Design UML Diagram

Subject: *SOFTWARE DESIGN AND ARCHITECTURE (CSE303)*

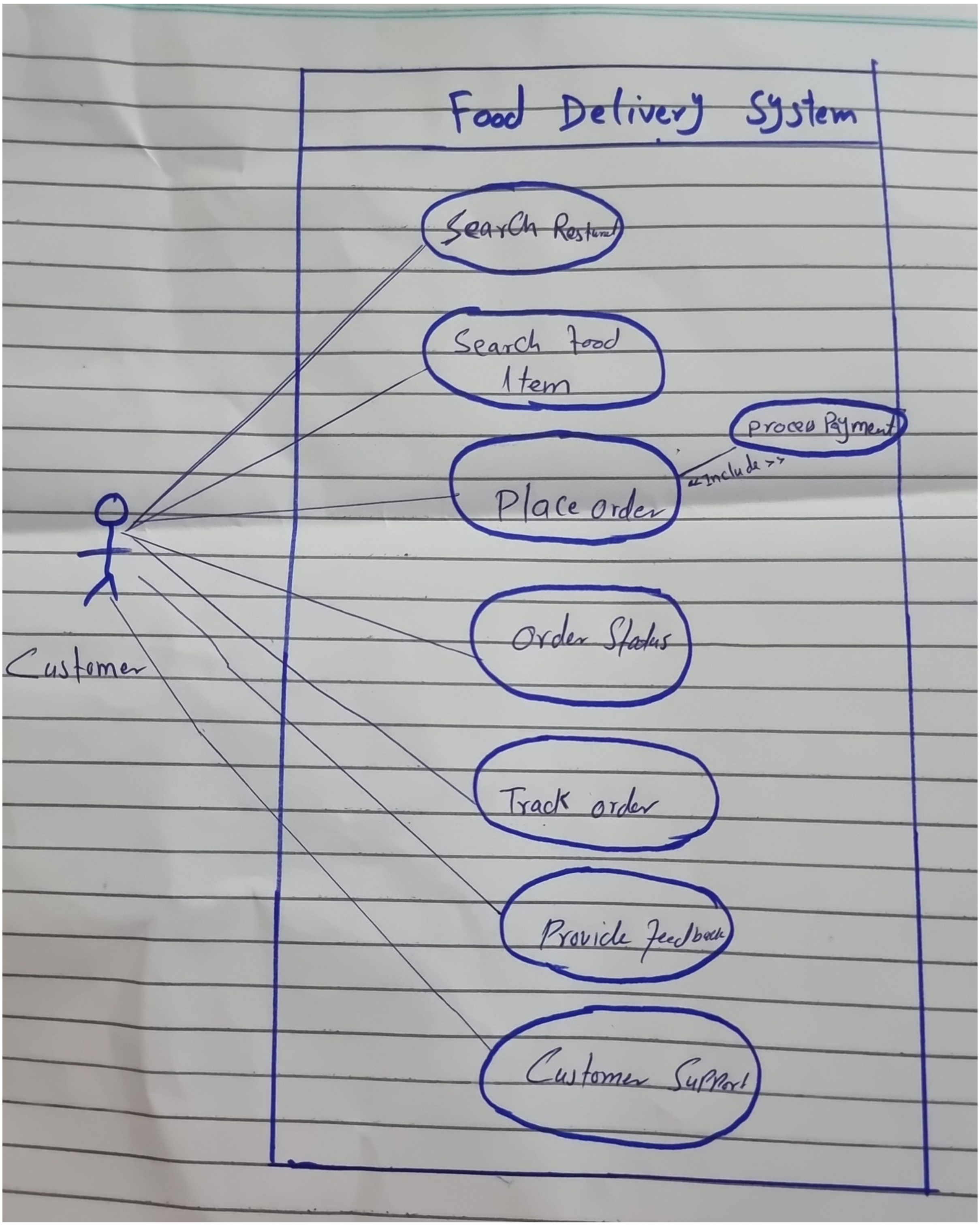
Submitted to: **Mr. MUKHTIAR ZAMIN**

Submitted by: **KASHIF KHAN**

Student Reg#: **FA22-BSE-O68**

Date: 16-10-2024

Use Case:


Food Delivery System
Search Restaurant
Search food Item
Process Payment
<<Include>>
Place order
Order Status
Track order
Provide feedback
Customer Support
Customer

Full Dress UseCase:

1. Use Case Name: Place Order

2. Primary Actor: Customer

3. Stakeholders and Interests:

1. **Customer:** Wants to order food quickly and easily.
2. **Restaurant:** Receives orders and fulfills them.
3. **Payment Provider:** Processes payments securely.
4. **Delivery Personnel:** Delivers the order to the customer.

4. Preconditions:

1. The customer must be logged into the system.
2. The restaurant must have its menu available and visible to the customer.
3. Payment methods should be set up and ready for transactions.

5. Postconditions:

1. The order is successfully placed and confirmed.
2. The payment is processed.
3. The customer receives an order confirmation.
4. The restaurant is notified of the new order.
5. Delivery is scheduled for the order.

6. Main Success Scenario (Basic Flow):

1. The customer selects a restaurant from the list.
2. The customer browses the menu and selects items to add to the cart.
3. The customer reviews the items in the cart and selects "Place Order."
4. The system prompts the customer to confirm the delivery address.
5. The system displays available payment options.
6. The customer selects a payment method.
7. The system processes the payment through the selected payment provider.
8. The system confirms the payment and displays a "Payment Successful" message.
9. The system sends an order confirmation to the customer via email/SMS.
10. The system notifies the restaurant of the new order.
11. The order status is set to "Preparing."
12. Delivery personnel are notified of the new order for pickup.

7. Extensions (Alternative Flows):

Restaurant Not Available:

1. If the restaurant is not accepting orders, the system displays a message: "This restaurant is currently not accepting orders. Please try later or select a different restaurant."

Menu Item Unavailable:

2. If a selected item is out of stock, the system displays an error and asks the customer to modify the order.

Customer Modifies Order:

3. The customer can update the cart (add or remove items) before proceeding to "Place Order."

Invalid Payment Method:

4. If the selected payment method is invalid, the system prompts the customer to choose a different payment method.

Payment Failed:

5. If the payment fails, the system displays an error message: "Payment could not be processed. Please try again."

Restaurant Cannot Fulfill Order:

6. If the restaurant cannot fulfill the order (e.g., kitchen closed), the system cancels the order, refunds the payment, and notifies the customer.

8. Special Requirements:

1. The system must handle a high volume of transactions efficiently.
2. The system must comply with PCI-DSS standards for payment processing.
3. The system should be accessible and easy to navigate on mobile devices.

9. Technology and Data Variations List:

1. **Payment Processing:** Support for credit cards, PayPal, and other digital wallets.
2. **Order Notifications:** Support for SMS, email, and in-app notifications.

10. Frequency of Use:

1. Multiple times a day, based on customer demand.

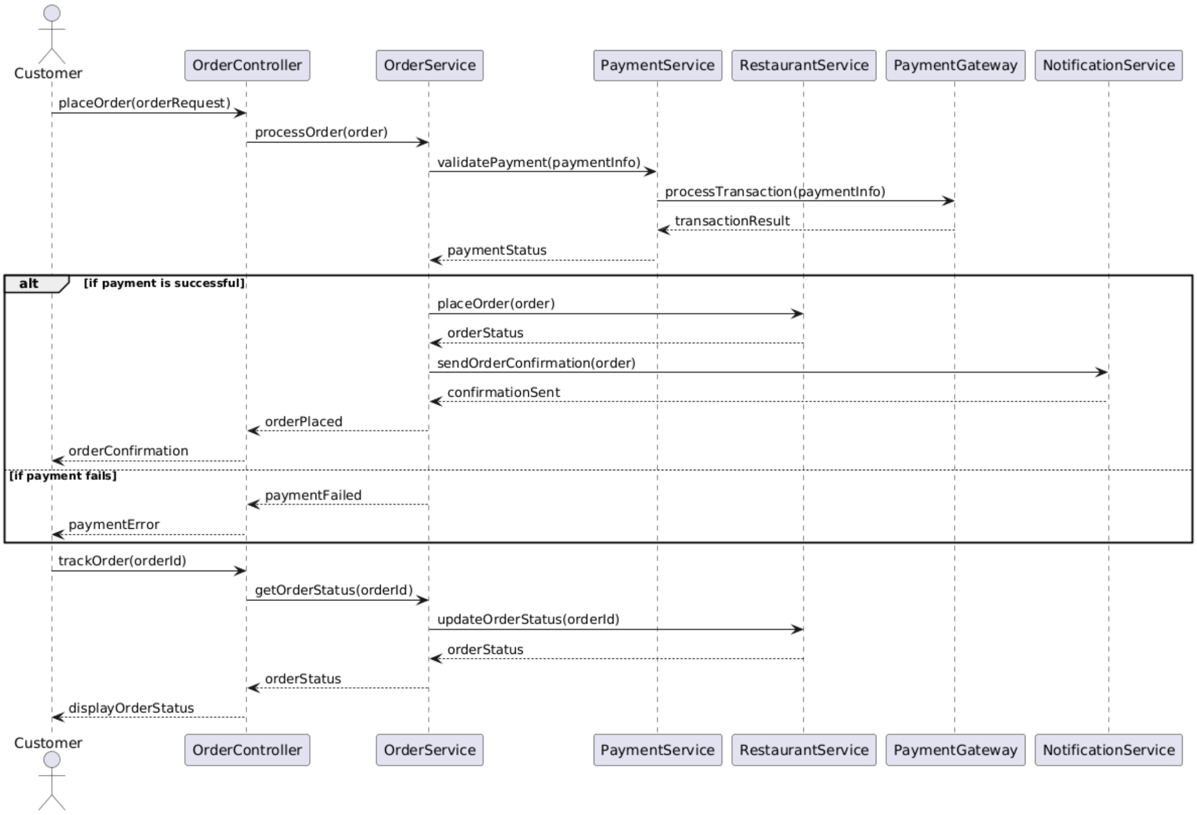
11. Assumptions:

1. The customer has a valid account and payment method set up.
2. The restaurant has an up-to-date menu and accurate item availability.

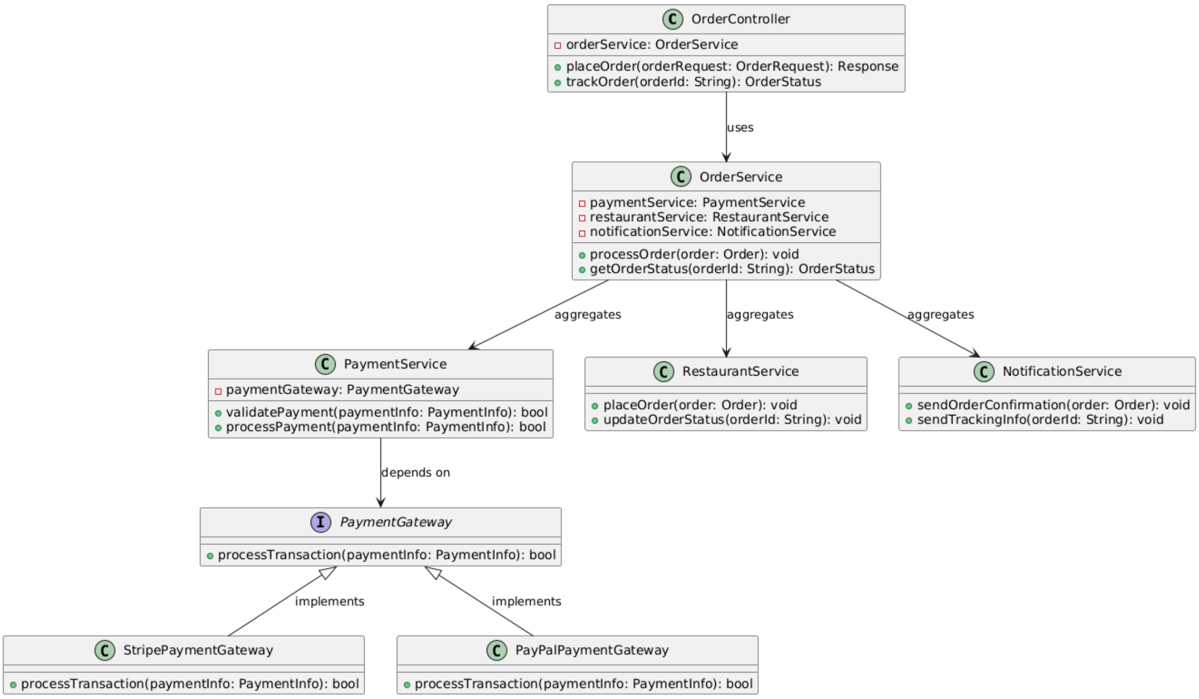
12. Open Issues:

1. How to handle situations where the delivery personnel are unavailable or the order is delayed.
2. Handling customer requests for changes after the order is placed.

Sequence Diagram:



Class Diagram


C OrderController
- orderService: OrderService
+ placeOrder(orderRequest: OrderRequest): Response
+ trackOrder(orderId: String): OrderStatus
uses
C OrderService
- paymentService: PaymentService
- restaurantService: RestaurantService
- notificationService: NotificationService
+ processOrder(order: Order): void
+ getOrderStatus(orderId: String): OrderStatus
aggregates
aggregates
aggregates
C PaymentService
- paymentGateway: PaymentGateway
+ validatePayment(paymentInfo: PaymentInfo): bool
+ processPayment(paymentInfo: PaymentInfo): bool
C RestaurantService
+ placeOrder(order: Order): void
+ updateOrderStatus(orderId: String): void
C NotificationService
+ sendOrderConfirmation(order: Order): void
+ sendTrackingInfo(orderId: String): void
depends on
I PaymentGateway
+ processTransaction(paymentInfo: PaymentInfo): bool
implements
implements
C StripePaymentGateway
+ processTransaction(paymentInfo: PaymentInfo): bool
C PayPalPaymentGateway
+ processTransaction(paymentInfo: PaymentInfo): bool