경력기술서 — 길현준

게임듀오

2025.01 — 현재

모바일 게임 개발 및 퍼블리싱 회사. 6개 라이브 게임 환경(총 12 브랜치 운영) 공통 백엔드 플랫폼 설계, 데이터 파이프라인 구축, 공통 패키지 체계 운영

프로젝트

실시간 게임 채팅 서버 (멀티테넌트)

2026.04 ~ 현재

여러 라이브 게임이 공유하는 멀티테넌트 실시간 채팅 서버를 설계하고, 6,000 동시접속·18,000 RPS 목표 용량과 Valkey fanout 병목을 부하 측정으로 검증

- **여러 게임이 공유하는 채팅 인프라의 용량 산정 기준이 없어 서버 사이징·SLO 목표를 정할 수 없음** — 추측 기반 산정 대신 상용 게임 채팅 솔루션의 공식 rate limit·PCU 기준을 채택해 멀티테넌트 통합 서버 1대 frame 용량 산정식 수립, 6,000 동시접속(sustained)·18,000 RPS·버스트 90,000 RPS 목표 확정 및 분기 OKR 반영
- **Redis Streams 기반 WAL이 6K msg/s 부하에서 싱글스레드 쓰기 병목(EngineCPU 99.9%, ack p95 767ms)** — WAL 스트림을 4-shard 클러스터로 분산하고 엔트리를 native field로 compact(Lua 직렬화 제거) — 싱글스레드 Redis는 쓰기를 순차 처리해 수직 확장으로 병목 해소 불가, 수직 확장 대신 샤딩 채택, ack p95 767ms→6.6ms(약 116배 단축), WAL 적체 80,000→2,838건 해소, 수용률 99.99% 달성
- **다중 테넌트 동시 부하에서 스코프 키 누락으로 메시지가 100% reject됨** — 5개 테넌트 동시 부하 측정 환경을 구성하고 hot-path 로그 제거·메시지 스코프 키 도입으로 reject 제거, 5분간 29.9만 메시지 sustained에서 ack p95 42ms(목표 <100ms)·에러율 0.04%(목표 <0.1%) 달성
- **x86 Fargate가 6,000 동시접속 부하에서 CPU·메모리 사용률이 높아 운영 비용 부담** — 아키텍처 가정 대신 6,000 동시접속 실부하에서 x86 대비 자원 효율을 실측 비교한 근거로 ARM64(Graviton) 전환 — 멀티플랫폼 이미지(x86/ARM64) 빌드로 양 플랫폼 동시 지원, 동일 부하에서 CPU 사용률 15%·메모리 54% 감소, 단가 동등 — 운영 비용 절감 근거 확보
- **hot-channel fanout 전환 후 병목이 Redis 서버 Lua, gateway event-loop, 단일 command queue, topology 중 어디인지 분리되지 않음** — 변경 전 attribution 계측을 먼저 추가하고 Valkey serverless SSUBSCRIBE, 채널별 구독, logicalChannelId sticky connection pool을 단계적으로 검증, sandbox verify 19/0/0 통과, command queue 5,940→716 감소, 878,205 메시지 처리 중 유실 0건 확인
- **오프라인 DM, 미확인수, 참여방 목록, 그룹 fanout을 지원할 유저 단위 영속 상태가 없어 채팅 기능 확장 시 서버·게임 도메인 책임이 섞일 위험** — per-user room index를 기준 모델로 수렴시키고, 채팅 서버는 mechanics/characterId까지만 책임지며 프로필 보강과 그룹 도메인 액션은 게임 서버가 소유하도록 경계 정리, DM·미확인수·참여방 목록·그룹 fanout 서버 기능을 하나의 인덱스 모델로 정리하고 Unity 연동 전 서버/sandbox QA 완료
- **성능 개선 후 gateway task OOM kill이 반복되었으나 outbound queue 누적, 관측 스택, heap cap 중 원인이 불명확** — egress budget 가설을 선등록해 기각하고 heap snapshot·진단 로그·AB 측정으로 retained heap 누적과 heap cap 정합성 문제를 분리, 수정 적용 후 OOM 0건, targeted plan drift 0건 확인으로 관측 스택 누적 원인 제거

기술 선택 이유

메시지 순서 보장을 위해 Redis Streams Consumer Group 기반 WAL(Write-Ahead Log)을 채택하고, 단일 노드 수직 확장 대신 4-shard Valkey 클러스터로 분산 — 싱글스레드 Redis는 코어 증설로 쓰기 병목을 해소할 수 없기 때문. ECS Fargate ARM64 전환은 동일 부하 자원 효율 비교 측정으로 검증 후 채택.

TypeScript · NestJS · Redis Streams · Valkey (Cluster) · Valkey Serverless · DynamoDB · AWS ECS Fargate · ARM64 (Graviton) · NLB · Terraform · k6 · Grafana

마케팅 통합 플랫폼

2025.02 ~ 현재

마케팅 전용 DB 분리 기반 위에 Google Ads/Meta/TikTok 캠페인·소재·지표를 단일 시스템에서 관리하는 통합 마케팅 플랫폼

- **3개 광고 플랫폼 캠페인 관리 분산** — Google Ads/Meta/TikTok 캠페인 생성·배포·수정과 리텐션 지표 관리를 단일 플랫폼에서 처리하는 API 정립, 3개 광고 플랫폼 통합 캠페인 자동화로 마케터 콘솔 전환 작업 제거
- **Meta 에셋 동기화 성능 병목** — 행별 ORM 단건 저장이 트랜잭션 오버헤드를 유발해, 단건 저장 대신 벌크 INSERT·일괄 삭제로 전환, Image 동기화 72% 단축(25.7s→7.1s), DB 트랜잭션 95% 단축(10~16s→0.3~0.5s)
- **마케팅 데이터 조회 비용 급증** — BigQuery Storage Read API 도입으로 gRPC 스트리밍 기반 고성능 데이터 읽기 전환, BigQuery 비용 82% 절감(TB당 $6.25→$1.1)
- **100+ 컬럼 테이블 조회 성능 저하(18초)** — 테이블 3개(메인/시계열/예측) 정규화, 인덱스+커서 페이지네이션 적용, 조회 성능 97% 개선(18s→0.5s)
- **마케팅 지표 아카이브 대용량 백필 중 Lambda 실행 63초·RDS CPU 99.8% 포화로 처리 ETA 15시간** — 근본 원인을 N+1 쿼리와 비SARGable 날짜 조건(DATE() 인덱스 미스)으로 규명 후 제거 — 행별 조회가 RDS를 포화시켜, 행별 조회 대신 30일치를 단일 배치 쿼리로 모아 in-memory에서 최신 버전만 dedup, 처리량 26배 개선(9.3→247건/분), Lambda 실행 63s→1.36s, RDS CPU 99.8%→5~7%
- **마케팅 RCP 데이터 엑셀 전용 + BigQuery 통합 테이블이 복수 소스 조합 구조라 콘솔 통합 조회 불가** — 엑셀 수작업·복수 소스 통합 테이블의 값 불일치 문제로, 복수 소스 결합 대신 단일 정규화 소스 테이블로 전환 + Criteria-Result 패턴 metadata API 신설로 표시 형식 하드코딩 제거, 콘솔 내 RCP 통합 조회 내장, 하드코딩 currency/소수점 표시를 BE metadata 기반으로 대체
- **국가·스토어별 플랫폼 그룹 분리로 동일 캠페인 운영에서 광고계정 선택·검증 흐름이 복잡하고 QA 데이터 복호화 실패가 코드 회귀와 섞임** — BNID/BNKR/BNTW를 단일 플랫폼 그룹으로 병합하고 계정 라벨링 API·계정 선택 UX·sandbox-safe 토큰 재암호화 검증을 분리 수행, 566개 캠페인 기준 Meta 3개·Google 2개·TikTok 2개 계정 매핑 검증, legacy 그룹 잔여 계정 0개 확인
- **TikTok 배포가 외부 API 부분 성공 후 내부 저장 전에 실패하면 재시도 시 고아 campaign/adgroup/ad가 누적되거나 배포 가드에 막힘** — Smart+ V2는 campaign/adgroup/ad 레벨별 get-or-create와 즉시 영속화, v1은 campaign-level 재사용, regular는 ad-level 플래그로 경로별 멱등성 통일, 부분 실패 재시도 경로 단위 테스트 31건 통과, 기존 고아 campaign 2개·adgroup 2개 재발 방지 정책 적용
- **외부 광고 API 범위 확장 시 수동 타입과 런타임 검증 의존이 섞여 campaign/adgroup/ad/creative 호출 레이어의 타입 drift와 라이브 호출 실패 위험 증가** — 공식 Swagger 기반 Orval+Zod codegen을 기본 경로로 두고 spec 공백 API는 manual spec으로 보강해 Smart+·표준 배포·파일·OAuth·creative/playable 경로를 SDK 호출로 전환, 24개 endpoint group 전환 완료 및 preflight 방어선 정리로 외부 API 타입·검증 경로 표준화
- **2,637만 행/27GB audit log 테이블에서 parent_id 조건 인덱스 부재로 화면 조회가 대량 풀스캔과 장시간 IO 포화를 유발** — 엔티티 인덱스 선언으로 재-drift를 막고, 저부하 윈도에 online DDL을 선적용한 뒤 중복 인덱스 no-op 마이그레이션과 실제 DDL 통합 테스트 보강, live 인덱스 실재와 EXPLAIN rows=3 확인으로 audit log 조회 풀스캔 장애 해소

기술 선택 이유

BigQuery Storage Read API로 비용 82% 절감. GCP Pub/Sub → AWS Lambda/SQS 하이브리드 파이프라인으로 이벤트 기반 처리 전환, Outbox 패턴으로 API 비차단 비동기 발행 설계.

AWS Lambda | Apache Arrow | BigQuery | GCP Pub/Sub | Google Ads API | Meta API | MySQL | NestJS | S3 | SQS | BigQuery (Storage Read API) | TikTok API | TypeORM | TypeScript | gRPC | TikTok Smart+ | Idempotency | Orval | Zod

사내 공통 라이브러리 체계

2025.07 ~ 현재

NestJS 유틸 10개 모듈 + 게임서버 Kit 2개 패키지 개발·운영으로 다중 프로젝트 코드 일관성 달성

- **서비스 간 공통 코드 중복 및 유지보수 비용 증가** — core, repository, cache, lock, slack, crypto, smb, hash, type, iac 10개 모듈 라이브러리 체계 설계, 7개 프로젝트 공통 코드 일원화 및 의존성 관리 표준화
- **Repository 계층 코드 비대화(2000+줄) 및 타입 안전성 부족** — read-then-write 방식이 동시 쓰기에서 경쟁 상태(중복 키 충돌)를 유발해, read-then-write 대신 atomic upsert + 감사 로그 자동 분기 구현 — 함수 오버로딩·제네릭으로 타입 좁히기 지원, SRP에 따라 비대한 Repository를 목적별 모듈로 분리, 2000+줄 코드 정리 후 공용 Repository 계층 도입으로 CRUD 중복 제거 및 데이터 접근 표준화
- **멀티 인스턴스 환경의 동시성 제어 부재** — ElastiCache (Redis) 기반 분산 락을 AOP 데코레이터로 추상화하여 비즈니스 로직과 분리, 분산 락 데코레이터로 멀티 인스턴스 동시성 제어 일관 적용
- **7개 프로젝트 패키지 업그레이드 수작업(3시간)** — workflow_dispatch+matrix 병렬 실행, 변경 패키지 CI 테스트 자동화(GitHub Packages), 업그레이드 배포 시간 3h→15m 단축
- **CI 테스트 실행 시간 과다(15m47s)** — @swc/jest는 decorator 메타데이터 호환성이 낮아 @swc/jest 대신 tsc 네이티브 transpiler 기반 ts-jest isolatedModules + Entity 타입 명시 채택, CI 61% 단축(15m47s→6m06s), 81 suites 978 tests 통과
- **게임서버 공통 코드가 5개 브랜치로 분산 + 한 게임은 게임 데이터 정의 모듈 17개가 외부 패키지와 2년+ diverged** — 게임 데이터 정의 모듈+5개 서브모듈을 독립 패키지로 추출 후, 한 게임에 610 파일/310+ import 일괄 마이그레이션 + config 주입·인터페이스 누락 등 9건 hotfix, 패키지 체계 확립 및 첫 게임 적용 — 9건 hotfix 무사고, 이후 데이터 모듈 업데이트는 npm 버전 bump 1건으로 자동 통합(수동 마이그레이션 610 파일→0)
- **게임 서버 공통 캐시 로직이 프로젝트마다 중복 구현되어 정합성·유지보수 부담** — 로컬 LRU(핫패스 지연 최소화)와 분산 캐시(인스턴스 간 일관성)를 결합한 Hybrid 전략을 데코레이터 기반 AOP 라이브러리로 패키지화 — 순수 분산은 네트워크 왕복, 순수 로컬은 인스턴스 간 불일치 문제로 결합형 채택. SRP에 따라 태그 인덱스·single-flight를 별도 모듈로 분리, 캐시 라이브러리 v0.5.0 릴리스 — 28 스위트 262 테스트 통과, 코드 표준 100% 준수

기술 선택 이유

ElastiCache(Redis) 기반 분산 락을 AOP 데코레이터로 구현하여 비즈니스 로직에서 락 코드 분리. Jest 30 VM 격리 대응 시 5가지 옵션 검토 후 poolSize=2 채택.

ElastiCache (Redis) | GitHub NPM | NestJS | Slack API | TypeORM | TypeScript

CS 상담 채팅 서버

2026.04 ~ 현재

게임 운영 고객 상담(CS)용 채팅 서버를 공유 배포 환경에서 ECS로 완전 분리하고, 인증·보안·부하 한계를 검증

- **고객 상담(CS) 채팅이 공유 배포 환경에 묶여 자원 경합·배포 결합 발생** — ECS Fargate로 서비스를 완전 분리하고 Terraform IaC(S3 state)·GitHub Actions OIDC·관측 sidecar로 배포 자동화 — 게임별 분산 토큰 검증 대신 중앙집중 game token 검증 경로 채택, 독립 서비스 cutover 및 Terraform 모듈화, Grafana 11-panel 대시보드·8 알림 구성
- **신규 CS 채팅 도메인이 계층 결합·임의 토큰 수명으로 보안·유지보수 리스크** — Clean Architecture 4계층으로 책임 분리하고 OAuth2 access(5분)/refresh(1시간) 토큰과 ETag/304 폴링을 도입 — 매 폴링마다 전량 응답하는 대신 변경분만 전송, 12 스위트 39 테스트 통과, IaC plan +10 리소스 무사고 적용
- **1.0 vCPU 한계·cold start 40~90초·이미지 비대로 스파이크 트래픽 대응 불가** — task 자원·ALB RPS 오토스케일 정책을 재설계하고 ARM64 native 빌드 + Dockerfile 경량화·마이그레이션 분리로 cold start 단축, 이미지 381→254MB(-127MB)·빌드 5→2분·cold start 30% 단축, k6 500 VU 스파이크 0% 실패
- **프로젝트 간 데이터 격리·요청 rate 제한·첨부 접근 제어 부재** — cross-project 격리(삼중 검증 헬퍼)와 Redis 저장소 기반 rate limit(인증 5/분·refresh 10/분·메시지 30/분), 첨부 FK 정합 마이그레이션을 도입 — 인메모리 throttler는 다중 인스턴스 우회 가능성으로 기각, 보안 통합 머지, 타입 검사 0 에러, cross-org/project 격리 spec 정립
- **CS 채팅방 하나에 여러 문의가 섞여 문의별 건수·처리시간 집계가 어렵고, 버그 문의 동영상 첨부도 지원하지 않음** — 메시지 불변성을 유지하는 답변 링크 테이블, 유저 단위 채팅 리스트, 동영상 업로드 전용 MIME/크기/TTL/일일 제한 경로를 additive API로 설계, 메시지 단위 답변·대상 변경, 유저 단위 리스트, 동영상 문의 열람, 문의 건수·처리시간 집계 기반 구축(20MB·10건/일·15분 TTL)

기술 선택 이유

공유 배포 환경의 자원 경합·배포 결합을 끊기 위해 ECS로 독립 분리(Terraform IaC + GitHub Actions OIDC)하고, 게임별 분산 토큰 검증 대신 중앙집중 game token 검증 경로를 채택. rate limit은 다중 인스턴스 정합을 위해 Redis 저장소 기반 throttler로 구현.

TypeScript NestJS TypeORM OAuth2 AWS ECS Fargate ARM64 (Graviton) Terraform GitHub Actions OIDC Redis k6 Grafana Presigned URL Redis Rate Limit

AWS Lambda 마이그레이션 & Event-Driven Architecture

2025.06 ~ 2025.08

마케팅 지표 60일 → 360일 확장에 따른 배치 작업 한계 해결 및 배치 처리 서버리스 전환

- **전체 서버리스 이관 시 운영 안정성 리스크** — API 서버(EC2)는 유지하고 배치/Job 처리만 Lambda로 분리하는 하이브리드 아키텍처 설계, API 가용성 무영향 보장, 배치 워크로드만 격리 이관해 운영 리스크 차단
- **배치 처리의 수집 기간 한계(60일) 및 처리 시간 2시간** — SQS+Lambda+EventBridge 기반 Event-Driven 비동기 처리 전환, 처리 시간 2h→5m 단축, 수집 범위 6배 확장(60일→360일)
- **비동기 전환 시 데이터 정합성 보장 필요** — 예약·지연 발행 기반 트랜잭션 아웃박스 패턴 적용, 이벤트 기반 아키텍처에서 데이터 정합성 보장
- **CDK deploy workflow에서 cache disk full 이후 Lambda handler webpack build V8 OOM이 재현되어 배포 속도 저하와 재발 방지 부재가 동시에 발생** — runtime transform 의존을 generate 산출물로 분리하고, 기본 handler batch는 키우되 실패 batch만 8→4→2→1로 adaptive split 재시도하도록 build 경로 개선, Lambda handler build OOM 원인 제거와 CI 재발 방지 게이트 정리, 3개 CI run 통과로 배포 경로 검증
- **Lambda 대량 실행 시 DB 커넥션 고갈** — RDS Proxy 풀링 기반 커넥션 관리 도입, DB 커넥션 고갈 문제 해결

기술 선택 이유

배치·이벤트성 작업이 서버 본체에 묶여 독립 배포 불가한 제약 해소를 위해 Lambda 선택. SQS로 비동기화하여 트래픽 변동 시 처리 지연과 확장성 한계 해결.

TypeScript NestJS AWS Lambda SQS SNS EventBridge RDS Proxy AWS CDK

게임 인앱 다국어 번역 시스템

2026.02 ~ 현재

게임 내 알림/공지 다국어 번역 체계의 도메인 모델 재설계 — AI 번역과 라이브러리 기반 변환을 3단계 파이프라인으로 분리

- **간체는 AI 지원 언어가 아님에도 AI 번역 설정 도메인 내부 옵션으로 모델링되어 도메인 의미 왜곡, 설정 저장·조회가 번체 AI 번역 상태에 의존하여 breaking change 없이 독립 on/off 불가** — 파생 변환 모듈을 AI 번역 모듈에서 독립 분리(9개 use case + 전용 entity/repository/scheduler), effective 설정 글로벌 상속 패턴 적용, 9개 use case 분리로 도메인 경계 명확화, 파생 변환 규칙 독립 토글 지원
- **설정 decoupling 과정에서 breaking API change 필요** — FE/BE/Gateway 동시 배포 전략과 역할 권한 마이그레이션으로 무중단 전환, breaking API change 장애 없이 전환 완료, effective 설정 DB 쿼리 50% 감소(4→2)
- **detect 크론의 버전 우선순위 부재로 최신 버전 변환 지연 + 30초 주기 응답 지연** — detect SQL에 버전 ID 기반 우선순위 정렬 추가, 처리 주기 30초→10초, 배치 limit 100→200 최적화, 최신 버전 변환 지연 해소, 진행률 UI와 총 건수 캐시로 사용자 가시성 확보
- **detect 스케줄러가 159개 버전을 한 번에 스캔(3.27M rows full scan) → RDS IOPS 포화로 사용자 로그인 60초 장애** — AI detect 모듈의 per-version loop 이식 + 분산락 timeout/recovery + NOT EXISTS full scan을 인덱스 staged lookup으로 대체, 쿼리 260× 단축(전체스캔→39ms 인덱스 lookup), driving filesort 제거, 사용자 로그인 latency 정상 복원

기술 선택 이유

AI 번역 도메인과 라이브러리 기반 파생 변환의 책임 경계를 코드 구조에 실체화. Detection→Processing(AI)→Conversion(라이브러리) 3단계 파이프라인을 독립 모듈로 분리하고, 2단계 race guard와 pessimistic write lock으로 동시 요청 정합성 확보

TypeScript NestJS TypeORM MySQL opencc

마케팅 플랫폼 Audit Log 시스템

2025.01 ~ 2025.04

게임 운영 중 데이터 변경 이력 추적 불가로 인한 밸런스 문제 대응 지연 해결

- **6개 게임 간 데이터 변경 이력 추적 불가** — UUID 기반 크로스 환경/프로젝트 엔티티 추적 Git-like 버전 관리 시스템 정립, 6개 게임 간 데이터 일관성 보장
- **엔티티 변경 기록의 수동 관리 부담** — Auditable 데코레이터 + TypeORM Subscriber 패턴으로 이벤트 소싱 기반 변경 자동 기록 파이프라인 설계, 엔티티 변경 이력을 자동 추적하고 모든 게임 환경에서 데이터 변경을 일관되게 기록
- **다중 환경 간 엔티티 충돌 감지 및 병합 부재** — 상위/하위 엔티티 충돌 감지와 유니크 제약조건을 고려한 3-Way Merge 엔진 구현, 6개 게임 다중 환경 간 버전 병합 자동화
- **버전 간 변경 사항 비교 비효율** — Base Audit 유무에 따른 증분 비교와 스냅샷 비교의 이중 전략 Diff 엔진 설계, 증분 및 스냅샷 이중 전략으로 버전 비교 성능 최적화
- **PK 의존 엔티티 추적의 환경 간 불일치** — 공유 식별자 기반 크로스 환경 엔티티 추적 체계 설계로 PK 의존 탈피, 마이그레이션/비교/병합 시 정확한 엔티티 추적 보장

> **기술 선택 이유**
> TypeORM EntitySubscriberInterface 동작과 제약을 별도 분석 후 채택. Auditable 데코레이터 + Subscriber 조합으로 엔티티 변경을 표준화된 Audit 파이프라인으로 자동 수집하는 AOP 방식 설계.

TypeScript NestJS TypeORM MySQL

Cloud Data 동기화 시스템

2025.08 ~ 현재

환경 간 동적 게임 데이터 불일치 문제를 해결하기 위한 S3 기반 동기화 및 DDL 자동 관리 시스템

- **환경 간 동적 게임 데이터 불일치** — S3 기반 개발/스테이징/프로덕션 환경 간 동기화 시스템 구축, 환경 간 데이터 일관성 보장
- **환경 간 스키마 불일치 수동 관리 부담** — PK 컬럼 타입 동적 결정, 컬럼 타입 불일치 감지·MODIFY, 인덱스 자동 생성·RENAME DDL 자동 관리 엔진 구현, 데이터베이스 스키마 변경사항 자동 추적 및 동기화
- **Cloud Data 적재 및 S3 업로드 성능 병목** — 병목 구간 분석 및 저장/업로드 최적화, TRUNCATE→DELETE 전환으로 대용량 데이터 처리 시간 77% 단축 (57.5초→12.9초)
- **동기화 작업 불안정(타임아웃, 스케줄링 충돌)** — job 분리, 스케줄링 방식 전환, timeout 조정, non-blocking 처리 적용, 마이그레이션 및 스키마 최적화로 데이터 계층 안정성 강화
- **데이터 동기화 서비스 운영 장애 4건(캐시 정합·메타데이터·복사 키·데이터 보호 옵션 누락)** — Redis 캐시 무효화, DDL SKIP 메타데이터 정리, 복사 키 삭제, 데이터 보호 옵션 전달 누락 수정, 4건 장애 원인 분석·수정으로 데이터 동기화 서비스 안정성 확보
- **데이터 마이그레이션 시 보호 옵션(클라우드 보관 데이터 제외)이 일부 경로에 미전파되어 전체 삭제 리스크** — 누락된 POST 경로 4곳에 보호 옵션 전달 보강, 운영 데이터 전체 삭제 리스크 차단

> **기술 선택 이유**
> S3 Lifecycle 정책(30일 Glacier IR, 90일 만료)으로 비용 최적화. 이벤트성 실행에서 스케줄링 방식으로 전환하여 동기화 누락 위험 감소.

TypeScript NestJS TypeORM MySQL S3

확률 계산 및 감사 로그 분석 파이프라인

2026.02 ~ 현재

게임 확률 검증 체계 부재에 따른 규제 리스크 해소를 위한 확률 계산 패키지와 CDK 기반 감사 로그 분석 인프라 개발

- **6개 라이브 게임 환경 (12 브랜치 운영) 확률 계산 로직 분산 및 규제 대응 부재** — NestJS DynamicModule 기반 5개 확률 함수+Kinesis 로깅을 확률 계산 공통 패키지로 분리, 6개 라이브 게임 환경 (12 브랜치 운영) 확률 패키지 통합 (cherry-pick main + 11 branches)
- **확률 감사 로그 분석 인프라 부재** — Kinesis→Firehose(Dynamic Partitioning+Parquet)→S3→Glue→Athena E2E 파이프라인 CDK 코드화, 6개 라이브 게임 환경 (12 브랜치 운영) 프로젝트에 통합하여 확률 공시 감사 로그 파이프라인 구축
- **모킹 기반 테스트의 회귀 신뢰도 부족** — 모킹 삭제, LocalStack+Testcontainers 기반 통합 테스트 교체, 회귀 신뢰도 강화, 12 suites 115 tests 커버리지 94%+ 확보

> **기술 선택 이유**
> Kinesis → Firehose(Dynamic Partitioning + Parquet) → S3 → Glue → Athena 파이프라인을 CDK로 코드화. Parquet 컬럼형 포맷으로 Athena SQL 분석 비용 최적화, LocalStack Testcontainers로 모킹 기반 테스트 제거.

기여도: *초기 구현체를 패키지 구조로 재설계. P0 버그 수정, 테스트 안정화(94%+ 커버리지), CDK 인프라 구축, 6개 라이브 게임 환경 (12 브랜치 운영) 적용은 단독 수행.*

TypeScript NestJS AWS CDK Kinesis Firehose S3 Athena Glue Parquet LocalStack Testcontainers

마케팅 데이터 아카이빙 & 커스텀 대시보드

2026.04 ~ 현재

마케팅 지표의 일별 스냅샷 아카이빙(불변성 보장), 4천 행 규모 커스텀 대시보드, 알림 관측성을 구축

- **4천 행 규모 마케팅 메트릭을 단일 대시보드에 표시 시 스크롤 성능 저하·셀 동기화 버그** — BE는 Raw SQL DISTINCT JOIN(ORM hydration 회피)+in-memory 캐시, FE는 가시 행 lazy fetch+200행 단위 청킹(네트워크 왕복과 렌더 지연의 균형점)+pair별 캐싱으로 분리 — 전량 로딩은 응답 크기·렌더 비용으로 기각, 4,458행 중 3,341행 매핑 표시, 응답 cold 573ms/warm 221ms로 안정화
- **동기화 경로가 reference-date cutoff 대신 최신 스냅샷을 사용해 아카이브 불변성 위반(데이터 오염)** — cutoff semantics를 준수하도록 usecase·repository를 재설계하고 stale-writer guard를 트랜잭션 처리, 동기화 상태에 tombstone을 도입, live 15,333개 아카이브 재구축(오염 0), 287,020개 archive-date 검증 통과
- **로그 쿼리 스택(Loki) 일시 장애 시 마케팅 알림 16건이 동시 발화해 운영 채널 도배** — 장애 원인을 규명한 뒤 알림 16건을 단일 분석 DB(ClickHouse) 쿼리로 전환하고 대시보드 API로 재작성, 레거시 로그 스택 폐기 정책 수립, 알림 16/16 마이그레이션(25분), 로그 스택 장애 시 알림 도배 재발 방지

기술 선택 이유

아카이브는 조회 시점이 아닌 reference-date cutoff 기준의 불변 스냅샷이어야 하므로 stale-writer guard와 tombstone 상태로 무결성을 강제. 대용량 그리드는 전량 로딩 대신 가시 영역 lazy fetch + chunk 캐싱으로 처리.

TypeScript NestJS React AWS Lambda BigQuery ClickHouse Grafana MySQL

외부 활동

Amazon Q Developer를 활용한 개발 생산성 향상 외부 발표 2025.10

Games on AWS 2025 고객 세션 발표

개발자 1인이 10일 만에 시간당 병렬 처리 Job 수 기준 데이터 파이프라인 용량 270배 확장한 사례 공유

제이앤피메디

개발2팀 Docs Squad Backend Engineer

2023.06 — 2024.08

누적 160억 투자 유치 임상시험 데이터 관리 솔루션 스타트업 (MSA 기반)

주요 성과

- 월 5건+ 발생하던 전자서명 처리 실패와 임상 데이터 정합성 리스크. FK 공유 락 교착 패턴을 재현·규명하고 트랜잭션 처리 경계를 조정해 동일 락 교착 재발을 방지. 전자서명 처리 실패 월 5건+ 해소.
- VDR 프로젝트 일정 지연 및 MVP 출시 압박. BE 2명 + FE 1명 팀으로 3주 긴급 투입, Event-Driven PDF 변환과 권한 기반 워크플로우 구현. MVP 출시 일정 준수.
- 이메일 발송 실패 사후 인지(수시간 지연). SES+SNS 이벤트 파이프라인 구축. 반송·바운스 감지 수시간→수초 단축, Slack 즉시 알림 체계 수립(기술 블로그 게재).

프로젝트

Maven Docs 전자 동의서 시스템

2023.07 ~ 2023.09

Cyan(사내 Node.js 프레임워크) 기반 임상시험 전자 동의서 수집 비효율성과 전자서명 규제 준수 요구사항 해결

- **임상시험 동의서 개별 발송의 비효율** — 임상시험별 참여자 그룹 관리 및 일괄 동의서 배치 발송 시스템 구현, 동의서 발송 프로세스 자동화
- **배치+S3 이벤트 기반 아키텍처의 이벤트 흐름 비가시성** — EDA+아웃박스 패턴으로 변경, 이벤트 흐름 가시화 및 로컬 테스트 환경 정비

> **기술 선택 이유**
> 동기 서비스 간 결합도 제거를 위해 EDA + 아웃박스 패턴으로 전환. 텍스트·체크박스·서명 등 입력 방식 확장을 위해 플러그인 구조 설계.

기여도: *EDA 전환 설계, 플러그인 구조 구현, 대상자 일괄 등록 + Signer 테이블 분리 마이그레이션 수행. BE 2명 체제.*

TypeScript | Express.js | Cyan | Aurora Serverless v2 | Knex.js

Maven Docs 시스템 안정성 개선

2023.08 ~ 2023.12

임상시험 전자서명 알림에서 발생하는 데이터베이스 Deadlock으로 인한 치명적 알림 누락 해결

- **월 5건+ 발생하던 운영 환경 Deadlock으로 전자서명 알림 누락** — 외래키 생성 시 공유락→배타락 승격 타이밍 이슈를 INNODB 잠금 테이블로 규명, SELECT FOR UPDATE 선점 적용, 전자서명 처리 실패 0건 해소, 락 교착 패턴 근본 원인 제거
- **동기 처리 방식의 알림 병목 및 장애 전파** — AWS SNS → SQS → Lambda Event-Driven Architecture 설계, 비동기 파이프라인 완성으로 알림 안정성 확보

> **기술 선택 이유**
> FK child INSERT 시 S-Lock → X-Lock 승격 교착 패턴이 근본 원인. INSERT 전 SELECT FOR UPDATE로 X-Lock 선점하여 교착 순서 자체를 제거.

기여도: *Deadlock 재현·분석·해결 방안 도출 단독 수행. DBeaver로 교착 시나리오를 SQL로 재현하고 INNODB 잠금 테이블로 확인.*

TypeScript | Express.js | Cyan | Slate.js | SNS | SQS | Lambda | React

Maven Mailing 시스템 고도화

2023.12 ~ 2024.02

이메일 발송 시스템의 발송 상태 추적 부족과 데이터 정합성 문제 해결

- **이메일 발송 실패 사후 인지(수시간 지연)** — SES 구성세트 → SNS → 이벤트 처리 파이프라인으로 발송/반송/바운스 상태 추적, Datadog 로깅 + Slack 실시간 알림 정착, 반송·바운스 감지 수시간→수초 단축

> **기술 선택 이유**
> AWS SES 비동기 특성상 발송 성공 여부를 동기적으로 수신 불가. 구성세트에 SNS 이벤트 대상을 추가하여 발송/반송/바운스 이벤트를 비동기 구독.

기여도: *SES 구성세트 → SNS → 이벤트 처리 파이프라인 설계·구현. 기술 블로그 게재 및 사내 세미나 발표 주도.*

TypeScript | Express.js | AWS SES | Nodemailer

Maven Auth 시스템 고도화

2024.01 ~ 2024.02

기존 1:1 사용자-조직 구조로 인한 사용자 불편과 시스템 확장성 한계 해결

- **1:1 사용자-조직 구조의 확장성 한계** — 1:N 구조로 관계 확장하여 다중 조직 관리 지원, 다중 조직 관리 기능 제공
- **구조 변경 시 기존 API 호환성 단절 리스크** — 기존 API 무중단 마이그레이션 설계, 서비스 연속성 유지

> **기술 선택 이유**
> 임상시험 도메인 특성상 연구자가 여러 조직에 동시 참여하는 요구사항 수용을 위해 계정-조직 1:N 재설계. 로그인 후 조직 선택 흐름 추가로 컨텍스트 전환 지원.

기여도: *스키마 마이그레이션, 인증 흐름 수정, 하위 서비스(Docs/Billing/Mailing) 계정-조직 참조 로직 일괄 수정까지 풀스택 변경.*

TypeScript | Express.js | Cyan | Aurora Serverless v2

Maven TMF 신규 프로젝트 개발 (프론트엔드)

2023.06 ~ 2024.08

임상시험 필수 문서(TMF) 관리 시스템의 프론트엔드 신규 개발

- **임상시험 필수 문서(TMF) 관리 시스템 부재로 규제 준수·문서 추적 비효율** — BE 1·FE 2(본인 포함) 팀에 긴급 투입, MVP 범위 정의 후 Admin/Dashboard에 문서 업로드·분류·버전 관리·감사 추적(Audit Trail) 집중 개발, 3개월 내 TMF 관리 프론트엔드 MVP 완성 — 임상시험 필수 문서 체계적 관리 기반 구축

> **기술 선택 이유**
> 3개월 내 출시 일정 제약으로 핵심 기능(문서 업로드·분류·버전 관리·감사 추적)만 MVP 범위로 정의하고 집중 개발.

기여도: *BE 1명 + FE 2명(본인 포함) 체제에서 FE 담당. Admin/Dashboard 페이지의 문서 업로드·분류·감사 추적 기능 구현.*

React TypeScript Jotai

Maven VDR MVP 개발

2024.06 ~ 2024.07 (3주)

진행 중이던 VDR 프로젝트의 개발 일정 지연과 MVP 출시 압박 상황에서의 긴급 투입

- **대용량 PDF 변환 동기 처리로 타임아웃 발생** — S3 업로드 트리거 → Lambda 경량 변환 → SNS/SQS 알림 EDA 파이프라인 설계, 동기 변환 타임아웃 해소, 대용량 문서도 비동기 처리로 안정화
- **의료진별 문서 접근 권한 체계 부재** — 역할 기반 문서 접근 및 승인 권한 관리 워크플로우 정의, 역할별 문서 접근·승인 워크플로우 정립
- **VDR 프로젝트 일정 지연 및 MVP 출시 압박** — 3주 긴급 투입, 핵심 기능 우선순위 선정 및 집중 개발, 3주 내 MVP 출시 달성 (BE 2명 + FE 1명 체제)

> **기술 선택 이유**
> 동기 PDF 변환 시 타임아웃 발생 문제를 구조적으로 해결하기 위해 EDA 기반 비동기 변환 파이프라인(S3 트리거 → Lambda 변환 → SNS/SQS 알림) 설계.

기여도: *BE 2명 중 1명. EDA 기반 문서 처리 파이프라인 설계·구현, 대용량 PDF 비동기 변환 워커 구현. 3주 긴급 투입으로 MVP 출시 달성.*

TypeScript Lambda S3 SNS SQS

Maven Billing 구독 관리 시스템

2023.06 ~ 2024.08

Maven 전체 서비스의 구독/플랜/라이센스 관리 시스템 개발

- **조직별 맞춤 요금제 제공 불가** — ORG별 커스텀 플랜 Internal API 구현, 조직별 맞춤 요금제 제공 체계 구축
- **갱신 불가 플랜의 구독 갱신·재구독 정책 미적용** — 갱신 불가 플랜 구독 갱신·재구독 차단 로직 작성, 비즈니스 정책 정합성 보장

> **기술 선택 이유**
> 구독 만료 후 비동기 정리 처리를 위해 event queue 도입. 갱신 불가 플랜의 재구독 차단을 bundles.is_renewable 스키마 속성으로 비즈니스 정책을 데이터 레벨에서 강제.

기여도: *ORG별 커스텀 플랜, event queue 기반 인벤토리 자동 삭제, 갱신 가능 여부 판별 로직 구현. 다수 PR 직접 기여.*

TypeScript Express.js Cyan Aurora Serverless v2 Knex.js

외부 활동

AWS SES 이벤트 로그를 통한 이메일 발송 모니터링 기술 블로그 — 2024.03

AWS SES 이벤트 로그 기반 이메일 발송 모니터링 시스템 개발 사례 공유

메일링 시스템 이메일 발송 결과 수신 기능 도입 사내 세미나 — 2024.02

메일링 시스템 이메일 발송 결과 수신 기능 도입 과정 및 아키텍처 공유

AWS SAA 스터디 스터디 — 2023.09 ~ 2023.12

제이앤피메디 사내 스터디, 클라우드 아키텍처 심화 학습

메드고

개발팀 Backend Engineer

2022.04 — 2023.06

누적 30만 다운로드 비대면 진료 및 약배달 플랫폼 스타트업

주요 성과

- Express.js 단일 파일(app.js) 중심 구조로 인증·예약·알림 기능 변경 시 회귀 위험 증가. 3단계 점진적 전환(단일 파일→레이어드→NestJS) + JS→TS 마이그레이션. TDD 80% 달성, 주요 릴리스 구간 장애 0건 유지.
- 수동 처방전 입력 병목(약사 건당 3~5분). Naver Clova OCR + 식약처 API 연동으로 처방전→복약지도 자동화. 약사 처리 시간 90% 단축(3~5분→30초).
- 진료·조제·배달 상태 반영 지연(수십 초). Socket.IO 실시간 동기화 + Web Push 알림 도입. 상태 반영 1초 미만으로 단축, 환자 대기 불만 해소.

프로젝트

Auth 서버 신규 개발

2022.04 (3주)

서브도메인 간 세션 충돌 문제를 해결하기 위한 JWT 기반 통합 인증 시스템 신규 개발

- **서브도메인 간 세션 충돌로 인한 인증 불안정** — NestJS + JWT 기반 토큰 인증 시스템 설계 및 구현, 웹(의사/약사/관리자) + 모바일 앱 통합 인증 정립
- **세션 방식의 서브도메인 간 사용자 전환 불가** — 세션→토큰 방식 마이그레이션, 서브도메인 간 원활한 사용자 전환 구현

> **기술 선택 이유**
> 서브도메인 간 쿠키 기반 세션 충돌을 JWT 무상태 토큰으로 구조적 해결. Redis 대신 MySQL 키 관리 — 트래픽 규모 대비 Redis는 오버엔지니어링으로 판단하여 인프라 단순화.

기여도: *3주간 단독 설계·구현. 웹(의사/약사/관리자 포털) + 모바일 앱 통합 인증 체계 전반.*

NestJS TypeScript JWT MySQL

공통 모듈 개발

2022.05 ~ 2022.06

서비스별 독립 관리로 인한 코드 중복과 일관성 부족 문제를 공통 모듈 통합으로 해결

- **서비스별 권한 검증 로직 중복 및 보안 일관성 부족** — 사용자 권한 확인 미들웨어 공통 모듈화, 보안성 강화 및 권한 검증 일원화

NestJS TypeScript

의사/약사 웹 서비스 고도화

2022.04 ~ 2023.05

의사·약사용 백오피스 시스템의 실시간 상태 반영 부재와 수동 알림 체계 문제를 Socket.io 및 Web Push 도입으로 해결

- **중복 예약 발생** — 의사ID+시간대 복합 유니크 키 적용, 중복 예약 원천 차단
- **진료·조제·배달 상태 반영 지연(수십 초)** — Socket.IO Room 기반 실시간 상태 동기화로 이벤트 즉시 반영, 상태 반영 1초 미만으로 단축, 환자 대기 불만 해소
- **외부 서비스(배달·결제) 장애 전파 리스크** — 배달(후다닥) 및 결제 외부 서비스 통합 API 게이트웨이 설계, Circuit Breaker 패턴 적용, 외부 서비스 장애 전파 차단

> **기술 선택 이유**
> Socket.IO Room 단위 이벤트 격리로 진료·조제·배달 상태 실시간 동기화. 외부 서비스 장애 전파 방지를 위해 API 게이트웨이 + Circuit Breaker 통합. DB 복합 유니크 키로 중복 예약을 데이터 레벨에서 차단.

기여도: *BE 2인 체계에서 Socket.IO 실시간 동기화, Web Push 알림, API 게이트웨이, 진료 예약 중복 방지 전 기능 직접 구현.*

Express.js NestJS TypeScript EJS MySQL Socket.io Web Push

처방전 자동 인식 및 복약지도 시스템

2022.11 ~ 2022.12

약사의 수동 처방전 입력 과정을 OCR 연동으로 자동화

- **처방전 수동 입력 병목** — Naver Clova OCR API 연동으로 처방전 이미지에서 텍스트 자동 추출, 처방전 입력 자동화
- **복약지도 수동 작성 비효율** — 공공데이터(식약처 API) 의약품 정보 활용한 자동 복약지도 생성, 약사 건당 처리 시간 90% 단축(3~5분→30초)

> **기술 선택 이유**
> 한국어 의약품 용어 인식률을 고려하여 Naver Clova OCR 선택. 의약품 정보 DB 자체 구축 대신 식약처 공공데이터 API 활용으로 유지보수 부담 제거.

기여도: *2개월간 단독 구현. 이미지 수신 → OCR 추출 → 식약처 API 매칭 → 복약지도 생성 파이프라인 전 구간.*

NestJS TypeScript Naver Clova OCR MySQL

백엔드 아키텍처 전환

2023.03 ~ 2023.05

Express.js 단일 파일(app.js) 구조에서 NestJS 모듈 아키텍처로 전환

- **Express.js 단일 파일(app.js) 10,000줄 레거시 유지보수 한계** — 3단계 점진적 이관: app.js 단일 파일 → 레이어드 아키텍처 (Controller/Service/Repository) → NestJS 프레임워크 도입, 10,000줄 단일 파일을 NestJS 모듈로 무장애 이관, 주요 릴리스 구간 장애 0건 유지
- **JavaScript 기반 코드의 타입 안정성 부족** — JavaScript → TypeScript 마이그레이션, 타입 안정성 강화
- **테스트 부재로 인한 회귀 리스크** — TDD 도입 및 단위 테스트 작성, 테스트 커버리지 80% 달성

기술 선택 이유

Express 단일 app.js 10,000줄 레거시의 유지보수 한계 해결을 위해 NestJS DI·모듈 시스템 도입. 3단계 점진적 전환(단일 파일 → 레이어드 → NestJS)으로 리스크 최소화. TDD 도입으로 전환 중 회귀 방지.

기여도: *단독 주도. 3단계 전환 전 구간, JS→TS 마이그레이션, TDD 도입, 팀 내 코드 컨벤션 확립 모두 직접 수행.*

NestJS TypeScript TypeORM Jest MySQL

심플한

개발팀 Backend Engineer

2021.07 — 2022.03

고객 맞춤형 소프트웨어 솔루션 기업

주요 성과

- 학습 관리(출석, 수료, 진도율) 수작업 의존. LMS 풀스택 개발(QR 출석, PDF 수료증, Excel 일괄 등록). 학기당 300명+ 수강생 관리 전 과정 디지털화.
- 레거시 PHP 시스템에 신규 매칭·상담 기능을 추가해야 하나 기존 코드 변경 리스크가 큼. 독립 Spring Boot 앱을 제한된 iframe/postMessage 경계로 통합하고 가중 평균 매칭 알고리즘 구현. 레거시 영향 범위를 분리한 상태로 JOB Agent 서비스 출시.

프로젝트

학습 관리 시스템(LMS) 개발

2021.07 ~ 2022.03

교육 기관의 학습 관리(출석, 수료, 진도율)가 수작업에 의존하여 운영 효율이 낮고 오류가 빈번한 문제 해결

- **출석 관리 수작업 및 중복 출석 발생** — 날짜+사용자ID 복합 유니크 키로 중복 방지하는 QR 기반 실시간 출석 처리 API 구현, 출석 관리 자동화 및 중복 출석 원천 차단
- **수강생 개별 등록의 비효율** — Apache POI 기반 행별 순차 처리로 데이터 정합성을 보장하는 Excel 일괄 등록 시스템 개발, 학기당 300명+ 수강생 일괄 등록 지원
- **수료증 수동 발급 부담** — JasperReports 기반 동적 PDF 수료증 자동 생성 API 구현, 수료증 발급 자동화

> **기술 선택 이유**
> 교육기관 납품 프로젝트로 Java/Spring Boot 표준 선택. QR 출석 중복 방지를 DB 레벨 복합 유니크 키(날짜+사용자ID)로 원천 차단. JasperReports 템플릿 기반 PDF 수료증 대량 발급.

기여도: *10개월간 핵심 기능 직접 구현. QR 출석, Excel 일괄 등록, PDF 수료증, 진도율 계산 등. 6인 팀 풀스택 체계에서 백엔드 전담.*

Java | Spring Boot | MyBatis | MySQL | Apache POI | JasperReports

인천 스마트 그린 산단 통합관제센터

2022.01 ~ 2022.03

산단 관제센터의 회의실 예약·모니터링·SMS 알림이 개별 시스템으로 분산되어 통합 관리가 불가능한 문제 해결

- **회의실 예약 충돌 발생** — 시간+장소 복합 유니크 키로 예약 충돌 방지하는 실시간 예약 API 구현, 예약 충돌 원천 차단
- **예약 확정 알림 동기 처리로 인한 응답 지연** — 비즈뿌리오 SMS API + @Async 비동기 처리 적용, 예약 확정 SMS 알림 비동기 발송 체계 구축

> **기술 선택 이유**
> 회의실 예약 충돌을 DB 복합 유니크 키(시간+장소)로 원천 차단. SMS 발송 시 API 응답 지연 해결을 위해 @Async 비동기 처리 — 트래픽 규모 대비 별도 메시지 큐는 오버엔지니어링으로 판단.

기여도: *4개월간 백엔드 전담. 예약 API, 대시보드 모니터링 API, SMS 비동기 연동 직접 구현.*

Java | Spring Boot | MySQL | 비즈뿌리오 SMS API

미래 서비스 JOB Agent 개발

2021.07 ~ 2022.03

기존 PHP 시스템의 확장성 한계로 새로운 기능 추가가 어렵고, 점수 계산과 상담 관리가 체계화되지 않은 문제 해결

- **레거시 PHP 시스템의 확장성 한계** — iframe 기반 Spring Boot 독립 앱을 기존 PHP 시스템에 임베드하여 레거시 영향 최소화, 레거시 시스템 영향 없이 신규 기능 확장
- **다차원 평가 점수 산정 체계 부재** — 가중 평균 기반 다차원 평가 점수 산정 알고리즘 구현, 복합 점수 계산 자동화

> **기술 선택 이유**
> 레거시 PHP 시스템 비침습 확장을 위해 독립 Spring Boot 앱을 iframe으로 임베드. postMessage API로 크로스 프레임 통신. 다차원 평가를 단일 점수로 산출하기 위한 가중 평균 알고리즘 구현.

기여도: *10개월간 백엔드 전담. iframe 통합 구조 설계, 가중 평균 알고리즘, postMessage 상태 동기화, 상담사 백오피스 직접 구현.*

Java | Spring Boot | MySQL

kfriends 회원 관리 시스템

2021.09 (3주)

5개국 분산 웹사이트에서 국가별 개별 로그인이 필요한 통합 인증 부재 문제 해결

- **5개국 분산 웹사이트의 국가별 개별 로그인 필요** — Spring Security + JWT 기반 중앙화된 통합 인증 서버 정립, 5개국 분산 도메인에 단일 인증 적용 완료

> **기술 선택 이유**
> 5개국 분산 웹사이트의 개별 로그인 문제를 JWT 기반 중앙 통합 인증 서버로 해결. CORS 다중 도메인 처리로 각국 도메인에서 동일 서버 호출 허용.

기여도: *3주간 단독 구현. 통합 인증 서버 전체.*

Java | Spring Boot | Spring Security | JWT | MySQL

사내 기술 블로그 리뉴얼

2021.09 (3주)

WordPress 기반 블로그의 호스팅 비용 부담과 수동 배포 문제 해결

• **WordPress 호스팅 비용 부담 및 수동 배포** — WordPress → Jekyll 정적 사이트 마이그레이션, 호스팅 비용 100% 절감

Jekyll Liquid Travis CI GitHub Pages

개인 프로젝트 (사이드)

플랫폼 · 툴링 엔지니어링

2026.04 — 현재

직무 외 자기주도 플랫폼/툴링 엔지니어링 — 개발 워크플로우 자동화 프레임워크 및 사이드 프로젝트

프로젝트

개인 개발 운영 하네스 프레임워크

2026.04 ~ 현재

개발 세션의 컨텍스트·비용·규칙·RAG·브라우저 런타임을 자동 관리하는 개인 워크플로우 프레임워크를 설계하고 3개 하네스에 배포·운영

- **마크다운 지식 베이스가 커지며 에이전트가 관련 컨텍스트를 수동 탐색하는 비효율** — 클라우드 벡터 DB(Pinecone 등)의 동기화 지연·비용 대신 로컬 임베딩(BGE-M3) 벡터 RAG를 도입해 마크다운 변경 시 즉시 자동 인덱싱 — sqlite-vec+FTS5 하이브리드(RRF fusion)로 정확도 확보, MCP로 에이전트가 직접 검색, 548 파일·3,347 청크 인덱싱, warm 쿼리 0.7초, 골드 쿼리 5건 모두 top-2/3 내 적중
- **RAG가 opt-in 참고 도구라 조사·검토 프롬프트에서 근거 검색이 누락** — UserPromptSubmit 기본 트리거로 전환하고 프롬프트 12자 이상이면 search_harness가 자동 호출되도록 3개 하네스에 포팅, 3개 하네스 15/15 default-on 테스트 통과, 실 프롬프트 smoke test에서 1회차 검색 적중
- **이력서·포트폴리오 JSON이 RAG 검색 범위 밖이라 JD 커스터마이징 시 근거 검색 누락** — portfolio/resume JSON chunker, target=resume 색인 경로, path-based auto-reindex, federation 검색 구현, resume corpus 109 청크 색인, 32개 테스트 통과, 4개 DB 통합 검색 동작 확인
- **AI 에이전트 토큰 비용의 95%가 시스템 프롬프트·서브에이전트 등 숨은 영역에서 발생해 추적 불가** — 단순 session/model 집계로는 숨은 비용을 추적 못 해, 단순 집계 대신 세션 트랜스크립트를 8개 카테고리로 분해하는 비용 분석기 구현 — residual 추적으로 다중 머신·다중 도구 사용량을 단일 프로파일로 통합, 숨은 비용 95%를 카테고리별로 가시화, 다중 인스턴스 사용량 집계 누락 버그 발견·수정
- **브라우저와 REPL 런타임의 신뢰 경계·fallback 순서가 불명확해 자동화가 세션별로 불안정** — 런타임 신뢰 경계와 environment-native Browser/Chrome fallback 순서를 문서화하고, 런처 생성기에 Chrome 플러그인 활성화 경로 반영, kh/gp/gd 3개 하네스에 런타임 등록, browser runtime 신뢰 경계 및 fallback 순서 정립

기술 선택 이유

문서 전용 규칙은 에이전트가 필요 시점에 읽지 않으므로 Git 훅 기반으로 강제(hook-first)하고, 다중 인스턴스 중복 관리·버전 드리프트를 막기 위해 런타임 서피스 컴파일러와 런처 패키지로 설정을 생성·검증.

TypeScript Python Bash MCP Ollama (BGE-M3) Vector RAG Git Hooks Homebrew

채용 정보 자동 수집 플랫폼

2026.04 ~ 현재

채용 공고를 자동 수집·정제하는 사이드 프로젝트의 배포·스케줄링 인프라 구성

- **무료 호스팅의 cron 미지원과 관리형 DB 무료 티어의 비활성 일시정지로 정기 작업 불가** — 호스팅 cron 대신 자체 Mac mini의 워크플로우 자동화(n8n) 스케줄러를 채택하고 warm-up 엔드포인트와 시크릿을 표준화, DB 일 1회 자동 warm-up으로 무료 티어 유지, manifest 기반 동기화 재구성

TypeScript Supabase n8n Vercel