Placing The Order

Placing an Order: A High-Level Approach

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

Step 1: Create the Order and OrderItem Model

A. The Order model represents the structure of an order and includes attributes such as order ID, customer details, order items, total price, and any additional metadata. Here's an example of the Order model:

```
public class Order {
    private Long id;
    private Customer customer;
    private List<OrderItem> items;
    private BigDecimal totalPrice;

    // Constructors, getters, and setters
}
```

In the above code snippet, create the Order class with the necessary attributes, constructors, and getter/setter methods.

B. The Order Item entity represents an individual item within an order and contains information such as the associated product, quantity, and price. Here's an example of the OrderItem entity:

```
@Entity
@Table(name = "order_items")
public class OrderItem {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @ManyToOne
    @JoinColumn(name = "order_id")
    private Order order;

    @ManyToOne
    @JoinColumn(name = "product_id")
    private Product product;
```

```
    private int quantity;
    private BigDecimal price;

    // Constructors, getters, and setters
}
```

In the above code snippet, the OrderItem entity is annotated with @Entity to map it to a database table named "*order_items*". A few of the other key components associated with the code above for the ***OrderItem*** entity are as follows:

Step 2: Define the Order Flow

The order flow now involves retrieving the user's cart, creating an order with basic information, converting cart items to order items, and sending a notification. Here's an outline of the updated general flow:

1. Get Cart: Retrieve the user's cart containing items.
2. Create an Order: Create an instance of the Order class and populate it with basic information.
3. Convert Cart Items to Order Items: For each item in the cart, create an OrderItem and add it to the order. Include product details, quantity, and price.
4. Delete Item from Cart: Remove the item from the user's cart.
5. Send Notification: Notify the user about the successful placement of the order.

Step 3: Implement the Order Service

The Order service contains the business logic for order management. It encapsulates the functionality required to perform the order placement process. Here's an example of the OrderService class:

```
@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);

        // Send Notification: Notify the user about the successful placement of
the order
        notificationService.sendOrderConfirmation(order);

        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, create the OrderService class with the necessary method for placing an order. Autowire any repositories or services required for data access and business logic.

Step 4: Create the Order Controller

The Order controller handles incoming HTTP requests related to order placement. It maps the appropriate endpoints and delegates the processing to the OrderService. Here's an example of the OrderController class:

```
@RestController
@RequestMapping("/api/orders")
public class OrderController {

    // Autowire the OrderService

    @PostMapping
    public ResponseEntity<Order> placeOrder(@RequestBody Order order) {
        // Implement the logic to handle the order placement request
        // Call the OrderService to place the order
        // Return the placed order with the appropriate HTTP status
    }
}
```

In the above code snippet, create the OrderController class with the necessary annotations and method for handling the order placement request. Autowire the OrderService to delegate the processing.

Step 5: Configure Security (Optional)

Configure security measures if your application requires authentication and authorisation for order placement. This may involve using Spring Security to define authentication providers, roles, and access control rules.

Usage

- To place an order using the implemented APIs, send a POST request to the ***/orders*** endpoint with the order data in the request body. The response will contain the placed order details.
- Ensure you have properly configured and deployed your Spring Boot application to handle the order placement process.