Career Description — Hyeonjun Kil

GameDuo

2025.01 — Present

Mobile game development and publishing company. Shared backend platform design for 6 live game environments (12 operating branches), data pipeline engineering, and common package system operations

Projects

Real-Time Game Chat Server (Multi-Tenant)

2026.04 ~ Present

Designed a multi-tenant real-time chat server shared across live games, validating the 6,000 CCU / 18,000 RPS capacity target and Valkey fanout bottlenecks through load testing

- **No capacity model for chat infrastructure shared across games, blocking server sizing and SLO targets** — Based the capacity model on commercial chat solutions' published rate-limit/PCU specs instead of internal guesswork, defining a sizing formula for a single multi-tenant server frame, Set 6,000 CCU (sustained) / 18,000 RPS / 90,000 RPS burst targets, adopted into quarterly OKRs
- **Redis Streams WAL hit a single-threaded write bottleneck at 6K msg/s (EngineCPU 99.9%, ack p95 767ms)** — Distributed the WAL stream across a 4-shard cluster and compacted entries into native fields (removing Lua serialization) — chose horizontal sharding over vertical scaling, since single-threaded Redis serializes writes and cannot scale them vertically, ack p95 767ms→6.6ms (~116x), WAL backlog 80,000→2,838 cleared, 99.99% accept rate at scale
- **Messages were 100% rejected under concurrent multi-tenant load due to a missing scope key** — Built a 5-tenant concurrent load-test environment and removed rejects via hot-path log removal and message scope keys, Achieved ack p95 42ms (target <100ms) and 0.04% error rate (target <0.1%) at 299K messages sustained over 5 minutes
- **High CPU/memory utilization on x86 Fargate under 6,000 CCU load drove up operating cost** — Migrated to ARM64 (Graviton) on measured evidence rather than assumption — compared resource efficiency against x86 under identical 6,000 CCU load, using multi-platform image builds to support both architectures, Reduced CPU utilization 15% and memory 54% at identical load with neutral unit pricing — established operational cost-saving basis
- **After hot-channel fanout migration, the bottleneck was not isolated between Redis server Lua, gateway event loop, a single command queue, and topology** — Added attribution instrumentation before changing architecture, then validated Valkey serverless SSUBSCRIBE, per-channel subscriptions, and logicalChannelId-sticky connection pools step by step, Passed sandbox verify 19/0/0, reduced command queue 5,940→716, and processed 878,205 messages with zero loss
- **Offline DMs, unread counts, joined-room lists, and group fanout lacked durable per-user state, risking mixed ownership between chat and game domains** — Converged R2/R3/R4 requirements onto a per-user room index, keeping the chat server responsible only for mechanics/characterId while game servers owned profile hydration and group-domain actions, Completed server-side DM, unread-count, joined-room, and group-fanout capabilities under one index model, with server/sandbox QA completed before Unity handoff
- **Gateway tasks were repeatedly OOM-killed after performance changes, while outbound queues, observability retention, and heap caps remained competing root-cause hypotheses** — Pre-registered and rejected the egress-budget hypothesis, then isolated retained-heap growth and heap-cap alignment through heap snapshots, diagnostic logs, and A/B measurements, Reduced post-fix OOM events to zero and confirmed zero targeted-plan drift after removing observability-stack retained-heap accumulation

Tech Rationale

Adopted a Redis Streams Consumer Group-based WAL (Write-Ahead Log) for message ordering, and distributed it across a 4-shard Valkey cluster instead of vertical scaling — a single-threaded Redis cannot resolve write bottlenecks by adding cores. ECS Fargate ARM64 migration was adopted only after validating resource efficiency under identical load.

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

Marketing Integrated Platform

2025.02 ~ Present

Unified marketing platform managing Google Ads/Meta/TikTok campaigns, creatives, and metrics in a single system, built on dedicated marketing DB separation

- **Fragmented ad platform management across Google Ads/Meta/TikTok** — Delivered unified campaign automation API for creation, deployment, modification, and retention metrics in a single platform, Eliminated marketer console switching across 3 ad platforms via unified campaign automation
- **Meta asset sync performance bottleneck** — Replaced per-row ORM saves — which incurred heavy transaction overhead — with bulk INSERT and batch deletes, Image sync reduced 72% (25.7s→7.1s), DB transactions reduced 95% (10~16s→0.3~0.5s)
- **High BigQuery query costs for marketing data reads** — Adopted BigQuery Storage Read API with gRPC streaming-based high-performance data reading, Reduced BigQuery costs by 82% ($6.25→$1.1 per TB)
- **Marketing query latency (18s) from 100+ column denormalized table** — Normalized table into main/time-series/prediction, tuned indexes + cursor pagination, Performance improved 97% (18s→0.5s)
- **Large marketing-metrics archive backfill stalled at 63s Lambda runtime and 99.8% RDS CPU, ETA 15 hours** — Diagnosed the root cause as N+1 queries and a non-SARGable DATE() predicate (index miss) and removed both — chose a single 30-day batch fetch with in-memory latest-version dedup over per-row reads that were saturating RDS, Improved throughput 26x (9.3→247/min), Lambda runtime 63s→1.36s, RDS CPU 99.8%→5~7%
- **Marketing RCP data was spreadsheet-only, and the BigQuery aggregation table combined multiple sources — blocking unified console queries** — Switched to a single normalized BigQuery source table over combining multiple sources — which had produced value mismatches against the manual Excel flow — and added Criteria-Result metadata APIs to remove hardcoded display formatting, Embedded unified RCP queries into the console and replaced hardcoded currency/decimal display with BE metadata-driven formatting
- **Country/store-level platform groups made account selection and validation complex for the same campaign workflow, while QA token decryption failures blurred data setup issues with code regressions** — Merged BNID/BNKR/BNTW into one platform group and separated account-labeling APIs, account-selection UX, and sandbox-safe token re-encryption verification, Validated mappings for 566 campaigns across 3 Meta, 2 Google, and 2 TikTok accounts, with 0 remaining legacy-group accounts
- **TikTok deployments created orphan campaign/adgroup/ad resources when an external API partially succeeded before internal persistence, or retries were blocked by deployment guards** — Unified idempotency per deployment path: Smart+ V2 level-by-level get-or-create with immediate persistence, v1 campaign-level reuse, and regular ad-level flags, Passed 31 unit tests for partial-failure retries and applied a policy to prevent recurrence of 2 orphan campaigns and 2 orphan ad groups
- **As external ad-API coverage expanded, manual types and runtime validation were mixed across campaign/adgroup/ad/creative callers, increasing type-drift and live-call failure risk** — Standardized official Swagger-based Orval+Zod codegen as the default path, supplemented missing-spec APIs with manual specs, and migrated Smart+, standard deployment, file, OAuth, and creative/playable flows to SDK calls, Completed conversion across 24 endpoint groups and standardized external-API typing, validation, and preflight guardrails
- **A 26.37M-row / 27GB audit-log table lacked an index for parent-based lookups, causing page queries to trigger large full scans and prolonged IO saturation** — Declared the index at the entity layer to prevent drift, pre-created the online DDL during a low-load window, and hardened the migration with duplicate-index no-op handling plus real DDL integration tests, Confirmed the live index and EXPLAIN rows=3, eliminating audit-log full scans on the lookup path

Tech Rationale

Reduced BigQuery costs 82% via Storage Read API migration. Designed hybrid GCP Pub/Sub → AWS Lambda/SQS pipeline for event-driven processing with Outbox pattern for non-blocking async event publishing.

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

Internal Common Library System

2025.07 ~ Present

Development and operation of NestJS utility (10 modules) + Game Server Kit (2 packages) for multi-project code consistency

- **Common code duplication across services increasing maintenance cost** — Designed 10-module library system: core, repository, cache, lock, slack, crypto, smb, hash, type, iac, Unified shared code across 7 projects with standardized dependency management
- **Repository module lacking bulk operations and audit logging capabilities** — Replaced a read-then-write pattern that caused write-race duplicate-key conflicts with an atomic upsert plus automatic audit-log branching; added type narrowing via overloading and generics, and split the oversized Repository into purpose-scoped modules per SRP, Refactored 2000+ lines into shared Repository abstraction layer eliminating CRUD duplication
- **Concurrency conflicts in multi-instance environments** — Abstracted ElastiCache (Redis) distributed locking into an AOP decorator, isolating lock logic from business code, Applied distributed lock decorator consistently for multi-instance concurrency control
- **Manual library upgrades taking 3 hours across 7 projects** — Added workflow_dispatch+matrix and changed-package CI tests on GitHub Packages, Reduced 7-project upgrade runtime 3h→15min
- **Slow CI pipeline (15m47s) due to test infrastructure inefficiency** — Chose tsc-native ts-jest isolatedModules with explicit Entity types over @swc/jest, which had weaker decorator-metadata compatibility, Cut CI time 61% (15m47s→6m06s); passed 81 suites/978 tests
- **Game-server shared code split across 5 branches + one game's 17 game-data definition modules were 2+ years diverged from the published package** — Extracted the game-data definition modules + 5 submodules into a standalone package, then migrated 610 files / 310+ imports in one game and shipped 9 follow-up hotfixes (config injection, missing interface methods), Package system established with first-game adoption — 9 hotfixes shipped with zero incidents; game-data module updates now flow through a single npm version bump (manual migration of 610 files → 0)
- **Game-server cache logic was duplicated per project, increasing consistency and maintenance burden** — Packaged a hybrid strategy combining local LRU (hot-path latency) and distributed cache (cross-instance consistency) as a decorator-based AOP library — chose hybrid over pure-distributed (network round-trips) and pure-local (cross-instance inconsistency), splitting tag index and single-flight into separate modules per SRP, Released cache library v0.5.0 — 262 tests across 28 suites passing, 100% code-standard compliance

Tech Rationale

Implemented ElastiCache (Redis) distributed lock as AOP decorator to separate lock logic from business code. Evaluated 5 options for Jest 30 VM isolation and adopted poolSize=2.

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

CS Support Chat Server

2026.04 ~ Present

Fully isolated the customer-support (CS) chat server onto ECS from a shared deployment, and validated authentication, security, and load ceilings

- **CS chat was coupled to a shared deployment, causing resource contention and deployment coupling** — Fully isolated the service onto ECS Fargate and automated deployment via Terraform IaC (S3 state), GitHub Actions OIDC, and an observability sidecar — adopted a centralized game-token verification path over per-game distributed checks, Completed independent service cutover with modularized Terraform, an 11-panel Grafana dashboard, and 8 alerts
- **Layer coupling and ad-hoc token lifetimes in the new CS chat domain posed security and maintenance risks** — Separated responsibilities into a 4-layer Clean Architecture and introduced OAuth2 access (5min)/refresh (1h) tokens with ETag/304 polling — sending only deltas instead of full payloads per poll, Passed 39 tests across 12 suites; applied IaC plan (+10 resources) with zero incidents
- **1.0 vCPU ceiling, 40~90s cold start, and bloated images blocked spike-traffic handling** — Redesigned task resources and ALB RPS autoscaling, and cut cold start via native ARM64 builds and Dockerfile slimming with migration separation, Reduced image 381→254MB (-127MB) / build 5→2min / cold start by 30%; 0% failure on a k6 500-VU spike
- **No cross-project data isolation, request rate limiting, or attachment access control** — Introduced cross-project isolation (triple-validation helper), Redis-backed rate limiting (auth 5/min, refresh 10/min, message 30/min), and an attachment FK integrity migration — rejected in-memory throttling for its multi-instance bypass risk, Merged the security integration with zero type errors and established cross-org/project isolation specs
- **Multiple inquiries were mixed inside one CS chat room, making per-inquiry counts and handling-time metrics difficult, while video attachments for bug reports were unsupported** — Designed additive APIs for immutable message-reply links, user-level chat lists, and a dedicated video-upload path with MIME, size, TTL, and daily limits, Established message-level reply/target changes, user-level lists, video inquiry playback, and inquiry-count/handling-time aggregation (20MB, 10 uploads/day, 15min TTL)

Tech Rationale

Isolated the service onto ECS (Terraform IaC + GitHub Actions OIDC) to break resource contention and deployment coupling, and adopted a centralized game-token verification path over per-game distributed checks. Rate limiting was implemented with a Redis-backed throttler for multi-instance consistency.

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

AWS Lambda Migration & Event-Driven Architecture

2025.06 ~ 2025.08

Resolved batch job limitations from marketing metrics 60-day → 360-day expansion and serverless transition for batch processing

- **Full serverless migration risked operational stability** — Designed hybrid architecture keeping API server on EC2 while separating batch/job processing to Lambda, Preserved API availability with zero impact, isolating only batch workloads to contain operational risk
- **Batch processing limited to 60-day collection range with 2-hour runtime** — Established Event-Driven flow with SQS+Lambda+EventBridge, Reduced batch time 2h→5min, expanded collection 6x (60→360 days)
- **Data consistency risk during event publishing** — Adopted Transactional Outbox Pattern for scheduled and delayed event publishing, Ensured data consistency across distributed event processing
- **After cache disk-full recovery in the CDK deploy workflow, Lambda handler webpack builds reproduced V8 OOM, combining slower deployments with no regression guard** — Separated runtime-transform dependencies into generated artifacts, raised the default handler batch, and retried only failed batches through an adaptive 8→4→2→1 split sequence, Removed the Lambda handler build-OOM root cause, added CI regression guardrails, and validated the deployment path through 3 passing CI runs
- **DB connection exhaustion during massive Lambda execution** — Introduced RDS Proxy connection pooling, Resolved connection exhaustion and stabilized database access

Tech Rationale

Chose Lambda to decouple batch/event workloads bound to the monolith server, enabling independent deployment. Adopted SQS for async processing to resolve scaling limitations under traffic fluctuation.

TypeScript NestJS AWS Lambda SQS SNS EventBridge RDS Proxy AWS CDK

In-Game Multi-Language Translation System

2026.02 ~ Present

Redesigned the in-game notification translation domain model — decoupling AI translation from library-based conversion into a 3-stage pipeline

- **Simplified Chinese was modeled inside the AI translation settings domain despite not being AI-supported, and settings storage/retrieval depended on the Traditional Chinese AI translation state — making independent toggling impossible without a breaking change** — Extracted the derived-conversion module from the AI translation module into 9 use cases with dedicated entity/repository/scheduler, and applied the global-inheritance pattern for effective settings, Clarified domain boundaries by extracting 9 use cases and enabled independent toggling of derived conversion rules
- **Decoupling required a breaking API change** — Simultaneously deployed FE/BE/Gateway with role permission migration for zero-downtime transition, Completed the breaking API change without incident and cut effective-settings DB queries by 50% (4→2)
- **The detect cron lacked version priority, delaying translation of the latest version; 30-second interval further slowed response** — Added version ID-based priority ordering to detect SQL, shortened the processing interval from 30s to 10s, and raised batch limit from 100 to 200, Eliminated latest-version translation lag; progress UI plus total-count caching improved user visibility
- **Detect scheduler scanned 159 versions in one shot (3.27M-row full scan) — 60s login outages from RDS IOPS saturation** — Ported the per-version loop from the AI detect module, added distributed-lock timeout/recovery, and replaced the NOT EXISTS full scan with an indexed staged lookup, Cut query time 260x (full scan → 39ms indexed lookup), removed driving filesort, restored login latency

Tech Rationale

Materialized the responsibility boundary between AI translation and library-based derived conversion inside the code structure. Split the Detection → Processing (AI) → Conversion (library) pipeline into independent modules and secured concurrent request integrity with a 2-stage race guard plus pessimistic write lock

TypeScript NestJS TypeORM MySQL opencc

Marketing Platform Audit Log System

2025.01 ~ 2025.04

Resolved delayed balance issue response caused by inability to track data change history during game operations

- **Inability to track data change history across environments and projects** — Designed Git-like version control system with UUID-based cross-environment/project entity tracking, Ensured data consistency across 6 games
- **Manual entity change recording prone to omission** — Applied Event Sourcing-based change tracking with Auditable decorator + TypeORM Subscriber pattern, Automated entity change tracking with consistent logging across all game environments
- **Multi-environment version conflicts during data merge** — Developed 3-Way Merge Engine with parent/child entity conflict detection and unique constraint handling, Enabled reliable version merging across 6 games
- **Expensive full-snapshot comparison for every version diff** — Designed Version Diff Engine with dual strategy: incremental comparison and snapshot comparison based on Base Audit availability, Dual-strategy version comparison (incremental + snapshot) for performance optimization
- **Entity tracking failures due to PK dependency during migration and merge** — Introduced shared identifier-based entity tracking decoupled from PK dependency, Ensured accurate entity tracking during migration, comparison, and merge

Tech Rationale

Adopted TypeORM EntitySubscriberInterface after dedicated analysis of subscriber behavior and constraints. Designed AOP-based approach combining Auditable decorator + Subscriber to automatically collect entity changes into a standardized audit pipeline.

TypeScript NestJS TypeORM MySQL

Cloud Data Sync System

2025.08 ~ Present

Built S3-based sync and automated DDL management system to resolve dynamic game data inconsistency across environments

- **Dynamic game data inconsistency across environments** — Built S3-based cross-environment data synchronization across development/staging/production, Unified game data state across all environments
- **Manual DDL schema management causing sync failures** — Created automated DDL management engine with dynamic PK column type resolution, column type mismatch detection with MODIFY, and automatic index creation/RENAME, Automated database schema drift detection and reconciliation
- **Large-scale Cloud Data ingestion and S3 upload bottlenecks** — Analyzed and optimized ingestion and upload pipeline, TRUNCATE→DELETE optimization for batch operations — 77% latency reduction (57.5s→12.9s)
- **Sync job instability causing operational issues** — Implemented job separation, transitioned scheduling approach, tuned timeouts, and introduced non-blocking processing, Data layer resilience through schema optimization and safe migration strategies
- **Four data-sync service bugs: Redis cache invalidation, DDL-SKIP metadata, copy-key deletion, and data-protection option propagation** — Diagnosed and fixed all four issues systematically, Diagnosed and fixed 4 critical bugs to stabilize data-sync operations
- **Risk of unintended full data deletion when a protection option was not forwarded on some migration paths** — Forwarded the data-protection option across the four POST migration paths that were missing it, Blocked unintended full data deletion risk

Tech Rationale

Applied S3 Lifecycle policies (30d Glacier IR, 90d expiry) for cost optimization. Switched from event-triggered to scheduled execution to reduce sync miss risk.

TypeScript NestJS TypeORM MySQL S3

Probability Calculation & Audit Log Analytics Pipeline

2026.02 ~ Present

Built probability calculation package and CDK-based audit log analytics infrastructure to address regulatory risk from lack of game probability verification

- **No reusable probability calculation module across 6 live game environments (12 branches)** — Packaged NestJS DynamicModule with 5 probability functions + Kinesis logging as shared probability package, Deployed across 6 live game titles (12 branches) — end-to-end audit log pipeline for probability verification
- **No audit log analytics infrastructure for probability verification** — Codified CDK analytics pipeline: Kinesis→Firehose (Dynamic Partitioning+Parquet)→S3→Glue→Athena, Deployed across 6 live titles (12 branches) — CDK-based analytics unlocking end-to-end probability verification
- **Mock-based tests lacking regression confidence for infrastructure code** — Replaced mocks with LocalStack + Testcontainers integration tests, Secured 94%+ coverage (12 suites/115 tests)

Tech Rationale

Codified Kinesis → Firehose (Dynamic Partitioning + Parquet) → S3 → Glue → Athena pipeline with CDK. Chose Parquet columnar format for Athena SQL cost optimization. Replaced mock-based tests with LocalStack Testcontainers for integration testing.

Contribution: *Restructured initial implementation into a package. Independently handled P0 bug fixes, test stabilization (94%+ coverage), CDK infrastructure, and deployment across 6 live game environments (12 branches).*

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

Marketing Data Archiving & Custom Dashboard

2026.04 ~ Present

Built daily-snapshot archiving of marketing metrics (with immutability), a 4K-row custom dashboard, and alert observability

- **Displaying 4K-row marketing metrics in one dashboard caused scroll lag and cell-sync bugs** — Split BE (Raw SQL DISTINCT JOIN avoiding ORM hydration + in-memory cache) and FE (visible-row lazy fetch + 200-row chunks balancing network round-trips and render latency + per-pair caching) — rejected full loading due to response size and render cost, Displayed 3,341 of 4,458 rows mapped, stabilized response at 573ms cold / 221ms warm
- **Sync paths used the latest snapshot instead of the reference-date cutoff, violating archive immutability (data contamination)** — Redesigned the usecase/repository to honor cutoff semantics, made the stale-writer guard transactional, and introduced tombstone states for sync, Rebuilt 15,333 live archives (zero contamination), validated 287,020 archive-date records
- **A transient log-query stack (Loki) outage fired 16 marketing alerts at once, flooding ops channels** — After diagnosing the root cause, migrated all 16 alerts to a single analytics DB (ClickHouse) query, rewrote them via the dashboard API, and set a legacy log-stack decommission policy, Migrated 16/16 alerts (25 min), preventing alert-flood recurrence during log-stack outages

Tech Rationale

Archives must be immutable snapshots keyed by a reference-date cutoff rather than read-time state, so integrity was enforced with a stale-writer guard and tombstone states. The large grid uses visible-range lazy fetch with chunk caching instead of full loading.

TypeScript NestJS React AWS Lambda BigQuery ClickHouse Grafana MySQL

Activities

Boosting Developer Productivity with Amazon Q Developer Conference Talk 2025.10

Games on AWS 2025 Customer Session

Shared a case study of a single engineer scaling data pipeline capacity 270x in 10 days, measured by hourly concurrent job throughput

JNPMedi

Dev2 Team Docs Squad Backend Engineer

2023.06 — 2024.08

Clinical trial data management solution startup with KRW 16B cumulative investment (MSA-based)

Highlights

- 5+ monthly e-signature failures and clinical data-integrity risk caused by deadlock patterns. Reproduced and diagnosed FK shared-lock deadlocks, then adjusted transaction boundaries to prevent recurrence. Eliminated 5+ monthly e-signature processing failures.
- VDR project schedule delay under MVP launch pressure. 3-week emergency sprint (2 BE + 1 FE), Event-Driven PDF conversion + permission-based workflow. MVP launched on schedule.
- Delayed email failure detection (hours). Constructed SES+SNS event pipeline. Bounce/complaint detection cut to seconds, Slack instant alerting (published tech blog).

Projects

Maven Docs Electronic Consent System

2023.07 ~ 2023.09

Resolved inefficiency in collecting participant consent and e-signature regulatory compliance using Cyan (in-house Node.js framework)

- **Inefficient individual consent dispatch for clinical trial participants** — Created batch dispatch system for participant group management and batch consent dispatch, Streamlined consent collection per clinical trial
- **Opaque event flow from batch + S3 event-based processing** — Migrated to EDA + Outbox pattern, Enabled event flow visualization and local test environment setup

> **Tech Rationale**
> Adopted EDA + Outbox pattern to eliminate synchronous service coupling. Designed plugin architecture for extensible input methods (text, checkbox, signature).

Contribution: *Designed EDA transition, implemented plugin architecture, built batch participant registration with Signer table migration. 2-person BE team.*

TypeScript Express.js Cyan Aurora Serverless v2 Knex.js

Maven Docs System Reliability Improvement

2023.08 ~ 2023.12

Resolved critical notification failures due to database Deadlocks in clinical trial e-signature notifications

- **5+ monthly Deadlock incidents causing e-signature notification loss in production** — Reproduced shared-to-exclusive lock escalation via INNODB lock tables, applied SELECT FOR UPDATE preemption, Reduced e-signature failures from 5+/month to zero, eliminating the lock contention root cause
- **Synchronous notification processing causing cascading failures** — Designed and implemented AWS SNS → SQS → Lambda Event-Driven pipeline, Decoupled notification delivery from main transaction flow

> **Tech Rationale**
> Identified FK child INSERT S-Lock → X-Lock escalation as deadlock root cause. Applied SELECT FOR UPDATE to pre-acquire X-Lock before INSERT, eliminating the deadlock ordering entirely.

Contribution: *Independently performed deadlock reproduction, analysis, and resolution. Reproduced deadlock scenario with SQL in DBeaver and verified via INNODB lock tables.*

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

Maven Mailing System Enhancement

2023.12 ~ 2024.02

Resolved lack of delivery status tracking and data consistency issues in email system

- **No email delivery status tracking — failures detected hours later** — Configured SES Configuration Set → SNS → event processing pipeline with Datadog logging + Slack real-time alerts, Bounce/complaint detection cut to seconds with instant Slack alerting

> **Tech Rationale**
> AWS SES cannot synchronously confirm delivery status due to its async nature. Added SNS event destination to SES Configuration Set for async delivery/bounce/complaint event subscription.

Contribution: *Designed and implemented SES Configuration Set → SNS → event processing pipeline. Led tech blog publication and internal seminar presentation.*

TypeScript Express.js AWS SES Nodemailer

Maven Auth System Enhancement

2024.01 ~ 2024.02

Resolved user inconvenience and system scalability limitations from 1:1 user-organization structure

- **1:1 user-organization structure limiting multi-organization management** — Expanded user-organization relationship from 1:1 to 1:N structure, Enabled multi-organization management per user
- **API migration risking service downtime** — Ensured backward compatibility throughout migration process, Achieved zero-downtime API migration with service continuity

Tech Rationale

Redesigned account-organization from 1:1 to 1:N to support clinical trial researchers participating in multiple organizations simultaneously. Added post-login organization selection flow for context switching.

Contribution: *Full-stack change: schema migration, auth flow modification, and batch update of downstream service (Docs/Billing/Mailing) account-organization reference logic.*

TypeScript | Express.js | Cyan | Aurora Serverless v2

Maven TMF New Project Development (Frontend)

2023.06 ~ 2024.08

Frontend development for clinical Trial Master File (TMF) management system

- **No systematic Trial Master File (TMF) management system — inefficient regulatory compliance and document tracking** — Mobilized as an emergency developer in a 1 BE + 2 FE team; defined MVP scope and built document upload, classification, version control, and Audit Trail on the Admin/Dashboard, Delivered the TMF management frontend MVP in 3 months, establishing systematic management of clinical-trial essential documents

Tech Rationale

Defined MVP scope focusing on core features (document upload, classification, version control, audit trail) under 3-month delivery constraint.

Contribution: *FE developer in 1 BE + 2 FE team. Implemented Admin/Dashboard document upload, classification, and audit trail features.*

React | TypeScript | Jotai

Maven VDR MVP Development

2024.06 ~ 2024.07 (3 weeks)

Emergency deployment for delayed VDR project under MVP launch pressure

- **Synchronous PDF conversion causing timeouts on large documents** — Designed Event-Driven PDF conversion triggered by S3 upload → Lambda lightweight conversion → SNS/SQS notification, Eliminated synchronous timeouts; large documents now flow through async pipeline reliably
- **No role-based document access control for medical workflows** — Engineered permission-based workflow managing document access and approval by medical role, Defined per-role document access and approval workflow
- **VDR project schedule delay under MVP launch pressure** — Executed 3-week emergency sprint prioritizing core features, Delivered MVP within 3-week timeline (BE 2 + FE 1 team)

Tech Rationale

Designed EDA-based async conversion pipeline (S3 trigger → Lambda conversion → SNS/SQS notification) to structurally resolve synchronous PDF conversion timeout issues.

Contribution: *1 of 2 BE developers. Designed and implemented EDA document processing pipeline and large-scale async PDF conversion worker. Delivered MVP within 3-week emergency sprint.*

TypeScript | Lambda | S3 | SNS | SQS

Maven Billing Subscription Management System

2023.06 ~ 2024.08

Subscription, plan, and license management system for all Maven services

- **No support for organization-specific subscription plans** — Shipped org-specific custom plan Internal API, Facilitated tailored subscription offerings per organization
- **Non-renewable plans could be incorrectly renewed or re-subscribed** — Added renewal/re-subscription blocking logic for non-renewable plans, Ensured business policy consistency for subscription lifecycle

Tech Rationale

Introduced event queue for async cleanup processing after subscription expiry. Enforced non-renewable plan re-subscription blocking via bundles.is_renewable schema attribute at the data level.

Contribution: *Implemented org-specific custom plans, event queue-based inventory auto-deletion, and renewal eligibility logic. Multiple PRs directly contributed.*

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

Activities

Email Delivery Monitoring through AWS SES Event Logs

Tech Blog

2024.03

Published technical article on building email delivery monitoring system using AWS SES event logs

Mailing System Email Delivery Result Tracking Feature Internal Seminar 2024.02

Presented mailing system email delivery result tracking feature implementation and architecture

AWS SAA Study Group Study Group 2023.09 ~ 2023.12

Internal study group for cloud architecture deep dive

Medgo

Dev Team Backend Engineer

2022.04 — 2023.06

Telemedicine and medication delivery platform startup with 300K cumulative downloads

Highlights

- Regression risk increased as auth, appointment, and notification changes accumulated around an Express.js single-file app.js structure. 3-stage progressive migration (single-file→layered→NestJS) + JS→TS. TDD 80% coverage, zero incidents during major releases.
- Manual prescription entry bottleneck (3-5min per case). Naver Clova OCR + MFDS (Korea FDA) API integration for full workflow automation. Pharmacist processing time cut 90% (3-5min→30sec).
- Status reflection delay in consultation/dispensing/delivery (tens of seconds). Socket.IO real-time sync + Web Push notifications. Reduced to under 1 second, eliminated patient wait complaints.

Projects

Auth Server Development

2022.04 (3 weeks)

Built JWT-based unified authentication system to resolve subdomain session conflicts

- **No unified authentication system for multi-subdomain architecture** — Designed and implemented NestJS + JWT token authentication system, Unified auth across web (doctor/pharmacist/admin) + mobile app
- **Session-based auth causing subdomain transition failures** — Migrated from session-based to token-based authentication, Enabled reliable user transitions across subdomains

> **Tech Rationale**
> Structurally resolved subdomain cookie-based session conflicts with stateless JWT tokens. Chose MySQL over Redis for key management — Redis deemed over-engineering given traffic volume, prioritizing infrastructure simplicity.

Contribution: *Independently designed and implemented over 3 weeks. Covered entire auth system for web (doctor/pharmacist/admin portals) + mobile app.*

NestJS TypeScript JWT MySQL

Common Module Development

2022.05 ~ 2022.06

Resolved code duplication and inconsistency from independent service management by consolidating shared modules

- **No centralized permission verification across services** — Implemented user permission verification middleware, Enhanced security with unified access control

NestJS TypeScript

Doctor/Pharmacist Web Service Enhancement

2022.04 ~ 2023.05

Eliminated manual notification workflows and lack of real-time status updates in doctor/pharmacist back-office systems by introducing Socket.io and Web Push

- **Duplicate appointment bookings causing scheduling conflicts** — Implemented Doctor ID + time slot composite unique key constraint, Prevented duplicate appointments at the database level
- **Status reflection delay (tens of seconds) in consultation/dispensing/delivery** — Deployed Socket.IO Room-based real-time status sync reflecting events to relevant users instantly, Reduced delay to under 1 second, eliminated patient wait complaints
- **External service failures (delivery, payment) cascading to main application** — Architected internal API gateway integrating delivery (Hoodadak) and payment services with Circuit Breaker pattern, Achieved fault isolation preventing cascading failures

> **Tech Rationale**
> Used Socket.IO Room-based event isolation for real-time consultation/dispensing/delivery status sync. Integrated API gateway + Circuit Breaker to prevent external service failure propagation. Enforced duplicate reservation prevention via DB composite unique key.

Contribution: *Directly implemented Socket.IO real-time sync, Web Push notifications, API gateway, and reservation dedup in 2-person BE team.*

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

Prescription OCR & Medication Guide System

2022.11 ~ 2022.12

Automated pharmacist's manual prescription entry process using OCR

- **Manual prescription text entry from images (3-5 min per case)** — Integrated Naver Clova OCR API for automatic text extraction from prescription images, Automated prescription data entry
- **Manual medication guide creation inefficiency** — Developed automated medication guide generation using MFDS (Korea FDA) drug information API, Reduced pharmacist processing time 90% (3-5min→30sec per prescription)

Tech Rationale

Selected Naver Clova OCR for Korean pharmaceutical terminology recognition. Integrated MFDS public data API instead of building proprietary drug database to minimize maintenance overhead.

Contribution: *Independently implemented over 2 months. Built entire pipeline: image intake → OCR extraction → MFDS API matching → medication guide generation.*

NestJS | TypeScript | Naver Clova OCR | MySQL

Backend Architecture Migration

2023.03 ~ 2023.05

Migrated from Express.js single-file (app.js) structure to NestJS modular architecture

- **Express.js single-file (app.js) 10,000-line codebase unmaintainable** — Executed 3-phase incremental migration: app.js single file → Layered Architecture (Controller/Service/Repository) → NestJS framework adoption, Migrated 10,000-line monolith to NestJS modules with zero production incidents during major releases
- **JavaScript lacking type safety causing runtime errors** — Migrated JavaScript → TypeScript across the codebase, Established type safety and improved developer experience
- **No automated testing culture or regression safety net** — Introduced TDD practices with comprehensive unit tests, Achieved 80% unit test coverage, zero incidents during major releases

Tech Rationale

Adopted NestJS DI/module system to resolve maintainability limits of 10,000-line Express single-file legacy. Minimized risk via 3-phase gradual migration (single file → layered → NestJS). Introduced TDD to prevent regression during migration.

Contribution: *Led independently. Performed all 3 migration phases, JS→TS migration, TDD adoption, and team coding convention establishment.*

NestJS | TypeScript | TypeORM | Jest | MySQL

SimpleHAN

Dev Team Backend Engineer

2021.07 — 2022.03

Custom software consulting firm

Highlights

- Manual learning management (attendance, completion, progress tracking). Built LMS full-stack (QR attendance, PDF certificates, Excel bulk registration). Digitized operations for 300+ students per semester.
- New matching and counseling features had to be added to a legacy PHP system, while direct changes carried high regression risk. Integrated an independent Spring Boot app through a constrained iframe/postMessage boundary and implemented a weighted-average matching algorithm. Launched the JOB Agent service while isolating the legacy-system impact surface.

Projects

Learning Management System (LMS) Development

2021.07 ~ 2022.03

Resolved manual learning management (attendance, completion, progress) causing low operational efficiency and frequent errors at educational institutions

- **Manual attendance tracking prone to duplicates and errors** — Built QR attendance API with date+userID composite unique key preventing duplicate check-ins, Digitized real-time attendance for 300+ students per semester
- **Manual member registration causing data inconsistency** — Built Apache POI-based Excel bulk registration with row-by-row sequential processing, Ensured data consistency for batch member registration
- **Manual certificate generation workflow** — Built JasperReports-based PDF auto-generation API, Automated dynamic certificate creation

Tech Rationale

Selected Java/Spring Boot as enterprise standard for educational institution delivery. Enforced QR attendance dedup via DB composite unique key (date+userID). Used JasperReports for template-based bulk PDF certificate generation.

Contribution: *Directly implemented core features over 10 months: QR attendance, Excel batch registration, PDF certificates, progress tracking. Backend lead in 6-person full-stack team.*

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

Incheon Smart Green Industrial Complex Control Center

2022.01 ~ 2022.03

Resolved fragmented reservation, monitoring, and SMS systems preventing unified management at industrial complex control center

- **Scheduling conflicts from concurrent facility reservations** — Built real-time booking API with time+location composite unique key, Prevented scheduling conflicts at the database level
- **Synchronous SMS sending blocking reservation confirmation flow** — Implemented Bizppurio SMS API + @Async non-blocking notification, Achieved non-blocking reservation confirmation notifications

Tech Rationale

Enforced meeting room booking conflict prevention via DB composite unique key (time+location). Used @Async for SMS to resolve API response delay — separate message queue deemed over-engineering for the traffic scale.

Contribution: *Backend lead over 4 months. Directly implemented reservation API, dashboard monitoring API, and async SMS integration.*

Java | Spring Boot | MySQL | Bizppurio SMS API

Future Service JOB Agent Development

2021.07 ~ 2022.03

Resolved legacy PHP system's extensibility limitations and lack of structured scoring and counseling management

- **Legacy PHP system could not be extended for new features** — Embedded Spring Boot independent app via iframe integration, minimizing legacy impact, Launched JOB Agent service without disrupting existing system
- **No structured scoring system for multi-dimensional assessments** — Implemented weighted average-based composite scoring algorithm, Delivered standardized multi-dimensional assessment scoring

Tech Rationale

Embedded independent Spring Boot app via iframe for non-invasive legacy PHP system extension. Used postMessage API for cross-frame communication. Implemented weighted average algorithm for multi-dimensional evaluation scoring.

Contribution: *Backend lead over 10 months. Designed iframe integration, implemented weighted scoring algorithm, postMessage state sync, and counselor back-office.*

Java | Spring Boot | MySQL

kfriends Member Management System

2021.09 (3 weeks)

Resolved lack of unified authentication requiring separate logins across 5 country-specific distributed websites

- **Separate logins required across 5 country-specific websites** — Built unified auth server with Spring Security + JWT-based centralized authentication, Enabled single sign-on across all 5 country sites

Tech Rationale

Resolved fragmented login across 5-country websites with centralized JWT-based unified auth server. Configured CORS multi-domain handling for cross-origin authentication.

Contribution: *Independently implemented over 3 weeks. Built entire unified auth server.*

Java | Spring Boot | Spring Security | JWT | MySQL

Internal Tech Blog Renewal

2021.09 (3 weeks)

Resolved hosting cost burden and manual deployment workflow of WordPress-based blog

- **WordPress hosting cost burden and manual deployment workflow** — Migrated WordPress → Jekyll with static site generation, Eliminated 100% hosting costs with automated deployment

Jekyll | Liquid | Travis CI | GitHub Pages

Personal Projects (Side)

Platform / Tooling Engineering

2026.04 — Present

Self-directed platform/tooling engineering outside work — development workflow automation framework and side projects

Projects

Personal Development Operations Harness Framework

2026.04 ~ Present

Designed and operated a personal workflow framework that automatically manages development-session context, cost, rules, RAG, and browser runtimes across 3 harnesses

- **Growing markdown knowledge base made manual context discovery by agents inefficient** — Adopted a local-embedding (BGE-M3) vector RAG over cloud vector DBs (e.g., Pinecone) — chose local to auto-reindex immediately on markdown changes and avoid sync latency/cost; combined sqlite-vec with FTS5 (RRF fusion) for accuracy, exposed via MCP for direct agent search, Indexed 548 files / 3,347 chunks, 0.7s warm query, all 5 gold test queries hit within top-2/3
- **RAG was an opt-in reference tool, so evidence search was missed during investigation and review prompts** — Moved to a default UserPromptSubmit trigger and ported it across 3 harnesses so prompts of 12+ characters call search_harness automatically, Passed 15/15 default-on tests across 3 harnesses and hit the target on the first search in a real-prompt smoke test
- **Resume and portfolio JSON were outside the RAG search scope, leaving JD customization without evidence search over those surfaces** — Implemented portfolio/resume JSON chunkers, target=resume indexing, path-based auto-reindexing, and federated search, Indexed 109 resume-corpus chunks, passed 32 tests, and verified federated search across 4 DBs
- **95% of AI agent token cost came from hidden areas (system prompt, subagents), making it untrackable** — Built a cost analyzer that decomposes session transcripts into 8 categories instead of simple session/model aggregation, which could not trace hidden costs — added residual tracking to consolidate multi-machine, multi-tool usage into a single profile, Made 95% hidden cost visible by category; found and fixed a multi-instance usage-aggregation bug
- **Browser and REPL runtime trust boundaries and fallback order were unclear, making automation unstable across sessions** — Documented runtime trust boundaries and environment-native Browser/Chrome fallback order, then reflected Chrome plugin activation paths in the launcher generator, Registered runtimes across kh/gp/gd harnesses and standardized browser runtime trust boundaries and fallback order

Tech Rationale

Chose hook-first enforcement because document-only rules are not always read at the moment of need; generated and verified runtime settings through a runtime-surface compiler and launcher package to prevent multi-instance drift.

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

Job Posting Aggregation Platform

2026.04 ~ Present

Built the deployment and scheduling infrastructure for a side project that auto-collects and normalizes job postings

- **Free-tier hosting lacked cron support and the managed DB's free tier auto-paused, blocking scheduled jobs** — Adopted a self-hosted Mac mini workflow-automation (n8n) scheduler instead of hosting cron, and standardized warm-up endpoints and secrets, Kept the free tier alive via daily DB warm-up and restructured manifest-based sync

TypeScript | Supabase | n8n | Vercel