Name: Muhammad Afnan Nadeem
SID: 65964
CID: 119005
Course: SRE-LAB

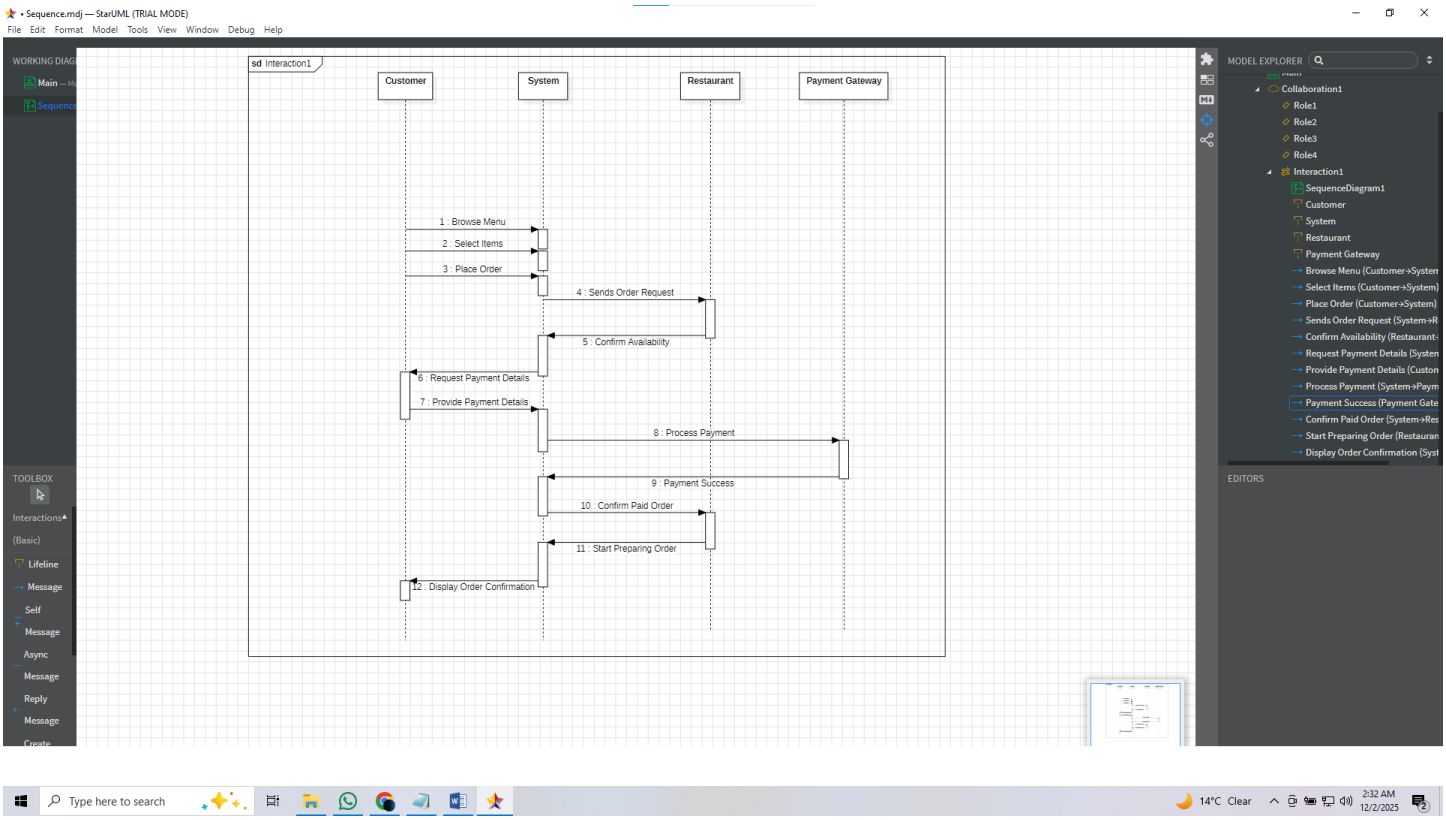
Assignment # 02

Question # 01: Requirements & Challenges:

a: **Functional requirements** and **Non-functional requirements**

Type	Requirements
Functional Requirements	➢ The system should enable customers to select a payment option (Credit/Debit Card, Wallet, or Cash on Delivery). ➢ The system must verify customer payment information before processing the transaction. ➢ The system should interact with the Payment Gateway to authorize the payment. ➢ Upon successful transaction, the system must update the order status to "Paid." ➢ The system must inform the customer whether the payment was successful or failed.
Non-Functional Requirements	➢ Payment information must be secured using encryption (SSL/TLS). ➢ Transactions should be completed within 3–5 seconds. ➢ The system should retry the payment if a gateway timeout happens. ➢ The payment service must maintain at least 99% availability. ➢ The payment interface should be user-friendly and easy to navigate.

a: Sequence Diagram:



Question # 02: SRS Document in IEEE Format:

Mini SRS Document (IEEE Format):

1. Introduction:

The Online Food Delivery System enables customers to order food online, restaurants to manage menus and process orders, and delivery personnel to deliver orders. The system ensures secure payments, real-time order tracking, and efficient communication between users.

2. Overall Description:

- **Users:** Customers, Restaurants, Delivery Boys, Admin.
- **Features:** Menu browsing, online ordering, secure payment, delivery tracking, ratings & reviews.
- **Constraints:** Must support concurrency, secure payment, and high availability.

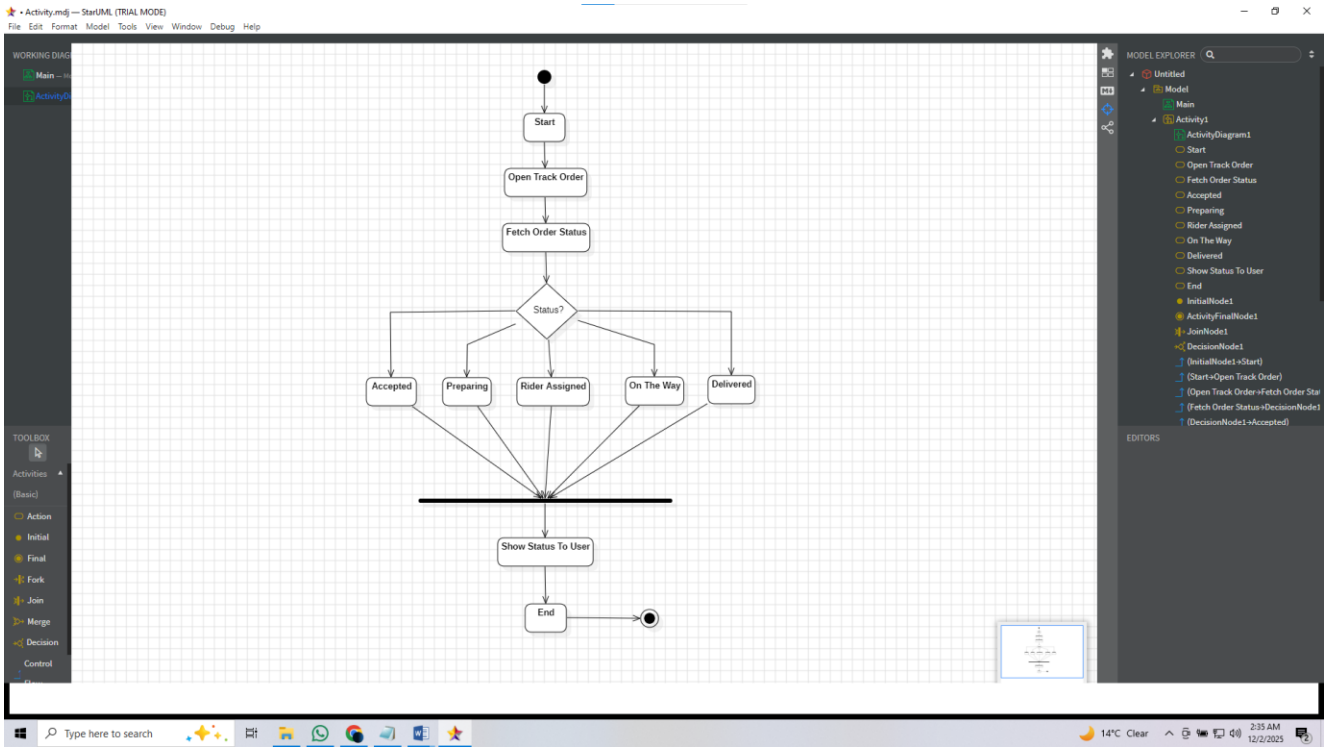
3. Functional Requirements:

1. The system must allow users to register and log in.
2. The system must allow customers to place orders from multiple restaurants.
3. Restaurants must be able to update menus.
4. The system must support secure online payments.
5. Delivery personnel must update delivery status (Picked → On the way → Delivered).
6. Customers must be able to track orders in real time.

4. Non-Functional Requirements:

1. **Performance:** The system should load pages within 2 seconds.
2. **Security:** All passwords and payment info must be encrypted.
3. **Reliability:** 99% uptime for the server.
4. **Scalability:** Must support thousands of concurrent customers.
5. **Usability:** Interface must be simple and mobile responsive.

Question # 03: Class Diagram:



Question # 04: Coding & Git Update:

```
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
        }
    }
}
```

```
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
    }
}
```

```
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
    }
}
```

GitHub: https://github.com/codebyafnan/SRE_Assignments

Question # 05: Security Analysis – Vulnerability Assessment:

a: 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.

b: 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.