이 소 연

GitHub Blog

010-6744-6992
wxy890@gmail.com

- 백엔드, 프론트엔드, DB 설계, 배포 경험이 있습니다.
- 코드의 가독성과 명확한 네이밍에 많은 노력을 기울입니다.
- 빠르고 효율적으로 데이터를 얻기 위한 아키텍처를 구상하는 것에 흥미를 느낍니다.
- 사용자(또는 동료) 친화적인 결과물을 도출하기 위한 고민을 합니다.

DEVELOPER COURSE

2024.07 ~ 2024.10 (10주)
항해99 취업 리부트 코스
- 자료구조 / 알고리즘
- 대기업 프로젝트

2024.06 ~ 2024.07 (1개월)
AWS 데브옵스 웹개발 과정
- 도커 컴포즈를 활용하여 AWS 배포
- EUREKA / SOLID 이론
- 자율 프로젝트

2023.08 ~ 2024.03 (6개월)
네이버 데브옵스 웹개발 과정
- 웹 어플리케이션 개발 팀 프로젝트
- DevOps 기반 기술 이해
- 개발자를 위한 DevOps
- NCP DB & 스토리지
- 도커 컨테이너 애플리케이션 배포
- 운영자를 위한 DevOps

WORK EXPERIENCE

경력 9년 / 비개발

2019.06 ~ 2023.06 (4년 1개월)
네오진명보습학원 / 경영지원팀-실장
- 교무 업무 & 학원 전반 경영 관리
- 엑셀 VBA을 이용한 검증 시스템 구축

2016.06 ~ 2017.06 (1년 1개월)
메가스터디 러셀 / 교무운영팀-사원
- 홍보 마케팅 & 교무 업무
- 엑셀 수강료 이슈 확인 시스템 구축

2010.10 ~ 2014.12 (4년 3개월)
팀일렉트로닉스 / 온라인판매팀-팀장
- 오픈마켓 및 홈페이지 운영

SKILLS

- **Back-end**

Java	Spring Boot	Spring Security
Spring Cloud	JPA MyBatis	JWT
PostgreSQL	Redis	MySQL
Docker / Docker Compose	Query DSL	Redisson
MSA • API Gateway • Service Discovery • Apache Kafka • Saga Pattern • **OpenFeign**		Swagger/ OpenAPI
		Gradle
		Junit SLF4J

- **Front-end**

JavaScript / TypeScript / React / Recoil

CERTIFICATIONS

- **정보처리기사 [필기]** 2024.07
- **SQL개발자 (SQLD 자격)** 2023.11

EDUCATION

- **전남대학교 시각디자인학과 (중퇴)** 2009.08
- **부영여자고등학교 졸업** 2008.02

기술적 스킬과 프로젝트 경험

체리 웨더 - 2023 / 네이버 데브옵스 부트캠프

날씨를 기반으로 한 커뮤니티와 AI 복장 추천을 결합한 서비스를 개발했습니다. 팀장으로서 프론트엔드와 백엔드를 모두 담당하며 풀스택 개발 경험을 쌓았습니다.
클럽, 멤버십, 좋아요, 피드 기능의 백엔드 개발을 담당했습니다. 특히 서비스 간 순환 의존성 문제를 해결하기 위해 이벤트 리스너를 활용한 느슨한 결합 구조를 구현했고, Specification API를 활용한 동적 쿼리로 복잡한 검색 기능을 구현했습니다. 이 프로젝트를 통해 전체 시스템 설계 능력과 팀 협업 경험을 쌓을 수 있었고, 더 깊이 있는 백엔드 기술을 학습하고 싶다는 동기를 얻게 되었습니다.

Project Avalanche 선착순구매시스템 - 2024 / 항해 99 취업리부트

MSA와 대용량 트래픽 처리를 학습하며, 선착순 구매 시스템을 주제로 선택해 진행한 프로젝트입니다. 단순한 CRUD를 넘어서 실제 서비스에서 마주할 수 있는 복잡한 문제들을 직접 경험해보고 싶었습니다. 가장 큰 도전은 여러 사용자가 동시에 같은 상품을 구매할 때 재고 오버셀링을 방지하는 것이었습니다. Redis 분산락(Redisson)을 도입해 동시성 문제를 해결했고, 실제로 분산락이 작동하는 것을 확인할 수 있었습니다. 또한 마이크로서비스 간 데이터 일관성 유지라는 새로운 난제를 만났습니다. 주문이 실패 했을 때 포인트 차감과 재고 예약을 모두 되돌려야 하는데, 서로 다른 서비스에 데이터가 분산되어 있어 기존 트랜잭션으로는 해결할 수 없었습 니다. Saga Pattern을 구현해 보상 트랜잭션을 통해 이 문제를 해결했습니다.

적합성 및 학습의지

효율적인 소통의 철학

모든 일의 결과물은 동료나 사용자가 가치를 매긴다고 생각합니다. 따라서 소통이란 내용을 상대방이 잘 이해하도록 돕는 과정이라고 여기고 "왜 전달해야하는지"와 "어떻게 하면 잘 전달할 수 있을지"를 항상 고민합니다. 이전 직장에서는 이러한 고민을 바탕으로 업무를 공유화하여 불필요한 소통을 줄이고, 반복 업무를 자동화하여 중요한 업무를 효율적으로 처리할 시간을 확보했습니다.

다양한 직무 경험을 바탕으로 코딩 과정에서도 다른 사람들이 쉽게 이해하고 유지 보수할 수 있도록 코드 컨벤션을 준수하고, 일관된 코드 스타일을 유지하며, 모듈화를 통해 코드의 재사용성을 높이는 데 중점을 두었습니다. 이러한 제 노력은 코드 품질을 높여 팀 협업을 효과적으로 만들었습니다.

적합성 및
학습의지

새로운 기술을 향한
끝없는 열정

저는 효율성과 트렌드를 중시하며, 새로운 기술 학습을 즐깁니다. 체리웨더 프로젝트 당시 프로젝트 특성상 학원에서 배운 MyBatis가 아닌 JPA를 사용하게 되었는데, 팀 전체가 처음 접하는 기술이었지만 빠르게 학습하여 프로젝트에 성공적으로 적용할 수 있었습니다. 이후 부트캠프에서 MSA, Kafka, 분산락 등 더 복잡한 기술들에 도전할 수 있었습니다.
새로운 기술을 배울 때는 단순히 사용법만 익히는 것이 아니라, 왜 이 기술이 필요한지, 어떤 문제를 해결하는지를 원리부터 이해하려고 노력합니다. 이러한 학습 방식을 통해 지속적으로 성장하는 개발자가 되고자 합니다.

팀워크와
커뮤니케이션

소통으로 만들어가는
리더십

팀장이 조기 취업으로 팀을 떠난 후, 제가 파이널 프로젝트의 팀장을 맡게 되었습니다. 저는 팀원들과의 소통 방식을 기존의 하향식 의사결정 구조에서 벗어나 모든 팀원이 의견을 자유롭게 제시하고 토론할 수 있는 환경을 조성했습니다.
특히 세미 프로젝트에서 프론트엔드를 담당했지만 백엔드를 지망했던 팀원들에게 이번에는 원하는 파트를 선택할 기회를 제공했습니다. 각자의 강점과 관심사를 고려한 업무 분배를 통해 팀원들의 동기부여를 높이고, 진행이 지연되는 부분에 대해서는 함께 해결 방안을 모색하며 프로젝트의 원활한 진행을 이끌어냈습니다.

효율적인 대화를 통한
이해와 협력

프로젝트 경험을 통해 하향식과 수평식 커뮤니케이션의 장단점을 깊이 이해하게 되었습니다. 하향식은 배울 점이 많을 때 매우 효과적이지만, 갑작스러운 기술을 이해하라는 요구는 팀원들에게 긴장과 초조함을 안겨주고 시간도 부족하게 만들었습니다. 반면, 비전공자들로 이루어진 팀 내에서 수평식 의사소통을 하며 기술 결정을 할 때는 모든 팀원이 동등한 의견을 제시할 기회를 가졌지만, 그 결정이 항상 효율적인 방법인지는 확신할 수 없었습니다.
이러한 경험은 저에게 다양한 소통 방식의 중요성을 체감하게 했습니다. 그리고 의사결정 과정에서 매 결정에 스스로 확신을 가질 수 있는 단단한 개발자가 되고 싶다는 목표를 갖게 되었습니다. 앞으로도 이러한 깨달음을 바탕으로 팀의 시너지를 극대화하는 데 기여하는 팀원이 되고자 합니다.

01 프로젝트 소개

선착순 구매 시스템

Project Avalanche

2024.08.07 ~ 2024.09.03
개인프로젝트

프로젝트 목표

- MSA 기반의 선착순 구매 시스템 구현
- 대용량 트래픽 하에서 동시성 문제 해결
- 안전하고 확장 가능한 사용자 인증 및 세션 관리
- 실시간 재고 관리 및 포인트 시스템 구축

백엔드 기술 스택

Java21 | SpringBoot 3.2.2 | Gradle | PostgreSQL | Redis | Redisson | JWT | Spring Cloud | API Gateway | Docker | Docker Compose | Kafka | Linux | JUnit | Git

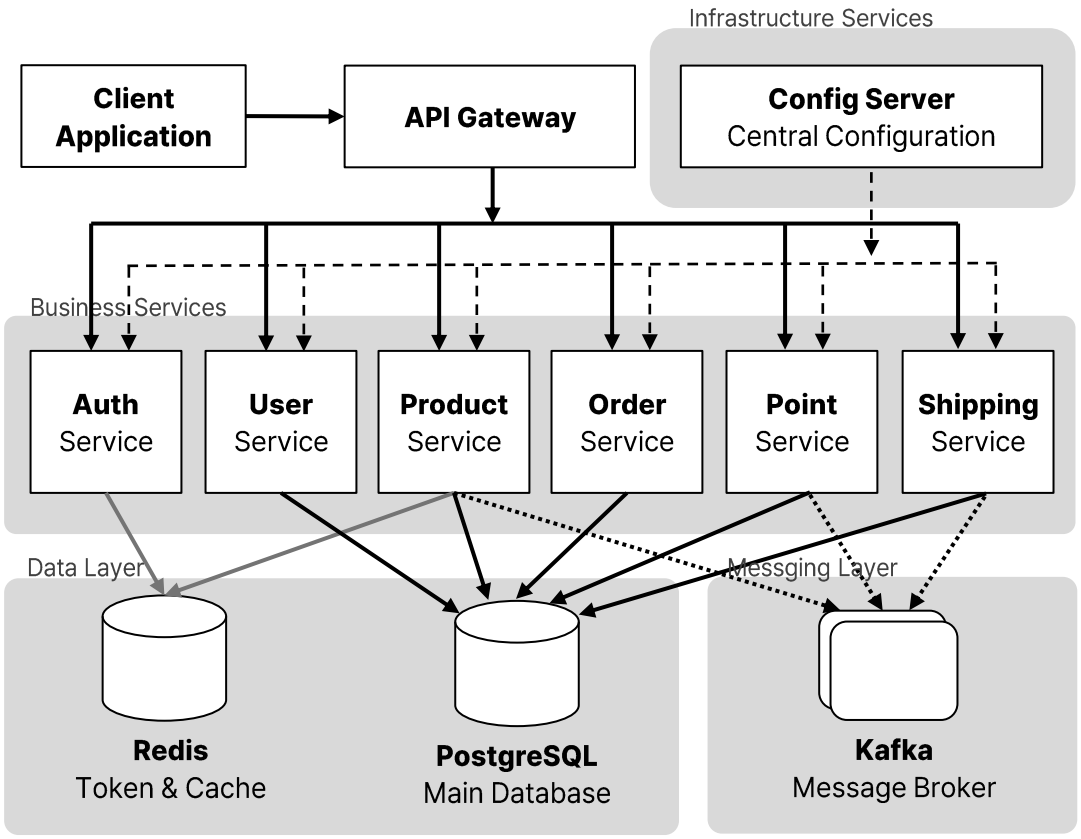
02 백엔드 기능 구현

MSA 마이크로 서비스 아키텍쳐	• Eureka를 이용한 서비스 디스커버리 구현 • DDD(Domain-Driven Design) 기반의 패키지 구조 설계
Auth Service 인증 및 보안	• JWT를 이용한 로그인 및 액세스 토큰 발급 • 회원가입 시 회원 인증을 위한 토큰 이메일로 발송 (Thymeleaf를 사용한 이메일 템플릿 제작) • 리프레시 토큰 발급 및 관리 • 로그아웃 기능 (단일 기기 및 모든 기기) • Redis를 활용한 토큰 저장 및 관리
API Gateway	• Redis 서버를 통한 사용자 토큰 검증 • 검증된 사용자 정보를 UserContext로 로컬 전달
Config Service 설정 관리	• API Gateway 라우팅 설정 • 마이크로서비스별 주요 설정 정보 중앙화 • 공통 설정 관리 (Redis, PostgreSQL, Redisson, Kafka, JWT 등)
User Service 사용자 관리	• 회원가입 프로세스 구현 • 사용자 정보 양방향 암호화 • 비밀번호 단방향 암호화 • 비밀번호 재발급 기능 • 사용자 정보 조회 API

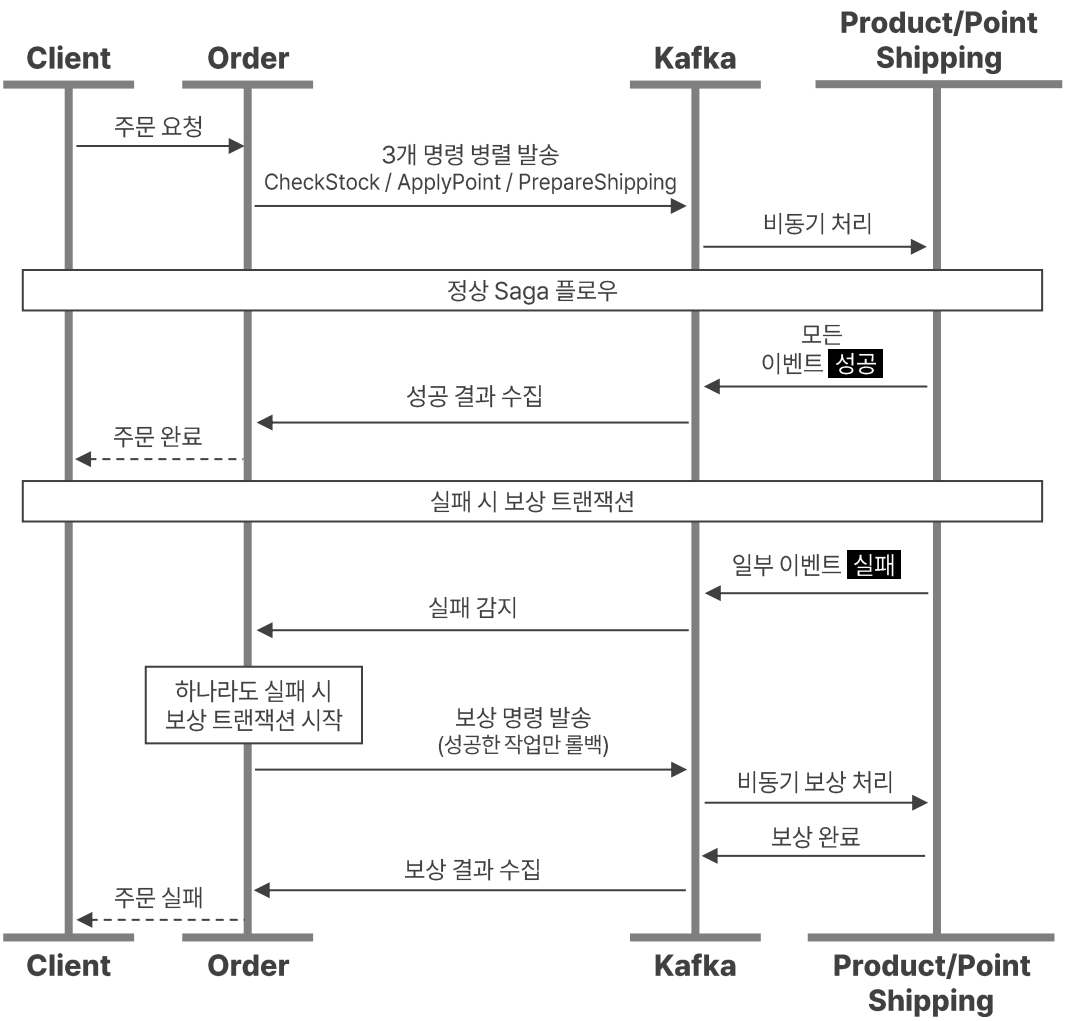
02
백엔드
기능 구현

Order Service 주문 관리	• Saga 패턴과 Kafka를 활용한 비동기 주문 생성 프로세스 • Redisson을 이용한 분산 락 구현으로 동시성 제어 • 주문 진행 중 오류 발생 시 보상 트랜잭션 처리 • QueryDSL과 페이징을 활용한 주문 내역 조회 기능
Product Service 상품 관리	• ProductService: 상품 CRUD 담당 • InventoryService: 선착순 구매 재고 관리 • Redis를 활용한 빠른 재고 접근 • 분산 락을 통한 안정적인 재고 차감 • 보상 트랜잭션 구현 • 주문 진행 중 예약 재고 관리 • 스케줄러를 통한 PostgreSQL 서버 재고 동기화
Point Service 포인트 관리	• Kafka를 통한 비동기 포인트 차감 • 포인트 관련 보상 트랜잭션 구현
Shipping Service 배송 관리	• Kafka를 통한 비동기 배송지 등록 • 배송 관련 보상 트랜잭션 구현
테스트 및 개발 지원	• 대용량 주문 처리를 위한 JUnit 기반 통합 테스트 환경 구성 • Kafka 메시징에 Avro 스키마 사용으로 데이터 직렬화 효율성 향상

03 MSA 아키텍쳐



04 분산 트랜잭션 처리 플로우



05
핵심 코드

01 동시성 제어 : Redis 분산 락

- 선착순 구매 테스트 시 1,000명이 동시에 같은 상품 구매 시도로 재고 부정합 발생
- 상품별 Redis 분산 락으로 순차 처리하여 재고 정합성 보장

```
@Transactional
public void checkAndReserveStock(CheckStockCommand command){
 String lockKey = "lock:product:" + productId;
 RLock lock = redissonClient.getLock(lockKey);
try {
    if (lock.tryLock(10, TimeUnit.SECONDS)) {
      try {
        if (reserveStockAndUpdateOrder(productId, orderId, quantity)){
            sendStockCheckedResult(orderId, true);
        } else {
            sendStockCheckedResult(orderId, false);
        }
      } finally {
         lock.unlock();
      }
    } else {
       sendStockCheckedResult(orderId, false);
    }
  } catch (InterruptedException e) {
      Thread.currentThread().interrupt();
      handleReservationError(orderId, productId, quantity, e);
  }
}
```

핵심 구현 사항

✓ Redisson 라이브러리 활용한 분산 락
✓ 10초 타임아웃으로 무한 대기 방지
✓ 상품별 개별 락 키로 성능 최적화
✓ finally 블록으로 락 해제 보장

---

02 분산 트랜잭션 : Saga 패턴 구현

- MSA 환경에서 재고확인, 포인트차감, 배송준비를 하나의 트랜잭션으로 처리 필요
- Saga 패턴으로 3개 서비스를 조율하는 분산 트랜잭션 구현

```
public void startSaga(Order order, OrderRequest request, UUID userId) {
  OrderSagaState state = OrderSagaState.createInitialState(order,
                         request, userId);
  redisTemplate.opsForValue()
               // 3개 서비스에 병렬 명령 전송
               .set(SAGA_KEY_PREFIX + order.getOrderId(),state)

  orderSagaProducer.sendCheckStockCommand(orderId, productId, quantity);
  orderSagaProducer.sendApplyPointsCommand(orderId, userId, amount,
                    activityType, productName);
  orderSagaProducer.sendPrepareShippingCommand(orderId, shippingInfo);
}
```

핵심 구현 사항

✓ Kafka 메시징으로 느슨한 결합 구현
✓ 각 단계별 성공/실패 추적 가능
✓ Redis 기반 Saga 상태 관리로 일관성 보장
✓ 3개 서비스(재고/포인트/배송) 병렬 처리로 성능 최적화

문제 상황

포인트/재고 차감 실패 시 보상 트랜잭션 멱등성 처리

구매 이벤트 시스템에서 실제로는 차감/감소가 실패했음에도 불구하고
보상 트랜잭션에서 복구 작업을 수행하는 문제가 발생했습니다.

1 포인트 부족으로 차감 실패 → 보상 트랜잭션에서 포인트 복구 시도 → 포인트 중복 지급

2 동시성 문제로 재고 감소 실패 → 보상 트랜잭션에서 재고 증가 시도 → 재고 수량 오류

문제 분석

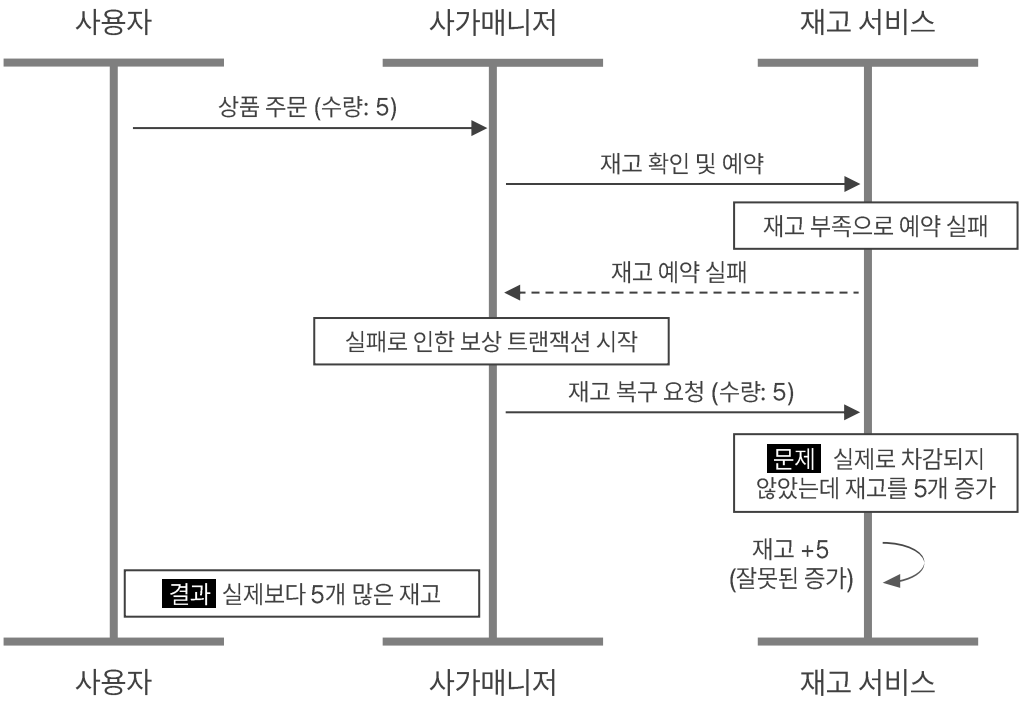
기존 코드의 문제점

```
@Transactional
public void refundInventory(ReleaseStockCommand command) {

    // 실제 재고가 차감되었는지 확인하지 않고 무조건 증가
    Product product = productService
                        .findById(command.getProductId());

    // 문제: 무조건 증가
    product.increaseStock(command.getQuantity());
}
```

문제 발생 시나리오

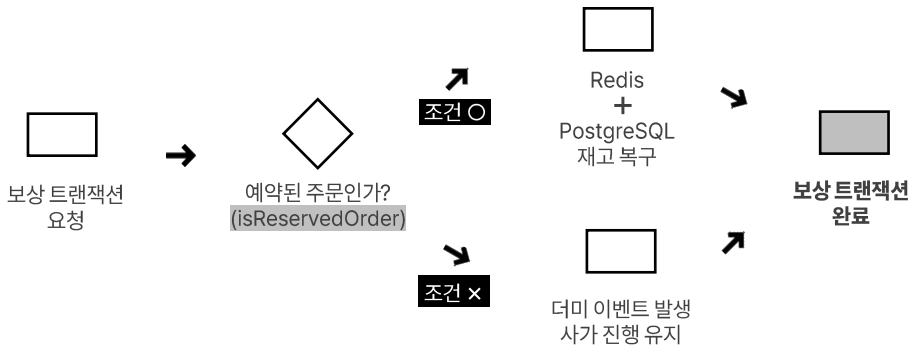


해결 방안

01 트랜잭션 로그 기반 멱등성 보장

주문 예약 상태 검증을 통한 중복 실행 방지 및 안전한 보상 트랜잭션 처리

02 멱등성 보장 플로우



03 구현 핵심 코드 – 멱등성이 보장된 보상 트랜잭션

```
@Transactional
public void refundInventory(ReleaseStockCommand command) {

  UUID productId = UUID.fromString(command.getProductId());
  String orderId = command.getOrderId();
  int quantity = command.getQuantity();

  try {
    if (!isReservedOrder(orderId)) {
        log.info("복구 스킵: 예약되지 않은 주문. 주문 ID: {}", orderId);
        sendStockReleasedResult(orderId, true);
        return;
    }
    boolean released = releaseStockAndUpdateStatus(orderId,
                                                   productId, quantity);
    sendStockReleasedResult(orderId, released);
  } catch (Exception e) {
    log.error("복구 중 오류 발생: 주문 ID: {}, 오류: {}", orderId,
                                                   e.getMessage(), e);
    sendStockReleasedResult(orderId, false);
  }
}
```

기술적 성과

- **주문 상태 기반 멱등성** - 예약 상태 검증을 통한 안전한 보상 트랜잭션
- **예외 처리 강화** - 전체 프로세스에 대한 포괄적 예외 처리 및 로깅
- **응답보장** - 성공/실패 여부를 명확히 반환하여 시스템 안정성 향상

핵심 개선 사항

- ✓ **예약된 상태 검증 로직** - 불필요한 보상 트랜잭션 방지
- ✓ **포괄적 예외 처리** - 시스템 안정성 향상
- ✓ **명확한 응답 처리** - 트랜잭션 결과 보장
- ✓ **구조화 된 로킹** - 디버깅 및 모니터링 개선