BACK-END ENGINEER · WEB

作って終わりにせず、デプロイとシークレット管理まで担うバックエンド開発者、キム・ジファン

バックエンドを中心に、設計・デプロイ・運用まで担い、サービスを安定して導きます。
B2Bプラットフォーム・B2C EC・金融ドメインでの開発・運用経験があります。

6+ 年の実務経験　**3** つのドメイン（製造・コマース・金融）

2022 — **LS ELECTRIC 在職中** · ウェブプラットフォームサービス開発

2018 中央大学 情報システム学 学士 卒業

2016 情報処理技士（国家技術資格）取得

2010 徳寿高等学校 卒業（デジタルコンテンツ専攻）

2007 情報処理機能士 · 電算会計運用士3級 · ワードプロセッサー1級·2級 取得

開発者とは、**ソフトウェアで問題を解決し、対話を通じて 持続的なビジネスを創出・拡大していく人**だと考えています。

だから私は、できない理由の代わりに 実現させる基準と手順を つくる開発者として働きます

- **運用まで責任** — 作って終わりではなく、デプロイ・モニタリング・保守まで担います。
- **フルスタック実務** — Java/Spring · Node.jsのバックエンドとVue/Reactのフロントエンドを行き来して開発します。
- **改善志向** — レガシーを Java 11 → 21 → 25、MySQL 8.0 → 8.4、CRA → Vite へ段階的に刷新しました。
- **コミュニケーション** — 顧客・運用者・開発者の間で業務の文脈を確認し、互いに伝わる言葉で調整します。

01 — Career

経歴

製造B2Bプラットフォーム、グループEC、金融インターフェース — 三つのドメインを経験しました。

LS ELECTRIC

電力・自動化分野のグローバルB2Bメーカー

2022.04 — 現在 · 自動化System)事業開発チーム

Solution Square — 営業支援・顧客向けウェブサービス

グローバル製品・技術ポータルのBFF、SAML SSO、データ連携、検索、セキュリティ、デプロイ基盤をモダナイズ。

Tech Square — B2Bスマート製造ウェブプラットフォーム

企業マッチング、メンタリング、中古設備、データ取込、ダッシュボード、通知までを含むフルスタック開発・運用。

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

教員クリエイティブ

教員グループのITサービス開発・運営系列会社

2020.01 — 2021.02 · ITサービス開発3チーム

Kメンバーズ レンタカーサービス — グループ統合メンバーシップモール

Kメンバーズモールの開発・運用、レンタカー相談サービスと管理画面、系列会社の売上レポート、SMS認証を実装。

Spring MyBatis Oracle JSP jQuery

ウェブキャッシュCNS

企業資金管理ソリューションを手がけるウェブキャッシュグループ系列会社

2019.01 — 2020.01 · KEBハナ支社

企業資金管理CMS技術支援 · ERP DB連携

企業向け資金管理CMSの運用を支援し、MS-SQLを基盤にCMS DBと企業ERP DB間の金融データ連携を担当。

MS-SQL 金融ドメイン

02 — Projects

プロジェクト

業務で担当したプラットフォームからSIまで、開発・運用したプロジェクトを紹介します。

Solution Square (Global)

PROFESSIONAL PROJECT · 2025 – PRESENT

LS ELECTRICのグローバル製品・エンジニアリングポータル。フロントエンドのリクエストを受けるAPIサーバー(BFF)を備えたマルチモジュールMSA環境で、ビジネスロジック、認証、セキュリティ、デプロイパイプラインを担当しました。

Tech Square

PROFESSIONAL PROJECT · 2022.04 – PRESENT

中小製造企業のスマートファクトリー導入を支援するB2Bプラットフォーム。マッチング、メンタリング、取引、データ、通知を横断してフルスタック開発・運用を担当しました。

ネットプレイス 顧客管理

SI PROJECT · 2026

顧客データベース管理のバックオフィス。Djangoのサーバーレンダリングと Tailwindビルドパイプラインで構築。

ネットプレイス ランディングページ

SI PROJECT · 2025

サービス紹介の静的ランディングページ。セマンティックマークアップとsitemap/robotsでSEO対応。

ネットプレイス マーケティングサービス

SI PROJECT · 2024

マーケティングサービスの顧客向けウェブとAPIサーバー。企画から運用までフルスタックで担当。

ネットプレイス マーケティング管理者

SI PROJECT · 2024

マーケティングサービス運営のための管理者バックオフィス。顧客・サービス状況の管理機能を提供。

Kメンバーズ

PROFESSIONAL PROJECT · 2020.01 – 2021.02

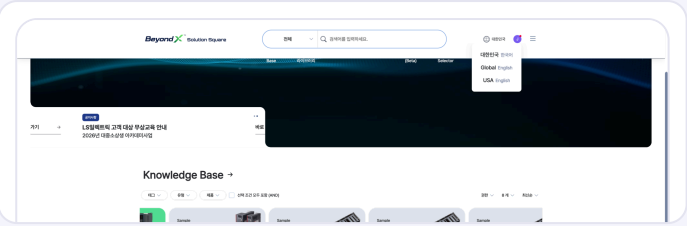
教員グループの統合メンバーシップモール。レンタカーの商品探索・相談申込と運用管理画面を開発し、既存サービスを運用しました。

PROFESSIONAL PROJECT · 2026

Solution Square (Global)

2025 – Present

画面



目的

製品情報、技術資料、選定ツール、エンジニアリング知識を複数の地域と言語で提供するLS ELECTRICのグローバル顧客ポータルです。

役割・技術スタック

Java/Springベースのバックエンドサービスと、フロントエンドのリクエストを受けるAPIサーバー(BFF)、React・TypeScriptのフロントエンドで、製品・ダッシュボード・検索・AIサービスの開発に参加しました。

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

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

実績 01

認証ロジックの共通化

背景・原因

グローバルポータルと資産管理サービスは同じルーツから出発した製品です。
当初はフロントエンドのリクエストを受けて認証とバックエンド連携を担うAPIサーバー（BFF）も一つでした。
担当組織の分離後、資産管理サービス側に同じ認証ロジックのハードコピーを持つAPIサーバーが生まれました。
両者の挙動がずれて原因特定も難しくなっていました。

検討・選択理由

運用対象の増加と全リクエストへのネットワークホップを二つのサービス規模で受け入れる理由がないため、専用の認証サーバー案は見送りました。
既存の共通モジュールの慣例に従い、セッション、認証トークン・SAMLユーティリティ、JWTの順で段階的に抽出しました。
アダプターを置いたのも同じ理由で、二つのアプリケーションを一度に変えず接続点だけを置き換えるためです。

結果

認証ロジックの変更が一箇所の修正で完結し、両アプリケーションの認証挙動が同一になりました。
移行で残ったデッドコードと重複依存を整理し、JWTライブラリも単一所有に統一しました。

振り返り

レガシー改善では、一度に作り直すより既存の慣例に従って変更範囲を減らし、検証可能な単位に分けて移す方が安全です。

実績 02

通知テンプレート・受信者管理システムの構築

背景・原因

メール通知テンプレートは言語別のHTMLファイル54個としてハードコードされていました。
テンプレートごとにリクエストクラスとエンドポイントがありました。
通知の種類とともにコードが増え、文言や受信者の変更にもデプロイが必要でした。

検討・選択理由

管理者がデプロイなしで文言と受信者を扱えるよう、テンプレートをコードからデータへ移しました。
チャネル、イベントタイプ、サービスタイプ、言語の四軸で識別し、新しい通知を行の追加で済ませました。
受信者レジストリの分離、管理画面用の統合API、共通のテンプレート送信により重複とAPI往復を抑えました。

結果

管理者がデプロイなしでテンプレートと受信者を動的に管理できるようになりました。
後続の機能もコード追加なしで既存の送信ロジックを再利用できました。
受信者・送信履歴は個人情報の暗号化対象としても整理しました。

振り返り

同じ形のコードが繰り返し増えるのは、データモデル不在の兆候です。
識別軸をデータへ移せば、新しい要求をコード変更ではなく行の追加で終えられます。

実績 03

個人情報の暗号化と検索の両立

背景・原因

個人情報保護の基準を高めるには、氏名や連絡先のようなフィールドを暗号化する必要がありました。
ただし暗号化すると、その値を条件にする既存の完全一致検索が壊れます。
保護と検索可能性が正面から衝突していました。

検討・選択理由

検索を後回しにして平文検索の経路を残さないよう、エンベロープ暗号化とブラインドインデックスを最初から併せて設計しました。
鍵バージョンを保存し、再暗号化の経路を残しました。
書き込み、検索、バックフィルの順に段階を分け、運用データの移行中は平文と暗号文の両方を読めるようにしました。

結果

暗号化された値でも完全一致検索を維持したまま、運用中のデータへ段階的に適用しました。
バックフィルは行の状態を基準に再開できるため、中断後の再実行も安全です。

振り返り

保全要件を運用中のシステムへ適用するには、機能を壊さない移行経路まで設計する必要があります。
暗号化移行の各段階は再進入可能な状態にすべきです。

実績 04

デプロイ通知・連続デプロイの自動化

背景・原因

複数モジュールをデプロイする環境で、誰がいつデプロイしたかがチームに共有されていませんでした。
コミット単位のレビューで見落としが生じる余地もありました。
ビルドサーバーのメモリ制約で並列ビルドができず、人が前のビルドを待って次のデプロイを手動で開始していました。

検討・選択理由

一つのチャネル障害で自動化が途切れないよう、チームが日常的に見るTeamsを基本とし、送信失敗時はメールへフォールバックしました。
AIレビューはマージリクエスト単位とプッシュ単位に分けました。
デプロイはサーバーメモリを増やす代わりに、パイプラインが前の完了を検知して次のビルドとデプロイを自動で開始するようにしました。

結果

デプロイの事実とレビュー結果が自動で共有され、連続デプロイで人が待つ必要がなくなりました。

振り返り

通知は成功経路だけでなく送信失敗時のフォールバックまで設計してこそ信頼できます。
自動化では作業そのものだけでなく、作業の間の待ち時間も対象にすべきです。

実績 05

Java 11 → 21 → 25 ランタイム現代化

背景・原因

モバイルクライアント認証と顧客サイト単位のファイル管理を担う新しいアプリケーションを追加する必要がありました。
既存のJava 11の方式では実装に制約があり、古いランタイムを残すこと自体も長期的な保守リスクでした。

検討・選択理由

現代化は新規アプリケーションの構築と合わせて始めました。
まずJava 21をマージし、新機能開発とランタイム移行を同じ期間に進めました。
続いてJava 25へ上げ、Spring Boot 4のBOMでバージョン管理を統合しました。
Jackson 3移行、JWTライブラリの単一所有への整理、モジュールごとのビルドJDK切り替えも併せて処理しました。
モジュール間で依存が分かれないようにするためです。

結果

アップグレードが壊したものも併せて復旧しました。
SAMLヘッダー処理でのHTTPS判定の反転、entity-idのスキームと非標準ポートの問題が出ました。
ステージングの403やプロキシ越しに失われたクライアントIPも、診断フィルターまで入れて原因を絞り戻しました。
同じ流れでフロントエンドのビルドをCRAからViteへ移し、検索クライアントも9.xへ上げました。
バックエンドとフロントエンドの古い依存を併せて取り除きました。
この移行により、後のSAML標準化もフレームワークが支援する方式の上で進められました。

振り返り

ランタイムのアップグレードで実際に手がかかるのは、バージョンを上げることではありません。
環境差で静かに壊れた挙動を見つけて戻すことが本体です。

実績 06

SAML標準化とSSO拡張基盤

背景・原因

新しいパートナーIdPとのSSO連携が求められました。
レガシーなSAML実装ではフレームワーク上、柔軟で拡張可能な構造を作ることが困難でした。
特にSpring Boot 4では従来のIdP実装方式が制限されます。

検討・選択理由

独自方式を修正し続けるのではなく、フレームワークが支援する標準方式へ移行しました。
標準の上にあればバージョンアップに耐え、拡張もフレームワークの流儀で扱えるからです。
SSOプロバイダーレジストリで提供者ごとの分岐を置き換え、専用のセキュリティチェーンでIdPの規則を分離しました。
汎用SAML 2.0 Mock SPで各パートナーの実環境を待たずに準拠を確認します。

結果

IdP起点の自動ログインが動作する標準基盤を整えました。
汎用SAML 2.0 Mock SPテストアプリケーションも用意し、新しい提供者との連携を繰り返し検証できるようにしました。

振り返り

次のフレームワーク版で制限される方式を先に確認して標準化すれば、目先の要求を超えた後続の拡張コストを下げられます。

実績 07

ルールベース製品構成ツール（Drive Panel Selector）

背景・原因

顧客は仕様に合うドライブ・パネル構成を探すたびに技術問い合わせを行う必要がありました。
営業からは、システム内で相談から見積もり依頼までつながるツールが求められていました。

検討・選択理由

学習モデルではなく明示的なルールを採用したのは、製品構成がオプション間の制約が明確な工学的問題だからです。
見積もりや受注につながる結果には説明可能性も必要です。
前段の選択に応じて後段の選択肢を動的に絞り込みました。
最終段階のモデル画像、顧客入力、問い合わせ履歴連携、重複依頼防止も相談・見積もりの流れにつなげました。

結果

顧客は技術問い合わせなしで仕様に合う構成を絞り込み、結果モデルに到達できます。
その場で問い合わせや見積もり依頼へ進めるようにもなりました。

振り返り

推薦のように見える要求も、オプション間の依存関係を明示的なルールにすれば予測可能に扱えます。
ツールは結果画面だけでなく、相談と見積もりまでつながって完成します。

開く — https://ssq.ls-electric.com

PROFESSIONAL PROJECT · 2026

Tech Square

2022.04 – Present

画面

目的

需要企業と供給企業がスマートファクトリーのパートナーを探し、メンタリング、中古設備取引、運用情報を利用できるB2B製造プラットフォームです。

役割・技術スタック

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

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

実績 01

JavaシステムからNode.jsプラットフォームへの段階的移行

背景・原因

既存のJava 8システムには品質と保守性の限界がありました。
Vue 3の顧客・管理者チャネルと技術構成を合わせた新しいプラットフォームが必要でした。

検討・選択理由

移行の意思決定には参加していません。
Node.js・Express・TypeORM基盤のシステムを共同構築し、データ移行を担当しました。
フロントエンドと言語エコシステムを統一し、協業の境界を減らしました。
旧・新システムを並行稼働して機能とデータを検証した後に運用を引き継ぎました。

結果

運用を止めずに新しいプラットフォームへ移行し、その後はAPIと顧客・管理者チャネルの運用・改善を単独で継続しました。

振り返り

運用システムの移行は、新技術だけでなく、データ移行、並行運用、引き継ぎ完了の基準を併せて設計してこそ安定して完了します。

実績 02

共通トランザクション層によるデータ整合性基準の統一

背景・原因

業務モジュールの増加に伴い、SQL実行とトランザクション境界がサービスごとに分散しました。
同時リクエストや部分失敗でデータ整合性を一貫して扱うことが難しくなっていました。

検討・選択理由

各サービスにTypeORMを直接制御させる代わりに、共通のSQL・トランザクション層を作りました。
分離レベルとコミット・ロールバックの責任を一箇所に集め、業務単位の成功・失敗基準を統一しました。
業務条件とトランザクションを併用して重複リクエストも防ぎました。

結果

複数の業務モジュールが同じ実行・復旧基準を使うようになり、同時リクエストでも重複保存と部分反映を制御する基盤を整えました。

振り返り

トランザクション共通化の目的はコード再利用ではなく、業務単位の成功と失敗の境界を一貫させることです。

実績 03

クエリ再構成による繰り返し参照の改善

背景・原因

会員、企業、メンタリングのデータ増加により、一覧の各項目で関連情報を再取得する構造が生まれました。
検索条件が増えるほどクエリの組み合わせも複雑になっていました。

検討・選択理由

制限時間を延ばしたり無計画にインデックスを追加したりせず、繰り返し参照を結合と一括取得に置き換えました。
動的な検索条件はパラメータバインディングに基づくクエリビルダーで整理し、実行構造と入力安全性をともに改善しました。

結果

繰り返し参照を減らし、検索条件を安全に組み合わせられるようにしました。
一覧とExcelダウンロードも同じ参照基準を共有しています。

振り返り

遅い参照では、まず結果集合と繰り返し回数を確認し、クエリ構造を正してから必要なインデックスを選ぶべきです。

実績 04

ローカルファイルをS3ベースの共通ファイルフローへ移行

背景・原因

アップロードしたファイルがサーバーのローカルパスに縛られ、環境によって参照・ダウンロード処理が変わっていました。
PDF、Excel、画像ごとに個別の例外処理も蓄積していました。

検討・選択理由

サーバーインスタンスとファイルのライフサイクルを分離するためS3へ移行しました。
既存ファイルは一度に廃止せず、両方のパスを読める期間を設けました。
アップロード、参照、削除、元のファイル名の復元を共通ユーティリティに集約し、保存方式の変更が業務コードへ広がらないようにしました。

結果

通知、報告書、見積書、画像など異なるファイルフローが共通の保存方式を使うようになり、環境依存のパスと重複処理を減らしました。

振り返り

ファイルストアの移行は単なるオブジェクトコピーではなく、既存キー、元の名前、参照パスの互換性も保つデータ移行です。

実績 05

認証と入力の境界を段階的に強化

背景・原因

ブラウザーからアクセスできる認証トークン、動的SQL、HTML入力、複数のアップロード経路がありました。
それぞれが異なるセキュリティリスクを生んでいました。

検討・選択理由

JWTをHttpOnly Cookieへ移し、ブラウザースクリプトからの直接アクセスを防ぎました。
同時の401応答は一回のトークン更新に収束させました。
SQLのパラメータ化とHTML・ファイルの許可リスト検証により、認証、データ、入力、ファイルの境界ごとに適切な防御を適用しました。

結果

顧客・管理者チャネルの認証フローを統一しました。
スクリプト入力、SQLインジェクション、危険なファイルレンダリングにつながり得る露出経路を減らしました。

振り返り

セキュリティは一つのフィルターでは解決できません。
認証、入力、データ、ファイルの境界をそれぞれ確認し、段階的に絞り込む必要があります。

実績 06

ログ断片をリクエストフローにつなぐ可観測性の構築

背景・原因

サービスごとの`console`ログだけでは、一つのリクエストがAPIとデータ処理をどこまで進んだかをつなげて確認することが困難でした。

検討・選択理由

ログを増やす代わりに、全ソースを構造化ロガーへ統一し、リクエストごとに相関IDを付与しました。
OpenTelemetryのtraceIdも接続し、ログ形式を保ちながらサービス境界を越えてリクエストフローを追跡できるようにしました。

結果

個々のログを別々に検索する代わりに、リクエスト単位で問題の発生箇所を絞り込める運用基盤を整えました。

振り返り

可観測性ではログ量より、異なる記録を同じリクエストフローにつなげられるかが重要です。

実績 07

シークレットとローカル開発環境を再現可能な構成へ移行

背景・原因

環境ごとの設定とシークレット管理方式が分散していました。
ローカルのデータベーストンネルが切れると、アプリケーション接続も不安定になっていました。

検討・選択理由

少人数で混成のチームにおける運用負担を考え、Vaultより導入障壁の低いInfisicalを選びました。
Docker Composeで実行環境を統一しました。
データベーストンネルは切断検知、指数バックオフ再接続、二重起動防止を備えた別プロセスに分離しました。

結果

シークレットをコードとデプロイファイルから分離し、開発者が同じ実行手順を再現できるようにしました。
トンネル障害もアプリケーション全体の再起動に頼らず復旧できます。

振り返り

開発環境の標準化はツールを揃えるだけではなく、設定、シークレット、外部接続の失敗と復旧方法をともに定義することです。

実績 08

バッチ・通知パイプラインによる反復業務の自動化

背景・原因

外部データの取込み、状態確認、担当者への案内を人が繰り返すと、漏れや処理遅延が発生する可能性がありました。

検討・選択理由

データベース往復を減らすため、大量データは一件ごとのリクエストではなく一括で取り込みました。
業務条件ごとにcronジョブと通知フローを構成しました。
重複実行、空の結果、環境ごとの送信可否を分け、例外状況で自動化が不要な作業を生まないようにしました。

結果

企業データの取込みと運用通知が定めた条件とスケジュールに従って実行され、担当者の繰り返し確認作業を減らしました。

振り返り

バッチ自動化では、定期実行そのものより、重複処理、空の結果、部分失敗を安全に扱うことが重要です。

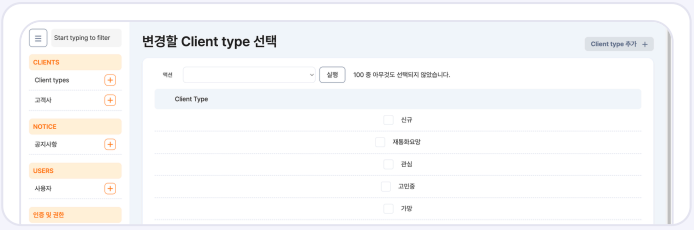
開く — https://tech-square.co.kr

SI PROJECT · 2026

ネットプレイス 顧客管理

2026

画面



目的

顧客企業のDB管理業務をシステム化するために進めました。

役割・技術スタック

Django 5 · MySQL · Tailwind CSS · Gunicorn

Django 5 MySQL Tailwind CSS Gunicorn

実績 01

再構築せずに環境を分離し、SQLiteをMySQLへ移行

背景・原因

顧客情報がExcel、文書、担当者ごとの資料に分散し、最新情報と重複データを区別しにくく、検索、分類、引き継ぎにも制約がありました。
既存の外注Pythonシステムは、ローカル、開発、運用の環境を分けないままSQLiteを使っていました。

検討・選択理由

全面的な再構築はコストが大きいため、既存のPythonシステムを維持し、リスクの高い実行環境とデータベースの境界から改善しました。
環境ごとに設定とシークレットを分けて変更影響を隔離しました。
ファイル単位のSQLiteをサーバー型MySQLへ置き換え、環境別データベースと複数ユーザー接続を独立して運用できるようにしました。
データはDjangoの`dumpdata/loaddata`で移行し、既存モデルと関係をできるだけ保持しました。

結果

開発変更が運用データへ直接影響しなくなり、環境ごとのテストと独立したデプロイが可能になりました。
移行前後のテーブル件数と主要CRUD画面を照合し、データ欠落なく既存機能が維持されていることも確認しました。

振り返り

既存システムを常に作り直すのではなく、使える資産は残すべきです。
データやデプロイのようなリスクの高い境界から段階的に分離する方が、コストと移行リスクをともに下げられます。

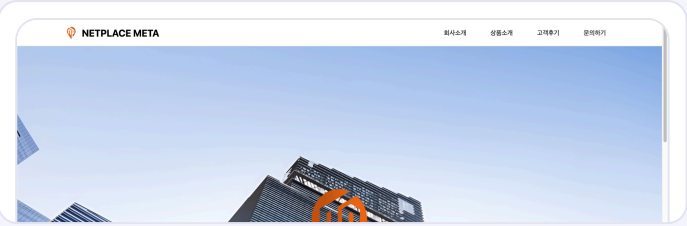
開く — https://dbcs.net-place.co.kr

SI PROJECT · 2025

ネットプレイス ランディングページ

2025

画面



目的

サービス紹介と検索流入（SEO）の確保のために進めました。

役割・技術スタック

HTML5 · CSS3 · SEO

HTML5 CSS3 SEO

実績 01

検索流入と問い合わせ転換をつなぐランディングページ改修

背景・原因

既存ページは検索露出とモバイル対応が不足し、保守コストと技術的負債が蓄積していました。
デザインも顧客が望む方向と異なり、検索、広告、営業案内から来た見込み顧客へサービス価値を明確に伝えにくい状態でした。

検討・選択理由

コンテンツ中心のページには、別のフレームワークよりセマンティックな静的構成が適すると判断しました。
ブラウザー実行コードとランタイム依存を減らせば、初期表示と検索エンジンの解釈に有利だからです。
サーバーアプリケーションなしで簡潔にデプロイもできます。
デザイナーと情報構造、レスポンシブ対応、CSSアニメーションを改善しました。
既存のMarketing APIとAWS SESを再利用し、メール認証情報と送信ロジックの重複を避けました。

結果

顧客が新しいデザインと構成を承認し、運用へ反映しました。
モバイルの使いやすさと検索エンジンのインデックス・露出を改善しました。
問い合わせフォームから既存APIを通じて相談メールを送る流れを完成させ、静的デプロイで保守手順も簡素化しました。

振り返り

紹介ページでは技術的な複雑さを増やすより、訪問目的、検索露出、問い合わせ転換に必要な最小構成を選ぶべきです。

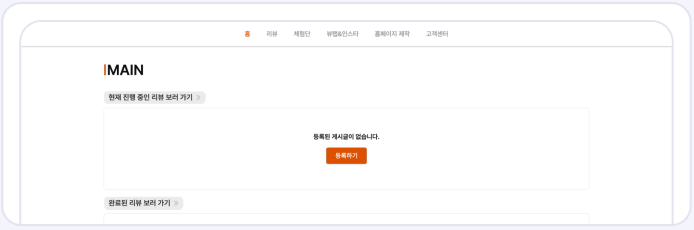
開く — https://welcome.net-place.co.kr

SI PROJECT · 2024

ネットプレイス マーケティングサービス

2024

画面



目的

地域マーケティングサービスのオンライン窓口を構築するために進めました。

役割・技術スタック

React 18 · NestJS · MySQL · Docker Compose

React 18 NestJS MySQL Docker Compose

実績 01

手作業の申請フローを24時間オンラインサービスへ移行

背景・原因

マーケティング商品の案内と申請は、担当者への問い合わせと手作業で処理していました。
情報の漏れや重複のリスクがあり、運用者は申請内容を繰り返し入力していました。
顧客との処理状況の共有も難しく、申請可能な時間は営業時間に限られていました。

検討・選択理由

顧客画面と業務APIをReactとNestJSで分け、独立した開発・デプロイ境界を作りました。
管理者チャネルも同じAPIとデータ契約を再利用するように構成しました。
画面変更が業務規則へ直接影響せず、顧客・管理者チャネルで申請データを重複実装しないためです。

結果

顧客が時間に関係なく会員登録・ログイン後に商品を閲覧し、サービスを申請できるようになりました。
申請情報は一貫した形式で保存され、漏れ、重複、運用者の繰り返し入力を減らし、両チャネルが同じ申請データを共有しています。

振り返り

オンライン化は画面を作るだけでは終わりません。
漏れと重複を防ぐデータ構造を先に定め、顧客申請と運用処理を一つの流れにつなぐ必要があります。

実績 02

JWTによる顧客・管理者の権限境界の分離

背景・原因

顧客と管理者が同じAPIを使うため、ログイン済みかどうかだけでは内部運用機能とデータアクセス範囲を安全に区別できませんでした。

検討・選択理由

顧客・管理者の認証経路を分け、APIにロールベースのアクセス制御を適用しました。
JWTはHttpOnly Cookieで渡し、ブラウザースクリプトがトークンへ直接アクセスできないようにしました。
それでも両チャネルは同じ認証契約を使えます。

結果

顧客と管理者が同じ業務APIを再利用しながらも、それぞれの役割に許可された機能だけにアクセスする境界を分離しました。

振り返り

認証ではログイン成功より先に、ユーザー種別ごとの権限とデータアクセスの境界を設計すべきです。

開く — https://net-place.co.kr

SI PROJECT · 2024

ネットプレイス マーケティング管理者

2024

画面

目的

マーケティングサービス運営（顧客・サービス状況管理）のためのバックオフィスとして進めました。

役割・技術スタック

React 18 · NestJS · MySQL

React 18 NestJS MySQL

実績 01

顧客サービスから分離した管理者運用チャネルの構築

背景・原因

顧客申請の確認・処理と、顧客、商品、担当者、状態の管理を行う内部チャネルが必要でした。
顧客に見せてはならない運用機能に加え、一般管理者と最高管理者の権限も分ける必要がありました。

検討・選択理由

管理者機能を顧客画面に条件付きで置かず、別のReactアプリケーションに分離しました。
顧客チャネルとデプロイ・露出の境界を分けながら、同じAPI契約は再利用するためです。
情報密度の高い運用画面はAnt Designで一貫して構成しました。

結果

一つの管理者チャネルで顧客、商品、申請業務、担当者の割当、状態変更を処理できるようになりました。
一般管理者は運用業務を、最高管理者は管理者アカウントと権限までを管理し、実際の運用環境へもデプロイしました。

振り返り

顧客チャネルと管理者チャネルはデータ契約を共有できますが、UI、権限、デプロイの境界は分けるべきです。
そうすれば一方の変更が他方を不必要に制約しません。

実績 02

繰り返される一覧・検索・フォームの共通構造化

背景・原因

顧客、商品、申請の管理画面ごとに、一覧、検索、フィルター、ページネーション、登録・編集フォームが繰り返されていました。
画面ごとに独立実装すると状態処理がばらつき、新しい画面を追加するたびに同じコードを再び書く必要がありました。

検討・選択理由

共通のテーブル、検索、フォームコンポーネントを作り、画面ごとの設定とデータだけを渡す構成にしました。
検索、ページネーション、API呼び出しの状態をカスタムフックへ分け、UIとデータ処理の責任を分離して画面間の動作を統一しました。

結果

繰り返される運用UIの操作方法を一貫して保ち、新しい管理画面の追加・変更範囲を減らしました。

振り返り

管理者画面では、繰り返される一覧、検索、フォームのパターンを先に共通化すべきです。
個別ページを急いで増やすより、長期的な変更コストを減らせます。

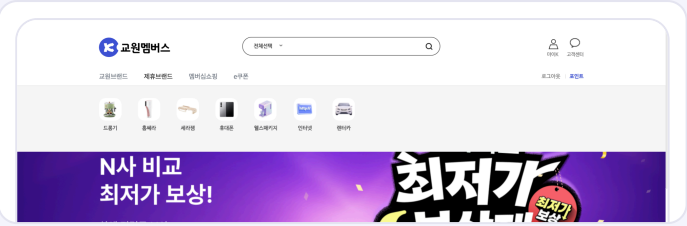
開く — https://admin.net-place.co.kr

PROFESSIONAL PROJECT · 2020

Kメンバーズ

2020.01 – 2021.02

画面



目的

教員グループの社員と会員がポイントでグループ商品や提携商品を利用する統合メンバーシップモールに、レンタカー商品と相談申込の導線を追加しました。

役割・技術スタック

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

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

実績 01

オフライン契約を含むレンタカー相談・ポイント運用フローの構築

背景・原因

K-Membersは従業員と会員が利用する教員グループの統合メンバーシップモールでした。
ただし、相談後のオフライン契約で売上が発生するレンタカー商品の販売フローはありませんでした。
相談受付から担当者割当、契約確認、ポイント付与まで、既存商品より複雑な運用手順が必要でした。

検討・選択理由

企画・運用組織が定義した申請受付 → 相談員の割当 → 処理完了の状態を、ユーザー画面とバックオフィスのロジックにつなげました。
ポイントは申請時ではなく、実際の契約が確認される処理完了時に付与し、取り消し得るオフライン契約とポイント方針の整合性を保ちました。

結果

モール内でレンタカー商品を探して相談を申し込める新しい販売フローを整えました。
運用者は担当者の割当、処理状態、ポイント付与まで管理できます。

振り返り

既存サービスに性質の異なる商品を追加する際は、画面より先に、契約、取消、補償方針が変わる状態をモデル化すべきです。

実績 02

固定月と可変の系列会社をともに扱う売上状況構造

背景・原因

運用者は教員グループの系列会社ごとの売上を月単位で確認する必要がありました。
月は1月から12月まで固定ですが、系列会社の一覧は変わり得るため、二つの軸を同じ固定配列で扱うことはできませんでした。

検討・選択理由

12か月は固定サイズの配列とし、各月に系列会社ごとの売上オブジェクト一覧を持たせました。
固定軸と可変軸を分けることで、月別の参照構造を保ちながら、系列会社の増減でデータ構造を作り直さないようにしました。

結果

運用者が系列会社ごとの月間売上状況を一つの画面で確認でき、系列会社の一覧が変わっても同じ月別集計構造を維持できるようになりました。

振り返り

固定された時間軸と変化する組織軸をともに扱うときは、二つの次元を分けてモデル化すると変更範囲を減らせます。

開く — https://www.kwmembers.com

ありがとうございました

Thank you

このポートフォリオは **https://jihwang.kim** でさらに詳しい情報をオンラインでご覧いただけます。