SRE LAB ASSIGNMENT 2

Name : Ubaid Zameer

St-Id: 65939

Class Id :119005

(QUESTION 1)

FUNCTIONAL REQUIREMENTS:

- The system should let the customer pick how they want to pay (Credit/Debit Card, Wallet, or Cash on Delivery).

- The system should check the payment information to make sure it's correct before completing the order.

- The system should connect with the payment service to approve the payment.

- If the payment goes through, the system should mark the order as "Paid."

- The system should inform the customer whether the payment was successful or if there was a problem.

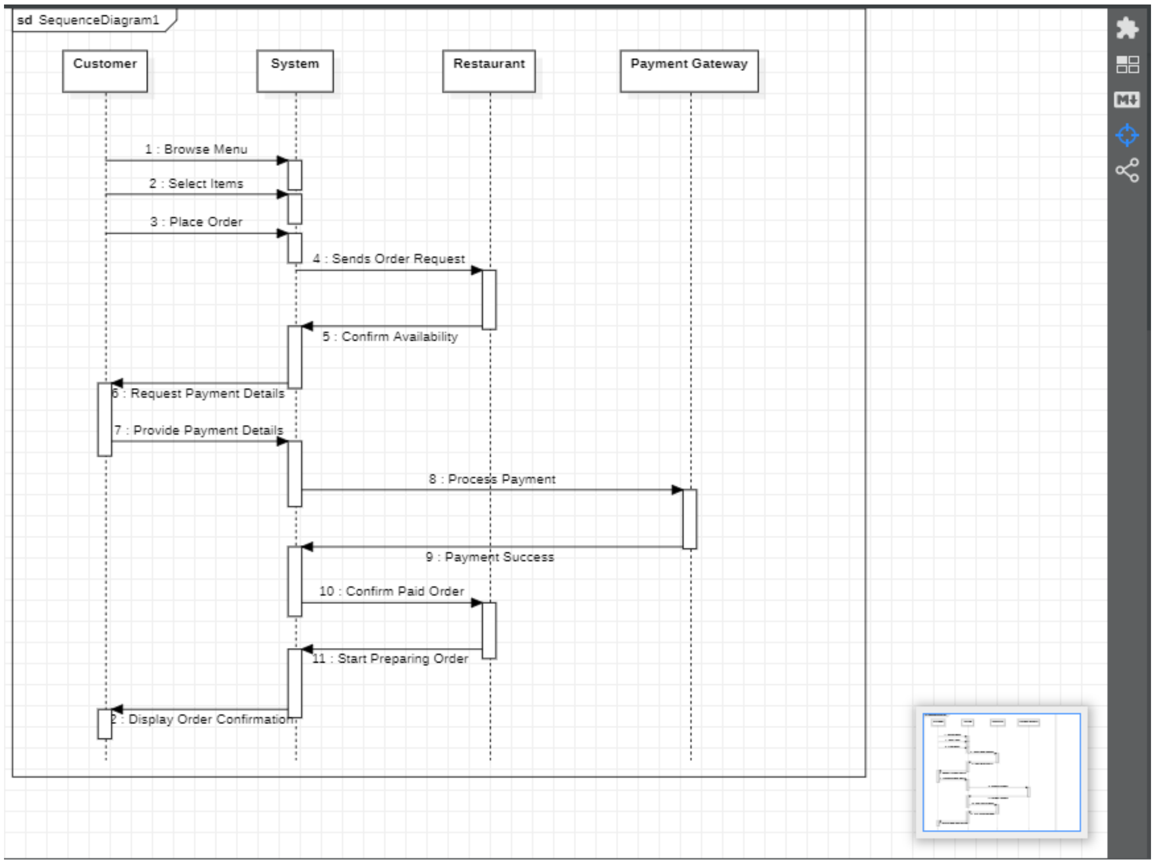
- The system should allow retrying payment if the first attempt fails.

- The system should keep a record of all payment attempts for reference.

NON-FUNCTIONAL REQUIREMENTS:

- **Security:** All payment details must be safely encrypted so that customer information stays private.
- **Speed:** Payments should go through quickly, ideally within 3–5 seconds, so customers aren't waiting.
- **Reliability:** The payment system should be up and running almost all the time (at least 99% availability).
- **Error Handling:** If a payment fails or the system times out, it should automatically try again and let the customer know what happened.

- **Ease of Use:** The payment page should be simple and easy to understand, with no confusing steps.
- **Instant Feedback:** Customers should immediately know if their payment was successful.
- **Flexibility:** The system should handle different payment options like cards, wallets, or cash on delivery without any hassle.
- **Monitoring:** Any failed payments or errors should be logged so the team can check and fix issues quickly.
- **Scalability:** The system should handle many payments at the same time without slowing down or crashing.



(QUESTION 2)

Mini SRS Document (IEEE Format)

1. Introduction

The **Online Food Delivery System** allows customers to order food from their favorite restaurants online, restaurants to manage their menus and process orders efficiently, and delivery personnel to deliver food on time. The system ensures **secure payments, real-time order tracking, and smooth communication** between all users. It aims to make ordering food convenient, fast, and reliable.

2. Overall Description

- **Users:**
  - **Customers** – place orders, make payments, track delivery, give reviews.
  - **Restaurants** – manage menus, accept orders, prepare food.
  - **Delivery Personnel** – pick up orders, update delivery status, deliver food.
  - **Admin** – monitor the system, manage users, handle issues.
- **Features:**
  - Browse menus from multiple restaurants.
  - Place orders online and choose payment methods.
  - Real-time tracking of order status.
  - Secure payment processing.
  - Rate and review restaurants.
  - Notifications for order updates.
- **Constraints:**
  - Must support multiple users at the same time (concurrency).
  - Payment and personal data must be secure.
  - System should always be available with minimal downtime.

3. Functional Requirements

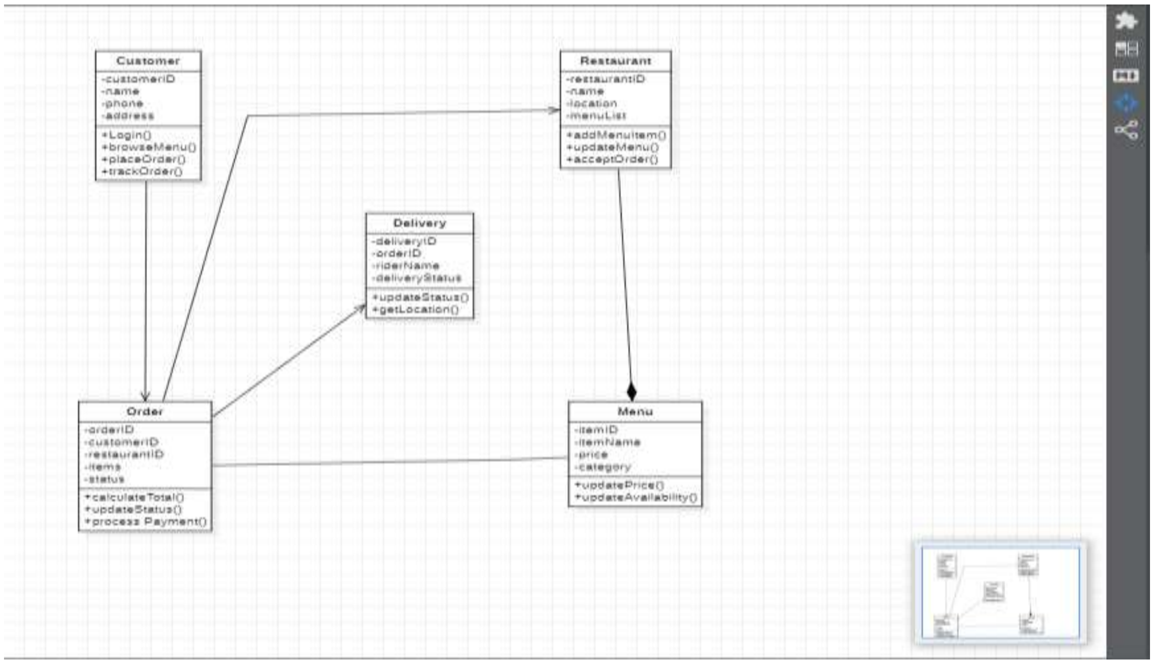
1. Users can **register and log in** to the system.
2. Customers can **browse menus and place orders** from one or more restaurants.
3. Restaurants can **update menu items and availability** in real time.
4. Customers can **pay online securely** using multiple options (card, wallet, or cash on delivery).
5. Delivery personnel can **update the delivery status** (Picked → On the way → Delivered).

6. Customers can **track their orders in real time** on the app or website.
7. The system sends **notifications** to customers about order confirmation, preparation, and delivery.
8. Customers can **rate and review** their orders and restaurants.

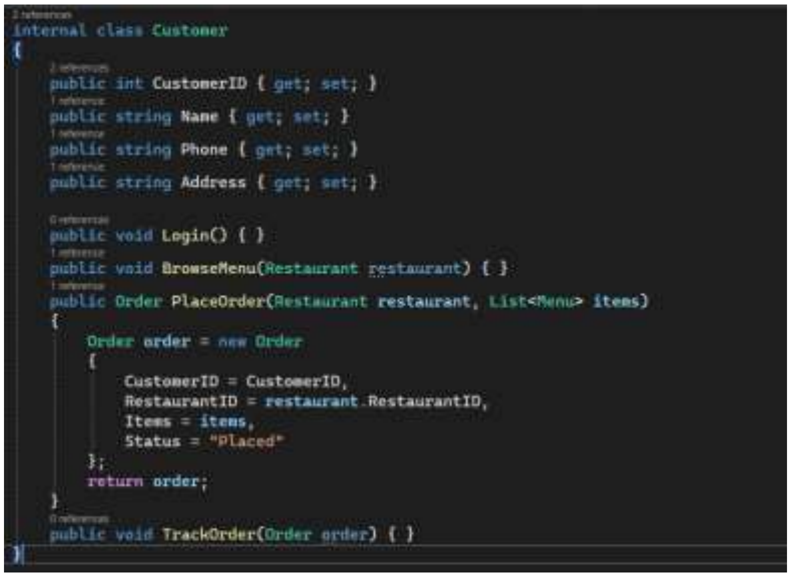
4. Non-Functional Requirements

1. **Performance:** Pages should load quickly (within 2 seconds).
2. **Security:** All sensitive information like passwords and payment details must be encrypted.
3. **Reliability:** The system should be up and running at least **99% of the time**.
4. **Scalability:** Must handle thousands of users and orders simultaneously without slowing down.
5. **Usability:** The interface should be **simple, easy to navigate, and mobile-friendly**.
6. **Feedback:** Users should receive clear messages for successful actions or errors.
7. **Error Handling:** Failed payments or other issues should be handled gracefully with instructions.
8. **Maintainability:** The system should be easy to update and maintain by developers.

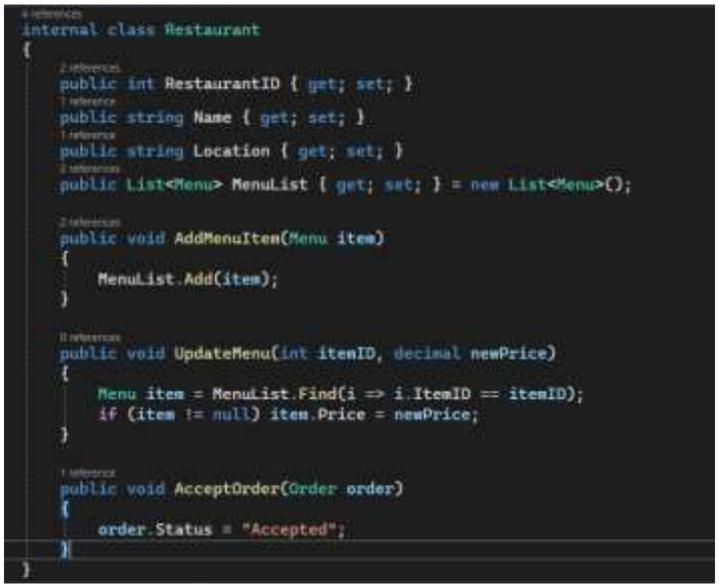
(QUESTION 3)



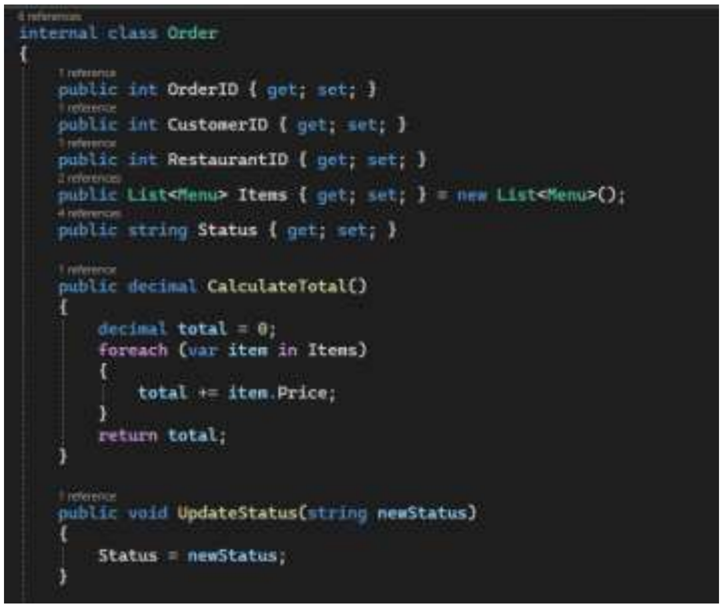
(QUESTION 4)



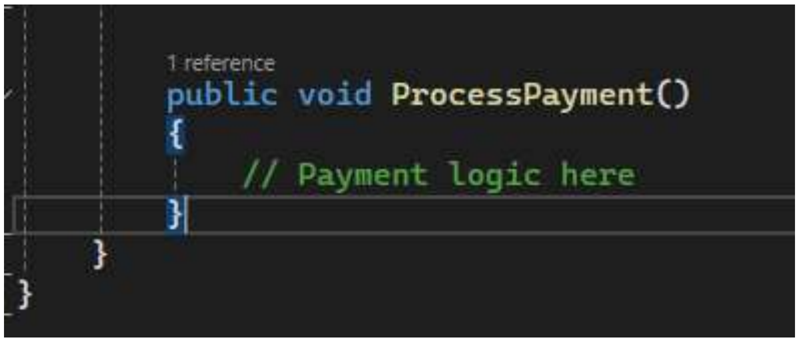
```
2 references
internal class Customer
{
    2 references
    public int CustomerID { get; set; }
    1 reference
    public string Name { get; set; }
    1 reference
    public string Phone { get; set; }
    1 reference
    public string Address { get; set; }

    0 references
    public void Login() { }
    1 reference
    public void BrowseMenu(Restaurant restaurant) { }
    1 reference
    public Order PlaceOrder(Restaurant restaurant, List<Menu> items)
    {
        Order order = new Order
        {
            CustomerID = CustomerID,
            RestaurantID = restaurant.RestaurantID,
            Items = items,
            Status = "Placed"
        };
        return order;
    }
    0 references
    public void TrackOrder(Order order) { }
}
```



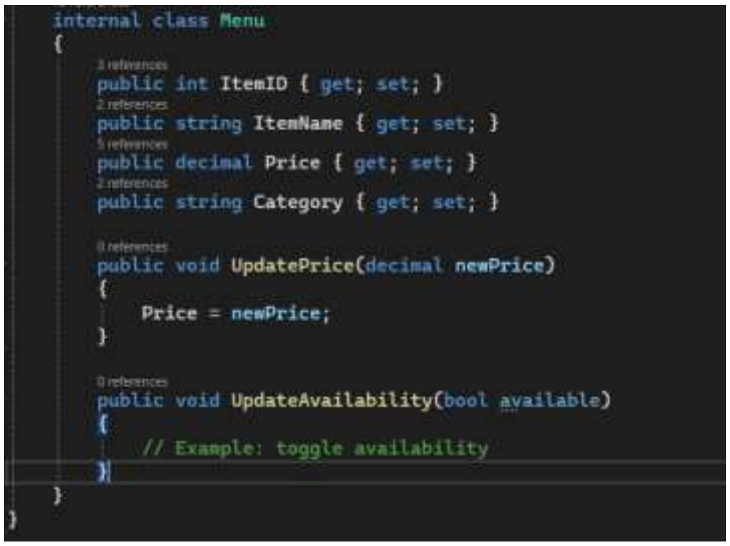
```
4 references
internal class Restaurant
{
    2 references
    public int RestaurantID { get; set; }
    1 reference
    public string Name { get; set; }
    1 reference
    public string Location { get; set; }
    2 references
    public List<Menu> MenuList { get; set; } = new List<Menu>();

    2 references
    public void AddMenuItem(Menu item)
    {
        MenuList.Add(item);
    }

    0 references
    public void UpdateMenu(int itemID, decimal newPrice)
    {
        Menu item = MenuList.Find(i => i.ItemID == itemID);
        if (item != null) item.Price = newPrice;
    }

    1 reference
    public void AcceptOrder(Order order)
    {
        order.Status = "Accepted";
    }
}
```



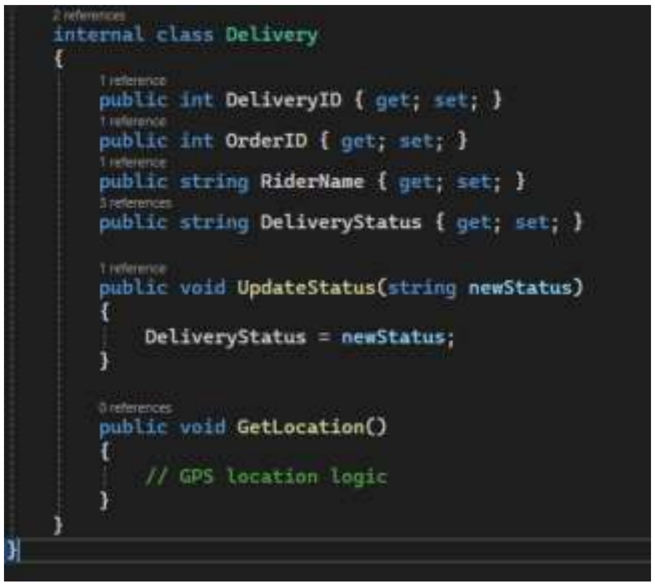
```
6 references
internal class Order
{
    1 reference
    public int OrderID { get; set; }
    1 reference
    public int CustomerID { get; set; }
    1 reference
    public int RestaurantID { get; set; }
    2 references
    public List<Menu> Items { get; set; } = new List<Menu>();
    4 references
    public string Status { get; set; }

    1 reference
    public decimal CalculateTotal()
    {
        decimal total = 0;
        foreach (var item in Items)
        {
            total += item.Price;
        }
        return total;
    }

    1 reference
    public void UpdateStatus(string newStatus)
    {
        Status = newStatus;
    }
```



```
        1 reference
        public void ProcessPayment()
        {
            // Payment logic here
        }
    }
}
```



```
internal class Menu
{
    3 references
    public int ItemID { get; set; }
    2 references
    public string ItemName { get; set; }
    5 references
    public decimal Price { get; set; }
    2 references
    public string Category { get; set; }

    0 references
    public void UpdatePrice(decimal newPrice)
    {
        Price = newPrice;
    }

    0 references
    public void UpdateAvailability(bool available)
    {
        // Example: toggle availability
    }
}
}
```



```
2 references
internal class Delivery
{
    1 reference
    public int DeliveryID { get; set; }
    1 reference
    public int OrderID { get; set; }
    1 reference
    public string RiderName { get; set; }
    3 references
    public string DeliveryStatus { get; set; }

    1 reference
    public void UpdateStatus(string newStatus)
    {
        DeliveryStatus = newStatus;
    }

    0 references
    public void GetLocation()
    {
        // GPS location logic
    }
}
}
```

(GITHUB LINK)

https://github.com/ubaidkz/SRE-LAB-ASSIGNMENTS

(QUESTION 5)

Three Security Vulnerabilities:

- Weak Authentication
  - Hackers can gain unauthorized access to customer accounts.
- Payment Fraud
  - Interception of payment data during transmission.
- Data Leakage
  - Customer info (addresses, phone numbers) may leak due to insecure database storage.

Mitigation Measures:

- Weak Authentication →Use multi-factor authentication (OTP, email verification).
- Payment Fraud →Implement SSL/TLS encryption + PCI-DSS compliant payment gateway.
- Data Leakage →Encrypt all data at rest and implement strict role-based access control.