유지상

제품 백엔드를 신규 구축하고, 운영 중인 서비스를 확장·개선해 온 Java/Spring 개발자입니다.

역할	Backend Engineer	위치	Seoul, Korea
연락	jidbyoo@gmail.com	GitHub	https://github.com/yjs1213
Portfolio	https://yjs1213.kr		

제품 규칙은 서버가 지켜야 합니다

금융기관 인증·조회 백엔드를 처음부터 구축했고, 기존 펌뱅킹에 추가한 API·SAP 연동 경로는 고객 환경에서 쓰이고 있습니다.
챌린지 플랫폼에서는 권한 검사를 공통 요청 경계로 옮기고 AWS 인프라를 구축·운영했습니다. 공동창업한 회사에서는
학습자가 코드를 작성하고 직접 공격해 결과를 확인하는 교육 제품을 팀 내에서 개발해 베타 공개했습니다.

경력

클라우드뱅크

2025.03.17 — 현재 · Backend Engineer · 선임 · 확인 2026-09-08

금융기관 인증·조회 백엔드를 처음부터 구축했고, 기존 펌뱅킹에는 API·SAP 연동 경로를 추가해 고객 환경에 배포했습니다.

Java 21 · Java 17 · Spring Boot · PostgreSQL · SAP JCo · SFTP · Spring Cloud OpenFeign · Playwright

맡은 일

- SAP와 연계되는 온프레미스 펌뱅킹 솔루션 개발
- 계좌 잔액·거래내역 등을 조회하는 금융기관 인증·조회 백엔드 신규 구축, 아키텍처와 구현 담당

금융기관 인증·조회 백엔드 신규 구축

기관마다 다른 인증·조회 요청과 응답을 Java로 구현하고, 기관별 차이는 Adapter와 Signer로 나눴습니다. 주요 흐름은 합성 데이터와 미리 저장한 응답으로 반복 실행하고, 서명 결과는 고정 입력·시각의 골든 벡터로 비교했습니다.

실제 금융기관 환경에서 인증서 로그인과 계좌 조회 동작을 확인했습니다. 기존 브라우저 자동화 경로를 유지하면서 직접 통신 경로로 옮기는 작업은 진행 중입니다.

SFTP 중심 FEP에 API 경로 추가

SFTP 파일 송수신과 SAP 전달 중심의 FEP에 HTTP API 조회와 SAP RFC 중계 경로를 추가했습니다. 첫 API 경로에서는 요청·응답 DTO와 Client·Service를 분리하고, SAP 조회 조건을 API 요청으로 바꾼 뒤 페이지 결과를 모아 SAP 응답으로 돌려주는 Handler를 구현했습니다.

후속 API 기관은 OAuth 인증과 보고서 생성·상태 조회·다운로드 절차가 달랐습니다. 해당 경로를 추가하면서 API 전용 기관은 SFTP scheduler에서 제외하고, SAP 채널과 기관의 허용 조합을 호출 전에 검사하도록 구성했습니다. 두 API 경로는 고객 환경에 배포돼 지금도 연동에 쓰이고 있습니다.

신규 전송의 상태 경계

신규 대량 금융 전송 기능에서 경쟁, 기존 전송 이력과 응답 불확실성을 사전에 구분해야 했습니다. 팀에서 상태 선점·중복 이력 차단·응답별 후속 처리 요구사항을 합의하고, 코드와 회귀 테스트를 구현·검토해 운영 환경에 배포했습니다.

중복 요청이 실제 전송으로 이어진 사고

전송 데이터가 늘면서 반복문 안의 건별 검증 조회가 길어졌고, 응답을 받지 못한 사용자가 같은 거래를 다시 요청하자 두 요청이 큐에 적재됐습니다. 재요청을 정상적인 실패 경로로 보고 큐 적재 전에 처리 상태를 변경했으며, 소비자가 전송 직전에 데이터를 다시 조회하도록 수정했습니다. 검증 데이터는 한 번에 조회하고 UPDATE는 batch 처리했으며, 운영 배포 후 2주 동안 같은 사고는 관찰되지 않았습니다.

설치형 전환

인증 처리를 고객사 내부에서 마쳐야 하는 단계에서는 연산 책임과 배포 산출물의 경계를 나눴습니다. 전체 인증·조회 로직을 설치형으로 옮기는 구현안은 별도 모듈과 native 바이너리로 구성하고 로컬 통제 환경에서 확인했습니다.

DB 데드락 대응

안정화 모니터링 중 결과 수신 처리가 금융기관 응답을 저장하지 못하는 DB 데드락을 발견했습니다. 송신·응답 로그와 InnoDB trace, 실행계획에서 독립된 두 트랜잭션의 record lock 순환 대기를 확인했습니다. 외부 중계망과 금융기관을 대체하는 테스트 서비스로 장애를 재현하고, 충돌하던 보조 인덱스를 제거한 뒤 UPDATE가 복합키를 사용하도록 변경했습니다. 운영 배포 후 2주 동안 같은 데드락과 응답 갱신 실패는 관찰되지 않았습니다.

컬처메이커스

2022.07.18 — 2025.03.14 · Backend Engineer · 매니저 · 확인 2026-09-08

챌린지 플랫폼의 역할·이벤트 소속 검사를 공통 요청 경계로 분리했습니다. AWS 인프라를 구축·운영하며 대회마다 응답 시간과 호출 경로를 확인했습니다.

Java 17 · Spring Boot 3 · PostgreSQL · Redis · AWS

맡은 일

- 챌린지 플랫폼 기능 개발과 도메인 재설계
- 백오피스와 권한 체계 개발
- 대회 기술 지원과 평가·집계 출력 자동화
- AWS 인프라 구축·운영·비용 관리
- Jira, Confluence, GitHub와 AWS 알림을 Slack 중심으로 연결하는 ChatOps 적용

흩어져 있던 권한 검사

권한 검사가 컨트롤러마다 흩어져 기능 코드와 뒤섞여 있었습니다. `@AuthZ`로 요구 정책을 표시하고, interceptor에서 로그인 사용자와 URL의 `eventId`를 조합해 컨트롤러 실행 전에 검사했습니다. 역할별 허용·거절과 잘못된 요청을 테스트한 뒤 운영에 반영했습니다.

Race condition을 이용한 제출 제한 우회

DB에서 제출 횟수를 조회·증가·판정하는 사이 동시 요청이 같은 횟수를 기준으로 통과해 제한을 초과할 수 있었습니다. 팀 단위 Redis Lua 카운터로 증가와 한도 검사를 한 번에 실행하도록 바꾸고, 동시 요청에서도 제출 제한이 유지되는지 테스트했습니다.

AWS 운영과 호출 경로 추적

ECS Fargate 전환 뒤 CloudWatch 로그는 유지하면서 관측을 두 갈래로 나눴습니다.
Actuator·Prometheus·Grafana에서는 응답 시간 추세를 보고, Pinpoint에서는 요청 호출 경로를 추적했습니다. 구축 뒤 대회마다 응답 시간을 중심으로 서비스 상태를 확인했습니다.

웨일즈랩

2020.07.06 — 2022.02.14 · Backend Engineer · 공동창업 · 확인 2026-09-08

2인으로 공동창업해 기술을 맡았고, 팀 내에서 시큐어코딩 실습 제품을 개발해 베타 공개했으며 공공 경진대회 예선 플랫폼과 40문제를 만들었습니다.

Java · Servlet · JSP · Spring Boot · MySQL · Python

이 제품은 회사보다 먼저 있었습니다. 케이쉴드 주니어 2기 교육에서 팀 프로젝트로 만들던 시큐어코딩 실습을 기반으로 2인이 회사를 세웠습니다.

창업 당시 최신 보안 흐름을 반영하기 어려운 정체된 콘텐츠와 일방향 온라인 강의를 문제로 보고, SW 전공 학생을 초기 대상으로 설정했습니다.

맡은 일

- 학습자 코드를 실제 웹 기능으로 실행하는 시큐어코딩 교육 플랫폼과 공격·방어 실습 시스템 개발
- 취약점의 원인과 방어 과정을 직접 경험하게 만드는 시나리오 기반 교육 콘텐츠 제작
- 풀이 공개, 댓글·반응과 다른 학습자의 생성 페이지 공격을 연결한 참여형 학습 기능 개발
- 소프트웨어 개발보안 경진대회 운영, 예선 40문제 출제와 온라인 플랫폼 개발
- 제품 기획과 사업계획, IR 자료 작성에 참여

실습 시스템의 구조와 담당 범위

학습자가 쓴 방어 함수를 취약점별 웹 기능에 넣고 그 기능을 직접 공격하게 하려면, 코드를 채점하는 대신 실행해야 했습니다. Java와 Python은 실행 방식이 달랐고, 학습자마다 코드와 판정 상태를 따로 유지하면서 서로의 풀이를 열어 공격할 수도 있어야 했습니다. Java는 문제별 JSP template에 학습자 함수와 검증 block을 조립하는 방식, Python은 학습자 함수와 공격 입력을 실행 스크립트로 구성하는 방식이었습니다.

개념 학습→방어 코드 작성→공격 실행→결과 확인·수정의 흐름을 팀 내에서 구현하고 케이쉴드 주니어 수료생에게 베타 공개했습니다.

경진대회 수주와 운영

회사가 KISA 발주 제8회 소프트웨어 개발보안 경진대회를 컬처메이커스와 컨소시엄으로 수주했습니다. 대회 운영을 맡고 예선 문제를 출제했으며, 온라인 플랫폼도 직접 만들었습니다.

예선 플랫폼은 대회 시간, 참가 트랙과 문제당 한 번의 제출을 서버에서 강제했습니다. 코드 실행과 판정은 기존 시큐어코딩 실습 제품의 API를 재사용하고, 대회용 실행 서버는 운영 서비스와 분리했습니다.

특허 공동 발명

이 실습 구조는 「시큐어 코딩 시스템 및 방법」으로 출원·공개됐고, 공동 출원인이자 공동 발명자 5인 중 1인으로 참여했습니다.

프로젝트

시큐어코딩 실습 제품 개발

웨일즈랩 · 2020 — 2022 · 백엔드 및 실습 콘텐츠 개발 · 확인 2026-09-08

방어 코드 작성·공격 실행·결과 확인과 학습자 간 풀이 공유·상호 공격을 잇는 실습 제품을 팀 내에서 구현했습니다. 교육 수료생을 대상으로 베타 공개했습니다.

Java · Servlet · JSP · JDBC · MySQL · Python · Tomcat

코드를 고치고 직접 공격해 보는 학습

창업 당시 최신 보안 흐름을 반영하기 어려운 정체된 콘텐츠와 일방향 온라인 강의를 시큐어코딩 교육의 문제로 보고, SW 전공 학생을 초기 대상으로 삼았습니다.

일반적인 온라인 저지가 코드의 출력값을 채점한다면, 이 시스템은 학습자의 방어 코드가 적용된 작은 웹 기능을 만들고 그 기능에 직접 공격을 보내 결과를 확인하게 합니다.

예를 들어 SQL Injection 문제에서는 학습자가 입력 처리 함수를 수정합니다. 시스템은 그 함수를 로그인 페이지에 반영하고, 학습자는 정상 로그인과 공격 문자열을 모두 입력해 방어 여부를 확인합니다.

학습 흐름은 다음과 같습니다.

- 취약점의 원인과 방어 방법을 학습합니다.
- 문제에서 요구하는 방어 함수를 작성합니다.
- 시스템이 코드를 취약점별 웹 기능에 반영합니다.
- 생성된 기능을 직접 공격하고 결과를 확인한 뒤 코드를 수정합니다.
- 풀이 코드를 공개하고 댓글과 반응을 남기며, 다른 학습자의 생성 페이지도 공격할 수 있습니다.

실습·판정·공유를 연결한 구현

실습 콘텐츠는 SW개발보안 가이드, OWASP, SANS, CWE의 취약점 분류와 대응 원칙을 참고해 구성했습니다.

- SQL Injection, XSS, OS Command Injection 등 SW개발보안 가이드 기반 취약점 실습을 제공합니다.
- Java 실습은 학습자 함수를 사용자별 실행 페이지에 조립하고, 8개 시나리오의 내장 검증 로직으로 성공·실패 상태를 저장했습니다.
- 풀이 코드를 취약점별로 저장·공유하고, 다른 학습자의 생성 페이지 공격과 댓글·반응 기능을 연결했습니다.

교육 플랫폼과 특허

이 실습 흐름을 팀 내에서 시큐어코딩 교육 플랫폼으로 구현했고, 「시큐어 코딩 시스템 및 방법」에 공동 출원인이자 공동 발명자 5인 중 1인으로 참여했습니다.

공식 기록은 https://doi.org/10.8080/1020190120363 에서 확인할 수 있습니다. 출원번호는 10-2019-0120363, 공개번호는 10-2021-0037895입니다.

베타 사용자 피드백

케이쉴드 주니어 수료생을 대상으로 베타 공개했습니다. Google 설문 응답에서 시나리오 실습이 시큐어코딩 학습에 도움이 된다는 정성 평가를 확인했습니다.

금융기관 인증·조회 백엔드 신규 구축

클라우드뱅크 · 2025 — 진행 중 · 백엔드 아키텍처 및 구현 · 확인 2026-09-08

금융기관 인증·조회 백엔드를 신규 구축했습니다. 기관별 요청·응답 차이를 Java 구현과 Adapter·Signer 경계에 반영하고, 실제 금융기관에서 로그인·조회 동작을 확인했습니다.

Java 21 · Spring Boot · BouncyCastle · PKCS#7/CMS · PKCS#8 · JDK HttpClient · Playwright

서명은 브라우저가 만들어 주지 않습니다

인증서 로그인은 페이지 조작만으로 통과되지 않습니다. 금융기관이 받는 값은 인증서로 만든 전자서명이고, 그 서명은 PC에 설치된 보안 모듈이 사람의 비밀번호 입력을 받아 만들어냅니다. 로그인을 자동화하려면 이 서명을 직접 만들어야 했습니다.

요청·응답과 보안 모듈 동작 분석

기관별로 로그인 요청과 응답을 캡처해 어떤 값이 어떤 순서로 오가는지 따라갔습니다. 서명이 만들어지는 지점은 브라우저 밖의 설치형 보안 모듈이라 코드를 읽을 수 없었고, 모듈이 내놓은 결과물을 리버스 엔지니어링해 무엇을 서명 대상으로 삼고 어떤 값을 함께 넣는지 역산했습니다. 서명 대상, 함께 담기는 속성과 인코딩은 기관마다 달랐습니다.

개인키 복호화와 서명 생성을 Java로 구현

공동인증서 개인키 파일은 표준 PKCS#8 외에 국내 표준 블록 암호를 쓰는 형식이 섞여 있습니다. 형식과 키 유도 방식을 판별해 복호화하는 처리를 직접 구현했습니다.

기관별 계약과 작업 실행 경계

복호화한 키로 서명을 만드는 코드는 기관별 Signer로 나누고 Factory가 요청에 맞는 구현을 선택하게 했습니다. 로그인과 조회 흐름은 기관별 Adapter에 담아 요청·응답 차이가 다른 기관의 흐름으로 번지지 않게 했습니다.

여러 작업은 직렬·병렬 실행 조건에 따라 queue에 넣고, watchdog이 실행 상태를 확인하며 완료 결과는 cache에서 조회하도록 구성했습니다.

실제 사용자 정보 없이 회귀 검증

주요 인증·조회 흐름은 합성 데이터와 미리 저장한 응답으로 반복 실행했습니다. 서명은 입력과 시각을 고정한 뒤 기대한 바이트와 같은지 골든 벡터로 비교해 암호 처리나 인코딩 변경을 확인했습니다.

인증서 로그인과 계좌 조회는 실제 금융기관 환경에서 동작을 확인했습니다. 기존 자동화 경로를 유지한 채, 브라우저 없이 직접 통신하는 경로로 옮기는 작업을 진행하고 있습니다.

설치형 전환의 두 산출물 경계

고객 인증서의 개인키를 고객사 밖으로 내보낼 수 없는 단계에서는 개인키 연산만 에이전트에 두고 기관별 서명 로직을 서버에 남겼습니다. 전체 인증·조회 로직을 설치형으로 옮기는 구현안에서는 보호할 책임을 native 바이너리 안에 모았습니다. 두 단계의 산출물 경계와 로컬 통제 검증은 별도 기술 사례로 분리했습니다.

채팅 메시지 캐시와 배치 저장

팀 프로젝트 · 2023 — 2024 · 백엔드 구현 · 확인 2026-08-29

Redis ZSet을 채팅 메시지 캐시로 사용하고, Spring Batch로 PostgreSQL에 이관하는 저장·조회 흐름을 구현했습니다.

Redis ZSet · Spring Batch · JDBC batchUpdate · PostgreSQL

최근 메시지는 Redis에서 읽기

채팅 메시지는 Redis ZSet에 먼저 저장했습니다. 조회는 cursor pagination으로 나누고, 캐시에 없는 구간만 PostgreSQL에서 채웠습니다.

배치로 영속 저장하기

Redis에 먼저 쌓인 메시지는 Spring Batch와 JDBC batchUpdate로 PostgreSQL에 옮겼습니다. 실시간 메시지 수신과 관계형 데이터베이스 저장을 나눈 구조입니다.

기술 사례

경진대회 예선 플랫폼

웨일즈랩 · 2021 · 예선 40문제 출제 및 플랫폼 개발 · 확인 2026-09-03

공공 경진대회 예선의 시간·트랙·1회 제출 규칙을 서버에서 강제하고, 코드 실행은 기존 제품에 맡기되 대회용 자원은 분리했습니다.

Spring Boot · Spring Security · JPA · Mustache · MySQL · CodeMirror

답을 고쳐 낼 수 없어야 하는 시험

공공 경진대회 예선을 온라인으로 치러야 했습니다. 참가 팀이 브라우저에서 답안 코드를 쓰고 제출하면 실행 결과로 판정하는 방식인데, 시험이라는 성격 때문에 플랫폼이 막아야 할 것이 한둘이 아니었습니다. 시작 전에 문제가 보이면 안 되고, 끝난 뒤에 들어온 제출이 받아지면 안 됩니다. 한 문제에 답을 여러 번 제출하며 맞춰 보는 것도 시험에서는 허용할 수 없습니다.

출제와 담당 범위

Java 20문제와 Python 20문제를 직접 냈고, 플랫폼 코드도 혼자 작성했습니다. 출제한 문제는 KISA가 구성한 평가위원단이 검수했습니다.

규칙을 서버 통제로 옮기기

대회 규칙	서버에서 강제한 경계
팀마다 Java와 Python 중 한 트랙에 참가	인증 뒤 경로를 트랙별로 분리하고 다른 트랙은 인가 단계에서 차단
운영자가 참가 계정을 발급	운영자 권한으로만 계정을 만들고 최초 로그인 때 비밀번호 변경 강제
시작 전 문제 노출과 종료 후 제출 금지	문제 조회마다 대회 시간 창 확인
반복 제출로 답을 추측하는 행위 금지	제출 이력을 확인해 문제당 한 번만 제출 허용

제출한 코드와 채점 결과는 팀 식별자와 문제 식별자, 제출 시각, 접속 IP와 함께 남겼습니다.

실행 엔진은 다시 만들지 않았습니다

코드를 실제로 돌려서 판정하는 부분은 이미 회사에 있었습니다. 시큐어코딩 실습 제품이 학습자 코드를 실행해 결과를 돌려주고 있었으니, 대회 플랫폼은 그 실행 API를 그대로 호출했습니다. 남은 일정에 실행 환경을 새로 만드는 대신, 플랫폼은 자격과 시간과 제출 횟수를 맡고 실행은 넘겼습니다.

자원까지 같이 쓰지는 않았습니다. 대회용 실행 서버는 운영 중인 서비스와 분리해 따로 올렸습니다.

확인한 구현 범위

저장소 검토로 40개 문제 경로, 트랙별 접근 제어, 시간 창 분기, 1회 제출과 실행 API 위임 구조를 확인했습니다.

예선 실행 경로의 응답 지연

웨일즈랩 · 2021 · 사후 원인 분석 · 확인 2026-09-03

서버를 분리해 두고도 제출이 몰리는 구간에서 응답이 밀렸고, 대회가 끝난 뒤 블로킹 호출을 병목 후보로 좁혔습니다.

Java · Tomcat · Thread · File I/O

떼어 놨는데도 밀렸습니다

예선의 코드 실행은 시큐어코딩 실습 제품의 API가 맡았고, 대회용 실행 서버는 운영 중인 서비스와 분리해 따로 올렸습니다. 자원을 나눠 쓰지 않으니 대회 트래픽만 받는 서버였는데도, 제출이 몰리는 구간에서 응답이 밀렸습니다.

대회가 끝난 뒤에 본 것

원인은 대회가 끝난 뒤에 확인했습니다. 실행 경로는 결과를 기다리는 데 `Thread.sleep` 호출을 쓰고 있었습니다. 요청이 하나씩 들어올 때는 드러나지 않습니다. 동시에 들어오면 스레드가 아무 일도 하지 않으면서 붙잡혀 있는 시간이 그대로 쌓입니다.

분석 범위와 개선안

실행 경로의 `Thread.sleep` 호출을 병목 후보로 좁혔습니다. 기다리는 구간을 non-blocking으로 바꾸고 실행 서버를 늘릴 수 있게 하는 방향을 개선안으로 제시했습니다.

SFTP 중심 FEP의 API 경로 확장

클라우드뱅크 · 2026 · API 연동 및 SAP 중계 구현 · 확인 2026-09-08

SFTP 파일 연계 흐름을 유지하면서 두 API 조회 경로와 SAP RFC 중계를 추가했습니다.

Java 17 · Spring Boot · SFTP · SAP JCo · Spring Cloud OpenFeign · OAuth 2.0

파일 경로를 없애지 않고 API를 더했습니다

FEP는 금융기관에서 파일을 내려받아 SAP로 전달하고, SAP가 만든 파일을 다시 올리는 SFTP 중심 애플리케이션이었습니다. 이 흐름을 유지하면서 SAP의 조회 요청을 금융기관 HTTP API로 보내고 결과를 바로 돌려주는 경로를 추가했습니다.

SFTP는 scheduler가 파일을 주고받는 비동기 처리였고, API는 SAP RFC 요청 안에서 외부 조회와 응답 변환을 끝내야 했습니다. 같은 금융 연동이어도 실행 시점과 데이터 계약이 달랐습니다.

첫 API 경로와 SAP 중계

API 요청과 응답은 타입이 있는 DTO로 정의하고 OpenFeign Client와 Service를 별도 패키지에 뒀습니다. Service가 요청 시각·식별자·거래 코드를 채우고 설정에 따라 호출 대상을 선택했습니다.

SAP RFC Handler는 SAP의 조회 조건을 API 요청으로 바꾸고, 여러 페이지를 끝까지 조회한 뒤 결과를 SAP 테이블과 응답 헤더로 변환했습니다. 무한 반복을 피하려고 페이지 수에 상한도 뒀습니다.

API 전용 기관을 SFTP 작업에서 제외

API만 사용하는 기관이 기존 SFTP scheduler에 들어가면 접속 정보가 없는 파일 작업이 반복해서 실패할 수 있었습니다. 기관별 SFTP 활성화 조건을 추가해 파일 처리 대상을 분리했습니다.

복수 SAP 채널에서는 채널과 기관의 허용 조합을 설정으로 관리했습니다. SAP 요청을 받은 Handler는 실제 API를 호출하기 전에 이 조합을 검사합니다.

두 번째 API 기관의 다른 인증·조회 절차

후속 API 기관은 OAuth 인증과 토큰 갱신이 필요했고, 조회도 보고서 생성 요청·상태 확인·파일 다운로드 순서로 진행됐습니다. 인증 Client, 토큰 갱신, 보고서 Client와 SAP RFC Handler를 별도 경계로 추가했습니다.

401 응답 뒤 토큰 갱신은 동시에 중복 실행되지 않게 직렬화했습니다. 보고서 식별자는 정해진 형식인지 확인하고, 다운로드는 설정된 크기를 넘으면 중단하도록 했습니다.

코드와 이력으로 확인한 범위

Git 이력에는 SFTP 중심 초기 구현 뒤 첫 API Client·Service, SAP RFC 중계, OAuth 기반 두 번째 API 기관이 순서대로 추가돼 있습니다. 첫 API Client·Service를 추가한 커밋은 기존 SFTP Java 코드를 수정하지 않았지만, SAP 중계까지 연결할 때는 기관 설정과 SFTP scheduler, Handler 등록부도 함께 변경했습니다.

저장소에는 호출 대상 선택, 페이지 수집, 채널·기관 허용 규칙, OAuth 흐름과 응답 변환을 검사하는 테스트가 있습니다. 두 API 경로는 실제 고객 환경에 배포돼 SAP와 금융기관 사이의 연동에 사용 중입니다.

설치형 전환의 산출물 경계

클라우드뱅크 · 2026 · 설치형 경계 설계·구현·검증 · 확인 2026-09-04

고객 인증서를 고객사 밖으로 내보낼 수 없고 설치 산출물이 고객 손에 남는 제약 아래, 개인키 연산 분리와 native 바이너리의 서로 다른 경계를 설계·검증했습니다.

Java 21 · Gradle Multi-Module · GraalVM Native Image · BouncyCastle · mTLS WebSocket

설치 산출물이 고객 환경에 남는 제약

서버가 보관하던 고객 인증서와 금융기관 로그인 로직을 고객사 환경으로 옮겨야 했습니다. 고객 인증서의 개인키는 고객사 밖으로 내보낼 수 없고, 설치한 산출물은 고객 손에 남아 디컴파일될 수 있으며 배포 뒤 원하는 시점에 고치기도 어렵습니다.

절대적인 역공학 방지를 목표로 삼지 않았습니다. 소스 형태의 직접 노출을 피하고 리버싱 비용을 높이는 보조 수단으로 보호 목표를 한정한 뒤, 구현 범위가 다른 두 단계의 경계를 따로 설계했습니다.

서버가 조회하는 단계는 개인키 연산만 분리

서버가 조회를 수행하는 단계에서는 개인키 연산만 고객사 에이전트에 두고 기관별 서명 로직은 서버에 남겼습니다. 패키지만 나누면 하나의 산출물에 함께 묶이므로 공유 계약·에이전트·서버를 별도 Gradle 모듈로 분리하고 컴파일 의존 방향으로 배포 경계를 강제했습니다.

에이전트는 업무 판단을 하지 않고 검증을 통과한 요청에만 개인키 연산을 수행합니다. 통신은 고객사에서 서버로 여는 상시 채널을 사용해 고객사 방화벽에 인바운드 연결을 추가하지 않도록 구성했습니다. 모듈 분리 뒤에는 에이전트 산출물에 기관별 서명 로직이 포함되지 않는 것을 빌드에서 확인하고 전체 테스트를 실행했습니다.

전체 로직을 옮기는 구현안은 보호할 책임만 native로

전체 인증·조회 로직을 설치형으로 옮기는 구현안에서는 기관별 로직을 서버에 남길 수 없었습니다. 인증·서명처럼 노출 비용이 큰 책임은 하나의 native 바이너리 안에 모으고, 오케스트레이션·영속·스케줄링은 밖으로 분리했습니다.

전면 재작성도 구현 대안으로 검토했습니다. 기존 인증·서명 구현은 실제 금융기관 대상 검증 이력이 있어 재작성 뒤 같은 범위를 다시 검증하는 부담이 컸습니다. 부분 native와 공유 라이브러리 방식은 보호 대상이 결국 에이전트 산출물에 함께 남으므로 선택하지 않았습니다.

기동 여부가 아니라 응답 본문까지 확인

도달 가능한 코드만 이미지에 포함되기 때문에, 작은 시험용 프로그램이 성공했다고 실제 데몬도 성공한 것으로 보지 않았습니다. 프로브와 실제 데몬을 각각 native로 빌드하고, 검증 시점에 정의된 endpoint 전체를 호출해 응답 본문을 단언했습니다. 메타데이터 누락은 빌드와 기동을 통과한 뒤 요청 시점에 드러날 수 있기 때문입니다.

기반 인증·조회 구현의 실제 금융기관 동작 확인과 설치 산출물의 검증 범위는 분리했습니다. 빌드 산출물의 모듈 경계와 native 바이너리의 응답 경로는 로컬 통제 환경에서 검증했습니다.

신규 금융 전송의 상태 경계

클라우드뱅크 · 2026 · 안전 요구사항 합의·구현·검증 · 확인 2026-09-03

신규 대량 금융 전송에서 상태 선점·중복 이력·불확실 응답 처리 규칙을 팀에서 합의하고, 코드와 회귀 테스트를 구현·검토해 운영 환경에 배포했습니다.

Java · Spring Transaction · MyBatis · JUnit · Mockito

외부 전송 전에 처리 주체부터 정했습니다

신규 대량 금융 전송 기능에서는 조회와 외부 전송 사이에 다른 실행이 끼어들 수 있었습니다. 이미 전송한 이력이 있는 데이터를 다시 보내거나, 성공 여부가 불확실한 응답을 실패로 단정하면 중복 전송과 결과 유실로 이어질 수 있었습니다.

실패 조건을 상태 경계로 바꾸기

위험	구현한 경계
두 실행이 같은 데이터를 조회한 뒤 모두 전송	외부 호출 전에 조건부로 상태를 선점하고, 다른 실행이 먼저 선점하면 전송 중단
기존 전송 이력이 있는 데이터의 재실행	전송 이력을 확인해 재전송 차단
외부 응답의 성공 여부가 불확실	성공·불확실·미전송 응답을 나눠 후속 처리

이 요구사항은 팀 회의에서 합의했습니다. 프로덕션 코드와 회귀 테스트를 구현하고 운영 배포 전에 직접 검토했습니다.

외부 의존성 없이 실패 경로 검증

실제 금융정보를 사용하지 않고 mock과 합성 응답으로 호출 순서, 경쟁 시 전송 차단, 중복 이력, 성공·불확실·미전송 응답을 반복 검증했습니다. 검증 범위와 경계 조건은 테스트 커버리지 기준과 고위험 기능 테스트 시나리오에 함께 기록했습니다.

운영 환경에 배포

사고가 난 뒤 보완한 기능이 아니라, 기능을 처음 만들 때 경쟁·중복 이력·응답 불확실성을 상태 경계로 정했습니다. 구현 코드와 회귀 테스트는 해당 기능을 사용하는 운영 환경에 배포했습니다.

금융 전송 중복 실행 사고 대응

클라우드뱅크 · 2026 · 장애 분석 및 백엔드 구현 · 확인 2026-09-03

처리 지연 중 동일 거래가 큐에 중복 적재된 사고를 재현하고, 적재와 소비 단계의 상태 검증을 함께 수정했습니다.

Java · SQL · Batch Update · Integration Test

응답 지연을 실패로 판단한 재시도가 중복 실행으로 이어졌습니다

전송 데이터를 수집하면서 데이터별 검증 조회를 반복해, 처리 건수가 늘어날수록 조회 횟수와 처리 시간도 함께 늘었습니다. 브라우저가 응답하지 않자 사용자가 새 브라우저에서 같은 거래를 다시 요청했고, 서버는 두 요청을 큐에 모두 넣었습니다. 큐에서 전송을 시작할 때도 최신 처리 상태를 확인하지 않아 동일 거래가 두 번 실행됐습니다. 이 사고는 안정화 기간에 전송 흐름을 지켜보던 중 발견했습니다.

재요청은 응답을 받지 못한 사용자가 할 수 있는 정상적인 행동입니다. 직접 원인은 같은 업무 요청을 큐에 다시 넣을 수 있었던 점이고, 실제 중복 실행으로 이어진 이유는 소비 단계에도 상태 검증이 없었기 때문입니다.

적재와 소비 단계에 나눈 방어

중복 적재와 소비 단계의 최신 상태 미확인이 각각 사고 원인이었기 때문에 한쪽만 바꾸지 않고 두 경계를 함께 수정했습니다.

전송 요청을 큐에 넣기 전에 처리 상태를 먼저 변경하도록 순서를 바꿨습니다. 큐 소비자는 저장된 데이터를 다시 조회하고 현재 상태를 확인한 뒤 외부 전송 여부를 결정하도록 수정했습니다. 같은 사용자의 같은 업무 전송이 진행 중이면 추가 실행도 막았습니다.

누적 데이터 처리 경로 변경

처리 지연을 만들던 반복 조회도 함께 수정했습니다. 반복문 안에서 데이터 한 건마다 검증 정보를 조회하던 구조를 필요한 데이터를 한 번에 가져와 검증하는 방식으로 바꾸고, 여러 UPDATE는 batch로 처리했습니다. 조회와 갱신에 사용하는 인덱스도 수정했습니다.

영향 범위에 따라 나눈 배포

전송 때마다 발급하던 전문 번호를 수집 단계에서 미리 부여하고, 전송 성공 뒤 사용 상태를 기록하는 보조 검증도 추가했습니다. 이 변경은 영향 범위와 테스트 부담이 더 커 사고 직후 패치와 분리했고, 별도로 검증한 뒤 운영에 반영했습니다.

장애 재현과 운영 확인

처리가 끝나기 전에 같은 거래를 다른 브라우저에서 다시 요청하는 흐름으로 사고를 재현했습니다. 변경 후 같은 시나리오에서 중복 전송이 일어나지 않는 것을 확인하고 운영 환경에 배포했습니다.

수정 전략과 테스트 결과는 팀에 공유했고, 같은 수정을 다른 업체 환경에도 순차 적용했습니다.

2주 동안 관찰했고 같은 중복 실행은 다시 나타나지 않았습니다.

금융 전송 DB 데드락 재현과 개선

클라우드뱅크 · 2026 · 장애 분석 및 백엔드 구현 · 확인 2026-09-03

운영에서 발생한 데드락을 재현하고, 원인이 된 인덱스와 갱신 쿼리를 수정했습니다.

InnoDB · SQL · Integration Test

데드락으로 이체 결과를 반영하지 못했습니다

안정화 기간에 금융 전송 흐름을 모니터링하던 중, 결과 수신 처리가 금융기관 응답을 데이터베이스에 반영하지 못하는 문제를 발견했습니다. 송신 상태를 바꾸는 트랜잭션과 결과를 저장하는 트랜잭션은 독립적으로 실행됐지만, 같은 전송 데이터를 서로 다른 인덱스 경로로 갱신하다가 데드락을 만들었습니다. 결과 수신 쪽 업데이트는 rollback됐습니다.

데드락 원인 확인

송신 시스템과 응답 중계 시스템의 로그로 장애 시점의 처리 흐름을 연결했습니다. DB 데드락 로그와 실행계획에서는 상태 조건이 포함된 보조 인덱스와 같은 행의 기본 키에서 record X lock의 획득 순서가 엇갈린 것을 확인했습니다. 두 트랜잭션이 상대에게 필요한 잠금을 보유해 순환 대기가 만들어진 경로였습니다.

외부 연동을 테스트 서비스로 재현

실제 흐름에는 외부 중계망과 금융기관이 있어 개발 환경에서 이체 요청부터 응답 수신까지 그대로 실행하기 어려웠습니다. 두 외부 역할의 요청과 응답을 대체하는 테스트 서비스를 만들고, 장애 당시 조건과 데이터 구조로 송신·응답 흐름을 연결했습니다.

변경 전에는 이 테스트에서 운영과 같은 데드락을 재현했습니다. 이 테스트 서비스는 장애를 확인한 뒤에도 버리지 않고, 이체 요청·응답의 회귀 검증에 계속 쓰고 있습니다.

원인이 된 인덱스와 갱신 쿼리 수정

로그와 실행계획, 재현 테스트로 특정한 충돌 경로에 맞춰 보조 인덱스를 제거하고, 상태 갱신 쿼리가 복합키로 대상 레코드를 찾도록 수정했습니다. 같은 테스트 흐름을 다시 실행했을 때 금융기관 응답까지 정상 반영되는 것을 확인한 뒤 운영 환경에 배포했습니다.

원인과 수정 내용은 팀에 공유했습니다.

2주 동안 관찰했고 같은 데드락과 응답 갱신 실패는 다시 나타나지 않았습니다.

Redis Lua Rate Limiter

컬처메이커스 · 2024 · 백엔드 구현 · 확인 2026-09-03

DB에서 제출 횟수를 조회·증가·판정하는 사이 동시 요청이 같은 횟수를 기준으로 통과할 수 있었습니다. 증가와 한도 검사는 팀 단위 Redis Lua 카운터에서 한 번에 실행하도록 묶었습니다.

Redis · Lua · Spring Boot · Spring AOP

애플리케이션 검사만으로 막을 수 없던 경쟁

CTF 답안 제출은 팀마다 허용 횟수가 정해져 있었습니다. 제출 횟수는 DB에 저장했지만, 제출 로직 안에서 조회·증가·한도 판정이 나뉘어 있었습니다. 두 요청이 거의 같은 시각에 들어오면 같은 횟수를 기준으로 둘 다 한도 안이라고 판단해 정해진 횟수보다 많은 제출이 통과할 수 있었습니다.

검사와 증가를 한 번에 실행하기

읽기와 쓰기 사이에 틈이 있는 한 애플리케이션에서 아무리 검사해도 같은 문제가 남습니다. 그래서 판단 자체를 Redis 안으로 옮겼습니다. Redis는 script를 실행하는 동안 다른 명령을 끼워 넣지 않으므로, 카운터를 올리고 한도와 비교하는 일이 하나의 실행 안에서 끝납니다.

카운터가 처음 만들어질 때만 TTL을 겁니다. 윈도가 지나면 key가 스스로 사라지므로 따로 정리하는 작업이 필요하지 않습니다.

제한 규칙을 한곳에 모으기

제한이 필요한 자리마다 Redis 호출을 직접 넣으면 규칙이 코드 곳곳에 흩어집니다. 어디에 걸리는지는 annotation으로 선언하고, 실제 검사는 AOP가 가로채도록 나눴습니다.

제출 처리에는 `@RateLimited(action = "submit", windowSeconds = 60, limit = 3)` 한 줄만 붙습니다. 다른 기능에 같은 제한이 필요해지면 값만 바꿔 붙이면 됩니다.

동시 요청을 테스트 코드로 확인하기

Spring Boot 테스트에서는 여러 작업이 같은 시점에 Lua script를 실행하도록 CountDownLatch로 시작 시점을 맞췄습니다.

테스트 결과

예시 시나리오는 요청 100건을 최대 20개 작업 스레드로 실행하고 허용 한도를 10건으로 둡니다. 허용 결과가 10건과 다르면 테스트가 실패합니다.

테스트 결과: 허용 10건, 차단 90건.

코드와 도식은 https://yjs1213.kr/projects/case-redis-lua-rate-limiter 에 있습니다.

지표 수집과 APM 호출 추적

컬처메이커스 · 2024 · 백엔드 구현 · 확인 2026-09-03

ECS Fargate 전환 뒤 CloudWatch 로그만으로 빠르게 보기 어려웠던 응답 시간 추세와 요청 경로를 지표 수집과 호출 추적으로 나눠 관측했습니다.

Spring Actuator · Prometheus · Grafana · Pinpoint

배포 방식이 바꾼 관측 조건

서버를 컨테이너화해 ECS Fargate로 배포하면서 개별 서버에 접속해 로그를 관리하기 어려워졌습니다. CloudWatch에서 로그는 확인할 수 있었지만, 응답 시간의 변화와 요청이 지나간 경로를 빠르게 파악하려면 다른 관측 수단이 필요했습니다.

지표와 호출 추적을 나눠서 붙이기

CloudWatch 로그는 근거 기록으로 유지했습니다. Spring Actuator가 노출하는 지표는 Prometheus로 수집해 Grafana에서 응답 시간 추세를 확인하고, 개별 요청이 지나간 구간은 Pinpoint APM으로 따로 추적했습니다.

구축 뒤 대회가 진행될 때마다 Grafana에서 응답 시간을 중심으로 서비스 상태를 확인했습니다.

요청 경계로 분리한 이벤트 권한 검사

컬처메이커스 · 백엔드 구현 · 확인 2026-09-08

컨트롤러마다 반복되던 역할·이벤트 소속 검사를 어노테이션과 인터셉터로 분리하고, 요청 경계에서 사용자와 경로 리소스를 조합해 검사했습니다.

Spring MVC · Spring Security · HandlerInterceptor

비즈니스 로직에 반복되던 권한 검사

역할과 이벤트 소속을 확인하는 코드가 컨트롤러의 비즈니스 로직마다 반복되고 있었습니다. 권한 정책과 기능 코드를 분리하기 위해 메서드에는 필요한 정책만 `@AuthZ`로 표시하고 실제 검사는 공통 요청 경계로 옮겼습니다.

인터셉터에서 조합한 세 가지 정보

권한 판단에는 요청을 처리할 메서드, 로그인 사용자와 URL의 `eventId`가 함께 필요했습니다. 인터셉터는 `HandlerMethod`에서 `@AuthZ` 정책을 읽고 SecurityContext의 사용자 ID와 경로 변수를 조합해 주최자·심사위원·이벤트 참여 관계를 검사했습니다. 검사를 통과한 요청만 컨트롤러를 실행했습니다.

실패 경로를 포함한 검증

각 역할의 허용·거절뿐 아니라 인증 정보가 없는 요청, 잘못된 `eventId`와 권한 없는 사용자를 가정한 테스트를 작성했습니다. 테스트를 통과한 권한 구조를 운영에 반영했습니다.

STOMP 메시지 인가

팀 프로젝트 · 2023 — 2024 · 백엔드 구현 · 확인 2026-08-29

WebSocket에서는 기본 필터가 동작하지 않아 프레임 단계에서 인가를 검사했습니다.

WebSocket · STOMP · JWT · ChannelInterceptor

로그인은 했지만 그다음이 비어 있었습니다

REST 요청은 필터가 매번 검사합니다. WebSocket은 다릅니다. 연결이 한 번 열리고 나면 그 위로 오가는 메시지는 필터를 지나지 않습니다.

그래서 남의 채팅방을 구독하거나 그 방으로 메시지를 보낼 수 있었습니다.

판단을 프레임 단계로 옮기기

인가를 연결 시점이 아니라 프레임 시점으로 옮겼습니다. CONNECT에서는 토큰의 서명과 만료를 확인해 세션 주체를 정합니다. 그 뒤 SUBSCRIBE와 SEND는 매번 destination이 가리키는 방과 세션 주체의 구성원 여부를 확인합니다.

접속 시점 검사와 나란히 돌리기

접속 시점에만 검사하는 구현을 따로 만들어 같은 시나리오를 양쪽에 돌렸습니다.

시나리오	프레임 단계 검사	접속 시점만 검사
토큰 없음·서명 위조·만료	세션 안 열림	세션 안 열림
비구성원의 구독	거부, 구독 저장 안 됨	통과
비구성원의 송신	거부	통과
거부된 세션의 메시지 수신	도달 안 함	도달함
방에서 제외된 뒤 송신	거부	통과
구성원 16·비구성원 16 동시 구독 후 브로드캐스트	구성원 16건만 수신	32건 모두 수신

이미지 변환과 업로드 병렬화

팀 프로젝트 · 2024 · 백엔드 구현 · 확인 2026-09-09

원본을 그대로 쓰던 미리보기와 순차 업로드를 S3 event Lambda 변환과 병렬 업로드로 바꾸고 테스트에서 측정했습니다.

AWS Lambda · S3 event · CompletableFuture

미리보기에도 원본 이미지를 그대로 썼고, 이미지를 여러 장 올릴 때는 한 장씩 순서대로 업로드했습니다.

변환을 업로드 경로 밖으로

S3 event로 Lambda를 실행해 압축과 리사이징을 애플리케이션 밖에서 처리하고, 업로드 자체는 CompletableFuture로 병렬화했습니다.

테스트에서 164.6 KB 원본은 1.6 KB 압축본과 10 KB 리사이즈본으로 변환됐습니다. 같은 환경에서 이미지 10장을 올렸을 때는 병렬 업로드가 순차 업로드보다 약 두 배 빨랐습니다.

세 OAuth Provider의 공통 흐름과 변경 지점 분리

팀 프로젝트 · 2023 — 2024 · 백엔드 구현 · 확인 2026-09-03

Google·Kakao·Naver 로그인에 반복되던 실행 순서는 template으로 묶고, endpoint와 응답 검증은 provider별 callback으로 분리했습니다.

Java · Spring Boot · OpenFeign

같지만 달랐던 세 로그인 흐름

Google·Kakao·Naver OAuth 로그인은 인가 코드를 토큰으로 바꾸고, 토큰으로 사용자 정보를 조회한 뒤 내부 로그인이나 가입으로 연결하는 순서가 같았습니다. 하지만 API endpoint, 요청값과 응답 검증은 provider마다 달라 별도 메서드에 공통 흐름까지 반복되고 있었습니다.

순서는 template, 차이는 callback

인가 코드→토큰→사용자 정보 조회 순서는 `OAuthApiTemplate`에 고정했습니다. 토큰 요청과 사용자 정보 변환은 `OAuthLogin`, `OAuthUserInfo` 인터페이스의 provider별 구현으로 분리하고, 결과를 공통 `OAuthUser`로 맞춰 내부 사용자 조회·JWT 발급·가입 연결을 하나의 흐름에서 처리했습니다.

Google·Kakao·Naver의 기존 로그인 흐름을 테스트한 뒤 운영에 반영했습니다.

LLM을 이용한 생성 문장 검수

개인 프로젝트 · 2026 · 평가 설계 및 구현 · 확인 2026-09-03

생성 문장의 품질을 정해진 기준으로 LLM에 평가시키고, 발견한 오류는 같은 문제가 반복되지 않도록 코드 검사에 추가했습니다.

LLM-as-judge · Structured output · JSONL · Node.js · Python

규칙 검사로 답할 수 없던 질문

무작위로 조합한 영어 문장이 일부러 엉뚱하게 만든 것으로 읽히는지, 그냥 틀린 영어로 읽히는지 구분해야 했습니다. 관사와 문법처럼 답이 정해진 문제는 코드로 검사했지만, 문장이 의도대로 읽히는지는 직접 읽어야 판단할 수 있었습니다. 작가 5종·충돌 유형 5종·발화 형식 6종을 조합하면 확인할 경우가 150개라 사람이 매번 모두 읽기 어려웠습니다.

평가 범위와 기준

- LLM 평가는 개발 과정에서만 사용했습니다. 실제 게임의 문장 생성, 점수 계산, 무작위 선택과 등급 판정에는 연결하지 않았습니다.
- 8가지 기준을 1~5점으로 평가하게 하고, 판정과 실패 이유는 미리 정한 값만 쓰도록 응답 형식을 JSON으로 고정했습니다.
- 문장만 보여 주는 평가와 문장이 만들어진 조건까지 보여 주는 평가를 나눴습니다. 전자는 독자가 받을 인상을, 후자는 문제가 생긴 원인을 확인했습니다.
- seed를 고정해 같은 조건에서는 같은 표본이 나오게 했습니다. 조합별로 표본을 뽑고, 평가에 사용한 입력은 회차별로 따로 보관했습니다.
- 두 평가가 다르면 장점은 더 낮게, 위험은 더 높게 잡았습니다. 한 명이라도 결함을 지적하면 검토 대상으로 남겼습니다.

모델이 찾은 결함은 코드 규칙으로 남겼습니다

검수는 네 차례 진행했습니다. 먼저 20개 문장으로 평가 기준을 시험했고, 150개 조합 가운데 표본을 만들 수 있었던 144개를 평가했습니다. 이어 25개를 다시 평가하고, 서로 결과를 공유하지 않는 두 모델에 30건을 평가하게 했습니다. 표본을 만들지 못한 조합은 6개였습니다.

144개 조합을 평가하는 과정에서 주어와 동사가 맞지 않는 문장을 찾았습니다. 모델이 제안한 문장으로 한 번만 고치는 대신, 오류를 일으키는 복수 명사를 해당 위치에서 제외했습니다. 이어 그 자리에 올 수 있는 모든 명사를 검사하도록 코드를 바꿨습니다. 무작위 문장 120개만 확인하던 기존 검사도 수정하고, 일부러 오류를 넣은 5가지 경우로 제대로 잡아내는지 확인했습니다.

모델 평가가 놓친 오류도 있었습니다. 본문만 평가했기 때문에 제목에 관사가 빠진 문제는 잡지 못했습니다. 제목 1,000개를 따로 확인해 잘못된 문장 비율을 9.6%에서 0%로 낮췄습니다.

학력·교육·수상·자격

학력

수원대학교 정보보호학과 학사

2014.03 — 2020.02

교육

케이쉴드 주니어 2기

2019

수상

제6회 소프트웨어 개발보안 경진대회 3위 · 행정안전부 장관상

2019

자격

정보처리기사 · 2024