Generating Order Summary

This documentation provides a high-level overview of the approach to generating an order summary in a Spring Boot application. It outlines the key steps involved and describes the general flow of data and interactions.

Continuing the PlaceOrder API

We first empty the user's cart and send the order summary through the notification service.

```
@Service
public class OrderService {

    @Autowired
    private OrderRepository orderRepository; // Assuming you have a repository
for Order entity

    @Autowired
    private OrderItemRepository orderItemRepository; // Assuming you have a
repository for OrderItem entity

    @Autowired
    private CartService cartService; // Assuming you have a service for
managing user carts

    @Autowired
    private NotificationService notificationService; // Assuming you have a
service for sending notifications
    @Transactional
    public Order placeOrder(OrderRequestDto orderRequestDto) {
        // Get Cart: Retrieve the user's cart containing items
        List<CartItem> cartItems = cartService.getCartItems(userCart);

        // Create an Order: Create an instance of the Order class and populate
it with basic information
        Order order = new Order();
        order.setCustomer(customer);
        order.setItems(new ArrayList<>()); // Initialize the list

        // Convert Cart Items to Order Items: For each item in the cart, create
an OrderItem and add it to the order
        for (CartItem cartItem : cartItems) {
            OrderItem orderItem = new OrderItem();
            orderItem.setOrder(order);
            orderItem.setProduct(cartItem.getProduct());
            orderItem.setQuantity(cartItem.getQuantity());
            orderItem.setPrice(cartItem.getProduct().getPrice()); // Assuming
product price is used

            order.getItems().add(orderItem);

            // Delete Item from Cart: Remove the item from the user's cart
            cartService.deleteCartItem(cartItem);
        }

        // Calculate Total Price
        BigDecimal totalPrice = calculateTotalPrice(order.getItems());
        order.setTotalPrice(totalPrice);

        // Persist the Order
        order = orderRepository.save(order);

        // Generating order summary, intimating the user
        cartService.deleteUserCartItems(userCart.getUserId());

        User user = userService.getUserById(userCart.getUserId());
        String subjectString = “Order placed with id: ” + order.getId();

        if(user!=null) {
            notificationService.sendOrderConfirmation(user.getEmail(),
subjectString);
        }
        return order;
    }

    private BigDecimal calculateTotalPrice(List<OrderItem> orderItems) {
        // Logic to calculate total price
        // Sum up the price of each order item
        return orderItems.stream()
                .map(OrderItem::getPrice)
                .reduce(BigDecimal.ZERO, BigDecimal::add);
    }
}
```

In the above code snippet, we continue the **createOrder** endpoint ***/api/orders*** that accept the orderRequest as a Request Body, generate the order summary as a string, and send a notification to the user using **NotificationService.**