menus, process orders, assign delivery personnel.
- **Delivery Personnel:** Receive delivery requests, update delivery status.

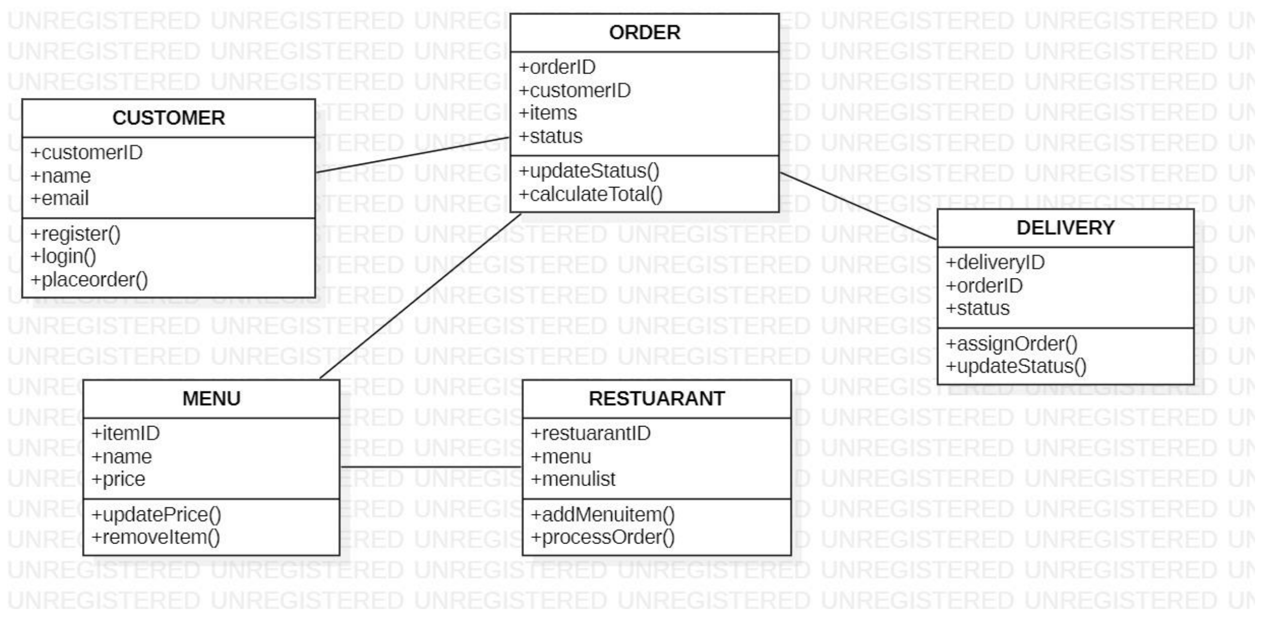
3. Functional Requirements

- Customer: Register/login, browse menus, place orders, track delivery, make payments.
- Restaurant: Manage menus, process orders, assign delivery personnel.
- Delivery Personnel: Receive delivery requests, update delivery status.
- Payment: Process multiple payment methods, update order status after payment.

4. Non-Functional Requirements

- **Performance:** Handle multiple users simultaneously.
- **Security:** Secure payment processing.
- **Usability:** Easy to use for all users.
- **Reliability:** Maintain order history accurately.

Question 03: Class Diagram



Question # 04: Coding & Git Update (2 Marks)

a) Implement a simple **console-based program** for placing and tracking orders using **Python or C#**.

```
class Customer:
    customer_counter = 1
    def __init__(self, name, email):
        self.customerID = Customer.customer_counter
        Customer.customer_counter += 1
        self.name = name
        self.email = email
        self.orders = []

    def register(self):
        print(f"{self.name} registered with ID {self.customerID}")

    def login(self):
        print(f"{self.name} logged in successfully")

    def placeOrder(self, order):
        self.orders.append(order)
        print(f"Order {order.orderID} placed by {self.name}")

class Order:
    order_counter = 1
    def __init__(self, customerID, items):
        self.orderID = Order.order_counter
        Order.order_counter += 1
        self.customerID = customerID
        self.items = items
        self.status = "Pending"

    def updateStatus(self, status):
        self.status = status
        print(f"Order {self.orderID} status updated to {self.status}")

    def calculateTotal(self):
        total = sum(item.price for item in self.items)
        print(f"Total for Order {self.orderID} is ${total}")
        return total

class Restaurant:
    restaurant_counter = 1
    def __init__(self, name):
        self.restaurantID = Restaurant.restaurant_counter
        Restaurant.restaurant_counter += 1
        self.name = name
        self.menuList = []

    def addMenuItem(self, menu):
        self.menuList.append(menu)
        print(f"{menu.name} added to {self.name}'s menu")

    def processOrder(self, order):
        order.updateStatus("Processing")
        print(f"{self.name} is processing Order {order.orderID}")

class Menu:
    item_counter = 1
    def __init__(self, name, price):
        self.itemID = Menu.item_counter
        Menu.item_counter += 1
        self.name = name
        self.price = price

    def updatePrice(self, price):
        self.price = price
        print(f"{self.name} price updated to ${self.price}")

    def removeItem(self):
        print(f"{self.name} removed from menu")

class Delivery:
    delivery_counter = 1
    def __init__(self, order):
        self.deliveryID = Delivery.delivery_counter
        Delivery.delivery_counter += 1
        self.orderID = order.orderID
        self.order = order
        self.status = "Not Assigned"

    def assignOrder(self):
        self.status = "Assigned"
        print(f"Delivery {self.deliveryID} assigned to Order {self.orderID}")

    def updateStatus(self, status):
        self.status = status
        self.order.updateStatus(status)
        print(f"Delivery {self.deliveryID} status updated to {self.status}")
```

b) Push all diagrams, SRS document, and code to a GitHub repository. Share the repository link

https://github.com/interludesher/SRE-LAB-BOTH-ASSIGNMENTS-

Question 05: Security Analysis – Vulnerability Assessment

Vulnerability	Mitigation Measure
Data Leaks	Encrypt sensitive data in transit & at rest
Payment Fraud	Use secure payment gateways (PCI DSS compliant)
Weak Authentication	Enforce strong passwords and 2FA