Frontend Developer

Portfolio

화면 너머의 사용자 경험을 상상하며 코딩하는 몰입형 개발자

이가을 (Gaeul Lee)

gaeulzzang@gmail.com

010-2345-7415

https://github.com/gaeulzzang

http://www.linkedin.com/in/gaeulzzang

안녕하세요,
화면 너머의 사용자 경험을
상상하며 코딩하는
몰입형 개발자 이가을입니다.

다양한 도메인을 통해 다져진 프론트엔드 사고

모바일(Android) 개발을 먼저 경험한 뒤 웹 프론트엔드 영역으로 확장하며, 플랫폼이 달라도 상태 관리와 사용자 흐름을 설계하는 본질은 동일하다는 것을 체감했습니다. 화면 전환과 상태 변화, 사용자 입력에 대한 반응을 다루는 과정은 결국 같은 문제를 다른 환경에서 풀어내는 일이라고 느꼈습니다.

이러한 경험을 통해 특정 플랫폼에 국한되지 않고, 사용자 경험과 상태를 중심으로 사고하는 프론트엔드 개발자로 성장해 왔습니다.

UX 완성도를 향한 열정

주어진 요구사항을 구현하는 데서 멈추지 않고, 사용자가 이 화면에서 무엇을 기대하며 어떤 흐름으로 접근하는지를 기준으로 개발해 왔습니다. 서비스 전반의 흐름을 살피며 불필요한 단계나 개선이 필요하다고 판단되는 지점에 대해서는 적극적으로 의견을 제안했습니다.

기획과 디자인 전반을 함께 이해하는 개발자를 지향하며, 기획 단계에서 놓치기 쉬운 사용자 플로우와 예외 케이스를 사전에 점검해 왔습니다. 사용자가 한 번 더 생각하게 되는 순간을 줄이기 위해, 사소해 보이는 부분까지 여러 번 되짚으며 구현합니다.

작업 효율을 높이기 위한 협업 체계 정립

협업 과정에서 작업 효율을 높이기 위해 개발 컨벤션과 상호 규칙을 노션에 정리하며 협업 체계를 구축해 왔습니다.
프로젝트 진행 시에는 Discord와 Slack에 웹훅을 연동해 PR과 이슈 상황을 즉각적으로 공유함으로써, 지연 없이 빠른 피드백이 이루어질 수 있도록 했습니다.

코드 리뷰 과정에서는 Pn 룰을 차용해 반영 우선순위를 명확히 구분하고, 논의가 필요한 사항과 즉시 반영 가능한 항목을 나누어 효율적인 리뷰 문화를 만들어 왔습니다.
또한 프로젝트 종료 후에는 회고를 통해 협업 과정과 개발 방식을 돌아보며, 다음 프로젝트에서 더 나은 방향으로 개선해 왔습니다.

다시 함께 일하고 싶은 팀원

서비스에 애정을 가지고 개발하기 때문에, 자연스럽게 밝고 적극적인 태도로 협업에 참여해 왔습니다. 논의가 많은 상황에서도 팀이 편안하게 의견을 나눌 수 있는 분위기를 만드는 데 기여했습니다.

의견 충돌이 발생할 때에는 각자의 생각을 정리해 공유하며, 감정적인 대립보다는 다음 행동으로 이어질 수 있는 방향을 함께 고민해 왔습니다. 그 결과 프로젝트 이후에도 다시 협업을 이어가거나, 지속적으로 연락을 주고받는 관계로 이어진 경험이 많습니다.

개발자는 기술뿐만 아니라 함께 일하는 경험을 만드는 태도 역시 중요한 경쟁력이라고 생각하며, 앞으로도 팀의 성과와 시너지를 높이는 개발자로 성장하고자 합니다.

Education

2021.03 ~ 2026.02 **숙명여자대학교 컴퓨터과학전공**
학점 4.21/4.3

Career

2025.09 ~ 2025.12 **푼타컴퍼니**
개발파트 / 프론트엔드 인턴

Experiences

2025.03 ~ 2025.08 **UMC**
대학생 IT 연합 사이드 프로젝트 동아리 | Web

2024.09 ~ 2025.05 **Google Developer Groups on Campus**
구글 학생 개발자 커뮤니티 | Android Core Member

2024.03 ~ 2025.02 **SOLUX**
교내 프로그래밍 중앙 동아리

2024.03 ~ 2024.07 **SOPT**
대학생 IT 연합 사이드 프로젝트 동아리 | Android

2023.09 ~ 2024.07 **DACOS**
교내 데이터 분석 학부 동아리

2021.03 ~ 2022.12 **APPS**
교내 게임 개발 동아리

Certificates

2025.05.24 **TOPCIT** 722점(수준 4)
2024.12.11 **정보처리기사**
2024.04.05 **SQLD 개발자** (SQLD)
2024.12.07 **OPIc** AL(영어)
2024.05.26 **TOEIC** 950점(영어)

Awards

2025.08.22 **8기 UMC 데모데이** 대상(1등)
2025.08.19 **숙명여자대학교 SOSP 우수활동** 우수상(3등)
2025.07.10 **숙명여자대학교 제 23회 TOPCIT 정기평가** 특별상(1등)
2024.12.18 **제 8회 개방형 클라우드 플랫폼(K-Paas) 기반 서비스 개발 공모전** 장관상(1등)
2024.11.09 **숙명여자대학교 SW중심대학사업단 연합해커톤** 대상(1등)
2024.07.20 **NOW SOPT APPJAM** 우수상(3등)
2024.05.19 **NOW SOPT SOPTKATHON** 대상(1등)
2023.02.28 **숙명여자대학교 학업우수상** 최우등상(1등)
2021.10.31 **숙명여자대학교 1기 해커톤** 우수상(3등)

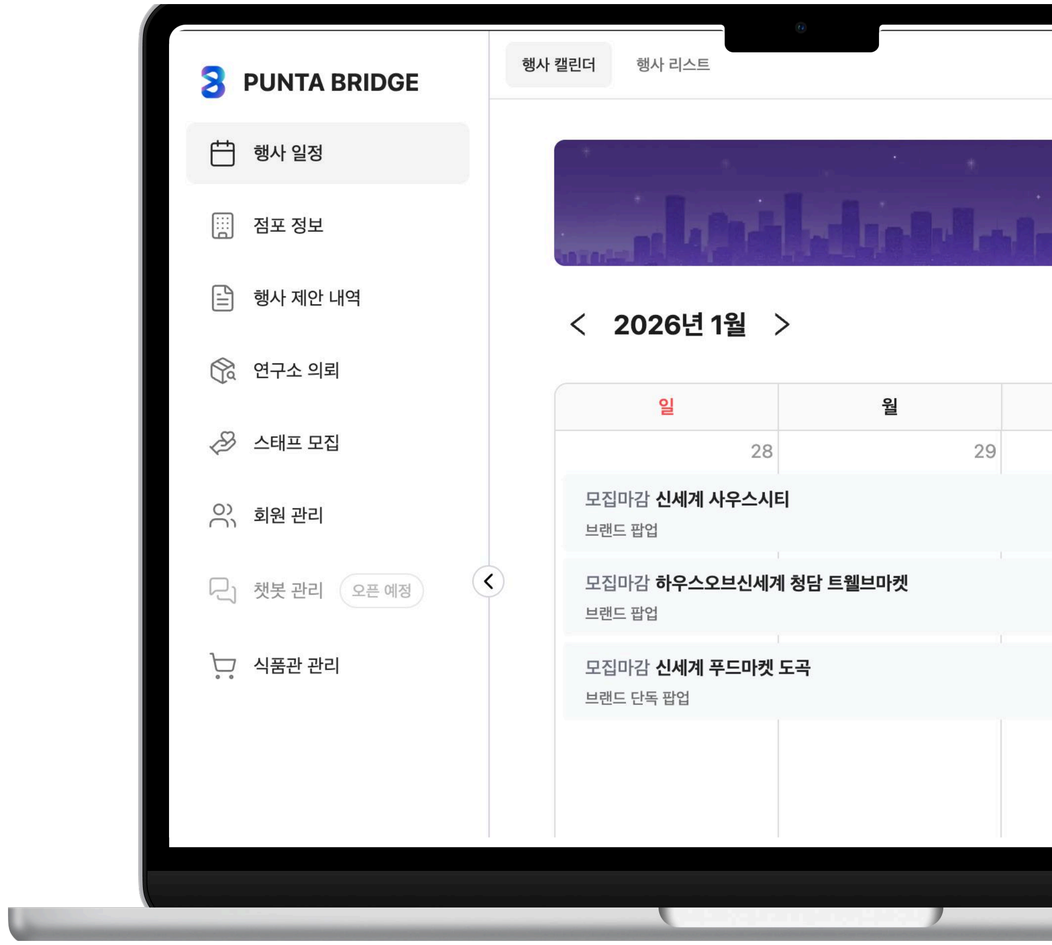
포트폴리오를 통해 개발자로서의 성장 과정과
프론트엔드 개발자로서 보유한 **역량**,
그리고 실제로 수행한 **프로젝트**들을
소개하고자 합니다.

Frontend Developer

Projects

Punta Bridge

브랜드의 오프라인 팝업 행사 운영을 연결하는 B2B 파트너 서비스



Punta Bridge Desktop / Mobile

2025.08 ~ 2025.12

개발 인원(4명) - 프론트엔드 2명 · 백엔드 2명

Typescript Next.js Vanilla Extract Ant Design Radix UI Zustand Tanstack Query Yarn GCP Vercel MongoDB

Overview

Punta Bridge는 브랜드의 오프라인 팝업 행사 운영 과정을 온라인 플랫폼으로 연결하는 **B2B 파트너 서비스**입니다. 팝업 제안서 작성부터 행사 일정 관리, 운영 결과 확인까지의 전 과정을 하나의 서비스로 통합해, 파트너는 보다 효율적으로 행사를 신청할 수 있고 관리자는 이를 일관된 흐름으로 운영할 수 있도록 **기획부터 프론트엔드 개발까지 전담**했습니다.

서비스 초기 단계부터 참여해 사내 인터뷰를 통해 운영 과정에서의 문제를 정의했습니다. 이후 개발–검증–배포–운영 가이드 문서화까지 전 과정을 경험하며, 단순 구현을 넘어 운영을 고려한 서비스 구조를 만드는 데 집중했습니다.

Tasks

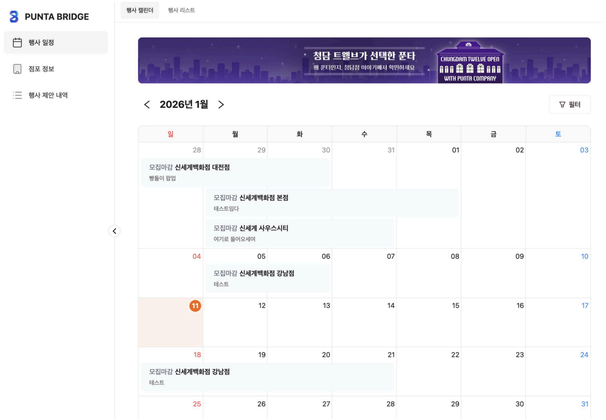
모바일 반응형 UI 구현 및 Headless UI 도입

AS-IS

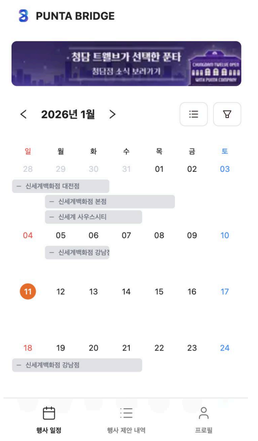
- 모바일·데스크톱 환경 모두에서 사용되지만 UI가 데스크톱 중심
- Ant Design 기반 컴포넌트는 CSR 기반이며 커스텀이 어려움
- 디자인 변경 시 컴포넌트 재사용성과 확장성이 떨어지는 문제 발생

TO-BE

- 미디어 쿼리를 활용해 사이드바 노출 여부를 화면 크기에 따라 제어
- 화면 너비 기준으로 UI를 분기 처리하는 useIsMobile 커스텀 훅 구현
- Vanilla Extract와의 호환성을 고려해 Headless UI 기반 Radix UI 도입
- 공용 컴포넌트를 직접 구현하며 Ant Design 의존성을 점진적으로 제거



데스크탑 UI/UX



모바일 UI/UX

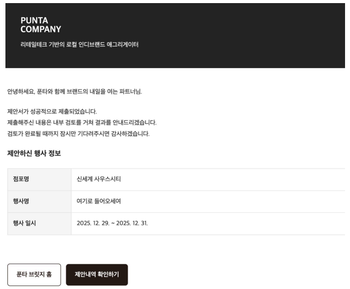
파트너 알림·제안서 기능 개편

AS-IS

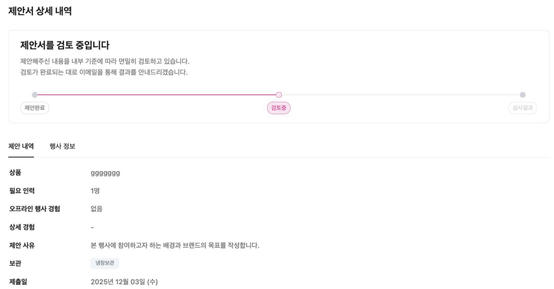
- 알림톡과 메일을 함께 사용해 비용이 발생하고, 알림 채널이 분산된 상태
- 메일 안내 시 서비스로 바로 진입할 수 있는 링크가 없어 사용률이 낮음 (ex. 새로운 행사 등록 안내 메일에서 행사 상세 페이지로 즉시 접근 불가)
- 제안서의 진행 상태를 한눈에 파악하기 어려운 UI 구조

TO-BE

- 알림톡을 제거하고 메일 중심의 알림 방식으로 통합, 단계별 이메일 UI 구현
- 메일 내 링크 클릭 시 middleware.ts에서 redirect URL을 파싱해 로그인 여부·권한 체크 후 제안서 상세 페이지로 바로 랜딩되도록 구현
- 제안서 진행 단계를 프로그레스바 UI로 시각화하여 피드백 UX 개선
- 제안서 상세 페이지를 CSR → SSR 구조로 전환해 초기 렌더링 성능 개선



이메일 UI 및 링크 버튼



제안서 상세 내역 UI/UX

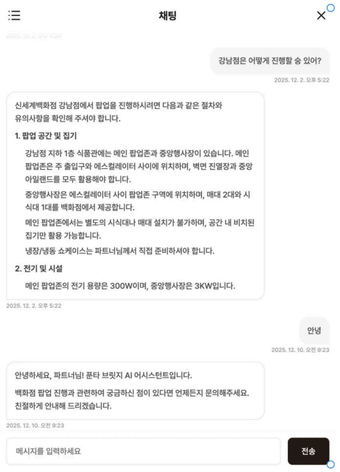
CS 챗봇 어드민 구현

AS-IS

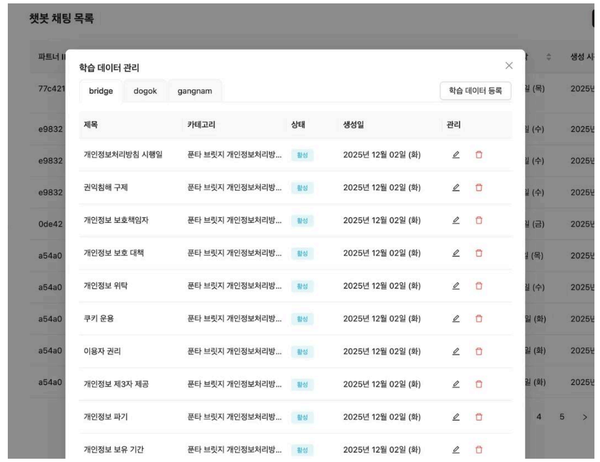
- 오프라인 팝업 행사 전 파트너사의 반복적인 문의로 CS 업무가 과도하게 발생
- 행사 진행 방식 및 점포별 특성을 수동으로 안내해야 하는 구조
- CS 대응을 자동화할 수 있는 수단 부재

TO-BE

- 행사 진행 방식과 점포별 특징을 안내하는 CS 챗봇 도입
- 챗봇 운영을 위한 어드민 화면 구현
- 채팅 내역을 세션 단위 리스트로 관리하여 사용 현황 확인
- React Hook Form 기반 학습 데이터 등록 폼 구현



CS 챗봇 UI/UX



CS 챗봇 어드민 UI/UX

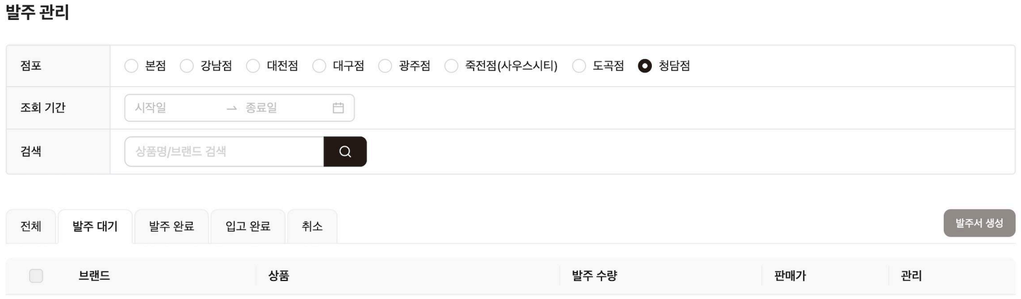
PDA 어드민 기능 구현

AS-IS

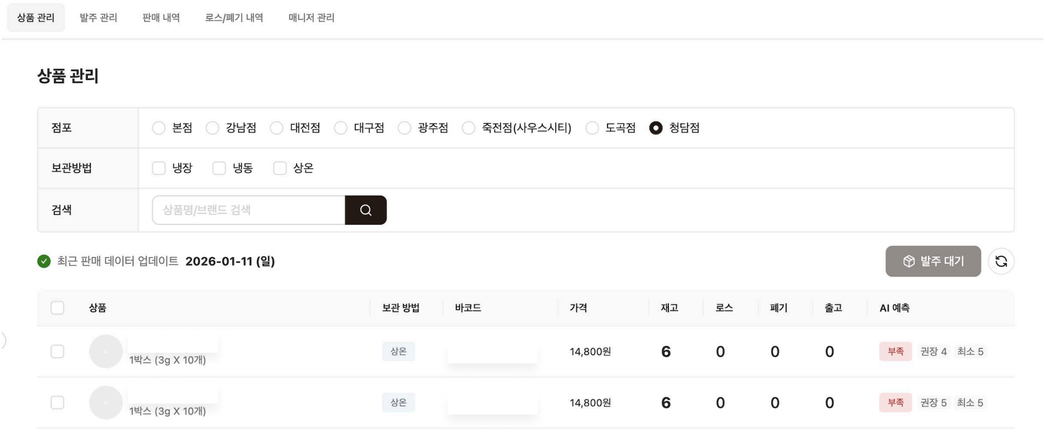
- 오프라인 현장에서 상품별 재고·로스·폐기·출고 수량을 즉시 확인할 수 없음
- 상품 판매 데이터와 발주 정보가 분산되어 있어 현장 운영 상황을 파악하기 어려움
- 발주 시 상품 리스트를 수동으로 정리해야 해 업무 시간이 오래 걸림

TO-BE

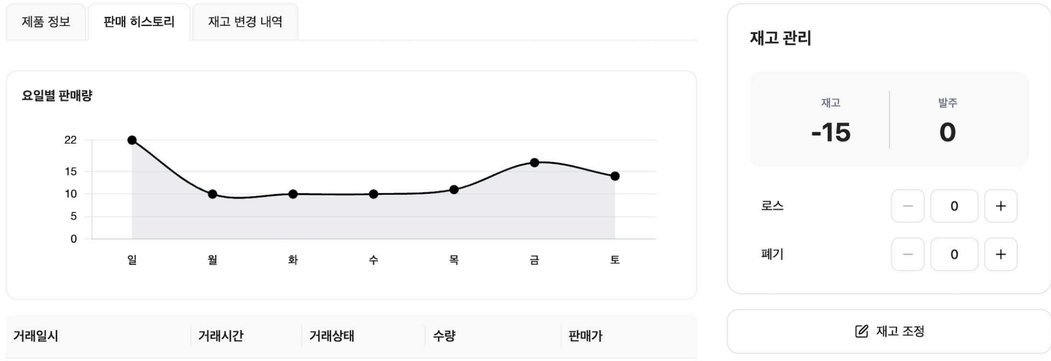
- 점포별 상품 판매 데이터와 발주 리스트를 통합해 확인할 수 있는 어드민 구현
- React Chart.js 기반 Line Graph를 활용해 상품별 판매 추이 시각화
- ExcelJS 기반 발주서 엑셀 템플릿 자동 생성
- 판매 데이터 기반 재고 수량 예측 AI 연동



발주 관리 UI/UX



상품 관리 페이지



상품 판매 데이터 시각화

모달 병렬 라우팅 트러블 슈팅

문제 상황 (Problem)

❗ **useState 기반 모달 상태 관리의 한계**

- 모달의 열림 여부를 useState로 관리하고 있었으며,
  모달 내부에서 상세 페이지로 이동했다가 다시 돌아왔을 때 이전 상태 그대로 모달이 유지되어야 하는 요구사항이 있었습니다.
- 그러나 라우트가 변경되면서 컴포넌트가 재마운트되었고, 그 결과 모달 상태가 유실되는 문제가 발생했습니다.

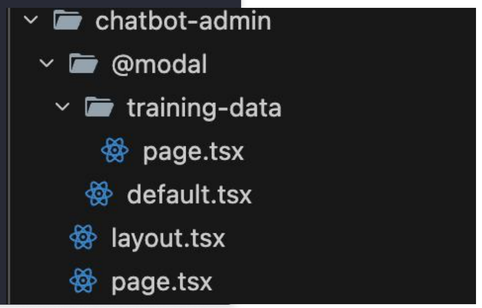
해결 과정 (Solution)

🔧 **병렬 라우팅(Parallel Routes)과 모달 슬롯(@modal) 적용**

- Next.js App Router의 병렬 라우팅을 활용하여
  모달 화면 자체를 별도의 URL 세그먼트로 분리했습니다.
- 메인 페이지는 그대로 유지한 채,
  특정 URL로 진입했을 때만 모달 슬롯(@modal)이 렌더링되도록 구성함으로써 뒤로가기 시에도 라우터가 상태를 자연스럽게 복원하도록 만들었습니다.
- 또한 모달이 없는 기본 상태를 명확히 하기 위해
  모달 슬롯의 기본 컴포넌트에서 null을 반환하도록 설정해 UI 상태를 조건 분기가 아닌 라우트 구조 자체로 표현했습니다.

📈 After 코드

```tsx
export default function Layout({
  children,
  modal,
}: {
  children: React.ReactNode;
  modal: React.ReactNode;
}) {
  return (
    <>
      {children}
      {modal}
    </>
  );
}
```



슬롯을 렌더링하는 레이아웃: src/app/admin/(main)/chatbot-admin/layout.tsx

```tsx
export default function Default() {
  return null;
}
```

모달 슬롯 기본값(null): src/app/admin/(main)/chatbot-admin/@modal/default.tsx

모달 병렬 라우팅 트러블 슈팅

결과 (Result)

- 모달 → 상세 페이지 뒤로가기 시 모달 상태 유지
- 동일한 라우트 구조로 데스크톱 환경에서는 모달, 모바일 환경에서는 바텀 시트로 렌더링하도록 분기처리
- @modal/default.tsx 와 같이 슬롯의 기본 상태를 명시적으로 정의함으로써, 모달의 유무를 useState 나 조건부 렌더링에 의존하지 않고 라우트 구조 자체로 표현
- 모달 컴포넌트에 전달해야 하는 props 수가 줄어들었고, 전체 UI 구조를 한눈에 파악할 수 있어 코드의 가독성과 유지보수성, 확장성이 함께 향상되었습니다.

배운 점 및 느낀점

- Next.js의 병렬 라우팅 기능을 처음 적용해보며, 모달 역시 단순한 오버레이 UI가 아니라 하나의 페이지처럼 라우팅을 통해 관리할 수 있다는 점이 인상 깊었습니다.

- 프레임워크가 제공하는 기능을 이해하고 적절히 활용하는 것이 복잡한 UI 상태 관리와 다양한 요구사항을 해결하는 데 큰 도움이 된다는 것을 배울 수 있었습니다.

📉 Before 코드

```
1    const [isModalOpen, setIsModalOpen] = useState(false);
2
3    return <Modal isOpen={isModalOpen} setOpen={setIsModalOpen} />;
```

📈 After 코드

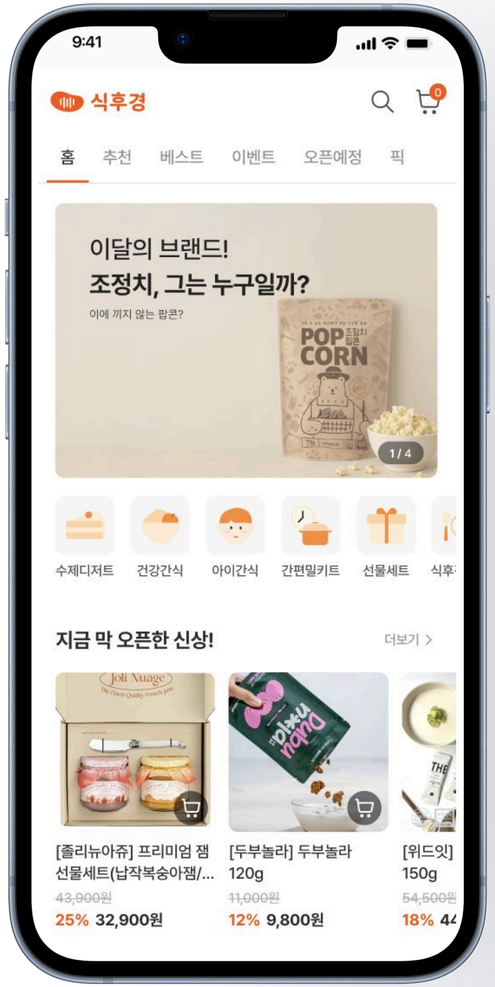
```
1  export default function TrainingDataModal() {
2    return <ChatbotTrainingData />;
3  }
```

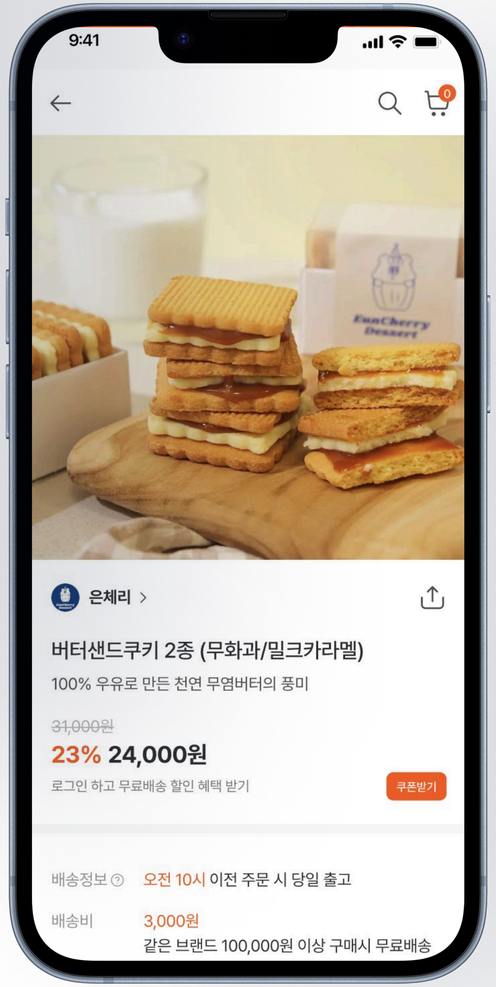
Frontend Developer

Projects

식후경

로컬 인디브랜드를 발굴하여 식품을 판매하는 B2C 커머스 서비스



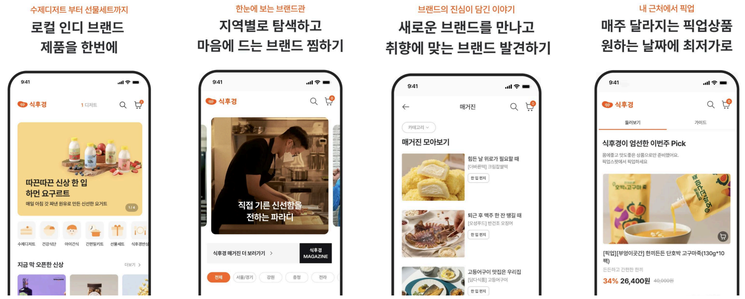


식후경 Mobile

2025.09 ~ 2025.10

개발 인원(5명) - 프론트엔드 2명 · 백엔드 2명· 디자인 1명

Typescript | Next.js | Vanilla Extract | Styled Components | Zustand | Tanstack Query | Yarn | GCP



Overview

식후경은 로컬 브랜드를 발굴해 식품을 판매하는 **B2C 커머스 서비스**입니다. 서비스 고도화 과정에서 **웹과 Android 개발을 함께 담당**했으며, 웹 영역에서는 Styled Components 기반 스타일 구조를 Vanilla Extract로 마이그레이션하고, Storybook을 활용해 공용 컴포넌트 문서화를 진행했습니다.

또한 모바일(Android) 환경에서는 Google Play Store 정책 변경에 대응해 SDK를 업데이트하고, 웹–앱 연동 구조를 점검하며 딥링크 및 결제 진입 이슈를 해결해 사용자 진입 흐름을 안정화했습니다.

Tasks

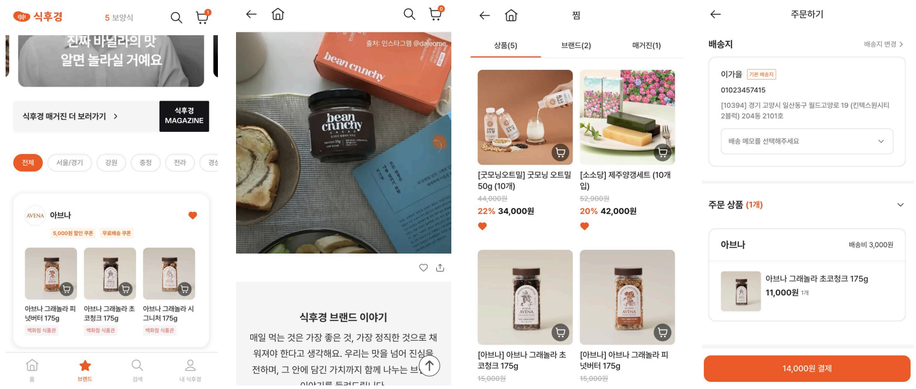
브랜드관·매거진·찜·결제 UI/UX 개편

AS-IS

- 쿠폰, 멀티 옵션, 결제 정책 등 커머스 기능 확장으로 UI/UX 복잡도 증가
- Styled Components 기반 CSS-in-JS 구조로 인해 초기 렌더링 과정에서 성능 저하와 UI 깜박거림 발생

TO-BE

- 브랜드관·매거진·찜·결제 영역 UI/UX 전면 개편
- 결제 영역에서 배송지, 마일리지·쿠폰 적용 UI 구현
- CSS-in-JS 문제를 해결하기 위해 Zero-runtime 기반 Vanilla Extract로 마이그레이션



브랜드관 페이지 | 매거진 페이지 | 찜 페이지 | 결제 페이지

디자인 시스템 및 공용 컴포넌트 문서화

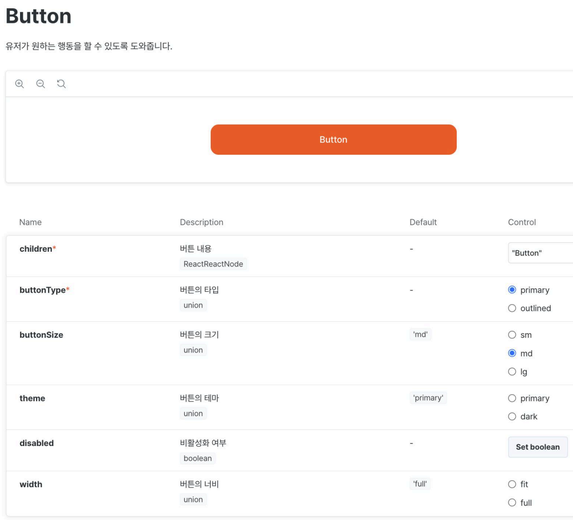
AS-IS

- 공용 컴포넌트에 대한 문서 부재로 props별 UI 동작을 코드로 직접 확인해야 하는 구조
- 디자이너와의 협업 시 컴포넌트 상태 공유가 어려움
- 신규 기능 개발 시 디자인과 구현 간 일치 여부를 검증하기 어려움

TO-BE

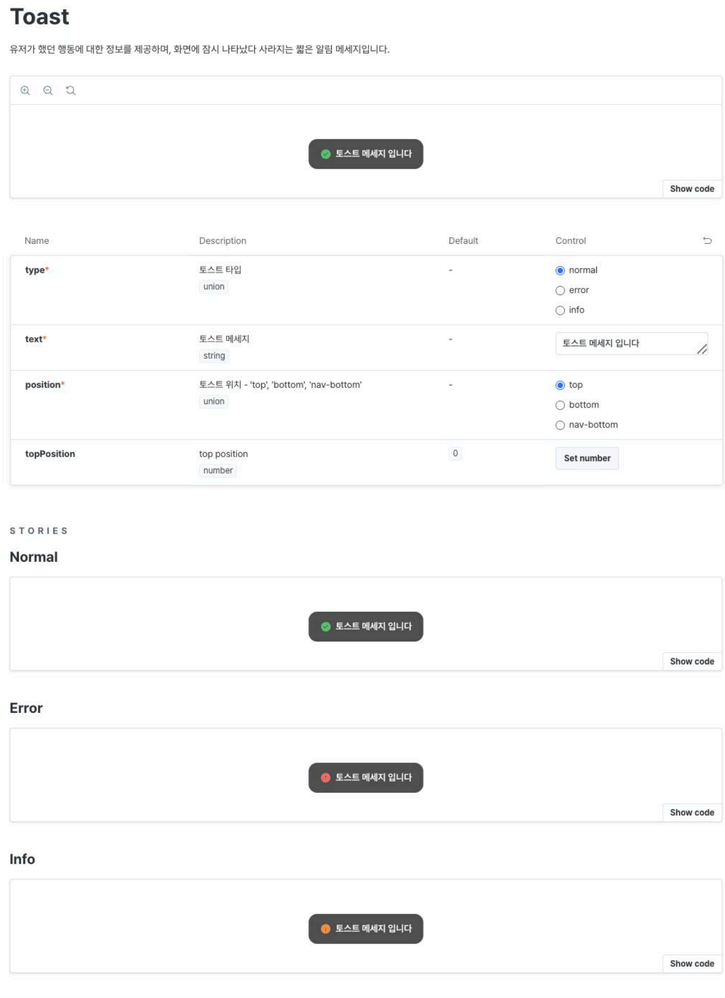
- Storybook 도입을 통해 컴포넌트 단위 문서화 환경 구축
- 재사용 빈도가 높은 공용 컴포넌트 15개를 선정해 구조 정리

버튼 디자인 시스템



버튼 Storybook 문서화

토스트 디자인 시스템



토스트 Storybook 문서화

Query Key 관리 구조 개선 트러블 슈팅

문제 상황 (Problem)

❗ Query Key 관리 구조의 한계

- Query Key의 최상위 키만 상수로 관리
- 같은 도메인 데이터임에도 서로 다른 키 네이밍 사용
- id, filter 등의 의존성이 각 쿼리마다 제각각 선언됨
- 계층 구조가 없어 prefix 기반 invalidate / refetch 어려움

📉 Before 코드

```
export const QUERY_KEY = {
  reviewList: 'review-list',
  reviewPhotos: 'review-photos',
  reviewPreview: 'review-preview',
};
```

```
useInfiniteQuery({
  queryKey: [QUERY_KEY.reviewList, productId, hasImageReviewOnly],
});
```

해결 과정 (Solution)

🔧 Query Key Factory 도입

- Query Key를 도메인 단위로 계층화
- Generic → Specific 구조로 키를 생성
- id 외 조건 값은 객체로 묶어 결정론적 해싱 보장
- 기능별 query.ts 파일로 Query Key 관리

```
-src
  - features
    - Review
      - api
      - components
      - hooks
      - constants
      - query.ts
```

각 기능 폴더 내에 query.ts 파일을 위치시킵니다.

📈 After 코드 (Query Key 정의)

```
export const reviewQuery = createQueryKeys('review', {
  product: (productId: string, hasImageReviewOnly: boolean) => [
    'product',
    productId,
    { hasImageReviewOnly },
  ],
  list: {
    photos: (productId: string) => ['photos', productId],
  },
});

```

Query Key 관리 구조 개선 트러블 슈팅

결과 (Result)

- Query Key가 도메인 중심으로 개별적으로 관리됨
- prefix 기반 캐시 무효화 가능

```
queryClient.invalidateQueries({
  queryKey: reviewQuery.list.all(),
});

```

- 같은 리소스를 여러 키로 관리하던 문제 해소
- Query Key 변경 시 query.ts만 수정하면 되도록 구조 개선

📈 After 코드

```
useInfiniteQuery({
  queryKey: reviewQuery.product(productId, hasImageReviewOnly),
});
```

배운 점 및 느낀점

- Query Key에 대한 문서화와 구조화가 이루어지지 않아 캐시 관리가 어려운 상황이었지만, Query Key Factory를 구현하고 Query Key를 전역 상수가 아닌 기능 단위로 계층화하면서 현재 사용 중인 Query Key의 구조와 역할을 명확히 파악할 수 있게 되었습니다.

- 필터나 파라미터를 배열 형태로 관리하면서 동일한 의미의 쿼리임에도 서로 다른 키로 인식되어 invalidate가 정상적으로 동작하지 않는 문제가 있었는데, Query Key Factory에서 조건 값을 객체로 관리하여 결정론적인 해싱을 보장함으로써 캐시 무효화가 안정적으로 동작하도록 개선할 수 있었습니다.

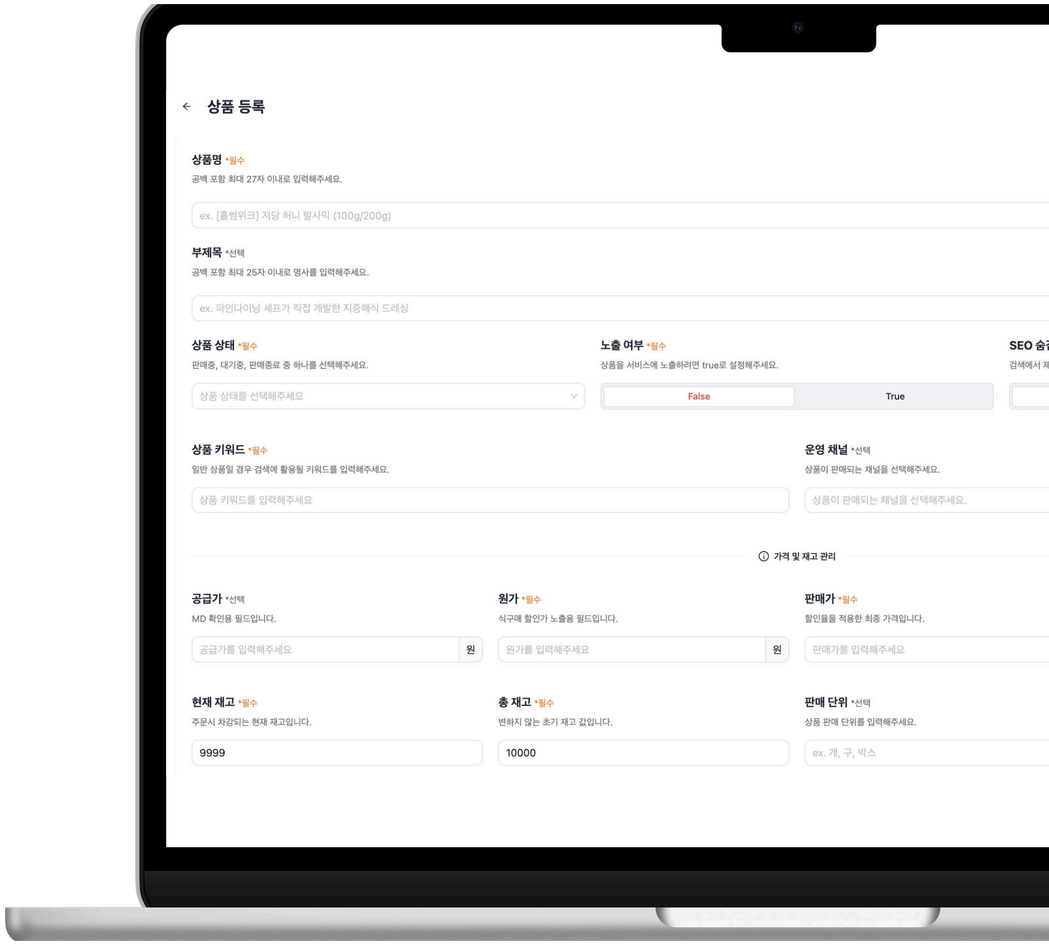
- Query Key 구조를 도입하게 된 배경과 설계 의도, 사용 규칙을 문서화하고 공유함으로써, 팀 내에서 Query Key를 일관된 기준으로 작성하고 관리할 수 있는 컨벤션을 정립할 수 있었습니다.

Frontend Developer

Projects

식후경 어드민

식후경 커머스 운영 기능을 관리하는 사내 백오피스



식후경 어드민 Desktop

2025.10 ~ 2025.12

개발 인원(3명) - 프론트엔드 1명 · 백엔드 2명

Typescript | React.js | TailwindCSS | Ant Design | Zustand | Tanstak Query | Yarn | GCP

Overview

식후경 어드민은 상품, 쿠폰, 브랜드, 파트너, 진열, 딥링크, 알림 등 커머스 전반의 운영 기능을 관리하는 **사내 백오피스 서비스**입니다.

기존에는 CMS(Strapi)를 활용해 운영 기능을 관리하고 있었으나,
다대다 관계 중심의 데이터 모델링, 단순 CRUD에 한정된 API 구조로 인해
커머스 도메인 확장과 복잡한 운영 로직을 처리하는 데 한계가 존재했습니다.
이에 CMS 중심 구조에서 벗어나, NestJS·MongoDB 기반의 서비스 중심 백엔드로 마이그레이션하며 운영 흐름에 맞는 어드민 기능을 새롭게 **기획·개발**했습니다.
사내 인터뷰를 통해 기존 CMS 사용 과정에서의 불편 요소를 정리하고, 운영 생산성과 확장성을 높일 수 있는 기능을 재설계하는 데 집중했으며, 기능 구현 이후에는 **운영 가이드 문서를 작성**하고 **사내 교육을 실시**해 운영팀이 실제 업무에 원활히 활용할 수 있도록 지원했습니다.

| 본 프로젝트에서는 대외 공개가 가능한 범위 내의 화면을 중심으로 소개합니다.

Tasks

테스트 환경 분리 및 dev 배포 CD 구현

AS-IS

- 로컬 환경에서만 기능을 검증한 뒤 바로 운영 서버에 배포해야 하는 구조로 배포 후 운영 환경에 즉시 영향을 줄 수 있는 리스크 존재
- 서버 요청 주소를 코드에서 수동으로 설정하고 있어 Axios 인스턴스별 분기 처리로 인한 복잡도
- 단일 env 파일로 모든 환경을 관리하고 있어 local / test / production 환경 간 구분이 명확하지 않음
- 테스트용 서버가 없어 운영 배포 전 동일한 환경에서의 사전 검증이 불가능한 상태

TO-BE

- local / development / production 환경으로 env 구조 분리
- 호출 대상 서버에 따라 Axios 인스턴스 분리
- 테스트 환경과 운영 환경을 명확히 분리해 운영 서버에 직접 배포하는 리스크 제거
- Github Actions를 활용한 CD 파이프라인 구축
  - dev 브랜치 푸시 시 테스트 서버에 자동 배포
  - GCP + Caddy 기반 배포 환경 구성
  - React dist 파일 서빙 구조

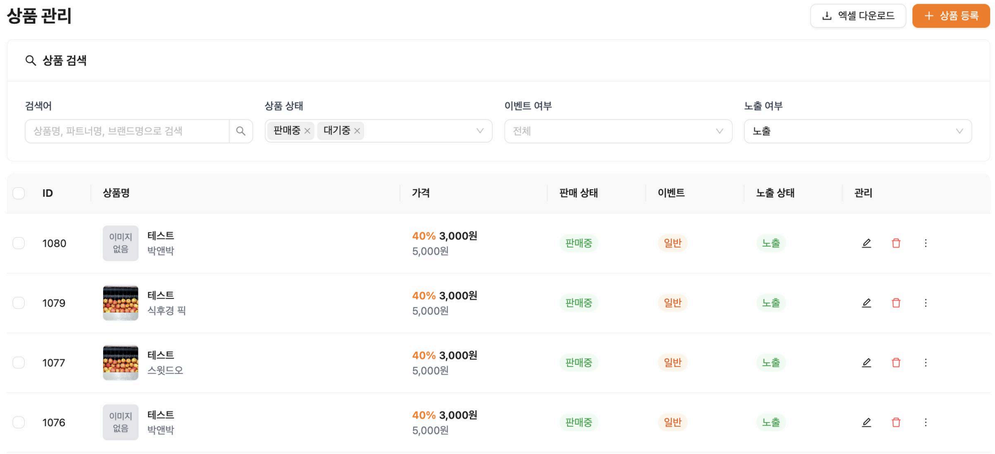
CMS(Strapi) 기능 이전 및 데이터 구조 재설계

AS-IS

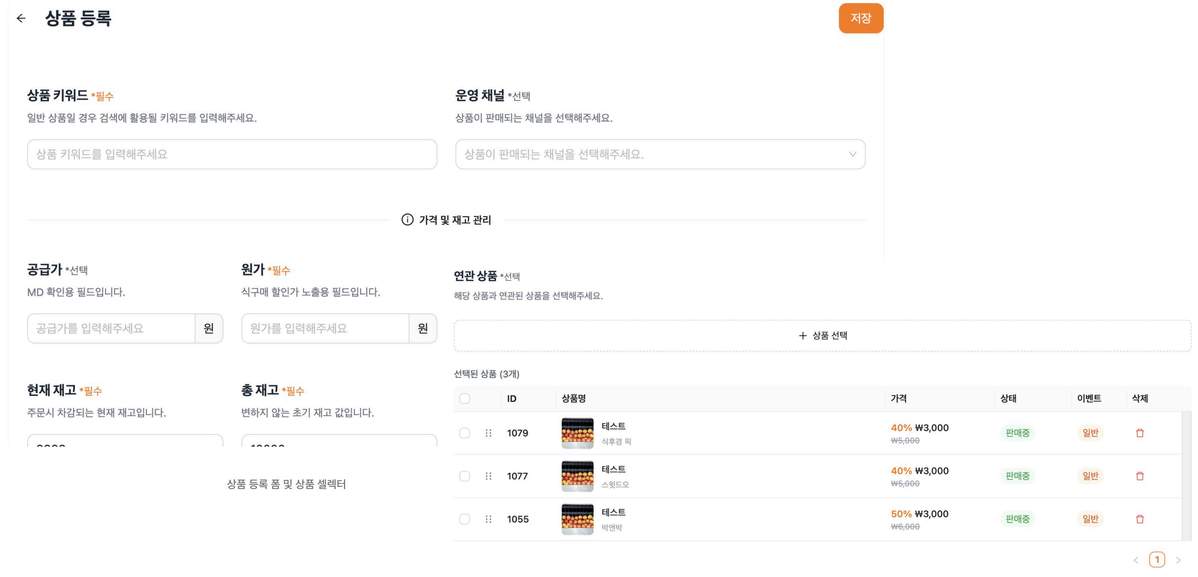
- Strapi의 다대다 관계 구조로 인해 모든 도메인에 중간 Link 테이블이 생성되고,실제로 사용하지 않는 필드값(order 등)까지 포함되어 데이터 구조가 불필요하게 복잡해진 상태
- 관계가 늘어날수록 Join 연산이 증가해 조회 성능 저하 및 데이터 관리 어려움
- 운영 로그를 확인할 수 없어 데이터 변경 이력 및 운영 과정 추적이 어려움
- 부서·담당자별로 서로 다른 양식이 사용되며 데이터 응집화가 되지 않는 문제 발생
- 결과적으로 운영 효율이 낮고, 반복 작업과 휴먼 에러가 빈번하게 발생하는 구조

TO-BE

- **상품·쿠폰·파트너·브랜드 관리 기능 구현**
- 도메인 간 연관관계를 양방향으로 설정할 수 있는 Selector 구현
- 기존 등록 데이터를 그대로 활용할 수 있도록 복제(Duplicate) 기능 구현
  → 반복적으로 입력하던 정보를 재사용할 수 있어 운영 시간 단축
- 운영 감사 및 변경 이력 추적을 위한 히스토리 기능 구현
- 데이터 응집화를 위한 엑셀 다운로드 기능 구현
  - 상품, 파트너, 가격 정보를 일관된 템플릿으로 다운로드 가능
  - 수기 입력에 의존하던 운영 문서 작업 제거
  - 부서·담당자별로 상이하던 문서 양식을 통합
- Ant Design Form을 활용한 폼 구현



상품 관리 UI/UX



상품 등록 폼 및 상품 셀렉터

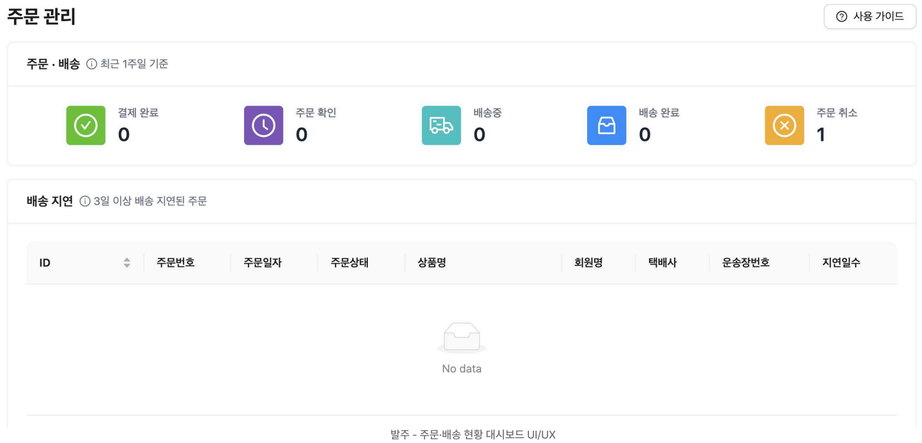
운영 중심 기능 구현

AS-IS

- CMS에서 콘텐츠 생성 시 불필요한 필드가 과도하게 많아 운영 복잡도가 높음
- 에디터 내 콘텐츠 미리보기 기능이 제공되지 않아 등록 결과를 즉시 확인하기 어려움
- 발주 기능이 구 어드민에 남아 있어 불필요한 서버 비용이 지속적으로 발생
- 진열 순서 및 노출 여부 변경 시 개발자 개입 또는 코드 수정이 필요한 구조

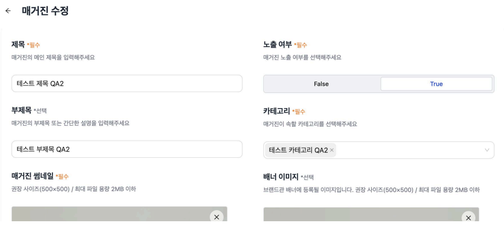
TO-BE

- **매거진 관리 기능을 백오피스로 이전**
  - 매거진–상품–브랜드 관계를 한 화면에서 설정 가능하도록 UI 재설계
- **발주 관리 기능을 백오피스로 이전**
  - 주문·배송 현황을 한눈에 확인할 수 있는 대시보드 UI 구현
  - 엑셀 기반 운송장 일괄 등록 기능 구현
  - 업로드 시 데이터 유효성 검증 로직 추가로 발주 정확성 개선
  - 구 어드민 서버 제거를 통해 불필요한 서버 비용 절감
- **GNB 및 이벤트 진열 관리 기능 구현**
  - dnd-kit을 활용한 드래그 앤 드롭 기반 진열 순서 관리
  - 진열 순서 및 노출 여부를 운영 화면에서 직접 제어



발주 - 주문·배송 현황 대시보드 UI/UX

발주 - 운송장 일괄 등록 기능



매거진 작성 폼 UI/UX

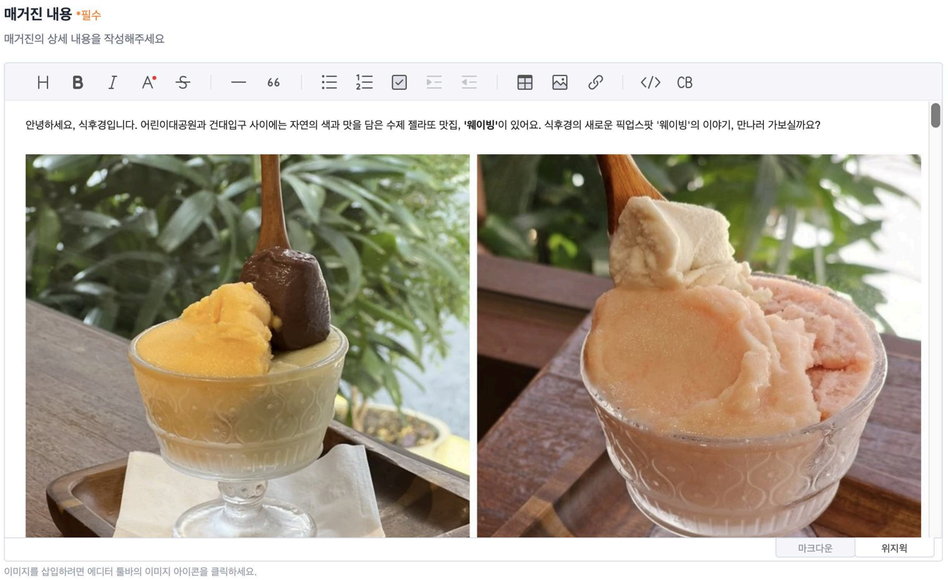
드래그 앤 드랍으로 GNB 진열 관리

에디터 초기화 시 자동 스크롤 트러블 슈팅

문제 상황 (Problem)

❗에디터 초기화 시 화면이 강제로 이동하는 현상

- 서버에서 받아온 기존 콘텐츠를 에디터에 주입하는 과정에서 화면이 에디터가 위치한 지점으로 자동 스크롤되는 현상 발생
- 폼 화면 상단에서 사용자가 입력하던 중에도 화면이 갑자기 이동하며, 입력 흐름이 끊기는 문제 발생
- Toast UI Editor(WYSIWYG)가 초기화 또는 콘텐츠 주입 과정에서 내부적으로 focus() 또는 scrollIntoView() 를 호출



해결 과정 (Solution)

🔧전역 스크롤 트리거 임시 차단

- 에디터가 초기화되는 동안 자동 스크롤을 무력화하기 위해, Element.prototype.scrollIntoView와 HTMLElement.prototype.focus를 빈 함수로 덮어써서 자동 스크롤을 차단
- 스크롤 잠금 상태에서만 에디터가 렌더링되도록 setTimeout을 사용해 초기 렌더링 타이밍 지연 및 제어

🔧에디터 로드 / 콘텐츠 주입 직후 스크롤 복원

- 에디터 로드 시점의 scrollY 값을 저장한 뒤, 로드 직후 window.scrollTo(0, scrollY) 를 호출해 사용자가 보고 있던 스크롤 위치로 화면 복원
- 초기 포커스로 인한 추가 스크롤 영향을 제거하기 위해 에디터 인스턴스의 current.blur() 호출

결과 및 배운점

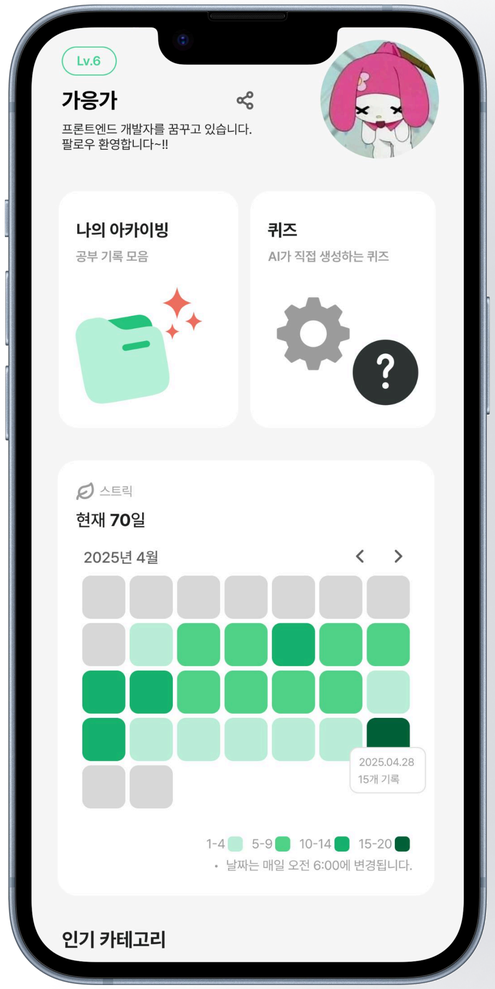
에디터 초기화 및 콘텐츠 주입 과정에서 발생하던 자동 스크롤 현상을 제거해, 사용자가 보고 있던 스크롤 위치가 안정적으로 유지되도록 개선했습니다. 이 경험을 계기로 DOM 포커스와 포커스 제어 시 초기화 타이밍을 함께 고려해야 한다는 점을 깨달았습니다.

Frontend Developer

Projects

StudyLog

학습 내용 기반으로 AI가 퀴즈를 생성해주는 복습 서비스



StudyLog Mobile

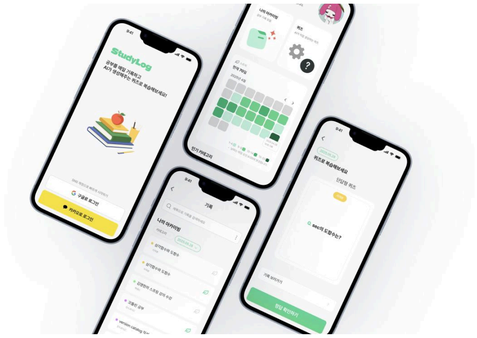
2025.10 ~ 2025.12

개발 인원(3명) - 프론트엔드 1명 · 백엔드 2명

Typescript | React.js | TailwindCSS | Storybook | Zustand | Tanstak Query | Pnpm | Vercel

공부한 내용을 기록하고 AI가 퀴즈를 생성해주는

StudyLog



Overview

StudyLog는 학습 내용을 텍스트와 파일로 기록하면, 기록 데이터를 바탕으로 AI가 난이도와 문항 수를 조절해 단답형·OX 퀴즈를 생성하는 **복습용 웹앱 서비스**입니다. 사용자는 학습 기록을 통해 복습 퀴즈를 자동 생성하고, 기록량과 연속 학습일을 스트릭으로 확인할 수 있으며, 팔로잉·랭킹 시스템을 통해 학습 동기를 유지할 수 있도록 설계했습니다.

본 프로젝트는 졸업 프로젝트로 진행되었으며, **서비스 기획, UI/UX 디자인, 프론트엔드 개발을 전담**했습니다.

Tasks

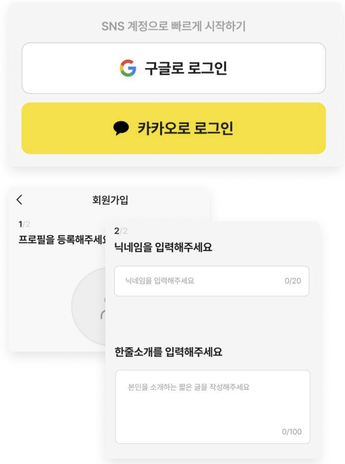
OAuth 2.0 + JWT 기반 소셜 로그인 구현

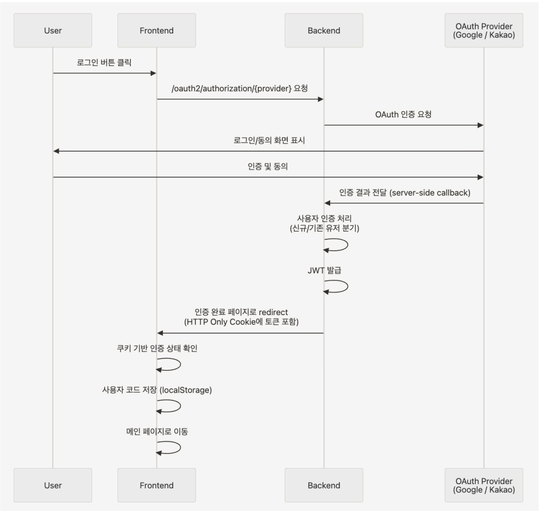
AS-IS

- 간편 로그인 도입이 필요했으나, OAuth 인증 흐름을 프론트에서 직접 처리할 경우 인가 코드 처리 및 토큰 관리 복잡도가 높음

TO-BE

- 구글·카카오 OAuth 인증과 토큰 발급을 백엔드에서 일괄 처리하고, 인증 완료 후 프론트로 페이지 리다이렉트되는 구조로 설계
- 토큰은 HTTP Only Cookie로 전달해 보안성을 확보
- 프론트엔드는 인증 결과에 따라 사용자 코드 저장 및 화면 전환만 담당





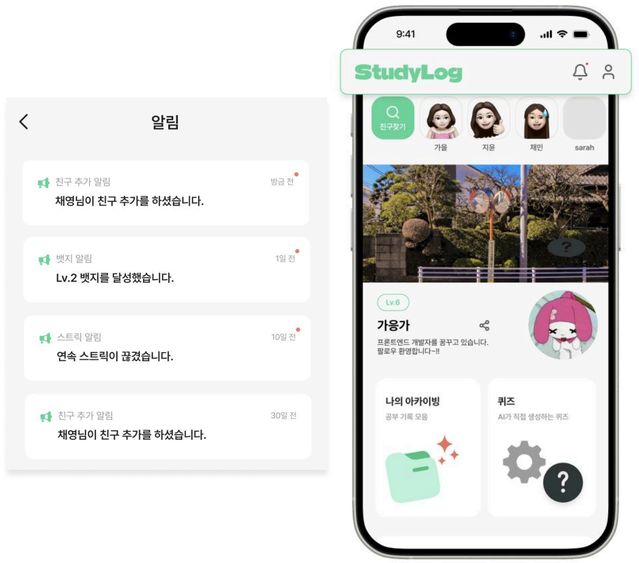
SSE 실시간 알림 구현

AS-IS

- 기존의 목록 조회 방식으로는 사용자가 새 알림을 인지하기 어려움
- 실시간 알림을 구현하기 위해 WebSocket 방식도 고려했으나, 양방향 통신이 불필요한 알림 특성상 구현 복잡도가 높다고 판단

TO-BE

- 서버 → 클라이언트 단방향 이벤트 전달에 적합한 SSE(Server-Sent Events) 방식 채택
- EventSource를 통해 알림 이벤트를 실시간 수신
- 수신 이벤트를 React Query 캐시와 연동해 알림 뱃지 및 리스트가 즉시 갱신되도록 구현



단방향 통신인 SSE로
실시간 알림 제공

팔로우·언팔로우·레벨업 이벤트 발생 시 수신

PDF 기반 학습 기록 자동화

AS-IS

- 사용자가 학습 내용을 직접 타이핑해서 기록하는 과정이 번거롭고 시간이 오래 걸림
- 많은 학습 자료가 PDF 형태지만, 이를 앱 내 데이터로 변환하려면 사용자가 일일이 복사/붙여넣기하거나 수동으로 요약해야 했음

TO-BE

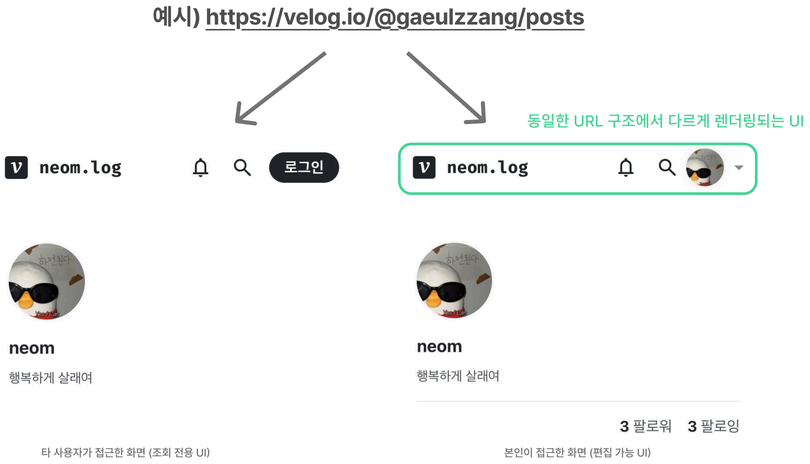
- react-pdftotext 라이브러리를 활용해 클라이언트에서 PDF 텍스트를 즉시 추출하고, 별도 백엔드 업로드 과정 없이 브라우저에서 처리해 속도를 높임
- 추출된 텍스트를 OpenAI API(gpt-4o-mini)에 전송해 핵심 제목과 요약(400자 이내)을 JSON 포맷으로 자동 생성
- 결과값(제목/요약)을 다음 단계 입력 폼에 자동으로 채워넣어, 사용자는 내용 검토/수정만 하면 되는 플로우 구현

단일 URL 구조 기반의 사용자 식별 트러블 슈팅

문제 상황 (Problem)

❗ 동일한 URL 구조에서 사용자 식별 기준 부재

- /:code 형식의 URL 구조 사용 환경
  ex) https://web.studylog.shop/UX320
- 동일한 URL 구조 내에서 본인 페이지와 타 사용자 페이지를 구분해 서로 다른 화면을 렌더링 해야 함
- URL만으로 페이지의 주체를 명확히 식별하기 어려워 외부 유입 및 페이지 공유 시 본인 여부를 라우팅 레벨에서 판단하기 어려움



해결 과정 (Solution)

🔧 고유 코드를 로컬 스토리지에 저장 후 분기 처리

- 로그인 시 서버로부터 전달받은 사용자 고유 코드를 로컬 스토리지에 저장
- 현재 URL에 포함된 사용자 코드와 로컬 스토리지에 저장된 사용자 코드 비교 후 분기 처리

```
const CodePage = () => {
  const { code } = useParams(); // URL의 code
  const myCode = localStorage.getItem(storageKey.USER_CODE); // 로그인한 유저 code

  // 동일한 라우트 진입 시, 코드 일치 여부로 본인 / 조회 페이지 렌더링
  return code === myCode ? <MainPage /> : <OtherUserPage />;
};
```

🔧 고유 유저 코드 페이지가 메인이 되도록 라우팅 설정

- 로그인 시 고유 코드와 함께 로그인 여부를 로컬 스토리지에 저장
- 로그인 상태면 홈 없이 바로 메인이 렌더링 되도록 커스텀 훅 구현

```
const useAuthRender = () => {
  useEffect(() => {
    // 로그인 상태면 홈 화면 노출 없이 바로 메인으로 이동
    if (isLoggedIn && userCode) {
      goMainPage(userCode);
    } else {
      setShouldRender(true);
    }
  }, [goMainPage]);
};
```

단일 URL 구조 기반의 사용자 식별 트러블 슈팅

결과 (Result)

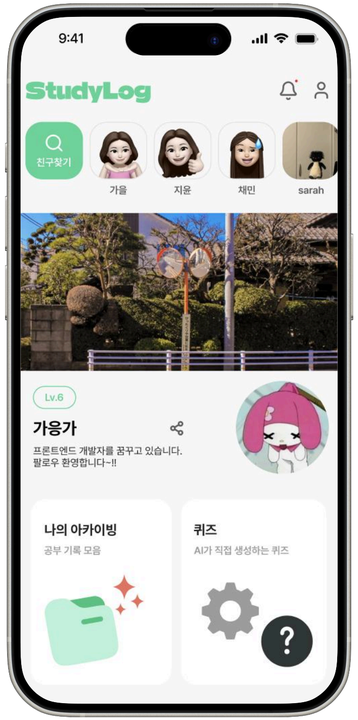
- 동일한 URL 구조를 유지하면서 사용자를 식별하여 분기 처리
  - 본인 페이지인 경우 편집 가능한 메인 페이지 노출
  - 타 사용자 페이지인 경우 조회 전용 페이지 노출

배운 점 및 느낀점

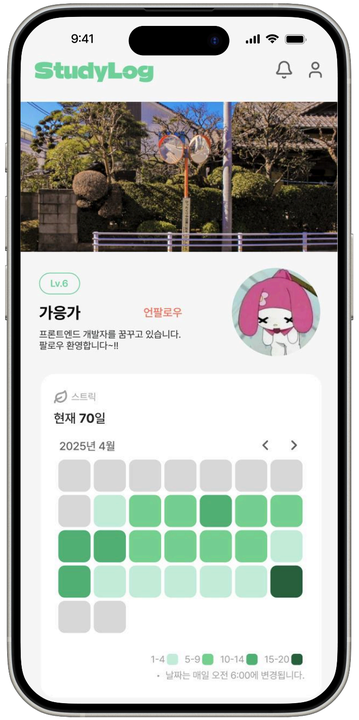
- 프론트엔드 개발자는 단순히 화면을 구현하는 역할에 그치지 않고, URL 구조와 라우팅 방식을 어떻게 설계하느냐에 따라 사용자 흐름을 결정한다는 점을 체감했습니다.

- 동일한 화면이라도 URL에 어떤 정보가 담겨 있는지에 따라 권한별로 접근 가능한 영역을 구분할 수 있었고, 이러한 설계가 페이지 공유나 외부 유입 상황에서도 중요한 역할을 한다는 것을 알게 되었습니다.

본인 페이지에서는
팔로워 목록이 상단에 노출되며
개인 기록 열람이 가능함



조회 전용 페이지에서는
스트릭과 인기 카테고리만 확인 가능하며
팔로우 기능을 통해 소셜 연결 유도



프론트엔드 개발 이외에도
여러 사이드 프로젝트를 진행하며
다양한 도메인의 역량을 쌓아왔습니다.

ScheTook

2024.11 ~ 2025.06

개발 인원(15명) - 기획 2명 · 디자인 3명
AOS 4명 · iOS 4명 · 백엔드 2명

Kotlin | Android Studio | Jetpack Compose | MVI | Clean Architecture | Multi Module | Google Play Store

ScheTook은 다인원 일정 조율을 위한 앱으로, 그룹장이 그룹을 생성해 약속을 만들면 구성원들의 투표를 바탕으로 참여 가능한 인원이 많은 순서와 가장 빠른 시간을 기준으로 약속 시간을 추천해 주는 서비스입니다.

안드로이드 파트 리더로 활동하며 서비스 기획부터 프론트엔드 개발 전반을 전담했습니다. 그룹 UI, 약속 추천 UI, 타임테이블 화면을 구현했으며, Single Activity 기반의 Jetpack Compose 구조에서 MVI 패턴과 Clean Architecture, 멀티모듈 구조를 적용해 확장성과 유지보수성을 고려한 앱을 설계했습니다.

또한 Github Actions를 활용해 CI 환경을 구축하고 Ktlint 검사를 적용했으며, Slack·Discord Webhook 연동을 통해 코드 변경 사항이 즉각적으로 공유되도록 개선했습니다. QA 과정에도 직접 참여해 서비스 완성도를 점검했고, 최종적으로 앱 출시까지 진행했습니다.

스케줄을 툭 던지면 모두가 가능한 시간을 찾아주는

ScheTook

가능 시간을 간편하게 수집

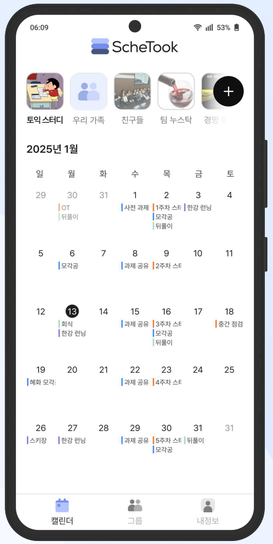
색의 진하기를 통해
그룹원의 가능 시간대를
쉽게 파악할 수 있어요

Best 시간대 추천

모두가 가능한 시간을
자동으로 분석하고,
최적의 시간대를 추천해줘요

한 눈에 보는 캘린더

캘린더를 통해
여러 그룹의 약속 일정을
한 곳에서 모아 볼 수 있어요



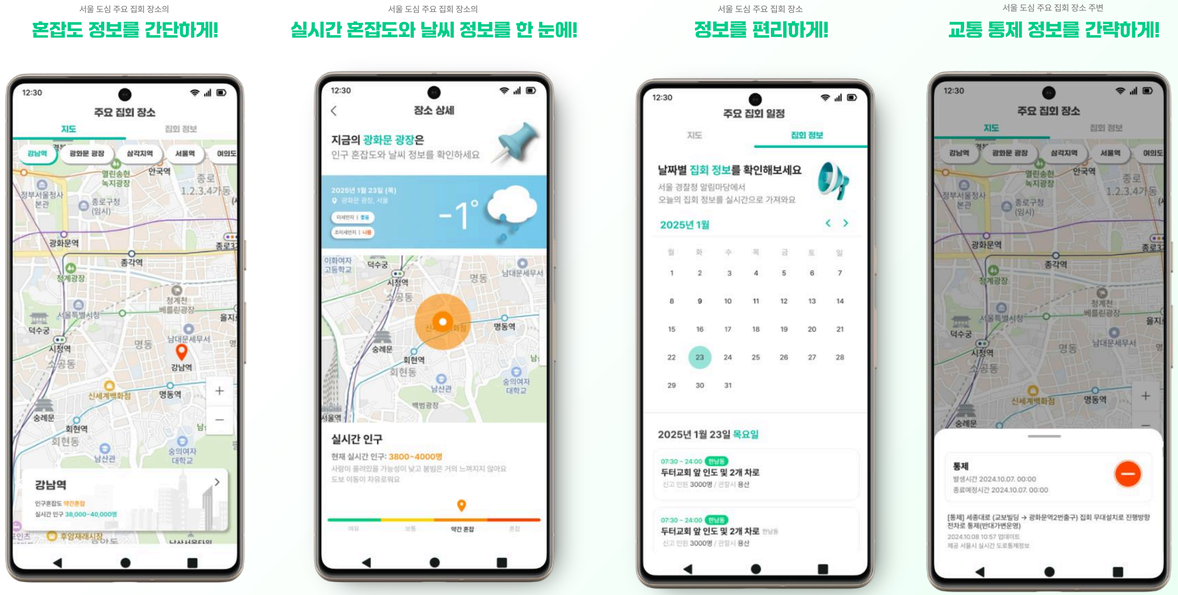
혼잡제로

2025.01 ~ 2025.02
개발 인원(8명) - 기획 1명 · AOS 4명
백엔드 2명 · 데이터 1명

Kotlin | Android Studio | Jetpack Compose | MVI | Clean Architecture | Multi Module | Google Play Store

혼잡제로는 주요 집회 장소인 강남역, 광화문 광장, 삼각지역, 서울역, 여의도역의 실시간 정보를 제공하는 안드로이드 앱으로, 서울 공공데이터를 기반으로 한 실시간 도로 상황과 혼잡도 정보를 지도 위에 시각화해 보여주는 서비스입니다.
상세 화면에서는 기온과 미세먼지 등 날씨 정보, 실시간 인구 수, 혼잡도 데이터를 함께 확인할 수 있으며, 서울경찰청 웹사이트에서 수집한 집회 정보를 제공해 현장 상황을 종합적으로 파악할 수 있도록 설계했습니다.

안드로이드 파트 리더로 참여해 **서비스 기획**과 **UI/UX 디자인**을 담당했으며, 상세 페이지 제작과 Naver Map API를 활용해 도로 정보를 표시했습니다. 특히 실시간 혼잡도 정보를 색상별 마커로 표현해, 사용자가 한눈에 혼잡도를 파악할 수 있도록 구현하며 최종적으로 앱 출시까지 진행했습니다.



Plog

2024.04 ~ 2024.10

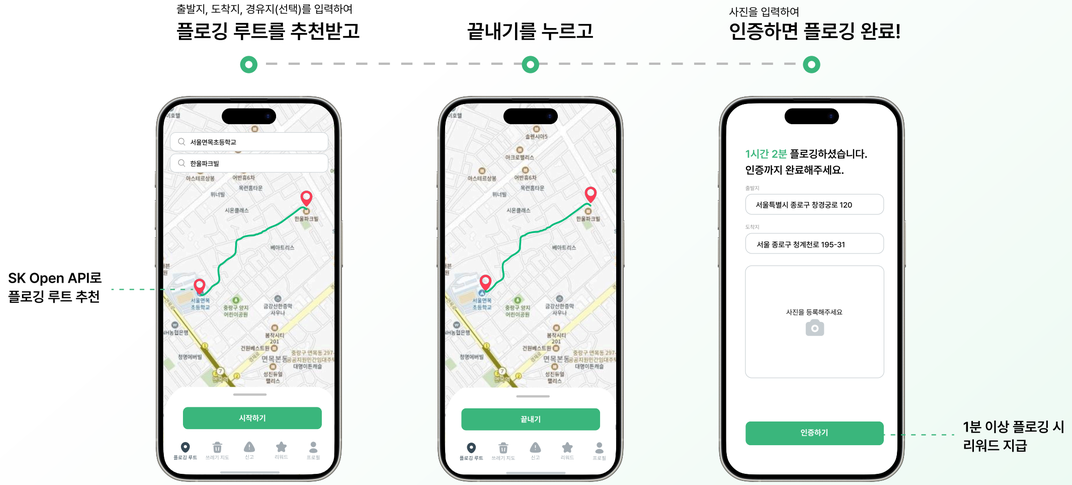
개발 인원(4명) - AOS 2명 · 백엔드 2명

Kotlin | Android Studio | Jetpack Compose | MVVM | Clean Architecture

Plog는 쓰레기를 주우며 걷는 플로깅 활동을 장려해 일상 속 환경 보호 참여를 유도하는 안드로이드 앱입니다.
사용자는 공공데이터포털에서 수집한 전국 쓰레기 투기 지역 정보를 '쓰레기 지도' 형태로 확인할 수 있으며, 쓰레기가 많이 쌓인 장소를 직접 신고하거나 청소 상태를 기록해 지역 환경 개선에 기여할 수 있습니다. 또한 리워드 시스템을 도입해 플로깅 활동이 지속적인 참여로 이어질 수 있도록 설계했습니다.

안드로이드 개발자로 참여해 **서비스 기획과 UI/UX 디자인**, 핵심 기능 구현을 담당했습니다. 자동 로그인과 소셜 로그인(카카오·네이버) 기능을 구현했으며, 네이버 지도 API를 활용해 쓰레기 위치를 마커와 캡션으로 시각화하고, 주소 검색 및 내비게이션 기능을 통해 실제 플로깅 루트를 표시했습니다. 또한 최근 검색어 저장·삭제 기능을 Room 라이브러리로 구현해 사용성을 높였습니다.

K-PaaS 공모전에서 금상(과학기술정보통신부장관상)을 수상하며 환경 문제를 사용자 참여형 서비스로 풀어낸 성과를 인정받았습니다.



알뜰밥상 in 서울

2024.11

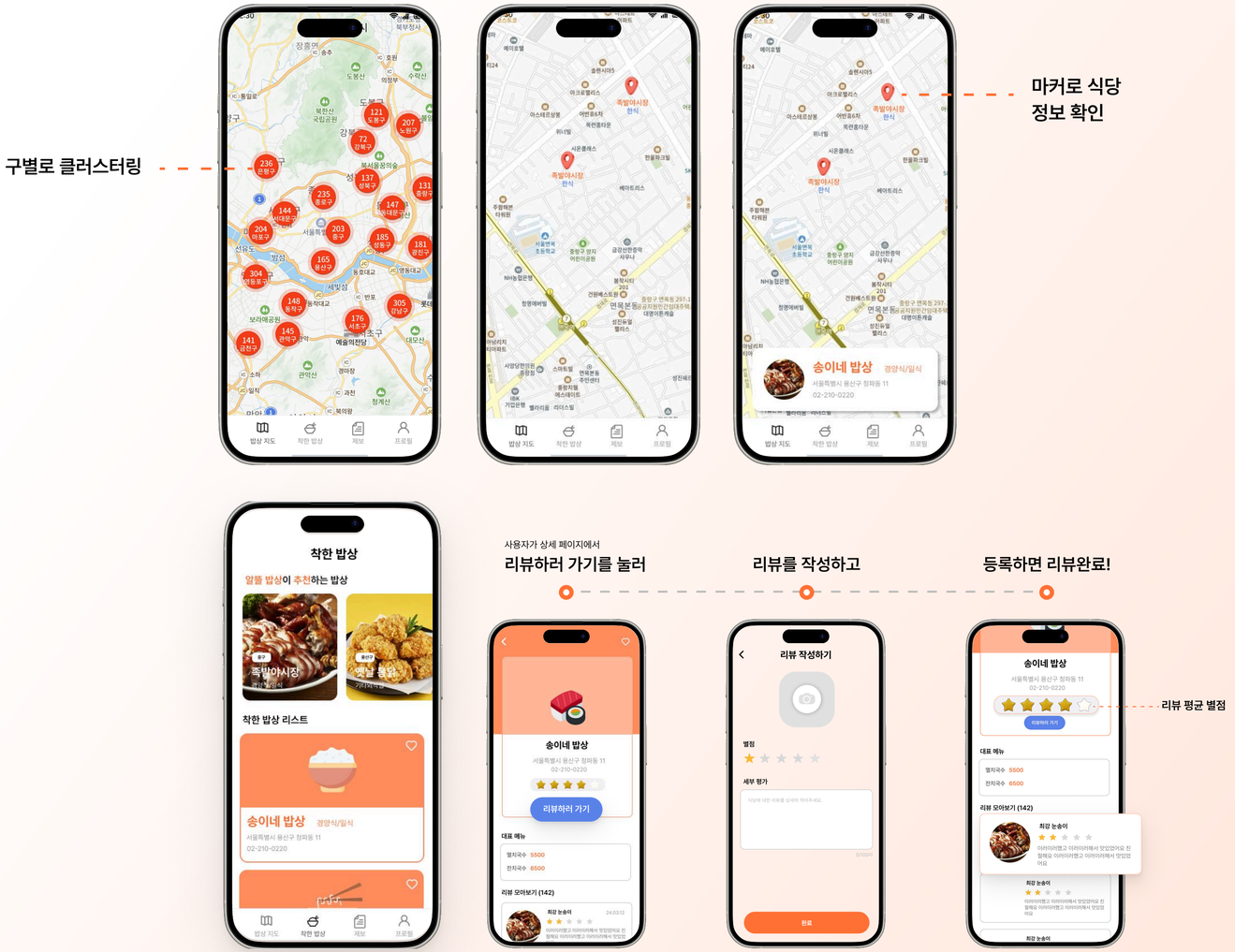
개발 인원(6명) - AOS 2명 · 백엔드 2명
데이터 2명

Kotlin | Android Studio | Jetpack Compose | MVVM | Clean Architecture

알뜰밥상 in 서울은 서울시 착한가격업소를 구별로 묶어 지도에서 한눈에 확인할 수 있도록 한 안드로이드 앱입니다.
사용자는 지도를 통해 착한가격업소를 탐색할 수 있으며, 상세 페이지에서는 각 업소의 대표 메뉴와 가격, 리뷰 정보를 확인할 수 있습니다. 또한 네이버 리뷰 데이터를 기반으로 한 감정 분석을 통해 별점을 제공하고, 사용자의 카테고리 선택, 지역구, 좋아요 기록을 반영해 맞춤 식당을 추천함으로써 탐색 부담을 줄였습니다.

팀 리더로 참여해 **서비스 기획**과 **UI/UX 디자인**, **안드로이드 개발** 전반을 담당했습니다. 지도 중심의 사용자 흐름을 설계하며 네이버 지도를 활용해 구별 클러스터링과 식당 마커 표시, 클릭 이벤트 처리를 구현했습니다.

프로젝트 전반의 기획 완성도와 사용자 경험 설계를 인정받아, SW중심대학사업단 연합해커톤에서 대상을 수상했습니다.



UniVoice

2024.06 ~ 2025.07
개발 인원(15명) - 기획 2명 · 디자인 3명
AOS 4명 · iOS 4명 · 백엔드 2명

Kotlin | Android Studio | XML | MVVM | Clean Architecture

UniVoice는 놓치기 쉬운 학생회 공지를 한곳에 모아, 학생들이 중요한 정보를 빠르게 확인할 수 있도록 돕는 앱입니다. 전국 대학교 API를 활용해 학교 목록을 불러오고, 사용자는 소속 학교를 선택해 회원가입을 진행합니다. 회원가입 요청은 Slack 연동을 통해 해당 학생회에 전달되며, 학생회 측의 승인 이후 계정이 활성화됩니다.

또한 바쁜 학생들을 위해 읽지 않은 공지사항을 약 200자 내외로 요약해주는 AI 기반 '퀵스캔' 기능을 제공해, 긴 공지를 빠르게 파악할 수 있습니다. 공지사항을 저장하거나 '좋아요'로 반응을 남길 수 있어, 개인화된 공지 관리 경험도 함께 제공합니다.

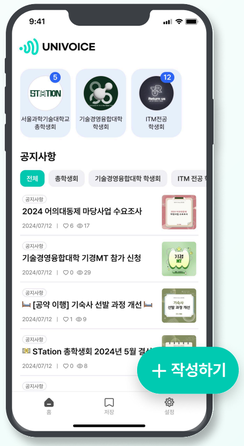
안드로이드 개발자로 참여해 퀵스캔 페이지와 로그인·저장 페이지 구현을 담당했습니다. 탭 기반 화면 전환과 스와이프 흐름을 고려해 사용자 경험을 구성했으며, 입력 포커스 처리와 디바운싱을 적용해 사용자 입력 과정에서의 사용성을 개선했습니다.

해당 프로젝트로 NOW SOPT APPJAM에서 우수상을 수상했습니다.

UNIVOICE

학생회와 학생의 목소리로 함께 만드는
대학생활 필수 앱

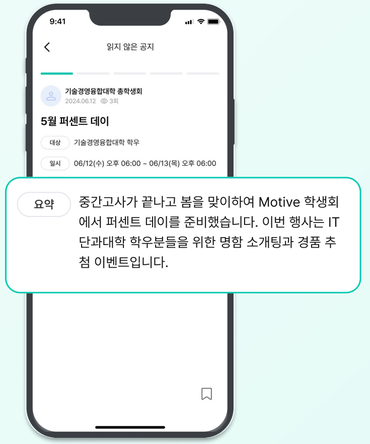
작성하기를 눌러



공지사항을 작성하고

공지사항 등록 완료!

직접 등록한 공지사항을
퀵스캔에서 확인해보세요!



Thank You

Gaeul Lee · Frontend Developer