2023-09-30

应用系统体系架构 — 作业2

学号：521030990006

姓名：VAHAGN GHAZARYAN

## The @Transactional annotation at the addOrder() method level ensures that all operations within that method are part of the same transaction. So, if any part fails, everything will be rolled back, maintaining data integrity.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | addOrder | orderDao.save | orderItem.save | result |
| 1 | normal | normal | normal | return savedOrder |
| 2 | result = 10/0 | normal | normal | Transaction rolled back |
| 3 | normal | result = 10/0 | normal | Transaction rolled back |
| 4 | normal | normal | result = 10/0 | Transaction rolled back |
| 5 | normal | normal | normal (required new) | return savedOrder |
| 6 | result = 10/0, before orderItem.save | normal | normal (required new) | Transaction rolled back |
| 7 | result = 10/0, after orderItem.save | normal | normal (required new) | orderItem saved, orderDao and addOrder are failed |
| 8 | normal | result = 10/0 | normal (required new) | Transaction rolled back |
| 9 | normal | normal | result = 10/0 (required new) | orderDao saved, orderItem failed, return savedOrder |

## Analysis on the advantages and disadvantages of the three front-end presentation methods

## *JavaScript Topic Monitoring*

## JavaScript will listen to the Kafka topic and refresh the page upon receiving a new message.

## Advantages: Real-time updates. Low latency.

## Disadvantages: Requires a Kafka client for JavaScript, adding complexity. Not scalable for many users.

## *Ajax Polling*

## Front-end sends periodic Ajax requests to the back-end to get the latest order status.

## Advantages: Simple to implement. Doesn't require any special client libraries.

## Disadvantages: Not real-time. Extra server load due to frequent polling.

## *WebSocket*

## The back-end listens to Kafka topic messages and pushes updates to the front-end via WebSocket.

## Advantages: Real-time updates. Efficient use of resources.

## Disadvantages: Requires setting up WebSocket, adding a little complexity.

## **Chosen Method: WebSocket**

## I have used SockJS and STOMP on the client-side to establish a WebSocket connection with the backend. The system is configured to listen to specific Kafka topics for order processing results and sends these results in real-time to the front-end via SimpMessagingTemplate template class *(see import org.springframework.messaging.simp.SimpMessagingTemplate)* under the "/topic/orders" topic. The frontend subscribes to this topic and presents the user with real-time updates using Ant Design's "message" component.

## 

## 

## **Messige format**

## In your @KafkaListener method, I converted an Java object (Order) to JSON using library Jackson and then send it via SimpMessagingTemplate. In front end I have used JSON.parse to parse received message.

## **Topic configuration**

## In the backend, WebSocket configuration is done by Spring's @EnableWebSocketMessageBroker annotation. The broker is configured to relay messages to the **"/topic"** destination and accepts incoming messages prefixed with **"/app"**. SockJS is employed to handle WebSocket fallback options, and the endpoint is set as **"/websocket-endpoint"**, with CORS configured to allow connections from **"http://localhost:3000".**

## 

## **DEMO**

## 