BACK-END ENGINEER · WEB

Not done at building — a back-end developer who owns deployment and secret management, Jihwang Kim

Back-end at the core, owning design, deployment, and operations to keep services running reliably.
Experienced across B2B platforms, B2C e-commerce, and finance.

6+ years of hands-on experience **3** domains — manufacturing · commerce · finance

2022 — **Currently at LS ELECTRIC** · Web platform service development

2018 B.S. in Information Systems, Chung-Ang University

2016 Engineer Information Processing certified

2010 Graduated Deoksoo High School (Digital Contents major)

2007 Craftsman Information Processing · Computerized Accounting Lv.3 · Word Processor Lv.1·2

I believe a developer is **someone who solves problems and communicates through software, creating and expanding sustainable business.**

So instead of reasons it can't be done, I work as a developer who builds the standards and procedures that make it happen

- **Owning operations** — not done at shipping: deployment, monitoring and maintenance included.
- **Full-stack practice** — moving between Java/Spring & Node.js back ends and Vue/React front ends.
- **Improvement-driven** — modernized legacy in stages: Java 11 → 21 → 25, MySQL 8.0 → 8.4 and CRA → Vite.
- **Communication** — confirming business context across clients, operators and developers, aligning in language each side understands.

01 — Career

Career

Three domains: manufacturing B2B platforms, group e-commerce, and finance interfaces.

LS ELECTRIC

Global B2B manufacturer in power and automation

2022.04 — Present · Automation System, Business Development Team

Solution Square — sales support & customer-facing web services

Modernized the BFF, SAML SSO, data interfaces, search, security, and delivery workflows for a global product and technical portal.

Tech Square — B2B smart manufacturing web platform

Full-stack development and operation across company matching, mentoring, used equipment, data ingestion, dashboards, and notifications.

Node.js · Vue 3 · Java · Spring Boot · React 18 · AWS · Azure · MSA/BFF · CI/CD

Kyowon Creative

IT service development and operations affiliate of Kyowon Group

2020.01 — 2021.02 · IT Service Development Team 3

K-Members rental-car service — group membership shopping mall

Developed and operated the K-Members mall, including rental-car consultation, its back office, affiliate sales reporting, and SMS authentication.

Spring · MyBatis · Oracle · JSP · jQuery

Webcash CNS

Webcash Group affiliate for corporate cash-management solutions

2019.01 — 2020.01 · KEB Hana Branch

Corporate cash-management CMS support · ERP DB integration

Supported corporate cash-management CMS operations and handled MS-SQL-based financial data integration between CMS and client ERP databases.

MS-SQL · Finance domain

Projects

A selection of platforms and SI work I developed and operated hands-on.

Solution Square (Global)

PROFESSIONAL PROJECT · 2025 – PRESENT

LS ELECTRIC's global product and engineering portal. Owned business logic, authentication, security, and delivery pipelines in a multi-module MSA built around API servers that receive frontend requests (the BFF pattern).

Tech Square

PROFESSIONAL PROJECT · 2022.04 – PRESENT

A B2B platform supporting smart-factory adoption for SMEs. Owned full-stack development and operations across matching, mentoring, commerce, data, and notification workflows.

Netplace Client Management

SI PROJECT · 2026

A back office for client database management. Django server-rendered pages with a Tailwind build pipeline.

Netplace Landing Page

SI PROJECT · 2025

A static landing page introducing the service. Semantic markup with sitemap/robots for SEO.

Netplace Marketing Service

SI PROJECT · 2024

Customer-facing web plus an API server for a marketing service — full-stack from planning to operations.

Netplace Marketing Admin

SI PROJECT · 2024

An admin back office for operating the marketing service — customer and service status management.

K-Members

PROFESSIONAL PROJECT · 2020.01 – 2021.02

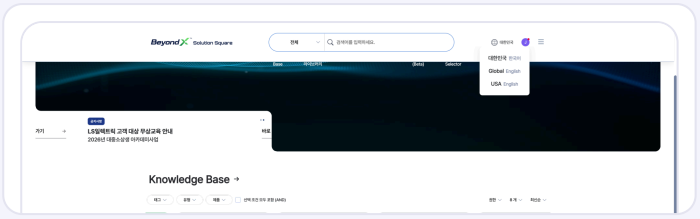
Kyowon Group's integrated membership mall. Built the rental-car discovery and consultation journey, its operations back office, and maintained the existing service.

PROFESSIONAL PROJECT · 2026

Solution Square (Global)

2025 – Present

Screens



Purpose

LS ELECTRIC's global customer portal, providing product information, technical resources, selection tools, and engineering knowledge across regions and languages.

Role · Tech Stack

Contributed to product, dashboard, search, and AI services across Java/Spring backend services, the API servers that receive frontend requests (BFF), and the React/TypeScript front end.

Java · Spring Boot · React · TypeScript · Azure · Elasticsearch · Jenkins

Java | Spring Boot | React | TypeScript | Azure | Elasticsearch | Jenkins

Achievement 01

Unifying Authentication Logic

Background & Cause

The global portal and an asset-management service grew from the same roots.
They originally shared one API server that received frontend requests and handled authentication (a BFF).
When the owning teams split, the asset-management side got its own API server with a hard copy of the same authentication logic.
The copies then diverged and became difficult to diagnose.

Decision & Rationale

I ruled out a dedicated auth server because its operating cost and an extra network hop were not justified for two services.
Following the repository convention for shared modules, I extracted session, authentication-token and SAML utilities, then JWT in stages.
Each stage was verified before the next, and adapters changed only the connection points rather than both applications at once.

Outcome

Authentication changes now end with one edit, and both applications behave identically.
The migration also removed leftover dead code and duplicate dependencies, leaving the JWT library under one owner.

Retrospective

For legacy work, follow existing conventions to reduce the change surface.
Migrating in verifiable units is safer than replacing everything at once.

Achievement 02

Building a Notification Template & Recipient Management System

Background & Cause

Email notification templates were hardcoded as 54 per-language HTML files, each with its own request class and endpoint.
Code grew with every notification type, and changing copy or recipients required a deployment.

Decision & Rationale

I moved templates from code to data so administrators could change copy and recipients without deployments.
A four-axis model—channel, event type, service type, and language—makes a new notification a row addition.
A separate recipient registry, a combined admin API, and one common send implementation avoid duplicated changes and round trips.

Outcome

Administrators can now manage templates and recipients dynamically without deployments.
Later features reuse the existing send logic without new code.
Recipient and delivery-history records were also included in personal-data encryption.

Retrospective

Repeated code of the same shape signals a missing data model.
Moving identifying axes into data lets new requirements end as row additions rather than code changes.

Achievement 03

Encrypting Personal Data Without Breaking Search

Background & Cause

Raising the standard for personal-data protection required encrypting fields such as names and contact details.
Encryption breaks the equality searches that use those values.
Protection and searchability were in direct conflict.

Decision & Rationale

Rather than leave a plaintext search path while solving search later, I designed envelope encryption and blind indexes together.
Key versions preserve a re-encryption path.
Staged write, search, and backfill phases let the service read both plaintext and ciphertext during the live-data transition.

Outcome

Equality search remains available on encrypted values, applied in stages to live data.
The backfill resumes from row state, making interruption and re-execution safe.

Retrospective

Security requirements can be applied to a running system only when their transition path preserves features.
Every encryption-migration phase should be re-enterable.

Achievement 04

Automating Deployment Notifications and Sequential Deploys

Background & Cause

In an environment deploying multiple modules, deployment facts were not shared with the team and commit-level review gaps were possible.
Build-server memory prevented parallel builds, so people waited for each build and manually started the next deployment.

Decision & Rationale

Teams was the primary notification channel, with email as a fallback so a channel failure would not silence automation.
I separated AI review by merge request and push.
Instead of increasing server memory, the pipeline detects completion and starts the next build and deployment automatically.

Outcome

Deployments and review results are shared automatically, and sequential deployments no longer require human waiting.

Retrospective

Notifications are trustworthy only when their failure fallback is designed too.
Automation should target the waiting between tasks as well as the tasks themselves.

Achievement 05

Java 11 → 21 → 25 runtime modernization

Background & Cause

A new application had to be added for mobile-client authentication and per-site project file management.
The existing Java 11 approach constrained what could be built, and leaving an old runtime in place was itself a long-term maintenance risk.

Decision & Rationale

Modernization started alongside that new application.
Java 21 was merged first so new feature work and the runtime move ran in the same period.
Java 25 followed, with a Spring Boot 4 BOM unifying version management.
Jackson 3 migration, single ownership for the JWT library, and per-module build JDK changes were handled together.
That kept dependencies from diverging across modules.

Outcome

What the upgrade broke was repaired alongside it.
HTTPS detection inverted in SAML header handling, the entity-id scheme broke on non-standard ports, and staging returned 403 for origins.
Client IPs lost behind a proxy were restored too, each case narrowed with diagnostic filters.
The same effort moved the front-end build from CRA to Vite and raised the search client to 9.x, clearing stale dependencies on both sides.
The move also let the later SAML standardization sit on a framework-supported path.

Retrospective

The real work in a runtime upgrade is not raising the version.
It is finding and restoring the behavior that quietly broke because of environment differences.

Achievement 06

Standardizing SAML for Future SSO Expansion

Background & Cause

A new partner IdP integration was requested.
The legacy SAML implementation made a flexible, extensible structure difficult to build within the framework.
Spring Boot 4 in particular restricts the legacy IdP approach.

Decision & Rationale

I switched from the homegrown approach to the framework-supported standard so upgrades and extensions can follow the framework path.
An SSO provider registry replaces per-provider branches, and a dedicated security chain isolates IdP rules.
A generic SAML 2.0 mock SP verifies compliance without waiting for each partner environment.

Outcome

IdP-initiated auto-login now runs on a standard foundation.
A generic SAML 2.0 mock SP test application makes each new provider integration repeatable.

Retrospective

Checking what the next framework version restricts and standardizing early lowers the cost of later expansion beyond the immediate request.

Achievement 07

Rule-Based Product Configurator (Drive Panel Selector)

Background & Cause

Customers had to make a technical inquiry every time they needed a drive and panel configuration matching their specifications.
Sales needed a tool that continued from consultation into a quote request within the system.

Decision & Rationale

I used explicit rules instead of a learned model because product configuration has clear constraints between options.
Results must also be explainable when they affect quotes and orders.
Earlier choices dynamically narrow later ones.
Final-step images, customer inputs, inquiry-history integration, and duplicate-request prevention extend the flow into quoting.

Outcome

Customers can narrow a configuration to match their specifications and reach the resulting model without a technical inquiry.
From there they move directly to an inquiry or quote request.

Retrospective

Requirements that look like recommendation can be solved predictably with explicit dependencies between options.
The tool is complete only when it connects to consultation and quoting, not just a result screen.

Visit — https://ssq.ls-electric.com

PROFESSIONAL PROJECT · 2026

Tech Square

2022.04 – Present

Screens

Purpose

A B2B manufacturing platform where demand and supply companies can find smart-factory partners and use mentoring, used-equipment trading, and operational information services.

Role · Tech Stack

Node.js · Express · TypeORM · MySQL · Vue 3 · AWS · GitLab CI/CD

Node.js | Express | TypeORM | MySQL | Vue 3 | AWS | GitLab CI/CD

Achievement 01

Gradually Transitioning from a Java System to a Node.js Platform

Background & Cause

The existing Java 8 system had quality and maintainability limits.
A new platform aligned with the Vue 3 customer and admin channels was needed.

Decision & Rationale

Although I did not participate in the transition decision, I co-built the Node.js, Express, and TypeORM system and handled data migration.
We unified the front-end and language ecosystem to reduce collaboration boundaries.
Old and new systems ran in parallel to verify functions and data before we took over operations.

Outcome

The platform transitioned without operational interruption.
I then independently continued operating and improving the API and the customer and admin channels.

Retrospective

An operational-system transition needs more than new technology.
It finishes reliably only when data migration, parallel operation, and handover-completion criteria are designed together.

Achievement 02

Standardizing Data-Integrity Boundaries with a Shared Transaction Layer

Background & Cause

As business modules grew, SQL execution and transaction boundaries became scattered across services.
That made it difficult to handle data integrity consistently under concurrent requests or partial failures.

Decision & Rationale

Instead of letting each service control TypeORM directly, I built a shared SQL and transaction layer.
Centralizing isolation levels and commit/rollback ownership standardized success and failure boundaries per business operation.
Business conditions and transactions together blocked duplicate requests.

Outcome

Multiple business modules now use the same execution and recovery rules.
That establishes a basis to control duplicate saves and partial writes under concurrent requests.

Retrospective

The purpose of shared transactions is not code reuse but consistent success and failure boundaries for each business operation.

Achievement 03

Improving Repeated Lookups by Restructuring Queries

Background & Cause

As member, company, and mentoring data grew, each list item triggered another lookup for related information.
Query combinations also became more complex as search criteria increased.

Decision & Rationale

Rather than increase time limits or add indexes indiscriminately, I replaced repeated lookups with joins and batched queries.
A parameter-binding query builder organized dynamic search conditions, improving both execution structure and input safety.

Outcome

Repeated lookups were reduced, search criteria can be combined safely, and lists and Excel downloads now share the same lookup rules.

Retrospective

For slow lookups, inspect result sets and repetition counts first.
Correct the query structure, then choose the indexes that are actually needed.

Achievement 04

Moving Local Files to a Shared S3-Based File Flow

Background & Cause

Uploaded files were tied to server-local paths, so browsing and download behavior changed with the environment.
Separate exceptions had accumulated for PDFs, Excel files, and images.

Decision & Rationale

To separate server instances from file lifecycles, I moved to S3 but kept a period in which both old and new paths could be read.
Shared utilities centralized upload, retrieval, deletion, and original-filename restoration.
Storage changes therefore did not spread through business code.

Outcome

Notices, reports, quotations, and images now use one storage approach, reducing environment-dependent file paths and duplicate handling.

Retrospective

A file-store migration is a data migration, not only a copy of objects.
It must preserve existing keys, original names, and retrieval-path compatibility.

Achievement 05

Strengthening Authentication and Input Boundaries in Stages

Background & Cause

Browser-accessible authentication tokens, dynamic SQL, HTML input, and multiple upload paths each created different security risks.

Decision & Rationale

I moved JWTs to HttpOnly cookies to prevent direct browser-script access and converged concurrent 401 responses into one token refresh.
Parameterized SQL and allowlist validation for HTML and files applied defenses appropriate to each boundary.
Authentication, data, input, and file boundaries were handled separately.

Outcome

Authentication flows were unified across customer and admin channels.
That reduced exposure paths leading to script input, SQL injection, or risky file rendering.

Retrospective

Security is not solved by one filter; authentication, input, data, and file boundaries must each be checked and narrowed in stages.

Achievement 06

Building Observability by Connecting Log Fragments into Request Flows

Background & Cause

Service-level `console` logs alone made it difficult to connect how far one request had progressed through API and data processing.

Decision & Rationale

Instead of adding more logs, I standardized all sources on a structured logger and assigned a correlation ID to each request.
Connecting an OpenTelemetry traceId preserves the log format while tracing request flow beyond service boundaries.

Outcome

Operations can now narrow problem locations by request rather than search individual log entries separately.

Retrospective

Observability depends less on log volume than on whether different records can be connected into the same request flow.

Achievement 07

Making Secrets and Local Development Reproducible

Background & Cause

Environment-specific configuration and secret-management approaches were scattered.
An interrupted local database tunnel also made application connectivity unstable.

Decision & Rationale

For a small team with mixed staffing, I chose Infisical over Vault because its adoption barrier was lower.
Docker Compose standardized the runtime environment.
The database tunnel became a separate process with disconnection detection, exponential-backoff reconnection, and duplicate-run prevention.

Outcome

Secrets are separated from code and deployment files, and developers can reproduce the same startup procedure.
Tunnel failures can recover without restarting the entire application.

Retrospective

Standardizing a development environment is not only about aligning tools.
It means defining failure and recovery for configuration, secrets, and external connections.

Achievement 08

Automating Repetitive Work with Batch and Notification Pipelines

Background & Cause

When people repeatedly load external data, check statuses, and notify owners, omissions and processing delays can occur.

Decision & Rationale

I batched bulk data instead of sending per-record requests to reduce database round trips.
Cron jobs and notification flows were configured by business condition.
Duplicate execution, empty results, and environment-specific sending are handled separately.
That prevents automation from creating unnecessary work under exceptional conditions.

Outcome

Company-data ingestion and operational notifications now run on defined conditions and schedules.
That reduces repeated confirmation work for owners.

Retrospective

Batch automation is defined less by periodic execution than by safely handling duplicate processing, empty results, and partial failures.

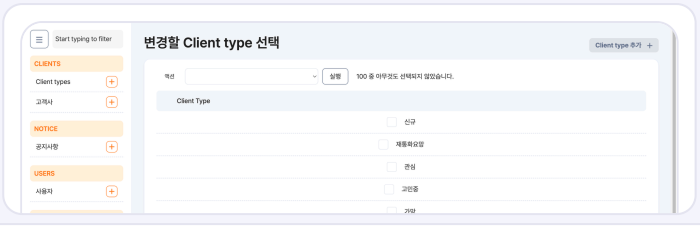
Visit — https://tech-square.co.kr

SI PROJECT · 2026

Netplace Client Management

2026

Screens



Purpose

Built to systematize client database management operations.

Role · Tech Stack

Django 5 · MySQL · Tailwind CSS · Gunicorn

Django 5 | MySQL | Tailwind CSS | Gunicorn

Achievement 01

Separating Environments and Moving SQLite to MySQL Without Rebuilding

Background & Cause

Client information was scattered across spreadsheets, documents, and owner-specific materials.
Current and duplicate data were difficult to distinguish, which limited search, classification, and handover.
The existing outsourced Python system used SQLite without separate local, development, and production environments.

Decision & Rationale

Because a full rebuild was costly, I retained the Python system and first improved the higher-risk runtime and database boundaries.
Separating settings and secrets by environment isolated change impact.
Server-based MySQL replaced file-based SQLite for independent per-environment databases and multi-user connections.
Django `dumpdata/loaddata` preserved existing models and relationships as far as possible.

Outcome

Development changes no longer directly affect production data, enabling environment-specific testing and independent deployment.
Comparing table counts and core CRUD screens before and after migration also confirmed that existing functions remained without data loss.

Retrospective

Rather than always rebuilding an existing system, preserve usable assets.
Separating high-risk boundaries such as data and deployment in stages lowers both cost and transition risk.

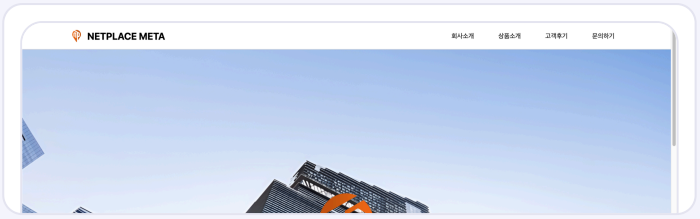
Visit — https://dbcs.net-place.co.kr

SI PROJECT · 2025

Netplace Landing Page

2025

Screens



Purpose

Built to introduce the service and drive search traffic (SEO).

Role · Tech Stack

HTML5 · CSS3 · SEO

HTML5 CSS3 SEO

Achievement 01

Redesigning a Landing Page Around Search Discovery and Inquiry Conversion

Background & Cause

The existing page lacked search visibility and mobile support, while maintenance cost and technical debt had accumulated.
Its design also differed from the client's intended direction.
That made it hard to convey service value to visitors arriving from search, advertising, or sales guidance.

Decision & Rationale

For a content-centered page, I chose semantic static composition over a separate framework.
Reducing browser code and runtime dependencies helps initial loading and search-engine interpretation.
It also allows simple deployment without a server application.
With the designer, I improved information architecture, responsiveness, and CSS animation.
Reusing the existing Marketing API and AWS SES avoided duplicated mail credentials and send logic.

Outcome

The client approved the new design and composition for production use, improving mobile usability and search-engine indexing and visibility.
The inquiry form now sends consultation email through the existing API, and static deployment simplified maintenance procedures.

Retrospective

An introductory page should not add technical complexity.
Choose the minimum composition needed for visitor intent, search discovery, and inquiry conversion.

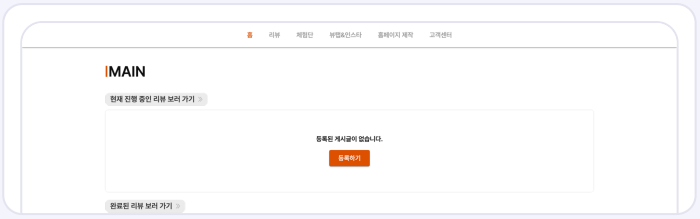
Visit — https://welcome.net-place.co.kr

SI PROJECT · 2024

Netplace Marketing Service

2024

Screens



Purpose

Built to establish an online storefront for a regional marketing service.

Role · Tech Stack

React 18 · NestJS · MySQL · Docker Compose

React 18 | NestJS | MySQL | Docker Compose

Achievement 01

Turning a Manual Application Flow into a 24-Hour Online Service

Background & Cause

Marketing-product information and applications were handled through staff inquiries and manual work.
That risked omissions and duplicates while requiring operators to re-enter application details.
Customers could not easily share processing status, and applications were limited to business hours.

Decision & Rationale

I separated the customer interface and business API into React and NestJS to create independent development and deployment boundaries.
The admin channel reuses the same API and data contract.
This keeps interface changes from directly changing business rules.
It also prevents duplicate application-data implementations across the customer and admin channels.

Outcome

Customers can sign up, log in, browse products, and apply for services regardless of time.
Application information is stored consistently, reducing omissions, duplicates, and repeated operator entry.
Both channels share the same application data.

Retrospective

Putting a process online does not end with the interface.
Define the data structure that prevents omissions and duplicates first, then connect application and operational processing in one flow.

Achievement 02

Separating Customer and Admin Authorization Boundaries with JWT

Background & Cause

Customers and administrators use the same API.
Checking only whether someone is logged in cannot safely distinguish internal operational functions and the scope of data access.

Decision & Rationale

I separated customer and admin authentication paths and applied role-based access control in the API.
JWTs are delivered in HttpOnly cookies, preventing direct token access by browser scripts.
Both channels still use the same authentication contract.

Outcome

Customers and administrators reuse the same business API while being limited to functions allowed for their respective roles.

Retrospective

Authentication design should begin with authorization and data-access boundaries for each user type, not merely successful login.

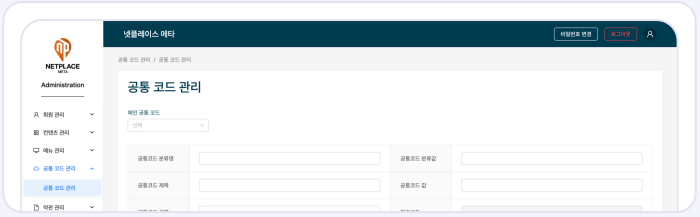
Visit — https://net-place.co.kr

SI PROJECT · 2024

Netplace Marketing Admin

2024

Screens



Purpose

Built as a back office for operating the marketing service — managing customer and service status.

Role · Tech Stack

React 18 · NestJS · MySQL

React 18 NestJS MySQL

Achievement 01

Building an Admin Operations Channel Separate from the Customer Service

Background & Cause

An internal channel was needed to view and process customer applications and manage customers, products, owners, and statuses.
Operational functions that must not be exposed to customers had to be separated.
Standard-administrator and super-administrator permissions also needed separation.

Decision & Rationale

Rather than conditionally place admin functions inside the customer interface, I separated them into a React application.
It keeps deployment and exposure boundaries separate while reusing the same API contract.
Ant Design keeps information-dense operations screens consistent.

Outcome

One admin channel now handles customer, product, and application work, owner assignment, and status changes.
Standard administrators handle operational work, and super administrators manage administrator accounts and permissions.
The channel was deployed to the live environment.

Retrospective

Customer and admin channels can share a data contract.
Their UI, authorization, and deployment boundaries should stay separate so one channel's changes do not unnecessarily constrain the other.

Achievement 02

Standardizing Repeated Lists, Search, and Forms into a Shared Structure

Background & Cause

Customer, product, and application-management screens repeated lists, search, filters, pagination, and create/edit forms.
Independent screen implementations would vary state handling and require the same code again for new screens.

Decision & Rationale

I created shared table, search, and form components and supplied only screen-specific settings and data.
Custom hooks separated search, pagination, and API-call state from UI responsibility, standardizing behavior across screens.

Outcome

The interaction pattern for recurring operations UI is consistent, and the scope of adding or modifying a new admin screen is reduced.

Retrospective

For admin interfaces, share common list, search, and form patterns before expanding individual pages.
That reduces long-term change cost.

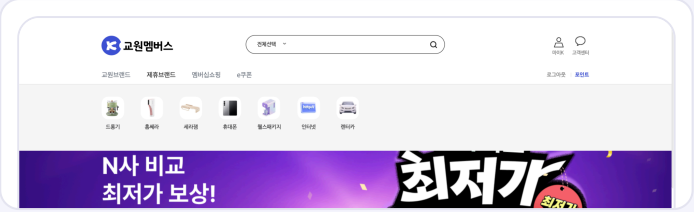
Visit — https://admin.net-place.co.kr

PROFESSIONAL PROJECT · 2020

K-Members

2020.01 – 2021.02

Screens



Purpose

Kyowon Group's integrated membership mall, where group employees and members use points for group and partner products, expanded with a rental-car catalog and consultation journey.

Role · Tech Stack

Spring Framework · MyBatis · Oracle · JSP · JavaScript · jQuery

Spring Framework | MyBatis | Oracle | JSP | JavaScript | jQuery

Achievement 01

Building a Rental-Car Consultation and Point-Operations Flow for Offline Contracts

Background & Cause

K-Members was an integrated Kyowon Group membership mall for employees and members.
It had no sales flow for rental-car products whose revenue arose from an offline contract after consultation.
The process ran from consultation intake through owner assignment, contract confirmation, and point issuance.
It required more complex operations than existing products.

Decision & Rationale

Planning and operations teams defined the application intake → consultant assignment → processing-complete states.
I connected those states to the user interface and back-office logic.
Points are issued when the actual contract is confirmed at processing completion rather than at application time.
That preserves consistency between cancellable offline contracts and point policy.

Outcome

The mall now has a new sales flow in which customers explore rental-car products and request consultation.
Operators manage owner assignment, processing status, and point issuance.

Retrospective

When adding a product with a different nature to an existing service, model the changing states first.
Contract, cancellation, and compensation policy come before the interface.

Achievement 02

Structuring Affiliate Sales Status Around Fixed Months and Variable Affiliates

Background & Cause

Operators needed to review sales by Kyowon Group affiliate each month.
Months are fixed from 1 through 12, while the affiliate list can change, so the two axes could not be handled as one fixed array.

Decision & Rationale

I kept the 12 months as a fixed-size array and stored a list of per-affiliate sales objects in each month.
Separating fixed and variable axes preserves the monthly lookup structure without redesigning the data structure when affiliates change.

Outcome

Operators can view monthly sales status by affiliate on one screen.
The same monthly aggregation structure remains when the affiliate list changes.

Retrospective

When fixed time and changing organization axes must coexist, model the two dimensions separately to reduce the scope of change.

Visit — https://www.kwmembers.com

Thank you

Portfolio

More information about this portfolio is available online at **https://jihwang.kim**.