오태준

xownswns@naver.com
01053309136

소개

“화면을 중심에 두고 사람과 사람을 잇는 개발자, **오태준**입니다.”

저는 사용자의 흐름과 팀의 흐름을 동시에 설계하는 습관이 있습니다.
협업과 설득의 힘을 믿으며, 이를 통해 팀을 하나로 묶고 더 나은 결과를 만들어 왔습니다.

저는 “공유되는 정보가 곧 집단의 지능이다”라는 말을 늘 기억합니다.
누구든 쉽게 이해할 수 있는 구조를 만들고, **함께 성장할 수 있는 환경**을 조성하는 데 집중합니다.

개발은 단순히 코드가 잘 돌아가는 것을 넘어서야 한다고 생각합니다.
팀원에게는 **유지보수가 쉬운 구조**를, **사용자**에게는 **자연스러운 경험**을 제공하는 것. 이 두 가지가 함께할 때 진짜 좋은 서비스가 완성된다고 믿습니다.

앞으로도 화면을 넘어 흐름을 설계하는 개발자가 되기 위해 꾸준히 배우고 성장하겠습니다.

장점

1. 저는 “몸이 몇 개냐”는 말을 자주 듣는, **열정적인 노력형 개발자** 오태준입니다.
2024년 군 제대 이후 처음 프론트엔드 개발을 시작했고, 늦게 시작한 만큼 누구보다 빠르게 성장하기 위해 꾸준히 공부하며 실무 경험을 쌓아왔습니다.
React와 Vue 등 다양한 프레임워크를 프로젝트에 적용하며 기술 스택을 확장했고, UX 관점에서 사용자 경험을 고려한 개발 역량도 키워왔습니다.

2. 개발만이 아니라 **협업과 소통**에도 적극적입니다.
어흥콘, DB 드림리더 등 다양한 활동에서 사람들과 함께 목표를 세우고 성과를 내며 협업의 힘을 배웠습니다.
“모른다”는 것을 부끄러워하지 않고 질문하며 배움을 이어온 덕분에, 지금은 13기 프론트엔드 트랙장과 홍보팀장을 맡아 팀원 교육과 대외 커뮤니케이션을 책임지고 있습니다. 이를 통해 기획력과 리더십, 협업 능력을 쌓아왔습니다.

3. **시간을 허투루 쓰지 않는 것**도 저의 강점입니다.
낮에는 개발을, 밤에는 배운 내용을 기술 블로그로 정리하며 지식을 공유합니다. 단순히 배우는 데 그치지 않고, 나만의 언어로 해석하고 더 나은 코드를 고민하는 습관이 자리잡았습니다.

최근에는 D-Order 프로젝트에서 약 10일간의 짧은 기간 동안 서비스를 제작·배포한 경험이 있습니다. 촉박한 일정 속에서도 리더십을 발휘해 API 명세를 실시간으로 공유했고, 팀원들의 작업 오류를 최소화하여 프로젝트를 성공적으로 마무리했습니다. 이 경험은 저에게 빠른 실행력과 협업 속도 조율 능력을 길러주었습니다.

앞으로도 저는 꾸준히 배우고, 나누고, 협력하며 **성장하는 개발자**가 되겠습니다.

스킬

JavaScript (TypeScript) · React · Next.JS · Github · Github actions · Figma · Notion · Slack · Trello

학력

동국대학교

컴퓨터정보통신공학부 정보통신공학전공

링크

https://velog.io/@taejun0/posts 블로그

https://github.com/taejun0 github

https://taejun0-portfolio.site/ 포트폴리오 사이트

2025 동아리 박람회 공식 웹 사이트, 동아리랑

개요

2025.02 - 2025.03 (2개월) / Web (Mobile View) / 9명 (FE 3, BE 4, PM 1, DESIGNER 1)

‘동아리랑'은 동국대학교 2025 동아리 박람회의 모든 부스 및 공연 정보를 한데 모은 통합 안내 웹 서비스입니다.
지도 그림 기반 UI를 통해 각 부스의 위치, 공연 시간표, 참여 동아리의 상세 정보까지 실시간으로 확인할 수 있습니다.

기술적 의사 결정

- React - 컴포넌트 기반 아키텍처로 UI를 효율적으로 분리할 수 있고, 프로젝트의 확장성과 유지보수에 강하며, 각 페이지 및 기능 단위를 독립적인 컴포넌트로 분리하였습니다.
- React Router v7 - 박람회 웹은 여러 페이지로 구성되어 있는 SPA로 설계하였기에 페이지 이동 및 상태 전달, 예외 처리 등이 필요했습니다.
- Styled-components - 박람회 사이트 특성상 다양한 UI를 많이 다뤄야했고, CSS-in-JS 방식은 컴포넌트 단위로 스타일을 관리할 수 있어 사용했습니다.
- Framer-motion - 메인 페이지와 같이 시각적으로 부드러운 모션 효과를 통해 몰입도를 높였습니다.

기여

- **메인 화면 및 페이지 전환 시스템 구성**
  - MainNavigation 컴포넌트로 헤더 네비게이션을 구현
  - Custom routing hook을 구성하여 goToPage, goBack 기능을 분리하여 페이지 간 이동 흐름을 일관성 있게 관리
  - Floating Bubble Animation을 Framer motino으로 구현하여 랜덤한 위치, 크기, 투명도를 가진 물방울을 주기적으로 생성하여 몽환적 시각적인 연출 구현
- **스타일링 및 반응형 대응**
  - Props 기반 동작 스타일링을 적용
  - $isActive, $isNothing 등 조건부 스타일 처리로 UI 시각화
- **부스/푸드트럭 통합 검색 기능 구현**
  - Custom Hook인 useBoothSearch 개발
- **예외 페이지 및 에러 UX 처리**
  - 404 또는 검색 결과 없음 처리로, 적절한 안내 메시지와 함께 ErrorBox를 제공
  - 페이지 구분 형 예외 메시지 렌더링
  - useReloadPage 훅을 이용하여 새로고침 기능이 필요한 페이지에 적용

트러블 슈팅 (1 / 2)

문제 상황

초기 개발 환경에서 npm을 사용하며 아래와 같은 문제가 발생했습니다.

- package-lock.json 의존성 충돌로 같은 버전 패키지라도 로컬 실행의 차이가 발생
- node_modules 내부에서 종속 패키지가 충돌하여 빌드 실패 지속

해결 시도 방법

1. npm ci를 도입하여 고정된 lock 파일 기반으로 설치하고자 함
   → 그러나, 여전히 lock 파일은 충돌하였고, 문제가 지속됨
2. .npmrc를 통해 strict-version 정책을 시도 (고정된 버전을 사용)
   → 그러나, 만약 일부 인원이 다른 버전을 다운로드 한 경우, 이를 수동으로 체크해야했고, lock file도 함께 정리해야하는 등 더 많은 소요가 투입됨
3. npm 대신 yarn으로 패키지 매니저를 전환 → 해결

알게된 점

- yarn은 초기 기대했던 lock 파일 충돌 문제 그 이상으로, yarn.lock 파일 구조와 버전 해석이 npm보다 일관적이고 안정적임.
  - 같은 버전의 패키지를 설치할 때, OS, Node Version, yarn Version이 다르더라고 lock 파일이 거의 동일하게 유지됨.
- yarn은 의존성을 설치할 때, 캐시를 통해 속도도 빠르며, 배포 자동화 과정 시에 반복되는 대기 시간을 크게 감소 시킴

즉, 단순한 패키지 매니저의 변경 만으로도 개발 안정성과 협업 효율성, 배포 속도까지 크게 개션됨을 경험하며, 초기 설계의 중요성을 체감했습니다.

트러블 슈팅 (2 / 2)

문제 상황

부스/푸드트럭 검색 기능 구현 당시, 사용자의 입력 타이밍마다 onChange 이벤트로 처리하고,
순식간에 너무 많은 API가 호출되어 다음과 같은 문제가 발생했습니다.

- 매 입력마다 API 요청으로 인해 서버 트래픽 과도화
- 느린 네트워크 환경에서 응답이 뒤섞이며 검색 결과가 중복되거나 꼬임
- UX 저하 및 페이지 렉 발생으로 높은 이탈률 예상

해결 시도 방법

1. onChange에 조건문을 넣어 입력 길이가 2자 이상일 때만 검색되도록 제한
   → 단기적인 효과는 있었으나, 2자 이상의 빠른 입력에는 여전히 중복 호출이 발생
2. 입력 타이밍을 조절하는 로직을 고민
   → 사용자가 입력을 멈출 때까지 기다렸다가 요청해야함을 깨닫고, debounce 적용 고려 → 해결

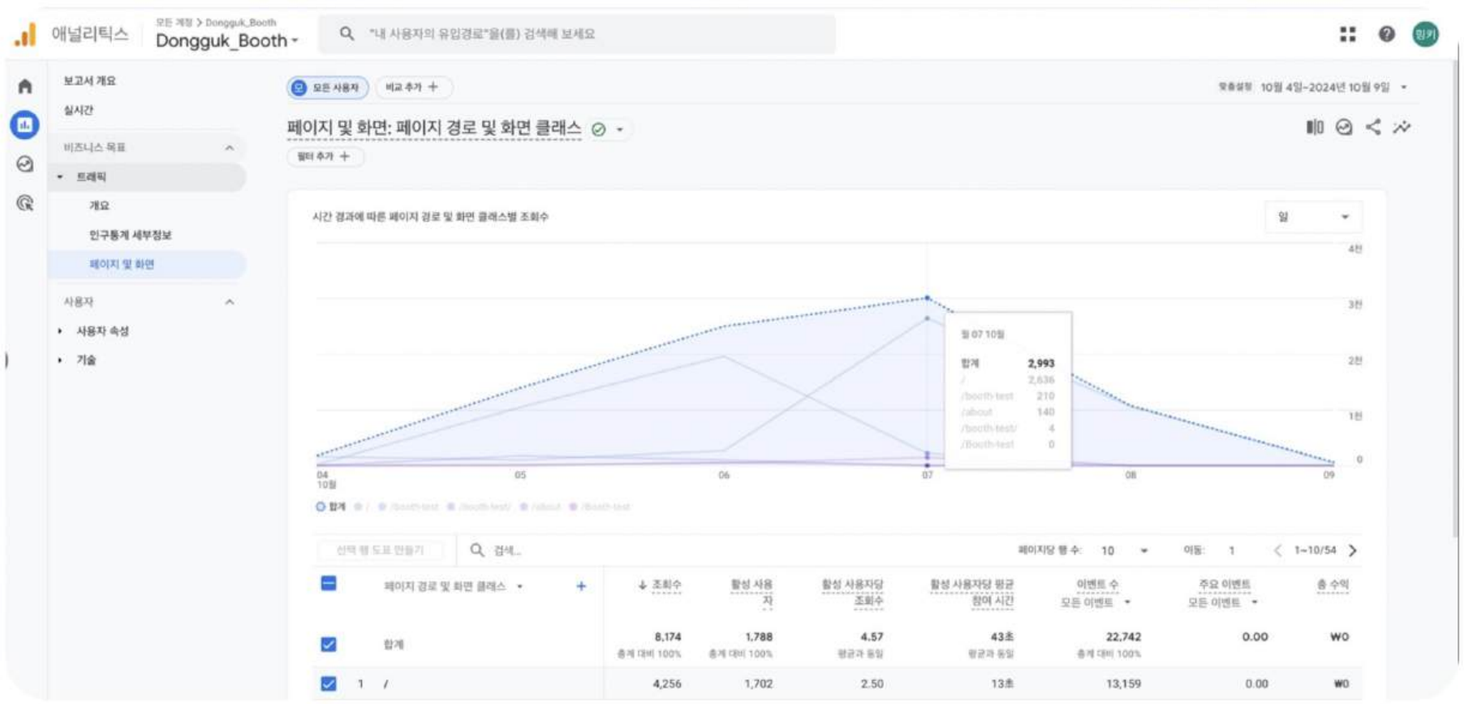
알게된 점

최종적으로 @uidotdev/usehooks에서 제공하는 useDebounce를 도입하였습니다.
사용자가 입력을 멈춘지 300ms 이후에만 API 호출이 발생하도록 설정하였으며, useEffect와의 연계로 debounceQuery가 변할 때만 요청이 전송되도록 하였습니다.

실제로, 간단한 검색어에도 수십번의 API 요청이 이어졌던 검색 페이지에서 2~3번의 API 요청으로 줄여 불필요한 호출을 제거하여 안정적인 UX를 제공했습니다.

- 단순히 호출을 줄이는 것보다 의도치 못한 요청을 제어하고, 사용자의 입력 흐름을 고려한 UX 설계의 필요성을 깨달았습니다.
- 사용자의 실시간 입력에 반응하는 직접적인 UX에 관련된 성능 요소는 디바운싱과 같은 기술이 필수적임을 알게 되었습니다.

별첨



성과

- 약 2,000 명의 활성 사용자
- 22,742 건의 이벤트 발생
- 평균 4.57회 페이지 조회, 사용자 평균 43초 체류
- 부스 페이지는 하루 최대 2,993회 조회 기록

실제 동아리 정보와 부스 정보를 미리 알아 볼 수 있어 도움이 되었다는 다수의 피드백

주소

Github https://github.com/LikeLion-at-DGU/2025_DongBak_Frontend.git

배포 환경 https://2025-dgu-dongbak.netlify.app

축제 부스 운영지원 웹 사이트, D-order

개요

2025.05 - 2025.08 (4개월, 2차 리팩토링) / Web / 9명 (FE 3, BE 4, PM 1, DESIGNER 1)

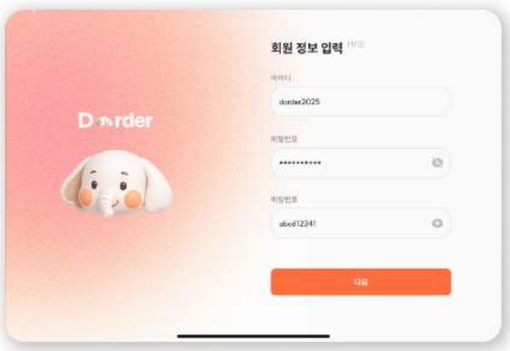
D-order는 행사/축제와 같은 오프라인 환경에서 혼잡한 주문 환경을 해결하기 위해 개발된 모바일 주문 및 운영 서비스입니다.
종이나 말로 이루어졌던 주문서 작성, 입금, 주문 확인 등 복잡한 과정을 디지털화하여 운영자와 고객 모두에게 효율적인 UX를 제공합니다.
1차 MVP 개발 만으로도 성공적인 성과를 얻었고, 2차 리팩토링을 거치며 1차 제공 시 사용자 피드백을 반영 중에 있습니다.

기술적 의사 결정

- React & TypeScript - UI 상태 변화가 많은 주문 시스템에 적합한 컴포넌트 기반 프레임워크인 React와 TypeScript로 타입 안정성을 확보했습니다.
- React Router v6 - 다단계 회원가입과 인증 분기처리를 위한 중첩 라우팅 구성을 위해 사용했습니다.
- Zustand - 장바구니 상태를 전역적으로 간단하게 관리할 수 있으며, 파일 자체가 가벼워 선택하게 되었습니다.
- Axios - 요청/응답 인터셉터 설정이 용이하고 토큰 갱신 로직 구현에도 적합하여 오류가 치명적인 주문 시스템에 적합하다 판단하여 사용했습니다.

기여

- 기능 중심 파일 구조로 개선
  - 기존 페이지/컴포넌트로의 단순 분리 구조를 넘어서, 기능 단위로 모든 폴더와 모듈을 재구성함으로 응집도가 높은 구조를 가진 프로덕트를 만듦
  - 1 ~ 3차 QA를 진행하며 발견된 문제들을 적극적으로 해결 (백엔드, 기획 및 디자인 팀과 Notion을 활용한 협업을 통해 문제 해결)



Admin Page

- **FE 인증 인프라 설계 경험**
  - 단순 로그인 처리에 그치지 않고, access + refresh 구조의 JWT 토큰 기반 인증 로직과 Axios 인터셉터 인증 흐름 구현
  - 토큰 만료 시 갱신 API를 통해 자동 갱신하고, 그 동안의 요청은 filledQeueu에 대기 후 일괄적으로 재요청 처리
  - 사용자 인증 상태에 따라서 접근 가능한 페이지를 제어하는 커스텀 훅을 구성하여 보안성과 UX를 모두 확보

User Page

- **MenuListPage의 전체적인 구조와 렌더링 흐름 구성**
  - 메뉴 데이터 호출 → 분류 → 섹션별 렌더링 → 수량 선택 → 장바구니 추가까지 UX를 고려한 Flow 구성
- **Zustand를 활용한 상태 관리**
  - 페이지 간 공유되는 장바구니 상태를 전역적으로 관리할 수 있도록 zustand를 적극 도입하여 Props Drilling 문제를 해결
  - Context나 Redux에 비해 설정이 간결해 빠른 개발, 유지보수 확보

트러블 슈팅 (1 / 2)

문제 상황

프로젝트 초기에는 hooks/, components/ 등 모든 코드가 전역 디렉토리에 위치한 구조를 사용하였고 다음과 같은 문제가 발생했습니다.

- 기능 간 경계가 모호하여, 특정 기능 수정 시에 전체 디렉토리를 탐색
- 유사한 이름의 컴포넌트들이 섞여 재사용성이 감소
- 파일이 어디서 사용되는지 추적이 어려워 유지보수성에 불리함

해결 시도 방법

팀장으로서 구조적 문제를 인지하고 기능 중심 폴더 구조로 전면 개편을 주도

1. 기능별 디렉토리를 설계하여 적용, 페이지 내의 관심사별 로컬 파일을 분리
2. 공통으로 사용되는 요소는 전역 components에 관리하며, 기능 내 요소는 페이지 내에서 관리하는 규칙을 정의
3. 작업 단위 또한 폴더 단위를 적용하여, 충돌 문제를 최대한 방지

알게된 점

- 전역 중심의 구조는 빠르게 개발하기엔 제격이나, 기능 확장 시에는 유지보수성과 협업 효율 모두를 저해한다는 사실을 알았습니다.
- 기능 중심 구조는 코드의 “응집도"를 높이고, 보다 명확한 구조를 제공하였습니다.
- 특히, 이런 구조는 끊임없는 논의를 통해서 단일화된 전략을 추구해야하며, 이는 팀 전체의 개발 문화와 속도에 영향을 미치는 일임을 깨달았습니다.

트러블 슈팅 (2 / 2)

문제 상황

뒷배경 UI를 위해 radial-gradient와 blur()를 조합한 4개의 Elipse 배경 요소를 도입하였으나
다음과 같은 문제가 반생하였습니다.

- GPU 과부하로 인한 input 입력 버벅임과 속도 저하 발생
- blur()가 겹쳐지면서 특정 기기에서는 배경이 깨지거나 늦게 나타나는 현상
- 문제 재현이 어려워, 원인을 분석하는데 큰 난항

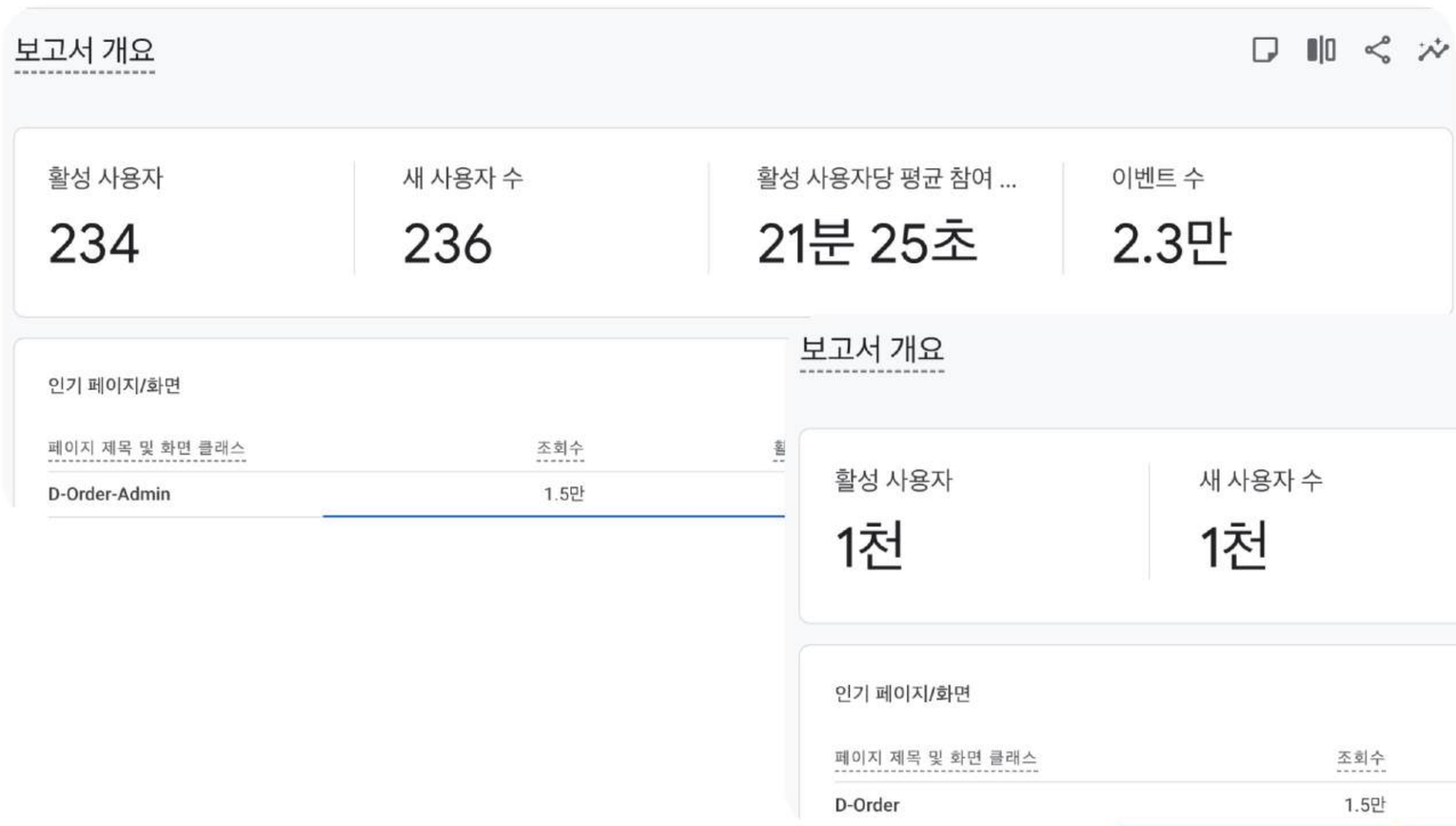
해결 시도 방법

1. DevTools 기반으로 성능 분석을 진행
   →Chrome DevTools의 Performance, Layers, Memory 탭을 통해 렌더링 명목 위치 확인
   →그러나, 모바일 성능 문제임을 깨닫지 못하고, input 입력 시 전체 DOM이 다시 그려져 무한 렌더링 발생
2. Lighthouse 모바일 진단을 자동화
   → 모바일 성능 기준 점수를 기준으로, PC나 Notebook이 아닌 모바일 (핸드폰, 패드) 환경에서 문제가 발생함을 깨달음.
   이는 곧, 모바일 기기의 성능이 원인임을 알게됨

알게된 점

- 먼저, 해당 문제에 직접적인 도움을 주진 않았지만, input 태그를 제대로 제어하지 않으면, DOM이 다시 그려져 모든 요소가 렌더링 됨을 알게 되었습니다. 이는 성능 병목의 주된 원인이 될 수 있어 이후로는 이를 방지하고 있습니다.
- CSS 기반 시각 효과는 브라우저, 기기별 처리하는 방식이 달라 항상 모바일 디바이스 우선 성능 테스트가 병행되어야함을 알게 되었습니다.
- 단순히 “동작하는 것"을 넘어서 사용자으 ㅣ환경은 물론 기기별 성능까지 고려한 UI가 제대로된 UX를 잡을 수 있음을 체감하는 계기가 되었습니다.

별첨



성과

- 3일간 약 7개의 축제 부스와 협력, 1,200 명의 유저를 확보
- 특히, 어드민은 약 21분, 유저는 약 2분의 긴 체류 시간과 유저, 어드민 페이지 모두 2만 건이 넘는 높은 이벤트 수 확보
- 실제 피드백들을 바탕으로 현재도 끊임없이 수정과 배포를 반복하며 “실제 현장에서 문제 없이 사용 가능한 서비스"를 만드는 경험

실제 배경 디자인으로 인한 렉이 있음에도 기존의 불편함을 해소하는데 큰 도움이 되었다는 피드백을 받았습니다.

주소

Github	https://github.com/LikeLion-at-DGU/2025-d-order-frontend-customer.git https://github.com/LikeLion-at-DGU/2025-d-order-frontend-admin.git
배포 환경	https://d-order.netlify.app/ https://d-order-admin.netlify.app/

개인 포트폴리오 사이트

개요

2025.06 - 2025.08 (3개월) / Web / Solo

저를 표현하는 포트폴리오 사이트입니다.

저를 가장 잘 표현할 수 있는 색은 회색이라고 생각하였고, 키 컬러로 사용하였습니다.
“흑과 백이 아닌 회색의 명도차이다" 어린 시절부터 가지고 있던 하나의 신념입니다.
갈등 상황 속에서도 다양한 의견과 스펙트럼이 있음을 기억하며 이들을 존중합니다.

기술적 의사 결정

- React & TypeScript - 명시적 타입 안정성을 추구하며 TypeScript를 사용하게 되었습니다.
- NextJS - 서버 구성요소(RSC) 와 App Router 방식으로 데이터 경계를 명확히 하고 로딩 성능을 확보하기 위해 사용했습니다.
- React Query - 서버 상태에 대한 공부를 하기 위해 낙관적 업데이트, 신선도 제어등의 공부를 위해 선택했습니다.
- Emotion - styled-components 레거시 이슈 및 Next와의 SSR 호환성으로 사용했습니다.

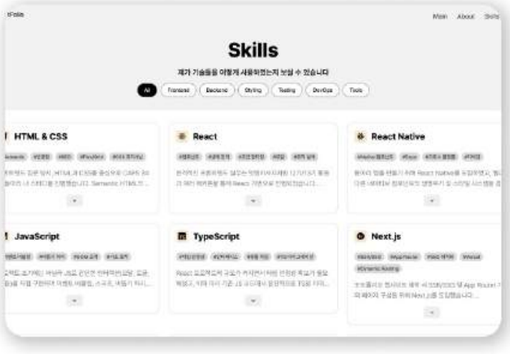
기여

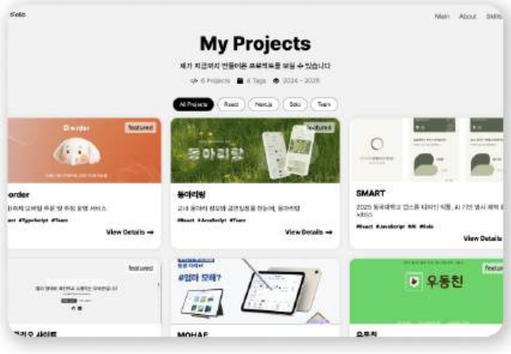
MainPage

- 각 섹션(About, Skills, Project)으로 들어가기 전에 간단한 소개를 담았습니다.
- 직접 작성한 글들을 모아둔 Articles 공간도 있으며, 최하단의 Contact를 통해 언제든 편하게 연락하실 수 있습니다.

About Page

- 작은 활동 기록과 함께 제가 중요하게 생각하는 가치들을 간단히 정리했습니다.
- 글을 읽으시면서 제 생각과 방향성을 조금 더 가까이 느끼실 수 있을 것입니다.





Skills Page

- 제가 경험한 기술들을 한눈에 보실 수 있도록 정리했습니다.

Projects Page

- 그동안 진행한 프로젝트 목록입니다.
- 설명, 트러블슈팅, 기여도를 한눈에 확인하실 수 있습니다.

챌린징 (1 / 2)

문제 상황

포트폴리오 사이트의 Lighthouse 점수 중 Performance·SEO가 낮게 측정되었습니다.
사용자 경험과 검색 노출도를 개선하기 위해 핵심 병목 원인을 진단하고 점진적으로 최적화를 진행하였습니다.

해결 시도 방법

1) SEO 메타 구성

- next/head(또는 App Router의 metadata)로 _app.tsx/레이아웃에 공통 메타 정의
- Open Graph / Twitter Card 삽입, 이미지 alt와 시맨틱 마크업 보완

2) 라우팅/배포 정책 정리

- 메인 도메인에 프로덕션 연결, 보조 도메인은 308 Permanent Redirect로 일원화
  - 불필요한 307(Temporary) 회전 제거 → 초기 재요청 비용 축소

3) 이미지 최적화 & LCP 개선

- 정적 자산 PNG → WebP 변환
- next/image 도입 → <picture>/WebP/사이즈 최적화 자동 처리
- LCP 후보에 우선순위 부여: 첫 카드/히어로 이미지를 priority로 지정

알게된 점

- 수치(점수)를 올린 게 아니라 사용자가 실제로 더 빠르게 페이지를 경험하도록 만드는 과정을 통해 성능·SEO·접근성은 단순한 지표가 아니라 UX에 직결되는 요소라는 걸 체감했습니다.
- 리디렉션 정책(307→308) 변경 하나로 FCP가 23.9s → 0.9s로 급격히 개선된 경험을 통해, 단순히 코드 최적화가 아니라 도메인·라우팅·배포 인프라까지 전체 흐름을 점검해야 한다는 깨달았습니다.
- 이미지 포맷(WebP), LCP 우선 로딩, 시맨틱 마크업 보완 → UX, SEO, 접근성까지 다루는 포괄적 역할 수행하며, “프론트엔드는 단순히 화면 그리는 역할이 아니다”라는 확신을 얻었습니다.

챌린징 (2 / 2)

문제 상황

포트폴리오 사이트에서 Next.js를 사용했으나 초반엔 단순히 파일 기반 라우팅의 편리함으로 도입했을 뿐 SSR이나 구조적 장점은 고려하지 않았습니다. 그러나 프로젝트가 커지면서 레이아웃 중복, 라우팅 복잡성, 상태 관리 혼합 등의 문제가 발생했고, 이를 해결하기 위해 App Router로 리팩토링을 진행했습니다.

해결 시도 방법

1) App Router 전환

- 중첩 레이아웃 구조: 경로별 layout.tsx 활용 + 라우팅 그룹으로 역할을 분리
- 서버 컴포넌트(RSC): 기본 서버 렌더링 → 불필요한 JS 제거, 보안 강화
- 클라이언트 컴포넌트 분리: 상호작용 영역에만 "use client" 선언
- URL 파라미터/쿼리 단순화: useRouter, useSearchParams 도입 → 직관적이고 가벼운 API
- SEO/파비콘 자동화: App Router의 metadata와 icon 설정으로 _document.tsx 불필요

2) 시행착오

- useRouter/useSearchParams의 차이 → 비동기 객체여서 동기 접근 불가, 클라이언트 전용으로 사용
- "use client" 선언 누락 시 오류 발생 → 서버/클라이언트 컴포넌트의 경계 개념을 이해하는 계기

알게된 점

- 단순 라우팅 → 중첩 레이아웃, 서버/클라 컴포넌트 분리로 프로젝트 아키텍처 자체가 단순·명확해지며 라우팅은 단순 경로가 아니라 전체 코드베이스 유지보수성에 직결된다는 인식을 얻었습니다.
- "use client" 선언과 서버 컴포넌트 경계에서 SSR, RSC 개념을 몸으로 체득하며 결과적으로 SEO/초기 로딩 속도 향상까지 직접 경험.했습니다.
- metadata, next/navigation 등 새로운 Next.js 13에 익숙해지며 앞으로의 프로젝트 또한 현대적 아키텍처로 설계할 수 있는 기반을 확보했습니다.

별첨

- 프론트엔드 개발은 “화면만”이 아니라 서비스 전체 품질 관리임을 깨달았습니다.
- 작은 설정 변경이 큰 성능 향상을 가져올 수 있음을 알았습니다.
- 마지막으로, 구조와 설계에 투자해야 장기적으로 유지보수·확장성이 보장됨을 알았습니다.

즉, 사용자 경험을 고려한 “구조적 설계"가 가능한 프론트엔드 개발자가라는 정체성을 얻는 소중한 경험을 했습니다.

주소

Github	https://github.com/taejun0/PortfolioPage
배포 환경	https://taejun0-portfolio.site/

결과만을 보는 것이 아닌, 과정을 생각합니다.

저는 어릴 때부터 LG 트윈스라는 야구팀을 좋아했습니다. 단순히 팀이 멋있어서가 아니라, 많이 지더라도 늘 묵묵히 자기 길을 가는 팀 분위기와 오너 마인드, 선수들의 태도, 팬을 대하는 진정성 있는 문화가 좋았습니다. 잘할 때보다 어려울 때 더 애정이 깊어졌고, 언젠간 꼭 우승을 함께하고 싶다는 마음으로 오랫동안 응원해왔습니다.

하지만 몇 년 전, 정말 기대가 컸던 시즌이 무너졌을 땐 크게 좌절했습니다. 공들여 응원해왔던 시간들이 헛되게 느껴졌고, 응원 자체가 무슨 의미가 있나 싶었습니다. 돌아보면 그땐 결과에만 집착하고 있었던 것 같습니다.
시간이 지나고 다시 경기를 보기 시작하면서 시선이 달라졌습니다. 누가 이기느냐보다, 한 구 한 구에 집중하는 즐거움을 알게 됐고, 긴 시즌 속에서 선수들이 쌓아가는 과정이 더 중요하다는 걸 깨달았습니다.

야구라는 종목은 현재 1등 팀도 승률이 겨우 6할을 넘기며 4할은 반드시 진다는 게 제가 생각하는 가장 큰 특징입니다. 결과에 목매달고 일희일비하지 않고 과정에 집중하는 태도가 더 건강한 응원 방식이라 생각합니다.
이후로 저는 일상에서도 그 태도를 적용하려 노력하고 있습니다. 단기적인 성과에 흔들리기보다 꾸준히 과정을 쌓는 습관을 만들고, 감정적으로 무너질 때는 스스로 호흡을 조절하며 중심을 잡습니다. 실패는 끝이 아니라 하나의 순간이고, 결과보다 과정을 이해하는 태도를 야구를 통해 배웠습니다.

2주만에 실사용 서비스를 만든 경험이 있습니다

교내 축제에서 실시간 부스 서비스인 'D-Order'를 만들게 됐습니다.

당시 교내 축제 부스에서는 대부분 모든 주문을 종이로 받아 적는 방식이었고, 단기 행사 특성상 포스기를 설치하는 것도 어려운 상황이었습니다. 그래서 부스 운영자와 손님 모두가 쉽게 쓸 수 있는 시스템이 꼭 필요하다고 생각하였습니다.

개발부터 배포까지 딱 2주밖에 남지 않았고, 저는 그 안에서 프론트엔드 리더를 맡게 됐습니다. 수백 명이 실제로 쓸 서비스라서 "제대로 동작하는 것"은 기본이고, "누구나 쉽게 쓸 수 있어야 한다"는 압박감도 있었습니다.

우선은 욕심을 줄이고, 핵심 흐름에 집중한 MVP를 먼저 만들자고 팀원들과 방향을 정했습니다. "주문 → 확인 → 수령"의 플로우가 끊기지 않는 걸 최우선으로 두고, 그에 맞춰 화면 구조를 설계했습니다.
백엔드와는 API 명세를 사전에 정리해 소통 오류를 줄였고, Figma를 활용해 컴포넌트 구조를 시각화하면서 팀원 간 혼선을 줄였습니다.

그 결과, 축제 시작 전에 무사히 완성하여 배포할 수 있었고, 기본적인 사용 가이드를 배포하여 운영진들에게 제공하였습니다. 또한, 3일간의 축제 기간동안 운영진이 항시 대기하며 즉각적인 요청에 대응하였습니다.
총 3일간 7개 부스에서 운영진, 사용자 합쳐 총 4만 건이 넘는 이벤트가 발생하였고, 1200명이 실제로 사용하는 서비스로 발전하게 되었습니다.

이 경험을 통해, 빠르게 구조를 정리하고 협업을 실천에 옮기는 것이 개발자에게 얼마나 중요한 가치인지 체감하였습니다.
앞으로도 해당 경험을 기반으로, 실사용자에게 필요한 서비스를 만드는 개발자로 성장해나가고 싶습니다.

다양한 사람들과 만나 그들을 잇는 역할을 해왔습니다

DB 드림리더 장학생 활동은 전국 각지의 대학생들과 팀을 이루어 ESG 중 '환경'을 주제로 캠페인을 기획하고, 기업과 함께 솔루션을 실현하는 프로젝트였습니다. 팀에는 PM을 준비하는 기획 중심의 학생, SNS 콘텐츠 제작에 능한 학생, 프로젝트 협업 경험이 많은 저까지, 각자의 배경과 성향이 뚜렷했습니다. 하지만 바로 그 다양성 때문에, 처음에는 팀 내 우선순위와 의사소통 방식이 엇갈리게 되었습니다.

저는 그 안에서 서로의 강점을 연결해주는 중간다리 역할을 자처했습니다. 캠페인의 핵심 키워드를 ESG 관점에서 정리해 Notion에 공유하고, 캠페인 이름부터 목표, 실행 단계별 과제를 체계적으로 정리해 공동의 방향을 시각화했습니다. SNS를 맡은 팀원과는 콘텐츠 유형과 업로드 일정을 함께 기획했고, 도시 녹지공간 조성을 위한 나무심기 프로젝트의 진행 과정과 의미를 담은 콘텐츠를 제작하며 직접 실행에도 참여했습니다.
또한 팀 전체의 흐름을 위해 콘텐츠 기획 템플릿과 데일리 체크리스트를 만들어 공유하면서, 각자의 역할이 유기적으로 연결되도록 조율했습니다.

그 결과, 캠페인은 높은 완성도를 인정받아 팀원 전원이 우수 장학생으로 선발되어 베트남 해외봉사에도 함께 참여할 수 있었습니다.

이 경험을 통해 저는 개발 외적으로도 서로 다른 배경과 가치관을 가진 사람들과의 소통, 그리고 하나의 목표 아래 협업을 조율하는 능력을 길렀습니다. 단순한 의견 전달을 넘어, 상대방의 입장을 이해하고 각자의 장점을 프로젝트에 녹여내는 과정이야말로 협업의 본질임을 몸으로 느꼈습니다.
앞으로도 다양한 동료들과 협력하며, 기술뿐 아니라 사람 중심의 개발자로 성장하고 싶습니다.
