

Geonwoo Kim Builder

I remove friction with technology.

Portfolio

geonwoo.kim
contact@geonwoo.kim
github.com/takealook97

Values

- I aim to be AI-Native. I value vendor neutrality and avoid lock-in to any single LLM.
- I believe an imperfect idea that ships is worth more than a good one that never does.
- I value going far together over going fast alone.

Background

- I have developed backends for a messaging service and Shopify-based e-commerce stores.
- I build integration servers, deployment automation and internal tools as needed.
- I am building my own services on a home server running Proxmox.

Now

- At Lukuku, I build Shopify backends, ERP integrations and custom apps for several brands.
- I design and build the agent operations system for deepeet.
- I bring the tools I build to my team and open-source them.

Workflow

- I develop with a multi-agent pipeline I built around Claude Code, Codex and Gemini CLI.
- I run multiple agents in parallel in terminal multiplexer environments such as tmux and Herdr.
- orch assigns and reviews work; vat tracks repository changes and manages memory.

Education

Dongguk University · Bachelor of International Trade — 2017.03 — 2023.02

Gimpo Foreign Language High School · English department — 2013.03 — 2016.02

Certifications

OPIc IH · ACTFL — 2025.09

SQLD · Korea Data Agency — 2024.04

Training

Samsung Youth SW Academy (SSAFY) · Java track — 2023.07 — 2024.06

Java, Spring Boot, MyBatis, DB, algorithms, Vue.js, web development · worked on 5+ team projects and led 3

Codesquad · Backend course — 2023.01 — 2023.06

CS, Java, Spring, DB, team projects and more

Activities

Notion Ambassador — 2024.07 — 2025.07

- Selected as a global ambassador
- Wrote articles on automating workflows with the Notion API
- Joined beta tests, QA and meetings with ambassadors abroad

Awards

Project Excellence Award · SSAFY — 2024.02

1st place in the common project

Contents

Lukuku

space-I · client OS	2026.10 — Present	3
Shopify B2B store for a US subsidiary · B2B backend	2026.09 — Present	4
Lukuku OS · company ontology and agent operations	2026.09 — Present	5
360° video app and AI video server for an installation art exhibition	2026.06 — 2026.08	7
Sales reconciliation and integration operations between a sports brand's Korean online store and its ERP	2026.04 — Present	8
Internal operations agent built on Hermes Agent	2026.04 — Present	9
offon · company-wide work management system	2026.04 — Present	11
Participant-only tournament voucher ordering for a sports brand's Korean online store	2026.04 — 2026.09	12
Shopify store features and integrations for brands	2026.04 — 2026.07	13

Biztalk

AWS MSK 3.8 migration	2025.08 — 2025.09	14
BGMS log masking module	2025.07	15
Overdue payment notice RPA	2025.06 — 2025.07	16
BGMS messaging system development and operations	2025.05 — 2025.10	17
Deployment pipeline setup and internal GitLab migration	2025.05 — 2025.10	17
Server configuration management and deployment scripts	2025.05 — 2025.10	18
KakaoTalk sending and template statistics collection server	2025.05 — 2025.06	18

deepeet

AI-Native commerce run by agents	2026.07 — Present	19
Advertising operations	2026.10	20
Pricing, promotions and settlement	2026.08	21
Routing and sales status control	2026.07 — 2026.09	22
Agent operations and organizational memory	2026.07 — 2026.08	23
Sourcing data	2026.07 — 2026.08	25
Development and infrastructure operations	2026.07 — 2026.10	26
Operations console and customer service	2026.07 — 2026.08	27
Product listing	2026.07 — 2026.08	28
Order automation	2026.07 — 2026.08	29

Personal

vortesso · coffee journal app	2025.10 — 2026.03	31
gloticker · macro indicators Chrome extension	2025.03 — 2025.05	32
underthekey · custom keyboard platform	2024.08 — 2024.11	34

SSAFY

seoulpop · location-based history walks	2024.04 — 2024.05	37
lyricit · real-time game built on song lyrics	2024.02 — 2024.04	39
jansorry · holiday nagging receipts	2024.01 — 2024.02	41
newjibs · real estate investment simulator	2023.11	43
triplet · travel expense diary	2023.09	45

Tools and open source

herdr-cockpit · Herdr layouts and mods for watching orch	2026.11 — Present	47
wiki-island · vendor-neutral AI memory	2026.11 — Present	48
orch · multi-agent orchestrator	2026.09 — Present	49
hermes-agent · open-source contribution to Hermes Agent	2026.08 — Present	50
vat · multi-repository workspace CLI	2026.08 — Present	51
offon · open-source Slack app for attendance and leave	2026.04 — 2026.08	52
sshift · SSH CLI with jump-server connections and password storage	2025.07 — 2025.08	53

Lukuku

space-I · client OS

Period	2026.10 — Present
Team	4 people Server development and shared frontend work
Role	Architecture Go API server Deployment pipeline
Stack	Next.js · TypeScript · Go · Vercel · AWS EKS · AWS S3

Advertising performance and daily sales across 3 overseas markets on one screen

Progress previously shared through separate documents and messages now lives in one place. I built the server and deployment pipeline.

Features

- Client login with an ID and password, staff login through Slack
- Local currency amounts shown with KRW reference amounts
- Schedule timeline and shared materials
- Internal CRM and inbound inquiries

Client data and internal data

Problem Client views and internal CRM needed to share an app while keeping each client's data separate.

Solution Next.js handles the interface while the Go API server handles authentication, DB, files and email. Client staff see only their own company's workspace. Files are isolated in per-client storage with credentials that can reach only that storage.

Branches piling up per client

Problem Writing each client's requirements straight into the code would add another branch with every new client.

Solution Split the server into shared features and per-client work modules. Data is organized in a hierarchy of client, brand, project or campaign, then task, and a rule blocks client-specific hard-coding. The server work is split into shared features and 11 work modules.

Screen changes that needed a developer

Problem PMs had to ask developers to merge and deploy every screen change.

Solution An internal deployment service checks branches every 2 minutes and deploys after validation. A PM's label triggers a merge only for PRs that pass checks. PMs can apply screen changes without developer help, and deployment tokens were removed from repositories.

Lukuku

Shopify B2B store for a US subsidiary · B2B backend

Period	2026.09 — Present
Team	Lukuku project team B2B backend
Role	B2B backend ERP integration server Integration interface review
Stack	Go · chi · pgx · sqlc · goose · OpenAPI · Shopify B2B

B2B server that serves catalogs and prices per account and handles orders from submission to partial shipment

A Shopify B2B store where a global sports brand's US subsidiary takes orders from business accounts. I owned the backend and built the B2B server and the ERP integration server.

Features

- Account, location, product and inventory synchronization
- Account-specific order quantity rules
- Signature-verified webhook storage, retries on failure and quarantine of invalid messages

One server for ERP communication

Problem If the B2B and B2C servers each kept their own ERP client, credentials and integration rules would be scattered. Every ERP-side change would need fixes in each server.

Solution Routed all ERP communication through one integration server. The B2B and B2C servers use only its internal API and events. Changes on the ERP side need updates only in the integration server.

Checking the assumptions behind the requested scope

Problem The development scope the client requested assumed an ERP connector already existed.

Solution Sorted the requested items into reuse, modification, new development and external confirmation. Before development began, I pointed out that the connector did not exist and which work needed a separate contract.

One validation for three order methods

Problem Requirements for per-account access, catalog visibility, bulk orders and prepaid versus postpaid billing arrived at once. Orders also had to come through a matrix, a catalog or product codes and CSV files.

Solution I opened a separate technical review for each policy and answered with three questionnaires that split what the client had to decide from what needed technical checks. All three order methods feed the same cart, so I designed one shared order validation API.

A changed order basis and confirmed policy

Problem In a meeting, the B2B order basis changed from production orders to current stock plus incoming quantities. The change conflicted with work already built.

Solution I recorded change requests apart from confirmed policy and listed where they conflicted with the existing implementation. Changes went through a set order: back up what was built, list the changes, then agree on the scope and schedule impact. Confirmed policy and requests still needing checks no longer get mixed up.

Constraints that mock responses missed

Problem Mock server responses could not reveal the real store's constraints.

Solution Sent synthetic data through the integration server and processor to a development store and checked actual prices and inventory. From the real responses, I found and fixed issues with identifier formats, account update fields and conflicting price list names.

Lukuku

Lukuku OS · company ontology and agent operations

Period	2026.09 — Present
Team	2 people on the internal agent system Architecture and implementation
Role	Ontology design Knowledge graph Answer evaluation Agent operations
Stack	Hermes Agent · MCP · PostgreSQL · AWS S3 · BGE-M3 · launchd

Links original sources from 15 places, including Slack, email and GitHub, to 120+ clients and projects

Staff and agents query the same graph of company information. Agents divide development work among themselves, subject to human approval.

Features

- A source archive that stores and searches more than 500,000 Slack messages

Searching several places to understand a client

Problem Understanding a client required separate searches through Slack, email, documents and repositories. The hand-maintained internal wiki had stopped being updated, with 51 pages left empty.

Solution The system collects 15 sources on a schedule and rebuilds the graph daily. It keeps only relationships with source links and retains the last working graph if collection fails. As of 2026.11, the graph contains about 280 entities and about 4,150 relationships, and every employee has used it since 2026.09.

Ontology: what links to what, and in which terms

Problem Mixing similarly named entities or relationships with different meanings produces incorrect answers. Without fixed relationship names, the same fact was spread across different wordings.

Solution Defined 20 relationship types to specify what can be connected and how. Each task is also linked to who took on which responsibility. This added 100 responsibility relationships across tasks.

Whether the graph can answer questions on its own

Problem A larger graph did not show whether it could answer actual work questions.

Solution Daily checks run fixed questions with expected answers drawn from original sources and locate the step where an unanswered question gets stuck. Relationships extracted from Slack enter the graph only when two different model families agree.

Vendors that lived only in attachments

Problem Payment, logistics and identity verification vendors appeared only in attachment text, not in the graph.

Solution I took proper nouns next to words describing a role as vendor candidates and added only those a person confirmed. I treated a guessed vendor as worse than a missing one. Vendors in the graph grew from 20 to 24.

Actions that are hard to undo

Problem Agents needed to execute work while actions that are hard to reverse remained restricted. One agent once ignored its instructions and replied in a client thread.

Solution Split permissions by stage of work (investigation, implementation and review) rather than by topic. Investigation and review agents only read, and the implementation agent commits but cannot merge or deploy. The agent that watches client channels has no permission to post. Apart from incident response, deployments, external messages and production data changes require human approval.

Architecture

Task coordinator agent
Request sorting · assignment · result collection

Company ontology

- **Collection**
  Slack · email · canvases · attachments · repositories · task records
- **Slack source archive**
  More than 500,000 messages · full-text search · attachment originals with a separate catalog
- **Wiki graph**
  Recalculated daily · only relationships with source links retained
- **Relationship rules**
  Allowed relationships and evidence grades defined in one file
- **Company records**
  Decisions · open questions · memories · human review before updates

Knowledge access

- **offon MCP**
  OAuth · writes within the requester's permissions
- **Answer evaluation**
  Runs fixed questions daily · diagnoses why answers failed

Development

- **Investigation**
  Read-only root-cause analysis
- **Implementation**
  Commits in a separate workspace · no merge or deployment
- **Review**
  Uses a different model from the author's

Task handling

- **Task monitoring**
  Posts tasks from Slack conversations to offon · leaves items with unclear evidence as they are

Request → Task assignment → Implementation and commit → Different model review → Human approval → Post-deployment check



The graph links about 280 entities, such as clients, projects, people and repositories, through about 4,150 relationships.

Lukuku

360° video app and AI video server for an installation art exhibition

Period	2026.06 — 2026.08
Team	Client project On-site app, server and dashboard
Role	Swift on-site app AI generation server Operations dashboard Results website
Stack	Swift · SwiftUI · Python · FastAPI · PostgreSQL · AWS S3 · TypeScript

Unattended on-site app that stitches camera SD card footage and delivers it by QR

Each artwork captured visitors' photos and footage differently. On-site apps and generation jobs stay separate per artwork, while a shared server, operations dashboard and results website serve them all.

Features

- Nalda · Swift on-site app for the balloon-chair experience
- Namu Block · server API creating roughly 15-second videos from images and words
- Bibi Hero · server API creating a character poster from a photo, then a roughly 10-second video with the selected motion
- Reels13 · server job that turns one chosen take out of three 13-second takes into AI video
- 360° balloon experience results website · mobile page for visitors to collect videos by QR
- Operations dashboard · orders, failures, retries and AI costs by location, device and service

Visitors waiting for results

Problem	Manually transferring and editing camera files made visitors wait for their results.
Solution	When the SD card goes in, the app picks only new files, removes duplicates and stitches footage in the background. Stitching and uploading at once could overload the on-site PC, so stitching runs one job at a time. It uploads a preview first to show the QR, followed by the original.

The previous visitor's QR

Problem	The next visitor could scan the previous visitor's QR before it was replaced.
Solution	Replaced the old QR with a standby video after a set interval. Connecting the camera displays copy progress before showing a new QR.

AI providers and failed requests

Problem	External AI services could change or fail. Changing providers should not mean rewriting both the server code and the job queue.
Solution	Chose external services over our own GPUs and kept provider names out of the code so switching providers changes only the connection layer. Failed requests are retried up to 3 times after the first attempt, and authentication, configuration and billing errors are not retried. The same rule is written into the code, the docs and the dashboard. A repeated request does not create a second generation job.

Lukuku

Sales reconciliation and integration operations between a sports brand's Korean online store and its ERP

Period	2026.04 — Present
Team	Lukuku operations team Reconciliation and integration server
Role	Sales reconciliation Reconciliation tool development Integration server operations Settlement corrections
Stack	Python · Shopify · Microservices

Reconciliation tool that compares about 390,000 rows from 58 days of records and sorts differences by cause

I handle month-end sales reconciliation and operate the integration server between the two systems. Among the differences the tool finds, I correct sales and settlements that were actually missing.

Features

- Key-by-key matching of orders, shipments, returns, exchanges, coupons and points
- Settlement correction process organized by how exchange orders lose settlement data
- Voucher prices pulled from the ERP to calculate discounts per order item

Real gaps versus counting differences

Problem The two systems record data differently and at different times, so spreadsheet comparisons mixed real gaps with counting differences.

Solution Sorted differences into ERP-only, store-only and amount mismatches, and added a step that queries both source records again for each difference and attaches a cause. After the partner said the counts looked inflated, I refined the rules separating analysis errors from real differences. Narrowed the records to 16,365 differing rows. The 10,203 rows the partner needed to check and correct went out as 49 files by category.

Timing differences in month-end missing-sales lists

Problem Each month-end list from the partner included sales that had already been sent after the list was generated.

Solution Checked each item against the store's send history to identify the cause. I left already sent items out to prevent duplicates. Resent the missing or partially sent sales data according to ERP rules.

Sales data after a size change

Problem For orders shipped in a different size, the change record, shipping label, shipping status and shipment integration all carried the new size. But the sales data alone reached the ERP with the original size.

Solution Narrowed the cause to the product-change integration logic. I resent the sales data to the ERP and then moved on to fixing the logic. Finished resending the affected sales data.

Monthly settlement checks

Problem Each month the store's orders had to match the ERP settlement. Resending orders with broken settlement data as they were could send duplicates.

Solution For September, I listed 41 orders with broken settlement data and the records missing master data. For October, I re-extracted 72 rows and sent them all except 2 that had to be excluded. I held back further resends that risked duplicates. Finished the September and October reconciliations.

Lukuku

Internal operations agent built on Hermes Agent

Period	2026.04 — Present
Team	Internal agent system Design, implementation and operations
Role	Agent permission design Operations server setup Automated incident handling Model evaluation
Stack	Hermes Agent · Python · macOS · launchd · OpenViking · BGE-M3 · Slack

Agent that splits Slack requests across nine role-specific agents and handles errors through deployment and recovery checks

An internal operations agent that staff call from Slack and Telegram. It runs on Nous Research's Hermes Agent on a single office Mac mini. It assigns each request to the right agent, collects the results and replies in the same thread. I designed, built and operate it.

Features

- Task lookup, task creation and leave requests handled in Slack threads
- A monitoring agent that turns requests in client channels into tasks · no permission to post in client channels
- A personal agent for each employee, refreshed daily from their Slack conversations
- Always on through 33 launchd jobs on one Mac mini · installed files checked against source every 5 minutes

Three fallback models in name only

Problem Once a conversation grew to about 141k tokens, replies took up to 1,221 seconds. Of the three fallback model slots, one had expired credentials and two used the same provider as the main model.

Solution The cause was that compacting the large conversation hit a 300-second limit and failed. Under a rule against paying through API keys, I built a local adapter that calls Claude through a subscription account. I connected it as a fallback path on a different provider from the main model.

Waiting for approval on every error

Problem At first the agent fixed only offon errors, and fixes and deployments needed human approval.

Solution Handed errors from every project to the agent, from root-cause investigation through deployment and a recovery check. For a false alarm, it fixes the detection rule instead of muting the alert. Humans approve only five things: external messages, transactions and refunds, deleting original records, permission changes and updates to company records.

Personal agents citing made-up links

Problem An agent answering on an employee's behalf must not cite links its tools had not returned.

Solution Compared models on nine questions and scored an answer 0 if it used even one link the tools had not returned. In its first real conversation, the agent never called the wiki tools. I updated its instructions to call the tools promptly and follow a specified lookup order. Candidates scored 2.86–4.54 out of 5, and none produced a made-up link.

Company records a day behind

Problem The company decision records were pulled once before dawn, so the agent could not find decisions made that day.

Solution Added record syncing to the collection job that runs every 10 minutes. Confirmed that the same question now gets an answer based on that day's decision.

Architecture

Coordinating agent
Interprets · plans · assigns · hands off · verifies · decides · checks results against the actual source

Knowledge and memory

- **Company records**
  Goals · decisions · operating rules · no updates without human approval
- **Operations settings repository**
  Agent settings · execution rules · kept separate from company records
- **Hermes Agent built-in memory**
  Separate memory for each agent
- **OpenViking search**
  Knowledge and conversation search · results not treated as company records

Execution and control

- **Systems the coordinator owns**
  offon · company records · settings repository · credentials · nothing marked done on a worker's report alone
- **Stage handoff ledger**
  Task · done criteria · owner · review · long runs · audit · retries
- **Scope and approval**
  Limited execution scope per role · workers do not reassign work · external messages need separate approval

Development

- **Investigation** Read-only
  Root causes and technical evidence · no product code changes
- **Implementation** Isolated
  Separate workspace · local tests · no final commit, push, merge or deployment
- **Verification** Independent QA
  Checks test evidence · pass or reject · final changes stay with the coordinator

Operations

- **Slack monitoring**
  Collects public channel messages · posts requests to offon · decisions return to the coordinator
- **Slack source archive** Archive
  More than 500,000 messages · daily collection from public channels except private and webhook rooms
- **Staff records**
  Organizes HR information · acts only within HR scope
- **Personal tasks**
  Personal offon tasks · leave · half-day leave · approval stays with the coordinator
- **Staff answers**
  Read-only offon access · an answer to staff is not company approval

Operations alerts

- **Error alert relay**
  Passes Slack webhook alerts from the error alert channel to the coordinating agent · runs no tasks

Role work → Results, evidence and risks returned → Coordinator checks the source → Applied after approval and checks → Completion confirmed by a fresh query

Lukuku

offon · company-wide work management system

Period	2026.04 — Present
Team	Internal product Design, implementation and operations
Role	Design　Full-stack development　MCP server　Operations
Stack	Next.js 16 · React 19 · TypeScript · Prisma · PostgreSQL · Vercel · MCP · OAuth 2.1
Links	github · attendance-only repository github.com/takealook97/offon

System that handles everything from attendance to CRM in Slack and serves as the MCP access point for AI agents

An internal Lukuku product I designed, built and operate.

Features

- Slack commands for attendance and break records, correction requests and approvals
- Automatic calculations for overnight work, holiday exclusions, leave grants and carry-over
- Table, kanban, calendar and Gantt views of tasks and assignee status
- Reminders asking assignees for updates on the next follow-up date
- Meeting room bookings and reminders 10 minutes before meetings
- Agent search backed by about 390,000 lines of Slack conversation

Requests copied into another tool

Problem　PMs and developers copied requests from Slack and email into a task tool. Most requests that needed checking were emails, and half of them were notification emails.

Solution　Built an inbox that turns requests into tasks with their original text and attachments, or links them to existing work. Messages in the same thread attach to existing tasks, while automated emails are excluded. When the rules first ran, 59 waiting emails were sorted without anyone touching them. Addresses dismissed in two or more different threads are filtered out from then on.

How much access to give agents

Problem　Separate DB and Slack permissions for every agent would scatter access controls. Requester identities supplied in headers could also be forged.

Solution　offon also serves as an OAuth 2.1 authorization server and accepts identity only from the token. Employees use the existing Slack login, and writes stay within the requester's permissions. Employees can use 21 tools with one login and no installation or API key.

Leave requests arriving at the same time

Problem　When the agent handled leave in Slack on an employee's behalf, the same request could arrive twice. Two simultaneous requests to cancel the same leave both succeeded, taking the used leave balance from 1 day to −1 day.

Solution　I also reproduced two requests for the same day both going through with only one day of leave left. Now only one of the overlapping requests is processed and the other is rejected as a conflict. I added 9 checks that run through the API routes against a real PostgreSQL database. In the same reproduction, the second request is rejected with 409.

A feature users could not find

Problem　Task assignment rules lived only on the server, so users could not find the feature on screen and only saw errors.

Solution　I surfaced the rules in the interface too. I now treat a feature people cannot find as a missing feature. Fixed on the day it was reported.

Lukuku

Participant-only tournament voucher ordering for a sports brand's Korean online store

Period	2026.04 — 2026.09
Team	3 people Backend
Role	Voucher ordering API Order validation Duplicate payment prevention Voucher close-out operations
Stack	Python · Shopify

Ordering backend that accepts only eligible products at the voucher price and blocks duplicate payments by the same member

Participants in a tennis club tournament run by the brand could buy at a discount with a voucher on a members-only page. I built the voucher ordering backend for it in about a week before the tournament.

Features

- API that returns only the products eligible for the voucher price
- Order validation that checks each item's style and color against the voucher's item list and blocks anything else
- Bundle products that showed as unavailable in staging tests, matched to their Shopify admin state

A voucher code shared by many

Problem	Many participants shared the same voucher code, so it could not identify duplicate payments.
Solution	Defined the cart key as a hash of the voucher code and the member ID. The same member with the same voucher always gets the same cart, and duplicate payments are rejected.

Closing the voucher period and held orders

Problem	Orders placed during the voucher period were held instead of being sent to the ERP. When the period ended, every held order had to go through.
Solution	After the period ended, I switched the voucher to inactive, released the hold and started the ERP integration. All 13 held orders reached the ERP in about 2 seconds, and no held orders remained.

Lukuku

Shopify store features and integrations for brands

Period	2026.04 — 2026.07
Team	Client maintenance Feature development
Role	Batch jobs Shopify apps and extensions Integration APIs
Stack	Shopify · Shopify Checkout UI extensions · React · Kakao AlimTalk

From AlimTalk reminders for unused coupons 7 days before expiry to marketplace order integration

I maintained client Shopify stores and built the features and external integrations each brand requested.

Features

- Two brands run by a sports group's Korean subsidiary · coupon-expiry AlimTalk batches
- Underwear brand's global store · checkout extension and localized text
- European sleep brand's Korean subsidiary · backend design and test environment for collecting domestic marketplace orders in Shopify
- Case brand · lookup API matching store product options to ERP items
- Beauty brand · App Store submission work for a quantity-discount app and cart support for a new theme

Skipping coupons already used

Problem	A customer could use a coupon after the batch selected recipients but before the notice was sent.
Solution	Rechecked coupon usage immediately before sending and skipped used coupons. The two brands started at different times, so each brand got an on/off setting and one went live first. The two expiry templates match the Kakao-approved body text and button URLs character for character. It went live for both brands.

Saving checkout policy consent

Problem	An underwear brand's global store needed to collect and save policy consent at checkout.
Solution	Added a consent checkbox with a Shopify Checkout UI extension and fixed a race condition when saving the value.

Timeouts during bulk collection

Problem	Collecting many product options in one request caused timeouts.
Solution	Split collection into smaller batches and saved each batch to avoid timeouts.

Biztalk

AWS MSK 3.8 migration

Period	2025.08 — 2025.09
Team	Solo
Role	Architecture design KRaft adoption in development and production SASL authentication Topic and replica configuration Message sending tests Kafka UI monitoring
Stack	AWS MSK 3.8 · Kafka KRaft · SASL/SCRAM · AWS Secrets Manager · Kafka UI

Configuration for about 160 topics recreated automatically on a new cluster

Support for production MSK 3.5 was set to end in 2025.10, so I built a new 3.8 cluster using KRaft metadata and recreated the existing topics on it.

Choosing the authentication method

Problem	The new cluster needed server authentication added. MSK supported IAM, TLS and SASL, among others, so the choice had to weigh compatibility with the internal environment against operating complexity.
Solution	After reviewing the options, I chose SASL/SCRAM. User accounts are stored in AWS Secrets Manager. Each server is wired to Secrets Manager and reads the account from it to authenticate.

Automating topic recreation

Problem	Every topic on the old cluster had to be recreated on the new one with the same settings. Partition counts, replicas and options differed by topic, so copying them by hand risked omissions and errors.
Solution	Exported the topic settings of the old cluster to a CSV file in one pass and reworked the replica configuration for topics that needed changes. A script reads the CSV and creates every topic on the new cluster. Topic names, partition counts, replicas and options carried over as they were, which greatly cut the time spent copying them by hand.

Biztalk

BGMS log masking module

Period	2025.07
Team	Solo
Role	Planning Module design Redis bottleneck fix Development and testing Patching
Stack	Java 8 · Spring Boot 2.7 · Redis · Gson · JaCoCo

Local caching avoids more than 25 Redis lookups per log entry

After the SK Telecom breach, some customers asked for log masking in addition to phone number encryption. It had to apply to 14 types of MSA servers, so I built it as a module that any server can add.

Features

- Checks whether a customer needs masking first, then wraps the log object before it is passed on and masks it inside the module

Redis bottleneck

Problem	Each log entry queried Redis to check whether the customer required masking. A single sending log averaged more than 25 Redis lookups, so the risk of a bottleneck grew quickly.
Solution	Added a local cache inside the module, so cache hits skip Redis. The list of customers that require masking is synchronized locally every day to stay current.

Finding fields in nested objects

Problem	Values passed to the logger were not plain strings but nested objects, lists and JSON. Finding the fields to mask meant searching every level, which getters and setters could not do.
Solution	Serialized the log object with Gson and traversed the JSON tree with DFS. During traversal, the bsid decides whether the customer requires masking, and fields beneath it such as recipient, phone_number and message are masked. Because fields are found the same way regardless of object structure, the module could be added to any server.

Biztalk

Overdue payment notice RPA

Period	2025.06 — 2025.07
Team	Solo
Role	Planning DB and flow design PDF generation automation Mail and AlimTalk notice automation Design Client page
Stack	Java 17 · Spring Boot 3.5 · MyBatis · MySQL · poi-ooxml · LibreOffice · JavaMailSender · Next.js 15 · React 19

Overdue notice work reduced from over 2 days to 10 minutes

At the start of each month, the planning team sent email and AlimTalk notices to overdue customers, typing into Excel and making PDFs by hand. I took on an RPA project, from planning onward, to move that work onto one page.

Features

- Overdue customer lookup, receivable registration and deletion on one page
- PDFs generated per customer with their overdue amount and due date
- Email and AlimTalk notices sent with one button

A different PDF and mail body for every customer

Problem Each month, the different amounts and due dates of more than 100 overdue customers were typed into an Excel template, exported to PDF and written into each mail body by hand.

Solution Mapped the template cells that change (document number, issue date, recipient, billing month, overdue amount and due date) to constants. poi-ooxml fills those cells with DB values, and LibreOffice turns the file into a PDF. The mail subject and body are filled in per customer, and JavaMailSender sends them with the PDF attached. Overdue notices that took over 2 days now take 10 minutes.

PDF conversion on a server without install rights

Problem Aspose was the first choice for converting Excel to PDF, but its free license stamped red text at the top of every PDF. I also had no permission to install software on the production server.

Solution Switched to LibreOffice and confirmed the conversion worked. Instead of installing it, I uploaded it as an AppImage and started the conversion process from the WAS with the headless option. Korean text in the PDFs came out broken, so I also uploaded the Malgun Gothic font, which the server lacked.

Regenerating the same document

Problem Requesting a document again under the same conditions could create the same file again and waste resources. Because these were IDC servers without S3, files had to be stored on the server's disk.

Solution Built a 16-character hash from the customer ID, issue date, billing month, due date and overdue amount. A document log table uses that hash and the file extension as its composite key, and files are stored in a directory per hash. Documents already in the log are reused, and new ones are created only when missing.

Progress during a bulk send

Problem The screen did not show progress when sending notices to many customers at once.

Solution Stored logs for each sent item and streamed progress events over SSE. Email and AlimTalk sending ran in parallel with 4 core threads and a maximum of 8.

Biztalk

BGMS messaging system development and operations

Period	2025.05 — 2025.10
Team	SD team BGMS development and operations
Role	Brand message features MSK Realtime topic setup Related logic refactoring Service operations
Stack	Java · Spring Boot · AWS MSK · Redis

Feature work and operations for a messaging system sending about 400 million messages a month

BGMS is the messaging system that sends Kakao AlimTalk and brand messages. Its 14 Kafka-based microservices run on 54 instances. It sends about 400 million messages a month on average and handles 2,000 per second at peak. I developed features for it and ran it throughout my time at Biztalk.

Features

- Blocking feature for KakaoTalk brand messages
- Reprocessing logic for KakaoTalk brand message targeting (N, I)
- Realtime topic in MSK and refactoring of the related logic

Biztalk

Deployment pipeline setup and internal GitLab migration

Period	2025.05 — 2025.10
Team	SD team Deployment pipelines and QA servers
Role	Jenkins pipelines AWS CodeDeploy deployment ELB redundancy QA server setup GitLab migration
Stack	Jenkins · AWS CodeDeploy · AWS ELB · Nginx Proxy Manager · Portainer · GitLab

17 deployment pipelines for QA and production environments

I built Jenkins deployment pipelines for MSA QA servers and a new commerce business. The new business's servers run redundantly behind an ELB.

Features

- Pipelines split into 13 for MSA QA servers and 4 for the new commerce business's QA and production (AWS CodeDeploy)
- Nginx Proxy Manager and Portainer on the QA servers
- Internal GitLab server migration and upgrade from version 11 to 18

Biztalk

Server configuration management and deployment scripts

Period	2025.05 — 2025.10
Team	SD team Server configuration and deployment scripts
Role	Configuration standards Patch automation Deployment scripts Configuration encryption and decryption tools
Stack	Shell script · scp

Configuration standards and patch automation for more than 50 production servers

I moved configuration and deployment for BGMS production servers into scripts. I also built tooling that encrypts sensitive values in configuration files and rolled it out company-wide.

Features

- Configuration versioning
- Configuration encryption and decryption library and scripts
- Internal encryption and decryption API server

Manual deployments to IDC servers

Problem	Some projects on IDC servers were deployed by uploading jar files with an FTP program and starting them with server scripts. Someone had to remember the order of steps, and the program often slowed down, so directories had to be refreshed again and again.
Solution	Combined build, transfer, stopping the old process, restart and health check in one scp and shell script. The previous jar and log are backed up as a timestamped tar.gz, and environment files are read from the server's conf directory. After a restart, the script checks /ping up to 20 times and sends a push notification on success or failure.

Biztalk

KakaoTalk sending and template statistics collection server

Period	2025.05 — 2025.06
Team	Solo
Role	DB and flow design Batch scheduler Aggregation API Re-collection API Data reprocessing scheduler
Stack	Java 17 · Spring Boot 3.5 · Spring Batch · MyBatis · MySQL

Sending statistics for over 8 million messages a month

The server covers AlimTalk, FriendTalk and brand messages. The statistics table was denormalized around user IDs to speed up queries.

Statistics that differ by elapsed day

Problem	Read and click statistics from the Kakao API arrive aggregated separately per elapsed day (0, 1, 2). Combining them in one pass produced wrong totals and broke data consistency.
Solution	Every day, the batch pulls all aggregated statistics from the Kakao API. Read and click values are accumulated in three tiers by elapsed days. The batch writes 500 rows at a time to speed up DB writes.

Duplicate rows on re-collection

Problem	A manual re-collection API, added for cases where the scheduled batch missed data, made it possible to accumulate the same data several times.
Solution	In the denormalized final table, I built a composite key from the main columns, leaving out those that change. An elapsed_mask column marks elapsed days 0, 1 and 2 as bits 1, 2 and 4. Accumulation checks these bits, so the same elapsed day is never added twice. Collecting the same data again creates no duplicate rows, and the statistics stay consistent.

— deepeet

AI-Native commerce run by agents

Period	2026.07 — Present
Team	Design and build of the product and agent operations system
Role	Co-founder · Builder Product design Agent operating model Backend Operations tooling
Stack	Go · Next.js · PostgreSQL · Hermes Agent
Links	deepeet.com

Reached 700+ orders and over ₩10M in revenue within three months through automation.

deepeet started as a consignment business selling fruit from wholesale suppliers on sales channels. The first piece was order automation, which turns sales-channel orders into purchase orders with wholesale suppliers. Routing, pricing, settlement, sourcing, listings, customer support and advertising followed, each built in-house on top of it. AI agents decide from evidence, and the commerce system executes only after checking permissions and safety conditions. Each result is then checked against the official status at the sales channel or supplier. Important code changes are verified by a different model from the one that wrote them.

Completion reports versus actual state

Problem	An agent's completion report can differ from the actual system state.
Solution	The lead reviews the report, and the system executes only procedures that pass the rules kept in vat. The actual state is checked again afterwards. All 19 underlying repositories follow the same rules.

Architecture

Lead agent for business operations
Single point of contact for the user · task assignment · result checks · final decisions

Knowledge and memory

- **Business records**
  Review before changes
- **Lead agent records**
  Principles · workflows · operations journal
- **Conversation memory**
  Summarized conversations and user information
- **Search index**
  Search material derived from records · not used as evidence

Execution and control

- **Commerce systems**
  Order automation · routing · pricing and settlement · sourcing · listings · operations console · advertising
- **Task board**
  Long-running work, handovers and retries
- **Release process**
  Changes across 19 repositories · validation of what changed · checks after deployment
- **Permissions**
  L0 lookups, classification and drafts · L1 running defined procedures
- **Unattended operation**
  Messenger connection monitoring · automatic start after reboot · encrypted backups

Development

- **Investigation** L0
  Root causes and technical evidence
- **Implementation** L0
  Work within assigned tickets
- **Review** L0
  Quality, regression and security checks

Business operations

- **Product** L1
  Planning, detail pages, listings and verification
- **Customer support** L0
  Classifying text inquiries and drafting replies
- **Finance** L0
  Source lookups, reconciliation and reports
- **Marketing** L0
  Advertising and marketing decisions
- **Supplier monitoring** L0
  Checking and organizing supplier notices

Agent work → Result report → Lead review → System action after permission checks → Actual state check

— deepeet

Advertising operations

Period	2026.10
Team	Solo
Role	Ad integrations Performance and change history collection Ad spend ledger Ad-driven order tagging
Stack	Go · PostgreSQL · TypeScript · Next.js

Handles Toss Shopping ad performance and changes on one screen and brings ad spend and ad-driven orders into profit

Each channel handles ad accounts, campaigns and billing differently. For Toss Shopping, an isolated runner per store reads daily performance, campaigns, ad groups and ads, and channel change history. Orders that came in through ads are identified from the benefit details in official order lookups and tagged. Coupang ads are handled through the same console menu and account structure. Paid billing and ad spend are recorded separately.

Features

- Toss Shopping daily ad performance, campaigns, ad groups and ads, and channel change history
- Tags for orders that came in through ads, with expected profit using the fee rate applied to each order
- Store profit with ad spend included
- A ledger and duplicate protection for change jobs, with changes of unknown outcome never resent
- An ad spend ledger that does not count ad top-ups as costs
- A Coupang ads menu and account structure

Ad top-ups mistaken for ad spend

Problem	Booking ad wallet top-ups directly as costs makes ad spend and profit drift from what was actually spent.
Solution	A ledger records wallet top-ups, billing sources and actual costs separately. Store profit subtracts only ad spend that was actually incurred.

Ad changes with unknown outcomes

Problem	If the response to a campaign change is lost, resending it could apply the same change twice.
Solution	Change jobs are kept in a ledger where entries are never deleted, and identical requests map to the same job. Each job has an assigned runner and processing order, so only one runner submits it. Changes with unknown outcomes are not resent and are confirmed through official lookups.

Fees on orders that came in through ads

Problem	Toss Shopping applies a different product sales fee to orders that came in through ads. Without telling them apart, expected profit drifts from the real figure.
Solution	Ad-driven items are confirmed from benefit evidence in official order lookups and tagged in order history and purchase tracking. Expected profit uses the fee rate applied to each item. The ad-driven tag appeared on live order screens and purchase tracking.

Connection status before an official lookup

Problem	If an account shows as connected before an official lookup succeeds, an empty screen or another account's performance can look like real data.
Solution	Only accounts with a successful official lookup show as connected, and channels without lookup results stay empty. When an account changes, the connection is dropped and earlier performance is not attached to the new account.

— deepeet

Pricing, promotions and settlement

Period	2026.08
Team	Solo
Role	Pricing policy Promotion execution Settlement and refund ledger Cash ledger Profit reconciliation
Stack	Go · PostgreSQL · TypeScript · Next.js

Updates channel sale prices to reflect supply costs and fees, and reconciles profit against settlements and actual cash

Supply costs for fresh food change often, and each channel has its own list prices, coupons and period prices. Target-margin sale prices are calculated from supply cost, fees and VAT. After each update, an official channel lookup confirms the price. Regular sale prices and promotion prices are managed separately. Channel settlements, refunds and bank transactions each go to their own ledger to reconcile expected and actual profit.

Features

- Price updates calculated from supply cost, channel fees and VAT for a target margin, confirmed by official lookups after each update
- Separate policies for regular and promotional prices, including Coupang coupons and instant discounts and Toss Shopping limited-time prices
- Price freezing that fixes listed sale prices and bounds promotion prices
- An append-only settlement ledger for official Coupang and Toss Shopping settlements that deducts only official refunds from order profit, exactly once
- Cash ledger that separates bank statement files from supplier deposits by location
- Profit reconciliation that shows profit per product and option, plus unallocated costs

Channel prices that drifted from policy

Problem	Even after a pricing policy changed, the price actually applied on a channel could differ, and old jobs overlapped with new ones.
Solution	Policy intent and channel results are recorded separately, and the latest evaluation closes earlier jobs for the same policy. Each update is confirmed by an official lookup and reapplied when needed.

Rate limits versus ordinary rejections

Problem	When prices changed in quick succession, the channel rejected requests for rate limiting, but these were closed as ordinary rejections and the cause was lost.
Solution	Each seller account has a minimum interval between price updates, and only confirmed rate limits are rescheduled. Rejection codes and reasons stay in the record. Temporary limits recover, while unknown outcomes and ordinary rejections are never resent automatically.

Customer prices that differed from channel calculations

Problem	Channels round discounts differently and cap percentage coupons differently, so calculated customer prices did not match the real ones.
Solution	Customer prices are calculated with each channel's actual rounding, and percentage coupons get a separate cap on the maximum discount.

A rejected bank integration

Problem	The open banking application for receiving bank transactions directly was rejected.
Solution	Encrypted transaction files downloaded from the bank are uploaded and imported instead. Transactions with the same time and amount are told apart, and categories corrected by hand are preserved.

Settlement profit mistaken for final accounts

Problem	Profit calculated from channel settlements could be read as the final accounting result.
Solution	Settlement-based profit and final accounting are kept apart. Costs without supporting records are never filled with zero. The screen states that profit is calculated on a settlement basis.

— deepeet

Routing and sales status control

Period	2026.07 — 2026.09
Team	Solo
Role	Routing policy design Product identity matching Automatic sales status control Stock maintenance
Stack	Go · PostgreSQL · TypeScript · Next.js

Picks the supplier whose specs match each sales option exactly, and pauses sales when no purchase route is left

The same fruit is labeled with different variety, weight, count and grade at each supplier. Each sales option shows its current purchase route and alternatives, and the operator can pin, prefer or hold a route. Only products whose specs match exactly are chosen, not candidates with similar names. Options that lose their purchase route stop selling and reopen when a route comes back.

Features

- Routing policy with PIN, PREFER and HOLD plus candidate exclusions on top of automatic selection
- Matching that identifies the same product by item, variety, weight, count, grade, size and channel option IDs
- Fallback from expired preferred routes to the remaining eligible candidates
- Automatic sales pause for options that lose their purchase route, and resumption once it recovers
- Per-channel relisting thresholds and stock maintenance for options the operator turned on

Matching products that names alone could not identify

Problem Channel weight labels and supplier count rules differed, and variety aliases and grade names varied by supplier.

Solution Channel weight and purchase count rules are stored separately, with rules for variety aliases and merged grades. Routes are chosen by the specs of what the customer ordered. Orders do not go to supplier products that only look similar by name.

Scattered options from past orders

Problem Placeholder options from past Coupang orders made one sales option appear as several.

Solution Placeholders were grouped under Coupang's immutable option IDs to clean up the option list.

Options that kept selling without supply

Problem Options with no purchase route after a supplier sold out could keep selling on the sales channel.

Solution Options that lose their purchase route stop selling automatically and reopen when a route returns. Sales status changes only on an operator action or a lost route.

Choices that could not be explained later

Problem Candidate lists kept changing, so there was no way to explain later why a past order went to a given supplier.

Solution The evidence at selection time is kept as an immutable record, and current candidates are stored separately.

— deepeet

Agent operations and organizational memory

Period	2026.07 — 2026.08
Team	Solo
Role	Role and permission design　Unattended operation and recovery　Incident records and alerts　Organizational memory
Stack	Hermes Agent · Codex CLI · Python · Shell · Telegram · Discord · launchd · OpenViking · Git

A lead agent assigns work to role-specific agents, and keeps incident and business records across restarts

The lead agent runs continuously on Hermes Agent and handles infrastructure monitoring, incident investigation and operating decisions. Role agents divide up accounting, development, listings, customer support and advertising. Execution is controlled by actual tool permissions and completion conditions, not role descriptions. Incidents are kept as records that survive restarts, and reviewed business facts are kept as Git documents.

Features

- A machine-readable permission list per role covering allowed work, forbidden actions and result and retry rules
- A hook that checks tool calls in advance so the finance role can use only designated lookup scripts and read-only tools
- Incident records and alerts that log incidents under fixed IDs per cause and do not count restarts as new incidents
- Unattended operation that starts automatically after outages, reboots and automatic updates, and confirms messenger connections
- Git documents holding reviewed business facts, with conversation memory and search indexes rebuilt from them
- Search that looks in current-state documents first for questions about the present and never presents past decisions as current facts

The same incident alerting again on every redeploy

Problem　Each restart or redeploy sent the same supply collection failure again as a new alert.

Solution　Incidents are recorded under fixed IDs by supply scope and cause, keeping the last success, the start of consecutive failures, occurrence count and alert status. Logs going quiet is not treated as recovery; recovery needs official evidence. Each incident alerts once and closes in the thread where it was first reported.

Messenger connection lost after an automatic update

Problem　After an automatic update, the agent restarted its messenger connection service itself and lost the connection entirely when the service registration went missing.

Solution　A separate runner handles updates and records success only when the service registration and the running process are both confirmed. An independent watcher checks the state and recovers it. In a test that took the service down on purpose, the watcher recovered it with a new process, and Telegram and Discord reconnected.

Recovery after outages and reboots

Problem　After an outage or reboot, the agent and search server stayed down until someone logged in.

Solution　The machine boots automatically when power returns, and after auto login it starts the agent, the search server and job workers. A post-boot script checks the processes, messenger connections and search server state. A real reboot passed all post-boot checks.

Search that made past decisions look current

Problem　Search results mix past and current decisions.

Solution　Questions about the present search current-state documents first and rank them above older records. For important decisions, the agent rereads the source, and archive folders are left out of default search.

Architecture

Lead agent
Single entry point for requests · final decisions

Interpret → Plan → Assign → Execute → Verify → Decide

Knowledge and memory

- **Business record**
  Goals · decisions · status · roadmap
  Never changed automatically outside the defined procedure
- **Agent rules**
  Principles · workflows · operating log kept in Git
  Kept apart from the business record
- **Runtime memory**
  Built-in Hermes Agent memory · fallback when the search layer is down
- **Derived search**
  OpenViking · re-indexes Git documents and conversation memory
  Never used as the record of truth

Execution and control

- **Task-specific execution systems**
  Commerce backend · operations console · sourcing collector · infrastructure · DB · each system checks permissions before it runs anything
  A role agent's report alone never marks work done
- **Task ledger**
  Only for long-running work, handoffs, retries and audits
- **Permission levels** L0 · L1
  L0 read · classify · draft / L1 run a step of a defined workflow
  A level alone grants no external writes
- **Operations alerts**
  Outage alerts are a signal that wakes the lead agent
  Alerts never make the decision

Development-only roles

- **Root-cause research** L0
  Reads code, logs and official docs to gather evidence
  No edits or state changes
- **Ticket implementation** L0
  Implements, tests, commits and verifies locally within the ticket
  No merges or deploys
- **QA and security review** L0
  Judges finished changes for regressions and security
  No implementation or release decisions

Domain roles

- **Product listing** L1
  Product planning · detail pages · official re-check after listing
  No direct DB or channel writes, no purchase orders
- **Customer messages** L0
  Classifies customer messages · gathers facts · drafts replies
  No sending, refunds or compensation
- **Finance lookup** L0
  Reads and reconciles revenue, cost, settlement and cash-flow sources
  No transfers or orders · unconnected sources are never filled with zero
- **Ad decisions** L0
  Ad and marketing decisions from confirmed evidence
  No spend, budget or bid changes
- **Supplier notices** L0
  Observes and normalizes supplier notices · keeps the source
  No price, stock or order changes

Result flow

Role agent runs → Result in a fixed format · evidence, checks, risks → Lead verifies against real sources
→ Business system runs after a permission check → Official re-check confirms completion

— deepeet

Sourcing data

Period	2026.07 — 2026.08
Team	Solo
Role	Supplier price collection Market and keyword analysis Sourcing recommendations Browser extension Supplier notice collection
Stack	Python · Playwright · PostgreSQL · MinIO · NumPy · TypeScript · Chrome Extensions

Combines wholesale supplier prices with search demand

and competition to choose what to sell among fruit that can be supplied

Choosing a supplier for each order first required knowing each supplier's products and prices. Collection started with wholesale supplier prices and grew to include Naver search demand, competition metrics and related keywords. These sources feed recommendations for what to sell next among items that can actually be supplied. Notices from supplier group chats are also collected so supply stops and price changes can be acted on.

Features

- Collection that reads official supplier spreadsheets first, then APIs and web pages, then keeps the last successful copy
- Supplier price ledger normalized by item, variety, weight, count and grade, keeping the original product names and prices
- Market analysis that stores Naver monthly top keywords, daily trends and a keyword ontology separately
- Sourcing recommendations shown with evidence on supply, demand, competition and timing
- Competitive price watch that extracts representative prices per item and option from popular Toss Shopping products
- A Chrome extension that structures and saves only the Coupang search and product pages a person opened
- Notice collection that preserves supplier KakaoTalk notices and ties return and exchange limits to supplier products and purchase candidate checks

One bad price row blocking the whole price list

Problem A single empty value or invalid price in a supplier file caused collection of the entire price list to fail.

Solution File-level structure checks run separately from row-level isolation, so only valid rows are updated. Bad rows keep the last valid price and observation time. Valid data is never lost, and unchecked prices never look fresh.

Catalogs that suddenly went empty

Problem If collection returned an empty or much smaller supplier catalog, it could overwrite the valid price list.

Solution Empty and sharply shrunken catalogs are treated as partial data, and the last valid copy is kept. A temporary collection glitch does not erase purchase routes.

Weight and packaging crowding the rankings

Problem Variants that differed only in weight, packaging or grade filled the top recommendation slots instead of distinct items.

Solution Weight, packaging and grade were moved down to options rather than item candidates. Seasonality and demand trends are calculated separately.

Late-rendering reviews and lost observations

Problem Reviews and ratings rendered later than names and prices, so empty values were saved, and observations that failed to send were lost.

Solution The extension watches card changes and reads once the page settles. Observations that fail to send stay in an outbox and are sent again. Late-rendering pages are read without endless waiting or scrolling.

— deepeet

Development and infrastructure operations

Period	2026.07 — 2026.10
Team	Solo
Role	Cross-repository contracts Impact-based validation Release process Server setup Backup and restore
Stack	Git · Python · Shell · Proxmox · Docker · Jenkins · PostgreSQL · MinIO · Cloudflare R2 · SOPS · Prometheus · Vercel

A release process that coordinates changes across 19 repositories with vat
and checks production state and business records after deployment

The backend, console, collectors and agent runtime are split across 19 repositories. vat manages changes that span repositories and the contracts between them, and only what changed is validated. A single Proxmox server hosts the API, PostgreSQL, MinIO and Jenkins separately, and the console runs on Vercel. Backups are encrypted, stored in Cloudflare R2 and confirmed by restoring them. Credentials are encrypted with SOPS.

Features

- Records that bundle changes across repositories with participating commits, validation and rollback points
- Contracts that declare the integrations between repositories and check them by machine
- Validation that selects checks from the base commit, current file hashes and dependencies, and blocks when the impact is unclear
- A release process that ends only after code checks, push, deployment, production lookups and business record updates all pass
- Jenkins deployment that runs validation, image transfer, migrations, service start and a check of the running commit
- Backups that encrypt PostgreSQL dumps and required objects, store them in Cloudflare R2 and confirm them by restoring into an isolated environment
- Credentials encrypted with SOPS and kept in Git

Changes to tables another repository reads

Problem Collector SQL that reads backend tables directly can break silently when backend migrations change.

Solution Tables, columns, types and allowed values in collector SQL are checked against the migrations, and dynamic SQL of unknown shape is blocked. Live database checks and schema comparison run separately.

Deployments that left the records behind

Problem If only the product is deployed and the business record stays local, the next person cannot tell what is actually deployed.

Solution Work ends only after code checks and push, deployment within the allowed scope, production lookups and business record updates all pass.

Backups never decrypted

Problem Having backup files does not show whether they can actually be decrypted and restored.

Solution A backup counts as done only after an offsite copy is made, decrypted and restored into an isolated environment.

Browser zombie processes exhausting PIDs

Problem Chromium child processes left by browser automation were never reaped, and the zombies used up the PID limit. Health checks and request threads could not start.

Solution An init process was enabled in the container, and each deployment checks that the real configuration has it turned on.

— deepeet

Operations console and customer service

Period	2026.07 — 2026.08
Team	Solo
Role	Operations screen design Server relay layer Permissions and business boundaries Customer texting Claim cases
Stack	Next.js 16 · React 19 · TypeScript · Tailwind CSS 4 · Vercel · Go · PostgreSQL

An operations console for orders, purchasing, pricing and settlement, with customer texts and claims linked to each order

Orders, purchasing, sale prices, listings, inquiries, refunds, settlement and cash are viewed and handled across 41 work screens. The interface displays backend responses without calculating amounts or determining workflow stages itself. Customer texts go through a handset on a line deepeet runs itself. Inquiries, returns, exchanges and refunds are grouped into claim cases per order.

Features

- Three layers of permissions: menu access, admin-only actions and a server relay that passes only allowed paths
- Switching between several businesses in one console, with orders, accounts and supplier execution isolated per business
- A sensitive-data mode that masks contact details and order numbers on screen, and layouts for narrow mobile screens
- A texting gateway that sends and receives customer texts through a self-run handset line and links them to orders
- A blacklist that places new orders on hold when customer details match
- Claim handling that groups texts, returns, exchanges and refunds into order-level cases and gets AI classification suggestions from masked input

Two-way texting that overseas services could not provide

Problem Overseas messaging platforms did not support two-way texts for Korean numbers, so customer replies could not be received.

Solution A handset on a self-run line serves as the gateway for texts and picture messages. Every sender address is preserved as received, and blocked senders are dropped before storage.

A relay layer that keeps failure types apart

Problem If the relay layer lumps errors together, timeouts, user cancellations and transport failures become indistinguishable.

Solution Timeouts return 504, user cancellations 499 and other transport failures 502. A request ID links to backend records, while URLs, bodies and headers stay out of logs. Network errors are not treated as logouts.

Photo request texts racing a cancellation

Problem If a customer replied or a cancellation arrived after a photo request was scheduled, an unnecessary text could go out.

Solution Right before sending, the case and the request are locked in order, and replies, refunds and the handling route are checked again. If the cancellation commits first, nothing is sent; if sending is confirmed first, only the remaining requests are cancelled. PostgreSQL concurrency tests verify both cancellation-first and send-first execution.

Return collection stopped by a new amount format

Problem A channel's return response sent the shipping fee as an amount object instead of a number, so the whole list failed to parse and collection stopped.

Solution Objects, numbers and numeric strings are all accepted, and failures are isolated per collection method. Diagnostics keep only field names and types. Collection resumed after deployment, and missed events were backfilled without sending alerts again.

— deepeet

Product listing

Period	2026.07 — 2026.08
Team	Solo
Role	Listing agent design Listing state flow Detail page generation and checks Channel publishing and verification
Stack	Go · PostgreSQL · MinIO · Hermes Agent · Node.js · Playwright · JSON Schema

The listing agent drafts the plan and detail page assets, then products that pass the checks are published on two channels and confirmed through official lookups

The listing agent handles one product from planning to confirmed publishing. From real product photos and the backend's price calculations, it drafts the product plan and detail page assets. The plan covers the product name, options and search terms. Once the defined procedure passes, the backend lists the product on Coupang and Toss Shopping. The agent then confirms the actual listing state through official channel lookups. The first real product was listed on both channels and its publishing confirmed.

Features

- A state flow that ties the plan draft, detail page assets, publishing on two channels and official lookup checks to the same listing
- Planning based on three different real product photos, with unsuitable photos swapped for other candidates
- A precondition that each option has at least two purchase routes before automatic listing
- Agent permissions that block bypassing the defined procedure, direct database or channel writes and supplier orders
- Manual listing records that keep the operator's chosen photos and listing inputs with hashes

Detail images that looked like different products

Problem Each detail page image was generated separately, so the same fruit could look like a different variety from one image to the next.

Solution The researched variety, skin and shape went into every image instruction unchanged. Each generated image was checked with a vision model to confirm it showed the same product. A job completed only when every image passed the check.

Image jobs that could start without evidence

Problem Without enough real photos, images could be generated from a different variety or a synthetic image.

Solution Before queuing an image job, the backend checked that the same listing had at least three different real photos. It stopped the job when the evidence was synthetic or generic, or when the research had changed. Jobs with mismatched evidence stopped before any paid generation.

Paid retries with unknown outcomes

Problem If a job finished but its completion response was lost, the same images could be generated and paid for again.

Solution Before a retry, the worker first checked the job state in the backend and the stored result. With a stored result it resent only the completion report, and with an unknown outcome it did not rerun paid generation automatically. Jobs with a stored result were not generated again.

Edits that changed untouched images

Problem Even a request to fix only the copy could change images and composition, and regenerating cost money again.

Solution Edits were split into three modes: chosen images, automatic selection and copy only. Unselected images were copied unchanged from the existing result, and only failed images were regenerated. Copy edits required no image generation, and hash checks confirmed that the original images were unchanged.

deepeet

Order automation

Period	2026.07 — 2026.08
Team	Solo
Role	Workflow definition and domain design Sales channel and supplier integrations Unattended purchasing and recovery Purchase tracking
Stack	Go · chi · PostgreSQL · pgx · sqlc · goose · OpenAPI · MinIO · Xvfb

This was the first thing deepeet built. It carries each order from customer payment to purchase order, tracking number and confirmed delivery.

deepeet started as a consignment business selling fruit from wholesale suppliers on Coupang and Toss Shopping. Each order needs a supplier, a purchase order and payment, and the supplier's tracking number registered back on the sales channel. This was the first job deepeet automated. The first paid customer order went to a supplier automatically, and the flow was then extended through confirmed delivery.

Features

- Order collection that stores sales-channel orders in one shared model without duplicates
- Dry-run purchasing that links order items to supplier products and shows candidates and costs without placing orders
- Purchase files that follow each supplier's spreadsheet template
- Unattended purchasing, payment and tracking collection with a job queue and duplicate-order protection
- Tracking through the sales channel's official delivery confirmation
- Pause and resume of automatic purchasing and payment per order or option

Purchase orders and payments with lost responses

Problem	When a response is lost after sending a purchase order or payment to a supplier, there is no way to tell whether it went through.
Solution	Requests with unknown results are marked as such and are not resent until an official supplier lookup confirms the outcome. They are not resubmitted to another supplier either. Suppliers where submission includes payment and those with a separate payment API are recorded separately. Retries cannot place or pay for the same order twice.

Handoffs lost to outages and network drops

Problem	If the server stopped after reading a tracking number from a supplier but before registering it on the channel, the next step was never created.
Solution	External lookup results are recorded together with the next processing step. If the server stops, processing restarts from the last point saved in the database. Remaining purchase and tracking handoffs continue even after a long outage.

Orders that lose money

Problem If supply costs rise after a sale price is set, every new order can deepen the loss.

Solution Just before submission and payment, the system recalculates from the actual payment minus fees, VAT and supply cost. Purchasing is blocked when the contribution margin is negative or cannot be verified.

Bot blocking on supplier logins

Problem When a supplier site blocked automated access at login, purchasing and tracking collection stopped.

Solution A browser runs on a virtual display on the server to recover the login, and the session is handed safely to the backend. Operators are never asked to work in a local browser.

A pause button colliding with running purchases

Problem Saving an option pause competed for a lock with supplier re-evaluation and failed with HTTP 500.

Solution The lock switched to shared mode so independent work runs in parallel, while real purchases and pauses still queue on it. Long waits return a 409 conflict before saving. A single order can resume even while the whole option is paused.

Architecture

Sales channels

- **Order collection**
  Coupang · Toss Shopping · resumes from the last successful point
- **Tracking registration**
  Supplier tracking numbers registered on channels · confirmed by official lookup

Purchase execution

- **Job queue**
  One runner per job for a set time · retries · duplicate-order protection
- **Supplier integrations**
  Submission · payment · tracking collection · per-supplier payment modes
- **Server browser**
  Recovers supplier login sessions

Records

- **Step-by-step processing records**
  External lookup results recorded with the next processing step
- **Audit history**
  Existing history kept, only new events appended · held when the outcome is unknown

Order received → Candidate selected → Product prepared → Supplier submission → Supplier payment → Shipment and tracking check → Channel tracking registration → In-transit check → Delivery confirmation

Personal

vortesso · coffee journal app

Period	2025.10 — 2026.03
Team	Solo
Role	Planning Flutter app Spring Boot API Home server deployment CI/CD Monitoring
Stack	Flutter · Dart · Java 17 · Spring Boot 3.5 · Spring Security · JPA · MariaDB · Redis · MinIO · Google Cloud Vision · OpenAI API · Docker · Jenkins · Proxmox
Links	github github.com/vortesso

A coffee journal that fills in entries from the bean details on card photos

An app for recording the taste and beans of each coffee. I built the Flutter app and Spring Boot API.

Features

- Cup records with hot or iced preparation, tasting notes and memos
- Sensory ratings and journal analysis
- Journal calendar and recording streaks
- Google login

Copying card information by hand

Problem Users had to type the bean details printed on coffee cards, such as origin, farm, altitude, process, variety and cup notes.

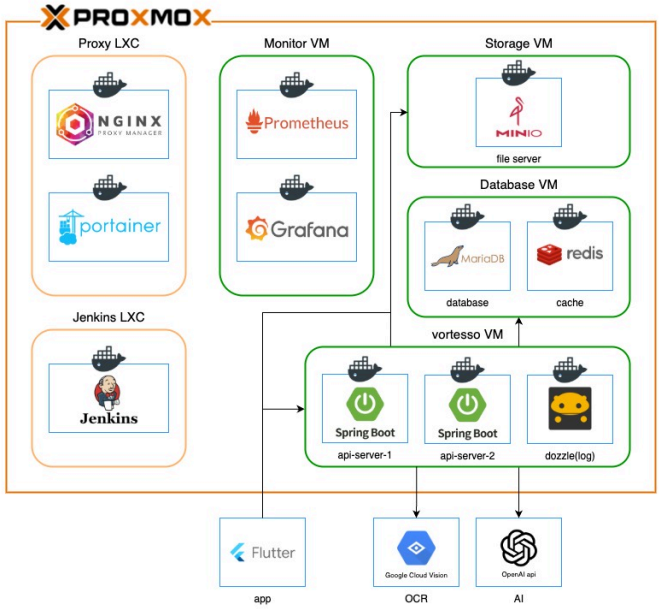
Solution The app uploads the card photo to MinIO, and Google Cloud Vision reads the text. The OpenAI API then turns that text into JSON with one field per bean detail. If the text has nothing to do with coffee, the server returns a not-a-coffee-card error. The photo stays at a temporary path and moves to its permanent path when the user saves the entry.

Running services on one machine

Problem The API, database, file storage, CI and monitoring needed to run on one server at home.

Solution I split the proxy, CI, monitoring, file storage, database and API into 6 VMs and LXC containers on Proxmox. I ran 2 API servers and gathered their container logs in Dozzle. Jenkins reaches the API VM over SSH through the Proxmox server, pulls the new image and starts it. After each deployment it calls a health-check endpoint up to 20 times and sends the outcome as a push notification.

Architecture



Six VMs and containers on a home Proxmox server, with two API servers

Personal

gloticker · macro indicators Chrome extension

Period	2025.03 — 2025.05
Team	Solo
Role	Planning and design FastAPI collection server Spring Boot streaming server Frontend Home server CI/CD Monitoring
Stack	Java 17 · Spring Boot · Python · FastAPI · Redis 7.2 · React · Proxmox · Docker · Nginx Proxy Manager · Jenkins · Prometheus · Grafana
Links	github github.com/gloticker

300+ cumulative users across 7+ countries

I built it because checking market indicators meant visiting 4–5 sites every time. I ran separate FastAPI collection and Spring Boot streaming servers on a home server.

Features

- Live index quotes: Nasdaq, S&P 500, Dow Jones, Russell 2000, 20-year Treasury, VIX, Fear and Greed Index
- Stock quotes: the Magnificent 7 (Apple, Nvidia, Microsoft, Amazon, Google, Meta, Tesla)
- Crypto quotes: Bitcoin, Ethereum, Solana, dominance
- Exchange rates: Dollar Index and the won against the dollar, euro, yuan and yen
- SSE live updates over a Redis message queue
- Market analysis with the ChatGPT API
- 30-day daily candle charts
- Settings for hidden items, theme and language

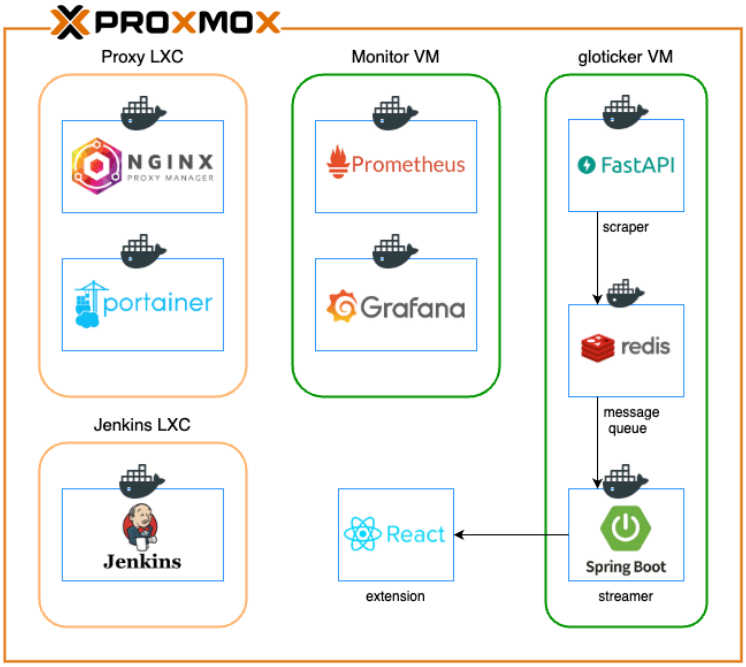
Getting blocked while scraping

Problem	Frequent yfinance calls caused 429 errors, and some servers identified the client as a bot and returned 401 responses. I needed a call rate that kept prices fresh without getting blocked.
Solution	Forum reports suggested about one call per second, so I randomized call intervals between 1 and 1.2 seconds. Each call carried a valid random header so the scraper was not flagged as a bot. Values that change slowly, such as the Fear and Greed Index, dominance and exchange rates, update once a minute.

API requests piling up on every popup

Problem	Opening the extension popup called the per-symbol price and change, chart, SSE subscription and analysis APIs at once. The home server needed protection from the resulting load.
Solution	Redis caches the latest data snapshot and sends it in one response, and the extension stores the latest data and its arrival time in local storage. One scraping round takes 23 seconds on average, so if the last update arrived within 20 seconds, the popup only subscribes to SSE. Charts use 30 days of daily candles, so they load once at market close, and Bucket4j limits each client to 6 requests per 10 seconds.

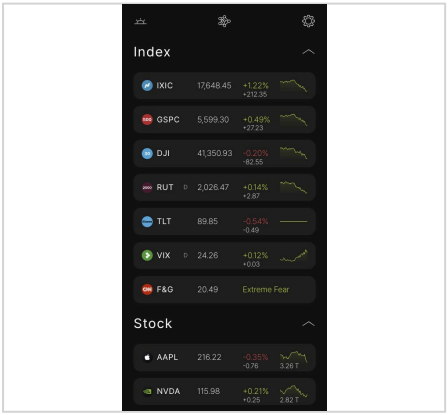
Architecture



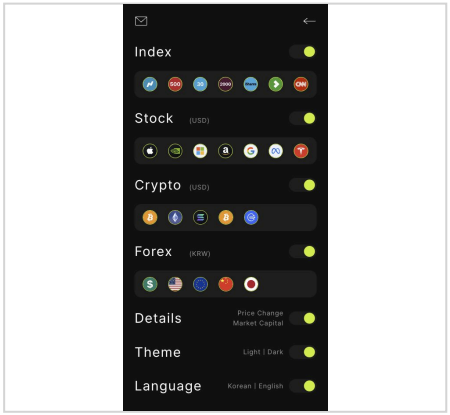
Price collection and streaming servers run on Proxmox.



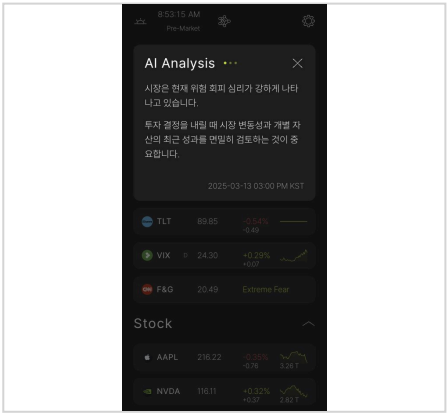
gloticker logo cover



Percentage changes and trend lines for each index on one screen



Choose the stocks and asset categories to follow.



Read market analysis alongside prices.

Personal

underthekey · custom keyboard platform

Period	2024.08 — 2024.11
Team	Solo
Role	Planning and design Switch recording and mixing Public random sentence API MSA core module Frontend Home server CI/CD Monitoring
Stack	Java 17 · Spring Boot 3.3.3 · Spring Data JPA · Querydsl · MariaDB · Redis 7.2 · MinIO · Next.js · Web Audio API · Proxmox · Docker · Nginx Proxy Manager · Jenkins
Links	github github.com/underthekey sentence-api github.com/underthekey/sentence-api

500 users in 2 days

A platform for handling custom keyboard builds in one place, where users practice typing with real keystroke sounds. I recorded and mixed the switch sounds myself.

Features

- 13 switch sources based on Cherry-style samples
- Input tests per keyboard layout
- Imager: sound-stage visualization
- Volume and language settings
- Public API serving about 5,000 Korean proverbs and about 300 English quotes by language and category
- Up to 20 sentences per request

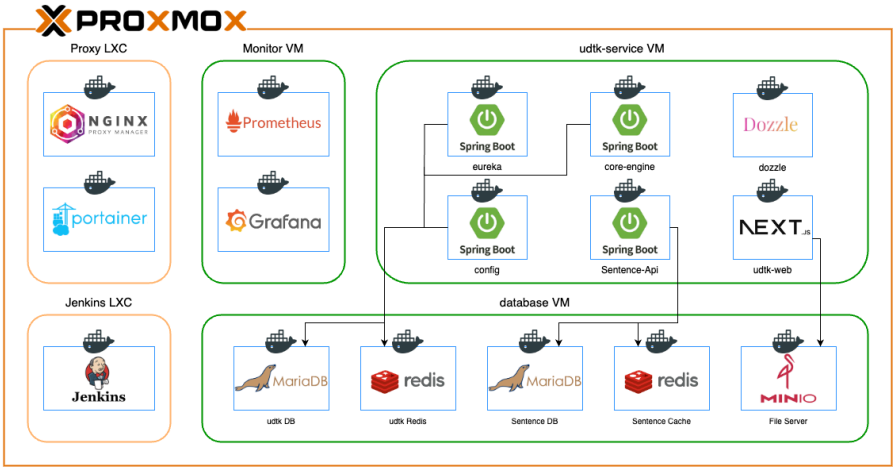
Audio files multiplying per layout

Problem	At first, a DAW template produced a source for each of about 110 key positions, so every switch meant sending 110 mixed files. The number of downloads hurt performance, and I needed a way to automate and shrink this step.
Solution	Moved the mixing from the DAW to the browser with the Web Audio API. One 8 KB source with panning, frequency adjustments and keycap-size filters forms a template that covers every layout. The number of files to download fell from 110 to 1.

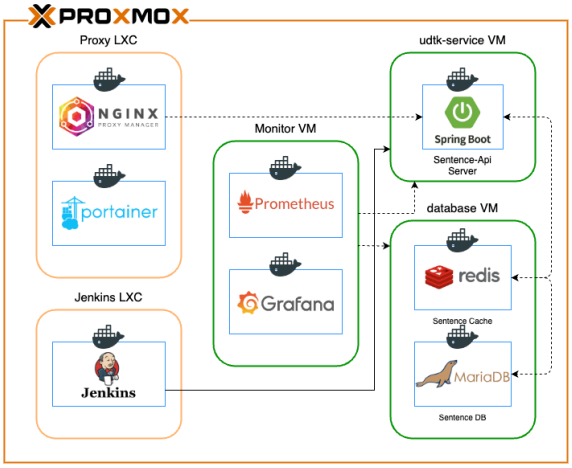
Too many requests to the public API

Problem	The random sentence API is public, so traffic could spike, and it is the core server of the service, so it had to stay up. Counting remote IPs directly would strain concurrency, and the limits needed detailed rules.
Solution	Chose Bucket4j for its accuracy and how well it works with Redis TTLs. Token limits allow 20 requests per 5 minutes, so excess requests do not turn into server load.

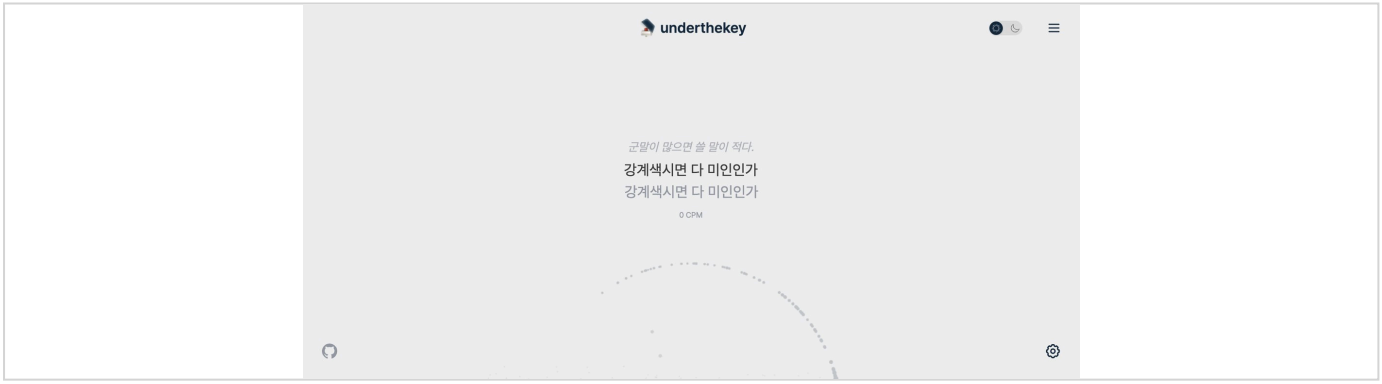
Architecture



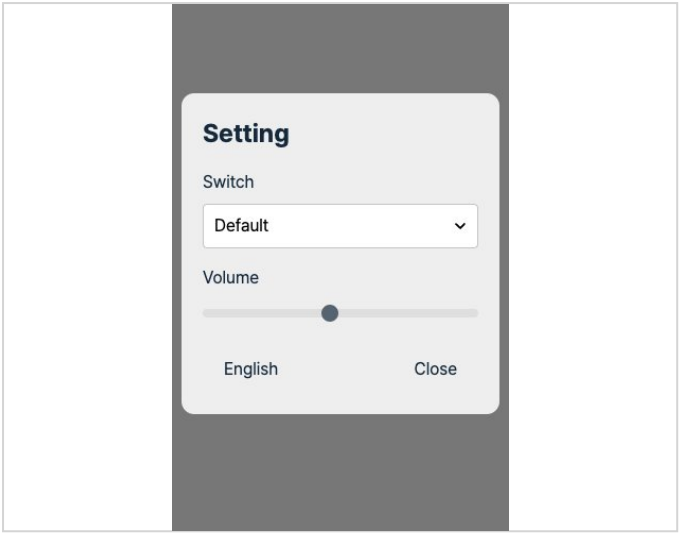
Services, databases, monitoring and CI split across VMs on one Proxmox host



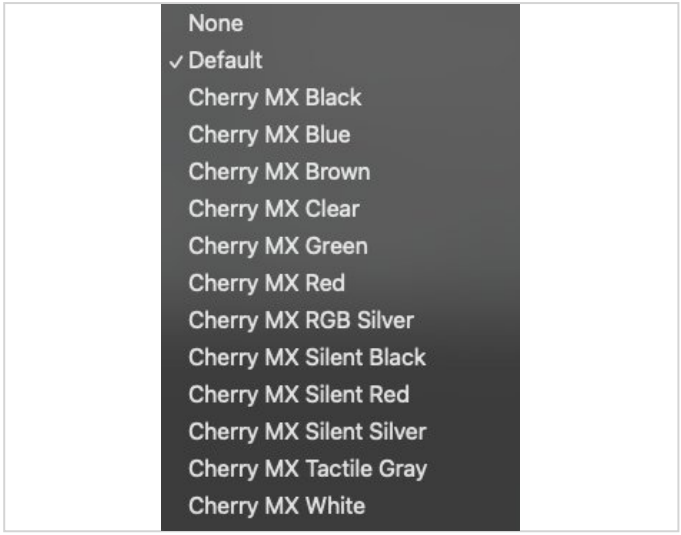
Sentence API server, cache and database split across VMs



underthekey screen with text input and typing effects



Adjust typing volume and the switch type.



Choose a typing sound by switch type.



Particle effects follow the typing.



Pressed keys appear on the virtual keyboard.

SSAFY

seoulpop · location-based history walks

Period	2024.04 — 2024.05
Team	6 people Backend 3, frontend 3
Role	Team lead Backend lead Server and CI/CD Member API Heritage and historic site lookup
Stack	Java 17 · Spring Boot 3.2.5 · Spring Data JPA · Spring Security · Querydsl · MySQL 8.0 · Redis 7.2 · AWS S3 · AWS EC2 · Nginx Proxy Manager · Jenkins · Docker · Uber H3
Links	github github.com/seoulpop/seoulpop-be

Team and backend lead for member and history lookup APIs and infrastructure

seoulpop is a walking guide to Seoul's heritage and historic sites that follows the user's location. I set up the servers and CI/CD alone.

Features

- Seoul history map
- GPS alerts for nearby heritage and historic sites
- Camera AR with real-time content rendering
- Location and route guidance
- History details built from crawled data
- Visit stamp collection

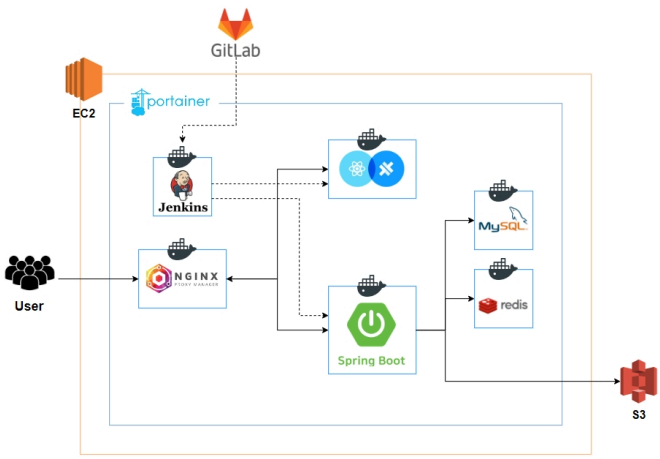
Distance calculations as users move

Problem	Coordinates of nearby heritage sites needed recalculating whenever a user moved. Finding the shortest distance every time grew costly fast, so we needed something better than a simple search.
Solution	The team compared partial search around the user's coordinates, GeoHash and Uber H3. Weighing real-time proximity, accuracy and ease of use, we chose H3. We tested several scales and settled on about 200 m, and a teammate implemented the distance calculation.

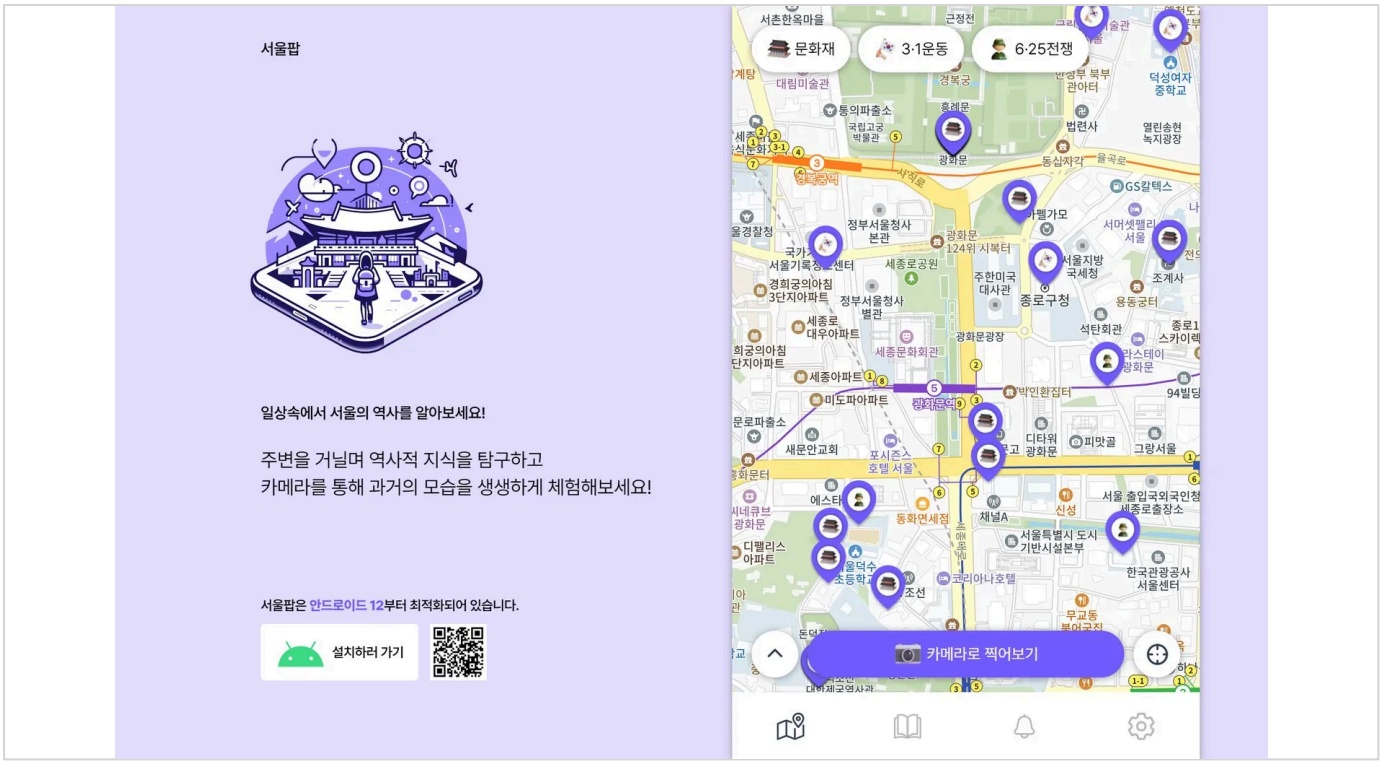
Tracing container logs

Problem	Logs became harder to trace in Portainer as the number of containers grew. Scrolling problems in the Portainer console also made older logs hard to find.
Solution	After looking at several tools, I introduced Dozzle, which is simple to run and shows logs from many containers together. Logs for each transaction could be read on one screen, and the whole team checked logs on the same view.

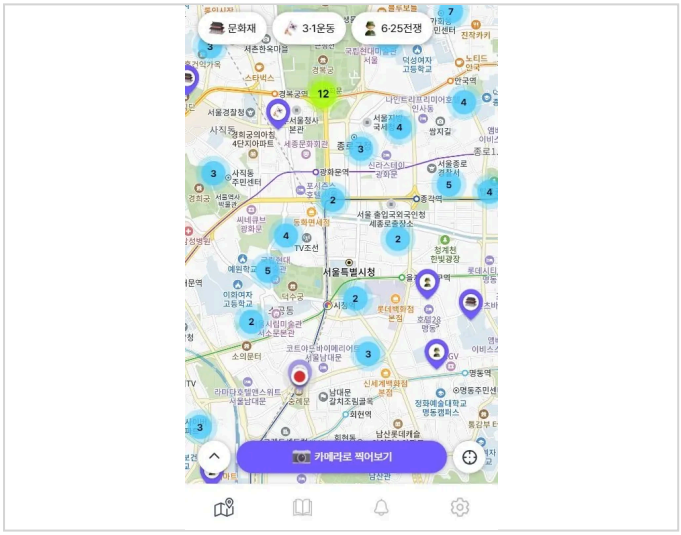
Architecture



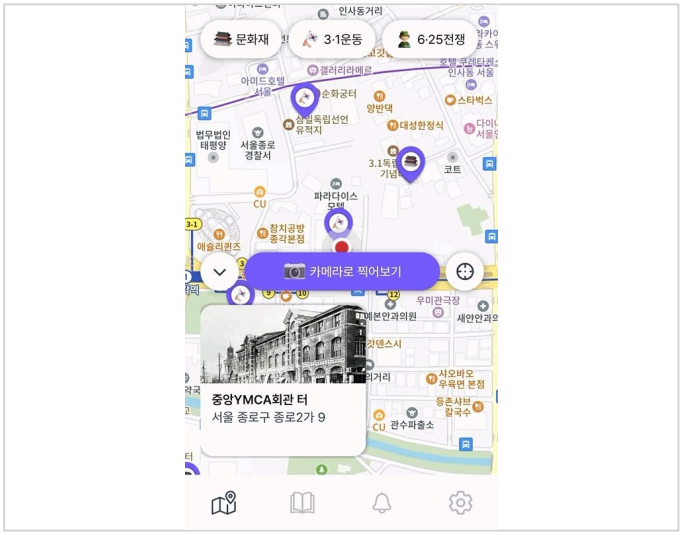
Flow from code push to deployment



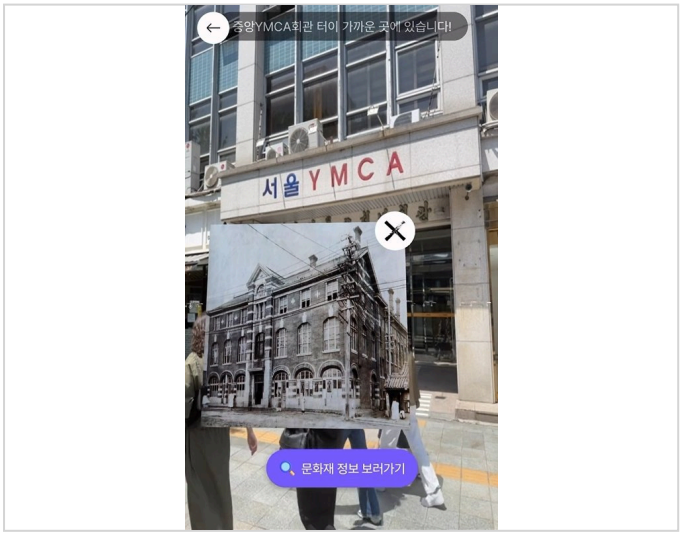
seoulpop cover with an introduction and map



Find nearby historic sites on the map.



Inspect a historic site selected on the map.



Compare a past photograph with the present street.



Read a historic site's summary and description.

SSAFY

lyricit · real-time game built on song lyrics

Period	2024.02 — 2024.04
Team	6 people Backend 2, frontend 2, data 2
Role	Backend lead CI/CD and infrastructure owner Member API WebSocket communication Chat Room creation and game join API Game flow refactoring Game timer scheduling
Stack	Java 17 · Spring Boot 3.2.1 · Spring Data JPA · WebSocket (STOMP) · MySQL 8.0.35 · Redis 7.2 · AWS EC2 · Jenkins · Docker
Links	github github.com/lyricit/lyricit-be

Backend lead for real-time chat, game messaging and infrastructure

In lyricit, players see lyric keywords each round and guess the song in real-time chat. The game uses lyrics from popular Korean songs over the years, crawled and processed in a distributed setup. I handled most of the backend work and set up the servers and CI/CD alone. I configured several deployment scenarios in Jenkins and kept refining usability through repeated QA rounds.

Features

- Live room info updates and chat
- Room creation, game preparation and start, game mode settings
- First correct answer in chat wins the round
- Lyrics data search

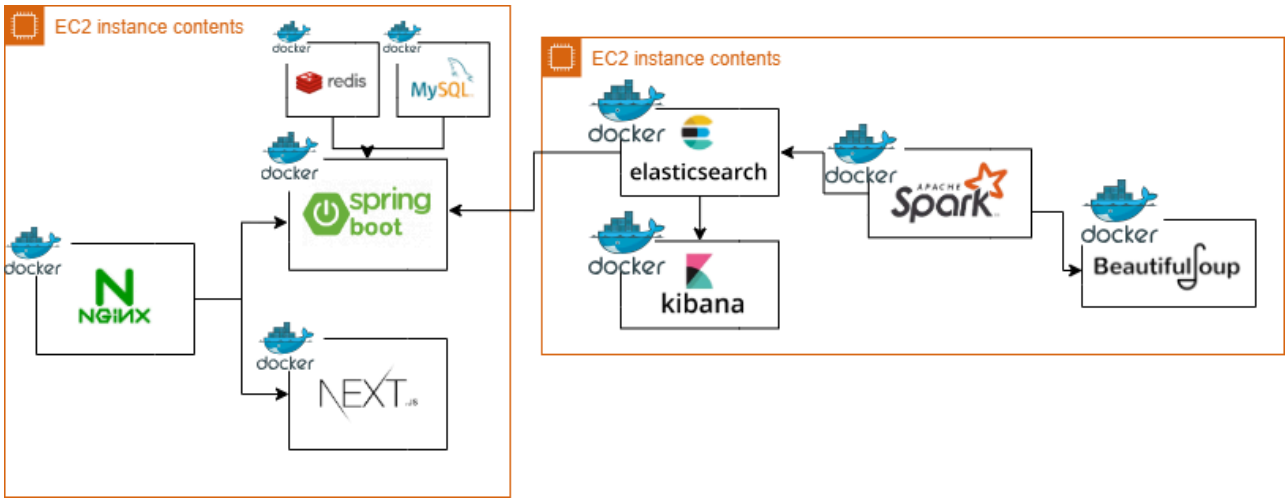
One real-time message channel per page

Problem	The game had many message channels, increasing resource use, channel management and development work. I had to choose between a channel per component and a channel per page, and the choice had to be simple enough to keep development costs down.
Solution	Combined the message channels into one per page. Responses share a common format, with message types distinguishing situations, which made channels simpler to manage. Resource use went down, and new features were added as new message types.

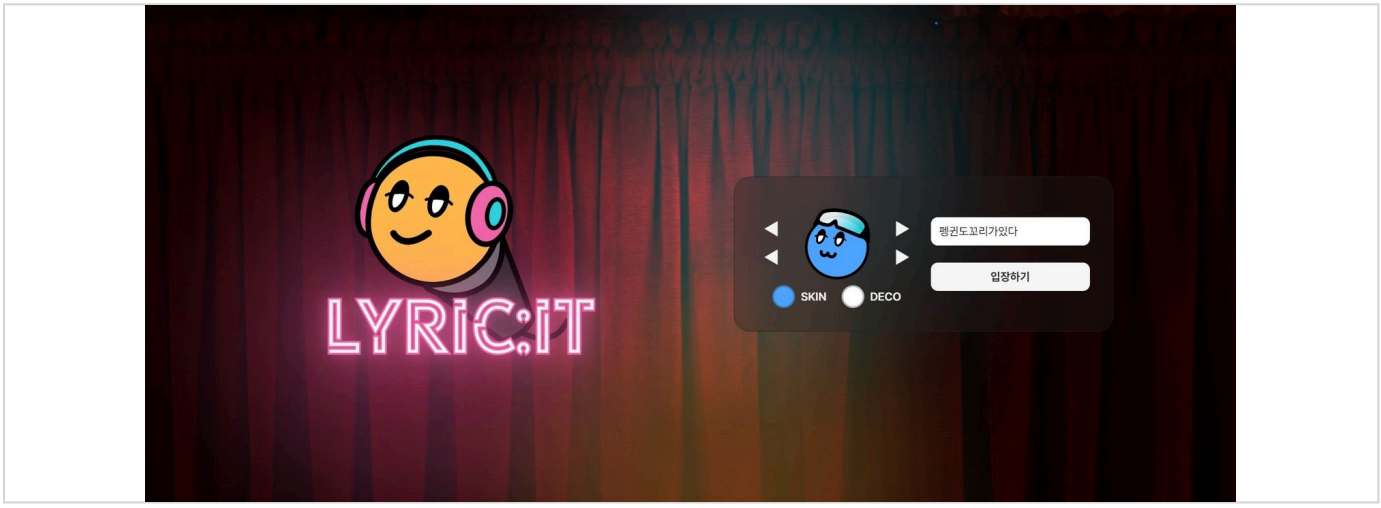
Dropped sessions

Problem	Players were marked offline when their socket session dropped, not when they logged out, so room and game state in Redis had to be updated at the same time. Timeouts also caused disconnections.
Solution	An interceptor checked STOMP commands and handled each connection state, and when a player dropped mid-game, the game and room data were deleted and updated. The channel interceptor passed work to an event listener to avoid circular references. Heartbeat ping and pong frames at fixed intervals stopped timeout disconnections.

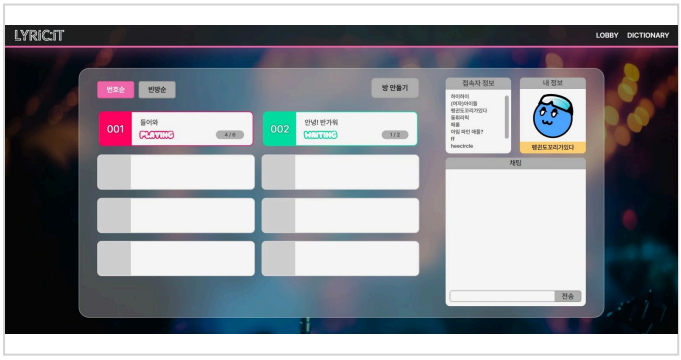
Architecture



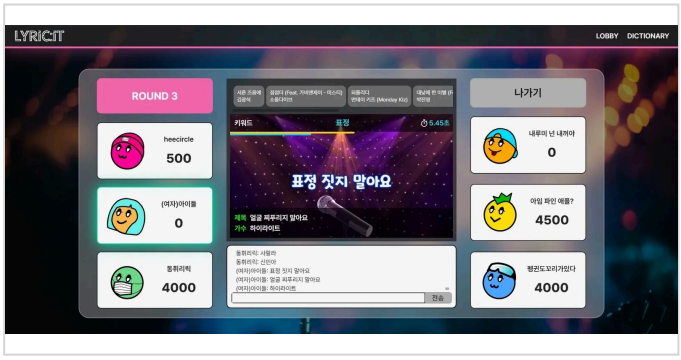
Service servers separated from the lyric search and crawling stack



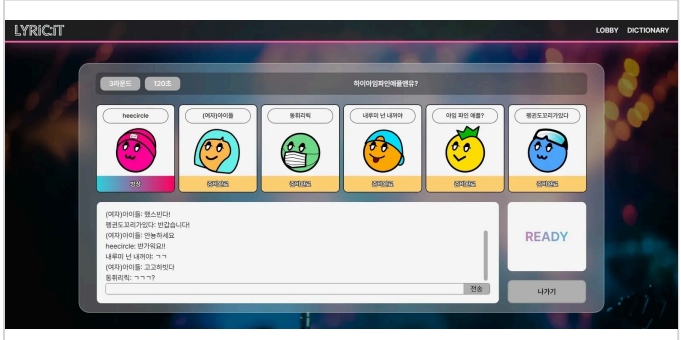
lyricit start screen with the logo and login form



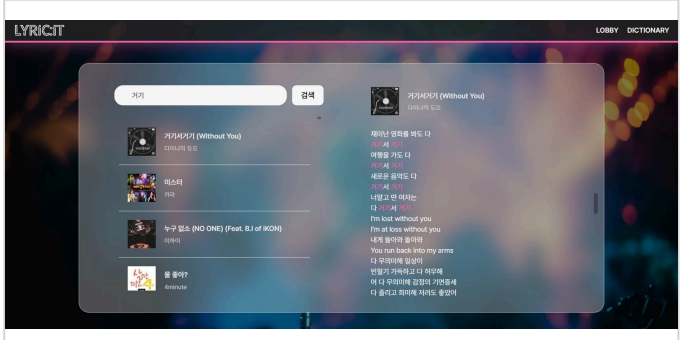
Find a game room to join from the lobby.



Play with lyric keywords and check player rankings.



Check players and game settings in the waiting room.



Search for songs and look up their lyrics.

SSAFY

jansorry · holiday nagging receipts

Period	2024.01 — 2024.02
Team	6 people Backend 3, frontend 3
Role	Team lead Backend lead Infrastructure and tech owner Nagging, like, follow and receipt API Batch
Stack	Java 17 · Spring Boot 3.2.1 · Spring Data JPA · Spring Security · Spring Batch · Querydsl · MySQL 8.0.35 · Redis 7.2 · AWS EC2 · GitLab Runner · Docker
Links	github github.com/jansorry/jansorry-be

About 200 sign-ups over Lunar New Year

The project started from the stress young people feel from holiday nagging and the gap between generations. Users share nagging and how they handled it, then turn those stories into receipts and statistics. A preliminary survey with 300+ participants collected holiday stress data, and the project won first place among SSAFY common projects and the Project Excellence Award. I handled more than half of the backend work and set up the servers and CI/CD alone.

Features

- Feed of nagging and responses
- Likes and follows
- Receipt printing and sharing
- Account page
- Statistics visualization
- Deployment to EC2 through GitLab Runner and Docker Hub
- Next.js, Spring Boot, MySQL and Redis containers behind Nginx

Two-way follow relations in Redis

Problem	Follow relationships had to live in a non-relational store. Unlike one-way likes, follows are many-to-many, so storing both directions uses more space while storing one direction slows lookups.
Solution	I decided wildcard scans should be avoided wherever possible and stored follows in both directions. Keys carry following and follower prefixes, so each user's following and follower lists are read directly. Follower and following lists became much faster to look up.

Full scans in the batch sync

Problem	The daily batch moving Redis data to MySQL scanned the entire dataset. I needed a way to move only the data that had changed.
Solution	Changed keys go into a Redis sorted set, scored by their update time. The batch writes only changes since the previous run to MySQL, then clears the set so it holds only new changes. Rows left empty after likes or follows were canceled were also removed from the table to free space.

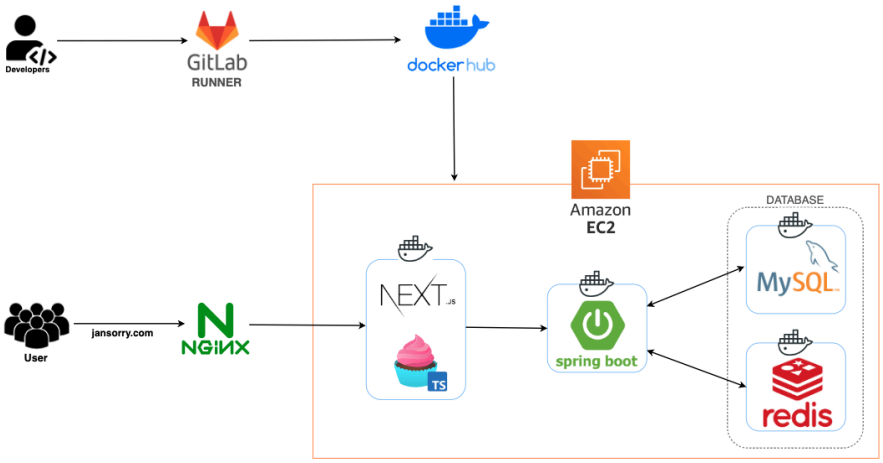
Top 5 receipts by price

Problem	The final statistics needed the 5 highest-priced receipts. Searching every receipt of every user at once was costly.
Solution	Reused the sorted-set approach from the batch sync. Saving a receipt adds it to a top-5 set scored by price, and the receipt pushed to sixth place is removed right away. This spreads the search load across saves instead of one moment, so no query scans all receipts at once.

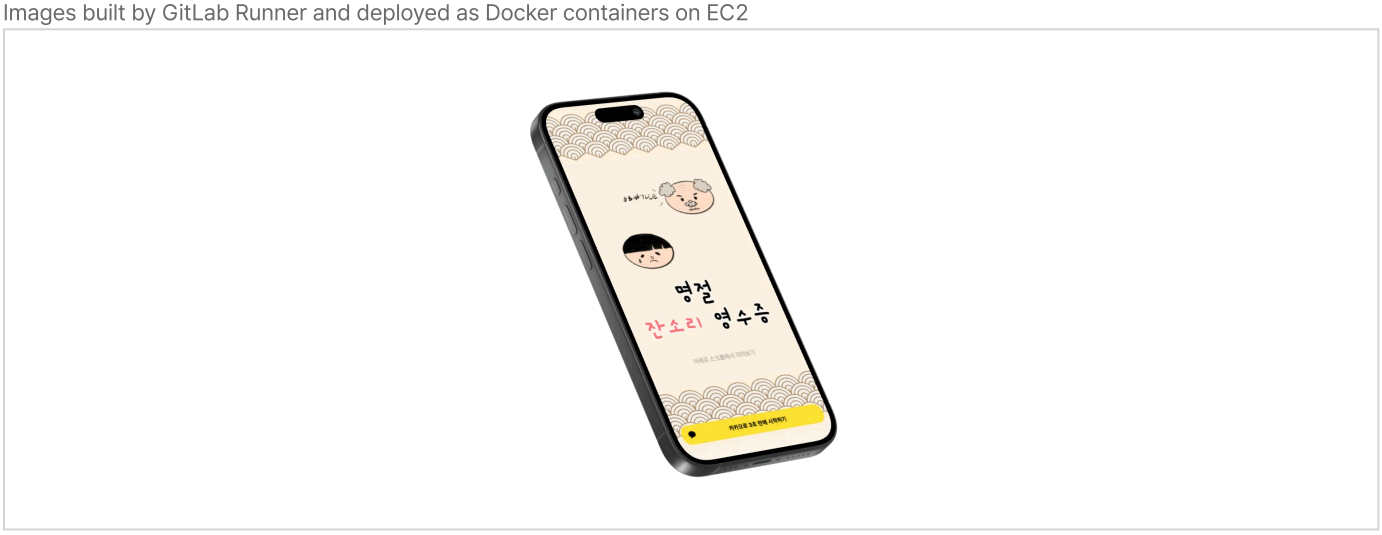
Deploying frontend and backend from one repository

Problem	The project could use only one repository, so the frontend and backend code shared it and deployment had to be automated from there. Jenkins had a high setup cost and took up a lot of server resources.
Solution	Chose GitLab Runner, which has fewer plugins but sets up a pipeline with little overhead. The pipeline has 4 stages: create the environment file, build, build the Docker image and push it to Docker Hub, then run it on the server. Secrets such as database credentials live in GitLab CI/CD variables and become a .env file at deploy time.

Architecture



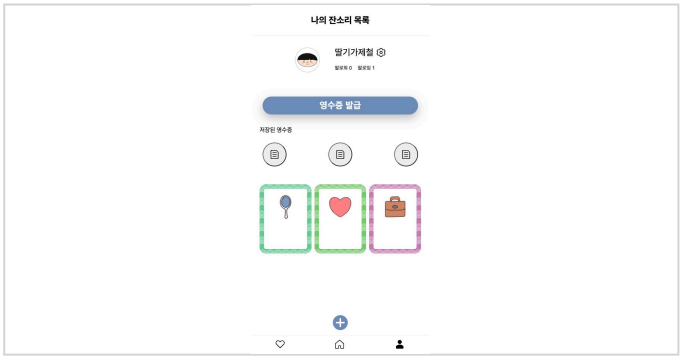
Images built by GitLab Runner and deployed as Docker containers on EC2



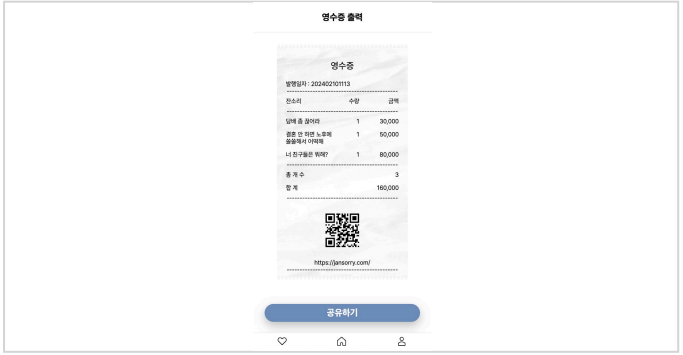
Smartphone mockup showing the jansorry start screen



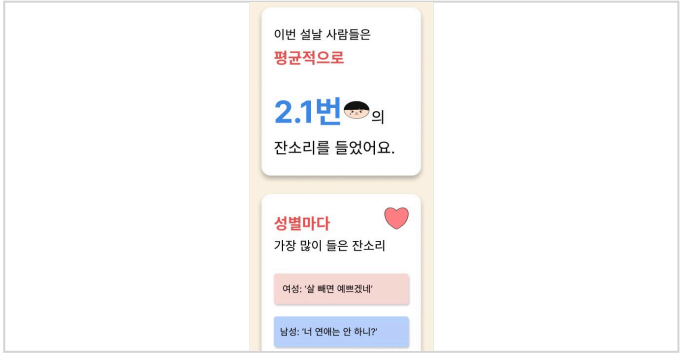
Filter nagging stories by hashtag.



Turn saved nagging stories into a receipt.



Nagging lines issued as a receipt



Review nagging counts and statistics by gender.

SSAFY

newjibs · real estate investment simulator

Period	2023.11
Team	2 people Backend 1, frontend 1
Role	Team lead Backend (solo) Server and CI/CD Simulated investing, ranking, members and notices News crawler scheduling
Stack	Java 8 · Spring Boot 2.7.16 · Spring Data JPA · Spring Security · MariaDB · Redis · AWS S3 · AWS EC2 · Cloudtype · Docker · GitHub Actions
Links	github github.com/NewJibs/newjibs-be

Sales records reduced from over 4 million to about 60,000

newjibs lets users invest a virtual budget in real estate and check returns two years later. It started from the steep rise in housing prices during COVID-19. I built the backend, servers and CI/CD alone.

Features

- Kakao Map cluster rendering by latitude and longitude
- Real estate information lookup
- Ranking
- Notices and real estate news
- JWT-based member authentication
- Vue.js frontend deployed on Netlify
- Spring Boot, MariaDB and Redis containers shipped to Cloudtype and EC2 by GitHub Actions

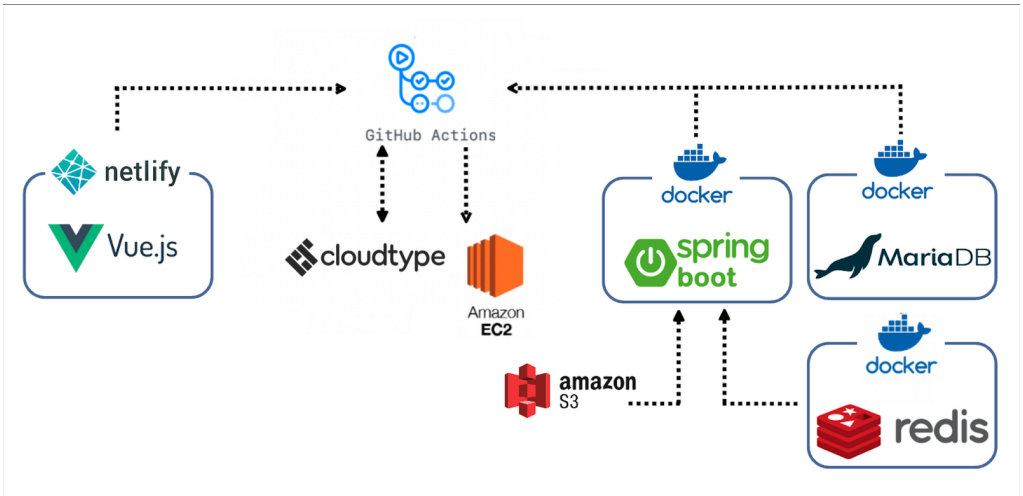
Shrinking the sales data

Problem	Sale records since 2016 exceeded 4 million rows and slowed rendering, and simulated investing did not need all of them. Prices rose sharply after COVID-19 began in 2020, so I used that as the basis for cutting the data.
Solution	Matched 2020 and 2022 sales one-to-one by floor and unit size, keeping price changes for the same homes. I judged that sales of the same home would make the investment results more trustworthy. Sales records fell from more than 4 million to about 60,000, and rendering became faster.

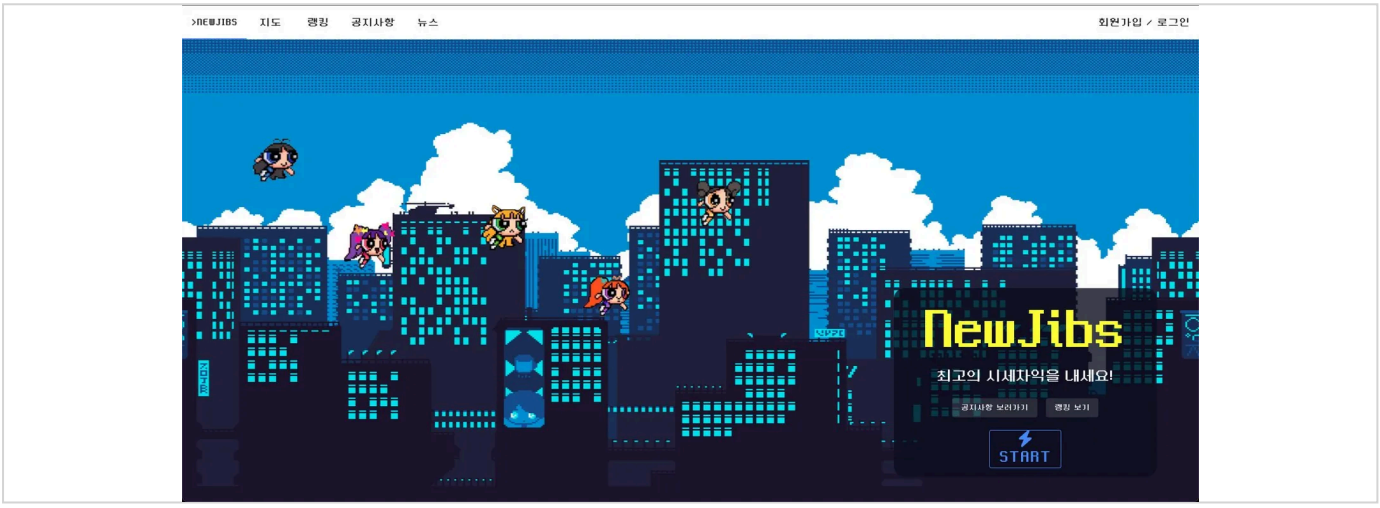
News crawling getting blocked

Problem	Collecting real estate news with jsoup on the hour got the server flagged as a bot and its IP blocked every time. I wanted to fix this inside the crawler rather than introduce Selenium.
Solution	Found that the on-the-hour timing and the request headers made the crawler look like a bot. The crawler now changes its request headers and adds a random delay before each call. With a less obvious call pattern, it stopped getting blocked.

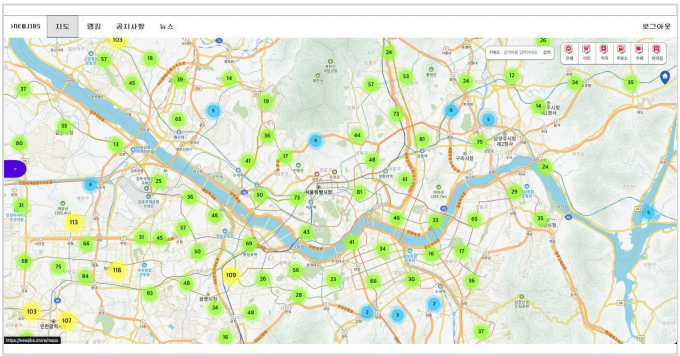
Architecture



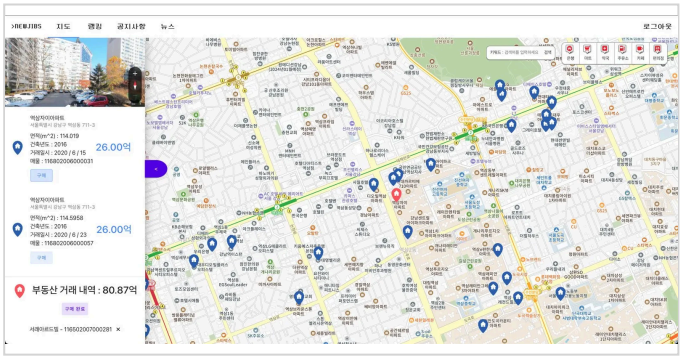
Deployment to Netlify, Cloudtype and EC2 through GitHub Actions



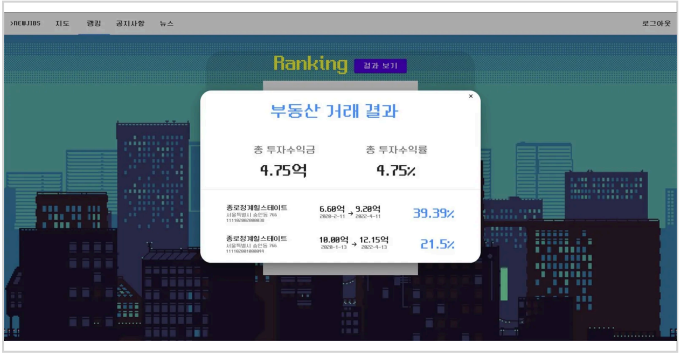
newjibs start screen with a city illustration



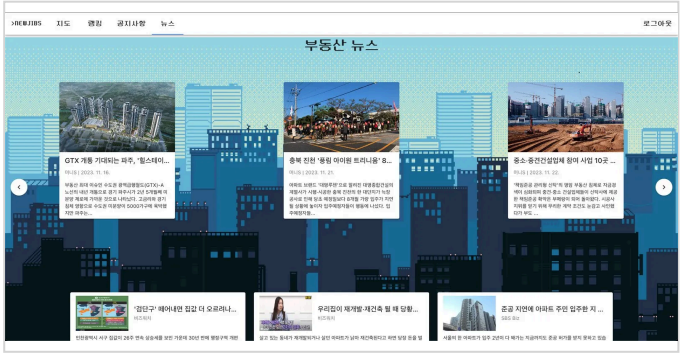
Listings clustered on the map



Review the sales history of a property listing selected on the map.



Check returns from a simulated investment.



Browse real estate news as a list of cards.

SSAFY

triplet · travel expense diary

Period	2023.09
Team	4 people Backend 3, frontend 1
Role	Backend API (members, trips, daily, image upload) Server and CI/CD
Stack	Java 11 · Spring Boot 2.7.10 · Spring Data JPA · MySQL 8.0 · AWS S3 · AWS EC2 · Docker · GitHub Actions
Links	github github.com/Triplet-Shinhan/Triplet

Selected for the Shinhan Bank hackathon from 8 applicants per place

triplet records foreign-currency cash and card spending together and automatically converts it to won at the current exchange rate. We built it at the Shinhan Bank hackathon during SSAFY. It was my first project with JPA and my first deployment. I handled half of the server and CI/CD setup.

Features

- Trip creation with budget and cash on hand
- Currency exchange requests
- Calendar-style expense diary
- Image upload

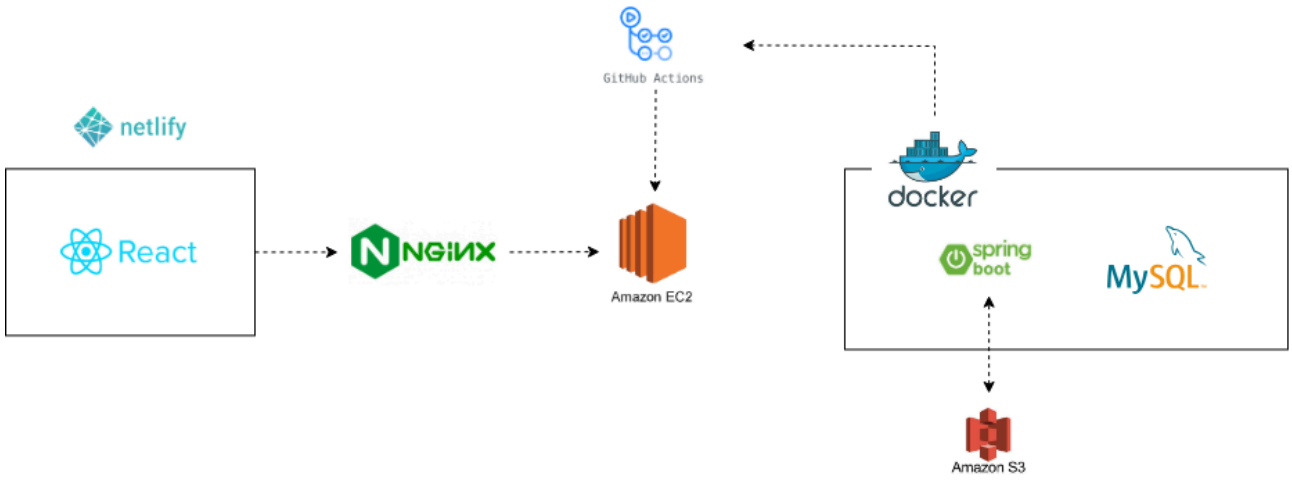
Working with a dummy bank API

Problem	Security and technical issues kept a real bank API out of reach, so the hackathon ran on a dummy-data API. We had to work out the transaction logic and exchange rate information ourselves.
Solution	Built additional APIs and dummy data for spending, exchange-rate and other cases. We used that data to reproduce each user scenario while building features.

Running out of instance resources

Problem	To run at no cost, we put all containers on one AWS Free Tier instance. Usage kept climbing until the instance shut down.
Solution	I considered upgrading to a paid size but configured memory swap on the 1 GB RAM instance instead. As it was my first deployment, finding the cause took a long time.

Architecture



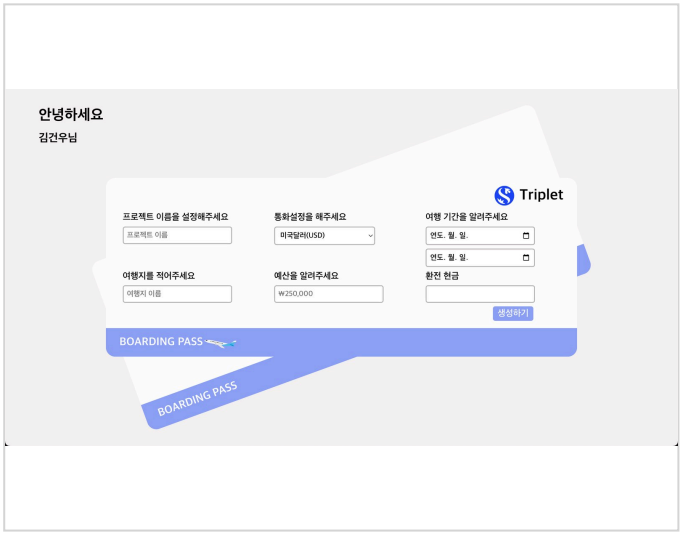
Deployment to EC2 with GitHub Actions



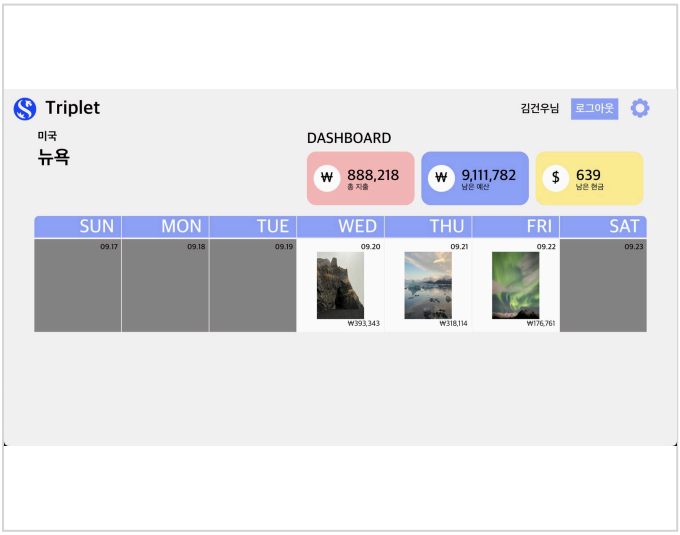
triplet start screen with an airplane illustration and login form



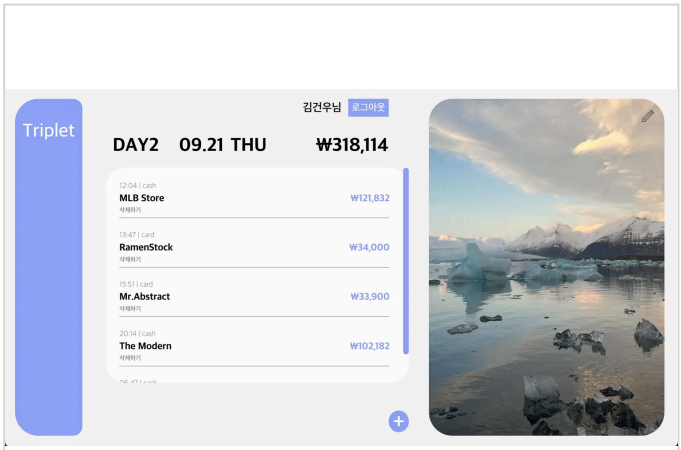
Manage trip projects in a list.



Create a trip project with a destination and dates.



Review daily spending on a calendar.



View expenses and trip photos together by date.

Tools and open source

herdr-cockpit · Herdr layouts and mods for watching orch

Period	2026.11 — Present
Team	Solo
Role	Design Herdr mod development Go helper
Stack	Go · JavaScript · Swift · Herdr.app · gopsutil
Links	github github.com/takealook97/herdr-cockpit

Sessions and usage across accounts and machines on one screen

herdr-cockpit is the set of Herdr layouts and mods I built so that orch runs are easy to follow. It shows the orchestrating main session, worker agent sessions spread across accounts and machines, and remaining usage allowances by provider on one screen. It also works with plain CLI sessions without orch.

Features

- Top bar with remaining usage allowances for Codex, Claude and Gemini, reset times, CPU and RAM
- Machine, space, tab and pane tree, with an agent list that counts workers per main session
- A grid of this computer's main-session terminals that takes input directly (hub)
- Routing spoken or typed instructions to the right main session, with Korean voice input (nexus)
- Hiding and restoring tabs of disconnected remote machines
- A Go helper on each machine that collects usage and pane data

Different logins and rate limits per provider

Problem Reading Codex, Claude and Gemini usage meant handling a different login and rate limit for each provider.

Solution Authentication, requests, caching and retry intervals live in a Go helper on each machine. The screen only receives JSON and never handles credentials.

Late responses while switching machines

Problem A late response from the previous machine could overwrite usage for the machine on screen after a switch.

Solution Responses are kept per machine, and only the selected machine's response updates the screen. Switching machines quickly no longer mixes usage from another machine into the view.

Instructions that could reach the wrong session

Problem A spoken or short typed instruction could be ambiguous about which main session it was for.

Solution A model picks the target from this computer's main sessions only. Below 0.6 confidence it asks with candidates, and on failure or after 20 seconds it sends nothing.

Overlapping layout changes

Problem In the view that gathers several main sessions into one grid, session list and layout changes could overlap asynchronously. A change that finished late could bring back a view that was already gone.

Solution Layout writes run one at a time, and the open view is kept as one shared state. When the session list changes, the view waits 500 ms and applies the change once. A late write no longer selects an outdated view.

Tools and open source

wiki-island · vendor-neutral AI memory

Period	2026.11 — Present
Team	Solo
Role	Design Rust development
Stack	Rust · SQLite FTS5 · MCP · Tokio · Git · Markdown
Links	github github.com/takealook97/wiki-island

Connects to 10 developer tools through 15 MCP tools without calling a model

wiki-island is an open-source project I built so that many AI tools share one memory that no single vendor owns, instead of each tool keeping its own. The memory is a Markdown folder on the user's own computer. Saving, searching and reading use no model and no network.

Features

- Search, read, write, edit, archive and restore through 15 MCP tools
- A single wi setup run connects 10 AI apps, including Claude Code, Codex and Cursor
- SQLite FTS5 search that finds two-character Korean words without spaces
- Git history with restore to earlier versions
- Works alongside Obsidian folders and imports PDF, Word, PowerPoint, CSV and images
- A background service that picks up edits, repairs the index and tidies notes

An index that can break

Problem Treating the search index as the source would lose memories when the index is damaged or synced between devices.

Solution Markdown files are the source, and the SQLite index is a per-device cache. When the index is missing or damaged, search scans the files directly and the index is rebuilt. Deleting the index loses no memories.

Edits that overlapped or stopped mid-save

Problem Several AI tools could edit the same note at once, or a process could stop in the middle of a save. That could overwrite newer content or lose a saved change.

Solution Each write checks the checksum the caller expects and rejects stale writes. It writes a temp file to disk, records an intent, swaps the file in and records completion. Interrupted work is recovered from these records. On platforms with atomic swaps, competing edits are kept as conflict copies.

Two-character Korean search

Problem Search that splits words on spaces could not find short Korean words such as 결제 or 환불 inside a sentence.

Solution I kept the default tokenizer and added an index column that splits text in scripts without spaces into two-character pieces. Multi-word queries show results matching every word first, then fill the remaining slots with ranked partial matches. Two-character queries such as 결제 and 환불, and mixed Korean and English queries, find the intended notes.

Git history in a synced folder

Problem Sync services such as iCloud Drive send files one by one, so a Git history inside the folder can arrive partly or out of order.

Solution When the notes folder is synced, Git history is kept outside it in a separate folder on each device. The notes folder contains only Markdown.

Tools and open source

orch · multi-agent orchestrator

Period	2026.09 — Present
Team	Solo
Role	Design Go CLI development
Stack	Go · YAML · Herdr · Claude Code · Codex · Antigravity
Links	github github.com/takealook97/orch

Work finishes only after a model other than the author verifies it

I built orch to keep one main session for judgment instead of doing everything inside a single agent session. The main session plans, assigns work and judges results. Worker agents execute from contract files, and a model from a different family than the author reviews each result.

Features

- Work assignment across 7 model keys from the Codex, Claude and Gemini families
- Contract files with scope and result format, plus result files and status records
- One-off lookups and edits without opening a pane, and test runs that return only a failure summary
- Independent review by a model family other than the author's
- Sequential and parallel runs by dependency, resumable after interruption
- Herdr pane layout and cleanup that keeps the user's focus where it is

Roles that instructions alone did not hold

Problem With rules only in instruction files, commands that used their own delegation skipped them. Worker agents read the same instructions and could delegate again.

Solution I split the rules into four layers: shared instructions, a Go executor, hooks that block bypasses and YAML procedures per task type. Workers start marked as workers and cannot delegate further. The executor applies model names and cleanup steps instead of the model restating them every time.

Focus pulled away mid-task

Problem Operating worker panes moved the user's view to those panes. Server logs recorded 55 focus moves on unzoom requests and 78 when sending input.

Solution orch now unzooms only panes that are actually zoomed, and sends prompts as one line of text followed by a separate Enter key. After the fix, measurement showed 0 focus moves caused by orch requests.

Parallel runs that were slower for small tasks

Problem In an evaluation that split small implementation tasks across 3 panes, parallel runs averaged 84 seconds against 49 seconds for a single worker.

Solution Short one-off tasks now go through ask and do without a pane, and tests run without a model and return only a failure summary. The delegation rule changed from elapsed time to the main session's token use.

Tools and open source

hermes-agent · open-source contribution to Hermes Agent

Period	2026.08 — Present
Team	Contribution
Role	Bug fix Tests
Stack	Python · ripgrep · macOS
Links	hermes-agent github.com/NousResearch/hermes-agent/commit/5fd6811dfe6fb18c81a510da354bf5356eb2449b

Fix that stops macOS privacy prompts during home-folder searches

Hermes Agent is an open-source AI agent from Nous Research. When the agent searched a broad path such as the home folder, macOS privacy prompts appeared even in unattended runs. I contributed a fix that leaves protected folders out before the search starts.

Features

- Seven protected folders such as Desktop, Documents and Downloads skipped when searching home or /Users
- The same exclusion for file-name and content search across ripgrep, grep and find
- Searches that name a protected folder directly still run
- A warning that lists skipped folders and how to search them directly
- No change outside macOS

Prompts before results are filtered

Problem	Filtering protected folders out of the results was too late, because entering a folder already triggered the permission request.
Solution	Protected folders are left out before traversal starts, with ripgrep exclude patterns and find -prune. The original PR reported 61 related tests passing along with results from broad searches on a real Mac.

Results that shrank silently

Problem	Skipping folders meant users could not tell that some results were missing.
Solution	A warning names the skipped folders and how to search them directly, and searches that target a protected folder are not filtered. Users know what was left out and can search a protected folder by naming it directly.

Same-named folders and remote environments

Problem	The first version excluded folders by name, so an ordinary subfolder with the same name could be skipped too. The macOS rule could also apply in remote environments.
Solution	A follow-up fix in the merged PR blocks traversal by path instead of by name. It also detects the runtime environment and skips the exclusion on remote machines. Ordinary same-named folders and remote searches no longer lose results for no reason.

Tools and open source

vat · multi-repository workspace CLI

Period	2026.08 — Present
Team	Solo
Role	Design Go CLI development
Stack	Go · YAML · Git
Links	github github.com/takealook97/vat

Records changes and decisions with sources so the next task picks them up

I built vat to develop many repositories at once from one parent folder. Git records each repository, and vat records what happened across them in one place. It also keeps the decisions and records that agents read while developing, each with its source and the date it was last checked.

Features

- Repository list and per-repository edit boundaries declared in one vat.yaml
- Batch sync that leaves local work alone and only fast-forwards when safe
- Return points, check results and the commits verified together across repositories
- Decision records with source commit and check date, and stale records taken out of citation
- Claude Code and Codex agent instructions generated from one role definition, with drift checks
- Configuration and environment checks with lint and doctor

Which commits were verified together

Problem	When an API change spanned several repositories, each git log could not tell which commits had been verified together. Nothing recorded where to go back if something broke.
Solution	vat records a return commit per repository when a change starts and links check results to the exact commits they ran on. Git ancestry shows whether those commits reached the release branch. The verified combination and return points are found in a single change record.

Batch pulls that reported success

Problem	Looping git pull over repositories reported success even when one repository had stashed local work.
Solution	vat checks each repository's state before syncing and only runs safe fast-forwards. Local changes, branch drift, remote mismatch and divergence are reported per repository.

Facts nobody had rechecked

Problem	A document's description of the current state kept being quoted as fact by people and agents long after anyone checked it.
Solution	Each record keeps its source repository, commit and check date. Records not rechecked within their window drop to a state that cannot be cited as evidence.

Tools and open source

offon · open-source Slack app for attendance and leave

Period	2026.04 — 2026.08
Team	Solo
Role	Design Full-stack development
Stack	Next.js 16 · React 19 · TypeScript · Tailwind CSS 4 · shadcn/ui · PostgreSQL · Prisma · Slack Web API · Vercel Cron
Links	github github.com/takealook97/offon

Split off from the company work management system, MIT-licensed and self-hosted

When I joined Lukuku, attendance was not being tracked properly, so I built a system for it after work hours. That system later grew into the company-wide work management system, now with the company ontology attached. I split off only the attendance and leave features into offon, an open-source Slack app.

Features

- Shifts past midnight, several sessions a day, and automatic work, break and overtime totals
- 60-minute meal breaks that set their end time at the start and send a return reminder
- Full-day and half-day leave excluding weekends and holidays, with remaining days tracked
- Attendance correction requests from the calendar, with times recalculated on approval
- One admin inbox for leave and correction approvals
- Slack one-time-code login, 5 slash commands and reminders for missed clock-ins and clock-outs

Clock-ins arriving at the same moment

Problem A Slack retry or a double click could send two clock-ins at almost the same time, and both passed the pre-check, leaving two open sessions.

Solution A PostgreSQL partial unique index allows one open session per attendance record. The second request gets the same already-working response as the pre-check.

Records made after a correction request

Problem If someone clocked out or took another break after requesting a correction, approval could overwrite those records.

Solution Approval compares the value at request time, the requested value and the current value. Fields the request did not touch keep later records, and a field changed differently on both sides is treated as a conflict.

Time zones on the server and locally

Problem Handling dates as-is on a UTC server and a Korean-time laptop could add the 9-hour offset twice or pick the wrong weekday.

Solution Times are read in Korean time and stored in UTC. Displaying times in Korean time and finding the weekday from a date are separate functions. The same time and weekday appear regardless of where the code runs.

Slack retries

Problem Slack resends a command when the reply is slow. Processing it again could create the same record and notification twice.

Solution The handler first checks the retry marker on the request and, for a retry, replies 200 at once without processing it. A slow reply no longer sends the same notification twice.

Tools and open source

sshift · SSH CLI with jump-server connections and password storage

Period	2025.07 — 2025.08
Team	Solo
Role	Design Go CLI development Homebrew distribution
Stack	Go · OpenSSH · sshpass · Homebrew
Links	github github.com/takealook97/sshift

Jump-server SSH to 50+ servers with one command

At Biztalk I connected to more than 50 servers over SSH for patch work. Going through the jump server every time was tedious, so I put jump-server connections and encrypted password storage into one CLI.

Features

- Interactive server picker, direct connection by ID or name, and remote commands
- Add, edit, delete and list servers and jump relations
- Password and key authentication combinations for jump and target servers
- Encrypted password storage with an optional custom key
- JSON export and import of servers and jump relations
- Install through Homebrew, an install script or release binaries

Jump relations that loop

Problem Adding jump relations freely could create a loop where A goes through B and B goes back through A.

Solution Before adding a relation, a depth-first search checks for a path from the target back to the source, and the relation is rejected if one exists. Saved jump relations never contain a loop.

A changed format for jump relations

Problem After jump relations moved to a graph format, server lists made by earlier versions might not load.

Solution If the graph format cannot be read, sshift reads the old relation array and converts it into a graph. Users do not have to rebuild their server lists.

Different authentication per server

Problem At Biztalk, patching more than 50 servers meant going through a jump server for every connection. The jump server and the target often used different authentication, such as a password on one and a key on the other.

Solution sshift builds the command from OpenSSH ProxyJump, ProxyCommand and sshpass depending on the password and key combination.

Passwords stored in plain text

Problem Storing passwords as-is in the server list exposed them to anyone who opened the file.

Solution Passwords are encrypted with AES-CTR and a random IV, and the config files are readable only by their owner. Opening the server list file shows passwords only as ciphertext.