Portfolio

김준현 | KIM JUNHYEON

> 사용자 경험을 개선하는 UI/UX 중심의 프론트엔드 개발자

프로필

Typescript와 React를 활용해 **사용자 중심의 UI/UX를 설계 및 개발**합니다. 협업 프로젝트를 통해 **Next.js 및 React.js 기반 서비스의 설계, 개발, 배포 경험**을 쌓았으며, **AWS 기반 CI/CD 파이프라인을 구축하여 자동 배포 환경을 구현한 경험**이 있습니다. 이 과정에서 실질적인 문제 해결 경험을 쌓았고, 협업 프로젝트를 위한 효과적인 커뮤니케이션 방법을 익혔습니다.

Figma를 활용한 UI/UX 디자인부터, React 컴포넌트 설계 및 최적화가 가능합니다. Notion과 Discord 등 협업 도구를 적극적으로 활용하여 원활한 커뮤니케이션을 추구합니다.

이메일

junhyeon0218@gmail.com

전화번호

+82 10-3411-1327

GitHub

GitHub/junhyeon0218

Velog

Velog/junhyeon0218

프로젝트

Calog

2024.08 ~ 진행 중

협업 프로젝트

관련 자료 및 링크

GitHub

Figma - UI/UX

FigJam- DDD

기술/스택

`NextJS` `Typescript` `Tailwind CSS` `Zustand` `GitHub` `AWS S3` `AWS CodeDeploy` `AWS EC2` `shadcn`

소개

Calog는 캘린더 기반의 일정 공유 및 소통 서비스 입니다. PC 환경에 최적화된 인터페이스입니다.

▼ 담당 역할 및 핵심 기여

폼 유효성 검사 및 UX 최적화

- `OnChange를` 활용해 불필요한 리렌더링 최소화 및 실시간 유효성 검사 구현
- 이메일 및 닉네임 중복 검사 API 요청에 `debounce` 적용으로 **불필요한 API 요청 감소**

프로필 이미지 업로드 최적화

- `react-image-crop`으로 이미지 크롭 기능 구현
- `sharp.js`를 활용해 2MB 이미지를 100KB 이하로 압축해 95% 용량 감
- 최적화된 이미지 적용 후 네트워크 로딩 속도 개

팔로우 API 연동 및 UI 구현

- 사용자 간 팔로우/언팔로우 API 연동 및 UI 구현
- `Zustand` 상태관리로 UI 실시간 반영 및 동기화 처리

CI/CD 파이프라인 구축

- `GitHub Actions`으로 빌드 및 S3에 압축 후 업로드
- `CodeDeploy`로 EC2에서 압축 풀기 후 빌드

UI/UX 설계

- `Figma`를 활용한 전체 와이어프레임 및 디자인 시스템 구축
- 공통 컴포넌트 설계로 재사용성 및 유지보수성 향상

▼ 개발 과정의 도전과 개선

CodeDeploy 배포 자동화 실패 (ApplicationDoesNotExistException, UnknownError)

- 고민: GitHub Actions → AWS CodeDeploy 파이프라인 구축 중 **배포가 반복적으로 실패**
  - `ApplicationDoesNotExistException` 발생 (애플리케이션/배포 그룹 인식 불가)
  - CodeDeploy 콘솔에서 **인스턴스 이벤트 미수신** 및 `UnknownError` 발생
  - EC2의 `codedeploy-agent` 가 이벤트를 받지 못해 배포 단계가 진행되지 않음
- 해결:
  1. **EC2 환경 점검**
     - `codedeploy-agent` 재설치 및 재시작
     - IAM Role 부여 및 `AWSCodeDeployRole` 권한 + S3 접근 권한 추가
  2. **GitHub Actions 수정**
     - `chmod +x scripts/*.sh` 추가하여 배포 ZIP 내 실행권한 보장
- 결과:
  - GitHub Actions → S3 업로드 → CodeDeploy → EC2 반영까지 **완전 자동화 성공**
  - 권한 문제 및 에이전트 미동작 이슈 해결
  - `AfterInstall` 스크립트 정상 실행, 서버에 안정적으로 반영됨

이미지 최적화 처리 위치 결정 (클라이언트 vs 서버)

- 고민: 이미지 크롭 및 최적화를 **프론트에서 할지, 서버에서 할지** 고민
- 해결:
  - 프론트엔드: `browser-image-compression` 사용 시 업데이트 없음, 번들 사이즈 큼
  - 서버: `sharp.js` 는 최근 업데이트 활발, 경량, `Next.js` 서버 사이드 렌더링과 잘 맞음

    → **서버에서 sharp.js로 이미지 처리 결정**
- 결과:
  - 클라이언트 부담 감소
  - 이미지 로딩 속도 **0.3초 → 0.02초로 개선**
  - 유지보수 및 확장성 측면에서도 장점 확보

복잡한 도메인 관계 설계 (DDD 일부 적용)

- 문제: 일정, 게시글, 유저, 태그 간의 복잡한 관계로 **모델링이 불명확**
- 해결:
  - 도메인 주도 설계(DDD) 방식 적용
  - 핵심 도메인 식별 → 하위 도메인 분리 → 바운디드 컨텍스트 구분
  - `유비쿼터스 언어` 사용으로 팀 내 개념 통일
    - 예: "쓰레드" → "댓글이 새 글로 변환되는 기능"으로 정의
- 결과:
  - **의사소통 효율성 향상**
  - 향후 유지보수 및 기능 확장 시 의존도 감소

📌 배운 점

- Next.js와 TypeScript를 활용한 SSR 및 정적 사이트 최적화 경험
- 이미지 최적화 문제 해결 경험
- 도메인 중심 사고(DDD) 적용을 통한 구조 설계 경험
- CI/CD 파이프라인 구축을 통한 자동화 및 협업 속도 개선
- 협업 툴을 활용한 팀 커뮤니케이션 능력 강화

poeun

2025.09 ~ 진행중

협업 프로젝트

관련 자료 및 링크

GitHub

poeun.vercel

기술/스택

NextJS
Typescript
Tailwind CSS
Zustand
GitHub
Vercel

소개

poeun은 개인 포트폴리오 사이트입니다. 노션을 데이터베이스로 사용했습니다. 지속적으로 업데이트를 진행하고 있습니다.

▼ 담당 역할 및 핵심 기여

노션 API 연동 및 데이터 파싱 로직 구현

- 최소 바디로 먼저 성공 확인
- 노션 API의 rich text을 파싱하기 위한 함수 개발

프로젝트 카드/모달 UI 구성 및 상태 관리 연결

- NotionProject 모델 정리와 역할/기여/도전/배운점 구조 통일 및 빈 섹션은 숨김 처리
- 텍스트 파싱을 특수문자를 사용해 규칙적으로 맞춤 및 중복 제거

SWR을 사용한 최적화

- 서버에서 먼저 패치해 첫 렌더 빠르게 하고 클라이언트는 swr 재검증으로 최신화

▼ 개발 과정의 도전과 개선

Notion API 버전 변경 대응

- 문제: API 버전 업데이트로 기존 쿼리가 작동하지 않음
- 해결:
  - 공식 문서를 통해 `data_sources` 엔드포인트로 마이그레이션
  - 최소 바디로 먼저 호출 성공을 확인
  - 정렬 옵션을 단계적으로 추가하여 안정성 확보
- 결과:
  - API 호출 성공

SWR 적용과 초기 데이터 최적화

- 문제: 초기 로딩 시간이 길어 사용자 경험 저하
- 해결:
  - 서버에서 선패치하여 초기 데이터 제공
  - SWR로 클라이언트 재검증 및 캐시 관리
  - 실패 시 캐시 데이터 유지하는 페일세이프 가드 구현
- 결과:
  - 네트워크 오류 시에도 안정적인 사용자 경험 제공

📌 배운 점

- Notion API 연동 및 data_sources 쿼리 적용 경험
- 최소 바디로 먼저 성공 확인 후 정렬/필터 옵션을 단계적으로 추가하는 방법 습득
- 서버 선패치 + SWR 재검증 흐름 이해 및 초기 페일세이프 설계 경험
- 실패 시 캐시 데이터 유지와 오류 가드로 사용자 체감 품질 유지

밥피엔스

2024.08 ~ 2024.09

협업 프로젝트

관련 자료 및 링크

GitHub

소개

밥피엔스는 개인의 입맛과 성향을 고려한 맞춤형 음식 추천 및 식사 모임 플랫폼입니다. 모바일 환경에 최적화된 인터페이스로 쉽게 이용 가능합니다.

▼ 담당 역할 및 핵심 기여

로그인 및 회원가입 기능 개발

- `React-Hook-Form`을 활용해 폼 상태 관리 간소화

Figma - UI/UX

기술/스택

`React`
`Typescript`
`Tailwind CSS`
`Zustand`
`AWS S3`
`AWS`
`CloudFront`
`GitHub`
`GitHub Action`

- `onBlur` 기반 유효성 검사로 UX 개선
- 이메일/닉네임 중복 검사에 `debounce(500ms)` 적용 → API 요청 절감

FTI 검사 페이지 구현

- 서버로부터 JSON 형태의 질문 데이터를 받아와 렌더링
- 사용자 응답을 분석하여 결과 유형 계산 및 시각화
- 카카오톡 공유 기능, URL 공유 기능 등 결과 공유 기능 구현

위치 기반 음식점 추천 기능 개발

- `Geolocation API`로 사용자 위치 수집
- `Kakao Map API`를 통해 2km 이내 음식점 필터링 및 지도에 마커 표시
- 마커에 **서비스 로고 커스터마이징** 적용 → UX 향상

API 통신 구조 설계 및 리팩토링

- `Axios Interceptors`로 인증 토큰 자동 주입 및 갱신 처리
- `jwt-decode`로 Access Token 만료 여부 확인 → 자동 Refresh 처리
- API 폴더 구조화: `utils / apis / services`로 역할 구분하여 유지보수성 향상

디자인 시스템 구성 및 공통 컴포넌트 제작

- `Button`, `Input`, `Tag` 등의 공통 컴포넌트 제작
- Prop 기반으로 크기/색상/이벤트 처리 가능하도록 설계
- 재사용성 높은 구조로 구성하여 생산성과 일관성 모두 확보

문서 작성 및 협업

- 사용자 요구사항 정의서, API 명세서, 화면 정의서 등 다양한 기획 문서 작성
- 백엔드와의 협업을 위한 API 설계 및 피드백 회의 주도

▼ 개발 과정의 도전과 개선

위치 기반 지도 서비스 개발 (Kakao + Geolocation API)

- 문제: 사용자 위치를 기준으로 정확한 음식점 마커 표시가 어려움
- 해결:
  - `Geolocation API`를 통해 현재 위치 수집
  - `Kakao Map API`의 공식 문서와 다양한 예제 분석
  - 마커를 서비스 로고 이미지로 커스터마이징 → 직관적 UI 제공
- 결과:
  - 위치 기반 추천 시스템 정확도 향상
  - 사용자 만족도 및 UI 완성도 개선

인증 흐름 및 API 관리 구조 설계

- 문제: Access Token 만료 시 로그인 실패 및 사용자 불편
- 해결:
  - `jwt-decode`로 만료 시점 사전 감지
  - `Axios Interceptors`를 통해 자동으로 Refresh Token으로 재발급
  - API 호출을 `instance`, `apis`, `services` 단위로 모듈화
- 결과:
  - 인증 실패 빈도 **감소**
  - API 재사용성과 유지보수성을 높임

📌 배운 점

- 사용자 경험을 고려한 모바일 최적화 UI 설계
- 인증 처리 로직의 복잡성과 보안 흐름 설계 경험
- 위치 기반 서비스의 사용자 중심 접근 방법
- `Axios`, `JWT`, `API 구조화`를 통한 실무형 코드 작성 역량 향상
- 팀원과의 문서 기반 협업 능력 강화