SAGA 패턴 기반 분산 트랜잭션 시스템 설계

STEP 1. 시나리오 선택 및 분석

아래 **4가지 시나리오 중 1개를 선택**하여 설계합니다.

출력물: 시나리오 분석 문서

항목	내용
선택한 시나리오	A
관련 서비스	주문, 쿠폰, 재고, 결제
정방향 트랜잭션	T1 → T2 → T3 → ...
실패 지점 (3개 이상)	쿠폰, 재고, 결제
보상 트랜잭션	실패 지점별 보상 액션

STEP 2. 설계 점검 체크리스트 ⭐

의도: 설계가 실무에서 동작하는지 **스스로 검증**합니다.

🔄 멱등성 (Idempotency)

질문	본인 답변
같은 환불/취소 요청이 2번 오면 어떻게 처리하나요?	Kafka 메시지의 eventId 또는 HTTP 요청의 idempotencyKey를 Inbox 테이블에 저장해서,이미 처리한 이벤트인지 여부를 선행 검사한 후 처리/무시를 결정합니다. Inbox는 정확히 1번 처리를 보장하는 역할을 합니다.
중복 요청을 어떻게 식별하나요? (어떤 키 사용?)	서비스에서 생성한 글로벌 유니크 ID(UUID) 기반의 eventId를 사용합니다. API 호출의 경우 클라이언트 혹은 서버가 생성한 idempotencyKey를 사용합니다.

⏰ 타임아웃 & 재시도

질문	본인 답변
외부 API(카드사, PG) 응답이 안 오	외부 API는 연결 타임아웃과 응답 타임아웃을 분리해서 관리합니다. 예를 들어 Connect Timeout 2초, Read Timeout 3초로 설정하고,

질문	본인 답변
면 얼마나 기다리나요?	실패 시 Resilience4j Retry로 3회까지 재시도합니다.
재시도는 몇 번까지, 어떤 간격으로 하나요?	고정 Backoff(5초) 또는 Exponential Backoff를 적용합니다.
재시도해도 계속 실패하면 최종적으로 어떻게 처리하나요?	DLQ로 보내집니다.

🔀 부분 실패

질문	본인 답변
5개 단계 중 3번째에서 실패하면, 1~2번은 어떻게 되나요?	1~2번에서 이미 반영된 변경을 상쇄(Compensate)하는 보상 트랜잭션을 실행합니다. 예를 들어 재고 감소를 롤백하는 대신 “재고 +1”과 같은 반대 작업을 수행합니다.
보상 트랜잭션 실행 순서는 어떻게 되나요?	우리 시스템은 각 단계별 보상 트랜잭션을 중앙에서 순차 실행하지 않습니다. 대신 각 단계의 보상 로직이 포함된 보상 이벤트(compensation event) 를 발행해서 해당 서비스가 자체적으로 보상 로직을 실행하는 방식(Saga Choreography)을 사용합니다.그래서 “정확한 역순 보상”을 보장하기보다는,각 단계가 스스로 일관성을 맞출 수 있도록 설계되어 있습니다.
보상 트랜잭션도 실패하면 어떻게 하나요?	보상 트랜잭션에 대한 Retry 및 Backoff 전략을 수립하고 이후에는 DLQ를 통해 관리될 듯 싶습니다.

📊 상태 관리

질문	본인 답변
현재 트랜잭션이 어느 단계인지 어떻게 알 수 있나요?	event 내의 context 속성으로 구분합니다.
중간에 서버가 죽었다가 재시작하	"서버 크래시(Crash) 발생 시, 주문 테이블 내 'Pending(처리 중)' 상태로 남아 있는 트랜잭션은 비정상 중단된 건으로 간주합니다. 시스템은 해당 건에 대해 스

질문	본인 답변
면 어디서부터 이어가나요?	케줄러를 통해 롤백(Rollback) 이벤트를 발행하여 보상 트랜잭션을 수행함으로써 기존 데이터를 정리(Clean-up)합니다. 이후 시스템 차원에서의 자동 재개는 이루어지지 않으며, 주문을 다시 진행하고자 할 경우 클라이언트(사용자)가 신규 API 요청을 보내야 합니다."

🚨 예외 케이스 (시나리오별)

시나리오 A: 쿠폰이 만료되어 복원 불가하면 어떻게 보상하나요?

- 쿠폰이 만료되어 복원 불가하면 쿠폰의 상태를 Expired로 바꾸고 추후 결제에 사용하지 못하도록 합니다.

STEP 3. 아키텍처 설계

3-1. 패턴 선택 및 근거

☑ ~~Orchestration~~

☐ Choreography

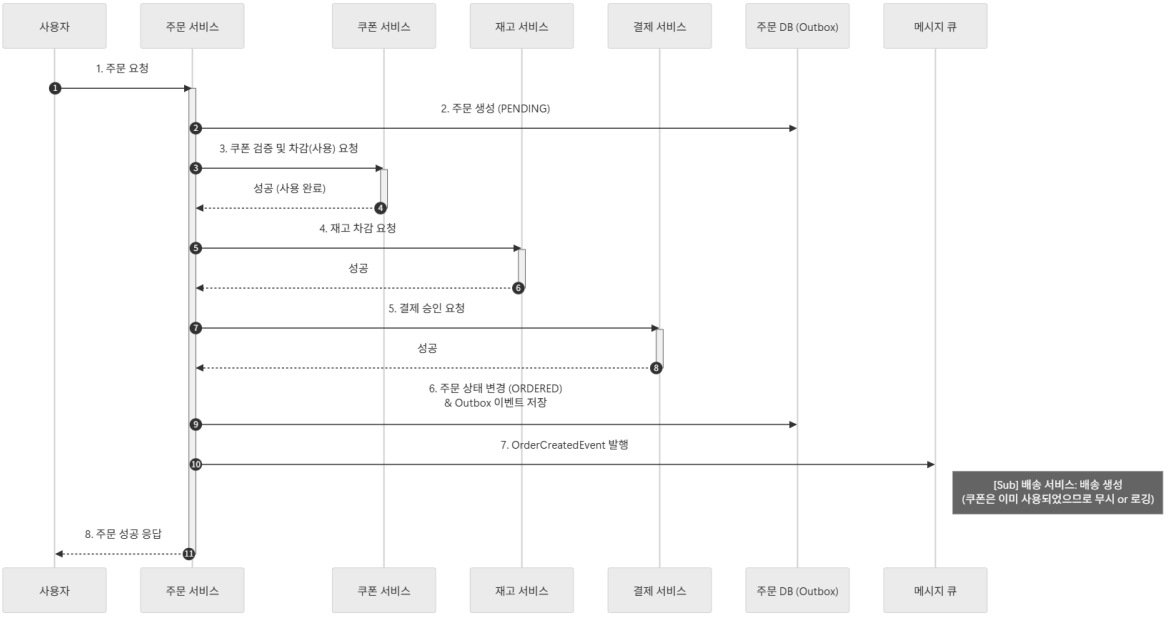
선택 이유: [2~3문장으로 근거 작성]

저희 시스템은 주문 서비스가 전체 프로세스를 주도하고, 각 단계가 **동기 흐름 내에서 강한 결합을 갖는 구조**라서 Orchestration이 더 적합하다고 생각했습니다. 중앙에서 흐름을 관리함으로써 장애 처리·보상 트랜잭션·상태 전이를 명확하게 통제할 수 있고, 서비스 간 순서를 보장해야 하는 요구사항도 안정적으로 충족할 수 있습니다. 이러한 이유로 Orchestration 패턴을 선택하게 됐습니다.

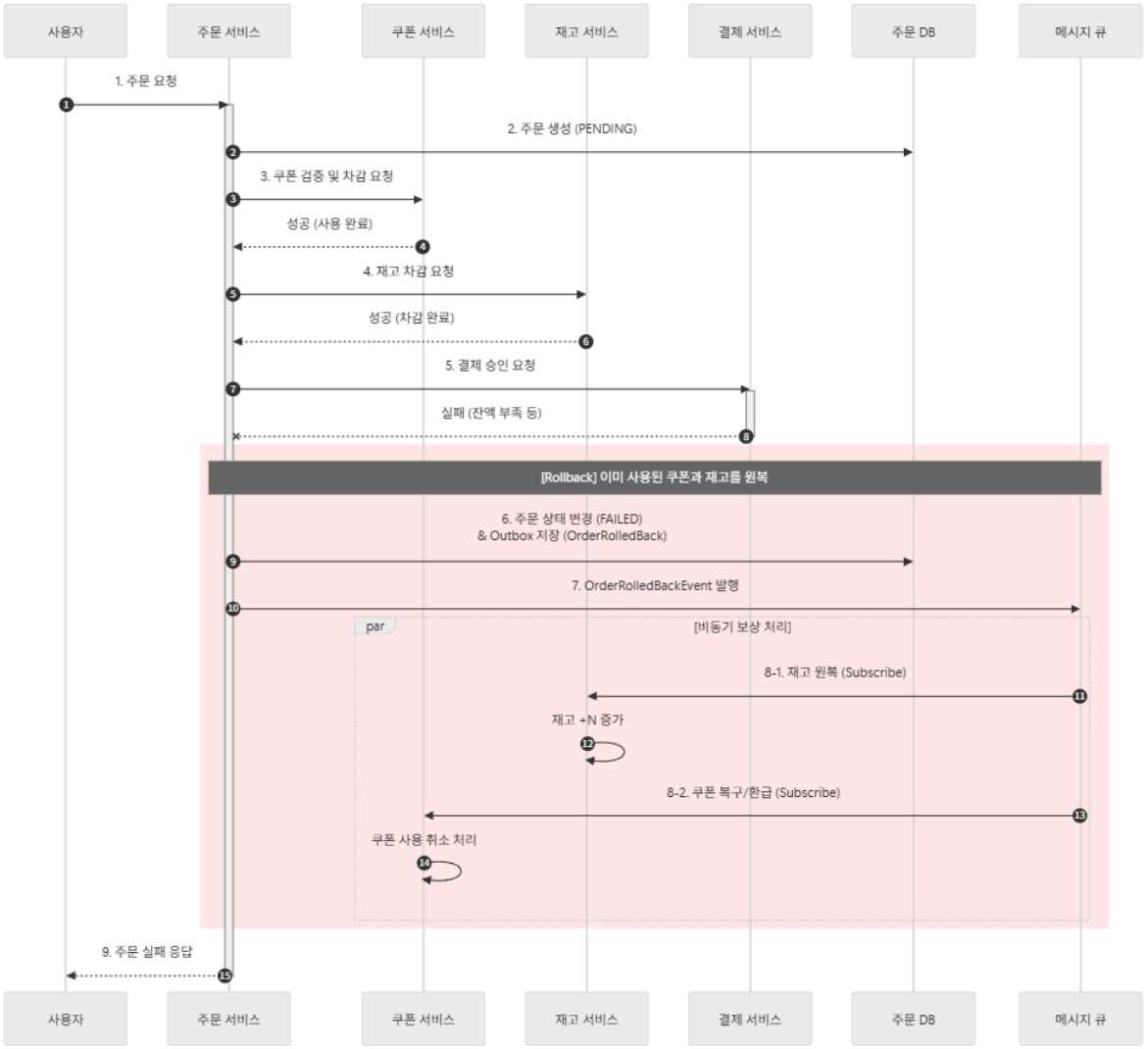
3-2. 시퀀스 다이어그램 (2개)

성공 시나리오 + 실패 → 보상 시나리오 작성

- 성공 시나리오

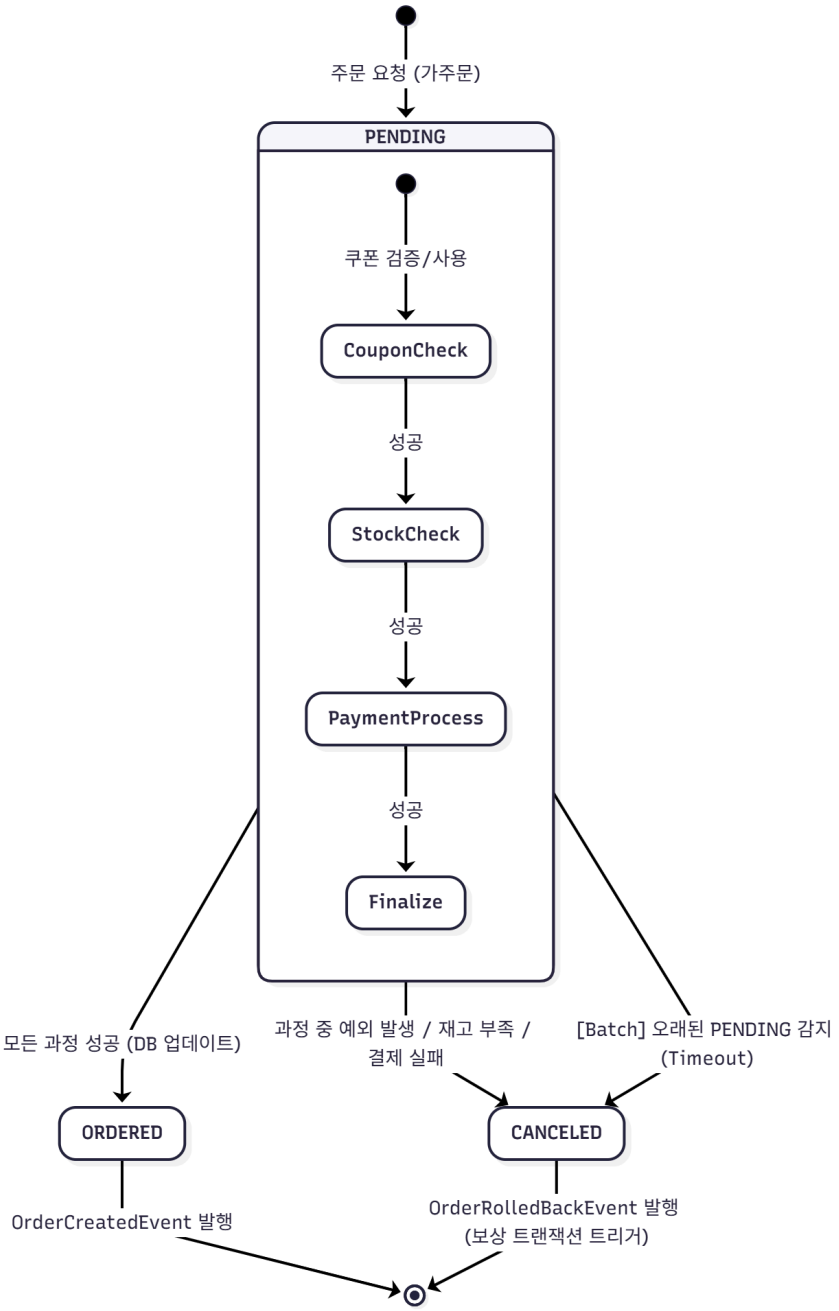

사용자
주문 서비스
쿠폰 서비스
재고 서비스
결제 서비스
주문 DB (Outbox)
메시지 큐
1. 주문 요청
2. 주문 생성 (PENDING)
3. 쿠폰 검증 및 차감(사용) 요청
성공 (사용 완료)
4. 재고 차감 요청
성공
5. 결제 승인 요청
성공
6. 주문 상태 변경 (ORDERED)
& Outbox 이벤트 저장
7. OrderCreatedEvent 발행
[Sub] 배송 서비스: 배송 생성
(쿠폰은 이미 사용되었으므로 무시 or 로깅)
8. 주문 성공 응답
사용자
주문 서비스
쿠폰 서비스
재고 서비스
결제 서비스
주문 DB (Outbox)
메시지 큐

- 실패 + 보상 시나리오



3-3. 상태 다이어그램

주문/구독/장바구니 등 핵심 엔티티의 **상태 전이**를 그립니다.


주문 요청 (가주문)
PENDING
쿠폰 검증/사용
CouponCheck
성공
StockCheck
성공
PaymentProcess
성공
Finalize
모든 과정 성공 (DB 업데이트)
과정 중 예외 발생 / 재고 부족 / 결제 실패
[Batch] 오래된 PENDING 감지 (Timeout)
ORDERED
CANCELED
OrderCreatedEvent 발행
OrderRolledBackEvent 발행 (보상 트랜잭션 트리거)