1️⃣

결제 시스템

문제 이해 및 설계 범위 확정

기능 요구사항

- 대금 수신 흐름
  - 결제 시스템이 판매자를 대신하여 고객으로부터 대금을 수령함
- 대금 정산 흐름
  - 결제 시스템이 . 전세계의 판매자에게 제품 판매 대금을 송금함

비기능 요구사항

- 신뢰성 및 내결함성
  - 결제 실패는 신중하게 처리해야 함
- 내부 서비스와 외부 서비스 간의 조정 프로세스
  - 시스템 간의 결제 정보가 일치하는지 비동기적으로 확인함
  - 내부 서비스: 결제 시스템, 회계 시스템
  - 외부 서비스: 결제 서비스 제공업체

개략적인 규모 추정

- 하루에 100만 건의 트랙잭션을 처리해야 함
- 100만 건의 트랜잭션 / 10^5 = 초당 10건의 TPS
- 일반적인 데이터베이스로 문제 없이 처리 가능함
- 처리 대역폭 대신 결제 트랜잭션의 정확한 처리에 초점을 맞춰 면접 진행하기

개략적 설계안 제시 및 동의 구하기

대금 수신 흐름

결제 서비스

- 사용자로부터 결제 이벤트를 수락하고 결제 프로세스를 조율함
- 일반적으로 AML/CFT와 같은 규정을 준수하는지, 자금 세탁이나 테러 자금 조달과 같은 범죄 행위의 증거가 있는지 평가하는 위험 점검을 가장 먼저 함
    - 복잡하고 고도로 전문화되어 있기 때문에 제3자 제공업체를 이용함

결제 실행자

- 결제 서비스 공급자(PSP)를 통해 결제 주문 하나를 실행함
- 하나의 결제 이벤트에는 여러 결제 주문이 포함될 수 있음

결제 서비스 공급자

- Payment Service Provider
- A 계정에서 B 계정으로 돈을 옮기는 역할을 담당
    - 본 설계안에서는 구매자의 신용 카드 계좌에서 돈을 인출하는 역할

카드 유형

- 카드사: 신용 카드 업무를 처리하는 조직
- 카드 생태계는 매우 복잡함

원장

- 결제 트랜잭션에 대한 금융 기록
- 결제 후 분석에서 매우 중요한 역할
    - 전자상거래 웹사이트의 총 수익을 계산하거나 향후 수익을 예측

지갑

- 판매자의 계정 잔액을 기록함
- 특정 사용자가 결제한 총 금액을 기록할 수도 있음

결제 흐름

1. 사용자가 “주문하기” 버튼을 클릭하면 결제 이벤트가 생성되어 결제 서비스로 전송
2. 결제 서비스는 결제 이벤트를 데이터베이스에 저장
3. 단일 결제 이벤트에 여러 결제 주문이 포함될 수 있음
    - 한 번 결제로 여러 판매자의 제품을 처리하는 경우
    - 전자상거래 웹사이트에서 한 결제를 여러 결제 주문으로 분할하는 경우
    - 결제 서비스는 결제 주문마다 결제 실행자를 호출함
4. 결제 실행자는 결제 주문을 데이터베이스에 저장
5. 결제 실행자가 외부 PSP를 호출하여 신용 카드 결제를 처리
6. 결제 실행자가 결제를 성공적으로 처리하면 결제 서비스는 지갑을 갱신하여 특정 판매자의 잔고를 기록
7. 지갑 서버는 갱신된 잔고 정보를 데이터베이스에 저장
8. 지갑 서비스가 판매자 잔고를 성공적으로 갱신하면 결제 서비스는 원장을 호출
9. 원장 서비스는 새 원장 정보를 데이터베이스에 추가함

결제 서비스 API

POST	/v1/payments	결제 이벤트를 실행함
		하나의 결제 이벤트에는 여러 결제 주문이 포함될 수 있음
			요청 매개변수
			{
			”buyer_info”:”구매자 정보”,
			”checkout_id”:”해당 결제 이벤트를 식별하는 고유 ID”,
			”credit_card_info”:”카드 정보/PSP마다 다른 값”,
			”payment_orders”: {
			// 결제 주문 목록
			”seller_account”:”대금을 수령할 판매자”,
			”amount”:”금액”,
			”currency”:”통화 단위”,
			”payment_order_id”,”주문을 식별하는 고유 ID”

			} }
GET	/v1/payments/{:id}	payment_order_id가 가리키는 단일 결제 주문의 실행 상태를 반환	

- 금액(amount) 필드가 문자열인 이유
  - 프로토콜, 소프트웨어, 하드웨어에 따라 직렬화/역직렬화에 사용하는 숫자 정밀도가 다를 수 있는데 이러한 차이가 의도치 않은 반올림 오류를 유발
  - 숫자가 매우 클 수도, 작을 수도 있음

결제 서비스 데이터 모델

- 결제 시스템용 저장소 솔루션을 고를 때 중점을 둬야 할 이슈
  - 안정성이 검증되었는가?
    - 다른 대형 금융 회사에서 수년동안 긍정적인 피드백을 받으면 사용되었는지
  - 모니터링 및 데이터 탐사에 필요한 도구가 풍부하게 지원되는가?
  - 데이터베이스 관리자 채용 시장이 성숙했는가?
    - DBA를 쉽게 채용할 수 있는지
- 일반적으로 NoSQL/NewSQL보다는 ACID 트랜잭션을 지원하는 전통적인 관계형 데이터베이스를 선호

결제 이벤트	
checkout_id	string PK
buyer_info	string
seller_info	string
credit_card_info	카드 제공업체에 따라 다름
is_payment_done	boolean

결제 주문	
payment_order_id	string PK
buyer_account	string
amount	string
currency	string FK
checkout_id	string
payment_order_status	string
ledger_updated	boolean
wallet_updated	boolean

1. payment_order_status가 SUCCESS로 변경되면 결제 서비스는 지갑 서비스를 호출하여 판매자 잔액을 업데이트하고 wallet_updated 필드 값은 TRUE로 변경
2. 위 절차가 끝나면 원장 서비스를 호출하여 ledger_updated 필드 값을 TRUE로 변경
3. 동일한 checkout_id 아래의 모든 결제 주문이 성공적으로 처리가 되면 is_payment_done 필드 값을 TRUE로 변경
4. 일반적으로 종결되지 않은 결제 주문을 모니터링 하기 위한 스케쥴링 작업을 마련해 둠

- chekout_id는 외래키
  - 한번의 결제 행위는 하나의 결제 이벤트를 만듦
  - 하나의 결제 이벤트에는 여러 개의 결제 주문이 포함될 수 있음
- 대금 수신
  - 구매자의 신용 카드에서 금액을 공제하기 위해 타사 PSP를 호출하면 판매자 대신 전자상거래 웹사이트의 은행 계좌에 이체가 이루어짐

  ⇒ 사용자의 결제를 처리하는 중에는 판매자의 은행 계좌가 아닌 구매자의 카드 정보만 필요함
- payment_order_status
  - 결제 주문의 실행 상태를 유지하는 열거 자료형
  - NOT_STARTED
  - EXCUTING
  - SUCCES
  - FAILED

복식부기 원장 시스템

- 원장 시스템에는 복식부기라는 아주 중요한 설계 원칙이 존재함
- 모든 결제 시스템에 필수 요소
- 정확한 기록을 남기는 데 핵심적인 역할
- 모든 결제 거래를 두 개의 별도 원장 계좌에 같은 금액으로 기록함
- 한 계좌에서는 차감이, 다른 계좌에서는 입금이 이루어짐
- 모든 거래 항목의 합은 0이어야 함
- 자금의 흐름을 시작부터 끝까지 추적할 수 있음

- 결제 주기 전반에 걸쳐 일관성을 보장

외부 결제 페이지

- 대부분의 기업은 신용 카드 정보를 내부에 저장하지 않음
  - 미국의 PCI DSS 같은 복잡한 규정을 준수해야 하기. 때문
- PSP에서 제공하는 외부 신용 카드 페이지를 사용
- 중요한 것은 PSP가 제공하는 외부 결제 페이지가 직접 고객 카드 정보를 수집한다는 것

대금 정산 흐름

- 대금 수신 흐름과 거의 유사함
- PSP를 사용하여 구매자의 신용 카드에서 전자상거래 웹사이트 은행 계좌로 돈을 이체하는 대신
- 타사 정산 서비스를 사용하여 전자 상거래 웹사이트 은행 계좌에서 판매자 은행 계좌로 돈을 이체한다는 점만 다름

상세 설계

PSP 연동

PSP와 연동하는 방법

- 회사가 민감한 결제 정보를 안전하게 저장할 수 있는 경우
  - 결제 웹페이지를 개발하고 민감한 결제 정보를 수집
  - PSP는 은행 연결, 다양한 카드 유형을 지원하는 역할
- 회사가 민감한 결제 정보를 저장하지 않는 경우
  - PSP는 카드 결제 세부 정보를 수집하여 PSP에 안전하게 저장할 수 있도록 외부 결제 페이지를 제공
  - 대부분의 기업이 이 방법을 선택

외부 결제 페이지 작동 방식

1. 사용자가 클라이언트 브라우저에서 결제를 요청하면 클라이언트는 결제 주문 정보를 담아 결제 서비스 호출
2. 결제 주문 정보를 수신한 결제 서비스는 결제 등록 요청을 PSP로 전송
    - 결제 금액, 통화, 결제 요청 만료일, 리디렉션 URL
    - 결제 주문이 정확이 한 번만 등록될 수 있도록 비중복 난수 UUID 필드를 둠
3. PSP는 결제 서비스에 토큰을 반환
    - 토큰은 등록된 결제 요청을 유일하게 식별하는 PSP가 발급한 UUID
    - 토큰을 사용하여 결제 등록 및 실행 상태 등을 확인 가능
4. 결제 서비스는 PSP가 제공하는 외부 결제 페이지를 호출하기 전에 토큰을 데이터베이스에 저장
5. 클라이언트는 PSP가 제공하는 외부 결제 페이지를 표시
    - 모바일 애플리케이션은 일반적으로 SDK를 연동
    - 웹 애플리케이션에서 사용하는 자바스크립트 라이브러리에는 결제 UI를 표시, 민감한 결제 정보 수집, 결제를 완료하는 등의 작업을 위해 PSP를 직접 호출하는 로직이 포함되어 있음
    - 민감한 정보는 우리 시스템으로 절대 넘어오지 않음
    - 필요한 정보
        - PSP에서 받은 토큰
        - 리디렉션 URL
6. 사용자는 신용 카드 번호, 쇼유자 이름, 카드 유효기간 등의 결제 세부 정보를 PSP의 웹 페이지에 입력한 다음 결제 요청
7. PSP가 결제 처리를 시작하고, 결제 상태를 반환
8. 사용자는 리디렉션 URL이 가리키는 웹 페이지로 보내짐
9. 비동기적으로 PSP는 웹훅을 통해 결제 상태와 함께 결제 서비스를 호출
    - 웹훅: 결제 시스템 측에서 PSP를 처음 설정할 때 등록한 URL
    - 결제 시스템이 웹훅을 통해 결제 이벤트를 다시 수신하면 결제 상태를 추출하여 결제 주문 데이터베이스 테이블의 payment_order_status 필드를 최신 상태로 업데이트

네트워크 등의 이유로 중간에 실패하는 경우, 체계적으로 처리할 수 있는 방법

⇒ 조정

조정

- 정확성을 보장하는 방법
- 관련 서비스 간의 상태를 주기적으로 비교하여 일치하는지 확인
- 일반적으로 결제 시스템의 마지막 방어선
- 매일 밤 PSP나 은행은 고객에게 은행 계좌의 잔액과 하루 동안 해당 계좌에서 발생한 모든 거래 내역이 기재되어 있는 정산 파일을 보냄
- 조정 시스템은 정산 파일의 세부 정보를 읽어 원장 시스템과 비교함
- 결제 시스템의 내부 일관성을 확인할 때도 사용됨
    - 원장과 지갑의 상태가 같은지
- 조정 중에 발견된 차이는 일반적으로 재무팀에 의뢰하여 수동으로 수정

발생 가능한 불일치 문제와 해결 방안

- 어떤 유형의 문제인지 알고 있음 + 문제 해결 절차를 자동화할 수 있음
    - 엔지니어는 발생한 불일치 문제의 분류와 조정 작업을 모두 자동화 가능
- 어떤 유형의 문제인지 알 수 있음 + 문제 해결 절차를 자동화할 수 없음
    - 발생한 불일치 문제는 작업 대기열에 넣고 재무팀에서 수동으로 수정
- 분류할 수 없는 유형의 문제
    - 불일치가 어떻게 발생하였는지 알지 못하는 경우
    - 특별 작업 대기열에 넣고 재무팀에서 조사하도록 함

결제 지연 처리

결제 처리가 지연되는 경우

- PSP가 해당 결제 요청의 위험성이 높다고 보고 담당자가 검토를 요구하는 경우

- 신용 카드사가 구매 확인 용도록 카드 소유자의 추가 정보를 요청하는 3D 보안 인증 같은 추가 보호 장치를 요구하는 경우

결제 처리 지연을 처리하는 방법

- PSP는 결제가 대기 상태임을 알리는 상태 정보를 클라이언트에 반환하고, 클라이언트는 이를 사용자에게 표시함과 동시에 고객이 현재 결제 상태를 확인할 수 있는 페이지 제공
- PSP는 우리 회사를 대신하여 대기 중인 결제의 진행 상황을 추적하고, 상태가 바뀌면 PSP에 등록된 웹훅을 통해 결제 서비스에 알림
    - 결제 서비스에 결제 상태 변경을 알리는 대신, 결제 서비스로 하여금 대기 중인 결제 요청의 상태를 주기적으로 확인하도록 하기도 함

내부 서비스 간 통신

- 동기식 통신
    - 소규모 시스템에서는 잘 작동하지만 규모가 커지면 단점이 분명해짐
    - 한 요청에 응답을 만드는 처리 주기는 관련된 서비스가 많을 수록 질어짐
    - 성능 저하
        - 요청 처리에 관계된 서비스 가운데 하나에 발생한 성능 문제가 전체 시스템의 성능에 영향을 끼침
    - 장애 격리 곤란
        - PSP 등의 서비스에 장애가 발생하면 클라이언트는 더 이상 응답을 받지 못함
    - 높은 결합도
        - 요청 발신자는 수신자를 알아야만 함
    - 낮은 확장성
        - 큐를 버퍼로 사용하지 않고서는 갑작스러운 트래픽 증가에 대응할 수 있도록 시스템을 확장하기 어려움
- 비동기 통신
    - 단일 수신자
        - 각 요청은 하나의 수진자 또는 서비스가 처리

        - 일반적으로 공유 메시지 큐를 사용하여 구현
        - 큐에는 복수에 구독자가 있을 수 있으나 처리된 큐에서 바로 제거됨
    - 다중 수신자
        - 각 요청은 여러 수신자 또는 서버가 처리
        - 카프카는 이런 시나리오를 잘 처리함
            - 소비자가 수신한 메시지는 카프카에서 바로 사라지지 않음
            - 동일한 메시지를 여러 서비스가 받아 처리할 수 있음
        - 결제 시스템 구현에 적합
            - 하나의 요청이 푸시 알림 전송, 재무 보고 업데이트, 분석 결과 업데이트 등의 다양한 용도에 쓰일 수 있음

결제 실패 처리

결제 상태 추적

- 결제 주기의 모든 단계에서 결제 상태를 정확하게 유지하는 것은 매우 중요
- 실패가 일어날 때마다 결제 거래의 현재 상태를 파악하고 재시도 또는 환불이 필요한지 여부를 결정
- 결제 상태는 데이터 추가만 가능한 데이터베이스 테이블에 저장

재시도 큐 및 실패 메시지 큐

- 재시도 큐
    - 일시적 오류 같은 재시도 가능 오류
- 실패 메시지 큐
    - 반복적으로 처리에 실패한 메시지
    - 문제가 있는 메시지를 디버깅하고 격리하여 성공적으로 처리되지 않은 이유를 파악하기 위한 검사에 유용

1. 재시도가 가능한지 확인
    a. 재시도 가능 → 재시도 큐

    b. 재시도 불가능 → 오류 내역을 데이터베이스에 저장
2. 결제 시스템은 재시도 큐에 쌓인 이벤트를 읽어 실패한 결제 재시도
3. 결제 거래가 다시 실패하는 경우
    a. 재시도 횟수가 임계값 이내 → 다시 재시도 큐
    b. 재시도 횟수가 임계값 이상 → 실패 메시지 큐

'정확히 한 번' 전달

- 결제 시스템에 발생 가능한 가장 심각한 문제 중 하나는 고객에게 이중으로 청구하는 것
- 결제 주문이 정확히 한 번만 실행되도록 설계하는 것이 중요
- 정확히 한 번만 실행되는 요건
    - 최소 한 번
    - 최대 한 번

재시도

- 재시도 메커니즘을 활용하면 최소 한 번은 실행되도록 보장 가능
- 얼마나 간격을 두고 재시도할지 정하는 것이 중요
    - 즉시 재시도
    - 고정 간격
        - 재시도 전에 일정 시간 기다림
    - 증분 간격
        - 재시도 전에 기다리는 시간을 특정한 양만큼 점진적으로 늘려가는 방법
    - 지수적 백오프
        - 재시도 전에 기다리는 시간을 직전 재시도 대비 두 배씩 늘려가는 방법
    - 취소
        - 요청을 철회
        - 실패가 영구적이거나 재시도를 하더라도 성공 가능성이 낮은 경우
- 일반적으로 적용 가능한 지침

    - 네트워크 문제가 단시간 내에 해결될 것 같지 않다면 지수적 백오프를 사용하라
- 에러 코드를 반환할 때는 Retry-After 헤더를 같이 붙여 보내는 것이 바람직
- 재시도 시 발생할 수 잇는 잠재적 문제는 이중 결제
    - 결제 시스템이 외부 결제 페이지를 통해 PSP와 연동하는 환경에서 클라이언트가 결제 버튼을 중복 클릭
    - PSP가 결제를 성공적으로 처리하였으나 네트워크 오류로 응답이 결제 시스템에 도달하지 못하여 사용자가 결제 버튼을 다시 클릭하거나 클라이언트가 결제를 다시 시도
    - 이중 결제를 장지하려면 결제는 '최대 한 번'만 이루어져야 함

      ⇒ 멱등성

멱등성

- 최대 한 번 실행을 보장하기 위한 핵심 개념
- API 관점에서 클라이언트가 같은 API 호출을 여러 번 반복해도 항상 동일한 결과가 나옴
- 클라이언트와 서버 간의 통신을 위해서는 일반적으로 클라이언트가 생성하고 일정 시간이 지나면 만료되는 고유한 값을 멱등키로 사용
- 결제 요청의 멱등성을 보장하기 위해서는 HTTP 헤더에 <멱등키: 값>의 형태로 멱등키 추가
- 시나리오
    - 결제 시스템이 외부 결제 페이지를 통해 PSP와 연동하는 환경에서 클라이언트가 결제 버튼을 중복 클릭
        - 사용자가 결제를 클릭하면 멱등키가 HTTP 요청의 일부로 결제 시스템에 전송
        - 일반적으로 전자상거래 웹사이트에서 명등키 == 결제가 이루어지기 직전의 장바구니 ID
        - 결제 시스템은 두 번째 요청을 처리하려고 할 때 이미 이전에 받은 멱등키라면 이전 결제 요청의 가장 최근 상태를 반환
        - 동일한 멱등키로 동시에 많은 요청을 받으면 결제 서비스는 그 가운데 하나만 처리하고 나머지는 Too Many Requests 상태 코드를 반환
        - 데이터베이스의 고유 키 제약 조건을 활용하는 것도 한 가지 방안

1. 결제 시스템은 결제 요청을 받으면 데이터베이스 테이블에 새 레코드를 넣으려 시도
2. 새 레코드 추가에 성공했다는 것은 이전에 처리한 적이 없는 결제 요청이라는 뜻
3. 새 레코드 추가에 실패했다는 것은 이전에 받은 적이 있는 결제 요청이라는 뜻 == 중복 요청

- PSP가 결제를 성공적으로 처리하였으나 네트워크 오류로 응답이 결제 시스템에 도달하지 못하여 사용자가 결제 버튼을 다시 클릭하거나 클라이언트가 결제를 다시 시도
  - 사용자가 결제 버튼을 다시 누른다고 해도 결제 주문이 같기 때문에 PSP에 전송되는 토큰도 같음
  - 이중 결제로 판단

일관성

- 결제 실행 과정에서 상태 정보를 유지 관리하는 여러 서비스가 호출됨
  - 결제 서비스는 비중복 난수, 토큰, 결제 주문, 실행 상태 등의 결제 관련 데이터를 유지 관리함
  - 원장은 모든 회계 데이터를 보관
  - 지갑은 판매자의 계정 잔액을 유지함
  - PSP는 결제 실행 상태를 유지함
  - 데이터는 안정성을 높이기 위해 여러 데이터베이스 사본에 복제 될 수 있음
- 분산 환경에서는 서비스 간 통신 실패로 데이터 불일치가 발생할 수 있음
- 내부 서비스 간의 데이터 일관성을 유지하려면 요청이 '정확히 한 번' 처리되도록 보장하는 것이 중요
- 내부 서비스와 외부 서비스 간의 데이터 일관성을 유지하기 위해 멱등성과 조정 프로세스를 활용
  - 외부 서비스가 멱등성을 지원하는 경우에는 결제를 재시도할 때 같은 멱등키를 사용해야 함
  - 외부 서비스가 멱등성을 지원하더라고 항상 옳다고 가정할 수 없기 때문에 조정 절차를 생략할 수 없음

- 데이터를 다중화하는 경우에는 복제 지연으로 데이터 불일치가 발생할 수 있음
  - 주 데이터베이스에서만 읽기와 쓰기 연산을 처리
    - 설정하기는 쉽지만 규모 확장성이 떨어짐
    - 사본은 데이터 안정성 보장에만 활용되고 트래픽은 처리하지 않아 자원이 낭비됨
  - 모든 사본이 항상 동기화되도록 함

보안

- 요청/응답 도청
  - HTTPS 사용
- 데이터 변조
  - 암호화 및 무결성 강화 모니터링
- 중간자 공격
  - 인증서 고정과 함께 SSL 사용
- 데이터 손실
  - 여러 지역에 걸쳐 데이터베이스 복제 및 스냅숏 생성
- DDoS
  - 처리율 제한 및 방화벽
- 카드 도난
  - 실제 카드 번호를 사용하는 대신 토큰을 저장하고 결제에 사용
- PCI 규정 준수
  - 브랜드 신용 카드를 처리하는 조직을 위한 정보 보안 표준
- 사기
  - 주소 확인
  - 카드 확인번호
  - 사용자 행동 분석

마무리

추가적으로 언급해도 좋을 이슈

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