스터디 트래커의 이벤트 기반 설계 적용과 회고

1. 개요

스터디 트래커는 요약 생성 및 피드백을 제공하는 기능을 중심으로 설계되었다. 현재 시스템은 절차적 동기 흐름이며, 스케줄러에서 사용자별로 요약을 생성하고 슬랙으로 전송한다. 다만, 요약 생성에 실패했을 경우 이를 Redis 큐에 저장하여 재시도하는 구조를 통해 부분적으로 이벤트 지향 아키텍처를 일부 도입했다.

이 보고서는 전체 시스템에 EDA 를 적용했을 때의 구조적 변화, 장단점, 구현 고려사항 등을 분석했다.

2. 문제 인식

기존 구조의 한계

- 실패한 유저를 식별하기 어렵고, 전체 실행 결과를 알 수 없다.
- 슬랙 전송, 요약 생성, DB 저장이 하나의 메서드에 Coupling 되어 있다.
- 확장성/병렬처리에 불리하며, 장애 전파의 가능성이 있다.

3. 이벤트 기반 구조 설계안

현재 시스템은 사용자별로 요약을 순차적으로 생성하고 슬랙으로 전송하는 **동기적 구조**를 따른다. 이벤트 기반 아키텍처(Spring EventPublisher, Kafka 등)는 전체적으로 도입되지 않았지만, **요약 생성 실패 시 Redis 큐를 활용한 재시도 처리 구조**를 통해 **Producer-Consumer 패턴 기반의 이벤트 지향 처리를 부분적으로 구현**했다.

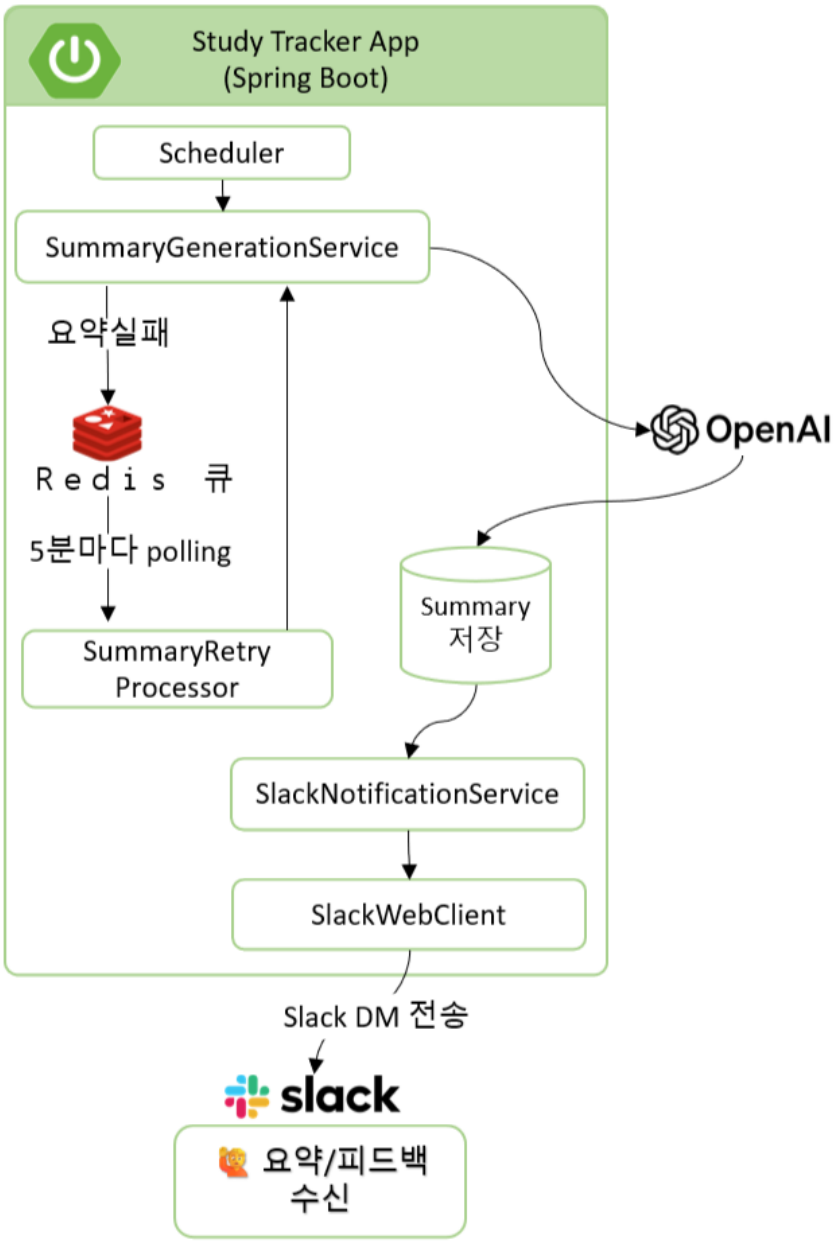
이 섹션에서는 현재 Redis 큐 기반 이벤트 처리 구조를 설명하고, 전체 시스템을 EDA 로 확장했을 때의 구조적 대안을 함께 제시한다.

A. 현재 구현: Redis 큐 기반 이벤트 처리

구성 설명

- 스케줄러가 요약 생성 중 실패한 요청을 Redis 큐(summary-retry-queue)에 저장한다.
- SummaryRetryProcessor 가 큐를 polling 하여 재시도한다.
- 최대 4 회까지 재시도하며, 실패 시 로그에 기록하고 폐기한다.

메시지 흐름 도식



장점

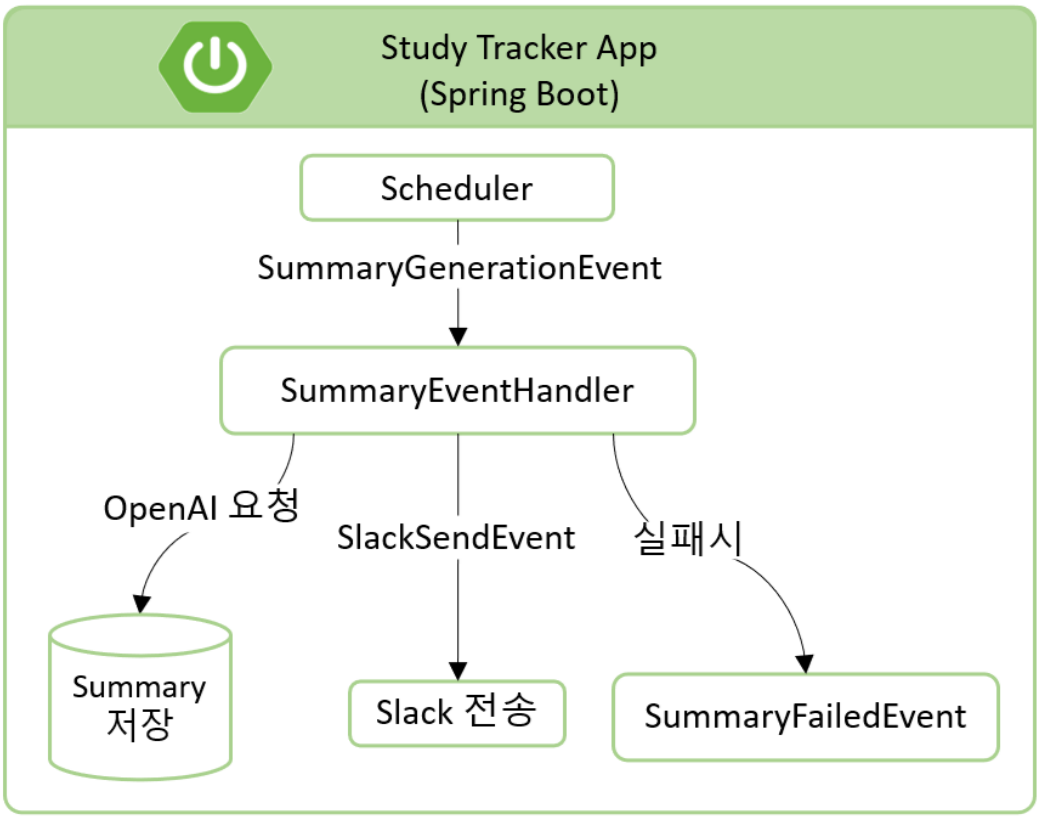
- 실패 요청이 유실되는 걸 방지한다. (at-least-once 보장)
- Redis 기반이므로 메시지의 영속성과 유연성이 확보된다.

B. 대안 설계안: Spring EventPublisher 기반 EDA

구성 설명

- 스케줄러가 각 사용자에 대해 SummaryGenerationEvent 를 발행한다.
- 이벤트 리스너(@EventListener)가 요약 생성과 저장, 슬랙 전송 처리를 한다.
- 처리 결과에 따라 SlackSendEvent , SummaryFailedEvent 등을 후속 발행한다.

이벤트 흐름 예시



장점

- 이벤트별로 책임이 분리된다. (SRP 만족)
- 테스트, 로깅, 트래킹 로직 삽입이 용이하다.
- 메시지 브로커(Kafka, Redis Stream) 도입 시 자연스럽게 확장이 가능하다.

정리 및 비교

항목	REDIS 큐 기반	SPRING EVENT 기반
메시지 저장 위치	✅ 외부 저장소 (Redis)	❌ JVM 메모리 내
메시지 유지	⚠️ 설정 시 유지 가능 (AOF/RDB)	❌ 앱 꺼지면 이벤트 유실
병렬 처리	✅ 가능(컨슈머 직접 구현)	❌ @Async 또는 다중 리스너 설정 필요
추적 및 모니터링	✅ Redis 큐 상태로 확인	❌ 별도 로깅 필요, 흐름 추적 어려움
장점	✅ 실패 보존에 최적화	✅ 확장성 우수, SRP

참고

- 현재 시스템은 전체 EDA 구조는 아니지만 Redis 기반 재시도를 통해 EDA 원칙(at-least-once, producer-consumer pattern)을 반영
- 이벤트 전파 구조(SlackSendEvent, SummaryFailedEvent)는 Redis Stream 이나 Kafka 와 같은 메시지 브로커로 확장 가능

4. 구조별 장단점

항목	현재 구조 (절차 + REDIS 큐)	전체 이벤트 기반 구조 (SPRING 이벤트 또는 메시지 브로커)
구조 단순성	✅ 단일 흐름으로 구성되어 단순	❌ 이벤트 흐름 분리로 인해 복잡

확장성	❌ Redis 큐 기반 재시도 로직에 한정	✅ 이벤트 단위로 유연하게 기능 확장 가능 (슬랙 전송, 통계 생성 등)
실패 유저 추적	❌ Redis 큐 외에는 별도 추적 체계 없음	✅ 이벤트/로그 기반으로 실패 유저 데이터 구조화 가능
병렬 처리	❌ RetryProcessor 가 while 루프로 하나씩 처리 → 병렬 처리 어려움	✅ 이벤트별 @Async 또는 Kafka/Redis Stream 의 Consumer Group 으로 병렬 처리 용이
디버깅/관찰	✅ 흐름이 단일 메서드에 집중되어 추적 용이	❌ 이벤트 체이닝/비동기로 인해 흐름 추적 및 디버깅이 어려울 수 있음

5. EDA 설계 인사이트 요약

- 이벤트 기반 아키텍처는 **구조적 유연성과 SRP 실현에 효과적**이지만, **복잡도 증가와 흐름 추적의 어려움**이라는 비용이 따른다.
- 현재 프로젝트처럼 **작고 단일 책임 흐름이 명확한 구조**에서는 절차적 설계가 더 실용적일 수 있다.
- 하지만 Redis 큐 기반의 실패 재시도 구조를 통해 **at-least-once 메시지 처리, 비동기 흐름의 이점**을 실제로 체감할 수 있었다.
- 이 경험을 통해, 이벤트 기반 설계 구현 시 고려해야 할 트레이드오프(SRP ↔ 디버깅, 확장 ↔ 관찰성)에 대해 깊이 고민할 수 있었다.
- Spring 의 ApplicationEventPublisher + @EventListener 는 **로컬 이벤트 설계를 빠르게 실험해볼 수 있는 좋은 도구**였다.