금융 서비스 장애 대응 관제 시스템 구축 (SSOK & SSOM) – 2025.04 ~ 2025.06

프로젝트 소개 백엔드 6명, 프론트엔드 1명, 인프라 1명

LG CNS AM INSPIRE CAMP 1기 최종 프로젝트로, **금융 서비스의 장애 발생 시 신속한 탐지 및 대응을 위한 관제앱과 장애 상황 재현을 위한 BLE 통신을 통해 송금할 수 있는 핀테크 송금앱**을 구현했습니다

- **Frontend** - React Native (Expo CLI), Typescript, Expo Router, Axios
- **State Management** – Zustand, Context API
- **Style** - Styled Components, Lottie, React Native Reanimated
- **IDE** - Cursor (Taskmaster, Context7)

기술 구현

- **[BLE 기반 근거리 송금 기능 구현]** - BLE 기반의 근거리 송금 기능을 핵심적으로 개발했습니다. iBeacon 프로토콜을 응용해 각 사용자의 기기가 고유 UUID를 포함한 신호를 광고(Peripheral Role)하도록 설계했으며, 다른 사용자는 이 신호를 스캔(Central Role)하여 주변 사용자를 탐지하여 별도의 계좌 번호 없이 송금할 수 있습니다.
- **[모듈화된 전역 상태 관리 아키텍처 설계]** - Zustand를 중심으로 전역 상태 관리 아키텍처를 설계하여 인증, 계좌 등 도메인별로 상태 로직을 Slice하고 비즈니스 로직의 응집도를 높였습니다. 이와 함께 Dialog와 같은 전역 UI 렌더링 및 제어에는 Context API를 병행하여 적용했습니다. 이처럼 데이터 상태(Zustand)와 UI 상태(Context)의 역할을 명확히 분리하여, 불필요한 리렌더링을 최소화하고 유지보수성이 뛰어난 상태 관리 구조를 구축했습니다.
- **[FCM 기반 푸시 알림 및 딥 링크 시스템 구축] -** Expo Notifications와 Firebase Cloud Messaging(FCM)을 연동하여 서비스의 주요 이벤트(ex. 송금 완료)를 사용자에게 알리는 푸시 알림 시스템을 구축했습니다. 알림 수신 및 상호작용을 처리하는 notificationService를 구현하여 로직을 중앙에서 관리하도록 했으며, 알림에 포함된 데이터 페이로드(data object)를 분석하여 expo-router와 연계하는 딥 링킹(Deep Linking) 기능을 구현했습니다.

핵심 기여

- **[팀장 및 프론트엔드 리드] -** 백엔드 개발자 6명, 인프라 담당자 1명으로 구성된 총 8명 규모의 팀을 이끌며, SSOK 및 SSOM 금융 서비스 앱 2종의 프론트엔드 개발을 전적으로 담당했습니다. 2개월이라는 촉박한 기한동안 프로젝트 일정 관리, 업무 분담, 주간 스프린트(1주 단위) 리뷰를 주도하며 프로젝트 목표를 달성했습니다
- **[전략적 아키텍처 설계] -** EXPO CLI를 도입하여 네이티브 환경 설정의 복잡성을 대폭 줄이고 개발 속도를 가속화했습니다. 추가로, 단독 개발 환경의 효율성을 극대화하기 위해, Redux 대신 학습 곡선이 낮고 보일러플레이트가 적은 Zustand를 선택하여 간결하고 효율적인 전역 상태 관리를 구축했습니다.
- **[복잡한 기술적 난관 극복] -** EXPO 환경의 BLE 통신 제약 극복을 위한 최적의 대안 라이브러리를 선정 및 구현했습니다. FCM 및 SSE를 활용한 실시간 알림 시스템을 성공적으로 설계 및 구축하여 안정적인 알림 기능을 제공했습니다.

성취

- JIRA 기준 **94%**라는 달성률을 통해 금융 서비스에 적합한 안정성과 사용자 경험을 갖춘 SSOK 앱과 SSOM 관제 시스템을 성공적으로 개발 완료했습니다
- LG CNS INSPIRE CAMP 1기 최종 프로젝트에서 **우수상**을 수상하여 기술적 구현 능력과 리더십, 문제 해결 능력을 동시에 입증했습니다

회고

[데이터 기반의 클라이언트 안정성 확보 한계]

- Sentry와 같은 실시간 에러 모니터링 시스템의 부재로**, BLE 기반 송금 기능처럼 기기 환경에 민감한 기능에서 발생하는 예측 불가능한 오류에 선제적으로 대응하지 못한 점은 아쉬움**으로 남습니다. 이로 인해 금융 서비스의 신뢰성과 직결되는 네트워크 및 인증 오류 발생 시, 사용자의 신고에 의존하는 사후 대응에 머무를 수밖에 없었고 이는 데이터 기반의 안정성 확보에 명확한 한계로 작용한다고 생각합니다.

하루에 하나 교양 지식 공유 웹 서비스 (HARUHAN) – 2025.02 ~ 2025.04

클릭! Github Website

프로젝트 소개 백엔드 2명, 프론트엔드 3명

매일 하나씩 이메일로 교양 지식을 전달하는 구독 기반 웹 서비스입니다. 사용자는 간단한 구독 절차를 통해 맞춤형 지식 콘텐츠를 메일로 받아볼 수 있으며, 북마크, 피드백, 개인 설정 등 다양한 편의 기능을 통해 콘텐츠를 효과적으로 관리할 수 있습니다

- **Frontend** - React (v18), React Router, Axios
- **State Management** – TanStack Query (v5)
- **Style** – CSS3, Emotion, Framer Motion, Lottie React
- **IDE** – VSCode

기술 구현

- **[이메일 기반 간편 인증 시스템 구축]** - 복잡한 회원가입 과정을 제거하고 이메일 파라미터 기반 인증 시스템을 핵심적으로 개발했습니다. `RouteGuard` 컴포넌트를 통해 URL 쿼리 파라미터에서 이메일을 추출하여 인증 여부를 판단하도록 설계했으며, 사용자는 별도의 로그인 절차 없이 이메일 링크만으로 개인화된 콘텐츠에 즉시 접근할 수 있습니다.
- **[React Query 중심의 서버 상태 관리 아키텍처 설계]** - React Query를 중심으로 서버 상태 관리 아키텍처를 설계하여 데이터 페칭, 캐싱, 동기화를 자동화했습니다. `queryKey`를 이메일 기반으로 구성하여 사용자별 데이터 격리를 보장했으며, `staleTime`, `enabled` 옵션을 전략적으로 활용해 불필요한 API 호출을 최소화했습니다.
- **[동적 미디어 콘텐츠 및 인터랙티브 UI 시스템 구축] -** 메인 히어로 섹션에 자동 전환 배경 비디오 시스템을 구현하여 사용자의 시각적 몰입도를 높였습니다. `useEffect`와 `setTimeout`을 활용한 타이머 기반 비디오 전환 로직을 구축했으며, Framer Motion과 Lottie 애니메이션을 전략적으로 배치하여 정적인 텍스트 기반 콘텐츠에 생동감을 부여했습니다.

핵심 기여

- **[프론트엔드 리드 및 팀원 기술 지원]-** 프론트엔드 개발의 **60%** 이상을 담당하며 프로젝트를 주도했습니다. React 컴포넌트 생명주기, 상태 관리, 비동기 처리 등 핵심 개념에 대한 지식을 팀원들과 공유하며 빠른 온보딩을 도왔습니다
- **[레거시 코드 리팩토링 및 컴포넌트 최적화]** - 팀원들이 AI를 활용하여 작성한 코드 중 컴포넌트 분리가 제대로 되지 않은 레거시 코드들을 체계적으로 리팩토링했습니다. 단일 책임 원칙(SRP)에 따라 거대한 컴포넌트를 기능별로 분해하고, 재사용 가능한 커스텀 훅으로 로직을 분리하여 코드의 가독성과 유지보수성을 크게 향상시켰습니다.
- **[AWS SES 연동 및 이메일 알림 시스템 구축] -** 사용자에게 매일 지식 콘텐츠를 전달하기 위한 AWS SES연동을 구현했습니다. 백엔드 팀과 협업하여 이메일 템플릿 디자인 및 설계부터 발송 로직까지 전체 이메일 시스템 아키텍처를 구축했습니다

성취

- 서비스 런칭 후 약 **100+** 명의 사용자를 확보하여 실제 운영 환경에서의 안정성을 검증했습니다.

회고

[Lighthouse 결과 및 성능 최적화의 필요성]

- Lighthouse 성능 분석 결과, **전체적으로 77점의 성능 점수를 기록했습니다**. 이는 사용 가능한 수준이지만, 사용자 경험 향상을 위해 개선이 필요한 영역들을 확인했습니다
- 현재 9,139KiB의 리소스를 매 요청 시 전달하고 있으며, 정적 리소스에 대한 브라우저 캐싱 전략이 부족하여 반복 방문 시에도 불필요한 재다운로드가 발생합니다. 향후 **CDN 도입**이나 적절한 **Cache-Control 헤더 설정**을 통해 이를 개선할 계획할 수 있다고 생각합니다
- 8,050KiB의 추가 절감이 가능한 상황입니다. 현재 사용 중인 이미지들을 **WebP 형식**으로 변환하고, 반응형 이미지 적용을 통해 디바이스별 최적화된 이미지를 제공할 필요성을 느꼈습니다

건강 식품 통합 관리 앱 HealingHub – 2024.08 ~ 2024.12

클릭! Github Playstore

프로젝트 소개 개인 프로젝트

건강기능식품을 판매하는 모바일 커머스 애플리케이션입니다. 사용자에게는 상품 카테고리별 탐색, 걸음 수 측정(만보기) 등 편의 기능을 제공하며, 관리자에게는 상품(추가, 수정, 삭제) 및 푸시 알림 발송 기능을 제공하여 효율적인 서비스 운영을 지원합니다

- **Frontend -** React Native (Expo), TypeScript, Expo Router, NativeWind, Axios
- **State Management -** React Context API
- **Backend -** Node.js, Express
- **Infrastructure -** AWS S3 (이미지 저장), AWS DynamoDB (데이터베이스)
- **IDE –** VSCode

기술 구현

- **[카테고리 기반 상품 탐색 시스템 구현]**- 건강기능식품을 비타민, 오메가3, 관절, 소화 등 7개 카테고리로 분류하여 사용자가 원하는 제품을 쉽게 찾을 수 있도록 했습니다. react-native-super-grid를 활용한 그리드 레이아웃과 인기순 정렬 기능을 구현했습니다.
- **[네이티브 기능 활용한 건강 관리 서비스]** - Expo Sensors의 Pedometer API를 활용하여 사용자의 걸음 수를 실시간으로 측정하는 만보기 기능을 구현했습니다. 단순한 커머스 앱을 넘어 사용자의 건강을 함께 관리할 수 있는 차별화된 커머스 서비스로 확장했습니다.
- **[관리자 전용 상품 관리 시스템 구축]** -관리자 전용 페이지를 구현하여 상품 추가, 수정, 삭제 기능을 제공했습니다. 이미지 업로드, 카테고리 선택 등 직관적인 UI를 통해 비개발자도 쉽게 상품을 관리할 수 있도록 설계했습니다.

핵심 기여

- **[풀스택 개발 경험 및 기초 아키텍처 구축]** - 프론트엔드(React Native)부터 백엔드(Node.js), 인프라(AWS)까지 전 영역을 직접 구현했습니다. 프로젝트 기획부터 배포까지의 전체 개발 사이클을 경험하며, 각 기술 스택 간의 연동 방식과 데이터 흐름을 이해할 수 있었습니다.
- **[Context API 기반 상태 관리 구조 설계]** - React Context API를 활용하여 ProductContext와 NotificationContext를 구현했습니다. 상품 데이터와 알림 상태를 전역적으로 관리하여 컴포넌트 간 Prop Drilling을 방지했으며, Provider 패턴을 통해 데이터와 UI를 분리하는 기본적인 아키텍처 설계 원칙을 학습했습니다.
- **[AWS 클라우드 서비스 연동 및 REST API 구축] -** Node.js와 Express를 사용해 상품 CRUD 기능을 제공하는 REST API를 구축했습니다. AWS S3를 연동하여 이미지 업로드/삭제 기능을 구현했으며, DynamoDB를 활용한 NoSQL 데이터베이스 설계 및 쿼리 작성을 경험했습니다.

성취

- 실제 운영 가능한 수준의 앱을 개발하고 **Google Play Store에 정식 출시**하며, 개발부터 배포까지의 전체 앱 개발 프로세스를 성공적으로 경험했습니다

회고

[상태 관리의 복잡성과 성능 최적화 부족]

- Context API를 사용하면서 Provider 중첩과 불필요한 리렌더링 문제를 경험했습니다. 상품 목록과 로딩 상태가 하나의 Context에 포함되어 있어, 로딩 상태만 변경되어도 상품 목록을 보여주는 모든 컴포넌트가 리렌더링되는 비효율을 경험했습니다. 향후 프로젝트에서는 **Zustand나 Redux Toolkit** 같은 상태 관리 라이브러리 도입을 고려해야겠다고 생각했습니다.

[테스트 코드 부재로 인한 안정성 우려]

- 기능 구현에 집중하다 보니 Jest나 Testing Library를 활용한 테스트 코드 작성을 소홀히 했습니다. 특히 상품 추가/수정 시 예상치 못한 에러가 발생했을 때 디버깅에 많은 시간을 소요되었으며, 이후 프로젝트에서는 **테스트 주도 개발(TDD)**의 중요성을 깨달았습니다.