결제 시스템

🔥 신용 카드 결제 처리를 지원하는 결제 시스템을 설계하자

▼ 1️⃣ 설계 범위 확정

- 하나의 통화만 지원
- 하루 100만 건의 거래가 이루어짐
- 매월 판매자에게 정산하는 절차 지원
- 내외부 서비스와 연동 중 상태 불일치에 대한 조정 작업 수행

기능 요구사항

- **대금 수신(pay-in)**: 결제 시스템이 판매자를 대신하여 고객으로부터 대금 수령
- **대금 정산(pay-out)**: 결제 시스템이 전 세계의 판매자에게 제품 판매 대금 송금

비기능 요구사항

- **신뢰성 / 내결함성**: 결제 실패에 대한 신중한 처리 필요
- **내부 서비스(결제/회계) - 외부 서비스(결제 서비스 제공업체) 간 조정 프로세스**: 시스템 간 결제 정보가 일치하는지 비동기적으로 확인

개략적 규모 추정

하루 100만 건 = 초당 10건 → 일반적인 데이터베이스로 처리 가능한 양

처리 대역폭 대신 결제 트랜잭션의 정확한 처리에 초점을 맞출 것

▼ 2️⃣ 개략적 설계

결제 흐름

대금 수신 흐름

- 결제 서비스

    - 사용자로부터 결제 이벤트를 수락하고 결제 프로세스 조율
    - 위험 확인을 통과한 결제만 처리 → 매우 복잡하므로 제3자 제공업체를 이용
- 결제 실행자
    - 결제 서비스 공급자를 통해 결제 주문 하나를 실행
    - 하나의 결제 이벤트에는 여러 결제 주문이 포함될 수 있음
- 결제 서비스 공급자
    - A 계정에서 B 계정으로 돈을 옮기는 역할을 담당
    - 구매자의 신용카드 계좌에서 돈을 인출하는 역할
- 카드 유형
    - 신용카드 업무를 처리하는 조직
- 원장
    - 결제 트랜잭션에 대한 금융 기록 (구매자—, 판매자++)
    - 총 수익을 계산하거나 향후 수익을 예측하는 등의 결제 후 분석에서 중요한 역할
- 지갑
    - 판매자의 계정 잔액 기록

대금 정산 흐름

대금 수신 흐름과 아주 유사하다.

대신 타사 정산 서비스를 사용하여 `전자상거래 웹사이트 은행 계좌 → 판매자 은행 계좌`로 돈을 이체한다.

일반적으로 결제 시스템은 대금 정산을 위해 외상 매입금 지급 서비스 제공업체를 이용한다.

결제 서비스 API

결제 이벤트 실행 (POST /v1/payments)

- **buyer_info**: 구매자 정보
- **checkout_id**: 결제 이벤트 ID
- **credit_card_info**: 암호화된 신용카드 정보 또는 결제 토큰
- **payment_orders**: 결제 주문 목록

- **seller_account**: 대금을 수령할 판매자
- **amount**: 해당 주문으로 전송돼야 할 대금
- **currency**: 주문에 사용된 통화 단위
- **payment_order_id**: 주문 ID

> ⚠️ **주의할 점**
>
> amount는 문자열이어야 한다!
>
> 직렬화/역직렬화에 사용하는 숫자 정밀도가 환경에 따라 다를 수 있기 때문. 의도치 않은 반올림 오류를 유발할 수 있다.
>
> **전송 및 저장 시 숫자는 문자열로 보관하고 표시하거나 계산할 때만 숫자로 변환한다.**

결제 주문 실행 상태 반환 (GET /v1/payments/{:id}

잘 알려진 일부 PSP의 API와 유사하다.

결제 서비스 데이터 모델

고려 사항

- 안정성이 검증되었는가?
- 모니터링 및 데이터 탐사에 필요한 도구가 풍부하게 지원되는가?
- DBA 채용 시장이 성숙한가?

🤔 어떻게 저장할 것인가

ACID 트랜잭션을 지원하는 전통적인 관계형 데이터베이스를 선호한다.

결제 주문 테이블

결제 주문의 실행 상태 저장

이름	자료형
payment_order_id	string PK
buyer_account	string

결제 이벤트 테이블

자세한 결제 이벤트 정보 저장

이름	자료형
checkout_id	string PK
buyer_info	string

이름	자료형	이름	자료형
amount	string	seller_info	string
currency	string	credit_card_info	카드 제공업체에 따라 다름
checkout_id	string FK	is_payment_done	boolean
payment_order_status	string		
ledger_updated	boolean		
wallet_updated	boolean		

- 한 번의 결제 행위는 하나의 결제 이벤트를 만든다.
- 하나의 결제 이벤트에는 여러 개의 결제 주문이 포함될 수 있다.
- 결제 처리 중에는 판매자의 은행 계좌는 필요하지 않다. (대금 수신, 정산)
- `payment_order_status`는 enum으로 관리한다.
  - NOT_STARTED: 초기값
  - EXECUTING: 결제 서비스가 결제 실행자에 주문을 전송한 상태
  - SUCESS: 결제 처리자가 성공 응답한 경우
  - FAIL: 결제 처리자가 실패 응답한 경우
- 프로세스
  - payment_order_status 값이 SUCESS로 업데이트
  - 결제 서비스는 지갑 서비스 호출
  - 지갑 서비스는 판매자 잔액 업데이트, wallet_updated 필드 TRUE로 업데이트
  - 결제 서비스는 원장 서비스 호출
  - 원장 데이터베이스의 ledger_updated 필드 TRUE로 업데이트
  - 동일한 checkout_id 아래의 모든 결제 주문이 성공 처리되면 결제 서비스는 결제 이벤트 테이블의 is_payment_done을 TRUE로 업데이트
  - 종결되지 않은 결제 주문을 모니터링하기 위해 주기적으로 실행되는 작업 마련 필요

복식부기(double-entry) 원장 시스템

복식부기?

- 정확한 기록을 남기는 결제 시스템의 필수 요소

- 모든 결제 거래를 두 개의 별도 원장 계좌에 같은 금액으로 기록
- 결제 주기 전반에 걸쳐 일관성을 보장

외부 결제 페이지

복잡한 규정이 있기 때문에 신용카드 정보를 내부에 저장하지 않는다 !

위젯, iframe, 결제 SDK에 포함된 사전에 구현된 페이지, ...

외부 결제 페이지가 직접 카드 정보를 수집한다.

▼ 3️⃣ 상세 설계

PSP 연동

외부 결제 서비스 작동 흐름

1. **사용자**가 클라이언트 브라우저에서 **결제 버튼** 클릭
2. **클라이언트**는 결제 주문 정보를 담아 **결제 서비스** 호출
3. **결제 서비스**는 결제 등록 요청을 **PSP**로 전송
4. **PSP**는 **결제 서비스**에 토큰을 반환
5. **결제 서비스**는 PSP가 제공하는 외부 결제 페이지를 호출하기 전 토큰을 저장
6. 토큰 저장 후 **클라이언트**는 **PSP가 제공하는 외부 결제 페이지**를 표시
    - 토큰과 리디렉션 URL이 필요
7. **사용자**는 카드 정보를 PSP 페이지에 입력 후 **결제 버튼**을 클릭
8. **PSP**가 결제 처리 후 결제 상태를 반환
9. **사용자**는 **리디렉션 URL**로 이동
    - 결제 상태가 URL에 추가됨
10. 비동기적으로 **PSP**는 웹훅을 통해 결제 상태와 함께 **결제 서비스**를 호출
    - 웹훅은 결제 시스템 측에서 PSP를 처음 설정할 때 등록한 URL
11. **결제 시스템**이 웹훅을 통해 결제 이벤트를 다시 수신하면 결제 상태를 추출하여 **결제 주문 데이터베이스** 테이블의 `payment_order_status` 필드를 최신 상태로 업데이트

조정

외부 결제 서비스 작동 중 장애가 발생한다면 어떻게 처리할까? - 비동기 통신에서 정확성을 보장하는 방법

1. 매일 밤 고객에게 정산 파일을 보낸다.
    - 정산 파일: 은행 계좌의 잔액과 하루 동안 해당 계좌에서 발생한 모든 거래 내역
2. 조정 시스템은 정산 파일의 세부 정보를 읽어 원장 시스템과 비교한다.
3. 조정 중 발견된 차이는 일반적으로 수동으로 고친다.

발생 가능한 불일치 문제

- **어떤 유형의 문제인지 안다 && 문제 해결 절차를 자동화할 수 있다**
    - 자동화 프로그램을 작성하는 것이 효율적
- **어떤 유형의 문제인지 안다 && 자동화할 수 없다**
    - 이 때 프로그램을 작성하는 것은 비용이 많이 든다.
    - 작업 대기열에 놓고 수동으로 수정하자.
- **분류할 수 없는 유형의 문제**
    - 특별 작업 대기열에 놓고 조사하자.

결제 지연 처리

결제 요청이 오래 걸리는 경우 ?

- PSP가 해당 결제 요청의 위험성이 높다고 보고 검토를 요구하는 경우
- 신용카드사가 구매 확인 용도로 카드 소유자의 추가 정보를 요청하는 경우

PSP는 어떻게 처리하는가 ?

- 결제가 대기 상태임을 알리는 정보를 클라이언트에 반환
- 클라이언트는 이를 사용자에게 표시함
- 클라이언트는 고객이 현재 결제 상태를 확인할 수 있는 페이지를 제공함
- PSP는 대신 대기 중인 결제 진행 상황을 추적하여 상태 변경 시 웹훅을 통해 결제 서비스에 알림

  웹훅 대신 결제 서비스가 폴링 방식으로 주기적 확인도 가능 !
- 결제 서비스는 내부 정보를 업데이트하고 고객에게 알림

내부 서비스 간 통신

동기식 통신

- **성능 저하 :** 서비스 중 하나에 발생한 문제가 전체 성능에 영향을 끼침
- **장애 격리 곤란 :** PSP 등의 서비스에 장애가 발생하면 더 이상 응답받지 못함
- **높은 결합도 :** 요청 발신자가 수신자를 알아야 함
- **낮은 확장성 :** 큐를 버퍼로 사용하지 않으면 갑작스러운 트래픽 증가에 대응하기 어려움

비동기식 통신

- 단일 수신자
  - 각 요청은 하나의 수신자/서비스가 처리
  - 일반적으로 공유 메시지 큐 사용
    - 처리된 메시지는 큐에서 바로 제거
- 다중 수신자
  - 각 요청은 여러 수신자/서버가 처리
  - 카프카 추천 !
    - 수신한 메시지도 바로 사라지지 않음
    - 하나의 요청이 여러 용도로 쓰일 수 있을 때 적절함
      - 결제 이벤트, 결제 시스템, 분석, 결제 청구, ...

→ 동기식 통신은 설계는 쉬우나 서비스 간 자율성을 높이기 어려움

→ 비동기식 통신은 설계의 단순성과 데이터 일관성을 확장성과 장애 감내 능력과 맞바꾼 결과

→ **비즈니스 로직이 복잡하고 타 서비스 의존성이 높다면 ?** 비동기 통신이 낫다 ~

결제 실패 처리

결제 상태 추적

실패가 일어날 때마다 결재 거래의 현재 상태를 파악하고 재시도 또는 환불이 필요한지 여부를 결정

데이터 추가만 가능한 데이터베이스 테이블에 보관

재시도 큐 및 실패 메시지 큐

- **재시도 큐**: 일시적 오류 같은 재시도 가능 오류
- **실패 메시지 큐**: 반복적으로 처리에 실패한 메시지, 디버깅하고 격리하여 원인 파악

동작 방식

1. 재시도가 가능한지 확인
2. 재시도 큐에 쌓인 이벤트를 결제 시스템이 읽어서 처리
3. 임계값을 넘는 재시도 이벤트는 실패 메시지 큐에 추가

정확히 한 번 전달

최소 한 번은 실행되게 하고 최대 한 번 실행하기 == 정확히 한 번

재시도 전략

- **즉시 재시도**
- **고정 간격** : 일정 시간 기다리기
- **증분 간격** : 점진적으로 대기 시간 늘리기
- **지수적 백오프** : 직전 재시도 대비 두 배씩 늘려서 기다리기
    - 네트워크 문제가 단시간 내 해결되지 않을 것 같다면 권장
- **취소 요청 철회** : 성공 가능성이 낮거나 실패가 영구적인 경우

→ 모든 상황에 맞는 해결책은 없다.

멱등성

재시도는 이중 결제 가능성이 있다. 최대 한 번 실행 하자 (= 여러 번 실행해도 그대로 보존되는 특성)

동작 원리

1. 클라이언트가 멱등 키 생성
    - 일정 시간이 지나면 만료되는 고유한 값
2. HTTP 헤더에 `<멱등 키: 값>`의 형태로 멱등 키를 추가한다.
    - 결제 요청의 멱등성을 보장할 수 있음

시나리오 1) 고객이 결제 버튼을 두 번 클릭하는 경우

- 두 번째 들어온 요청은 멱등 키를 이전에 받은 적이 있으므로 재시도로 처리한다.
  - 이전 결제 요청의 가장 최근 상태를 반환
- 동일한 멱등 키로 동시에 많은 요청을 받으면 하나만 처리하고 `429 Too Many Requests` 상태 코드를 반환한다.
- 데이터베이스의 **고유 키 제약 조건**을 활용해서 멱등성을 지원할 수 있다.

시나리오 2) PSP가 결제 처리를 했지만 네트워크 오류로 결과 전달이 안돼서 사용자가 다시 결제하는 경우

- PSP에 비중복 난수를 전송하고 PSP는 해당 난수에 대응되는 토큰을 반환한다.
  - 결제 주문을 유일하게 식별하는 도구
- 사용자가 결제 버튼을 다시 눌러도 결제 주문이 같으니 PSP로 전송되는 토큰도 같으므로 이중 결제로 판단한다.

일관성

분산 환경에서는 서비스 간 통신 실패로 데이터 불일치가 발생할 수 있다.

내부 서비스 간 데이터 일관성 유지

- 요청이 정확히 한 번 처리되도록 보장하자

내부 서비스 - 외부 서비스 간 데이터 일관성 유지

- 멱등성과 조정 프로세스 사용
- 외부 서비스가 멱등성을 지원하면 결제를 재시도할 때 같은 멱등 키를 사용하면 됨
- 외부 서비스가 항상 옳다고 가정할 수 없으므로 조정 절차 필요

주 DB - 부 DB 간 데이터 일관성 유지

- 주 데이터베이스에서만 읽기/쓰기 연산 처리
  - 규모 확장성이 떨어지고 자원이 낭비됨
- 모든 사본이 항상 동기화되도록 처리

보안

- **요청/응답 도청**: HTTPS 사용
- **데이터 변조**: 암호화 및 무결성 강화 모니터링

- **중간자 공격**: 인증서 고정과 함께 SSL 사용
- **데이터 손실**: 데이터베이스 복제 및 스냅샷 생성
- **DDoS**: 처리율 제한 및 방화벽
- **카드 도난**: 토큰화 - 실제 카드 번호 대신 토큰을 저장하고 결제에 사용함
- **PCI 규정 준수**: 브랜드 신용 카드를 처리하는 조직을 위한 정보 보안 표준
- **사기**: 주소 확인, 카드 확인번호, 사용자 행동 분석 등

▼ 4️⃣ 마무리

- 대금 수신/정산 흐름
- 재시도, 멱등성, 일관성
- 결제 오류 처리
- 보안

더 논의할만한 주제들

- 모니터링
- 경보
- 디버깅 도구
- 환율
- 지역
- 현금 결제
- 구글/애플 페이 연동
- ...