최종적 일관성 제공하기 (by Transaction Outbox)

7기 BE 투다

목차

- 발표 대상
- 문제 상황
- 트랜잭션 아웃박스란?
- 생각해보기

발표 대상

외부 API를 사용하면서 기능 가용성을 챙기고 싶은 분

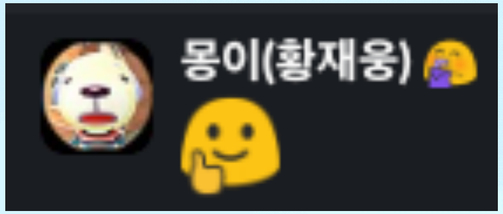
트랜잭션 아웃박스가 무엇인지 궁금하신분

문제 상황

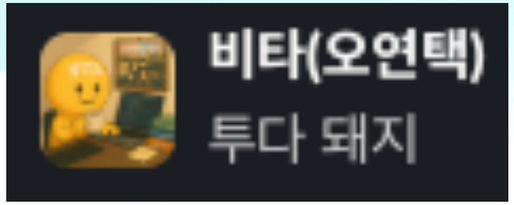
때는 레벨2 어느때..

'Today I Learned(TIL) 작성 플랫폼 MATILDA'을 진행 중..

문제 상황

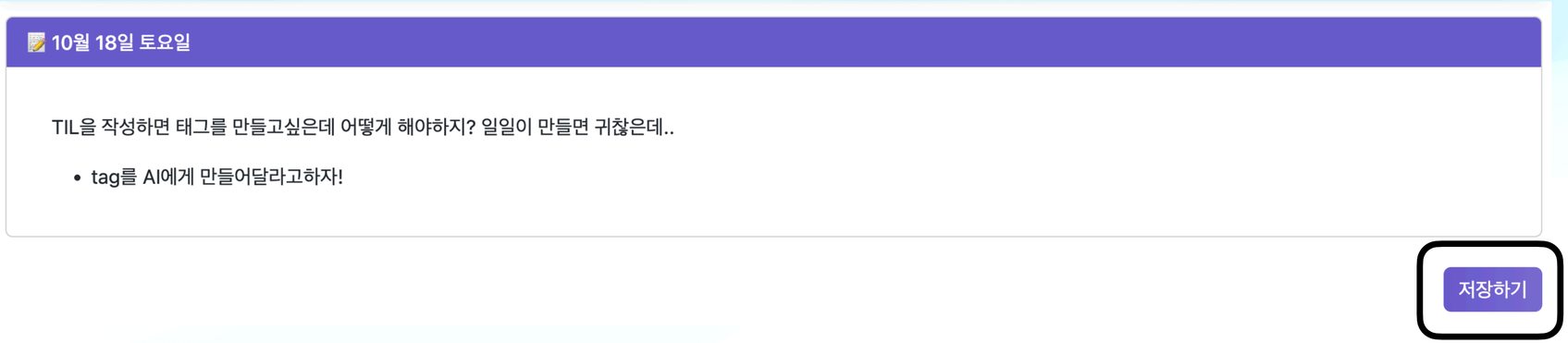


AI API 활용한 기능이 필요하다!



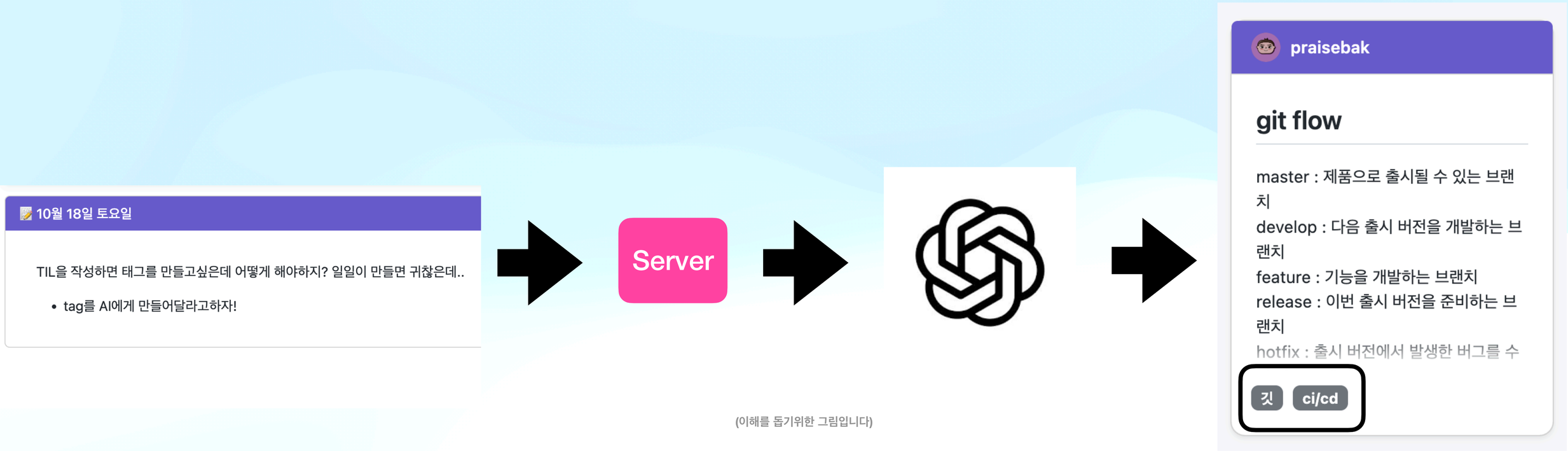
함께 프로젝트를 진행하던 크루들

문제 상황



해당 기능은 TIL 저장시..

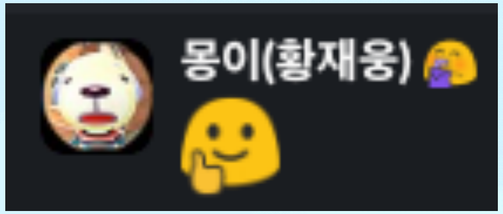
문제 상황

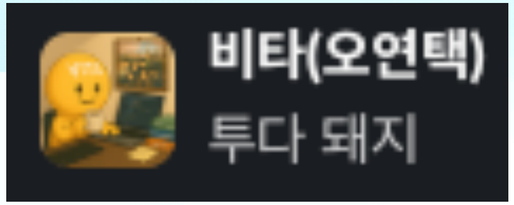


(이해를 돕기위한 그림입니다)

태그가 생성되는 기능!

문제 상황



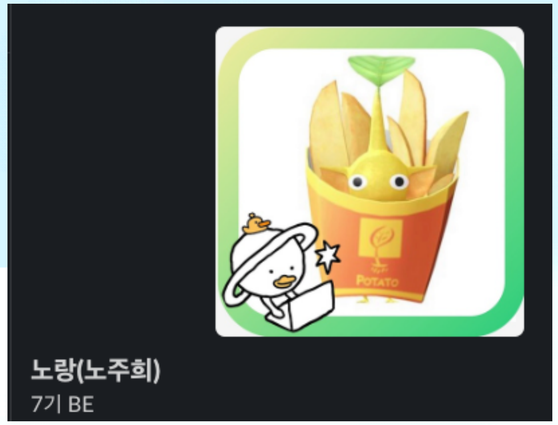


LGTM

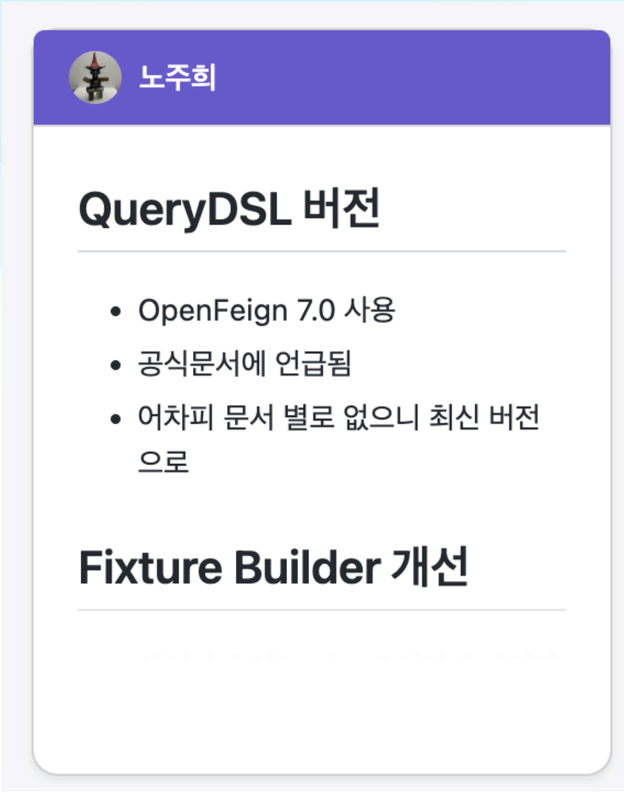
문제 상황

그러던 어느날..

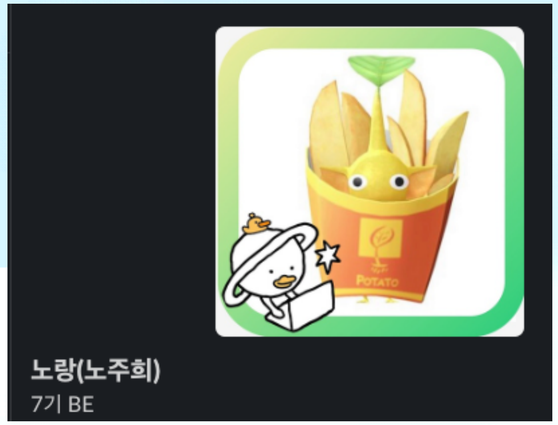
문제 상황



갑자기 태그가 안만들어져요!

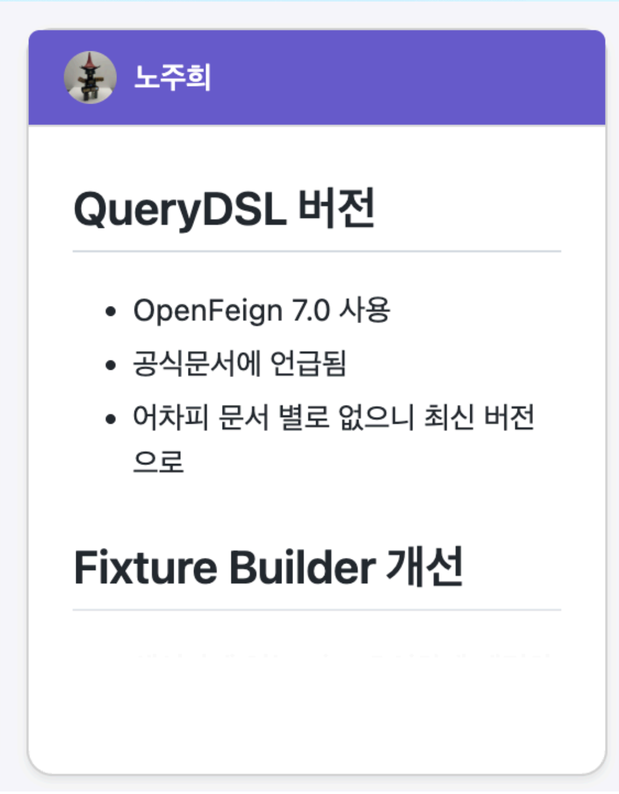


문제 상황

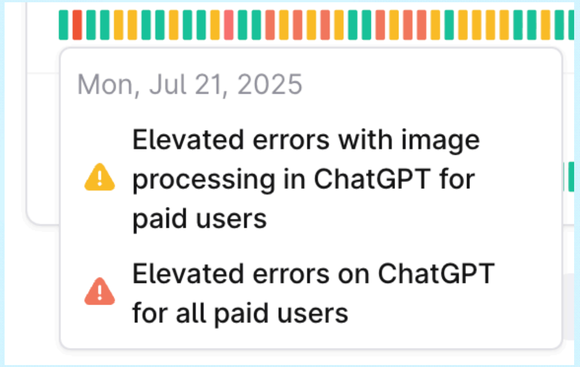


갑자기 태그가 안만들어져요!

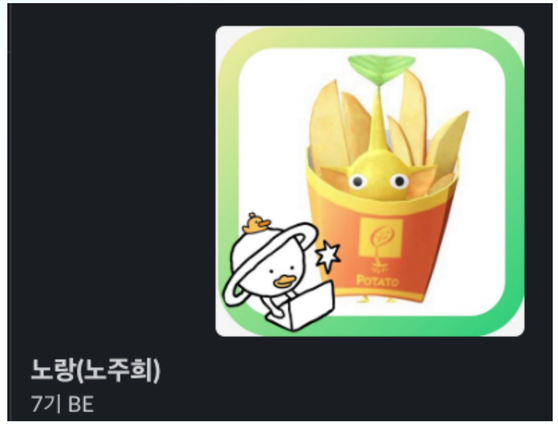
태그 어디갔지..?



태그의 행방



7월 21일 openai 에서 기능이 제공되지 않는 문제가 있었다.



왜 하필 ㅠㅠ

어떻게든 기능 제공하기

어떡하면 간헐적으로 실패하는 외부 API 사용 기능을

안정적으로 제공할 수 있을까?

어떻게든 기능 제공하기

재요청 하면 어떨까?

1. 재시도가 실패할 수 있다
2. 재시도하는 동안 유저는 기다린다
3. 재시도 중 서버 장애시 재시도는 휘발된다

어떻게든 기능 제공하기

2. 요청이 실패했어요

1. 요청 실패시

User

Server

서비스 제공 불가상태

빠른 실패는 어떨까?

1. 유저가 기능을 사용하지 못함
2. 유저 이탈 가능성

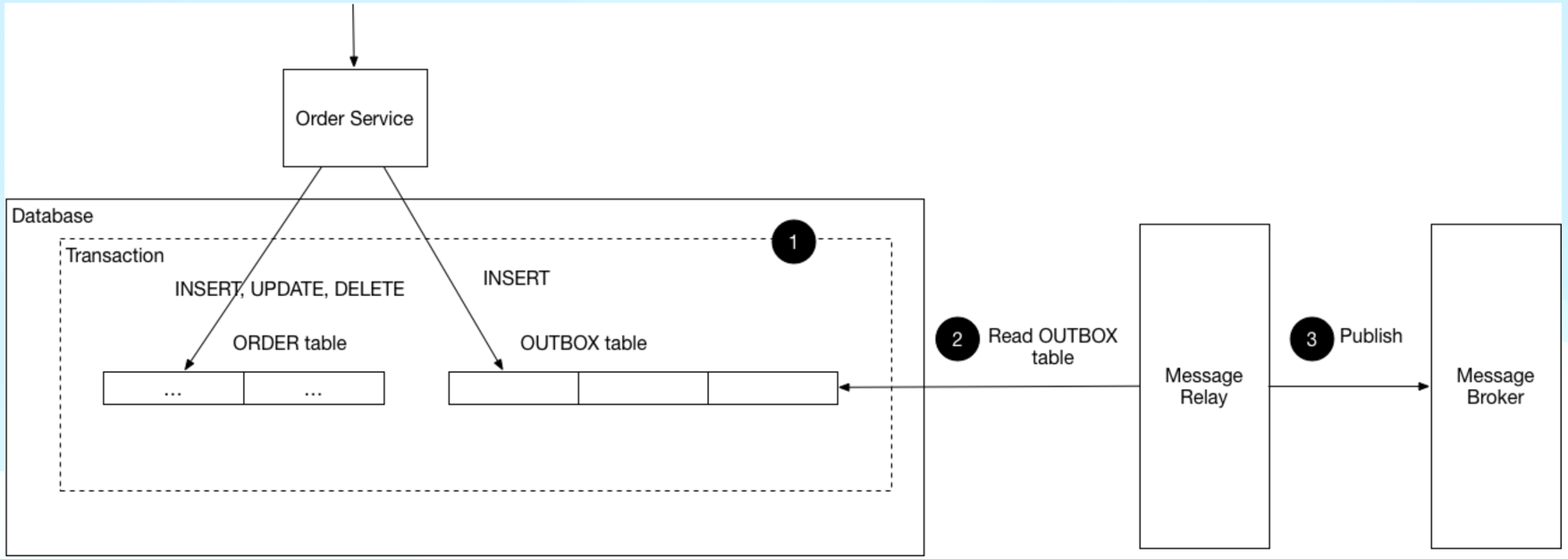
어떻게든 기능 제공하기

"영구적으로" 요청(메시지)를 저장할 방법이 필요한것같아..

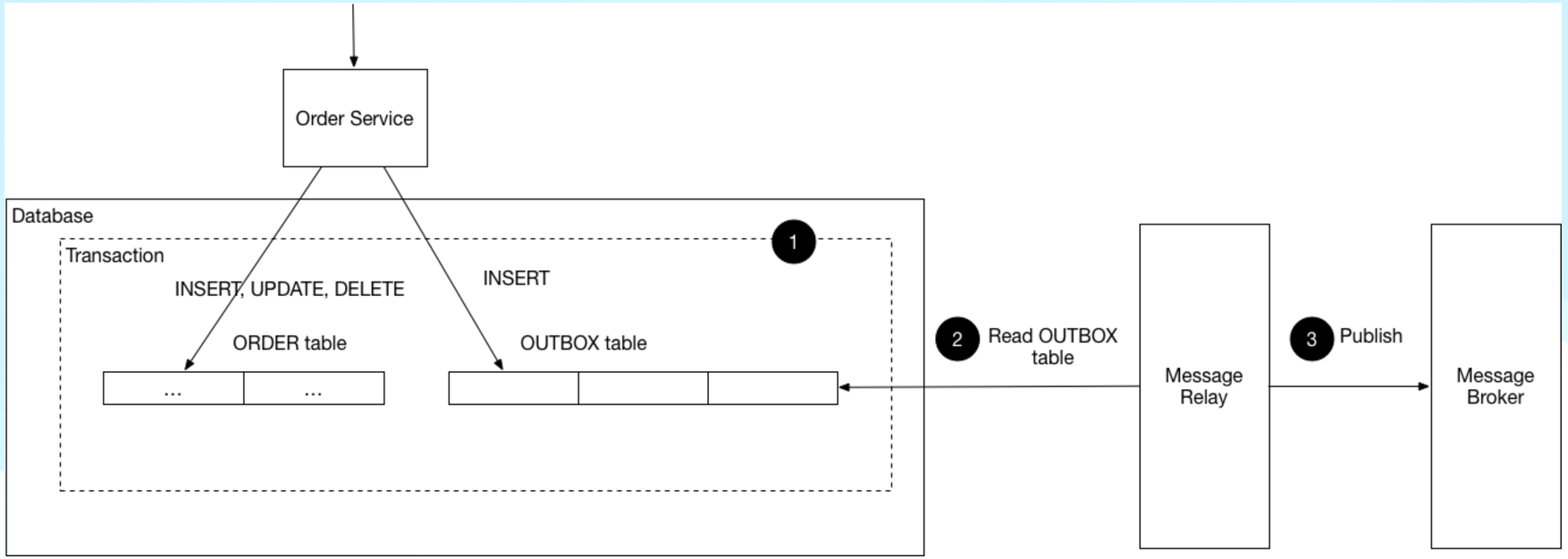
어떻게든 기능 제공하기

제어권을 외부에서 우리로 가져와야해!

트랜잭션 아웃박스

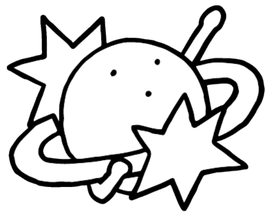

Order Service
Database
Transaction
1
INSERT, UPDATE, DELETE
INSERT
ORDER table
OUTBOX table
...
...
2
Read OUTBOX table
Message Relay
3
Publish
Message Broker

트랜잭션 아웃박스



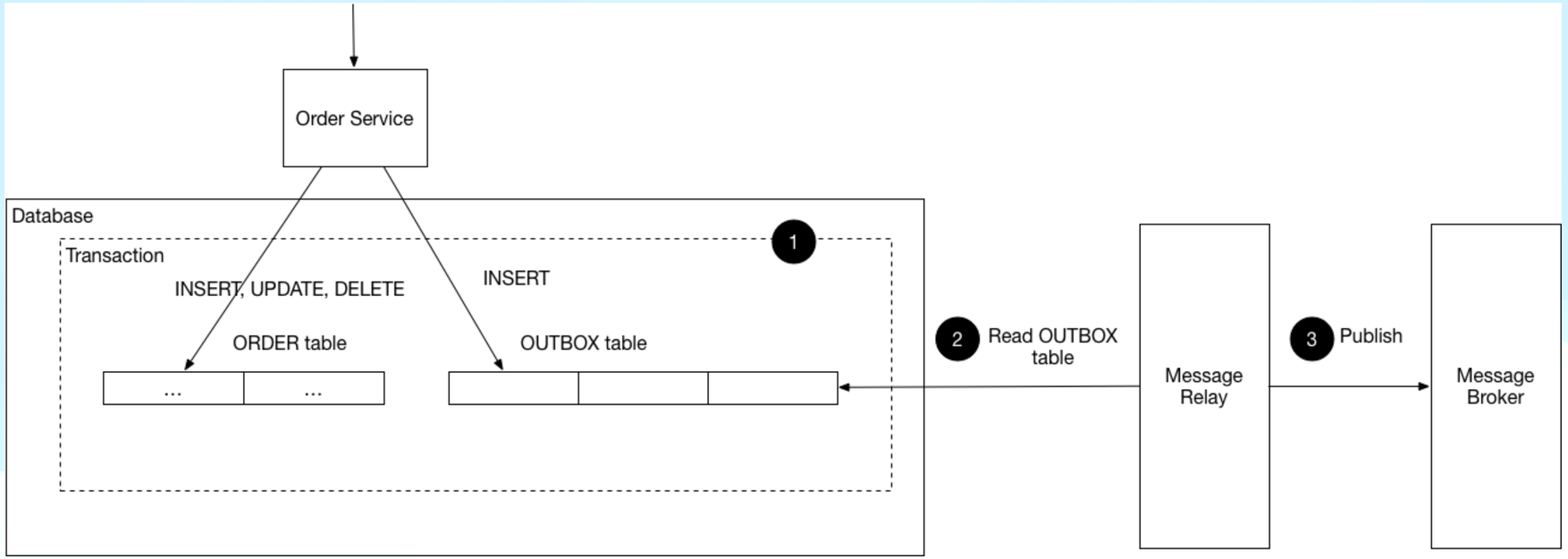
microservices.io에 따르면 다음과 같은 것을 트랜잭션 아웃박스라고 합니다.

> The solution is for the service that sends the message to first store the message in the database as part of the transaction that updates the business entities. A separate process then sends the messages to the message broker.



번역 : 메시지를 전송하는 서비스가 비즈니스 엔티티를 업데이트하는 트랜잭션의 일부로서, 먼저 메시지를 데이터베이스에 저장하는 것이다.
그 후, 별도의 프로세스가 데이터베이스에 저장된 메시지를 메시지 브로커로 전송한다.

트랜잭션 아웃박스

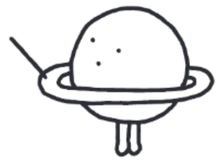


간단하게는, 메시지 전송(외부 api 호출)시에 메시지 그 자체를 데이터베이스에 저장하고, 이후에 전송한다고 볼 수 있습니다.

트랜잭션 아웃박스

외부 API 호출은 제어할 수 없으니

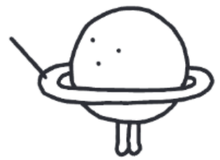
제어할 수 있는 우리 DB에 저장하자!



트랜잭션 아웃박스

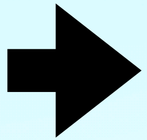
1. DB에 이벤트 자체를 저장한다

2. 스케줄러로 주기적으로 처리한다



트랜잭션 아웃박스

1. **DB에 이벤트 자체를 저장한다**



기능과, 외부 API 호출을 원자적으로 처리하고싶은것

=> 같은 트랜잭션 내에서 처리해야한다

2. 스케줄러로 주기적으로 처리한다

트랜잭션 아웃박스

```java
@Transactional
public Til createTil(final TilDefinitionRequest tilCreateDto, final long userId) {
    boolean exists = tilRepository.existsByDateAndTilUserIdAndIsDeletedFalse(tilCreateDto.date(), userId);
    if (exists) {
        throw new IllegalArgumentException("같은 날에 작성된 게시물이 존재합니다!");
    }

    TilUser user = userService.findById(userId);
    Til newTil = tilCreateDto.toEntity(user);
    Til til = tilRepository.save(newTil);

    tilOutboxService.saveOutbox(new TilCreatedEvent(til.getTilId(), til.getContent(), user.getId()));

    return til;
}
```

기능과 outbox 저장이 같은 트랜잭션 내에서 묶어야한다는 점에 주목

트랜잭

1. DB에

2. 스케줄

```java
@Entity
@Table(name = "tag_creation_outbox_events")
public class TagCreationOutboxEvent {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    // [무엇을?] 태그를 생성할 TIL 게시글의 ID
    @Column(name = "til_id", nullable = false)
    private Long tilId;

    // [어떤 내용으로?] 태그 생성을 위해 필요한 TIL 게시글의 내용
    @Column(name = "til_content", columnDefinition = "TEXT", nullable = false)
    private String tilContent;

    // [상태는?] 이 작업의 현재 처리 상태 (예: PENDING, COMPLETED, FAILED)
    @Enumerated(EnumType.STRING)
    @Column(name = "status", nullable = false)
    private OutboxEventStatus status;
}
```

outbox에는 외부 API를 호출할 수 있는 정보들을 담는다

트랜잭션 아웃박스

1. DB에 이벤트 자체를 저장한다

2. 스케줄러로 주기적으로 처리한다

```java
@Scheduled(fixedDelay = 30000) // 30초마다
@Transactional
public void processPendingEvents() {
    List<TagCreationOutboxEvent> pendingEvents =
        outboxRepository.findPendingEvents(LocalDateTime.now());

    for (TagCreationOutboxEvent event : pendingEvents) {
        try {
            processEvent(event.getId());
        } catch (Exception e) {
            log.error("Failed to process pending event {}", event.getId(),
e);      }
    }
}
```

Pending 상태인 이벤트들을 가져와서 처리한다

(실패시 실패처리한다)

트랜잭션 아웃박스

1. DB에 이벤트 자체를 저장한다

2. **스케줄러로 주기적으로 처리한다**

outbox를 저장했으니, 호출해야한다!

스케줄러를 이용해서 주기적으로 처리하자.

트랜잭션 아웃박스

1. DB에 이벤트 자체를 저장한다

2. **스케줄러로 주기적으로 처리한다**

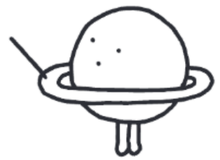
```
    /**
     * 실패한 이벤트들 재시도
     */
    @Scheduled(fixedDelay = 60000) // 1분마다
    @Transactional
    public void retryFailedEvents() {
        List<TagCreationOutboxEvent> retryableEvents =
                outboxRepository.findRetryableFailedEvents(LocalDateTime.now());

        if (!retryableEvents.isEmpty()) {
            log.info("Retrying {} failed tag creation events",
retryableEvents.size());
            for (TagCreationOutboxEvent event : retryableEvents) {
                try {
                    processEvent(event.getId());
                } catch (Exception e) {
                    log.error("Failed to retry event {}", event.getId(), e);
                }
            }
        }
    }
```

Failed 상태인 이벤트들을 가져와서 처리한다

추가로 고려해야할 것들

1. DB에 데이터가 쌓이지 않게 "Cleanup" 해주자
2. 만약 서버가 여러개라면 스케줄링 동기화를 고려하자
3. 메시지가 빨리 쌓인다면 메시지 기술 도입을 고려하자



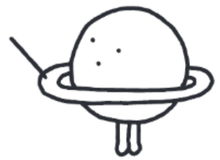
주의할 점

모든 기능에 최종적 일관성을 도입한다고 좋을까?

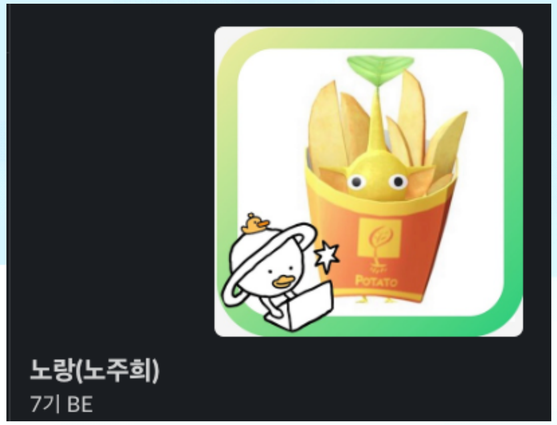
유저에게 즉각적으로 응답을 줘야한다면

빠른 실패 등을 고려하자

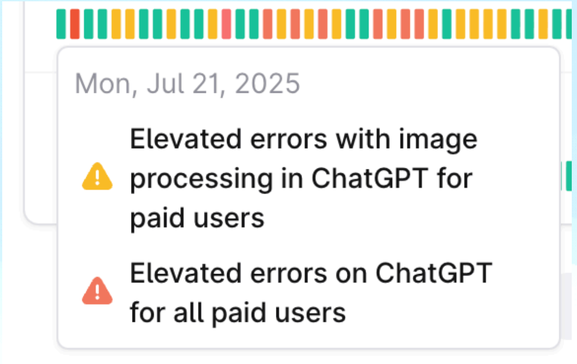
ex) 결제기능 - 결제했는데 돈이 늦게 빠져나갈 수 있음



결과



굿 👍



간헐적 실패에도 기능을 안정적으로 제공할 수 있다.

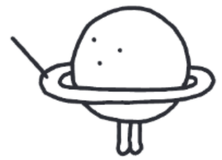
결과

	입력 토큰 100만 개당 가격	입력 토큰 100만 개당 가격(캐시됨)	출력 토큰 100만 개당 가격	가동 시간 SLA[3]	레이턴시 SLA[3]
GPT-5 Long-Context 제외[1]	US$2.50	US$0.250	US$20.00	99.9%	99% > 50개의 토큰/초[2]
GPT-5 mini Long-Context 제외[1]	US$0.45	US$0.045	US$3.60	99.9%	99% > 80개의 토큰/초[2]
GPT-4.1 Long-Context 제외[1]	US$3.50	US$0.875	US$14.00	99.9%	99% > 80개의 토큰/초[2]
GPT-4.1 mini Long-Context 제외[1]	US$0.70	US$0.175	US$2.80	99.9%	99% > 90개의 토큰/초[2]
GPT-4.1 nano Long-Context 제외[1]	US$0.20	US$0.050	US$0.80	99.9%	99% > 100개의 토큰/초[2]
GPT-4o ↳ gpt-4o-2024-11-20 ↳ gpt-4o-2024-08-06	US$4.25	US$2.125	US$17.00	99.9%	99% > 80개의 토큰/초[2]
↳ gpt-4o-2024-05-13	US$8.75	—	US$26.25	99.9%	99% > 80개의 토큰/초[2]
GPT-4o mini	US$0.25	US$0.125	US$1.00	99.9%	99% > 90개의 토큰/초[2]
o3	US$3.50	US$0.875	US$14.00	99.9%	99% > 80개의 토큰/초[2]
o4-mini	US$2.00	US$0.500	US$8.00	99.9%	99% > 90개의 토큰/초[2]

1 프롬프트 토큰 12만 8,000개 초과 시 예상 요청 수
2 5분마다 p50 요청 레이턴시로 계산됩니다. 기존 엔터프라이즈 계약에 1분마다 p50 요청 레이턴시로 계산되는 레이턴시 SLA가 포함된 고객은 기존의 SLA 역시 계속 적용됩니다.
3 Enterprise 고객에게만 적용됩니다

일반적으로 외부 API들은 서비스 수준 목표(SLO)를 제공한다

SLO와 적절한 패턴으로 **최종적 일관성**을 제공할 수 있다.



감사합니다