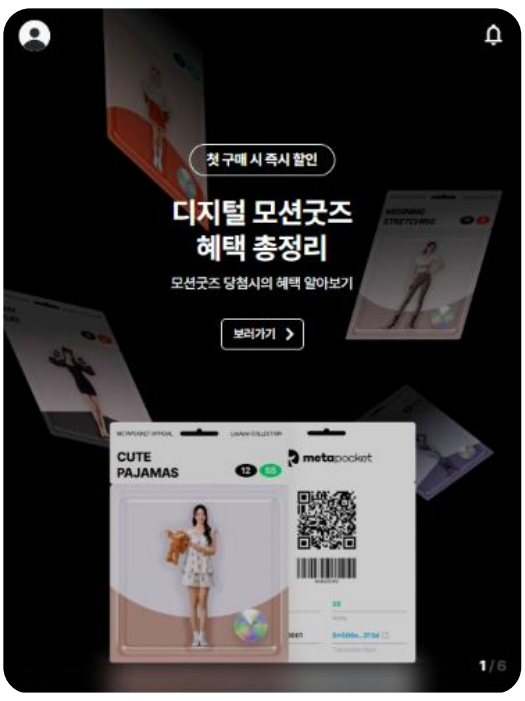
웹표준과 접근성을 고려한 UI 구현에
강점이 있는 프론트엔드 개발자 박민혜입니다.

박민혜 / minhyepark.dev@gmail.com



📌 프로젝트명 : 메타포켓 (사이트 바로가기)

📅 진행기간 : 2022.04 ~ 2024.08

💼 역할 : 프론트엔드 개발자

🛠️ 기술스택 : React, TypeScript, Zustand, React-Query, Axios, Emotion

🌐 프로젝트 소개

인플루언서와 협업하여 희소성 있는 디지털 굿즈를 제작하고,
이를 다양한 등급으로 구성된 랜덤 박스를 통해 제공하는 플랫폼입니다.
유저들은 랜덤 박스를 개봉하여 굿즈를 획득하고, 희망 금액 거래 및 즉시 거래 등
다양한 방식으로 자유롭게 거래할 수 있습니다.
이를 통해 인플루언서는 새로운 수익 모델을 창출하고, 팬들은 수집과 교환의
재미를 경험할 수 있는 디지털 굿즈 마켓을 제공합니다.

🚀 담당 역할 및 기여

1. 웹사이트 프론트엔드 개발 및 퍼블리싱

- 프로젝트의 개발 환경을 구성하고, 공통 UI 스타일 가이드 및 컴포넌트 시스템 구축으로 유지보수성을 높이고 개발 생산성 향상
- Axios 및 React-Query, Zustand를 활용한 API 연동 및 글로벌 상태 관리
- 결제 시스템 프론트엔드 연동을 담당하여 사용자 친화적인 결제 경험 구현
- 디지털 굿즈 트레이드 기능 개발을 통해 유저 간 거래 지원
- 한국어, 영어 다국어 처리

2. 관리자 페이지(1.0) 개발

- 관리자 페이지 기획 및 공통 UI 스타일 가이드에 맞춰 디자인 및 컴포넌트 구축
- Axios를 활용한 서버 API 연동 및 데이터 처리를 구현하여 운영 효율성 향상
- Nest.js 기반 관리자 페이지용 API 개발을 담당하여 일부 백엔드 기능 구현

🎯 주요 성과

✅ 결제 및 트레이드 기능 개발을 통해 사용자 중심의 거래 환경 구축
✅ 공통 컴포넌트 및 상태 관리 최적화로 유지보수성과 확장성 향상
✅ 관리자 페이지(1.0) 개발을 통해 운영 효율성 개선 및 관리 기능 강화

⏳ 어려웠던 부분

결제 연동 구현 시 어려움 발생

- 국내 PG사 API 문서를 기반으로 백엔드 개발자와 협업하여 결제 기능을 구현했으나, PG사마다 결제 요청 시 전달해야 하는 데이터 구조가 달라 이를 일관되게 처리하는 데 어려움이 있었고, 두 개의 PG 결제 시스템을 연결하는 과정에서 API 요청 방식과 응답 데이터 형식이 다르다 보니 통합하는 데 시간이 걸렸다.

해결 방법
- PG사별 요청 및 응답 데이터 구조를 비교 분석하여 공통된 인터페이스를 설계
- 백엔드 개발자와 협업하여 각 PG사의 결제 요청을 처리할 수 있도록 API 로직을 조정
- 테스트 환경에서 다양한 결제 시나리오를 검증하며 예상치 못한 오류를 사전 대응

결과
- 두 개의 PG사 결제 시스템을 원활하게 연동하여 결제 기능을 안정적으로 구현
- 다양한 결제 시나리오에 대한 테스트를 거쳐 실제 결제 성공률을 95% 이상으로 유지
- API 설계를 개선하여 향후 추가적인 PG사 연동 시에도 확장성을 고려한 구조로 구성

📝 회고

🔷 프로젝트 회고

이 프로젝트에서는 프론트엔드 개발부터 API 연동, UI/UX 개선까지 폭넓은 역할을 수행하며, 기술적 성장뿐만 아니라
사용자 경험과 운영 효율성을 고려한 개발이 얼마나 중요한지 배울 수 있었다.
처음에는 단순히 기능을 구현하는 데 집중했지만, 프로젝트를 진행하면서 코드의 유지보수성과 확장성을 고려하는 것이 중요하다는 것을 깨달았다. 특히, 공통 UI 스타일 가이드와 컴포넌트 시스템을 구축하면서 개발 생산성이 크게 향상되었고,
이후 기능 추가 및 유지보수가 훨씬 용이해졌다.

🔷 도전과 해결 과정

✅ 결제 시스템 연동의 어려움
결제 시스템을 연동할 때, PG사마다 요구하는 데이터 형식이 달라 이를 통합하는 데 어려움을 겪었다.
또한, 테스트 환경과 실제 운영 환경에서 API 응답이 다르게 동작하는 문제가 발생하여 예상치 못한 오류를 마주했다.
이 과정에서 백엔드 개발자와 긴밀히 협업하며 API 요청 및 응답 처리 방식을 조정했고, 그 결과, 결제 성공률을 안정적으로 유지하면서도,
다양한 결제 시나리오에 대응할 수 있는 구조를 마련할 수 있었다.

✅ 상태 관리 최적화
프로젝트 초기에는 API 호출 및 상태 관리가 분산되어 있어 데이터 동기화가 어려웠다.
이를 개선하기 위해 Zustand와 React-Query를 활용한 글로벌 상태 관리 패턴을 적용했고,
API 요청 최적화 및 캐싱을 통해 로딩 속도를 단축할 수 있었다.

✅ 관리자 페이지 기획 및 개발
관리자 페이지를 개발하면서, 단순한 기능 구현이 아니라 운영팀이 효율적으로 데이터를 관리할 수 있도록
UI/UX를 설계하는 것이 중요하다는 점을 배웠다. 기획부터 개발까지 주도적으로 참여하며 사용자 중심의 인터페이스를 구현했다.

🔷 배운 점 & 개선할 점

🌱 배운 점
- 컴포넌트 시스템과 상태 관리 최적화를 통해 유지보수성을 높이는 것이 중요함
- API 연동 시 백엔드 개발자와의 협업 및 문서화의 필요성을 경험함
- UI/UX 기획이 개발만큼 중요하며, 운영자의 사용성을 고려한 설계가 필수적임

🔍 앞으로의 개선점
- 결제 및 API 연동 시 다양한 예외 케이스를 사전에 정의하고 대비할 것
- 테스트 자동화 도입을 통해 기능 변경 시 발생할 수 있는 오류를 줄일 것
- 프로젝트 초기에 디자인 시스템과 상태 관리 패턴을 명확하게 정리하여 개발 일관성을 유지할 것

🎉 마무리

이 프로젝트를 통해 단순한 기능 구현을 넘어, 사용자 경험과 유지보수성을 고려한 개발이 얼마나 중요한지 깊이 이해할 수 있었다.
앞으로도 더 나은 코드 구조와 협업 방식을 고민하며 성장하는 개발자가 되고 싶다.