

2024 역량진단

역량진단_예상문항.docx

Word 문서 · 25KB



A고객사의 시스템 개요

A고객사는 전자상거래 및 티켓팅 시스템을 운영하는 글로벌 기업으로, 다양한 제품과 서비스를 온라인으로 제공하고 있습니다. 고객사는 웹, 모바일 앱, 그리고 제3자 API 등을 통해 고객에게 접근할 수 있는 다채널 플랫폼을 구축하고 있습니다. 최근 사용자 수와 거래량의 급격한 증가로 인해 시스템의 확장성, 유연성, 성능 및 보안 문제가 대두되고 있으며, 이를 해결하기 위한 종합적인 개선 방안이 요구되고 있습니다.

1. 인증 및 세션 관리 개선 방안

업무 요건

- 보안 강화: 사용자 인증 과정에서 발생할 수 있는 보안 취약점을 최소화하고, 사용자의 신원을 확실하게 검증할 수 있는 방안을 도입해야 합니다. 특히, 최근 발생하는 세션 탈취 및 피싱 공격에 대비한 다중 요소 인증(MFA) 도입이 필요합니다.
- 성능 최적화: 증가하는 사용자 수와 동시 접속자 수에 대비하여, 인증 및 세션 관리의 성능을 최적화해야 합니다. 기존의 중앙 집중식 세션 관리에서 벗어나, 분산 세션 관리 시스템을 통해 성능 병목을 해소해야 합니다.
- 유연한 채널 통합: 웹, 모바일, API 등 다양한 채널에서 일관된 인증 경험을 제공해야 하며, 각 채널에 적합한 인증 방식을 통합하여 관리해야 합니다.

시나리오

- 시나리오 1-1: OAuth 2.0 및 OpenID Connect 도입

- 배경: A고객사는 다양한 디바이스와 채널을 통해 사용자가 접근하는 복잡한 환경을 가지고 있습니다. 기존의 인증 방식은 단일한 로그인 정보로 모든 채널에 접근할 수 있도록 설계되지 않았습니다.
- 문제: 사용자가 한 채널에서 인증된 후 다른 채널로 이동할 때, 세션이 유지되지 않거나 추가 인증이 요구되는 불편함이 발생하고 있습니다. 이로 인해 사용자 경험이 저하되고 있습니다.
- 해결 방안: OAuth 2.0과 OpenID Connect를 도입하여, 통합된 인증 체계를 구축합니다. 이를 통해 사용자는 한 번의 인증으로 모든 채널에 접근할 수 있으며, 사용자 정보를 안전하게 보호할 수 있습니다. 또한, 다중 요소 인증(MFA)을 도입하여 보안을 강화합니다.

- 시나리오 1-2: 분산 세션 관리 시스템

- 배경: 고객사의 사용자 수가 증가함에 따라 기존의 세션 관리 시스템에서 병목 현상이 발생하고 있습니다. 특히, 이벤트나 프로모션 기간 동안 동시 접속자가 급증하여 인증 속도가 크게 저하되는 문제가 발생합니다.
- 문제: 현재의 세션 관리 방식은 중앙 집중식 구조로 인해, 서버 과부하 및 성능 저하가 발생하고 있습니다.
- 해결 방안: 분산 세션 관리 시스템을 도입하여, 세션 정보를 여러 서버에 분산시켜 관리합니다. 이를 통해 서버 부하를 분산시키고, 세션 관리의 신뢰성과 확장성을 향상시킵니다.

2. 상품 관리 데이터 모델 설계

업무 요건

- 데이터 모델 유연성: 다양한 상품 속성(예: 크기, 색상, 브랜드 등)을 효율적으로 관리할 수 있는

데이터 모델이 필요합니다. 상품 속성은 동적으로 추가되거나 변경될 수 있어야 하며, 새로운 상품 카테고리나 프로모션 조건에도 유연하게 대응할 수 있어야 합니다.

- 성능 최적화: 대규모 상품 데이터를 처리하는 과정에서 성능 저하를 방지하기 위해, 데이터베이스 설계 단계에서부터 최적화를 고려해야 합니다. 특히, 상품 검색 및 필터링 기능의 성능이 중요한 과제입니다.

- 통합 재고 관리: 상품 관리와 재고 관리 시스템의 통합이 필요합니다. 실시간 재고 관리를 통해 정확한 재고 정보를 제공해야 하며, 재고 부족 시 자동으로 주문이 중단되거나 대체 상품을 추천하는 기능이 필요합니다.

시나리오

- 시나리오 2-1: NoSQL 기반의 상품 관리 시스템 도입

- 배경: A고객사는 다양한 속성을 가진 상품을 관리해야 하며, 특히 속성이 자주 변경되는 제품군(예: 패션, 전자제품 등)을 다룹니다.

- 문제: 기존 RDBMS 기반의 데이터 모델은 정적이며, 새로운 속성을 추가하거나 상품의 구조를 변경하는 데 한계가 있습니다. 또한, 데이터베이스의 조인 연산이 많아지면서 성능 저하가 발생하고 있습니다.

- 해결 방안: NoSQL 데이터베이스(예: MongoDB, Cassandra)를 도입하여, 비정형 데이터와 동적 속성 관리를 지원합니다. 이를 통해 상품 속성을 유연하게 관리할 수 있으며, 분산 처리로 검색 성능을 향상시킬 수 있습니다.

- 시나리오 2-2: 하이브리드 데이터베이스 구조 설계

- 배경: 다양한 종류의 상품 데이터를 통합적으로 관리하고, 실시간 재고 관리를 효율적으로 수행해야 합니다.

- 문제: 기존 시스템에서 RDBMS와 NoSQL의 장점을 모두 활용하지 못하고 있어, 특정 작업에서 성능 저하가 발생합니다.

- 해결 방안: RDBMS와 NoSQL을 혼합한 하이브리드 데이터베이스 구조를 설계합니다. RDBMS는 핵심 트랜잭션과 관계형 데이터를 처리하고, NoSQL은 비정형 데이터와 고속 검색을 처리하는 역할을 합니다. 이 구조는 성능과 유연성을 동시에 제공할 수 있습니다.

3. 예약 처리 프로세스와 동시성 문제 해결

업무 요건

- 동시성 제어: 다수의 사용자가 동시에 예약 및 주문을 시도할 때, 데이터의 일관성을 유지하고 중복 예약을 방지할 수 있어야 합니다. 특히, 한정된 재고를 가진 상품의 예약 시, 동시성 문제로 인해 발생할 수 있는 불필요한 트랜잭션 및 중복 예약 문제를 해결해야 합니다.

- 실시간 트랜잭션 관리: 예약 및 주문 프로세스는 실시간으로 처리되어야 하며, 트랜잭션의 성공 여부를 즉시 사용자에게 피드백해야 합니다. 이를 위해, 분산 트랜잭션 관리와 데이터 잠금(Locking) 메커니즘을 효율적으로 사용해야 합니다.

- 트랜잭션 복구: 동시성 문제로 인해 발생하는 트랜잭션 오류를 복구할 수 있는 메커니즘을 마련해야 합니다. 실패한 트랜잭션을 자동으로 롤백하거나 재시도할 수 있는 기능이 필요합니다.

시나리오

- 시나리오 3-1: 분산 트랜잭션 관리 시스템 도입

- 배경: A고객사는 매일 수천 건의 예약 및 주문을 처리하며, 이벤트나 프로모션 기간에는 트랜잭션이 급증합니다.

- 문제: 다수의 사용자가 동일한 상품을 동시에 예약하려고 시도할 때, 트랜잭션이 충돌하거나 중복 예약이 발생하는 문제가 보고되고 있습니다.

- 해결 방안: 분산 트랜잭션 관리 시스템을 도입하여, 각 트랜잭션을 분산된 서버에서 병렬로 처리하고, 충돌을 방지하기 위해 트랜잭션 수준의 잠금(Locking) 메커니즘을 구현합니다. 이 시스템은 트랜잭션 실패 시 자동 롤백 기능을 제공하여, 데이터 일관성을 유지합니다.

- 시나리오 3-2: 이벤트 기반 비동기 처리

- 배경: 대규모 이벤트나 프로모션이 발생할 때, 동시성 문제가 더욱 빈번하게 발생합니다. 예약이 시작되는 시간에 수많은 사용자가 한꺼번에 몰리며, 시스템의 부하가 급증합니다.
- 문제: 현재의 동기 처리 방식에서는 트랜잭션 병목이 발생하여, 예약 시스템의 성능이 저하되고 사용자가 대기 시간을 겪습니다.
- 해결 방안: 이벤트 기반의 비동기 처리 방식을 도입하여, 예약 요청을 큐(queue)에 넣고 순차적으로 처리합니다. 이를 통해 트랜잭션 충돌을 방지하고, 사용자는 요청이 접수된 후 대기하는 동안 진행 상태를 실시간으로 확인할 수 있습니다. 또한, 필요 시 예약 완료 후 알림을 전송하는 비동기 알림 시스템을 함께 도입하여, 사용자 경험을 개선합니다. 이 방식은 트랜잭션이 실패하거나 충돌이 발생할 경우 자동으로 재시도하거나 롤백할 수 있는 기능도 포함하고 있습니다.

4. 추가적인 고려사항

업무 요건

- 사용자 경험(UX) 최적화: A고객사는 다양한 디바이스에서 일관된 사용자 경험을 제공해야 합니다. 특히, 모바일 사용자들이 빠르게 증가하고 있으므로, 모바일 환경에 최적화된 UI/UX 설계가 필요합니다. 반응형 웹 디자인과 네이티브 모바일 앱의 UX를 통합하여, 사용자 인터페이스를 직관적이고 사용하기 쉽게 설계해야 합니다.
- 로그 및 모니터링 시스템 강화: 시스템의 안정성을 유지하고 장애 발생 시 신속하게 대응하기 위해, 로그 수집 및 모니터링 시스템을 강화해야 합니다. 실시간으로 트랜잭션 상태와 시스템 성능을 모니터링하고, 장애 발생 시 자동으로 알림을 전송하는 시스템을 구축해야 합니다. 이를 통해, 잠재적인 문제를 조기에 발견하고, 시스템 가용성을 최대화할 수 있습니다.
- 데이터 보안 및 개인정보 보호: A고객사는 글로벌 기업으로서, 다양한 국가의 개인정보 보호 규정을 준수해야 합니다. 특히, 유럽의 GDPR, 미국의 CCPA 등 엄격한 데이터 보호 법규를 준수하기 위해, 고객 데이터의 암호화, 접근 제어, 데이터 익명화 등의 보안 조치를 강화해야 합니다. 또한, 개인정보 처리 방침을 명확히 하고, 사용자에게 투명하게 공개해야 합니다.

시나리오

- 시나리오 4-1: 반응형 웹 및 모바일 UX 통합

- 배경: A고객사는 다양한 디바이스(PC, 태블릿, 스마트폰)에서 동일한 쇼핑 경험을 제공하고자 합니다. 현재의 웹 인터페이스는 PC에 최적화되어 있지만, 모바일 디바이스에서는 사용이 불편한 경우가 많습니다.
- 문제: 모바일 사용자의 증가에 따라, 사용자가 모바일에서 쇼핑하거나 예약을 진행하는 과정에서 UI가 불편하거나 로딩 속도가 느려지는 문제가 발생하고 있습니다.
- 해결 방안: 반응형 웹 디자인을 도입하여, 모든 디바이스에서 일관된 사용자 경험을 제공할 수 있도록 UI/UX를 개선합니다. 또한, 네이티브 모바일 앱에서의 UX를 최적화하고, 빠른 로딩 속도와 직관적인 네비게이션을 제공하도록 설계합니다.

- 시나리오 4-2: 실시간 로그 수집 및 모니터링 시스템 구축

- 배경: A고객사의 시스템에서 발생하는 트랜잭션, 사용자 활동, 시스템 성능 등의 데이터를 실시간으로 모니터링하고 분석할 수 있어야 합니다. 특히, 장애 발생 시 빠르게 대응하기 위해서는 실시간 로그 분석이 필요합니다.
- 문제: 현재의 로그 수집 및 모니터링 시스템은 장애 발생 시 신속하게 원인을 분석하거나 대응하기 어렵습니다. 실시간 분석 기능이 부족하여, 문제 발생 후에야 원인을 파악할 수 있습니다.
- 해결 방안: ELK 스택(Elasticsearch, Logstash, Kibana)과 같은 실시간 로그 수집 및 분석 도구를 도입하여, 시스템에서 발생하는 모든 이벤트를 실시간으로 모니터링합니다. 또한, 이상 징후가 감지되면 자동으로 알림을 전송하는 기능을 추가하여, 장애에 신속하게 대응할 수 있도록 합니다.

- 시나리오 4-3: 개인정보 보호 및 데이터 보안 강화

- 배경: A고객사는 전 세계적으로 서비스를 제공하고 있으며, 각국의 데이터 보호 규정을 준수해야 합니다. 특히, GDPR, CCPA 등의 규정을 준수하여 고객의 개인정보를 안전하게 보호할 필요가 있습니다.
- 문제: 개인정보 보호와 관련하여 다양한 국가의 규정을 준수하기 위해, 데이터 암호화 및 접근 제어 시스템이 강화되어야 합니다. 또한, 사용자 데이터의 익명화와 개인정보 처리 방침의 투명성도 중요한 과제입니다.
- 해결 방안: 데이터 암호화 기술(AES-256)과 접근 제어 정책을 강화하여, 고객의 개인정보를 안전하게 보호합니다. 또한, 개인정보의 사용 및 처리 방침을 투명하게 공개하고, 사용자로부터 명확한 동의를 얻는 절차를 도입합니다. 이를 통해, 글로벌 규제에 대응하고 고객의 신뢰를 유지할 수 있습니다.

서비스 도출

서비스 이름	설명	주요 기능
인증 서비스	OAuth 2.0 및 OpenID Connect를 기반으로 통합된 인증 체계를 제공하며, 다중 요소 인증(MFA) 지원.	사용자 인증, 다중 요소 인증(MFA), 토큰 관리
세션 관리 서비스	분산 세션 관리 시스템을 통해 성능 최적화 및 보안 강화. 다양한 채널에서 일관된 세션 관리 제공.	세션 생성 및 관리, 분산 세션 저장, 세션 보안 강화
상품 관리 서비스	NoSQL 및 하이브리드 데이터베이스 구조를 기반으로 상품 속성, 카테고리, 가격, 재고 등을 효율적으로 관리.	상품 속성 관리, 카테고리 관리, 가격 및 재고 관리
재고 관리 서비스	상품의 재고를 실시간으로 관리하고, 재고 부족 시 주문을 중단하거나 대체 상품을 추천하는 기능 제공.	재고 실시간 업데이트, 재고 부족 시 알림 및 대체 상품 추천
예약 관리 서비스	다수의 사용자가 동시에 예약 및 주문을 처리할 때 데이터 일관성을 유지하고, 중복 예약을 방지하는 서비스.	예약 처리, 예약 확인 및 취소, 중복 예약 방지
트랜잭션 관리 서비스	분산 트랜잭션 관리 시스템을 통해 트랜잭션 충돌을 방지하고, 트랜잭션 실패 시 자동 롤백 기능 제공.	트랜잭션 분산 처리, 트랜잭션 롤백, 트랜잭션 모니터링

로그 및 모니터링 서비스	실시간 로그 수집 및 모니터링 시스템을 통해 트랜잭션 상태, 시스템 성능 등을 모니터링하며, 장애 발생 시 자동으로 알림을 전송.	실시간 로그 수집, 시스템 성능 모니터링, 장애 알림
데이터 보안 서비스	데이터 암호화, 접근 제어, 데이터 익명화 등을 통해 개인정보 보호 규정을 준수하며, 사용자 데이터의 안전을 보장.	데이터 암호화, 접근 제어, 데이터 익명화, 개인정보 처리 방침 관리
사용자 경험(UX) 서비스	반응형 웹 디자인 및 모바일 UX 최적화로 모든 디바이스에서 일관된 사용자 경험을 제공.	UI/UX 디자인, 반응형 웹 디자인, 모바일 UX 최적화

서비스별 API 기능 정의

1. 인증 서비스

API 기능	설명	동기/비동기	트랜잭션 유형	보상 트랜잭션	필요 보상 트랜잭션
사용자 인증 API	사용자의 인증 요청을 받아 인증을 처리하고, 토큰을 발급	동기	피봇	인증 취소 API	인증 실패 시 인증 취소 필요
인증 취소 API	잘못된 인증 시 발급된 토큰을 취소	동기	보상 가능	없음	없음
다중 요소 인증 API	추가적인 보안 인증 절차를 수행하고, 인증 상태를 갱신	동기	재시도 가능	인증 취소 API	인증 실패 시 인증 취소 필요
토큰 관리 API	사용자의 토큰 상태를 확인하고, 토큰을 갱신 또는 삭제	동기	피봇	토큰 무효화 API	토큰 갱신 실패 시 무효화 필요

토큰 무효화 API	만료된 토큰이나 잘못된 토큰을 무효화	동기	보상 가능	없음	없음
------------	----------------------	----	-------	----	----

2. 세션 관리 서비스

API 기능	설명	동기/비동기	트랜잭션 유형	보상 트랜잭션	필요 보상 트랜잭션
세션 생성 API	사용자의 세션을 생성하고 관리하기 위한 초기화 작업	동기	피봇	세션 삭제 API	세션 생성 실패 시 삭제 필요
세션 삭제 API	세션이 만료되거나 오류가 발생한 경우 세션을 삭제	동기	보상 가능	없음	없음
분산 세션 저장 API	분산된 시스템에 세션 데이터를 저장하고, 동기화 수행	동기	재시도 가능	세션 삭제 API	저장 실패 시 삭제 필요
세션 보안 강화 API	세션에 대한 보안 설정을 강화하고, 암호화 및 접근 제어 수행	동기	피봇	세션 무효화 API	설정 실패 시 무효화 필요
세션 무효화 API	보안 위험이 감지된 세션을 무효화	동기	보상 가능	없음	없음

3. 상품 관리 서비스

API 기능	설명	동기/비동기	트랜잭션 유형	보상 트랜잭션	필요 보상 트랜잭션
--------	----	--------	---------	---------	------------

상품 등록 API	새로운 상품을 등록하고 관련 속성 및 데이터를 저장	동기	피봇	상품 삭제 API	등록 실패 시 삭제 필요
상품 삭제 API	잘못 등록된 상품을 삭제	동기	보상 가능	없음	없음
상품 속성 관리 API	상품의 속성을 추가, 수정, 삭제하며 관련 데이터를 업데이트	동기	재시도 가능	속성 복구 API	속성 변경 실패 시 복구 필요
속성 복구 API	잘못된 속성 변경이 발생한 경우 원래 상태로 복구	동기	보상 가능	없음	없음
가격 및 재고 관리 API	상품의 가격과 재고 상태를 관리하고, 필요한 경우 이를 갱신	동기	피봇	가격/재고 복구 API	갱신 실패 시 복구 필요
가격/재고 복구 API	잘못된 가격 변경이나 재고 업데이트가 발생한 경우 원래 상태로 복구	동기	보상 가능	없음	없음

4. 재고 관리 서비스

API 기능	설명	동기/비동기	트랜잭션 유형	보상 트랜잭션	필요 보상 트랜잭션
재고 업데이트 API	실시간으로 재고를 업데이트하며, 재고 부족 시 경고를 전송	동기	피봇	재고 복구 API	업데이트 실패 시 복구 필요

재고 복구 API	잘못된 재고 업데이트가 발생한 경우 원래 상태로 복구	동기	보상 가능	없음	없음
대체 상품 추천 API	재고 부족 시, 대체 가능한 상품을 추천하고 관련 데이터를 전송	비동기	재시도 가능	추천 취소 API	추천 실패 시 취소 필요
추천 취소 API	잘못된 추천이 발생한 경우 추천을 취소	동기	보상 가능	없음	없음

5. 예약 관리 서비스

API 기능	설명	동기/비동기	트랜잭션 유형	보상 트랜잭션	필요 보상 트랜잭션
예약 생성 API	사용자의 예약 요청을 받아 처리하고, 예약 정보를 저장	동기	피봇	예약 취소 API	예약 실패 시 취소 필요
예약 취소 API	잘못된 예약이나 실패한 예약을 취소	동기	보상 가능	없음	없음
예약 확인 API	예약이 성공적으로 완료되었는지 확인하고, 사용자에게 확인 메시지 전송	동기	재시도 가능	확인 취소 API	확인 실패 시 취소 필요
확인 취소 API	잘못된 예약 확인을 취소	동기	보상 가능	없음	없음

6. 트랜잭션 관리 서비스

API 기능	설명	동기/비동기	트랜잭션 유형	보상 트랜잭션	필요 보상 트랜잭션
--------	----	--------	---------	---------	------------

트랜잭션 처리 API	분산 트랜잭션을 처리하고, 트랜잭션 상태를 관리	동기	피봇	트랜잭션 롤백 API	처리 실패 시 롤백 필요
트랜잭션 롤백 API	실패한 트랜잭션을 원래 상태로 롤백	동기	보상 가능	없음	없음
트랜잭션 모니터링 API	실시간으로 트랜잭션 상태를 모니터링하 고, 이상 상태를 감지	비동기	재시도 가능	모니터링 복구 API	모니터링 실패 시 복구 필요
모니터링 복구 API	잘못된 트랜잭션 모니터링 데이터를 복구	동기	보상 가능	없음	없음

7. 로그 및 모니터링 서비스

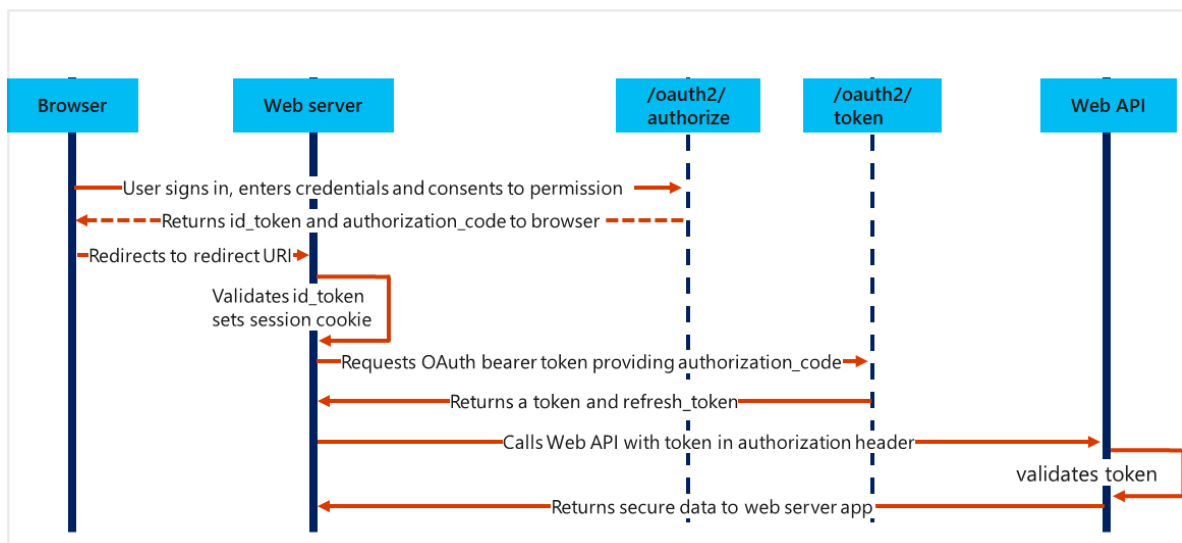
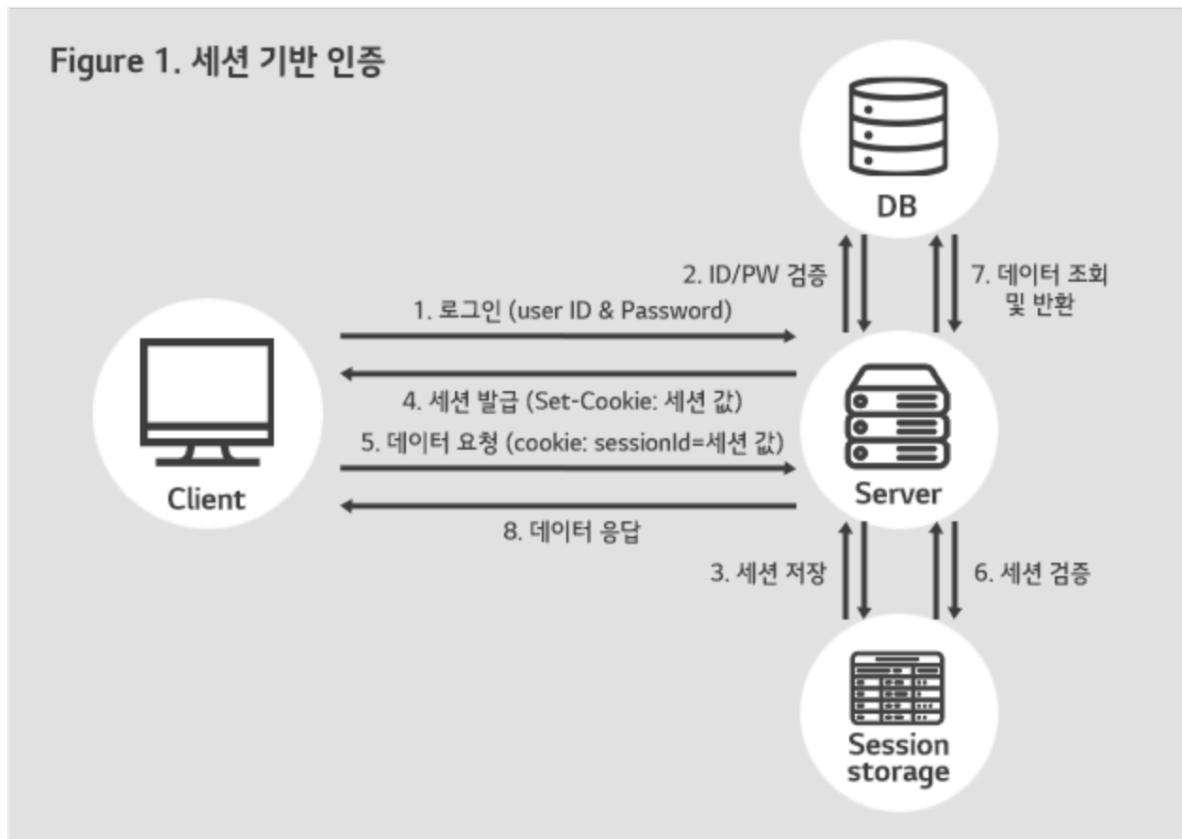
API 기능	설명	동기/비동기	트랜잭션 유형	보상 트랜잭션	필요 보상 트랜잭션
로그 수집 API	시스템에서 발생하는 로그를 실시간으로 수집	비동기	피봇	로그 복구 API	수집 실패 시 복구 필요
로그 복구 API	잘못된 로그 수집 데이터를 복구	동기	보상 가능	없음	없음
시스템 모니터링 API	시스템 성능 및 상태를 실시간으로 모니터링	비동기	재시도 가능	모니터링 복구 API	모니터링 실패 시 복구 필요
장애 알림 API	장애 발생 시 자동으로 알림을 전송	동기	보상 가능	없음	없음

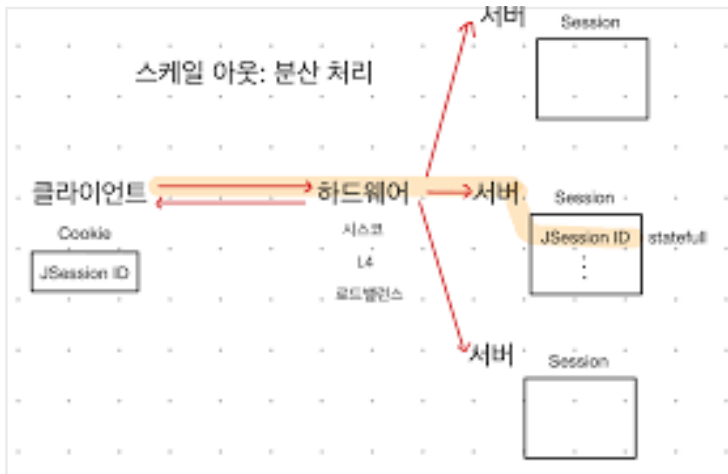
8. 데이터 보안 서비스

API 기능	설명	동기/비동기	트랜잭션 유형	보상 트랜잭션	필요 보상 트랜잭션
--------	----	--------	------------	------------	---------------

데이터 암호화 API	사용자 데이터를 안전하게 보호하기 위해 데이터 암호화 처리	동기	피봇	암호화 복구 API	암호화 실패 시 복구 필요
암호화 복구 API	암호화 처리 중 오류가 발생한 경우 데이터를 원래 상태로 복구	동기	보상 가능	없음	없음
접근 제어 API	시스템 접근 제어를 강화하고, 보안 정책을 관리	동기	피봇	접근 제어 복구 API	접근 제어 실패 시 복구 필요
접근 제어 복구 API	접근 제어 중 오류가 발생한 경우 원래 상태로 복구	동기	보상 가능	없음	없음
데이터 익명화 API	민감한 데이터를 익명화하여 개인정보를 보호	동기	피봇	익명화 복구 API	익명화 실패 시 복구 필요
익명화 복구 API	익명화 처리 중 오류가 발생한 경우 데이터를 원래 상태로 복구	동기	보상 가능	없음	없음
개인정보 처리 방침 관리 API	개인정보 처리 방침을 관리하고, 사용자 동의를 관리	동기	피봇	방침 복구 API	방침 처리 실패 시 복구 필요

방침 복구 API	개인정보 처리 방침 업데이트 중 오류가 발생한 경우 복구	동기	보상 가능	없음	없음
-----------	---------------------------------	----	-------	----	----





[문1] 아래와 같은 모바일쿠폰의 사용(인증)처리 프로세스를 설계하시오.

<문항 지정 전제조건>

- 자사의 프랜차이즈 매장에서 사용할 수 있는 자체 발행 쿠폰이다.
- 쿠폰은 지정 상품과 1:1교환하는 교환권, 원하는 금액만큼 차감하며 사용하는 잔액관리형이 있다.
- 잔액관리형은 충전식이 아닌 금액 소진시 소멸성이며, 부분환불은 지원하지 않는다.
- 교환권은 같은 금액의 타 메뉴로 변경이 가능하다. 액면금액 이하 교환시 남은 금액은 환불되지 않는다.
- 오프라인 매장과 온라인 사이트에서 모두 사용이 가능하다.
-

문1-1) 위 쿠폰의 사용/환불 유즈케이스와 시퀀스 다이어그램을 정리하시오.

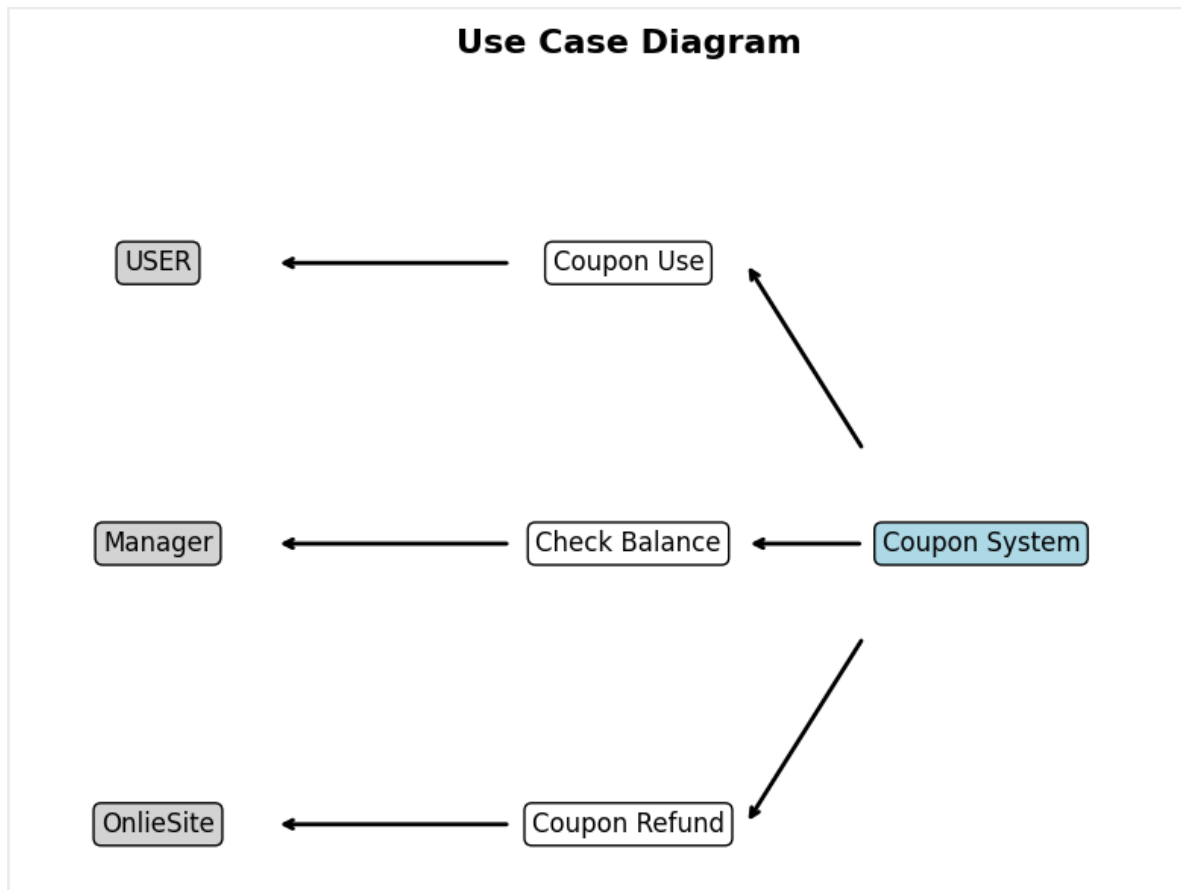
1. 유즈케이스 다이어그램

액터

- 고객: 쿠폰을 사용하는 주체.
- 매장 직원: 오프라인 매장에서 쿠폰을 처리하는 주체.
- 온라인 사이트: 온라인에서 쿠폰을 처리하는 시스템.
- 쿠폰 시스템: 쿠폰의 상태를 관리하고, 사용 및 환불을 처리하는 시스템.

유즈케이스

1. 쿠폰 사용:
 - 교환권 사용 (오프라인/온라인)
 - 잔액관리형 쿠폰 사용 (오프라인/온라인)
2. 쿠폰 환불:
 - 교환권 환불 (오프라인/온라인)
 - 잔액관리형 쿠폰 환불 (오프라인/온라인)
 - 부분환불은 지원하지 않음
3. 잔액 확인:
 - 잔액관리형 쿠폰의 잔액 확인 (오프라인/온라인)



고객. 쿠폰사용
 매장직원 잔액확인 쿠폰시스템
 온라인사이트 쿠폰환불

관계

- 고객은 "쿠폰 사용", "잔액 확인" 유즈케이스에 참여합니다.
- 매장 직원은 "쿠폰 사용", "잔액 확인", "환불" 유즈케이스에 참여합니다.
- 온라인 사이트는 "쿠폰 사용", "잔액 확인", "환불" 유즈케이스에 참여합니다.
- 쿠폰 시스템은 모든 유즈케이스에 포함되며, 쿠폰의 상태를 업데이트하고 기록합니다.

2. 시퀀스 다이어그램

1) 쿠폰 사용 - 교환권

- **고객:** 교환권을 매장 직원/온라인 사이트에 제시합니다.
- **매장 직원/온라인 사이트:** 쿠폰 시스템에 교환권 사용 요청을 보냅니다.
- **쿠폰 시스템:** 쿠폰이 유효한지 확인하고, 사용 상태로 변경한 후, 사용 가능 여부를 매장 직원/온라인 사이트에 전달합니다.
- **매장 직원/온라인 사이트:** 고객에게 교환이 완료되었음을 알리고 상품을 제공합니다.

2) 쿠폰 사용 - 잔액관리형

- **고객:** 잔액관리형 쿠폰을 매장 직원/온라인 사이트에 제시하고, 사용 금액을 지정합니다.
- **매장 직원/온라인 사이트:** 쿠폰 시스템에 사용 요청과 함께 금액을 전달합니다.
- **쿠폰 시스템:** 잔액을 차감하고, 쿠폰 상태를 업데이트한 후, 사용 가능 여부를 매장 직원/온라인 사이트에 전달합니다.
- **매장 직원/온라인 사이트:** 고객에게 사용 완료되었음을 알리고 상품/서비스를 제공합니다.

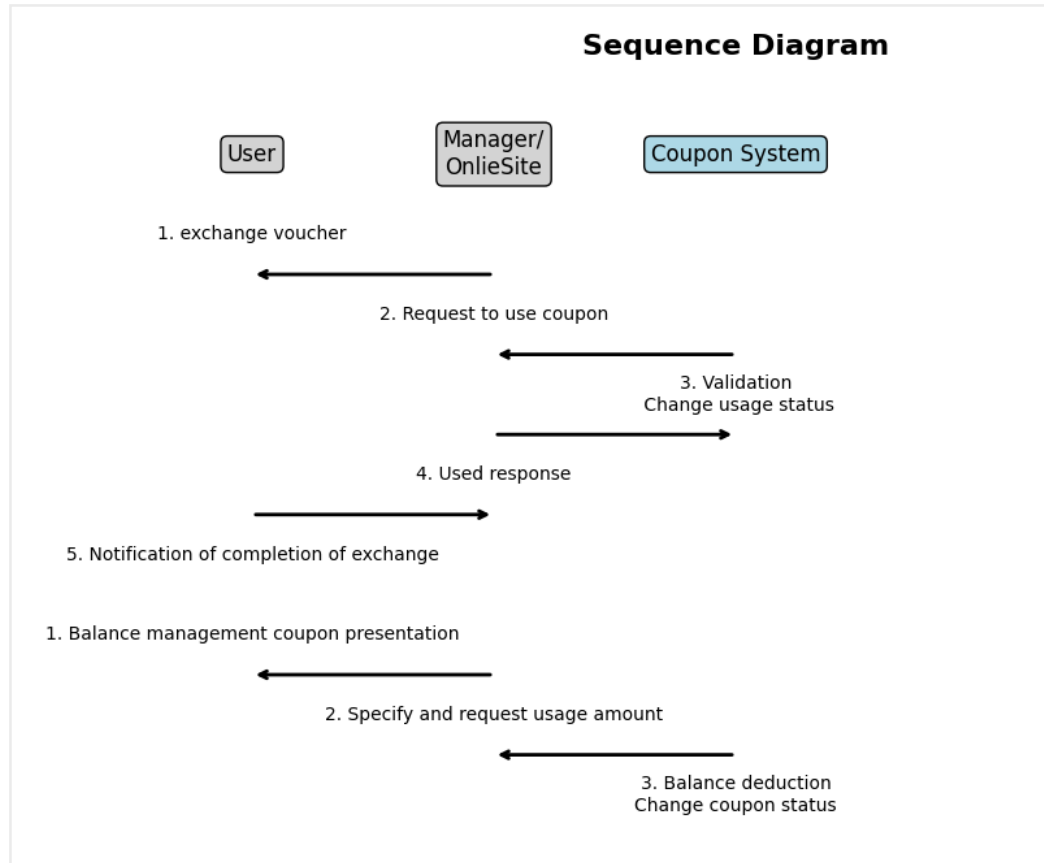
3) 쿠폰 환불 - 교환권

- **고객:** 교환권의 환불을 요청합니다.

- **매장 직원/온라인 사이트:** 쿠폰 시스템에 환불 요청을 보냅니다.
- **쿠폰 시스템:** 환불 가능 여부를 확인하고, 환불을 처리합니다.
- **매장 직원/온라인 사이트:** 고객에게 환불이 완료되었음을 알립니다.

4) 잔액 확인

- **고객:** 잔액 확인 요청을 매장 직원/온라인 사이트에 요청합니다.
- **매장 직원/온라인 사이트:** 쿠폰 시스템에 잔액 조회 요청을 보냅니다.
- **쿠폰 시스템:** 잔액을 확인하여 매장 직원/온라인 사이트에 전달합니다.
- **매장 직원/온라인 사이트:** 고객에게 잔액 정보를 제공합니다.



고객 매장직원 쿠폰시스템

1. 교환권제시
2. 쿠폰사용요청
3. 유효성 확인 및 사용 상태 변경
4. 사용 완료 응답
5. 교환 완료 알림

1. 잔액 관리형 쿠폰 제시
2. 사용 금액 지정 및 요청
3. 잔액 차감 및 쿠폰 상태 변경

문1-2) 인증구간 내 필요한 보안 정책을 HW, SW관점에서 각각 도출하여 목록화 하고 해결방안을 제시하시오.

1. 하드웨어(HW) 관점

1.1. 보안정책: 보안 칩 및 TPM (Trusted Platform Module) 사용

- **문제점:** 모바일 기기에서의 데이터 위변조 위험
- **해결방안:** 모바일 기기나 POS 시스템에 보안 칩 또는 TPM을 내장하여, 민감한 정보(예: 쿠폰 인증 키, 사용자 인증 정보)를 안전하게 저장하고 관리합니다. TPM은 기기 인증과 데이터 암호화를 하드웨어 수준에서 처리하여 보안성을 높입니다.

1.2. 보안정책: 안티탐퍼링(Anti-tampering) 기술 적용

- **문제점:** 하드웨어 해킹이나 무단 접근 시도가 있을 수 있음.
- **해결방안:** 안티탐퍼링 기술을 적용하여 하드웨어에 물리적인 변조 시도를 감지하고, 이를 방지하는 메커니즘을 도입합니다. 변조가 감지되면 시스템이 자동으로 데이터를 파기하거나, 접근을 차단하도록 설정합니다.

1.3. 보안정책: 지리적 위치 기반 접근 제어

- **문제점:** 허가되지 않은 지역에서의 쿠폰 사용 시도가 있을 수 있음.
- **해결방안:** GPS와 같은 위치 기반 서비스를 활용하여 쿠폰 사용이 지정된 지역에서만 가능하도록 하드웨어 단에서 접근 제어를 구현합니다. 이를 통해 특정 지역 외에서의 불법적인 쿠폰 사용을 방지할 수 있습니다.

2. 소프트웨어(SW) 관점

2.1. 보안정책: 강력한 인증 및 인가 시스템

- **문제점:** 불법적인 접근 시도나 인증 과정에서의 위변조 위험
- **해결방안:** 다중 인증(Multi-factor Authentication) 및 역할 기반 접근 제어(Role-Based Access Control)를 도입하여 인증 및 인가 과정을 강화합니다. 특히 사용자 인증 시 암호화된 토큰을 사용하고, 세션 관리 시 일정 시간이 지나면 자동으로 로그아웃되도록 설계합니다.

2.2. 보안정책: 데이터 암호화

- **문제점:** 전송 중 데이터 탈취나 위변조의 위험
- **해결방안:** SSL/TLS 프로토콜을 이용해 모든 통신을 암호화합니다. 또한, 쿠폰 코드나 인증 정보는 AES-256과 같은 강력한 암호화 알고리즘을 사용하여 저장합니다. 이로 인해 중간자 공격이나 데이터 유출에 대비할 수 있습니다.

2.3. 보안정책: 애플리케이션 보안 취약점 방지

- **문제점:** 애플리케이션에서 발생할 수 있는 보안 취약점 (예: SQL 인젝션, XSS, CSRF)
- **해결방안:** OWASP(오픈 웹 애플리케이션 보안 프로젝트) 가이드라인을 따르며, 정기적인 보안 테스트(침투 테스트, 코드 리뷰)를 수행합니다. 입력 데이터의 유효성 검사를 철저히 하여 악성 데이터가 시스템에 들어오지 않도록 방어하고, 최신 보안 패치를 즉시 적용합니다.

2.4. 보안정책: 로그 및 모니터링 시스템 구축

- **문제점:** 보안 사고 발생 시 원인 추적과 대응이 어려울 수 있음.
- **해결방안:** 쿠폰 사용과 인증 과정에서의 모든 이벤트를 기록하는 로그 시스템을 구축합니다. 또한, 실시간 모니터링을 통해 비정상적인 활동을 감지하고, 보안 경고를 발생시켜 신속하게 대응할 수 있도록 합니다. 로그는 암호화하여 안전하게 저장하며, 필요시 감사 추적에 활용할 수 있습니다.

이러한 보안 정책을 통해 하드웨어와 소프트웨어 관점에서 모바일 쿠폰 시스템의 인증 구간에 대한 보안성을 강화할 수 있습니다. 이로써 인증 과정에서 발생할 수 있는 다양한 보안 위협을 효과적으로 방지하고 관리할 수 있습니다.

문1-3) 오프라인 매장에는 서로 다른 종류의 POS기기를 혼용하여 사용중이다. 업체에 제공할 API목록을 도출하고 규격을 정의하시오.

1. API 목록

1. 쿠폰 조회 API (CheckCoupon)

- **기능:** 쿠폰의 유효성 및 잔액을 확인합니다.
- **설명:** 고객이 제시한 쿠폰의 상태(사용 가능 여부, 잔액 등)를 조회하는 기능입니다.
- **메소드:** POST
- **엔드포인트:** /api/v1/coupon/check
- **요청 파라미터:**
 - ◆ coupon_code (string, required): 쿠폰 코드
 - ◆ store_id (string, required): 매장 ID
- **응답 파라미터:**
 - ◆ status (string): 쿠폰 상태 (유효, 무효, 사용 완료 등)
 - ◆ balance (decimal): 잔액관리형 쿠폰의 잔액 (해당되는 경우)
 - ◆ expiry_date (string): 쿠폰 만료일
 - ◆ message (string): 추가적인 정보 또는 오류 메시지

2. 쿠폰 사용 API (RedeemCoupon)

- **기능:** 쿠폰을 사용하여 결제를 처리합니다.
- **설명:** 고객이 쿠폰을 사용해 결제할 때 호출되는 API로, 쿠폰의 금액 또는 교환권 사용을 처리합니다.
- **메소드:** POST
- **엔드포인트:** /api/v1/coupon/redeem
- **요청 파라미터:**
 - ◆ coupon_code (string, required): 쿠폰 코드
 - ◆ store_id (string, required): 매장 ID
 - ◆ amount (decimal, required): 사용하려는 금액 (잔액관리형 쿠폰인 경우)
 - ◆ product_id (string, optional): 교환권 사용 시 교환할 상품 ID
 - ◆ transaction_id (string, required): 거래 ID (POS 시스템에서 생성)
- **응답 파라미터:**
 - ◆ status (string): 처리 결과 (성공, 실패 등)
 - ◆ remaining_balance (decimal): 잔액관리형 쿠폰의 남은 잔액
 - ◆ message (string): 추가적인 정보 또는 오류 메시지

3. 쿠폰 환불 API (RefundCoupon)

- **기능:** 쿠폰 사용을 취소하고 환불 처리합니다.
- **설명:** 사용된 쿠폰을 환불하거나 결제 취소를 처리할 때 호출됩니다.
- **메소드:** POST
- **엔드포인트:** /api/v1/coupon/refund
- **요청 파라미터:**
 - ◆ coupon_code (string, required): 쿠폰 코드
 - ◆ store_id (string, required): 매장 ID
 - ◆ transaction_id (string, required): 원 거래 ID (환불할 거래의 ID)

- 응답 파라미터:
 - ◆ status (string): 환불 결과 (성공, 실패 등)
 - ◆ message (string): 추가적인 정보 또는 오류 메시지

4. 잔액 확인 API (CheckBalance)

- 기능: 잔액관리형 쿠폰의 잔액을 확인합니다.
- 설명: 고객이 사용하려는 잔액관리형 쿠폰의 잔액을 확인합니다.
- 메소드: POST
- 엔드포인트: /api/v1/coupon/balance
- 요청 파라미터:
 - ◆ coupon_code (string, required): 쿠폰 코드
 - ◆ store_id (string, required): 매장 ID
- 응답 파라미터:
 - ◆ balance (decimal): 잔액
 - ◆ expiry_date (string): 쿠폰 만료일
 - ◆ message (string): 추가적인 정보 또는 오류 메시지

5. 거래 내역 조회 API (GetTransactionHistory)

- 기능: 특정 쿠폰의 거래 내역을 조회합니다.
- 설명: 쿠폰이 사용된 거래 내역을 조회하여 사용 기록을 확인합니다.
- 메소드: POST
- 엔드포인트: /api/v1/coupon/transactions
- 요청 파라미터:
 - ◆ coupon_code (string, required): 쿠폰 코드
 - ◆ store_id (string, required): 매장 ID
 - ◆ start_date (string, optional): 조회 시작 날짜 (필요 시)
 - ◆ end_date (string, optional): 조회 종료 날짜 (필요 시)
- 응답 파라미터:
 - ◆ transactions (array): 거래 내역 리스트 (거래 ID, 날짜, 금액 등 포함)
 - ◆ message (string): 추가적인 정보 또는 오류 메시지

2. API 규격 정의

- 통신 프로토콜: HTTPS를 사용하여 모든 데이터를 암호화하여 전송합니다.
- 인증 방식: 각 POS 기기는 OAuth 2.0 또는 API 키 방식으로 인증을 수행합니다. 요청 시 헤더에 Authorization 토큰을 포함시켜 보안을 강화합니다.
- 데이터 포맷: 모든 요청 및 응답은 JSON 형식으로 통일합니다.
- 에러 처리: 각 API는 HTTP 상태 코드(예: 200, 400, 401, 404, 500 등)와 함께 status 및 message 필드를 포함한 응답을 반환하여 에러 상황을 명확하게 전달합니다.
- 타임아웃: API 요청에 대한 응답 타임아웃은 5초로 설정하며, 타임아웃 발생 시 적절한 에러 메시지를 반환합니다.
- 버전 관리: API는 버전 관리를 통해 호환성을 유지하며, /api/v1/ 형태로 엔드포인트를 구성하여 버전 변경 시에도 기존 POS 기기와의 호환성을 보장합니다.

이 API 목록과 규격은 다양한 POS 기기에서 일관된 방식으로 모바일 쿠폰을 처리할 수 있도록 설계되었습니다. 이를 통해 매장 내 시스템 통합의 복잡성을 줄이고, 안정적인 쿠폰 사용 환경을 제공할 수 있습니다.

[문2] 파일기반 대용량 배치성 메시지발송 서비스의 ERD이다.

<문항 지정 전제조건>

- 파일은 FTP를 통해 매일 0시 인입되며, 배치모듈이 파일을 읽어 insert하며, 그룹의 구분 기준은 event_code이다.
- 사용자가 그룹 단위로 발송처리를 confirm 하면 배치모듈이 일괄 메시지발송 작업을 수행한다.

문2-1) 사용자가 발송처리 confirm시 예약발송을 할 수 있도록 ERD에 관련 설계를 반영하시오.

ERD 수정 사항:

1. SendRequest 테이블에 예약 발송 시간 컬럼 추가:

- 예약 발송 시간을 저장하기 위해 scheduled_time 컬럼을 추가합니다.
- 데이터 타입은 datetime 또는 timestamp로 설정합니다.
- 발송이 즉시 이루어질 경우, 이 컬럼은 NULL로 설정합니다.

2. SendRequest 테이블에 예약 여부 컬럼 추가:

- 예약 발송 여부를 확인하기 위해 is_scheduled 컬럼을 추가합니다.
- 데이터 타입은 char(1)으로 설정하고, 'Y' 또는 'N' 값을 가집니다.

3. SendHistory 테이블에 예약 발송 실행 시간 컬럼 추가:

- 예약 발송이 실제로 실행된 시간을 기록하기 위해 executed_time 컬럼을 추가합니다.
- 데이터 타입은 datetime 또는 timestamp로 설정합니다.

수정된 ERD 설계

1. SendRequest 테이블:

- scheduled_time datetime NULL (예약 발송 시간)
- is_scheduled char(1) NOT NULL (예약 여부)

2. SendHistory 테이블:

- executed_time datetime NULL (예약 발송이 실행된 시간)

최종 ERD 예시 (텍스트 설명):

- **SendRequestGroup**: 변경 없음
- **Product**: 변경 없음
- **SendRequest**: 기존 필드 외에 scheduled_time (datetime), is_scheduled (char(1)) 필드 추가
- **SendHistory**: 기존 필드 외에 executed_time (datetime) 필드 추가

이제 사용자가 발송 처리 시 예약 발송을 설정하면, 시스템은 해당 발송 요청을 예약된 시간에 처리하고, SendHistory 테이블에 해당 발송의 실제 실행 시간을 기록하게 됩니다. 이로써 사용자가 원하는 시간에 메시지를 발송할 수 있도록 기능을 확장할 수 있습니다.

문2-2) 처리가 실패한 경우 재처리 기능을 추가하고자 한다. 관련 프로세스와 정책을 정리하고 설계를 반영하시오

1. 재처리 기능을 위한 정책

1. 재처리 횟수 제한:

- 재처리 시도 횟수를 제한하여 무한 루프에 빠지지 않도록 합니다.
- 기본적으로 3회로 설정하되, 필요에 따라 조정할 수 있습니다.

2. 재처리 대기 시간:

- 실패한 후 재처리를 시도하기 전 일정한 대기 시간을 설정합니다.
- 예를 들어, 5분에서 10분 정도의 대기 시간을 둘 수 있습니다.

3. 재처리 로그 기록:

- 재처리 시도를 로그에 기록하여, 문제가 발생했을 때 원인 분석에 활용할 수 있도록 합니다.
- 로그에는 시도 횟수, 재처리 결과, 오류 메시지 등이 포함되어야 합니다.

4. 최종 실패 처리:

- 재처리 횟수를 초과하면, 해당 요청을 '재처리 불가' 상태로 변경하고, 별도의 에러 큐에 추가하여 관리자에게 알림을 보냅니다.

2. 프로세스 흐름

1. 처리 실패 감지:

- 메시지 발송 작업이 실패할 경우, 해당 실패를 기록하고 상태를 '실패'로 설정합니다.

2. 재처리 큐에 추가:

- 실패한 작업을 재처리 큐에 추가하고, 대기 시간을 설정합니다.

3. 재처리 시도:

- 대기 시간이 지난 후 재처리 작업을 시도합니다. 성공 시에는 정상적으로 발송 처리 완료 상태로 변경합니다.
- 재처리가 실패할 경우, 시도 횟수를 증가시키고 다시 대기 큐에 추가합니다.

4. 최종 실패 처리:

- 재처리 횟수를 초과하면, 해당 작업을 '재처리 불가' 상태로 변경하고 관리자에게 알림을 보냅니다.

3. ERD 수정 사항

1. SendRequest 테이블에 재처리 관련 필드 추가:

- retry_count int NOT NULL DEFAULT 0: 재처리 시도 횟수를 기록합니다.
- retry_status char(2) NOT NULL: 현재 재처리 상태를 기록합니다. ('PD' - 대기 중, 'FL' - 실패, 'SC' - 성공, 'NR' - 재처리 불가)

2. SendHistory 테이블에 재처리 로그 필드 추가:

- retry_count int NOT NULL: 해당 기록이 몇 번째 재처리인지를 기록합니다.
- error_message char(255) NULL: 실패 시 발생한 오류 메시지를 기록합니다.

최종 ERD 예시 (텍스트 설명):

• SendRequest:

- retry_count (int, NOT NULL, DEFAULT 0): 재처리 시도 횟수
- retry_status (char(2), NOT NULL): 재처리 상태 ('PD', 'FL', 'SC', 'NR')

• SendHistory:

- retry_count (int, NOT NULL): 재처리 횟수 기록
- error_message (char(255), NULL): 오류 메시지 기록

이 설계 변경을 통해 재처리 기능이 추가된 프로세스를 지원할 수 있습니다. 실패한 메시지 발송 요청이 적절하게 재처리될 수 있도록 하며, 여러 번의 재시도 후에도 실패할 경우 이를 별도로 처리하여 문제를 해결할 수 있도록 합니다.

문2-3) 위 서비스에 TCP 기반의 실시간 단건 발송 서비스를 병합하고자 할 때 ERD에 관련 설계를 반영하시오

1. 실시간 단건 발송 서비스를 위한 설계 고려 사항

1. 발송 요청의 구분:

- 배치성 발송 요청과 실시간 단건 발송 요청을 구분할 수 있어야 합니다.

2. 발송 처리 상태 관리:

- 실시간 발송 요청에 대해서도 발송 상태를 관리하고 추적할 수 있어야 합니다.

3. 실시간 발송 요청 기록:

- 실시간 발송 요청에 대한 이력을 남기고, 실패 시에도 재처리가 가능하도록 해야 합니다.

2. ERD 수정 사항

1. SendRequest 테이블 수정:

- request_type 컬럼 추가: 배치성 요청인지 실시간 요청인지 구분합니다.
 - ◆ 데이터 타입: char(1)
 - ◆ 값: 'B' (배치성), 'R' (실시간 단건)
- tcp_connection_id 컬럼 추가: 실시간 요청 시 연결된 TCP 세션 ID를 저장합니다.
 - ◆ 데이터 타입: varchar(100)
 - ◆ 이 필드는 실시간 요청에만 사용되며, 배치성 요청의 경우 NULL일 수 있습니다.

2. SendHistory 테이블 수정:

- request_type 컬럼 추가: 어떤 종류의 요청에 대한 이력인지 구분합니다.
 - ◆ 데이터 타입: char(1)
 - ◆ 값: 'B' (배치성), 'R' (실시간 단건)
- 이 추가를 통해 배치성 발송과 실시간 단건 발송 이력을 명확히 구분할 수 있습니다.

3. TCP 세션 관리 테이블 추가 (TCPConnection):

- tcp_connection_id varchar(100) PRIMARY KEY: TCP 세션의 고유 ID를 저장합니다.
- start_time datetime: TCP 연결이 시작된 시간을 기록합니다.
- end_time datetime: TCP 연결이 종료된 시간을 기록합니다.
- client_ip varchar(45): 요청을 보낸 클라이언트의 IP 주소를 저장합니다.

최종 ERD 예시 (텍스트 설명):

● SendRequest:

- request_type (char(1), NOT NULL): 요청의 유형 ('B' - 배치성, 'R' - 실시간 단건)
- tcp_connection_id (varchar(100), NULL): 실시간 요청의 경우 연결된 TCP 세션 ID

● SendHistory:

- request_type (char(1), NOT NULL): 요청의 유형 ('B' - 배치성, 'R' - 실시간 단건)

● TCPConnection:

- tcp_connection_id (varchar(100), PRIMARY KEY): TCP 세션 ID
- start_time (datetime, NOT NULL): TCP 연결 시작 시간
- end_time (datetime, NULL): TCP 연결 종료 시간
- client_ip (varchar(45), NOT NULL): 클라이언트 IP 주소

이 설계를 통해 기존 배치성 메시지 발송 시스템에 TCP 기반의 실시간 단건 발송 서비스를 통합할 수 있습니다. 실시간 요청은 별도로 구분하여 관리하며, 실패 시에도 배치성 요청과

동일하게 재처리할 수 있도록 설계되었습니다. 새로운 TCPConnection 테이블은 실시간 요청과 관련된 TCP 세션 정보를 관리하는 데 사용됩니다.

[문3] 라이브방송에 대한 OTT서비스를 제공중인 A서비스가 있다.

<전제 조건>

- A서비스는 중계 역할을 제공하는 서비스 이다
- 사용자는 모든 채널에 자유롭게 접근이 가능 하다
- 채널은 80개 이상이며, 공중파, 예능, 스포츠, 영화등의 분류가 존재
- 각 채널별 라이브/스캐줄 조회 URL 이 방송사에서 제공 중이다
- 클라이언트는 각 채널별 URL을 이용하여 정보를 획득 하고 있다
- (재생도 채널별 URL로 진행 하나요??)
- 방송 스케줄 URL의 경우, 프로그램명/회차명/방송 연령등급/저작권에 따른 방송 가능 여부 정보를 제공한다
- 각 방송사는 30분 마다 내부적으로 편성정보를 갱신 하며, 3일치를 제공 한다
- 클라이언트는 상기 내용을 판단하여 재생 여부를 판단 한다

문 3-1) 사용자 증가에 따라 채널별 성능 저하가 발생하고 있다, 해결 방안을 설계 하시오

1. 캐싱 전략 적용

1.1. 스케줄 정보 캐싱

- **문제:** 방송사의 스케줄 정보가 30분마다 갱신되므로, 클라이언트가 빈번하게 URL을 호출하면서 서버 부하가 증가할 수 있음.
- **해결방안:** 각 채널의 스케줄 정보를 캐싱하여, 일정 기간(예: 30분 동안) 동일한 데이터를 제공하도록 합니다. 이를 통해 서버의 부하를 줄일 수 있습니다.
- **구현 방법:**
 - Redis와 같은 인메모리 데이터베이스를 사용하여 스케줄 정보를 캐싱.
 - 30분마다 캐시를 갱신하는 스케줄러를 설정.

1.2. CDN(Content Delivery Network) 사용

- **문제:** 사용자들이 동일한 방송을 시청할 때 원본 서버에 과도한 부하가 발생.
- **해결방안:** CDN을 도입하여 각 지역에 분산된 서버로 콘텐츠를 전달하고, 클라이언트는 가장 가까운 CDN 서버로부터 콘텐츠를 스트리밍 받을 수 있도록 합니다.
- **구현 방법:**
 - 방송 채널의 URL 스트리밍을 CDN에 분산 배포.
 - CDN이 각 채널의 스트리밍 데이터를 캐싱하여 지역적으로 서비스 제공.

2. 로드 밸런싱

2.1. 로드 밸런서 도입

- **문제:** 특정 서버에 요청이 집중될 경우 성능 저하가 발생.
- **해결방안:** 로드 밸런서를 도입하여 트래픽을 여러 서버에 고르게 분산시킵니다. 이를 통해 특정 서버에 부하가 집중되는 문제를 해결할 수 있습니다.
- **구현 방법:**
 - 클라우드 기반의 로드 밸런서(AWS ELB, GCP Load Balancing 등)를 사용하여 트래픽 분산.
 - 서버 인스턴스를 수평적으로 확장(Scale-Out)하여 증가하는 트래픽을 처리.

3. 서버 및 네트워크 아키텍처 개선

3.1. 서버 오토스케일링

- **문제:** 특정 시간대에 트래픽이 급증할 때 서버 자원이 부족하여 성능 저하가 발생.
- **해결방안:** 오토스케일링을 도입하여 사용자가 증가하면 서버 인스턴스를 자동으로 추가하고, 트래픽이 줄어들면 인스턴스를 축소하여 자원을 효율적으로 관리합니다.
- **구현 방법:**
 - 클라우드 환경에서 오토스케일링 규칙을 설정하여 서버 인스턴스의 수를 동적으로 조정.

3.2. 마이크로서비스 아키텍처 도입

- **문제:** 단일 애플리케이션 구조에서 특정 기능이 병목현상을 일으켜 전체 서비스 성능에 영향을 미침.
- **해결방안:** 마이크로서비스 아키텍처를 도입하여 각 기능을 독립적인 서비스로 분리하고, 필요에 따라 개별적으로 확장 가능하게 합니다.
- **구현 방법:**
 - 방송 스케줄 관리, 사용자 접근 관리, 스트리밍 서비스 등을 독립적인 서비스로 분리.
 - 각 서비스가 개별적으로 확장될 수 있도록 설계.

4. 데이터베이스 최적화

4.1. 데이터베이스 쿼리 최적화

- **문제:** 클라이언트가 데이터를 조회할 때 복잡한 쿼리가 빈번하게 실행되어 DB 부하가 증가.
- **해결방안:** 쿼리 최적화 및 인덱스 튜닝을 통해 데이터베이스 조회 성능을 개선합니다.
- **구현 방법:**
 - 빈번하게 사용되는 쿼리에 인덱스를 추가하여 조회 속도를 개선.
 - 데이터베이스 샤딩 또는 파티셔닝을 도입하여 데이터 접근 성능을 향상.

4.2. 읽기 전용 복제본(Read Replica) 사용

- **문제:** DB에 대한 읽기 요청이 많아질 경우 성능 저하 발생.
- **해결방안:** 읽기 전용 복제본을 생성하여 읽기 요청을 분산시킵니다.
- **구현 방법:**
 - DB 복제본을 생성하고, 로드 밸런서를 통해 읽기 요청을 분산.

5. 사용자 세그먼트 및 QoS (Quality of Service) 적용

5.1. 사용자 세그먼트 기반 서비스 제공

- **문제:** 모든 사용자에게 동일한 품질의 서비스를 제공하려다 보니 자원 낭비와 성능 저하 발생.
- **해결방안:** 사용자를 세그먼트로 나누어 우선순위를 설정하고, 중요도에 따라 자원을 할당합니다.
- **구현 방법:**
 - 프리미엄 사용자와 일반 사용자를 구분하여, 프리미엄 사용자에게 더 많은 자원을 할당.

5.2. QoS 적용

- **문제:** 모든 사용자에게 균등하게 자원을 제공할 경우, 일부 사용자의 과도한 사용으로 인해 다른 사용자에게 영향이 갈 수 있음.
- **해결방안:** QoS를 적용하여 사용자의 네트워크 대역폭을 관리하고, 특정 사용자나 서비스에 우선순위를 부여합니다.
- **구현 방법:**
 - 네트워크 레벨에서 QoS 설정을 통해 트래픽을 관리.

6. 성능 모니터링 및 자동화 도구 도입

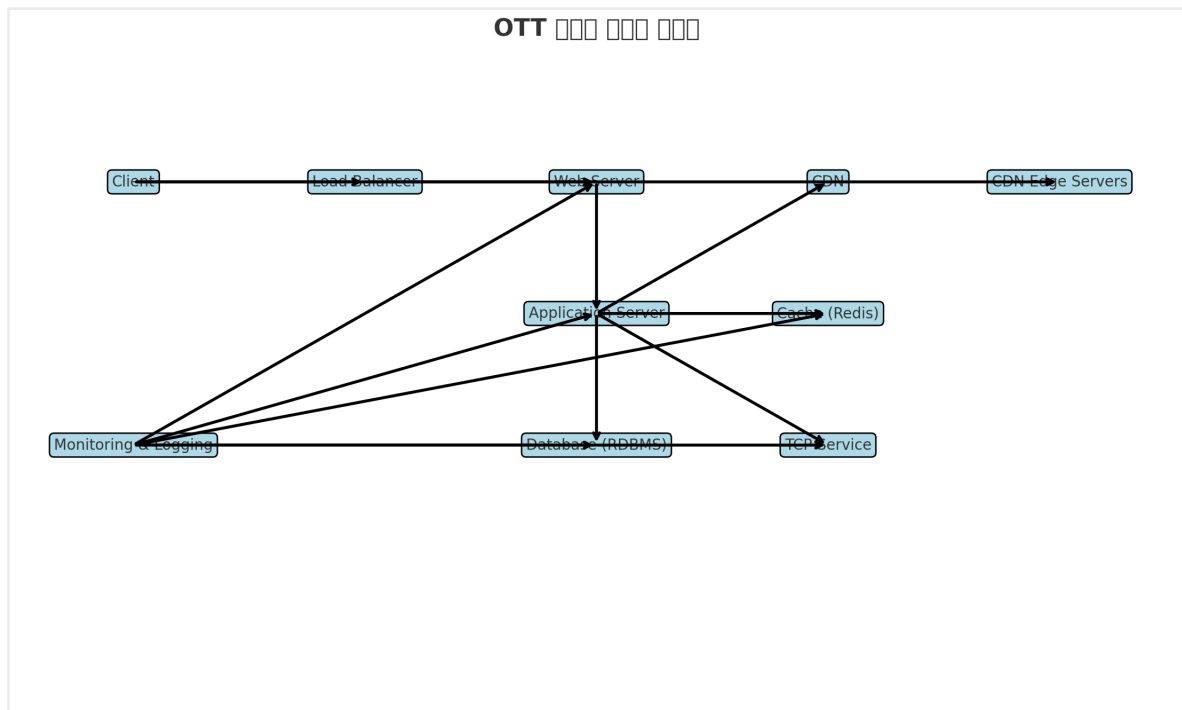
6.1. 성능 모니터링 도구 사용

- **문제:** 문제 발생 시 원인을 추적하기 어려움.
- **해결방안:** 성능 모니터링 도구를 사용하여 실시간으로 시스템 성능을 모니터링하고, 문제가 발생하면 즉시 대응할 수 있도록 합니다.
- **구현 방법:**
 - Prometheus, Grafana, New Relic 등의 모니터링 도구를 사용하여 실시간 모니터링.
 - 알림 설정을 통해 이슈 발생 시 신속히 대응.

6.2. 자동화 스크립트 및 배포 도구 사용

- **문제:** 새로운 서버를 추가하거나 설정 변경 시 수작업으로 인해 시간이 소요되고 실수가 발생할 수 있음.
- **해결방안:** 자동화 스크립트 및 배포 도구를 사용하여 서버 설정, 배포, 스케일링을 자동으로 처리합니다.
- **구현 방법:**
 - Ansible, Terraform 등을 사용하여 인프라 설정을 자동화.
 - CI/CD 파이프라인을 구축하여 코드 배포를 자동화.

위와 같은 해결 방안을 통해 사용자 증가로 인한 채널별 성능 저하 문제를 해결할 수 있습니다. 각 방안은 시스템의 다양한 측면에서 성능을 최적화하고, 확장성과 안정성을 개선하는 데 중점을 두고 있습니다.

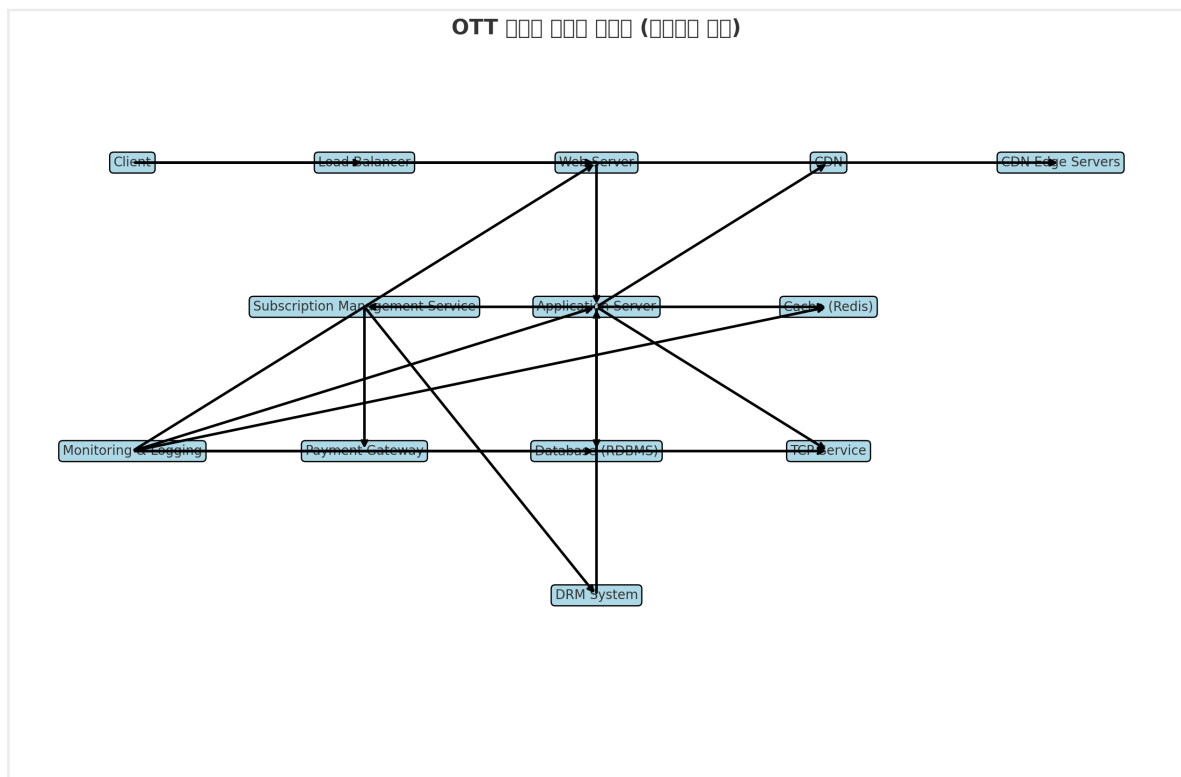


위 이미지는 OTT 서비스 시스템의 구성도를 나타낸 것입니다. 각 구성 요소와 그들 간의 연결을 시각적으로 표현하였으며, 주요 구성 요소들은 다음과 같습니다:

- **Client:** 사용자가 OTT 서비스를 이용하는 클라이언트 디바이스.
- **Load Balancer:** 트래픽을 여러 서버로 분산시키는 역할을 하는 로드 밸런서.
- **Web Server:** 클라이언트의 요청을 처리하는 웹 서버.
- **CDN (Content Delivery Network):** 전 세계에 분산된 CDN 서버를 통해 콘텐츠를 사용자에게 빠르게 전달.
- **Application Server:** 비즈니스 로직을 처리하는 애플리케이션 서버.

- **Cache (Redis):** 자주 조회되는 데이터를 캐싱하여 데이터베이스 부하를 줄이는 캐시 서버.
- **Database (RDBMS):** 사용자의 데이터와 방송 스케줄 정보를 저장하는 관계형 데이터베이스.
- **TCP Service:** 실시간 단건 발송 서비스를 위한 TCP 기반의 서비스.
- **Monitoring & Logging:** 시스템의 성능을 모니터링하고 로그를 관리하는 모니터링 및 로깅 시스템.
- **CDN Edge Servers:** 사용자에게 가장 가까운 위치에서 콘텐츠를 제공하는 CDN 엣지 서버.

문 3-2) 유료채널을 추가 하기 위한 방안에 대하여 설계 하시오



위 이미지는 유료 채널 기능을 포함한 OTT 서비스 시스템의 구성도입니다. 기존의 무료 채널 서비스에 추가된 유료 채널 기능을 지원하기 위해 **Subscription Management Service**, **Payment Gateway**, **DRM System**과 같은 새로운 컴포넌트가 추가되었습니다.

1. 요구 사항 정의

1. 사용자 구독 모델:

- 유료 채널에 접근하기 위해 사용자는 특정 채널 또는 채널 패키지에 대해 구독을 해야 합니다.
- 다양한 구독 옵션(월간, 연간, 채널별, 패키지별)을 제공할 수 있습니다.

2. 유료 채널 접근 제어:

- 유료 채널은 구독한 사용자만 접근할 수 있도록 접근 제어가 필요합니다.
- 구독 상태를 확인하여 접근 권한을 부여해야 합니다.

3. 결제 및 과금 시스템 통합:

- 구독을 위한 결제 시스템과의 통합이 필요합니다. 결제 후 구독 상태를 업데이트하고,

결제 이력도 관리해야 합니다.

4. 유료 채널 콘텐츠 보호:

- 유료 콘텐츠의 불법 공유나 접근을 막기 위해 DRM(Digital Rights Management) 시스템을 도입할 수 있습니다.
- 스트리밍 과정에서 암호화된 콘텐츠를 제공하고, 허가된 사용자만 복호화할 수 있도록 합니다.

2. 시스템 설계

2.1. 데이터베이스 설계

1. UserSubscription 테이블:

- **user_id** (foreign key): 사용자의 ID
- **channel_id** (foreign key): 구독한 유료 채널의 ID
- **subscription_start_date** (datetime): 구독 시작일
- **subscription_end_date** (datetime): 구독 만료일
- **payment_status** (varchar): 결제 상태 (결제 완료, 결제 실패 등)
- **auto_renewal** (boolean): 자동 갱신 여부

2. PaymentHistory 테이블:

- **payment_id** (primary key): 결제 ID
- **user_id** (foreign key): 사용자의 ID
- **channel_id** (foreign key): 구독한 유료 채널의 ID
- **amount** (decimal): 결제 금액
- **payment_date** (datetime): 결제 날짜
- **payment_status** (varchar): 결제 상태

3. Channels 테이블 (기존):

- **is_paid** (boolean): 유료 채널 여부 추가

2.2. 시스템 컴포넌트 설계

1. Subscription Management Service:

- 기능: 사용자 구독 상태를 관리하며, 구독 시작, 종료, 갱신, 취소 등의 작업을 처리합니다.
- 설계:
 - ◆ 구독 요청이 들어오면 결제 시스템과 연동하여 결제를 처리하고, 결제가 성공하면 유료 채널에 대한 접근 권한을 부여합니다.
 - ◆ 구독 갱신 시 자동으로 결제를 처리하고, 실패 시 사용자에게 알림을 보냅니다.

2. Payment Gateway Integration:

- 기능: 외부 결제 시스템과의 연동을 통해 결제를 처리합니다.
- 설계:
 - ◆ 사용자로부터 결제 정보를 받아 결제를 진행하고, 결제 성공 여부를 Subscription Management Service에 전달합니다.
 - ◆ 결제 이력을 관리하고, 환불이나 결제 실패 처리도 지원합니다.

3. Access Control Middleware:

- 기능: 유료 채널에 접근하려는 사용자에게 대한 접근 권한을 검증합니다.
- 설계:
 - ◆ 사용자가 유료 채널에 접근하려고 할 때, 현재 구독 상태를 확인합니다.
 - ◆ 구독이 유효한 사용자만 스트리밍 서비스에 접근할 수 있도록 제한합니다.
 - ◆ 접근이 허가된 경우 DRM 키를 발급하여 콘텐츠 복호화가 가능하도록 합니다.

4. DRM (Digital Rights Management) System:

- 기능: 유료 채널의 콘텐츠를 보호합니다.
- 설계:
 - ◆ 콘텐츠를 암호화하여 제공하고, 유효한 접근 권한을 가진 사용자에게만 복호화 키를 제공합니다.
 - ◆ 불법적인 접근이나 콘텐츠 복제를 방지합니다.

3. 서비스 흐름

1. 유료 채널 구독:
 - 사용자는 유료 채널 목록을 확인하고, 원하는 채널을 구독합니다.
 - 결제 과정이 완료되면, Subscription Management Service에서 사용자의 구독 상태를 갱신합니다.
2. 채널 접근 시:
 - 사용자가 유료 채널에 접근하면, Access Control Middleware에서 사용자의 구독 상태를 확인합니다.
 - 구독이 유효한 경우에만 채널 스트리밍을 허용하고, DRM 키를 발급합니다.
3. 콘텐츠 스트리밍:
 - 유료 채널의 콘텐츠는 암호화되어 제공되며, DRM 시스템을 통해서만 복호화됩니다.
 - 사용자는 합법적인 경로로만 콘텐츠를 시청할 수 있습니다.
4. 구독 갱신 및 관리:
 - 구독 만료일이 가까워지면 자동으로 결제를 시도하고, 성공 시 구독을 갱신합니다.
 - 결제 실패 시 사용자에게 알림을 보내고, 일정 기간 후 구독이 종료됩니다.

4. 보안 고려 사항

- 사용자 데이터 보호: 구독과 결제 정보는 모두 암호화하여 저장하고, 전송 시에도 TLS(Transport Layer Security)를 사용해 보호합니다.
- DRM 및 콘텐츠 보호: DRM 시스템을 강화하여 유료 콘텐츠의 불법적인 공유를 막고, 허가된 사용자만 콘텐츠를 사용할 수 있도록 합니다.
- 이중 인증: 결제 과정에서 이중 인증(MFA)을 도입하여 보안을 강화합니다.

[문4] 쇼핑몰 아키텍처를 domain driven design 방식으로 설계하시오

1) 주요 aggregate 를 도출해보시오.

1. 주요 Aggregate 도출

1.1. Product (상품)

- 설명: 쇼핑몰에서 판매되는 상품에 대한 정보를 관리하는 애그리게이트입니다.
- 포함되는 엔티티와 값 객체:
 - **Product**: 상품의 기본 정보(이름, 설명, 가격, 카테고리 등)를 포함하는 엔티티.
 - **Category**: 상품이 속한 카테고리 정보.
 - **Price**: 상품의 가격 정보를 캡슐화한 값 객체.
 - **Stock**: 상품의 재고 상태를 나타내는 값 객체.
 - **ProductOption**: 색상, 사이즈 등 상품의 다양한 옵션을 관리하는 엔티티.

1.2. Order (주문)

- 설명: 고객이 상품을 구매하기 위해 생성하는 주문에 대한 정보를 관리하는 애그리게이트입니다.
- 포함되는 엔티티와 값 객체:
 - **Order**: 주문의 기본 정보(주문 번호, 주문 날짜, 총 금액 등)를 포함하는 엔티티.

- **OrderItem**: 주문된 각 상품의 상세 정보(상품, 수량, 개별 가격 등)를 포함하는 엔티티.
- **ShippingAddress**: 배송지 정보를 캡슐화한 값 객체.
- **Payment**: 결제 정보(결제 방식, 결제 상태 등)를 관리하는 엔티티.
- **OrderStatus**: 주문의 상태(주문 접수, 결제 완료, 배송 중, 완료 등)를 나타내는 값 객체.

1.3. Customer (고객)

- **설명**: 쇼핑몰의 고객 정보를 관리하는 애그리거트입니다.
- **포함되는 엔티티와 값 객체**:
 - **Customer**: 고객의 기본 정보(이름, 이메일, 전화번호 등)를 포함하는 엔티티.
 - **Address**: 고객의 여러 배송지 주소를 관리하는 값 객체.
 - **PaymentMethod**: 고객의 결제 수단(카드, 계좌, 포인트 등)을 관리하는 엔티티.
 - **LoyaltyPoints**: 고객의 적립 포인트를 관리하는 값 객체.
 - **CustomerPreferences**: 고객의 선호도(예: 뉴스레터 구독 여부, 선호 카테고리 등)를 관리하는 엔티티.

1.4. Cart (장바구니)

- **설명**: 고객이 구매를 고려 중인 상품들을 임시로 저장하는 장바구니를 관리하는 애그리거트입니다.
- **포함되는 엔티티와 값 객체**:
 - **Cart**: 장바구니의 기본 정보(고객 ID, 생성 날짜 등)를 포함하는 엔티티.
 - **CartItem**: 장바구니에 담긴 개별 상품과 수량을 나타내는 엔티티.
 - **Coupon**: 장바구니에 적용된 할인 쿠폰을 관리하는 엔티티.

1.5. Inventory (재고)

- **설명**: 각 상품의 재고 상태를 관리하고, 주문 시 재고를 차감하거나 복구하는 역할을 담당하는 애그리거트입니다.
- **포함되는 엔티티와 값 객체**:
 - **Inventory**: 상품별 재고를 관리하는 엔티티.
 - **Warehouse**: 재고가 보관되는 창고 정보를 관리하는 엔티티.
 - **StockMovement**: 재고 이동 내역(입고, 출고, 환불로 인한 재고 증가 등)을 관리하는 엔티티.

1.6. Payment (결제)

- **설명**: 결제 과정 및 결제 상태를 관리하는 애그리거트입니다.
- **포함되는 엔티티와 값 객체**:
 - **Payment**: 결제의 기본 정보(결제 금액, 결제 일시 등)를 포함하는 엔티티.
 - **PaymentMethod**: 사용된 결제 수단(신용카드, 계좌이체 등)을 관리하는 엔티티.
 - **Transaction**: 결제 트랜잭션의 상태(성공, 실패 등)를 관리하는 엔티티.

1.7. Delivery (배송)

- **설명**: 상품의 배송 과정을 관리하는 애그리거트입니다.
- **포함되는 엔티티와 값 객체**:
 - **Delivery**: 배송의 기본 정보(배송 번호, 발송 일시, 배송 상태 등)를 포함하는 엔티티.
 - **DeliveryAddress**: 배송지 정보를 관리하는 값 객체.
 - **DeliveryStatus**: 배송 상태(배송 준비 중, 배송 중, 배송 완료 등)를 나타내는 값 객체.

2. Aggregate 간의 관계

- **Order** 애그리거트는 **Customer, Product, Inventory, Payment, Delivery** 애그리거트와 밀접하게 연관됩니다.
- **Cart** 애그리거트는 **Customer** 및 **Product** 애그리거트와 연관됩니다.
- **Inventory** 애그리거트는 **Product** 애그리거트와 직접적으로 연관되어, 상품의 재고 상태를 관리합니다.

2) 서비스 간 통신방식을 설계하시오.

1. 서비스 간 통신 방식 개요

1.1. 동기식 통신 (Synchronous Communication)

- **설명:** 한 서비스가 다른 서비스에 요청을 보내고, 응답을 받을 때까지 기다리는 방식.
- **주요 프로토콜:** HTTP/HTTPS, gRPC, REST, SOAP
- **사용 사례:** 실시간 응답이 필요한 경우, 서비스 간 데이터 일관성이 중요할 때.

1.2. 비동기식 통신 (Asynchronous Communication)

- **설명:** 한 서비스가 요청을 보낸 후, 응답을 기다리지 않고 다른 작업을 계속할 수 있는 방식. 응답은 별도의 메시지로 비동기적으로 수신.
- **주요 프로토콜:** 메시지 큐 (RabbitMQ, Kafka), 이벤트 스트리밍, 웹훅(Webhook)
- **사용 사례:** 이벤트 기반 아키텍처, 작업이 비동기적으로 처리되어도 되는 경우, 트래픽이 높은 상황에서 서비스 간 결합도를 줄여야 할 때.

2. 쇼핑몰 서비스 간 통신 방식 설계

각 애그리거트에 대한 서비스 간 통신 방식을 설계하겠습니다.

2.1. Order 서비스와 Payment 서비스

- **통신 방식:** 동기식 통신 (REST API 또는 gRPC)
- **설명:** 주문이 생성된 후 결제가 바로 이루어져야 하므로 실시간 응답이 필요합니다. Order 서비스는 Payment 서비스에 결제 요청을 보내고, 결제 결과를 받아 주문 상태를 업데이트합니다.
- **사용 프로토콜:** HTTP/HTTPS 또는 gRPC
- **장점:** 결제 결과를 실시간으로 확인할 수 있으며, 주문 상태의 일관성을 유지할 수 있습니다.

2.2. Order 서비스와 Inventory 서비스

- **통신 방식:** 동기식 통신 (REST API 또는 gRPC)
- **설명:** 주문이 확정되면 해당 상품의 재고를 차감해야 하므로 실시간 통신이 필요합니다. Order 서비스는 Inventory 서비스에 재고 차감 요청을 보내고, 재고가 충분한지 확인한 후 주문을 확정합니다.
- **사용 프로토콜:** HTTP/HTTPS 또는 gRPC
- **장점:** 재고 차감의 정확성과 주문의 일관성을 보장합니다.

2.3. Order 서비스와 Delivery 서비스

- **통신 방식:** 비동기식 통신 (메시지 큐)
- **설명:** 주문이 결제된 후 배송 요청이 발생하므로, Delivery 서비스로의 통신은 반드시 실시간일 필요는 없습니다. 주문이 완료된 후 메시지 큐를 통해 Delivery 서비스에 배송 요청을 비동기적으로 전달합니다.
- **사용 프로토콜:** 메시지 큐 (RabbitMQ, Kafka)
- **장점:** 시스템의 유연성을 높이며, 주문이 대량으로 들어올 때도 안정적인 처리 가능.

2.4. Customer 서비스와 Order 서비스

- **통신 방식:** 동기식 통신 (REST API)
- **설명:** 고객의 정보를 기반으로 주문이 생성되므로, Order 서비스는 Customer

서비스에서 고객 정보를 조회해야 합니다. 이때 동기적으로 고객 정보를 가져와야 합니다.

- **사용 프로토콜:** HTTP/HTTPS
- **장점:** 고객 정보의 정확성을 보장하고, 주문 처리의 일관성을 유지할 수 있습니다.

2.5. Inventory 서비스와 Product 서비스

- **통신 방식:** 동기식 통신 (REST API)
- **설명:** 재고 상태를 업데이트하기 위해 Inventory 서비스는 Product 서비스에서 상품 정보를 조회할 수 있습니다.
- **사용 프로토콜:** HTTP/HTTPS
- **장점:** 상품 정보와 재고 관리의 일관성을 유지할 수 있습니다.

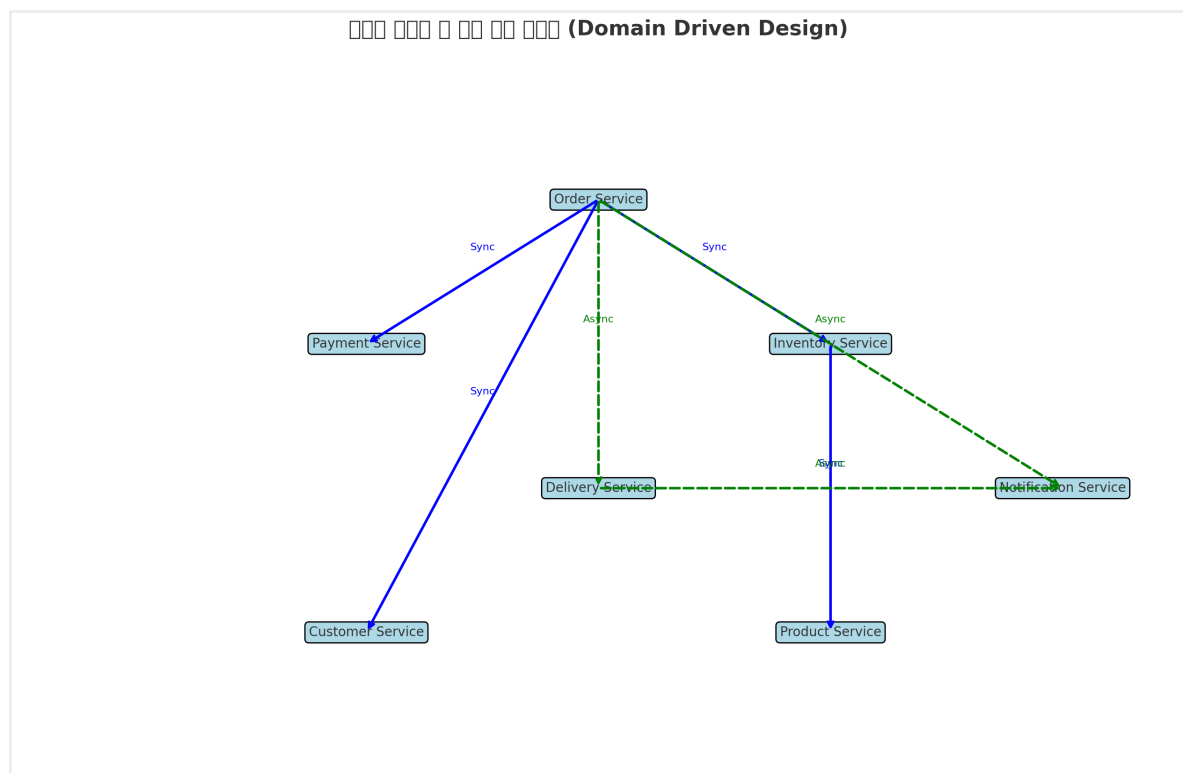
2.6. Notification 서비스

- **통신 방식:** 비동기식 통신 (메시지 큐 또는 이벤트 스트리밍)
- **설명:** 주문 상태 변경, 배송 상태 업데이트, 결제 완료 등의 이벤트가 발생하면 Notification 서비스에서 고객에게 알림을 보냅니다. 이러한 이벤트는 비동기적으로 처리할 수 있습니다.
- **사용 프로토콜:** 메시지 큐 (RabbitMQ, Kafka), 이벤트 스트리밍
- **장점:** 시스템 부하를 줄이고, 이벤트 기반으로 확장 가능한 아키텍처 구축 가능.

3. 통신 방식을 고려한 시스템 구성도

서비스 간 통신 방식을 반영한 시스템 구성도를 다음과 같이 설계할 수 있습니다:

1. **Order 서비스:** Payment 서비스와 Inventory 서비스와 동기식으로 통신, Delivery 서비스와 비동기식으로 통신.
2. **Customer 서비스:** Order 서비스와 동기식으로 통신.
3. **Inventory 서비스:** Product 서비스와 동기식으로 통신.
4. **Delivery 서비스:** 비동기 메시지 큐를 통해 Order 서비스로부터 요청을 수신.
5. **Notification 서비스:** 비동기 메시지 큐 또는 이벤트 스트리밍을 통해 다양한 서비스로부터 이벤트를 수신.



위 이미지는 쇼핑몰 서비스 간 통신 방식을 도식화한 구성도입니다. 도식에서는 동기식 통신

(Synchronous Communication)과 비동기식 통신(Asynchronous Communication)을 각각 파란색과 녹색으로 구분하여 표시했습니다.

- **동기식 통신(Sync):** 파란색 실선으로 표시된 통신 방식은 실시간 응답이 필요한 서비스 간의 연결입니다. 주로 주문(Order Service)과 결제(Payment Service), 재고(Inventory Service) 간의 통신에서 사용됩니다.
- **비동기식 통신(Async):** 녹색 점선으로 표시된 통신 방식은 서비스 간의 독립성을 유지하면서 비동기적으로 이벤트를 처리하는 방식입니다. 예를 들어, 주문이 완료된 후 배송(Delivery Service) 요청이나 알림(Notification Service) 발송 등이 비동기 방식으로 처리됩니다.

3) 성능 개선을 위한 방안을 작성하시오.

1. 데이터베이스 최적화

1.1. 인덱스 최적화

- **설명:** 데이터베이스 쿼리에서 자주 사용되는 필드에 인덱스를 추가하여 검색 성능을 향상시킵니다.
- **예시:** 주문 테이블의 `order_date`, `customer_id`와 같은 필드에 인덱스를 추가하여 조회 속도를 개선합니다.

1.2. 쿼리 최적화

- **설명:** 불필요한 복잡한 조인과 서브쿼리를 줄이고, 쿼리를 단순화하여 처리 속도를 높입니다.
- **예시:** 자주 사용되는 쿼리를 리팩토링하거나, 필요한 경우 뷰(View)를 생성하여 복잡한 쿼리를 단순화합니다.

1.3. 데이터베이스 샤딩

- **설명:** 데이터를 분할하여 여러 데이터베이스에 분산 저장함으로써 데이터베이스의 부하를 분산시킵니다.
- **예시:** 고객 데이터를 여러 서버에 분산하여 저장하고, 고객 ID 범위에 따라 샤딩하여 데이터베이스 접근 성능을 향상시킵니다.

1.4. 캐싱 전략 도입

- **설명:** 자주 조회되는 데이터를 캐시에 저장하여 데이터베이스의 부하를 줄이고, 응답 시간을 단축시킵니다.
- **예시:** Redis나 Memcached를 사용하여, 자주 조회되는 상품 정보나 주문 상태를 캐시에 저장하여 조회 속도를 높입니다.

2. 애플리케이션 레벨 최적화

2.1. 로드 밸런싱

- **설명:** 트래픽을 여러 서버로 분산시켜 특정 서버에 부하가 집중되지 않도록 합니다.
- **예시:** 클라우드 기반의 로드 밸런서를 도입하여 웹 서버와 애플리케이션 서버에 트래픽을 균등하게 분산합니다.

2.2. 애플리케이션 서버 오토스케일링

- **설명:** 트래픽이 증가할 때 자동으로 애플리케이션 서버 인스턴스를 추가하여 처리 용량을 확장하고, 트래픽이 감소할 때 자동으로 인스턴스를 줄입니다.
- **예시:** AWS Auto Scaling을 활용하여 서버 인스턴스를 동적으로 조정합니다.

2.3. 비동기 처리 및 작업 큐 사용

- **설명:** 시간이 많이 걸리는 작업(예: 이메일 발송, 보고서 생성 등)을 비동기적으로 처리하여

응답 시간을 줄이고, 작업 큐를 사용해 안정적으로 작업을 처리합니다.

- **예시:** Celery와 같은 작업 큐를 사용하여 비동기 작업을 관리하고, 백그라운드에서 처리합니다.

2.4. 애플리케이션 코드 최적화

- **설명:** 애플리케이션 코드의 성능을 향상시키기 위해 알고리즘 최적화, 불필요한 연산 제거 등을 수행합니다.
- **예시:** 반복적인 데이터베이스 호출을 줄이고, 복잡한 계산을 미리 처리하여 응답 시간을 단축합니다.

3. 네트워크 최적화

3.1. 콘텐츠 전송 네트워크(CDN) 사용

- **설명:** 정적 파일(이미지, CSS, JavaScript 등)을 CDN을 통해 전 세계에 분산 배포하여 사용자에게 가까운 서버에서 빠르게 콘텐츠를 제공하도록 합니다.
- **예시:** CloudFront와 같은 CDN을 사용하여 이미지와 정적 파일을 사용자에게 빠르게 전달합니다.

3.2. HTTP/2 및 TLS 최적화

- **설명:** HTTP/2를 사용하여 여러 요청을 병렬로 처리하고, TLS 연결의 성능을 최적화하여 안전하면서도 빠른 통신을 보장합니다.
- **예시:** 웹 서버가 HTTP/2를 지원하도록 설정하고, TLS 인증서 및 설정을 최적화하여 보안과 성능을 동시에 향상시킵니다.

4. 모니터링 및 로그 관리

4.1. 실시간 모니터링 도구 사용

- **설명:** 시스템의 성능을 실시간으로 모니터링하여 문제가 발생하기 전에 예측하고 대응할 수 있도록 합니다.
- **예시:** Prometheus와 Grafana를 사용하여 서버 상태, 애플리케이션 성능, 데이터베이스 쿼리 시간을 모니터링합니다.

4.2. 로그 분석 및 알림 설정

- **설명:** 시스템 로그를 분석하여 성능 저하의 원인을 파악하고, 문제 발생 시 알림을 설정하여 신속히 대응할 수 있도록 합니다.
- **예시:** ELK Stack (Elasticsearch, Logstash, Kibana)을 사용하여 로그를 중앙에서 관리하고, 특정 패턴이 감지되면 알림을 받도록 설정합니다.

5. 프론트엔드 최적화

5.1. 브라우저 캐싱

- **설명:** 자주 변하지 않는 리소스에 대해 브라우저 캐싱을 설정하여 재방문 시 로딩 시간을 줄입니다.
- **예시:** 이미지, CSS, JavaScript 파일에 대한 캐시 헤더를 설정하여 브라우저가 이 리소스를 캐싱하도록 합니다.

5.2. 코드 스플리팅과 번들링

- **설명:** JavaScript와 CSS 파일을 필요한 시점에만 로드하도록 코드 스플리팅을 하고, 불필요한 자원을 줄이기 위해 파일 번들링을 수행합니다.
- **예시:** Webpack을 사용하여 코드 스플리팅을 구현하고, Terser와 같은 도구로 번들링된 파일을 압축합니다.

5.3. 이미지 최적화

- **설명:** 이미지 파일을 최적화하여 로딩 시간을 줄이고, 웹사이트의 성능을 향상시킵니다.
- **예시:** 이미지 파일을 WebP 포맷으로 변환하거나, 이미지 크기를 자동으로 조정하여 전송합니다.

[문5] 로그인 방식에 소셜로그인 추가할 경우 변경되는 ERD 작성하시오.

1) 개념적 데이터 모델링 변경사항 작성하시오.

1. 개념적 데이터 모델링 변경 사항

1.1. User 테이블 확장

- 기존 User 테이블:
 - 기존에는 사용자의 이메일, 비밀번호, 이름, 연락처 등의 기본 정보를 관리하고 있었을 것입니다.
- 변경 사항:
 - 소셜 로그인 사용자를 관리하기 위해 추가 필드 및 테이블 확장이 필요합니다.
 - login_type: 사용자가 어떤 방식으로 로그인했는지를 나타내는 필드(예: 'local', 'google', 'facebook' 등).
 - social_id: 소셜 로그인 시 사용되는 고유한 ID(예: Google ID, Facebook ID 등)를 저장할 필드.
- 변경된 User 테이블:
 - user_id (primary key): 고유한 사용자 ID.
 - email: 사용자의 이메일 주소.
 - password: 로컬 로그인 사용자를 위한 비밀번호. 소셜 로그인 사용자의 경우 NULL 가능.
 - login_type: 사용자의 로그인 타입 ('local', 'google', 'facebook', etc.).
 - social_id: 소셜 로그인 사용자에 대한 고유 ID.
 - name: 사용자의 이름.
 - phone: 사용자의 연락처 정보.

1.2. SocialAccount 테이블 추가

- 새로운 테이블:
 - 소셜 계정 정보와 기존의 User 테이블을 분리하여 관리할 수 있습니다.
 - 각 소셜 로그인 제공자(Google, Facebook 등)에 대한 추가 정보(예: 토큰, 프로필 이미지 URL)를 저장할 수 있습니다.
- SocialAccount 테이블:
 - social_account_id (primary key): 고유한 소셜 계정 ID.
 - user_id (foreign key): User 테이블의 사용자 ID와 연결.
 - provider: 소셜 로그인 제공자(예: 'google', 'facebook').
 - social_id: 소셜 로그인 제공자에서 제공하는 사용자 고유 ID.
 - profile_image_url: 소셜 프로필 이미지 URL (선택적).
 - access_token: 소셜 로그인 시 받은 접근 토큰 (선택적).
 - refresh_token: 접근 토큰 갱신을 위한 리프레시 토큰 (선택적).

2. 개념적 데이터 모델링의 변경 사항 요약

2.1. User 테이블:

- 추가 필드: login_type, social_id.
- 변경 사항: 기존 필드에 추가적인 정보가 저장될 수 있도록 수정.

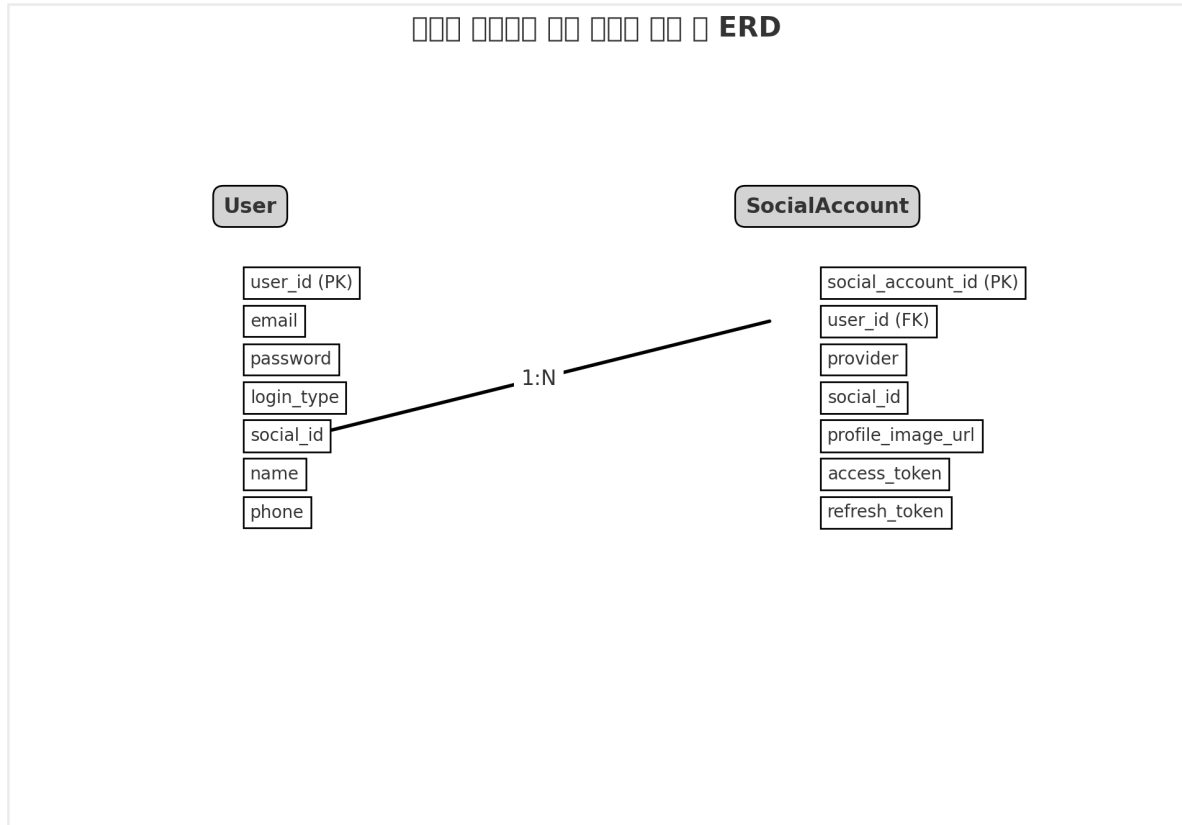
2.2. SocialAccount 테이블 추가:

- 새로운 테이블: 사용자와 소셜 계정 간의 관계를 관리하고, 소셜 로그인 관련 정보를 저장.

3. 개념적 데이터 모델링의 시각화

ERD(개념적 데이터 모델링)에서 각 테이블 간의 관계를 다음과 같이 설정할 수 있습니다:

- User 테이블은 기본 사용자 정보를 저장하고, 로그인 타입 및 소셜 ID와 같은 추가 필드를 통해 소셜 로그인을 지원합니다.
- SocialAccount 테이블은 User 테이블과 1 관계를 가지며, 각 사용자가 여러 소셜 계정을 연결할 수 있도록 설계합니다(필요한 경우).



변경된 ERD 설명:

1. **User** 테이블: 기존 사용자 정보 외에 소셜 로그인 관련 정보(login_type, social_id)가 추가되었습니다.
2. **SocialAccount** 테이블: 소셜 계정 정보를 관리하는 새로운 테이블이 추가되었습니다. 이 테이블은 여러 소셜 계정을 관리할 수 있도록 설계되었으며, User 테이블과 1 관계를 가집니다.

2) 논리적 데이터 모델링 변경사항 작성하시오.

1. User 테이블의 변경 사항

- **login_type** (VARCHAR(50), NOT NULL):
 - 설명: 사용자가 로그인할 때 사용한 로그인 방식(local, google, facebook 등)을 저장합니다.
 - 목적: 사용자가 어떤 방식으로 로그인했는지 추적할 수 있습니다.
 - 제약 조건: 이 필드는 NULL이 될 수 없으며, 지정된 값만 허용됩니다(예: 'local', 'google', 'facebook' 등).
- **social_id** (VARCHAR(255), NULL):
 - 설명: 소셜 로그인 사용자의 고유 ID를 저장합니다. 로컬 로그인 사용자의 경우 이

- 필드는 NULL이 될 수 있습니다.
- **목적:** 각 소셜 플랫폼에서 제공하는 사용자 ID를 저장하여 해당 사용자와 연관된 소셜 계정을 관리합니다.
- **제약 조건:** 이 필드는 소셜 로그인 사용자의 경우에만 값이 존재합니다.

2. SocialAccount 테이블 추가

- **social_account_id** (BIGINT, AUTO_INCREMENT, PRIMARY KEY):
 - **설명:** 소셜 계정 테이블의 고유 식별자입니다.
 - **목적:** 소셜 계정 정보를 고유하게 식별하고 관리합니다.
- **user_id** (BIGINT, NOT NULL, FOREIGN KEY):
 - **설명:** User 테이블과의 관계를 정의하는 외래 키입니다.
 - **목적:** 소셜 계정이 어느 사용자의 것인지 연결합니다.
 - **제약 조건:** 반드시 유효한 User ID여야 하며, 해당 ID가 User 테이블에 존재해야 합니다.
- **provider** (VARCHAR(50), NOT NULL):
 - **설명:** 소셜 로그인 제공자의 이름을 저장합니다(예: 'google', 'facebook', 'twitter' 등).
 - **목적:** 어떤 소셜 플랫폼에서 로그인했는지 식별합니다.
 - **제약 조건:** 이 필드는 NULL이 될 수 없으며, 정해진 소셜 제공자 값만 허용됩니다.
- **social_id** (VARCHAR(255), NOT NULL):
 - **설명:** 해당 소셜 플랫폼에서 제공하는 사용자 고유 ID입니다.
 - **목적:** 사용자의 소셜 계정을 식별하고 연계하는 데 사용됩니다.
 - **제약 조건:** 이 필드는 NULL이 될 수 없으며, 소셜 플랫폼에서 제공하는 유니크한 값이어야 합니다.
- **profile_image_url** (VARCHAR(500), NULL):
 - **설명:** 소셜 계정의 프로필 이미지 URL입니다.
 - **목적:** 사용자의 소셜 계정 프로필 이미지를 관리합니다.
 - **제약 조건:** 선택적으로 사용되며, 소셜 로그인 시 제공되지 않으면 NULL일 수 있습니다.
- **access_token** (VARCHAR(500), NULL):
 - **설명:** 소셜 로그인 시 발급된 접근 토큰입니다.
 - **목적:** 소셜 플랫폼과의 추가 통신을 위해 사용될 수 있습니다.
 - **제약 조건:** 이 필드는 선택적이며, 필요 시 소셜 로그인 인증 과정에서 사용됩니다.
- **refresh_token** (VARCHAR(500), NULL):
 - **설명:** 접근 토큰이 만료된 경우 새로 발급받기 위한 리프레시 토큰입니다.
 - **목적:** 소셜 로그인 세션 유지를 위해 사용됩니다.
 - **제약 조건:** 이 필드도 선택적이며, 제공된 경우에만 값을 가집니다.

3. 데이터베이스 제약 사항 및 인덱스 추가

- **User 테이블 인덱스:**
 - **login_type:** 이 필드에 대한 인덱스를 생성하여, 로그인 타입에 따른 검색 성능을 향상시킵니다.
 - **social_id:** 소셜 로그인을 사용할 때 소셜 ID로 사용자를 빠르게 조회할 수 있도록 인덱스를 추가합니다.
- **SocialAccount 테이블 제약 사항:**
 - **user_id와 provider의 복합 유니크 제약:** 동일한 사용자가 동일한 제공자

(provider)에 대해 중복된 소셜 계정을 가질 수 없도록 user_id와 provider 필드에 대해 복합 유니크 제약을 추가합니다.

- **social_id 유니크 제약:** 소셜 ID는 각 제공자(provider) 내에서 고유해야 하므로, 이 필드에도 유니크 제약을 추가합니다.

4. 관계 및 외래 키 설정

- **User와 SocialAccount 간의 1**

관계:

- 하나의 사용자는 여러 개의 소셜 계정을 가질 수 있으므로, User 테이블과 SocialAccount 테이블 간에 1 관계를 설정합니다.
- SocialAccount 테이블의 user_id 필드에 외래 키(FK) 제약 조건을 설정하여, 관련된 User 레코드가 삭제되면 해당 사용자의 소셜 계정도 삭제되도록 처리할 수 있습니다(ON DELETE CASCADE).

5. 논리적 데이터 모델링의 변경 사항 요약

- **User 테이블:** 새로운 필드(login_type, social_id)가 추가되고, 인덱스가 생성됩니다.
- **SocialAccount 테이블:** 새로운 테이블이 추가되며, 외래 키 및 유니크 제약 조건이 설정됩니다.
- **제약 조건 및 인덱스:** 적절한 인덱스와 제약 조건을 통해 데이터 무결성을 유지하고 성능을 최적화합니다.

3) 물리적 데이터 모델링 변경사항 작성하시오.

1. User 테이블의 물리적 변경 사항

- 테이블 이름: users
- 추가 필드:
 - **login_type VARCHAR(50) NOT NULL:**
 - ◆ 설명: 사용자가 로그인에 사용한 방법을 저장합니다.
 - ◆ 예시 값: 'local', 'google', 'facebook' 등.
 - ◆ 기본값: 'local' (기본 로컬 로그인 방식).
 - **social_id VARCHAR(255) NULL:**
 - ◆ 설명: 소셜 로그인 사용자의 고유 ID를 저장합니다. 로컬 로그인 사용자의 경우 NULL이 될 수 있습니다.
 - ◆ 인덱스: 성능 최적화를 위해 인덱스를 설정합니다.
- 인덱스 추가:
 - 인덱스: idx_users_login_type (login_type): login_type 필드에 대해 인덱스를 추가하여, 사용자의 로그인 타입에 따른 검색 성능을 개선합니다.
 - 인덱스: idx_users_social_id (social_id): social_id 필드에 대해 인덱스를 추가하여, 소셜 ID로 사용자를 검색하는 성능을 향상시킵니다.

2. SocialAccount 테이블 추가

- 테이블 이름: social_accounts
- 필드 정의:
 - **social_account_id BIGINT AUTO_INCREMENT PRIMARY KEY:**
 - ◆ 설명: 소셜 계정 테이블의 고유 식별자입니다.

- **user_id BIGINT NOT NULL:**
 - ◆ **설명:** users 테이블의 외래 키로, 소셜 계정이 속한 사용자를 연결합니다.
 - ◆ **외래 키 제약:** FK_social_accounts_user_id 외래 키 제약을 추가하여 users 테이블과의 관계를 설정합니다.
- **provider VARCHAR(50) NOT NULL:**
 - ◆ **설명:** 소셜 로그인 제공자의 이름을 저장합니다.
 - ◆ **예시 값:** 'google', 'facebook', 'twitter' 등.
- **social_id VARCHAR(255) NOT NULL:**
 - ◆ **설명:** 소셜 로그인 제공자가 발급한 사용자 고유 ID를 저장합니다.
 - ◆ **유니크 제약:** UK_social_accounts_social_id_provider 유니크 제약을 추가하여, 동일한 소셜 ID와 제공자 조합이 중복되지 않도록 합니다.
- **profile_image_url VARCHAR(500) NULL:**
 - ◆ **설명:** 사용자의 소셜 프로필 이미지 URL을 저장합니다.
- **access_token VARCHAR(500) NULL:**
 - ◆ **설명:** 소셜 로그인 시 발급된 접근 토큰을 저장합니다.
- **refresh_token VARCHAR(500) NULL:**
 - ◆ **설명:** 접근 토큰 갱신을 위한 리프레시 토큰을 저장합니다.
- **인덱스 및 제약 조건:**
 - **복합 유니크 제약 조건:** UK_social_accounts_social_id_provider (social_id, provider): 동일한 사용자가 동일한 제공자에서 중복된 소셜 계정을 생성하지 못하도록 제약을 설정합니다.
 - **인덱스:** idx_social_accounts_user_id (user_id): user_id 필드에 대해 인덱스를 추가하여, 사용자와 소셜 계정 간의 관계를 빠르게 조회할 수 있도록 합니다.
- **외래 키 제약:**
 - **FK_social_accounts_user_id:** user_id 필드에 외래 키 제약을 추가하여, users 테이블과의 관계를 설정하고, 관련된 사용자가 삭제되면 연관된 소셜 계정도 삭제되도록 설정합니다 (ON DELETE CASCADE).

3. 제약 조건 및 트리거 설정

- **제약 조건:**
 - **User 테이블의 social_id 제약:** social_id는 로컬 로그인 사용자의 경우 NULL 일 수 있으나, 소셜 로그인 사용자의 경우 NULL이 되지 않도록 로직을 추가할 수 있습니다. 이 로직은 애플리케이션 레벨에서 처리하거나, DBMS의 CHECK 제약 조건을 활용할 수 있습니다.
- **트리거 (Optional):**
 - **소셜 계정 동기화 트리거:** 만약 소셜 계정 정보가 업데이트될 때마다 관련 사용자 정보가 자동으로 동기화되도록 하려면, 트리거를 설정할 수 있습니다. 예를 들어, 사용자가 소셜 프로필을 업데이트할 때 해당 정보를 users 테이블에도 반영하도록 트리거를 설정할 수 있습니다.

4. 물리적 데이터 모델링의 변경 사항 요약

- **User 테이블:** 새로운 필드(login_type, social_id)가 추가되고, 인덱스가 설정됩니다.
- **SocialAccount 테이블:** 새로운 테이블이 추가되며, 외래 키 제약 조건, 유니크 제약 조건, 인덱스가 설정됩니다.
- **제약 조건 및 트리거:** 데이터 무결성을 보장하기 위해 적절한 제약 조건과 트리거를 설정할 수 있습니다.

●

[문6] 해외 인기 밴드의 내한 공연을 위한 공연 예매표를 판매하는 이벤트 사이트를 설계 하시오
<전제 조건>

- 예약은 선착 순으로 진행 되나 결제는 지정일 이내에 완료 하도록 한다
- 지정일 이후 결제는 불가능 하다
- 예약 기간이 종료되면 예약 서비스는 다른 이벤트로 변경 될 수 있다.

문 6-1) 예약을 위한 서비스를 설계 하시오

1. 시스템 요구사항 분석

1. 선착순 예약:

- 사용자들이 좌석을 선착순으로 예약할 수 있어야 합니다.
- 예약된 좌석은 일정 기간 동안 유지되며, 이 기간 동안 결제가 완료되어야 합니다.

2. 결제 유효 기간:

- 예약 후 지정된 기간(예: 24시간) 내에 결제를 완료해야 하며, 이 기간이 지나면 예약이 자동으로 취소됩니다.
- 결제 유효 기간을 사용자별로 관리할 수 있어야 합니다.

3. 예약 기간 종료:

- 예약 기간이 종료되면 현재 이벤트의 예약 서비스를 종료하고, 다음 이벤트로 전환할 수 있어야 합니다.
- 이미 예약된 사용자는 여전히 결제를 완료할 수 있지만, 새로운 예약은 불가능해야 합니다.

2. 서비스 구성 요소 설계

2.1. 예약 서비스 (Reservation Service)

- **기능:** 사용자가 좌석을 예약하는 기능을 제공하며, 선착순으로 좌석을 배정합니다. 예약 상태와 결제 유효 기간을 관리합니다.
- **주요 작업:**
 - 좌석 배정 및 예약 정보 저장.
 - 결제 유효 기간 설정 및 관리.
 - 예약 상태 조회 및 업데이트.

2.2. 결제 서비스 (Payment Service)

- **기능:** 예약된 좌석에 대한 결제를 처리하며, 결제 유효 기간 내에 결제가 완료되지 않은 경우 예약을 자동으로 취소합니다.
- **주요 작업:**
 - 결제 처리 (결제 게이트웨이 연동).
 - 결제 완료 후 예약 상태 업데이트.
 - 결제 유효 기간 경과 시 예약 취소 처리.

2.3. 이벤트 관리 서비스 (Event Management Service)

- **기능:** 현재 활성화된 이벤트와 예약 가능한 상태를 관리하며, 예약 기간 종료 후 이벤트를 변경합니다.
- **주요 작업:**
 - 이벤트 활성화 및 종료 관리.

- 예약 가능한 이벤트 조회.
- 새로운 이벤트로 전환.

3. 서비스 흐름 설계

3.1. 예약 프로세스

1. **이벤트 조회:** 사용자가 예약 가능한 이벤트를 조회합니다.
2. **좌석 선택:** 사용자가 좌석을 선택하고 예약을 요청합니다.
3. **예약 생성:** 예약 서비스에서 좌석을 선착순으로 배정하고 예약 정보를 저장합니다. 결제 유효 기간을 설정합니다.
4. **결제 대기:** 사용자는 지정된 기간 내에 결제를 완료해야 합니다.
5. **결제 완료:** 결제가 완료되면, 결제 서비스에서 예약 상태를 업데이트하고 좌석을 확정합니다.
6. **결제 실패 또는 유효 기간 경과:** 지정된 기간 내에 결제가 완료되지 않으면 예약이 자동으로 취소됩니다.

3.2. 이벤트 전환 프로세스

1. **예약 기간 종료:** 현재 이벤트의 예약 기간이 종료되면 이벤트 관리 서비스에서 예약 서비스의 상태를 "종료"로 설정합니다.
2. **새로운 이벤트 설정:** 새로운 이벤트를 활성화하고, 다음 예약을 시작할 준비를 합니다.

4. 데이터베이스 설계

4.1. Reservation 테이블

- **reservation_id** (PRIMARY KEY): 예약의 고유 식별자.
- **user_id** (FOREIGN KEY): 예약을 생성한 사용자의 ID.
- **event_id** (FOREIGN KEY): 예약된 이벤트의 ID.
- **seat_number**: 예약된 좌석 번호.
- **reservation_status**: 예약 상태 ('예약 중', '결제 완료', '취소됨').
- **payment_due_date**: 결제 유효 기간.

4.2. Payment 테이블

- **payment_id** (PRIMARY KEY): 결제의 고유 식별자.
- **reservation_id** (FOREIGN KEY): 관련된 예약의 ID.
- **payment_status**: 결제 상태 ('결제 중', '결제 완료', '취소됨').
- **payment_date**: 결제 완료 날짜.

4.3. Event 테이블

- **event_id** (PRIMARY KEY): 이벤트의 고유 식별자.
- **event_name**: 이벤트의 이름.
- **start_date**: 이벤트의 시작 날짜.
- **end_date**: 이벤트의 종료 날짜.
- **is_active**: 현재 이벤트의 활성화 상태.

5. 서비스 아키텍처

서비스 간의 통신 방식은 다음과 같이 설정할 수 있습니다:

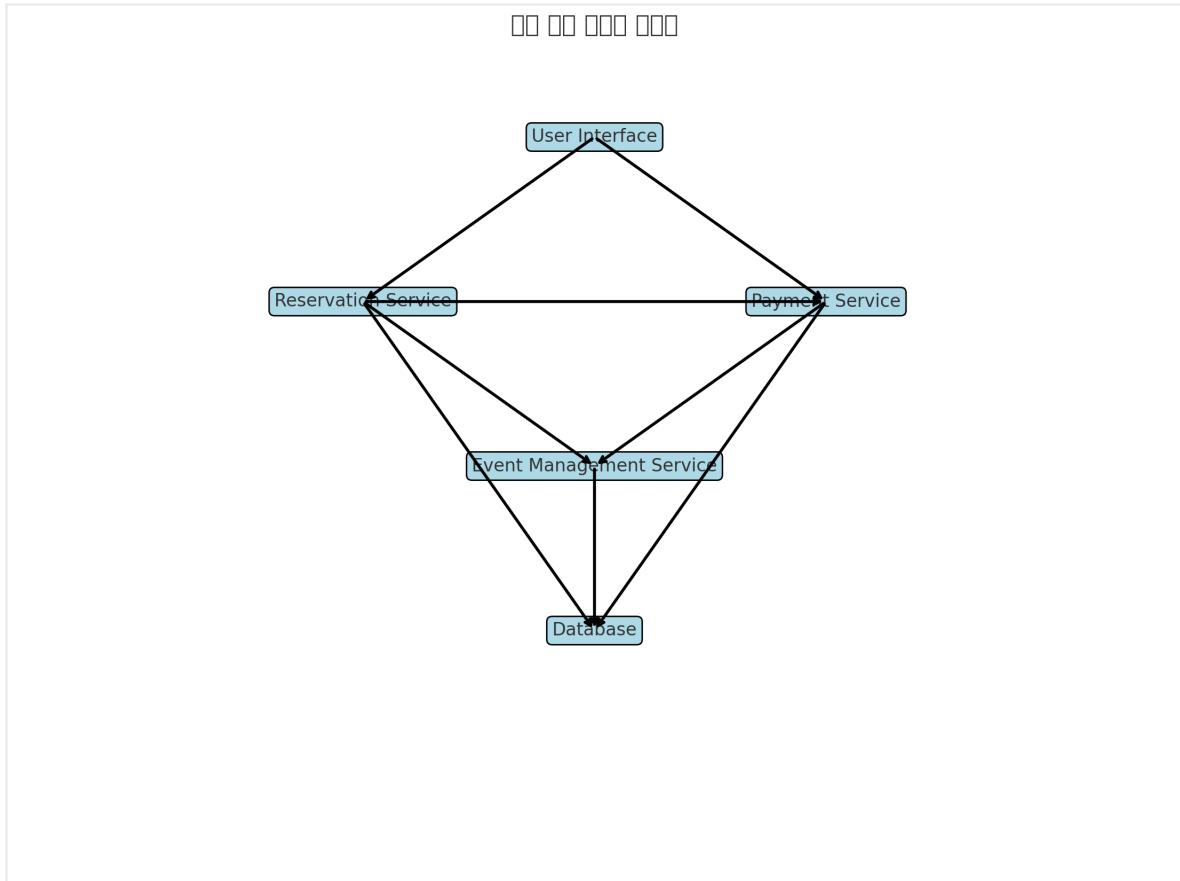
- **동기식 통신:** 예약 서비스와 결제 서비스 간의 통신은 동기식으로 처리합니다. 예약이 완료되면 즉시 결제 처리를 시작할 수 있도록 합니다.
- **비동기식 통신:** 이벤트 관리 서비스는 비동기적으로 예약 서비스와 통신하여 이벤트 변경 사항을 처리할 수 있습니다.

6. 확장 고려 사항

- **좌석 관리:** 좌석 배정 알고리즘을 최적화하여 여러 사용자가 동시에 예약을 시도할 때

발생할 수 있는 충돌을 방지합니다.

- **결제 유효성 검사:** 결제 유효 기간 동안 결제가 완료되지 않으면 자동으로 예약이 취소되도록 처리합니다.
- **이벤트 관리:** 새로운 이벤트를 손쉽게 추가하고 관리할 수 있는 기능을 제공하며, 예약 기간이 종료되면 자동으로 이벤트가 전환되도록 설정합니다.



위 이미지는 공연 예약 서비스의 구성도를 나타낸 것입니다. 각 주요 서비스(User Interface, Reservation Service, Payment Service, Event Management Service) 간의 관계와 데이터베이스와의 상호작용을 시각적으로 표현하였습니다.

- **User Interface:** 사용자가 공연 예약, 결제 등을 진행하는 인터페이스입니다.
- **Reservation Service:** 사용자의 좌석 예약을 관리하며, 결제 서비스와 연동됩니다.
- **Payment Service:** 결제 과정을 처리하며, 예약과 이벤트 관리 서비스와 통신합니다.
- **Event Management Service:** 이벤트의 상태와 예약 가능 여부를 관리합니다.
- **Database:** 각 서비스의 데이터를 저장하고 관리합니다.

문 6-2) 예약과 결제는 분리된 서비스로 구성하여 설계 하시오

1. 서비스 정의 및 역할

1.1. 예약 서비스 (Reservation Service)

- **역할:** 좌석 예약을 관리하며, 사용자가 선택한 좌석을 일정 시간 동안 유지합니다. 예약 상태를 관리하고, 결제 서비스와 통신하여 결제 상태를 업데이트합니다.
- **기능:**
 - 사용자의 좌석 선택 및 예약 처리.

- 예약 정보 저장 및 관리.
- 결제 유효 기간 설정 및 관리.
- 결제 서비스와 통신하여 결제 상태 확인 및 업데이트.
- 예약 기간 종료 후 자동 취소 처리.

1.2. 결제 서비스 (Payment Service)

- **역할:** 예약된 좌석에 대한 결제를 처리하고, 결제 완료 시 예약 서비스에 상태를 업데이트합니다.
- **기능:**
 - 결제 처리 (결제 게이트웨이와 통신).
 - 결제 상태 관리 (결제 중, 완료, 실패 등).
 - 결제 실패 또는 유효 기간 경과 시 예약 서비스에 알림.

2. 서비스 간 통신 및 상호작용

2.1. 동기식 통신

- **예약 서비스 ↔ 결제 서비스:**
 - 예약 서비스는 사용자가 좌석을 예약한 후, 결제 서비스를 호출하여 결제를 진행합니다.
 - 결제 서비스는 결제 요청을 처리한 후, 결과(성공/실패)를 예약 서비스로 반환합니다.
 - 동기식 API 호출을 통해 실시간으로 결제 결과를 예약 서비스에 반영합니다.

2.2. 비동기식 통신

- **예약 서비스 ↔ 결제 서비스:**
 - 결제 서비스에서 결제 실패, 유효 기간 경과 등의 이벤트 발생 시 비동기 메시지 큐 또는 이벤트 스트리밍을 통해 예약 서비스에 알림을 보냅니다.
 - 예약 서비스는 이 메시지를 수신하여 예약을 자동으로 취소하거나, 사용자에게 알림을 발송합니다.

3. 데이터베이스 설계

3.1. Reservation 테이블

- **reservation_id** (PRIMARY KEY): 예약의 고유 식별자.
- **user_id** (FOREIGN KEY): 예약을 생성한 사용자의 ID.
- **event_id** (FOREIGN KEY): 예약된 이벤트의 ID.
- **seat_number**: 예약된 좌석 번호.
- **reservation_status**: 예약 상태 ('예약 중', '결제 대기', '결제 완료', '취소됨').
- **payment_due_date**: 결제 유효 기간.

3.2. Payment 테이블

- **payment_id** (PRIMARY KEY): 결제의 고유 식별자.
- **reservation_id** (FOREIGN KEY): 관련된 예약의 ID.
- **payment_status**: 결제 상태 ('결제 중', '결제 완료', '실패', '취소됨').
- **payment_date**: 결제 완료 날짜.

4. 서비스 간 API 설계

4.1. 예약 서비스 API

- **예약 생성 API:**
 - 메소드: POST
 - 엔드포인트: /api/v1/reservations
 - 설명: 좌석 예약을 생성하고 예약 정보를 반환합니다.
 - 요청 데이터: user_id, event_id, seat_number
 - 응답 데이터: reservation_id, payment_due_date

- **예약 상태 조회 API:**
 - 메소드: GET
 - 엔드포인트: /api/v1/reservations/{reservation_id}
 - 설명: 예약 상태를 조회합니다.
 - 응답 데이터: reservation_id, reservation_status, seat_number
- **예약 취소 API:**
 - 메소드: DELETE
 - 엔드포인트: /api/v1/reservations/{reservation_id}
 - 설명: 예약을 취소합니다.

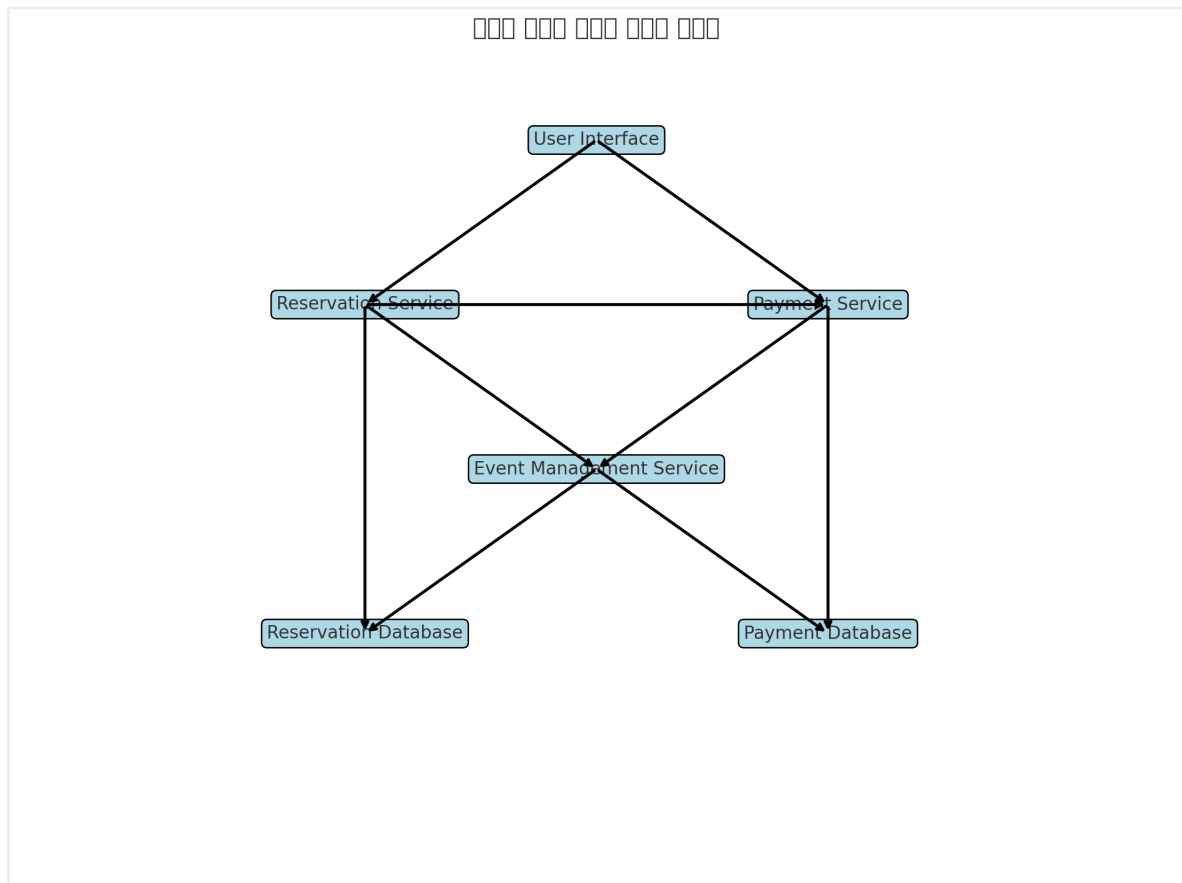
4.2. 결제 서비스 API

- **결제 처리 API:**
 - 메소드: POST
 - 엔드포인트: /api/v1/payments
 - 설명: 예약된 좌석에 대한 결제를 처리합니다.
 - 요청 데이터: reservation_id, payment_method
 - 응답 데이터: payment_id, payment_status
- **결제 상태 조회 API:**
 - 메소드: GET
 - 엔드포인트: /api/v1/payments/{payment_id}
 - 설명: 결제 상태를 조회합니다.
 - 응답 데이터: payment_id, payment_status, payment_date

5. 서비스 흐름

5.1. 예약 프로세스

1. **좌석 선택 및 예약:** 사용자가 좌석을 선택하고, 예약 서비스에 예약 요청을 보냅니다.
2. **예약 확정:** 예약 서비스가 좌석을 선착순으로 배정하고, 결제 유효 기간을 설정합니다.
3. **결제 요청:** 사용자는 결제 서비스에 결제 요청을 보냅니다.
4. **결제 처리:** 결제 서비스가 결제를 처리하고, 결제 상태를 예약 서비스에 알립니다.
5. **예약 확정 또는 취소:** 결제가 완료되면 예약 서비스가 예약 상태를 "결제 완료"로 업데이트합니다. 결제가 실패하거나 유효 기간이 지나면 예약이 취소됩니다.



위 이미지는 예약과 결제를 분리한 서비스 구성도를 나타낸 것입니다. 각 서비스가 독립적으로 운영되며, 필요한 데이터와 상태를 공유하기 위해 서로 통신하는 구조를 시각적으로 표현하였습니다.

- **User Interface:** 사용자가 공연 예약과 결제를 진행하는 인터페이스입니다.
- **Reservation Service:** 좌석 예약을 관리하는 서비스로, 예약 정보와 상태를 관리합니다.
- **Payment Service:** 결제 처리를 담당하는 서비스로, 결제 상태를 관리하고 예약 서비스와 상호작용합니다.
- **Event Management Service:** 이벤트 관리와 예약 가능 여부를 통제하는 서비스입니다.
- **Reservation Database:** 예약 데이터를 저장하는 데이터베이스입니다.
- **Payment Database:** 결제 데이터를 저장하는 데이터베이스입니다.

[문7] A 쇼핑몰에서 가입자 해지 이후 구매 상품을 확인 할 수 있도록 하는 서비스를 설계 중이다
<전제 조건>

- 가입자 해지이 후 모든 포인트는 소멸 한다
- 해지 고객은 최대 1년간 구매 상품을 확인 할 수 있다
- 해지 고객은 더이상 상품을 구매 할 수 없다

문 7-1) 해지 고객 구매 상품 확인 서비스를 위한 ERD를 설계 하시오

1. 요구 사항 분석

- **가입자 해지:** 고객이 서비스 해지 시, 포인트는 소멸하고, 1년간 구매 상품을 확인할 수 있습니다.
- **구매 상품 확인:** 해지 고객은 상품을 구매할 수 없지만, 기존에 구매한 상품을 최대 1년 동안 확인할 수 있습니다.

2. 주요 엔티티(Entity) 도출

2.1. Customer (고객)

- **고객 ID** (customer_id, PRIMARY KEY): 고객의 고유 식별자.
- **이름** (name): 고객의 이름.
- **이메일** (email): 고객의 이메일 주소.
- **해지 일자** (cancellation_date): 고객이 서비스에서 해지된 날짜.
- **해지 상태** (is_cancelled): 고객의 해지 여부 (true/false).

2.2. Purchase (구매 내역)

- **구매 ID** (purchase_id, PRIMARY KEY): 구매 내역의 고유 식별자.
- **고객 ID** (customer_id, FOREIGN KEY): 구매를 진행한 고객의 ID (Customer와 연관).
- **상품 ID** (product_id, FOREIGN KEY): 구매한 상품의 ID (Product와 연관).
- **구매 일자** (purchase_date): 상품을 구매한 날짜.
- **구매 금액** (amount): 구매한 상품의 금액.

2.3. Product (상품)

- **상품 ID** (product_id, PRIMARY KEY): 상품의 고유 식별자.
- **상품 이름** (product_name): 상품의 이름.
- **가격** (price): 상품의 가격.
- **상품 설명** (description): 상품에 대한 설명.

2.4. Logistic (물류)

- **물류 ID** (logistic_id, PRIMARY KEY): 물류 처리의 고유 식별자.
- **구매 ID** (purchase_id, FOREIGN KEY): 관련된 구매의 ID (Purchase와 연관).
- **배송 상태** (shipping_status): 배송 상태 (배송 중, 배송 완료 등).
- **배송 일자** (shipping_date): 상품이 배송된 날짜.

3. 제약 조건 및 비즈니스 규칙

1. **고객 해지 상태 관리:** 고객이 해지된 상태로 표시되면 더 이상 상품을 구매할 수 없으며, 1년 동안만 구매 내역을 조회할 수 있습니다.
2. **포인트 소멸 처리:** 해지된 고객의 포인트는 모두 소멸됩니다. 이 정보는 서비스에서 관리되므로 ERD에서는 생략 가능합니다.
3. **구매 내역 유지:** 해지 고객의 구매 내역은 1년 동안 유지되며, 이후에는 삭제되거나 조회할 수 없도록 처리됩니다.

4. ERD 설계

4.1. Customer 테이블

- **customer_id** (PRIMARY KEY): 고객의 고유 ID.
- **name:** 고객의 이름.
- **email:** 고객의 이메일.
- **cancellation_date:** 해지 일자.
- **is_cancelled:** 해지 상태 (true/false).

4.2. Purchase 테이블

- **purchase_id** (PRIMARY KEY): 구매 내역의 고유 ID.
- **customer_id** (FOREIGN KEY): 고객 ID.
- **product_id** (FOREIGN KEY): 상품 ID.
- **purchase_date**: 구매 일자.
- **amount**: 구매 금액.

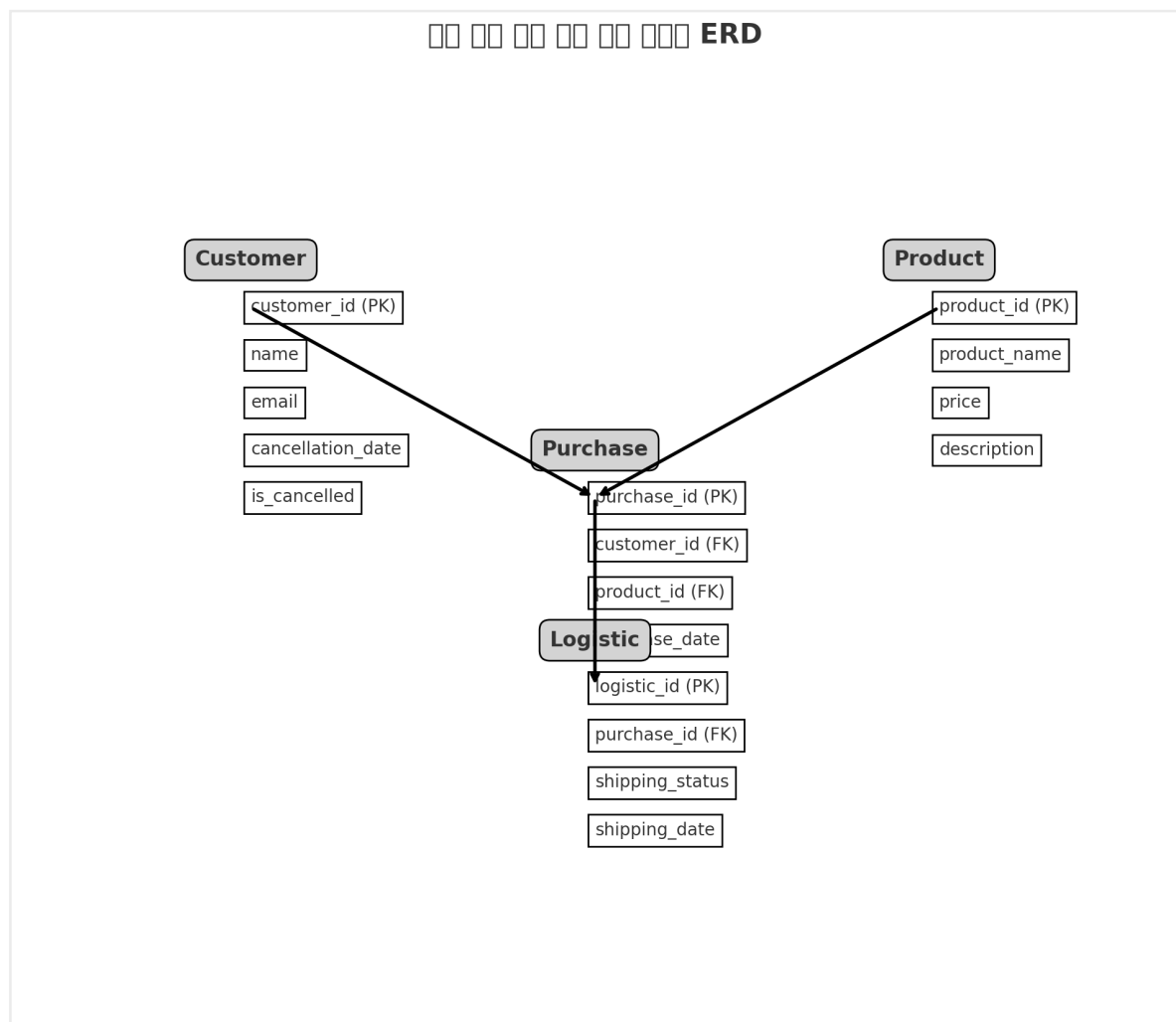
4.3. Product 테이블

- **product_id** (PRIMARY KEY): 상품의 고유 ID.
- **product_name**: 상품의 이름.
- **price**: 상품의 가격.
- **description**: 상품 설명.

4.4. Logistic 테이블

- **logistic_id** (PRIMARY KEY): 물류 ID.
- **purchase_id** (FOREIGN KEY): 구매 ID.
- **shipping_status**: 배송 상태.
- **shipping_date**: 배송 일자.

이 ERD는 해지된 고객이 1년간 구매 내역을 확인할 수 있는 기능을 지원하는 데 필요한 주요 테이블과 관계를 정의합니다. 각 테이블은 고객, 구매, 상품, 물류 정보를 관리하며, 해지된 고객의 상태를 반영하여 조회 및 관리가 이루어지도록 설계되었습니다.



위 이미지는 해지 고객의 구매 상품 확인 서비스를 위한 ERD입니다. 이 ERD는 고객 (Customer), 구매 내역(Purchase), 상품(Product), 그리고 물류(Logistic) 테이블을 포함하며, 각 테이블 간의 관계를 나타냅니다.

- **Customer:** 고객의 기본 정보와 해지 상태를 관리합니다.
- **Purchase:** 고객의 구매 내역을 관리합니다.
- **Product:** 쇼핑몰에서 판매되는 상품 정보를 관리합니다.
- **Logistic:** 구매한 상품의 배송 정보를 관리합니다.

문 7-2) 해지 고객 서비스에 대한 프로세스를 설계 하시오

1. 해지 고객 서비스 프로세스 개요

고객이 서비스에서 해지되면, 다음과 같은 프로세스가 시작됩니다:

1. **해지 처리:** 고객의 계정이 해지되고, 관련된 포인트와 서비스 이용 권한이 소멸됩니다.
2. **구매 상품 조회 제한 설정:** 해지된 고객은 최대 1년간 구매 상품을 조회할 수 있도록 시스템에서 관리합니다.
3. **구매 제한:** 해지된 고객은 더 이상 새로운 상품을 구매할 수 없도록 제한됩니다.
4. **구매 내역 관리:** 해지 후 1년이 지난 구매 내역은 자동으로 조회 불가 상태로 변경되거나 삭제됩니다.

2. 해지 고객 서비스 프로세스 단계별 설명

2.1. 해지 요청 및 처리

- **고객이 서비스 해지를 요청하면,** 시스템은 고객의 상태를 "해지됨"으로 변경합니다.
- **포인트 소멸:** 해지 시 고객의 모든 포인트가 소멸되며, 포인트는 더 이상 복구되지 않습니다.
- **계정 정보 업데이트:** 고객의 해지 일자를 기록하고, 계정의 상태를 "해지됨"으로 설정합니다.

2.2. 구매 상품 조회 서비스 제공

- **구매 내역 조회 제한:** 해지된 고객은 로그인 후 최대 1년 동안 자신의 구매 내역을 확인할 수 있습니다. 이 기간 동안은 상품 상세 정보(상품명, 가격, 구매 날짜 등)와 배송 상태를 확인할 수 있습니다.
- **구매 내역 접근 제한:** 해지된 고객은 더 이상 새로운 상품을 구매할 수 없습니다. 이를 위해 시스템은 구매 요청을 차단합니다.

2.3. 구매 내역 만료 및 삭제

- **구매 내역 유지 기간:** 해지 일자로부터 1년이 경과하면, 고객의 구매 내역은 더 이상 조회할 수 없도록 만료 처리됩니다.
- **자동 삭제:** 선택적으로, 시스템은 1년이 지난 구매 내역을 자동으로 삭제하여 데이터베이스 용량을 관리할 수 있습니다. 삭제 시 고객에게 별도의 알림을 보내지 않습니다.

2.4. 구매 내역 조회 프로세스

- **로그인 처리:** 해지된 고객이 로그인하면, 시스템은 고객의 해지 상태를 확인하고 구매 내역 조회 페이지로 안내합니다.
- **조회 가능 항목:** 고객은 해지 이전에 구매한 상품의 목록, 상세 정보, 배송 상태 등을 확인할 수 있습니다.
- **구매 불가 처리:** 시스템은 해지된 고객이 상품을 추가로 구매하거나 장바구니에 담는 것을 차단합니다.

3. 해지 고객 서비스 프로세스 흐름도

1. 고객 요청:

- 고객이 서비스 해지를 요청.
- 시스템은 고객의 상태를 "해지됨"으로 업데이트.

- 고객의 포인트를 소멸시키고, 구매 기능을 제한.

2. 고객 로그인:

- 해지된 고객이 로그인 시도.
- 시스템은 해지 상태를 확인하고 구매 내역 조회 페이지로 리다이렉트.

3. 구매 내역 조회:

- 고객은 1년 동안 구매 내역을 확인 가능.
- 시스템은 1년 후 자동으로 구매 내역을 만료 처리.

4. 구매 내역 삭제 또는 보관:

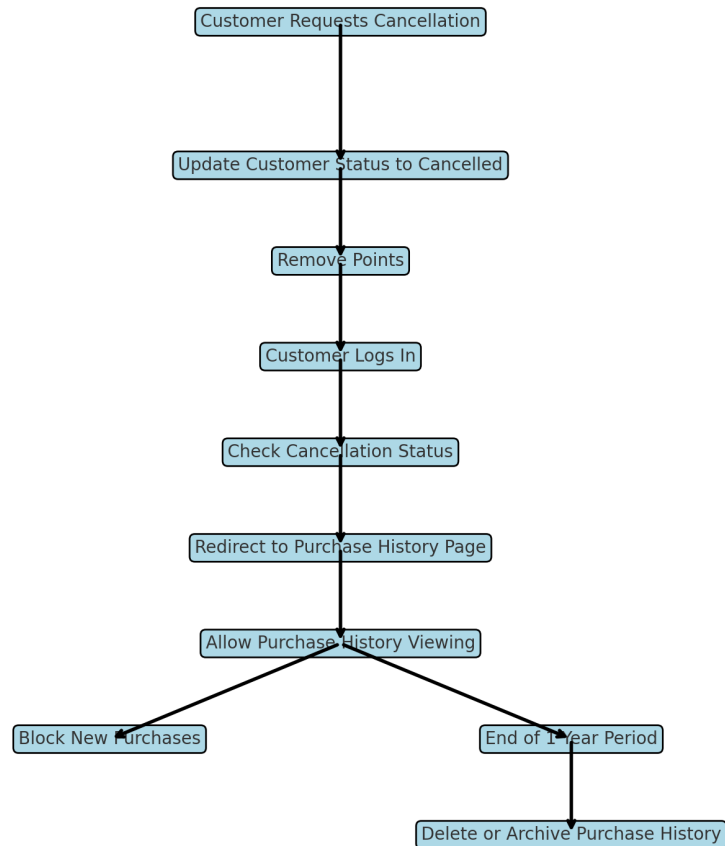
- 선택적으로, 1년이 지난 구매 내역을 삭제하여 데이터베이스 관리.
- 보관 정책에 따라 삭제 여부 결정.

4. 보안 및 데이터 관리

- **데이터 보안:** 해지된 고객의 개인 정보와 구매 내역은 보호되어야 합니다. 시스템은 데이터 접근 권한을 철저히 관리하여, 해지된 고객의 정보가 부당하게 노출되지 않도록 해야 합니다.
- **데이터 정리:** 1년이 지난 구매 내역은 데이터베이스에서 정리(삭제 또는 아카이빙)하여 시스템의 성능과 효율성을 유지합니다.

5. 프로세스 구현에 대한 추가 고려사항

- **알림 서비스:** 해지 후 1년이 지나기 전에 고객에게 알림을 보내 구매 내역이 곧 만료된다는 정보를 제공할 수 있습니다.
- **사용자 경험:** 해지된 고객이 남은 기간 동안 구매 내역을 편리하게 조회할 수 있도록, 사용자 인터페이스(UI)를 설계합니다.
- **법적 준수:** 해지된 고객의 데이터 보관 및 삭제는 해당 지역의 데이터 보호 법률에 따라 이루어져야 합니다.



각 단계는 다음과 같습니다:

1. **Customer Requests Cancellation:** 고객이 해지를 요청합니다.
2. **Update Customer Status to Cancelled:** 고객의 상태가 '해지됨'으로 업데이트됩니다.
3. **Remove Points:** 고객의 포인트가 소멸됩니다.
4. **Customer Logs In:** 고객이 다시 로그인합니다.
5. **Check Cancellation Status:** 시스템이 고객의 해지 상태를 확인합니다.
6. **Redirect to Purchase History Page:** 고객은 구매 내역 페이지로 이동됩니다.
7. **Allow Purchase History Viewing:** 고객은 최대 1년간 구매 내역을 확인할 수 있습니다.
8. **Block New Purchases:** 고객은 더 이상 새 상품을 구매할 수 없습니다.
9. **End of 1 Year Period:** 1년이 경과한 후.
10. **Delete or Archive Purchase History:** 구매 내역이 삭제되거나 보관됩니다.

문 7-3) 해지 고객 서비스를 위한 API를 설계 하시오

1. API 설계 개요

1.1. 고객 해지 처리 API

- **API 이름:** Cancel Subscription
- **메소드:** POST
- **엔드포인트:** /api/v1/customers/{customer_id}/cancel
- **설명:** 고객이 서비스 해지를 요청할 때 호출됩니다. 고객의 상태를 "해지됨"으로 변경하고,

포인트를 소멸시킵니다.

- **요청 데이터:**
 - customer_id: (URL 파라미터) 해지할 고객의 ID
- **응답 데이터:**
 - status: 처리 결과 (예: "success", "failure")
 - message: 결과에 대한 설명
 - cancellation_date: 해지 일자
- **예시 응답:** json코드 복사

```
{  
  "status": "success",  
  "message": "Customer subscription cancelled.",  
  "cancellation_date": "2023-08-09"  
}
```

1.2. 해지 고객 로그인 확인 API

- **API 이름:** Check Cancellation Status
- **메소드:** GET
- **엔드포인트:** /api/v1/customers/{customer_id}/status
- **설명:** 고객이 로그인할 때, 고객의 해지 상태를 확인하고 해지 여부에 따라 구매 내역 조회 페이지로 리다이렉트합니다.
- **요청 데이터:**
 - customer_id: (URL 파라미터) 확인할 고객의 ID
- **응답 데이터:**
 - is_cancelled: 고객의 해지 상태 (true/false)
 - cancellation_date: 해지 일자 (해지된 경우에만 반환)
- **예시 응답:** json코드 복사

```
{  
  "is_cancelled": true,  
  "cancellation_date": "2023-08-09"  
}
```

1.3. 해지 고객의 구매 내역 조회 API

- **API 이름:** Get Purchase History
- **메소드:** GET
- **엔드포인트:** /api/v1/customers/{customer_id}/purchases
- **설명:** 해지된 고객이 자신의 구매 내역을 조회할 수 있습니다. 구매 내역은 해지 후 1년간만 조회 가능합니다.
- **요청 데이터:**
 - customer_id: (URL 파라미터) 구매 내역을 조회할 고객의 ID

- **응답 데이터:**
 - purchases: 고객의 구매 내역 목록 (구매 ID, 상품명, 구매일, 금액 등 포함)
- **예시 응답:** json코드 복사

```
{
  "purchases": [
    {
      "purchase_id": 12345,
      "product_name": "Concert Ticket",
      "purchase_date": "2023-07-15",
      "amount": 150.00
    },
    {
      "purchase_id": 12346,
      "product_name": "Band T-Shirt",
      "purchase_date": "2023-07-16",
      "amount": 25.00
    }
  ]
}
```

1.4. 구매 내역 삭제 또는 만료 처리 API

- **API 이름:** Expire Purchase History
- **메소드:** DELETE
- **엔드포인트:** /api/v1/customers/{customer_id}/purchases/expire
- **설명:** 해지된 고객의 구매 내역이 1년이 지나 만료될 경우, 해당 내역을 삭제하거나 접근을 제한합니다.
- **요청 데이터:**
 - customer_id: (URL 파라미터) 구매 내역을 만료 처리할 고객의 ID
- **응답 데이터:**
 - status: 처리 결과 (예: "success", "failure")
 - message: 결과에 대한 설명
- **예시 응답:** json코드 복사

```
{
  "status": "success",
  "message": "Customer purchase history expired."
}
```

1.5. 신규 구매 차단 API

- **API 이름:** Block New Purchases

- 메소드: POST
- 엔드포인트: /api/v1/customers/{customer_id}/block_purchases
- 설명: 해지된 고객이 새로운 상품을 구매할 수 없도록 차단합니다. 해당 API는 해지 시 자동으로 호출됩니다.
- 요청 데이터:
 - customer_id: (URL 파라미터) 신규 구매를 차단할 고객의 ID
- 응답 데이터:
 - status: 처리 결과 (예: "success", "failure")
 - message: 결과에 대한 설명
- 예시 응답: json코드 복사

```
{
  "status": "success",
  "message": "Customer is blocked from making new purchases."
}
```

2. API 사용 시나리오

- 고객 해지 처리:
 1. 고객이 서비스 해지를 요청하면, Cancel Subscription API가 호출되어 고객의 상태를 "해지됨"으로 변경하고 포인트를 소멸시킵니다.
 2. Block New Purchases API를 호출하여 고객의 신규 구매를 차단합니다.
- 고객 로그인:
 1. 해지된 고객이 로그인할 때, Check Cancellation Status API를 통해 고객의 해지 상태를 확인합니다.
 2. 해지된 경우, 고객을 구매 내역 조회 페이지로 리다이렉트합니다.
- 구매 내역 조회:
 1. 고객은 Get Purchase History API를 통해 자신의 구매 내역을 조회할 수 있습니다.
 2. 해지 후 1년이 지나면 Expire Purchase History API가 자동으로 호출되어 구매 내역을 만료 처리합니다.

3. 보안 및 접근 제어

- 모든 API는 인증된 사용자만 접근할 수 있도록 JWT 또는 OAuth와 같은 인증 방식을 사용합니다.
- API 호출 시 고객의 해지 상태와 권한을 확인하여, 적절한 서비스에 접근하도록 제한합니다.

[문8] 글로벌 OTT 서비스를 개발 하는 과정에서 대표 사용자가 인증한 그룹 계정 사용자 1개에 대하여 동시 접속 가능한 서비스를 제공하고자 한다 해당 서비스를 제공하려고 한다.

<전제 조건>

- 대표 사용자 및 그룹 계정 사용자는 1개 단말에서만 접속이 가능하다
- 2개의 계정은 서로 사용자 Profile을 공유하지 않는다.

- 각각의 사용자는 개별 인증을 통해 접속 한다

문 7-1) 개별 사용자에게 대한 로그인 프로세스를 설계하시오

1. 로그인 프로세스 개요

1.1. 단말기 고유 식별

- 각 사용자가 로그인할 때, 사용자가 접속한 단말기의 고유 식별자(Device ID)를 확인합니다. 이 식별자는 사용자가 다른 단말기에서 로그인하려고 할 때 기존의 세션을 만료시키거나, 접속을 차단하는 데 사용됩니다.

1.2. 사용자 인증

- 사용자 인증은 OAuth2, JWT 또는 SAML과 같은 인증 방법을 통해 이루어지며, 인증된 세션에 대한 토큰을 발급합니다. 이 토큰은 사용자가 동일한 단말기에서만 인증된 세션을 유지할 수 있도록 관리됩니다.

1.3. 단말기 제어

- 시스템은 사용자 ID와 단말기 ID를 결합하여, 하나의 사용자 계정이 한 번에 하나의 단말기에서만 로그인할 수 있도록 제한합니다.

2. 로그인 프로세스 단계

2.1. 로그인 요청

- 사용자가 로그인 페이지에서 이메일 또는 사용자 ID와 비밀번호를 입력하여 로그인 요청을 보냅니다.

2.2. 단말기 ID 확인

- 시스템은 로그인 요청을 받은 후, 해당 단말기의 고유 식별자(Device ID)를 확인합니다.
 - 만약 해당 사용자가 이미 다른 단말기에서 로그인한 상태라면, 기존 세션을 만료시키고 새로운 세션을 허용하거나, 로그인을 차단합니다.

2.3. 사용자 인증

- 사용자가 입력한 자격 증명(예: 이메일/비밀번호, 소셜 로그인 등)을 인증 서버에 전달하여 인증을 시도합니다.
 - 성공 시, 시스템은 인증 토큰(JWT 또는 세션 토큰)을 발급하고, 이를 사용자의 단말기에 저장합니다.

2.4. 단말기 등록 및 세션 관리

- 시스템은 발급된 인증 토큰과 함께 단말기 ID를 등록하여 해당 사용자가 현재 어떤 단말기에서 로그인했는지 기록합니다.
 - 사용자가 동일한 계정으로 다른 단말기에서 로그인할 경우, 기존 세션을 만료시키거나 차단합니다.

2.5. 세션 유지 및 검증

- 사용자가 서비스에 접근할 때마다 시스템은 인증 토큰과 단말기 ID를 확인하여 유효성을 검증합니다.
 - 토큰이 만료되거나 단말기 ID가 일치하지 않으면, 사용자는 로그아웃되거나 다시 인증 절차를 밟아야 합니다.

2.6. 로그아웃 및 세션 종료

- 사용자가 로그아웃을 시도하면, 시스템은 해당 세션과 연결된 단말기 ID를 제거하고, 인증 토큰을 만료시킵니다.
 - 로그아웃 후 동일한 계정으로 다시 로그인하려면, 새로 인증 과정을 거쳐야 합니다.

3. 로그인 프로세스 흐름도

1. 로그인 요청:

- 사용자가 이메일/사용자 ID와 비밀번호를 입력하고 로그인 요청을 보냅니다.

2. 단말기 ID 확인:

- 시스템이 현재 접속한 단말기의 고유 식별자(Device ID)를 확인합니다.

3. 기존 세션 확인:

- 동일한 사용자 ID로 다른 단말기에서 활성화된 세션이 있는지 확인합니다.
- 있으면 기존 세션을 만료시키거나, 새 로그인을 차단합니다.

4. 사용자 인증:

- 자격 증명을 검증하고, 성공 시 인증 토큰을 발급합니다.

5. 단말기 등록:

- 새로운 세션과 단말기 ID를 시스템에 등록하여 관리합니다.

6. 서비스 접근:

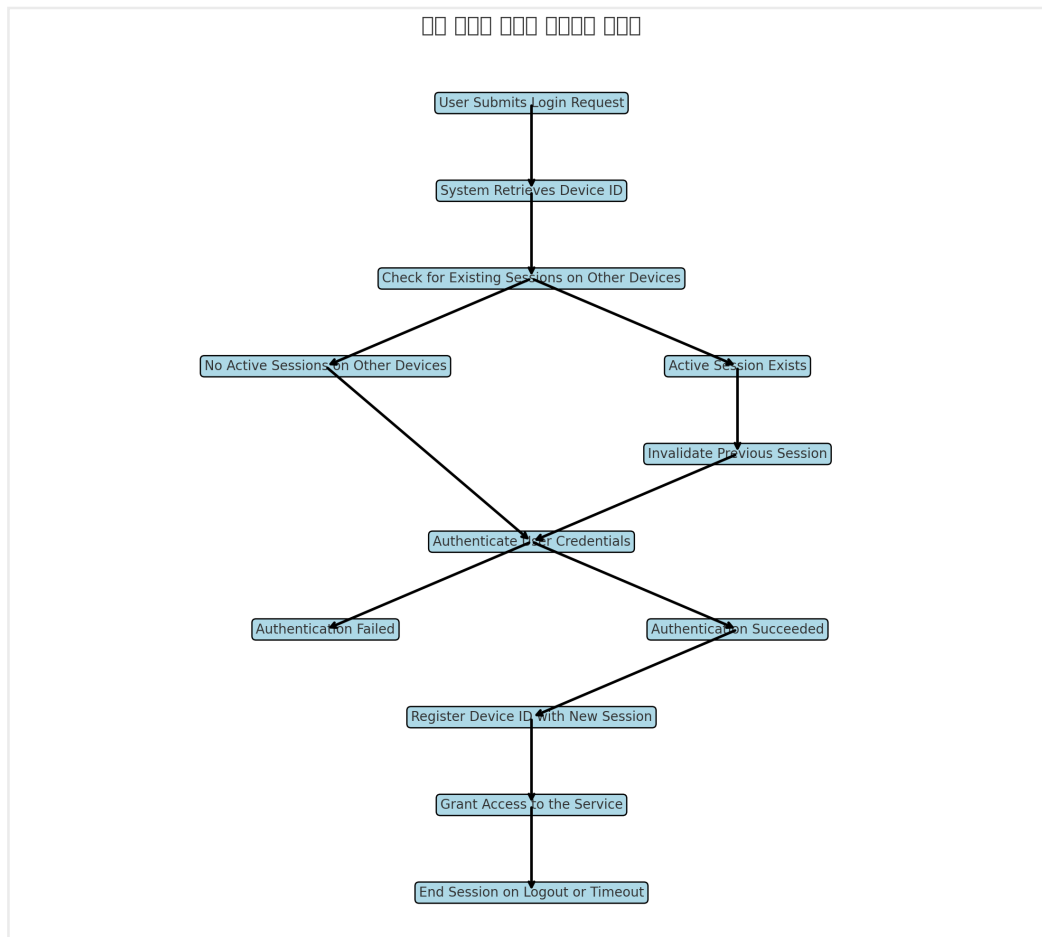
- 사용자는 인증된 세션을 통해 서비스에 접근할 수 있습니다.

7. 로그아웃 및 세션 만료:

- 로그아웃 시 세션과 단말기 ID를 시스템에서 제거하고, 인증 토큰을 만료시킵니다.

4. 보안 고려 사항

- **단말기 ID 관리:** 고유한 단말기 ID를 안전하게 관리하며, 이 정보가 외부로 노출되지 않도록 보호합니다.
- **세션 타임아웃:** 일정 기간 동안 사용자가 활동하지 않으면 자동으로 세션이 만료되도록 설정합니다.
- **2단계 인증:** 추가 보안을 위해 2단계 인증(2FA)을 도입할 수 있습니다.
- **비정상 활동 탐지:** 동일한 계정으로 여러 단말기에서 비정상적인 로그인 시도를 탐지하고, 보안 경고를 발송합니다.



위 이미지는 개별 사용자 로그인 프로세스의 흐름도를 시각적으로 표현한 것입니다. 이 흐름도는 사용자가 로그인 요청을 제출한 후, 단말기 ID를 확인하고, 기존 세션을 검사한 다음 사용자 인증을 처리하는 전체 과정을 단계별로 보여줍니다.

- **User Submits Login Request:** 사용자가 로그인 요청을 보냅니다.
- **System Retrieves Device ID:** 시스템이 현재 접속한 단말기의 고유 식별자 (Device ID)를 확인합니다.
- **Check for Existing Sessions on Other Devices:** 동일한 사용자 계정으로 다른 단말기에 활성화된 세션이 있는지 확인합니다.
- **Invalidate Previous Session:** 기존에 활성화된 세션이 있으면 만료시키고, 새 세션을 허용합니다.
- **Authenticate User Credentials:** 사용자의 자격 증명을 검증합니다.
- **Register Device ID with New Session:** 새 세션을 단말기 ID와 함께 등록합니다.
- **Grant Access to the Service:** 인증이 완료되면 서비스를 이용할 수 있도록 합니다.
- **End Session on Logout or Timeout:** 로그아웃 또는 세션 타임아웃 시 세션을 종료합니다.

문 7-2) 콘텐츠 재생 또는 구매 이력 확인 서비스등에서 로그인 인증을 위한 프로세스를 설계하시오

1. 로그인 인증 프로세스 개요

이 프로세스는 사용자가 콘텐츠 재생이나 구매 이력 확인을 시도할 때, 해당 사용자의 로그인 상태와 세션 유효성을 확인하고, 필요에 따라 인증 절차를 다시 수행하게 됩니다.

2. 로그인 인증 프로세스 단계

2.1. 사용자 요청 (Request)

- 사용자가 콘텐츠 재생 또는 구매 이력 확인 요청을 보냅니다.
- 요청 시, 사용자는 자신의 세션 토큰(JWT 또는 세션 쿠키 등)을 포함하여 서버에 전달합니다.

2.2. 세션 유효성 검사 (Session Validation)

- 서버는 요청에 포함된 세션 토큰을 확인하여, 세션이 유효한지, 만료되지 않았는지, 그리고 해당 토큰이 현재 사용 중인 단말기에서 발급된 것인지 검증합니다.
 - **세션 유효:** 사용자가 요청한 콘텐츠 또는 구매 이력에 접근할 수 있도록 허용합니다.
 - **세션 만료:** 세션이 만료되었거나 유효하지 않은 경우, 사용자는 재인증을 위해 로그인 페이지로 리다이렉트됩니다.

2.3. 재인증 요청 (Re-authentication)

- 세션이 유효하지 않거나 만료된 경우, 사용자는 다시 로그인하도록 요구됩니다.
- 사용자가 로그인하면, 새로운 세션 토큰을 발급하고, 기존의 만료된 세션을 무효화합니다.

2.4. 액세스 권한 확인 (Access Authorization)

- 세션이 유효한 경우, 서버는 사용자가 요청한 리소스에 접근할 권한이 있는지 추가로 확인합니다.
 - 예를 들어, 콘텐츠 재생 권한이 있는지, 구매 이력을 볼 수 있는지 확인합니다.

2.5. 서비스 제공 (Content Delivery / Purchase History Access)

- 인증된 세션이 유효하고 사용자가 접근 권한을 가지고 있다면, 요청한 콘텐츠를 스트리밍하거나, 구매 이력을 제공합니다.
- 만약 접근 권한이 없다면, 접근 거부 메시지를 반환하거나, 추가 결제가 필요함을 알립니다.

3. 로그인 인증 프로세스 흐름도

1. 사용자 요청:

- 사용자가 콘텐츠 재생 또는 구매 이력 확인 요청을 보냅니다.
- 세션 토큰을 포함하여 서버에 전달됩니다.

2. 세션 유효성 검사:

- 서버가 세션 토큰의 유효성을 확인합니다.
- 세션이 유효하지 않으면, 로그인 페이지로 리다이렉트하여 재인증을 요구합니다.

3. 재인증 요청:

- 세션이 만료되었거나 유효하지 않을 경우, 사용자는 다시 로그인해야 합니다.
- 새로운 세션이 발급되고, 기존 세션이 만료됩니다.

4. 액세스 권한 확인:

- 서버는 사용자가 요청한 콘텐츠 또는 구매 이력에 접근할 권한이 있는지 확인합니다.

5. 서비스 제공:

- 사용자가 인증된 상태라면, 요청한 콘텐츠를 스트리밍하거나, 구매 이력을 제공합니다.
- 접근 권한이 없으면, 접근 거부 메시지를 반환합니다.

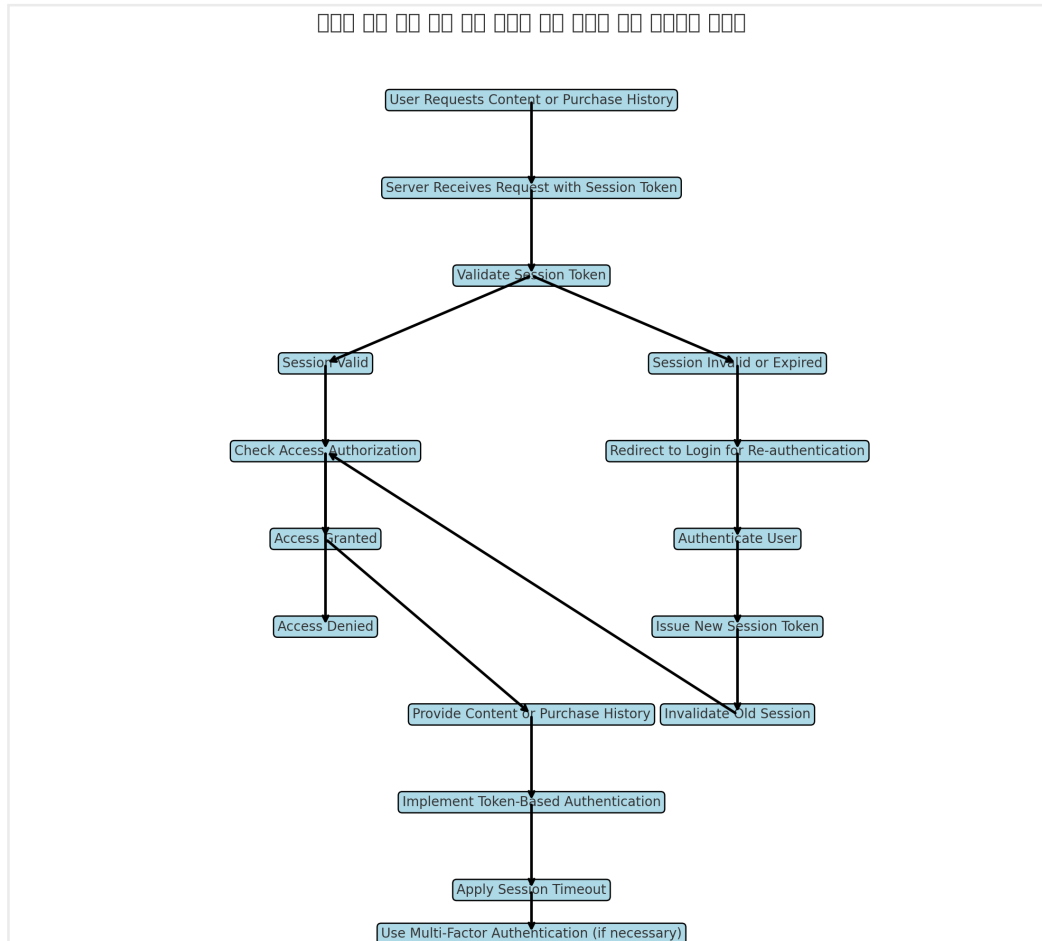
4. 보안 고려 사항

- **세션 관리:** 세션 토큰이 유출되지 않도록 보안 조치를 강화하며, HTTPS를 사용하여 통신을 암호화합니다.
- **세션 타임아웃:** 일정 시간 동안 활동이 없을 경우 세션을 자동으로 만료시킵니다.
- **재인증 주기:** 중요한 콘텐츠나 민감한 정보에 접근할 때 주기적으로 재인증을 요구할 수

있습니다.

5. 프로세스 구현 고려사항

- **토큰 기반 인증:** JWT를 사용하여 세션 상태를 관리하고, 각 요청마다 토큰을 검증하여 유효성을 판단합니다.
- **다중 인증 (MFA):** 필요에 따라 다중 인증을 통해 보안 강화를 할 수 있습니다.
- **사용자 피드백:** 만료된 세션에 대한 명확한 메시지와 재인증 절차를 사용자에게 안내합니다.



위 이미지는 콘텐츠 재생 또는 구매 이력 확인을 위한 로그인 인증 프로세스의 흐름도를 시각적으로 표현한 것입니다. 이 흐름도는 사용자의 요청이 처리되는 과정을 단계별로 보여주며, 세션 관리, 재인증, 그리고 서비스 접근 권한 확인과 같은 중요 단계를 포함합니다.

주요 프로세스는 다음과 같습니다:

1. **사용자 요청:** 사용자가 콘텐츠 재생 또는 구매 이력 확인을 요청합니다.
2. **세션 유효성 검사:** 서버가 세션 토큰을 확인하고, 유효성을 검사합니다.
3. **재인증:** 세션이 만료되었거나 유효하지 않은 경우, 사용자는 다시 로그인해야 합니다.
4. **액세스 권한 확인:** 세션이 유효한 경우, 사용자가 요청한 리소스에 대한 접근 권한을 확인합니다.
5. **서비스 제공:** 사용자가 인증된 상태라면, 요청한 콘텐츠나 구매 이력을 제공합니다.

문 7-3) OTT 계정 이외에 추가적으로 외부 계정을 연동한 로그인 서비스를 제공 하고자 하는 경우 방법을 설계 하시오.

1. 외부 계정 연동 로그인 서비스 개요

1.1. 기존 OTT 계정과 외부 계정 연동

- 사용자는 OTT 서비스에 로그인할 때, 기존 OTT 계정을 사용하거나, 외부 계정(Google,

Facebook, Apple ID 등)으로 로그인할 수 있는 옵션을 선택할 수 있습니다.

- 외부 계정으로 로그인하는 경우, OTT 서비스는 외부 인증 제공자로부터 사용자 정보를 받아 이를 OTT 계정 시스템에 연동하거나 새로운 계정을 생성합니다.

1.2. OAuth 2.0 및 OpenID Connect 활용

- 외부 계정 연동을 위해 OAuth 2.0 또는 OpenID Connect를 활용합니다. 이 프로토콜들은 외부 인증 제공자로부터 사용자 인증을 받고, 인증된 사용자 정보를 OTT 서비스에 전달합니다.
- OTT 서비스는 외부 계정에서 제공받은 사용자 정보를 이용해 기존 계정과 연동하거나, 새로운 계정을 생성합니다.

2. 외부 계정 연동 로그인 서비스 프로세스

2.1. 로그인 페이지 제공

- 사용자가 OTT 서비스 로그인 페이지에 접속하면, OTT 계정으로 로그인할지 외부 계정(Google, Facebook, Apple 등)으로 로그인할지 선택할 수 있는 옵션을 제공합니다.

2.2. 외부 계정 로그인 요청

- 사용자가 외부 계정으로 로그인을 선택하면, 해당 외부 인증 제공자(Google, Facebook 등)로 리다이렉트됩니다.
- 사용자는 외부 인증 제공자의 로그인 페이지에서 자신의 자격 증명(이메일/비밀번호 등)을 입력하고 인증을 시도합니다.

2.3. 외부 인증 제공자와의 통신

- 인증이 성공하면, 외부 인증 제공자는 OTT 서비스에 인증 코드 또는 토큰을 전달합니다.
- OTT 서비스는 이 인증 코드를 사용해 외부 인증 제공자로부터 액세스 토큰과 사용자 정보를 요청합니다.

2.4. OTT 계정 연동 또는 생성

- OTT 서비스는 외부 계정에서 제공받은 사용자 정보를 바탕으로 기존의 OTT 계정과 연동합니다.
 - 만약 해당 이메일이나 외부 계정 ID가 이미 OTT 계정에 연동되어 있다면, 해당 계정으로 로그인합니다.
 - 연동되지 않은 경우, 새로운 OTT 계정을 생성하거나 사용자가 연동을 선택할 수 있는 옵션을 제공합니다.

2.5. 세션 생성 및 서비스 제공

- 인증이 완료되면 OTT 서비스는 사용자에게 세션을 발급하고, 정상적으로 서비스를 이용할 수 있도록 합니다.

3. 보안 고려사항

3.1. CSRF 보호

- 외부 계정 로그인 프로세스에서 발생할 수 있는 CSRF(Cross-Site Request Forgery) 공격을 방지하기 위해, 로그인 요청에 CSRF 토큰을 포함하여 검증합니다.

3.2. OAuth 2.0 취약점 방어

- OAuth 2.0 프로토콜을 구현할 때, 리디렉션 URI의 엄격한 검증과 PKCE(Proof Key for Code Exchange)와 같은 보안 강화 메커니즘을 도입하여 OAuth의 잠재적 취약점을 방어합니다.

3.3. 데이터 보호 및 암호화

- 외부 계정에서 전달받은 민감한 사용자 정보는 안전하게 저장 및 처리해야 하며, 데이터 전송 시 TLS/SSL을 사용하여 암호화합니다.

4. 프로세스 흐름도

4.1. 사용자 선택

1. **OTT 서비스 로그인 페이지:** 사용자가 OTT 계정 또는 외부 계정으로 로그인 옵션을 선택합니다.

4.2. 외부 계정 인증

2. **외부 인증 제공자 리다이렉트:** 사용자가 외부 계정을 선택하면 해당 인증 제공자의 로그인 페이지로 리다이렉트됩니다.

3. **사용자 인증:** 외부 인증 제공자에서 사용자가 자격 증명을 입력하고 인증합니다.

4.3. OTT 계정 연동

4. **인증 코드 수신 및 액세스 토큰 요청:** 인증이 성공하면 OTT 서비스는 인증 코드를 받아 액세스 토큰과 사용자 정보를 요청합니다.

5. **사용자 정보 수신:** 외부 인증 제공자가 사용자 정보를 OTT 서비스로 전달합니다.

6. **OTT 계정 연동:** OTT 서비스가 받은 사용자 정보를 바탕으로 기존 계정과 연동하거나 새로운 계정을 생성합니다.

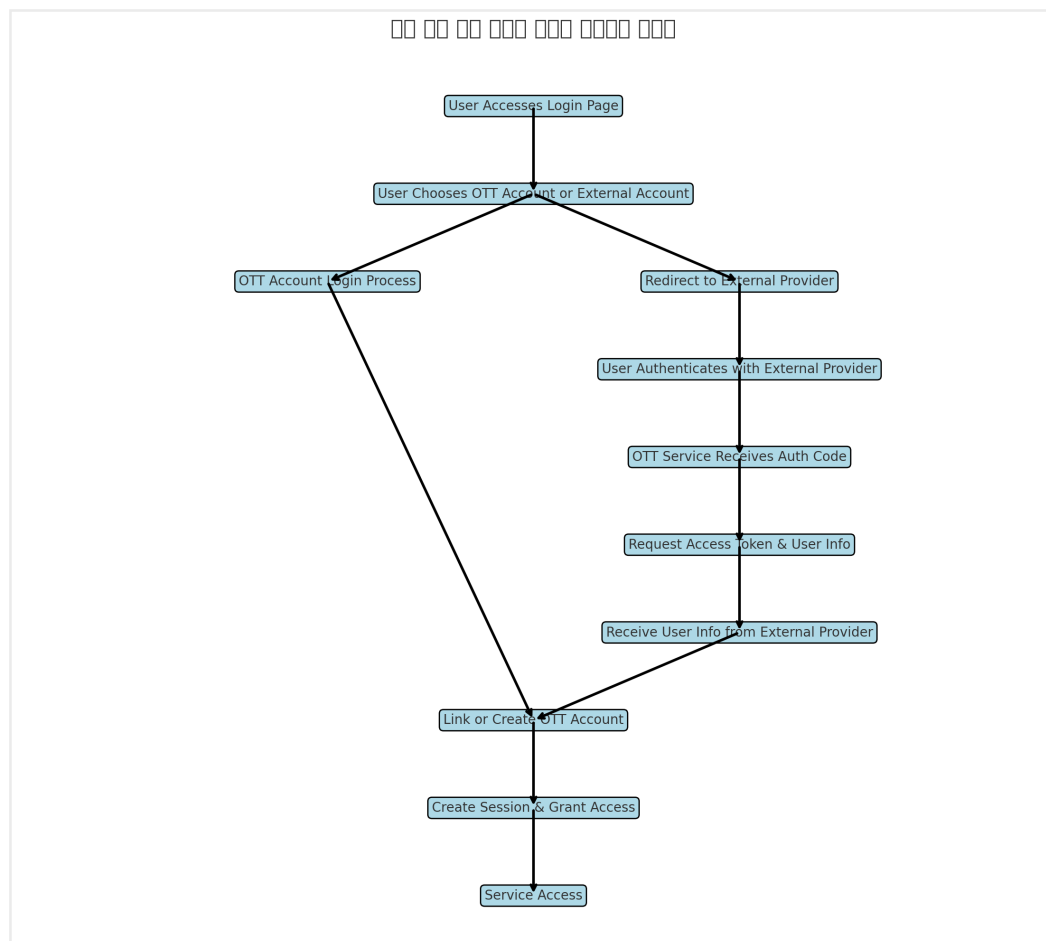
4.4. 세션 생성 및 서비스 제공

7. **세션 생성:** OTT 서비스가 세션을 생성하고, 사용자에게 로그인 성공 메시지를 반환합니다.

8. **서비스 접근 허용:** 사용자는 정상적으로 OTT 서비스에 접근하여 콘텐츠를 이용할 수 있습니다.

5. 프로세스 흐름도 시각화

이를 기반으로 외부 계정 연동 로그인 서비스의 전체 흐름도를 시각적으로 표현하겠습니다.



위 이미지는 외부 계정 연동 로그인 서비스의 프로세스 흐름도를 시각적으로 표현한 것입니다. 이 흐름도는 사용자가 OTT 서비스에 로그인할 때 외부 계정을 선택하고, 인증 절차를 거쳐 OTT 계정과 연동하는 전체 과정을 단계별로 보여줍니다.

주요 프로세스는 다음과 같습니다:

1. **User Accesses Login Page:** 사용자가 OTT 서비스의 로그인 페이지에

접근합니다.

2. **User Chooses OTT Account or External Account:** 사용자는 OTT 계정 또는 외부 계정(Google, Facebook 등)으로 로그인할지 선택합니다.
3. **OTT Account Login Process:** OTT 계정을 선택한 경우, 기존의 OTT 로그인 프로세스를 거칩니다.
4. **Redirect to External Provider:** 외부 계정을 선택한 경우, 해당 인증 제공자(Google, Facebook 등)로 리다이렉트됩니다.
5. **User Authenticates with External Provider:** 사용자가 외부 인증 제공자에서 로그인합니다.
6. **OTT Service Receives Auth Code:** OTT 서비스는 외부 인증 제공자로부터 인증 코드를 수신합니다.
7. **Request Access Token & User Info:** OTT 서비스는 인증 코드를 사용해 액세스 토큰과 사용자 정보를 요청합니다.
8. **Receive User Info from External Provider:** 외부 인증 제공자로부터 사용자 정보를 받습니다.
9. **Link or Create OTT Account:** 받은 정보를 바탕으로 OTT 계정을 연동하거나 새로 만듭니다.
10. **Create Session & Grant Access:** 세션을 생성하고 사용자가 서비스에 접근할 수 있도록 합니다.
11. **Service Access:** 사용자는 정상적으로 OTT 서비스에 접근합니다