Portfolio

PROFICIENCY

• SpringBoot 및 Java 를 활용하여 애플리케이션 개발 및 구현이 가능합니다.
• React, Next.js, Typescript, Tailwind CSS 등의 기술을 사용한 프론트엔드 개발이 가능합니다.
• AWS, Docker, Nginx 및 Github Actions 등을 이용하여 CI/CD 구축 경험이 있습니다.

(주)다날

해외통합결제 자동취소 서비스

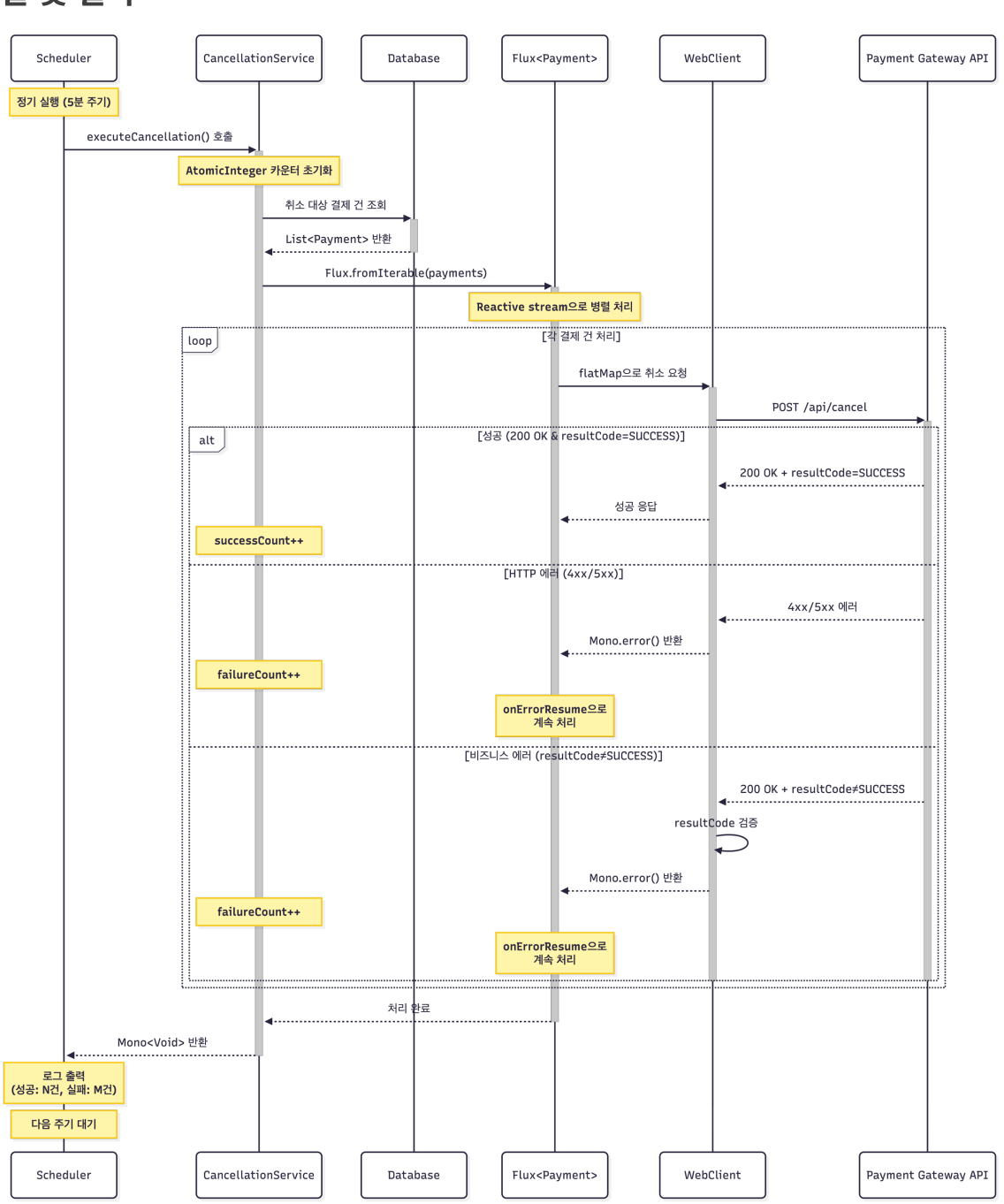
배경 지식, 문제

- 가맹점 테스트 결제 후 취소 누락 시 매입 처리되어 해외사업팀이 복잡한 결재/공문 등으로 수동 취소 해야 하는 위험 존재
- 위험 부담으로 운영계에서 테스트 가맹점 ID 미제공, 검증계→운영계 이중 테스트 강제로 비효율 발생
- 자동취소 시스템 부재로 테스트 결제 건 매입 처리 리스크 상존 및 해외사업팀 수동 취소 업무 부담
- GW에서 특정 해외 결제수단의 거래내역을 적재하지 않고 있어 취소 대상 누락 가능성

기술적 검토

- **동기 처리 방식** (RestTemplate + 순차 처리)
  - 장점: 구현이 단순하고 디버깅 용이
  - 단점: 외부 API 응답 지연 시 스레드 블로킹으로 처리량 급감, 대량 요청 시 스레드 풀 고갈 위험
- **비동기 처리 방식** (WebFlux + Reactive Streams)
  - 장점: 논블로킹 I/O로 적은 스레드로 높은 동시성 처리, 외부 API 지연에도 스레드 효율적 사용
  - 단점: 설계 복잡도 증가, 디버깅 난이도 상승

해결 및 결과



추가적인 내용은 2페이지를 참고해주세요.

Portfolio

비동기 병렬 취소 처리 구현
- WebClient + Reactor의 Flux.flatMap을 활용하여 다수의 GW API 취소 요청을 비동기 병렬 전송

개별 실패 격리 메커니즘 구현
- onErrorResume을 통해 개별 취소 요청 실패 시에도 전체 배치 중단 없이 나머지 처리 계속 진행
- 성공/실패 건수 집계 및 상세 로그 기록

다중 데이터 소스 통합 구조 설계
- 추상 클래스로 공통 속성 정의 및 결제수단별 확장 클래스 구현 (유사한 DB 스키마 활용)
- Repository 계층에서 GW 거래내역 및 특정 결제수단 DB를 모두 조회하여 누락 없이 취소 처리

운영 안정성 강화
- 5분 주기 스케줄러로 미처리 취소 건 자동 조회 및 처리
- 네이버 웍스를 통한 실시간 장애 알림 체계로 즉각적인 오류 대응 가능

회고

- 스케줄러 기반 배치 시스템으로 MVC + WebClient만으로 동일한 비동기 처리 가능
- JPA(JDBC) 블로킹으로 리액티브 스택의 이점 미활용, R2DBC 도입이 적절했다고 판단
- 이후 기술 도입 시 시스템 병목 지점을 먼저 파악하고 선택하는 것을 원칙으로 삼게 됨

성과

- 자동취소 시스템 도입으로 테스트 결제 건의 매입 처리 위험 제거 및 운영계에서 안전한 테스트 가맹점 ID 제공 가능
- 검증계-운영계 이중 테스트 프로세스 제거로 가맹점 및 내부 운영 부담 감소, 신규 가맹점 온보딩 시간 단축
- 해외사업팀의 수동 취소 업무(복잡한 결재/공문 프로세스) 완전 제거
- 비동기 병렬 처리 및 개별 실패 격리로 높은 동시성 처리 및 시스템 안정성 확보
- GW 미적재 거래내역 문제 해결 및 확장 가능한 아키텍처로 신규 결제수단 추가 용이

부분취소 게이트웨이 개선

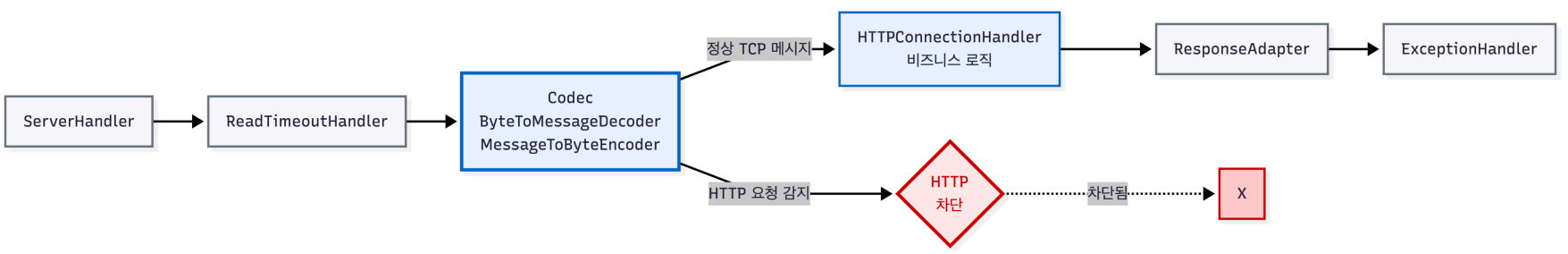
배경 지식, 문제

- Netty 기반 TCP 소켓 서버에서 원인 불명의 타임아웃 에러 빈발
- HTTP 요청(curl, 브라우저 등)이 TCP 소켓 전용 서버로 유입되는 상황 발견
- 커스텀 디코더가 HTTP 헤더를 패킷 길이로 오해석하여 비정상적인 값(1,195,725,856 바이트) 기대
- 디코딩 실패로 인한 타임아웃 발생 및 불필요한 에러 알림 발송

원인 분석

- **HTTP 요청의 프로토콜 불일치**
  - 잘못된 URL 접속, 개발자 테스트 등으로 HTTP 요청이 TCP 서버로 유입
  - Netty 파이프라인은 들어오는 모든 바이트를 디코더로 전달
- **디코딩 실패 과정**
  - HTTP 요청 "GET / HTTP/1.1"의 앞 4바이트를 int(패킷 길이)로 해석
  - 'G','E','T',' ' → 0×47455420 → 1,195,725,856 바이트로 인식
  - 해당 길이만큼 데이터 대기 → ReadTimeout 발생

해결 및 결과



HTTP 요청 조기 감지 로직 구현
- decode() 메서드 초반에 버퍼의 첫 바이트를 검사하여 HTTP 패턴("GET", "POST", "HTTP/") 감지
- HTTP 요청 감지 시 400 Bad Request 응답 전송 후 연결 종료
- 정상 TCP 메시지만 기존 디코딩 로직으로 진행

추가적인 내용은 3페이지를 참고해주세요.

Portfolio

- 프로토콜 레벨 오류 차단으로 불필요한 타임아웃 에러 완전 제거
- 운영팀 알림 노이즈 감소로 실제 장애에 집중 가능
- 네트워크 프로토콜 수준의 문제 분석 및 근본 원인 해결
- 서버 안정성 향상 및 리소스 효율적 사용

EatMate

JOINED 전략을 사용한 모임 테이블 구성

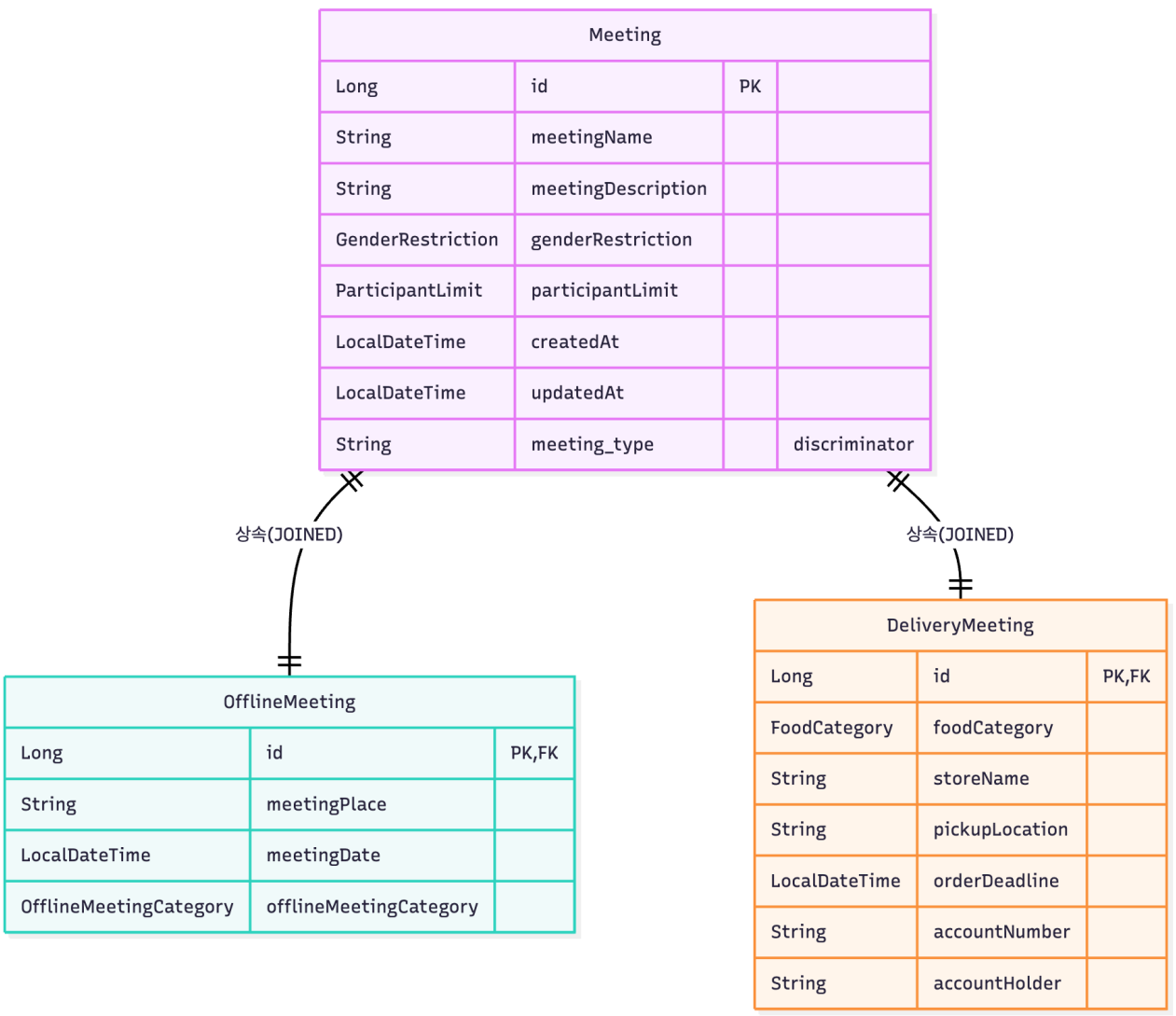
배경 지식, 문제

- 모임 서비스에서 배달모임과 대면모임이라는 두 가지 타입의 모임을 설계해야 하는 상황
- 각 모임 타입은 공통 속성(모임명, 설명 등)과 함께 타입별 고유 속성 (배달모임: 계좌번호, 픽업시간 / 대면모임: 만남장소, 음식 종류)을 가짐
- 데이터 중복을 최소화하면서도 효율적인 조회가 가능한 테이블 설계 전략 필요
- 추후 새로운 모임 타입 추가 시에도 유연하게 확장 가능한 구조가 요구됨

기술적 검토

- **SINGLE_TABLE 전략 (단일 테이블)**
  - 장점: JOIN 불필요로 조회 성능 우수, 구조 단순
  - 단점: 타입별 고유 컬럼에 null 값 다수 발생, 테이블 비대화, 데이터 무결성 제약 약함
- **TABLE_PER_CLASS 전략 (구체 클래스별 테이블)**
  - 장점: 각 타입이 독립적인 테이블로 관리되어 명확함
  - 단점: 공통 속성이 모든 테이블에 중복 저장, 전체 모임 조회 시 UNION 발생으로 성능 저하
- **JOINED 전략 (조인 전략)**
  - 장점: 데이터 정규화로 저장공간 효율적, null 컬럼 없음, 공통 속성 변경 시 부모 테이블만 수정
  - 단점: 조회 시 JOIN 연산 발생으로 성능 오버헤드 존재

해결 및 결과



추가적인 내용은 4페이지를 참고해주세요.

Portfolio

JPA JOINED 상속 전략 기반 테이블 설계
- 공통 속성은 Meeting 테이블에서 관리, 타입별 고유 속성은 자식 테이블에서 관리
- 각 자식 테이블은 Meeting 테이블의 PK를 FK로 참조하여 데이터 일관성 유지
- @SuperBuilder 어노테이션을 통해 부모-자식 클래스의 필드를 모두 포함한 빌더 패턴 구현으로 객체 생성 코드 직관성 향상

데이터 정규화 및 확장성 확보
- 저장공간을 효율적으로 사용하고 데이터 일관성 확보
- 공통 속성 변경 시 Meeting 테이블만 수정하면 되어 유지보수성 향상
- 새로운 모임 타입 추가 시 기존 테이블 구조 변경 없이 새로운 자식 테이블만 추가하면 되어 확장성 확보
- JOIN 연산 오버헤드가 있지만, 데이터 정합성과 확장성 측면에서 이점이 더 크다고 판단하여 채택

비관적 락을 이용한 동시성 문제 해결

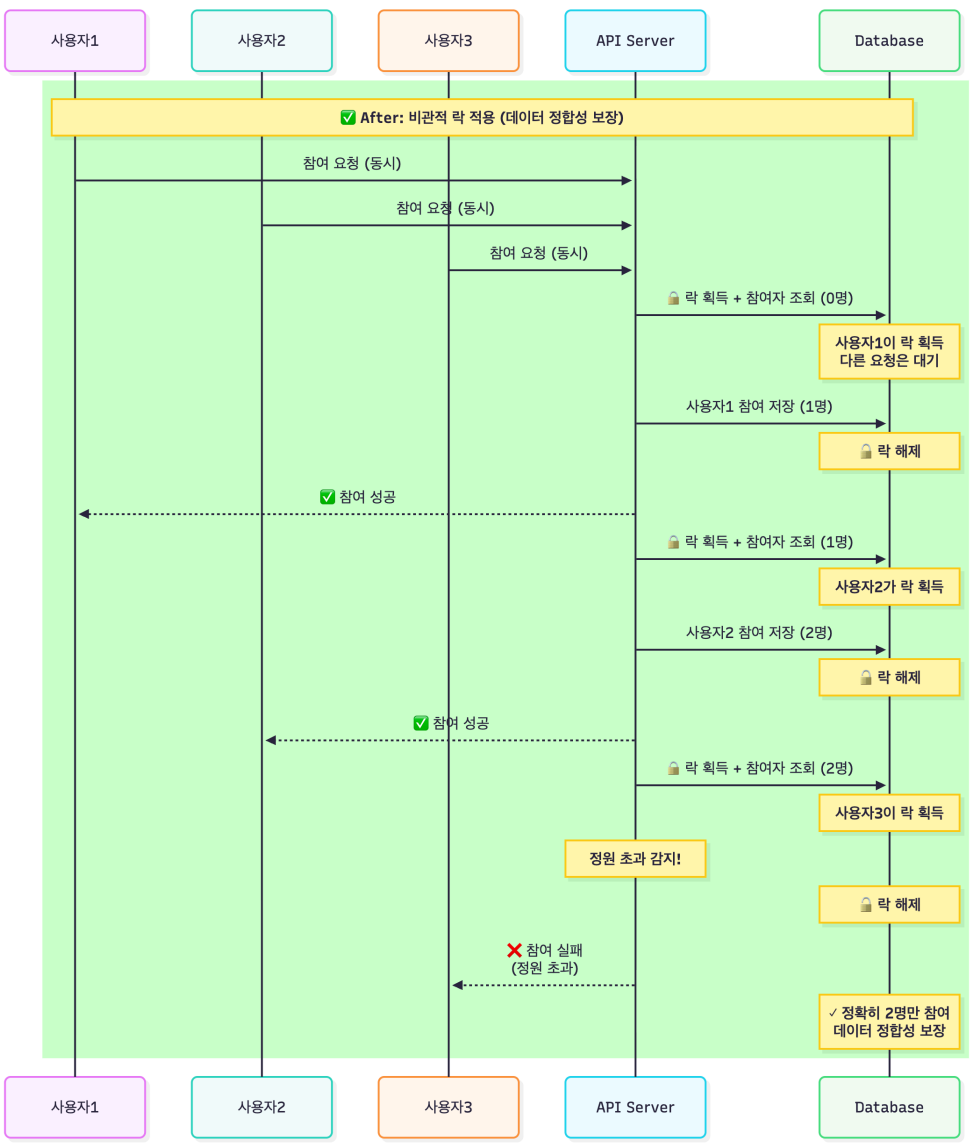
배경 지식, 문제

- 다수의 사용자가 동시에 모임에 참여를 시도할 때 발생하는 동시성 문제 상황
- 모임의 최대 참여 인원이 제한된 경우(예: 남은 정원 1명), 동시 요청 시 제한 인원을 초과하는 Race Condition 발생
- 동시에 2명이 참여 시도 시 둘 다 참여 성공 처리되어 정원 초과 문제 발생
- 데이터 정합성 보장과 사용자 경험을 모두 고려한 동시성 제어 전략 필요

기술적 검토

- **낙관적 락 (@Version)**
  - 장점: 충돌이 적을 때 성능 우수, DB 락 부담 없음
  - 단점: 충돌 시 예외 발생으로 재시도 로직 필요, 높은 경합 상황에서 빈번한 롤백으로 리소스 낭비, 사용자 경험 저하
- **비관적 락 (PESSIMISTIC_WRITE)**
  - 장점: DB 수준 배타적 락으로 동시 접근 원천 차단, 데이터 정합성 확실히 보장, 높은 경합 상황에서 효율적
  - 단점: 락 대기로 인한 성능 저하 가능성, 데드락 발생 가능성

해결 및 결과



Portfolio

JPA 비관적 락 기반 동시성 제어 구현

- @Lock(LockModeType.PESSIMISTIC_WRITE) 어노테이션으로 모임 조회 시 배타적 락 획득
- 모임 참여는 제한된 자리를 선착순으로 획득하는 특성상 높은 경합이 예상되어 비관적 락 선택
- DB 레벨에서 동시성을 제어하여 데이터 정합성 보장
- 충돌 시 재시도 없이 즉시 실패 응답 가능

성능 테스트 및 검증

- Junit을 사용하여 10명의 사용자가 동시에 10회씩 남은 정원 1명인 모임에 참여 시도하는 부하 테스트 진행
- 1명만 정상 참여하고 나머지는 실패 처리되어 동시성 문제 완전 해결 확인
- 처리량 246TPS, 처리 시간 406ms 기록
- 데드락 발생 없이 안정적으로 동작 확인

Hiphadi Menu

Nginx를 이용한 개발/배포서버 분리 구축

배경 지식, 문제

- 하나의 서버/컨테이너로 운영 중인 서비스라 추가 개발/유지보수 시 위험성이 높은 상황
- 개발 중 발생하는 오류나 테스트가 실제 운영 서비스에 직접 영향을 미칠 수 있는 위험
- 개발 환경과 운영 환경의 분리가 필요했으나 인프라 비용 부족
- 각 환경별로 독립적인 도메인과 SSL 인증이 필요했으나, 수동으로 관리하기 복잡함

기술적 검토

- **멀티 서버 구성 (물리적 분리)**
  - 장점: 완전한 환경 격리, 리소스 독립적 관리
  - 단점: 비용 증가, 관리 포인트 증가, 개인 프로젝트에 과도한 리소스
- **단일 서버 + 포트 기반 분리**
  - 장점: 비용 효율적, 구성 단순
  - 단점: 환경 격리 불완전, 의존성 충돌 가능성, 도메인 구분 어려움
- **단일 서버 + Docker 컨테이너 분리 + Nginx 리버스 프록시**
  - 장점: 비용 효율적이면서 환경 완전 격리, 도메인 기반 라우팅으로 명확한 구분, SSL 통합 관리 가능
  - 단점: Docker와 Nginx 학습 필요, 초기 구축 복잡도 존재

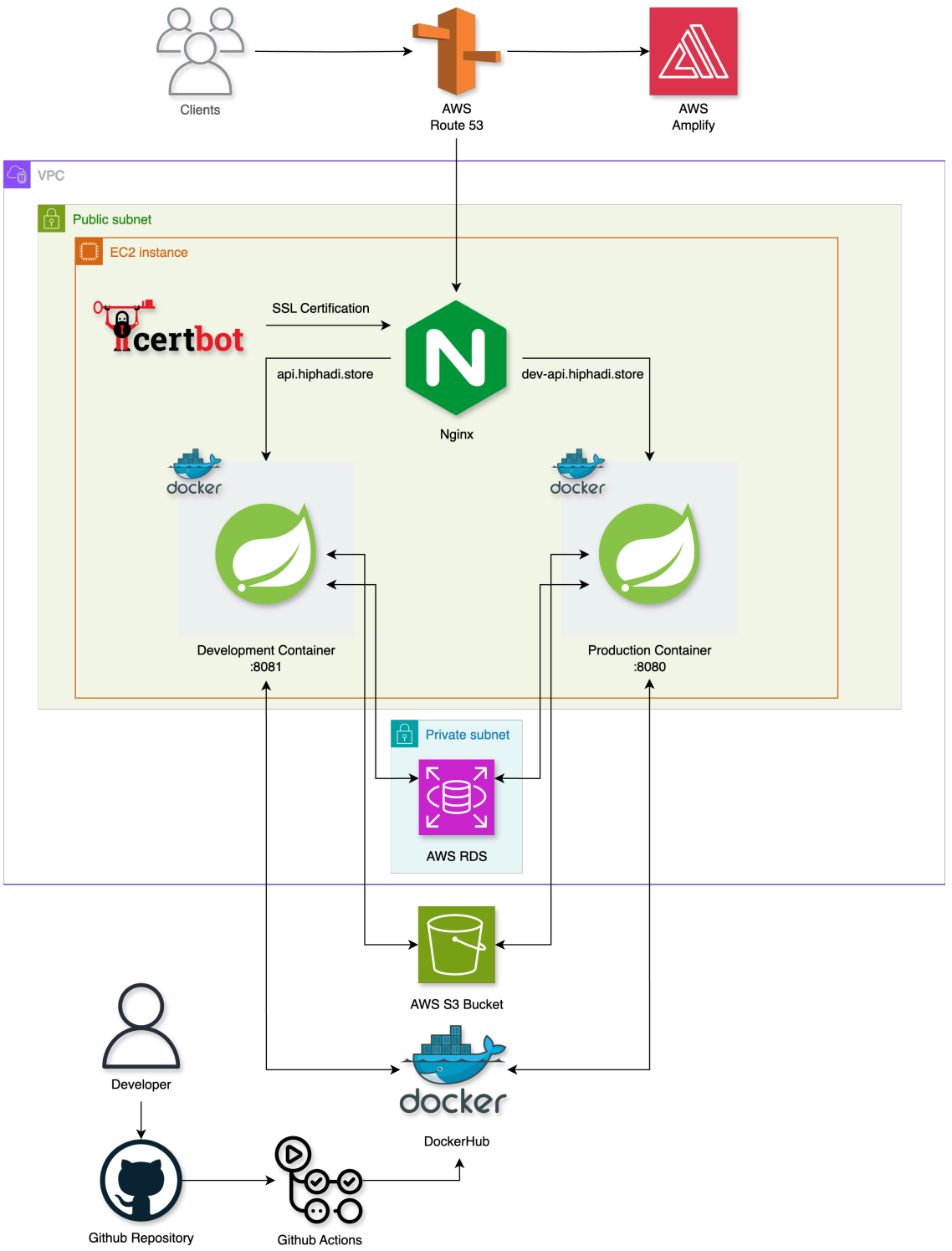
해결 및 결과

다음페이지에 아키텍쳐 구조가 있습니다.

- **Docker + Nginx 리버스 프록시 기반 환경 분리 구축**
  - Docker 컨테이너로 개발/운영 환경을 완전히 격리
  - Nginx 리버스 프록시를 통해 서브도메인 기반으로 각 환경에 라우팅
  - Certbot을 활용한 SSL 인증서 자동 발급 및 갱신으로 보안 관리 자동화
- **운영 안정성 및 유지보수성 향상**
  - 개발 중 발생하는 오류가 실제 서비스에 영향을 미치지 않도록 차단하여 운영 안정성 확보
  - 서브도메인 기반 명확한 환경 구분으로 유지보수성 향상

추가적인 내용은 6페이지를 참고해주세요.

Portfolio



-EOF-