

Backend Portfolio – Jang Wonseok

목차

1. Profile & Summary
2. Overview of boot-base Project
3. System Architecture
4. Authentication & Security Architecture
5. Backend Module Design
6. Performance & Reliability

github: <https://github.com/No1stone>

profile: <https://www.origemite.com>

1. Profile & Summary

이름: 장원석 (Jang Wonseok)

포지션: Backend Engineer

경력: 백엔드 5년

저는 AICC 솔루션 개발, SaaS 운영 경험과

1,700억 매출 규모의 민팃(Mintit)에서 백엔드 운영 및 차세대 MSA 개발

경험을 보유한 Backend Engineer입니다.

1) 민팃(Mintit) – 대규모 운영 및 차세대 MSA 개발 경험

민팃에서는 백엔드 운영 및 차세대 신규 MSA 시스템 구축에 참여했습니다.

운영 경험

전국 6,600대 ATM 단말기에서 발생하는 실시간 트래픽 운영

통신 3사의 중고폰 반납 데이터 배치 처리 문서화

반납 → 회수 → 물류 입고까지 이어지는 디바이스 추적 프로세스 문서화

물류센터 수선/검수/재고 전환 등 복잡한 백엔드 업무 흐름 분석 및 문서화

차세대 MSA 신규 개발

민팃 ATM의 AI 판정 모델을 위한 **약 1억 장 이미지** 기반 데이터 파이프라인 개발

민팃safe (디바이스 개인정보 완전 삭제 앱) 백엔드 전면 개발

SK네트웍스 계열사로서 요구되는 **ISO27001** 기반 보안 요건 준수 개발

정보보호팀과 협업하여 보안 요구사항을 운영, 배포 단계에 반영

2) (아이컴시스) AICC 솔루션 개발 및 SaaS 구축 경험

아이컴시스에서 AICC(상담/콜센터) 솔루션을 개발하며

고객 응대 시스템 구조와 대규모 음성/시나리오 기반 서비스 아키텍처를 깊게 이해하였습니다.

고객사(LG U+) "우리가게 AI" 프로젝트의 어드민 시스템 개발

(사용자 권한관리, 시나리오 관리, NLP 인텐트/엔티티 관리, 통계 등)

내부 SaaS 플랫폼을 인프라 → 백엔드 → 배포까지 신규 구축하여 오픈

신규 Outbound 솔루션을 직접 설계, 신규 개발 → 기존 SaaS에 성공적 통합 및 운영

AICC 업무를 통해 채널 시스템 구조, 음성봇, 챗봇 연동, 시나리오 기반 서비스 설계 등

복잡한 비즈니스 로직을 설계, 운영한 경험을 갖추었습니다.

2. Project Overview – boot-base Monorepo

프로젝트 개요

boot-base는 Spring Boot 3 기반의 백엔드 서비스를
실서비스 수준 아키텍처, 공통 라이브러리,
인프라 연동까지 포함해 한 번에 구성할 수 있도록 만든 개인 백엔드 베이스 프로젝트입니다.
실제 엔터프라이즈 환경에서 요구되는 인증, 로깅, 모듈 분리, MSA 확장 연동을 검증하기 위한 토대이며,
새로운 서비스가 생기면 Monorepo에 서비스 모듈만 추가해서 확장할 수 있도록 설계했습니다.

단일 저장소(Monorepo) 설계 목적

- 공통 코드 중앙 관리
- 서비스 간 로직/DTO 불일치 문제 최소화
- 하나의 저장소에서 관리해 버전 관리와 배포 파이프라인을 단순화

모듈 구성

gateway

- 외부 트래픽 진입 지점
- 경로 기반 라우팅, 인증 필터(JWT 검증), 로깅 처리 담당

api-auth

- 인증, 인가 담당
- Vault Transit/JWKS 관리

lib-common

- 공통 Exception, 공통 응답 포맷, 유ти리티 설정 등
- 서비스 모듈이 참조하는 베이스 라이브러리

lib-model

- 공통 DTO/Domain Model 정의
- 서비스 간 데이터 규격을 통일하고 중복 정의 방지

기술 스택 요약

Language & Framework

- Java 17, Spring Boot 3.5
- Spring Web / Spring Data / Spring Cloud

Architecture

- Monorepo + Gradle MultiModule
- API Gateway + Auth + service

Security

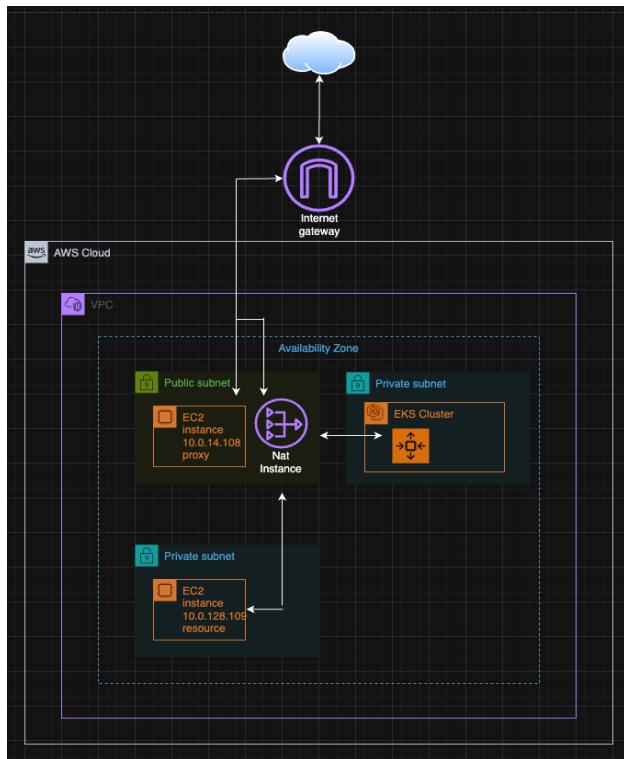
- Vault Transit 기반 RSA 키 관리 및 JWKS 로테이션
- JWT/Refresh Token 기반 인증 구조

Observability

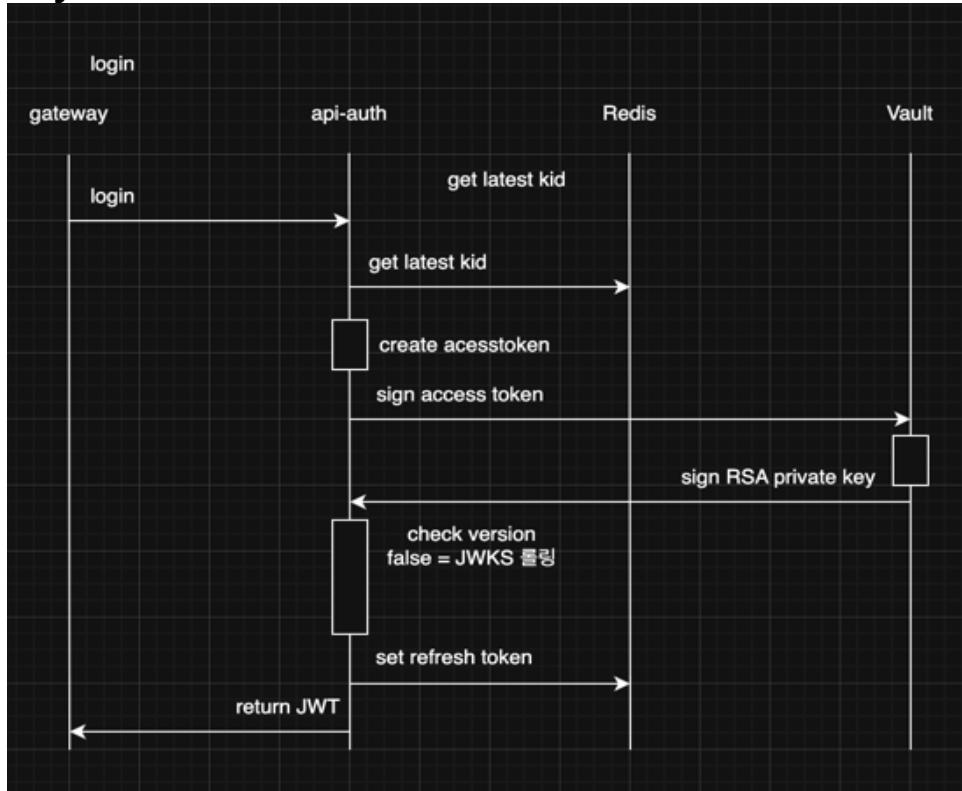
- Otel / Grafana / Prometheus / Loki /zipkin 연동

3. System Architecture

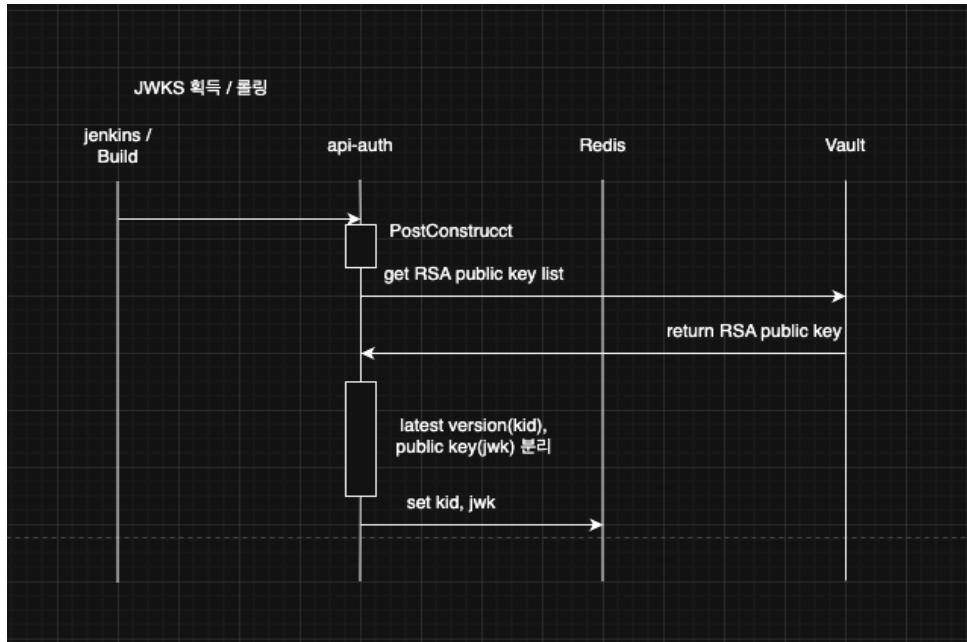
3.1 인프라 구성



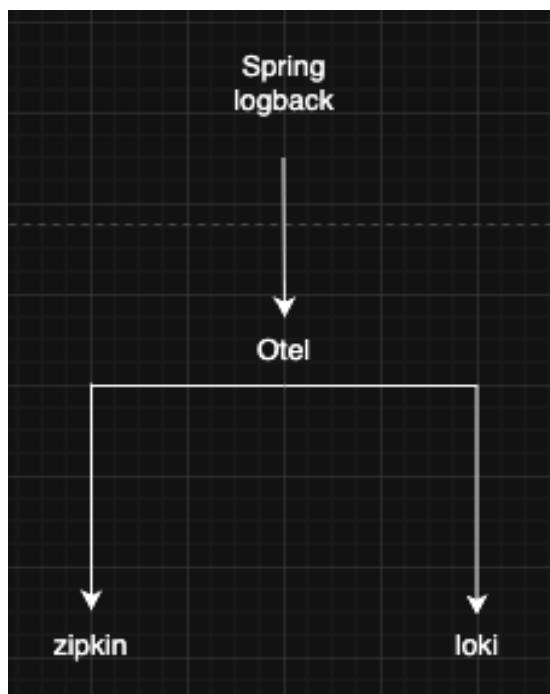
Gateway → Auth → Vault



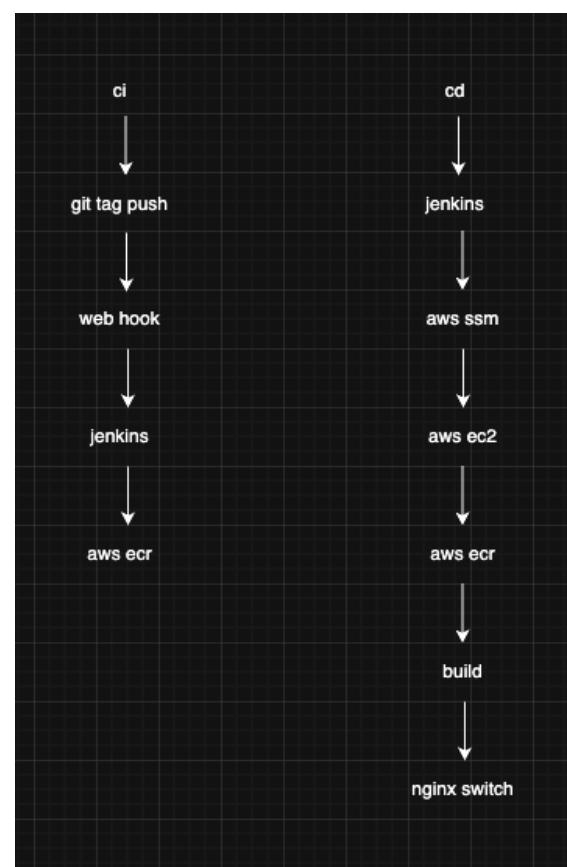
Vault Transit



Observability



CI/CD 흐름 (ci/cd 분리)



4. Authentication & Security Architecture

4.1 시스템 인증 구조

Gateway → Auth → Service 흐름에서

JWT 서명 검증과 JWKS 조회가 정상 동작하는지를 검증하는 데에 집중했습니다

Access/Refresh Token, Token Versioning 등은

실제 계정/테이블이 추가되면 그 구조를 그대로 적용할 수 있도록 모듈을 분리해 두었습니다.

4.2 Vault Transit 기반 RSA 키, JWKS 로테이션

HashiCorp Vault Transit를 사용해 애플리케이션 코드에서 키를 직접 보관하지 않고

Vault가 RSA 키 생성, 서명, 조회 역할을 수행하도록 구현했습니다.

서버는 검증용 public key 만 조회하고 제공할수 있도록 구현하였습니다.

public key는 최신키와 직전키 1개를 관리하는 구조입니다.

4.3 보안 정책

모든 외부 요청은 **Gateway**를 통과하며,

Gateway Filter에서 **JWT 존재 여부와**

필요시 서명 검증을 선행한 뒤 내부 서비스로 라우팅하는 구조를 사용합니다.

5. Backend Module Design

5.1 API 레이어

Controller / api-{{service}}

HTTP/JSON 요청을 받는 가장 얇은 레이어로

통상적으로 사용하듯 (Validation), HTTP Status 처리만 담당합니다

Facade / api-{{service}}

도메인 Facade 클래스를 두어 API에서 필요한 Use Case를 모아두는 레이어입니다

내부적으로 Query 메서드와 Command 메서드를 구분(CQRS 개념 차용)하여,

조회용 메서드는 읽기 전용 흐름을, 변경용 메서드는 트랜잭션이 필요한 흐름을 명확히 나눕니다

클라이언트에 반환할 DTO 형태는 **API/BFF 관점에서 최적화된 응답 구조**를 사용하고,

내부 Service/Repository 구조는 숨김겨 **외부 인터페이스를 안정적으로 유지할 수 있도록** 했습니다

엄격한 CQRS 프레임워크를 적용한 것은 아니지만,

읽기/쓰기 책임 분리 + API 응답을 위한 BFF 역할을 실용적으로 구현한 계층입니다

Service / lib-model

Facade에서 호출하는 순수 도메인 로직 레이어입니다

향후 REST 외에 gRPC/GraphQL 등 다른 인터페이스에서도 재사용할 수 있게 하는 목적이 있습니다

Repository / lib-model

JPA Interface 저장소입니다.

6. Performance & Reliability

6.1 Mintit 장애 대응 및 운영 개선 경험

문제

민팃의 기존 시스템은 외주에서 개발된 레거시 구조로 인해

다음과 같은 운영, 성능, 보안 문제가 반복적으로 발생하고 있었습니다.

비정규화 테이블(img1~img30) 수평확장 설계

조인/카디널리티 오류 및 다수 장애 발생

MyBatis 기반 쿼리에 **비즈니스 로직이 혼재**, 유지보수 난이도 증가

동일 의미의 중복 공통코드, 하드코딩 로직 다수

영업제휴부서가 운영 DB를 직접 조회하는 **비인가 보안 루트 존재**

운영/설계 문서가 간접되지 않아 실무, 운영 간 불일치 발생

WAF 업그레이드 후 Redis빌드 초기에 **Fail-Fast + Race Condition**으로 서비스 지연 발생

외부 API(가상계좌, 결제) **불필요한 비용 발생**

조치

데이터/쿼리 안정화

암시적 조인 → 명시적 JOIN 구조로 리팩토링

PK/카디널리티 재정렬을 통해 **조회 성능 정상화**

배치는 **REQUIRES_NEW 트랜잭션** 로직 추가

API & 외부 연동 개선

외부 API 문서 재정비 및 신규 API 흐름 문서 작성

가상계좌 송금 사용자 이름 계좌번호 확인 API의호출 제거 → 월 단위 비용 절감

운영/보안 리스크 제거

WAF 업그레이드 이후 전체 서비스 장애 원인을 직접 분석하고

→ 인프라팀 WAF 업그레이드 절차 Redis 빌드순서 체크리스트 추가

운영 DB 무단 접근 루트를 파악하여

→ IAM/DB 접근 권한을 전면 재설계

AWS IAM Key 유출 사고를 조기에 탐지하고

→ 즉시 키 회수 + 재발 방지 프로세스 구축

문서화 및 구조화

운영 DB, 배치, API, 연동 등 모든 핵심 흐름을 도식화

차세대 MSA 전환 프로젝트를 위해

기존 레거시 구조를 문서/도면 단위로 재정의

결과

레거시 운영 안정성과 인력 의존도 감소

운영 DB 무단 접근 **100% 차단**

배치 실패 로직 정확성 향상 및 데이터 정합성 안정화

WAF 업그레이드시 장애 재발 방지

문서, 구조를 구축하고 인수인계 가능 수준으로 체계화

6.2 아이컴시스 운영 개선 경험

문제

아이컴시스의 기존 AIICC 솔루션은 전자정부프레임워크 기반 구조

혼합된 코드베이스, 노후화된 인프라로 인해 다음과 같은 문제가 지속 발생했습니다

전자정부프레임워크 기반의 모놀리식 구조

JPA 미도입으로 도메인 중심 개발 불가, MyBatis 쿼리에 로직 혼재

운영 문서/DB 스키마 관리 부재 → 솔루션 납품 고객사 환경별 불일치 다수

AWS 인프라 비용 과다 + 서버 public 노출 및 TLS 미적용

JSP 기반 어드민 페이지 유지보수 난이도 증가

대규모 이력 데이터 조회 시 성능 저하

jquery DataTable 조회 시 페이징 미적용

조치

1. 전자정부프레임워크 → **Spring Boot 2.7**로 전면 마이그레이션

Java 8 → **Java 11** 업그레이드

Maven → **Gradle** 전환, yml 기반 환경 분리

Egov 권한관리 → AWS IAM 기반 구조 재정립

JPA + QueryDSL 도입으로 도메인 중심 개발 체계 수립

DB 형상관리에 **Flyway** 도입 (개발·운영 스키마 불일치 제거)

2. LG U+ 우리가게 AI 고도화

JPA, S3 등 신규 기술 가이드 제작

월 300만 건 대규모 로그/이력 데이터 성능 테스트

인덱스 재정비 및 쿼리 최적화 진행

고도화 DB 마이그레이션 절차 수립

운영 문서(배포/용량 산정/롤백 계획) 정리

3. 사내 인프라 고도화

SaaS 서비스 오픈을 위한 **AWS 인프라 신규 구축**

ACM 기반 HTTPS 인증서 자동화

GitLab → **Gitea** 이전

Bastion, ELB, Snapshot/Backup 구성

개발 서버 스케줄링으로 월 300 USD 비용 절감

전체 AWS 비용 700 → 300 USD 수준으로 감소

4. 운영 개선 및 서비스 안정화

JSP → Thymeleaf 전환

서브도메인/Nginx/서드파티 구성 체계화

DB Flyway 스크립트 관리, 운영 반영 자동화

NAS, VPN 신규 설치 및 운영

Session → **JWT** 기반 인증 구조로 전환

jQuery DataTable 문제 해결 → 서버 페이징 적용

결과

전자정부 구조 → **Spring Boot** 기반 구조로 전환 성공

JPA/QueryDSL 도입 후 **도메인 기반 개발 체계 확보**

운영/배포 문서화로 팀 내 인수인계 비용 감소

AWS 비용 절감(약 **50~60%**)

SaaS 라인업 확장 기반 마련

인증체계(Session → JWT) 전환으로 **보안성과 확장성 향상**

감사합니다.