정규상 | 프론트엔드 3년차 경력기술서

주식회사 타르트

프로덕트팀 | 프로덕트 엔지니어(프론트) 2022.05.03 ~ 2025.01.22

미술품, 음악, 부동산 등 STO 상품 비교 서비스 PRAP

React 18 | React Native 0.73 | RN Web | React-Query v5 | Emotion | Jotai

크로스플랫폼 앱 아키텍처 재구축

- 레거시 Expo 기반 애플리케이션 → React Native CLI 마이그레이션 → 확장성 확보 및 빌드 번들 최적화
- 불필요한 Expo 런타임 의존성 제거 → 성능 안정성 및 앱 초기 구동 속도 개선
- 핵심 비즈니스 로직 및 라우팅 플로우 재구성 → 코드 일관성 및 유지보수성 향상
- AppCenter CodePush 연동 → OTA 업데이트 자동화 및 안정적인 버전 관리 체계 구축

멀티플랫폼 자산 관리 시스템 개발

- 10개 이상 플랫폼 데이터를 통합 관리하는 포트폴리오·자산 관리 시스템 설계 및 개발
- Client-side 데이터 Aggregation 적용 → 실시간 자산 변동 정보 제공
- 대규모 자산 데이터 처리에 최적화된 → 캐싱 및 상태 관리 전략 수립

React Native Web 성능 최적화

- RN Web 환경에서 Virtualized List 비호환성 문제 해결 → 렌더링 실패 및 스크롤 지연 이슈 제거
- 웹 번들 및 리소스 구조 리팩토링 → 전반적인 사용자 경험 및 핵심 성능 지표 개선
  → LCP 86% 단축
  → 초기 진입 시 리소스 로딩 94% 감소

디자인 시스템 아키텍처 구축

- Emotion 기반 CSS-in-JS 디자인 시스템 설계 → 토큰 기반 스타일 일관성 확보
- 서비스 간 공유 가능한 컴포넌트 라이브러리 구축 → 중앙화된 디자인 토큰 체계 정립
- 다크 모드 대응 UI 아키텍처 설계

API 및 데이터베이스 구조 개선

- Repository 패턴 적용
  → API 요청 흐름 모듈화 및 추상화
  → 유지보수성 향상 및 중복 코드 최소화
- 레거시 로컬 DB → MMKV 마이그레이션
  → 데이터 접근 속도 200ms → 50ms (4배 개선)
- Async-storage / Encrypted-storage 제거 → 중복 구현 제거 및 앱 로딩 성능 개선

국내 최초 STO 거래소 PRAP X

React 18 | NextJS 14.2 | React-Query v5 | Styled-component → Tailwind | framer-motion

계약 프로세스 디지털 트랜스포메이션

- React Hook Form + Zod 기반 선언적 폼 상태 관리 및 스키마 검증 시스템 구축
  → 일관된 데이터 구조 유지 및 사용자·개발자 입력 오류 최소화
- 계약 조건에 따라 변화하는 동적 유효성 검사 로직 설계
  → 데이터 무결성 확보 및 복잡한 계약 흐름에서도 UX 개선
- 양수,양도 계약서 자동 생성 시스템 개발
  → 계약 문서 생성·관리 프로세스 표준화 및 자동화

디지털 서명 및 결제 시스템 구축

- HTML Canvas + Vanilla JS 기반 디지털 서명 모듈 개발
  → 서명 입력 시 지연 없는 고성능 처리 및 주요 브라우저 호환성 확보
- 서명 이미지 캡처·저장·압축 기능 구현
  → 문서 전송 비용 절감 및 처리 효율 개선
- Stripe.js 결제 연동
  → 안정적이고 일관된 결제 플로우 구축

사용자 인증 및 알림 시스템

- 카카오톡 알림톡 API 기반 간편 로그인 및 본인 인증 기능 개발
  → 고객 접근성 향상 및 인증 프로세스 단순화
- 인증 단계별 보안 검증 로직 설계
  → 안전한 사용자 인증 흐름 구축

모노레포 아키텍처 마이그레이션

- 기존 개별 웹 서비스를 모노레포 구조로 통합
  → 공통 모듈 및 디자인 라이브러리 관리 효율성 극대화
- UI 시스템 Styled-Components → Tailwind CSS 마이그레이션
  → 스타일 일관성 확보 및 개발 생산성 향상
- 컴포넌트 재사용성 개선
  → 서비스 간 유지보수 비용 절감

문제해결 사례 →

문제해결 사례

계약 프로세스 디지털 트랜스포메이션

문제

- 입력된 정보를 기반으로 내부 인력이 계약서를 직접 수기 작성하는 비효율적인 프로세스 존재
- 계약서 양식이 다양해 작성 오류 및 처리 시간 증가

해결

- 계약서 동적 생성 프로세스를 직접 기획·개발
- 계약서 양식을 HTML 템플릿으로 구조화
- 입력 정보 및 디지털 서명을 DOM에 반영한 뒤 캡처하여 계약서 이미지 생성
- 생성된 계약서를 백엔드로 전송하는 구조 설계

성과

- 반복적·수동적인 계약 작성 업무 제거로 팀 생산성 향상
- 클라이언트에서 계약서 이미지 생성, 백엔드는 평문 데이터만 저장
  → 스토리지 처리 비용 절감
- PDF 대신 DOM 캡처 방식 채택
  → 100ms 이내의 빠른 계약서 생성 성능 확보

모노레포 마이그레이션

문제

- 2024년 하반기~2025년 상반기 출시 예정 서비스들이 동일한 디자인 시스템과 API를 공유해야 하는 상황
- 기존 PRAP 앱, PRAP X 웹 서비스가 컴포넌트 / 디자인 라이브러리 / API를 각각 독립적으로 관리
- 프로젝트별 Lint / Prettier 설정이 달라 컨벤션 불일치 및 개발자 생산성 저하 발생

해결

- 개인 과업으로 Turborepo 기반 모노레포 아키텍처 도입 제안 및 구축
  → 기존 서비스와 신규 서비스 간 UI, 개발 경험 일관성 확보
- 프로젝트별 src 변경 시에만 자동 CI가 동작하도록 워크플로우 스크립트 개선
  → 불필요한 빌드, 테스트 비용 감소
- PRAP-web / PRAP-x-web / PRAP-ui 세 프로젝트로 구조 분리
  → 공통 UI 컴포넌트 및 디자인 시스템 재사용성 강화

성과

- 신규 프로젝트 개발 시 공통 컨벤션 및 컴포넌트 즉시 활용 가능
  → 개발 초기 세팅 비용 감소 및 생산성 향상
- 향후 팀 인원 증가 및 서비스 확장 상황에서도
  프로젝트 관리 및 협업 효율성 개선 → 팀 전체 생산성 증대

디자인 시스템 아키텍처 구축

문제 상황

- 초기 디자인 시안에 반응형 규칙과 다크모드 가이드가 부재
- 동일 목적의 컴포넌트임에도 여러 미세 차이가 존재
- 디자인 기본 원칙이 일관적으로 적용되지 않아 유지보수 비용 증가

해결 과정

- 다크모드 전용 컬러 팔레트 설계 및 전체 프로덕트 적용
- Atomic Design 패턴 도입으로 UI 컴포넌트 체계 재정비
- 전체 spacing 시스템을 4px 단위로 규격화하여 컴포넌트의 시각적 일관성 확보

성과

- 디자이너가 다크모드 시안을 별도로 복잡하게 고려할 필요가 없어져
  → 디자인·개발 협업 효율 및 팀 전체 생산성 향상
- Atomic 패턴 기반 UI 재구축으로
  → 재사용 가능한 컴포넌트 증가, 유지보수 비용 감소 및 개발 생산성 크게 향상

API 및 데이터베이스 개선

문제 상황

- 자산 기능을 개발하면서 백엔드 저장소 외 로컬 데이터베이스를 사용해야하는 상황 발생
- 로컬 데이터베이스로 Async-Storage 채택
  ⇒ 타사 플랫폼의 로그인 정보를 다루기 때문에 암호화 필요 ⇒ Encrypted-Storage 함께 사용
  ⇒ 중복코드 발생
- 7일~10일 간격 로컬 데이터베이스 데이터 유실되는 문제 발생
- API 관리방법 부재로 저장소 접근 로직과 비지니스 로직이 섞여 가독성이 매우 낮음

해결 과정

- 7일 간격 캐시 초기화 아이폰 정책과는 무관함 → 안드로이드에서 동일현상 발생
- 7일~10일 간격으로 발생하는 이슈를 트러블슈팅 하기에는 이미 사용하는 유저들에게 크리티컬한 문제
  ⇒ 코드유지 및 트러블슈팅 vs 리팩토링 최대한 빠른 해결에 대한 트레이드오프
- 로컬 데이터베이스 MMKV 로 전환
- API 관리 방법으로 Repository 패턴 채택 및 로컬 데이터베이스 통신 로직 리팩토링

성과

- 빠른 판단으로 2일 안에 문제 해결
- MMKV 마이그레이션
  ⇒ Async, Encrypted Storage 중복코드 200+ line 제거
  ⇒ 데이터 읽기 속도 200ms → 50ms 향상
- Repository 패턴 사용으로 다양한 문제 해결
  ⇒ 비지니스 로직, 저장소 접근 로직 분리 → 개발 생산성 향상
  ⇒ 어뎁터 의존성 주입으로 유연한 저장소 선택 가능 → 개발 생산성 향상

사이드 프로젝트

2024.07.15 ~ 진행 중

발로란트 스킨/에셋 검색 및 랭킹 서비스 Popskin

React 19 | NextJS 15 | React-Query v5 | Tailwind | Kotlin | SpringBoot → NestJS

https://popskin.gg - MAU 5.3K

개요

Next.js 15 신기능(App Router 개선, Server Actions 등) 적용 및 학습을 위해 신규 기술 스택 기반 전체 서비스를 설계 및 프론트/백엔드 아키텍처 설계

주요 담당 업무 및 성과

- Next.js 15 기반 프론트엔드 아키텍처 설계 및 구현
  - Server Actions, React 19 기반 최적화 기능 적극 활용 렌더링 성능 및 페이지 응답성을 개선

- 백엔드 기술 스택 Kotlin Spring → NestJS로 마이그레이션
  - Kotlin Spring 환경 대비 개발 생산성과 프론트엔드 연동성 개선을 위해 NestJS로 이전
  - API 재구성 및 서비스 모듈 기반 구조 정립
  - Puppeteer 크롤링 기술을 활용한 데이터 수집 및 정제

인증 및 사용자 식별

- Supabase 기반 DB·인증·스토리지 연동
- 로그인 없는 환경에서도 신뢰도 높은 사용자 상호작용을 제공하기 위해
  브라우저 fingerprint 기반 사용자 식별 로직 설계
- 중복 좋아요 방지 및 상태 일관성 확보

인프라 & 트래픽 대응

- Nginx Reverse Proxy 기반 배포 환경 구성
  - 최근 30일 기준:
    - 총 요청 수 360K
    - 총 데이터 제공량 5GB
- Cloudflare CDN을 통한 정적 자산 캐싱 → 캐시 히트율 43.25%, 캐시 데이터 2GB 달성
- 이미지 트래픽 비중이 높은 특성을 고려해 CDN 전략을 단계적으로 고도화 중

- Shell Script 기반 Docker 이미지 자동 빌드·배포 파이프라인 구축
  - 프론트엔드와 백엔드 각각에 대해 Dockerfile 작성 및 멀티스테이지 빌드 최적화
  - 자체 작성한 Shell Script를 활용해 다음 프로세스를 단일 명령으로 자동화
    - Git Pull → 의존성 설치 → Docker 이미지 빌드 → 태깅 및 OCI 레지스트리 푸시
    - OCI Linux 서버에서 최신 이미지 자동 Pull → 컨테이너 재시작
  - 서버 접속 후 배포 작업을 직접 수행할 필요 없이, 로컬에서 스크립트 실행만으로 전체 배포 자동화
  - 배포 과정 표준화 및 인적 오류 최소화, 배포시간 단축