Global Software Engineering Project

WooZoo Application Document

https://github.com/MyKnow/woju.git

https://github.com/MyKnow/woju_backend.git

김도익, 임석범, 정민호, 조민혁

Mobile System Engineering

32217072, 32203743, 32194074, 32204292

목 차

1. 프로젝트 개요

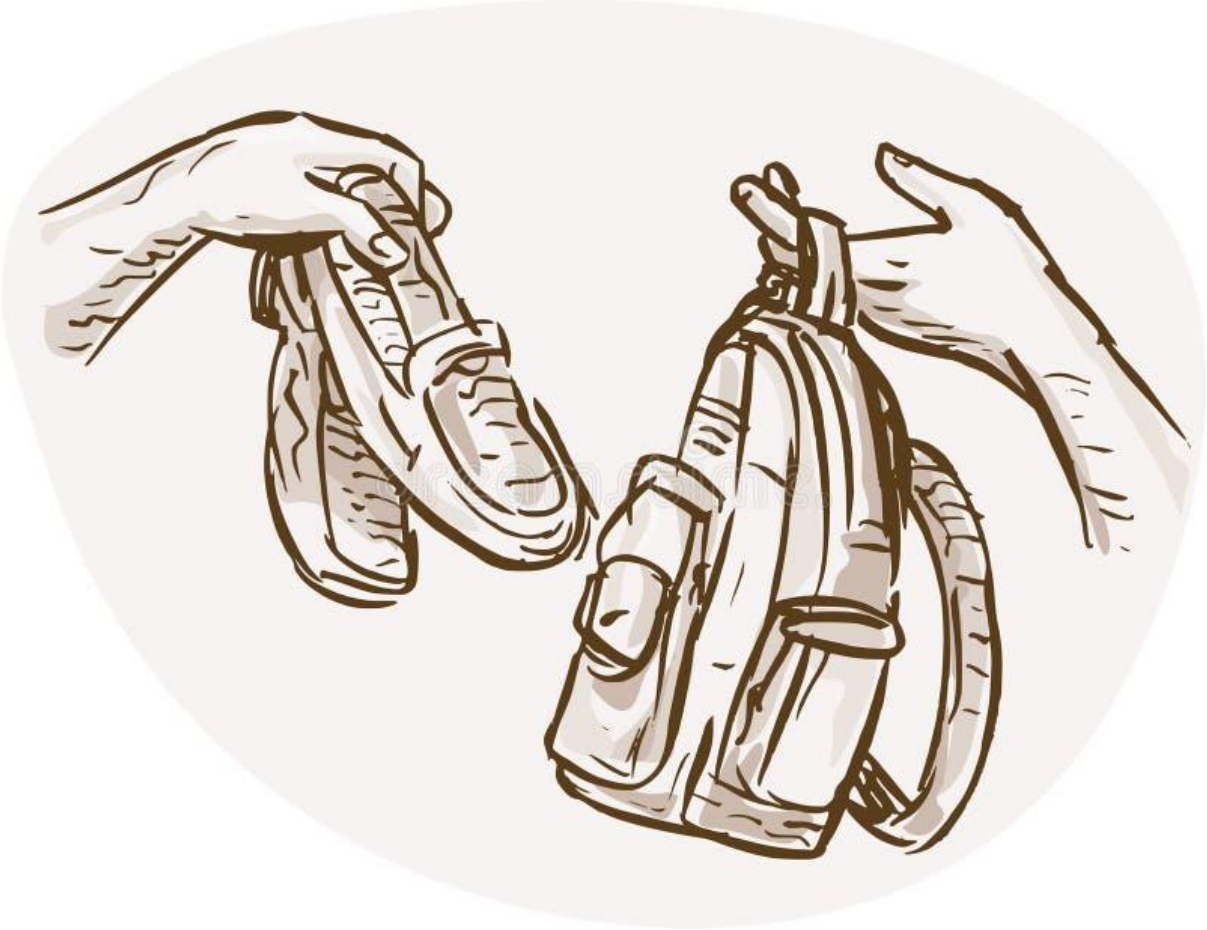

그림 1. 물물교환 개요

현대 사회에서 모바일 애플리케이션과 웹 애플리케이션의 사용이 급증하면서 다양한 서비스 플랫폼이 등장하고 있다. 특히 의류, 신발, 악세서리, 생필품과 같은 상품을 중심으로 한 온라인 마켓 시장은 오프라인 시장을 압도하고 있으며, 대표적인 플랫폼으로 쿠팡, 무신사, 당근마켓 등이 있다. 이러한 플랫폼은 판매자와 구매자 간의 거래를 기반으로 물품을 제공하고 금전을 지불하는 형태로 운영되며 대부분 금전 거래 중심의 서비스만을 제공하고 있는 것이 현실이다.

그러나 물물 교환 서비스에 대한 수요는 분명 존재하지만, 해당 시장은 활성화되지 못한 채 이용자도 매우 제한적인 상황이다. 이에 본 프로젝트는 기존 거래 중심의 플랫폼에서 벗어나 물물 교환의 활성화를 목표로 '물물 교환 서비스 애플리케이션'을 개발하고자 한다. 기존 플랫폼들이 제공하지 못했던 물물 교환 기능을 강조하면서, 나아가 재능과 같은 무형 자원까지 교환할 수 있는 기능을 통해 차

별화된 가치를 제공한다.

이 애플리케이션은 빅데이터와 머신러닝 기술을 기반으로 사용자 맞춤형 기능을 도입했다. 사용자의 과거 게시글 데이터를 분석하여 선호하는 카테고리를 도출하고 이를 바탕으로 교환 가능한 물품이나 재능을 추천하는 시스템을 구현한다. 교환 과정을 통해 얻은 첫 번째 물품과 마지막 물품의 가치 차이를 분석하여 실질적인 교환 이익을 사용자에게 제공하는 결산 기능도 포함된다. 또한, 신제품 가격과 사용 기간, 사용감 등을 종합적으로 판단해 중고 물품의 적정 가치를 예측하고 유사한 가격대의 물품을 추천하는 기능을 제공한다. 이를 통해 물물 교환의 효율성을 높이고 사용자 경험을 극대화한다.

물품 교환에 그치지 않고 재능과 같은 무형 자원의 교환 기능을 도입함으로써 플랫폼의 활용 범위를 확장하였다. 이는 기존 거래 중심의 서비스와 차별화되며 물물 교환 서비스 시장의 새로운 가능성을 제시한다.

본 프로젝트 문서는 프로젝트의 전체 구성과 특징을 설명하는 것으로 시작한다. 물물 교환 서비스 애플리케이션의 필요성과 목표를 명확히 하고 이를 시각 자료와 함께 제시한다. 시스템의 세부 구조를 설명하기 위해 프론트엔드와 백엔드의 구현 방식을 UML 다이어그램과 함께 구체적으로 소개하며, 이를 통해 기술적 이해를 돕는다. 또한 구현한 애플리케이션의 소프트웨어 퀄리티를 제시함으로써 소프트웨어의 가치를 평가한다.

본 프로젝트는 물물 교환 서비스를 활성화하고 사용자 간의 상호작용을 증진하는 한편, 기존의 금전 거래 중심 플랫폼과 차별화된 새로운 가치를 제공하고자 한다. 이를 통해 물물 교환 시장의 새로운 패러다임을 제시하며, 사용자 경험을 혁신적으로 변화시키는 것을 목표로 한다.

2. 프로젝트 전체 구성 및 특징

본 프로젝트의 백엔드는 모노레포(Monorepo) 기반으로 여러 서비스를 통합적으로 관리하며, 서비스별 독립성을 유지하는 멀티 서버 아키텍처를 채택하였다. 위 구조는 코드 재사용성과 개발 생산성을 극대화하고 서비스간 협업과 운영을 체계화할 수 있도록 설계되었다.

멀티 서버 아키텍처는 여러 서버가 협력하여 하나의 시스템을 구성하는 방식이다. 단일 서버에서 모든 작업을 처리하는 단일 서버 아키텍처와 다르게, 작업을 분산하고 서버간 역할을 분리하여, 시스템의 성능, 안정성, 확장성을 높일 수 있다. 해당 프로젝트에서는 각 서버를 도메인 기반으로 분리하였으며, server_chat, server_item, server_user 와 같이 특정 도메인에 집중하도록 설계하였다.

모노레포 구조는 단일 저장소에서 여러 서비스를 관리하는 구조이다. 모노레포는 각 프로젝트나 라이브러리의 각자의 저장소를 가지는 멀티 레포와 대조된다. 모노레포는 모든 프로젝트의 라이브러리를 하나의 저장소에 저장하여 변경 사항을 관리할 수 있다. 모노레포는 다음과 같은 장점을 가진다.

① **코드의 재사용성:** shared 디렉토리를 통해 중복되는 모델 및 로직을 한곳에서 관리하여 유지 보수성을 개선한다.

② **일관적인 개발 환경:** 모든 서비스가 같은 종속성 관리 도구와 빌드 시스템을 공유하며, 이에따라 일관적인 개발 환경이 제공된다.

③ **효율적인 종속성 관리:** pnpm-workspace.yaml 을 통해 서비스간 종속성을 효율적으로 관리한다.

이 프로젝트에서 Package 디렉토리의 여러 Server 디렉토리는 각 서비스의 소스 코드와 공용 리소스를 함께 관리한다. 아래는 각 서비스별 소스 디렉토리의 구조이다.

① server_chat: 채팅 기능을 담당하는 서비스

② server_item: 아이템(상품, 콘텐츠) 관리 관련 서비스

③ server_user: 사용자 정보 관리 및 인증 서비스

④ shared: 여러 서비스에서 공통으로 사용되는 모델, 유틸리티 함수, 데이터 타입 정의 등을 포함한 공용 모듈

위 멀티 서버 아키텍쳐와 모노레포 구조가 효율적으로 운영되기 위해서는 자동화된 CI/CD 파이프라인을 사용하여야 한다. 해당 프로젝트에서는 GitHub Acrtions와 자동화된 스크립트를 통해 개발 및 프로세스를 체계화한다. Github/workflows에 정의된 CI/CD 설정을 통해 코드 변경시 자동으로 빌드, 테스트, 배포 작업을 수행할 수 있다.

마지막으로, 환경변수와 민감한 정보는 암호화된 파일을 통해 관리가 된다. 회원 약관과 서비스 약관에 따라 분류 할 수 있는 민감 정보는 .env.enc와 .env.sha256 을 사용하여 환경 변수를 암호화 하며, 무결성 검증을 통해 보안이 강화된다. 위와 같은 보안 관리는 민감 정보 노출을 방지하며, 안전한 운영 환경을 제공하도록 기여한다.

본 프로젝트의 프론트엔드 코드는 모듈화된 구조를 기반으로 UI, 데이터 모델, 상태 관리를 기능별 폴더로 체계적으로 분리하여 관리한다. 앱의 상태를 효율적

으로 관리하고, 공통 위젯을 활용해 코드 중복을 최소화하며 유지보수성과 재사용성을 높인 UI설계를 구현하였다. 또한 라우팅과 입력 포커스 관리가 정교하게 이루어져 있으며, 사용자 경험을 개선하기 위한 다양한 기능을 제공한다. 이를 통해 확장성과 유지보수에 장점을 가진 구조를 갖추고 있다.

본 프로젝트의 프론트 엔드는 다음과 같은 구조로 구성 되어있다.

① Model: 앱의 데이터 구조와 상태 관리를 정의한다. 물품 정보와 사용자 데이터를 관리하고, 앱 설정과 상태(로딩, 오류 등)를 담당한다.

② Page: 앱의 화면 출력과 사용자 상호작용을 담당한다. 각 페이지는 사용자의 행동에 따라 데이터를 표시하고 입력을 처리하며, 화면 전환과 함께 앱의 주요 기능을 제공한다. 이를 통해 사용자가 물품 등록, 거래 확인, 로그인/회원가입과 같은 작업을 수행할 수 있도록 UI와 연결하는 역할을 한다.

③ Provider: 앱의 상태관리와 데이터 흐름을 담당한다. 화면 전반에 걸친 상태(앱 상태, 라우팅, 입력 포커스, 테마)를 관리하며, 이를 통해 UI와 비지니스 로직을 연결한다.

④ Service: 앱 사용시 핵심기능을 지원하는 유틸리티 로직을 담당한다.

⑤ Theme: 앱의 디자인과 스타일을 정의하는 역할을 담당한다. 앱의 전체적인 색상, 폰트, 위젯 스타일을 설정하여 일관된 UI를 제공한다.

3. 프로젝트 세부 구성

3-1. Front End

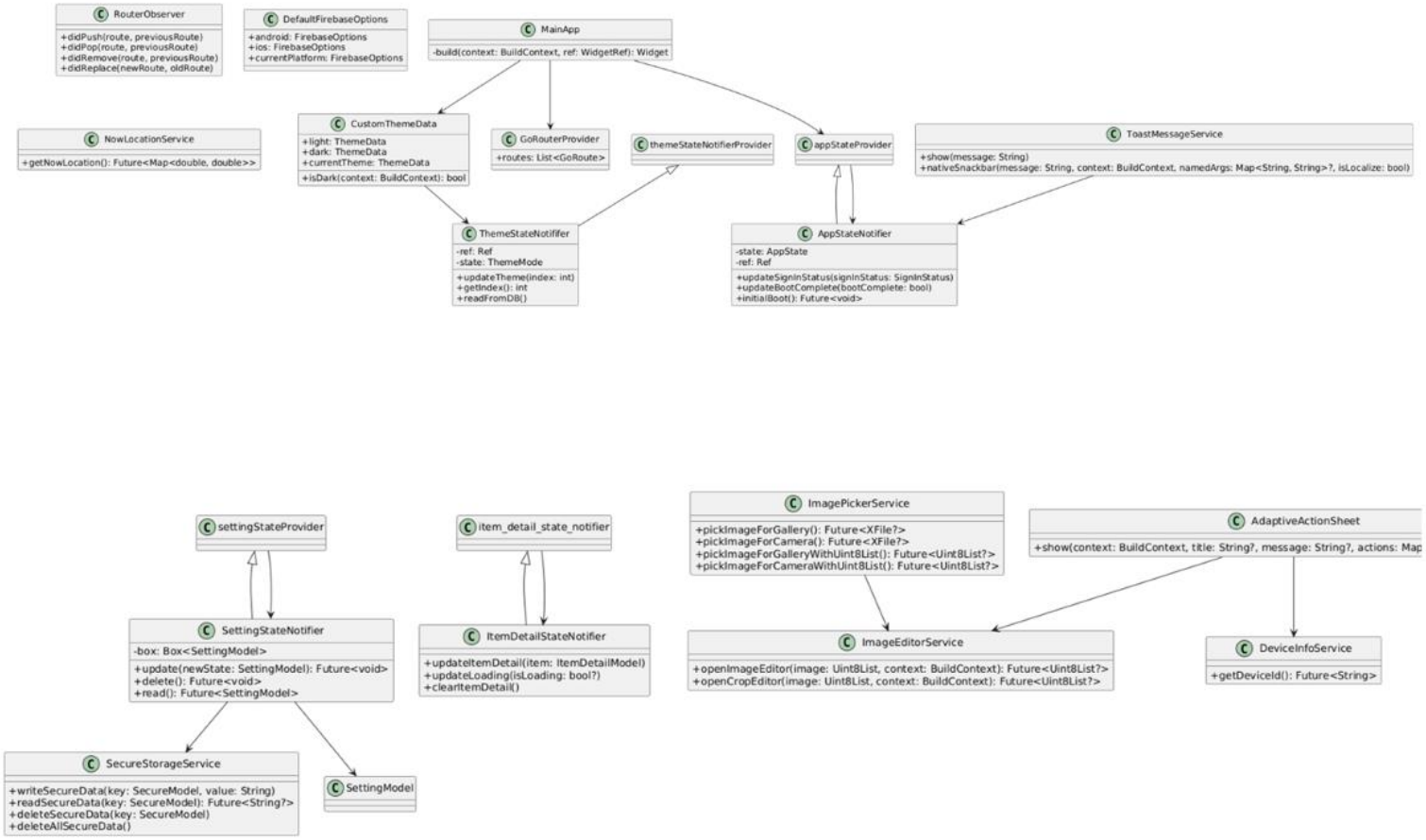

그림 2. UML Diagram - Front-End

3-1-1. model/item

Item 디렉토리는 물물교환 앱의 상품 데이터 모델과 관련된 핵심 로직을 담당하며, 상품의 정보, 카테고리, 거래 장소를 체계적으로 관리하고 유효성 검사와 상태 업데이트를 제공한다. 이를 통해 상품 등록, 수정, 조회와 같은 물물교환 앱의 핵심 기능을 뒷받침하는 데이터 관리 및 로직 처리의 중심 역할을 한다.

① **상품 데이터 관리:** ItemModel과 ItemDetailModel을 통해 물품의 기본 정보와 상세 정보를 구조화한다.

② **유효성 검사:** 상품 이름, 설명, 가격, 이미지 리스트 등의 유효성을 검사하

여 데이터의 무결성을 유지한다.

③ **거래 장소 관리:** Location 모델을 사용해 거래 장소의 간단한 이름, 전체 주소, 위도/경도 정보를 저장하고 유효성을 검사한다.

④ **상태 관리:** AddItemStateModel을 통해 상품 등록 시 입력된 상태와 컨트롤러를 관리하며, 수정 모드 여부를 확인한다.

⑤ **카테고리 관리:** CategoryModel과 Category Enum을 활용해 물품 카테고리를 정의하고, 로컬라이징된 이름과 이미지 아이콘을 제공한다.

⑥ **데이터 변환:** 모든 모델은 JSON 형태로 변환 및 역변환 기능을 지원하여 서버와의 통신 및 데이터 저장이 용이하다.

⑦ **상태 업데이트:** copyWith()메서드를 통해 변경된 상태를 기반으로 새 객체를 생성하여 데이터의 불변성을 유지한다.

⑧ **데이터 표시:** 사용감, 가격, 이미지 리스트와 같은 데이터를 UI에 표시하기 위해 포맷된 문자열이나 아이콘을 제공한다.

⑨ **상세 정보 처리:** 서버에서 받아온 데이터를 ItemDetailModel로 관리하고, 생성 날짜나 상태를 문자열로 변환하여 표시한다.

3-1-2. model/onboarding

Onboarding 디렉토리는 회원 인증 및 초기 사용자 등록과정을 담당한다.

① **로그인 처리(SignInModel):** 사용자 입력(전화번호, 비밀번호, 사용자 ID)을 관리한다. 로그인 시도 후 상태(성공, 실패, 서버 오류 등)를 관리하기 위해 SignInStatus를 통해 상태를 정의한다. 입력된 로그인 정보를 JSON으로 변환하여 서버와 통신할 수 있도록 준비한다. 로그인 성공 여부와 상태를 통해 UI에 반영할 수 있는 로직을 제공한다.

② **회원 가입 처리(SignUpModel):** 회원가입 시 필요한 사용자 데이터(전화번호, 비밀버번호, ID, 닉네임, 성별, 생년월일)를 관리한다. 프로필 이미지, 약관 동의 상태 등 회원가입에 필요한 부가 정보를 포함하고 있고, SignUpError를 통해 인증 오류, 서버 오류, 중복 가입 여부 등 다양한 회원가입 실패 원이늘 정의하고 처리한다. JSON 변환 기능을 통해 사용자 데이터를 서버로 전송하여 회원가입 프로세스를 완료한다.

③ **유효성 검사 및 데이터 초기화:** 초기 상태 제공(initial): 필드에 대한 기본 값을 제공해 로그인/회원가입 화면의 초기 상태를 설정한다. CopyWith 메서드를 활용하여 데이터 상태를 업데이트하거나 특정 필드를 초기화한다. 회원가입 시 프로필 이미지나 비밀번호 변경 여부를 관리한다.

④ **디바이스 정보 연동:** DeviceInfoService를 사용해 사용자의 장치 고유 ID를 가져와 로그인이나 회원가입 데이터에 포함시킨다.

3-1-3. model/status

Status 디렉토리는 앱의 상태관리와 버전 관리를 담당하는 역할을 한다. 사용자의 활동 상태, 시스템의 버전 상태 등 앱의 동작과 흐름을 제어하는 상태 정보를 관리하는 모델과 믹스인을 제공한다.

StatusMixin을 통해 다양한 상태(오류, 성공)를 로컬 라이징된 메세지로 변환하여 UI에 표시한다. Version 클래스는 현재 버전, 최신 버전, 필수 버전 정보를 관리하며, 업데이트 필요여부를 판단한다. 이를 통해 앱의 상태 흐름을 제어하고 유지보수성을 높이는 역할을 하며, 사용자에게 일관된 상태 메시지와 원활한 버전 관리를 제공한다.

3-1-4. model/user

User디렉토리는 앱의 사용자 관련 데이터와 상태 관리를 담당하는 모듈이다. 이 디렉토리는 사용자 인증, 프로필 관리, 비밀번호, 닉네임, 전화번호와 같은 사용자 관련 정보를 관리하고, 다양한 유효성 검사와 상태 처리를 제공한다. User에서 제공하는 기능은 다음과 같다.

① **사용자 인증 및 상태 관리:** 사용자 인증 코드 발송, 검증, UID 관리 등의 기능을 제공하며, 이를 통해 사용자가 앱에 안전하게 로그인하거나 회원가입을 수행할 수 있도록 한다. UserAuthModel 은 인증 상태와 관련된 유효성 검사 및 상태 업데이트를 담당하며, AuthStatus와 같은 상태를 활용해 다양한 인증 결과를 관리한다.

② **사용자 프로필 관리:** 사용자 닉네임, 성별, 생년월일, 프로필 이미지와 같은 정보를 관리하며, 이를 편리하게 수정할 수 있는 상태 관리 기능을 제

공한다. UserProfileEditState는 사용자 프로필 수정 중 발생하는 모든 데이터와 상태를 관리하며, 백업 데이터를 유지하여 변경 전후를 비교할 수 있게 한다.

③ **유효성 검사 및 상태 메시지 처리:** 사용자 닉네임, 비밀번호, 전화번호와 같은 입력 데이터를 유효성 검사하고, 이를 사용자에게 적절한 상태 메시지로 반환한다. StatusMixin과 연동하여 입력값의 상태를 UI에 표시한다.

④ **비밀번호 관리 및 변경:** UserPasswordModel 은 비밀번호의 강도와 유효성을 검사하며, 비밀번호 변경과 관련된 로직을 제공한다. UserPasswordChangeModel은 현재 비밀번호와 새 비밀번호를 비교 및 업데이트하는 기능을 지원한다.

⑤ **유저 세부 정보 관리:** UserDetailInfoModel 은 사용자의 고유 UUID, 국가 코드, 즐겨찾기 카테고리, 토큰 등 상세정보를 관리한다. 이 데이터는 앱의 개인화된 UX를 제공하는데 사용된다.

⑥ **데이터 직렬화 및 API 통신 지원:** 대부분의 모델은 JSON 직렬화 및 역직렬화 기능을 포함하여 API와 데이터를 효율적으로 주고받을 수 있도록 설계되었다.

3-1-5. model/app_state_model

앱의 전역 상태를 관리하는 역할을 한다. 사용자 로그인 상태(signInStatus), 앱 초기화 여부(isBootComplete), 앱 오류 상태(appError)등을 포함한다. 기본 상태 값을 설정하고(initialState), copyWith를 통해 상태를 업데이트 하는 기능을 제공한다.

3-1-6. model/Hive_Box_enum

Hive 데이터베이스에서 사용할 박스와 어댑터를 관리한다. 사용자 정보, 설정, 성별 등의 데이터를 영구적으로 저장하고 불러오도록 지원한다. 박스를 열거나 어댑터를 등록하는 기능을 제공한다. 저장소 데이터를 효율적으로 관리하고, 저장된 데이터를 쉽게 접근할 수 있도록 설계되었다.

3-1-7. model/secure_model

앱의 보안 모델을 정의한다. 각 보안 모델에 이름을 부여하거나 문자열에서 모델로 변환하는 기능을 제공한다. 앱의 민감한 데이터를 처리하거나 저장할 때 사용된다.

3-1-8. model/server_state_model

서버 연결 상태를 정의하고, 각 상태에 대응하는 메시지를 제공한다. 서버가 온라인, 오프라인, 연결 중 또는 오류 상태일 때의 상태를 메시지를 반환하여 사용자에게 현재 서버 상태를 전달한다.

3-1-9. model/text_field_model

텍스트 필드의 유효성 검사를 포함한 입력 데이터를 관리하는 기본 믹스인이다. 사용자 입력 값, 유효성, 라벨, 에러 메시지를 통합적으로 관리한다. 입력값을 검사하고 유효하지 않으면, 에러 메시지를 반환하거나 유효성 검사를 수행한다. 다른 모델에서 공통적으로 사용하는 텍스트 필드 관리 로직을 캡슐화하여 재사용성을 높였다.

3-1-10. page/error

Error 디렉토리는 앱에서 발생하는 에러 상황을 사용자에게 안내하고 문제를 해결할 수 있도록 지원하는 페이지를 제공한다. 라우팅 에러와 서버 에러 상황을 처리하며, 사용자 경험을 향상시키는 데 중요한 역할을 한다.

RouterErrorPage는 잘못된 경로로 이동하거나 정의되지 않은 페이지에 접근했을 때 호출된다. 이 페이지는 사용자에게 라우팅 에러 발생 사실을 알리고, 앱의 초기 화면으로 돌아갈 수 있는 뒤로가기 버튼을 제공한다. 버튼을 누르면 앱의 루트 경로로 이동하도록 설계되어, 사용자가 에러 상황에서 앱의 정상적인 사용 상태로 쉽게 복귀할 수 있다.

ServerErrorPage는 서버와의 연결이 실패했을 때 호출되는 페이지다. 이 페이지는 서버 에러 상황을 사용자에게 명확히 알리고, 재시도 버튼을 통해 서버 연결 상태를 다시 확인할 수 있는 기회를 제공한다. 이 버튼은 서버 상태를 점검하고

복구 가능 여부를 확인하는 CheckServerConnection() 메서드를 호출하여 사용자 경험의 연속성을 유지한다.
두 페이지 모두 공통적으로 CustomScaffold와 Customtext를 사용하여 앱 전반의 디자인과 일관성을 유지한다. 또한, 뒤로 가기 버튼을 비활성화하여 불필요한 동작으로 인한 추가적인 에러 발생을 방지하며, 사용자에게 명확하고 직관적인 에러 메시지와 해결 방안을 제시한다.

3-1-11. page/home

Home 디렉토리는 물물교환 앱의 핵심적인 사용자 경험을 구성하는 주요 화면과 기능을 제공하는 중심 모듈이다. 이 디렉토리는 앱의 기본 구조를 정의하며, 네비게이션 바와 함께 주요 탭들을 통해 사용자에게 직관적이고 효율적인 화면 전환을 제공한다. MainPage는 홈, 매칭, 채팅, 그리고 내 물품과 같은 주요 탭을 포함하며, 각각의 탭은 사용자 정보를 확인하고, 추천된 물품을 탐색하거나, 채팅을 통해 소통하고, 사용자가 등록한 물품을 관리할 수 있는 기능을 제공한다. 또한, 버튼을 통해 새로운 물품을 쉽게 추가할 수 있는 기능을 제공하며, 전체 UI는 사용자의 행동에 따라 동적으로 변화하여 높은 유연성을 보장한다. 전반적으로 main 디렉토리는 물물교환 앱의 모든 주요 인터페이스와 상호작용의 중심 역할을 하며, 직관적이고 통합된 사용자 경험을 제공한다.

3-1-12. provider

provider 디렉토리는 물물교환 앱에서 상태를 중앙에서 체계적으로 관리하며, 로직을 화면과 분리하여 유지보수성과 확장성을 높인다. 이 디렉토리는 앱 전반의 상태를 효율적으로 처리하고, 비동기 작업을 안정적으로 관리하여 앱의 안정성과 성능을 향상시킨다.

앱의 각 모듈은 StateNotifier를 기반으로 상태를 관리하며, 이를 통해 UI와 상태 간의 결합도를 낮추고 동적인 상태 업데이트를 가능하게 한다. 예를 들어, SignUpStateNotifier는 회원가입 과정에서 전화번호 인증, 아이디 중복 체크, 비밀번호 설정 등의 작업을 수행하며, 사용자가 입력한 데이터를 실시간으로 검증

하고 상태를 업데이트한다. 또한, UserDetailInfoStateNotifier는 사용자 정보를 로컬 데이터베이스인 Hive를 통해 관리하며, 사용자 정보의 저장, 수정, 삭제 작업을 처리한다.

상태 업데이트는 사용자의 실시간 상호작용을 기반으로 동적으로 이루어지며, 이러한 작업은 비동기적으로 처리한다. 이를 통해 API 호출, 데이터베이스 작업과 같은 복잡한 비동기 작업을 효율적으로 관리한다. 예를 들어, SignUpStateNotifier는 Firebase를 활용한 전화번호 인증과 백엔드 서버와의 데이터 통신을 담당하며, 이를 통해 사용자 입력값을 검증하고 인증 절차를 진행한다. AppStateNotifier는 앱의 전반적인 상태, 예를 들어 부트 완료 여부나 서버 연결 상태를 관리하여 앱이 원활하게 작동하도록 지원한다.

또한, GoRouteProvider는 화면 간의 라우팅을 관리하며, 사용자 경험의 흐름을 매끄럽게 만든다. UI는 StateNotifierProvider를 통해 상태를 구독하고, 상태 변경 사항에 따라 자동으로 업데이트되도록 설계한다. 사용자의 입력이나 이벤트에 따라 화면 요소가 즉각적으로 반응하며, 이를 통해 사용자 경험을 개선한다.

결론적으로, provider 디렉토리는 상태 관리, 비즈니스 로직 처리, 비동기 작업, 화면 전환 관리 등의 기능을 통합적으로 수행하며, 앱의 전반적인 사용자 경험과 성능을 높인다. 이를 통해 개발과 유지보수를 단순화하고, 확장 가능하며 안정적인 앱 구조를 제공한다.

3-1-13. service

서비스 디렉토리는 앱에서 사용자 경험을 향상시키고 앱의 주요 기능을 지원하는 로직을 담당한다. 이 디렉토리는 다양한 서비스 모듈을 통해 앱의 안정성과 보안, 편의성을 강화하며, 각 서비스는 서로 유기적으로 작동해 사용자에게 원활한 앱 사용 환경을 제공한다. Service에서 제공하는 기능은 다음과 같다.

① 플랫폼에 맞춘 적응형 액션 시트를 제공하여 사용자 인터페이스를 최적화한다. 이를 통해 iOS와 Android에서 모두 직관적이고 일관된 사용자 경험을 제공한다.

② 디버깅과 로깅을 지원하여 개발 과정에서 오류를 빠르게 확인하고 수정할

수 있도록 돕는다. 이를 통해 앱의 안정성을 유지하고 유지보수 작업을 효율적으로 수행할 수 있다.

③ 디바이스 정보를 수집하여 사용자 기기에 맞는 맞춤형 동작을 가능하게 한다. 예를 들어, 사용자의 디바이스 종류와 OS에 따라 최적화된 기능을 제공한다.

④ 사용자 물품의 이미지 처리를 담당한다. 사용자는 이 서비스를 통해 물품 이미지를 선택, 편집, 업로드할 수 있어 등록 프로세스가 간소화된다.

⑤ 위치 기반 서비스를 제공하여 거래 장소를 설정하거나 주변 추천 장소를 제안하는 데 사용된다. 이를 통해 사용자 간의 거래를 더욱 편리하게 지원한다.

⑥ 사용자 데이터 보안을 강화하며, 민감한 정보(예: 비밀번호, 인증 토큰 등)를 안전하게 저장하고 관리한다. 이를 통해 데이터 유출 위험을 최소화하고 사용자 신뢰를 높인다.

⑦ 사용자 액션에 대한 즉각적인 피드백을 제공하여 앱의 응답성과 사용성을 개선한다. 예를 들어, 물품 등록 성공, 오류 메시지 등을 실시간으로 전달한다.

3-1-14. theme

theme 디렉토리는 물물교환 앱의 시각적 스타일과 사용자 경험(UX)을 체계적으로 관리하며, 앱의 전반적인 디자인 일관성을 유지한다. CustomThemeData를 통해 라이트 모드와 다크 모드의 테마를 정의하고, 색상, 텍스트 스타일, 버튼과 카드의 디자인을 통합적으로 관리한다. 이를 통해 사용자 환경에 맞는 최적의 시각적 경험을 제공하며, 앱의 품질과 신뢰성을 높인다.

또한, 이 디렉토리는 플랫폼에 따라 다르게 렌더링되는 적응형 다이얼로그와 버튼을 제공하며, 사용자 친화적인 디자인을 구현한다. 나라 코드 선택기, 날짜 선택기와 같은 커스터마이징된 위젯을 포함하여 사용자 작업을 간소화하고, 다국어 지원 기능을 통해 다양한 언어와 지역 설정에 유연하게 대응한다. 이를 통해 사용자가 앱을 더 직관적이고 편리하게 사용할 수 있도록 지원한다.

결론적으로, theme 디렉토리는 물물교환 앱의 디자인 정체성을 확립하고 사용자

중심의 인터페이스를 설계하며, 시각적 만족도를 높이는 동시에 앱의 유지보수성과 확장성을 보장한다.

3-2. Back End

본 프로젝트의 백엔드 시스템은 모노레포 기반으로 설계된 백엔드 시스템으로, 채팅 서비스, 아이템 관리, 사용자 관리를 중심으로 구성되어 있다. 또한, 각 서비스는 멀티 서버 아키텍처를 기반으로 구축되었으며, 공통 모듈을 통해 각 서버 간 독립성을 유지하면서 공통 모듈을 재사용하여 효율적인 개발 및 유지 보수가 가능하도록 설계하였다. 다음은 백엔드 부분의 UML과 이에 대한 세부 설명이다.

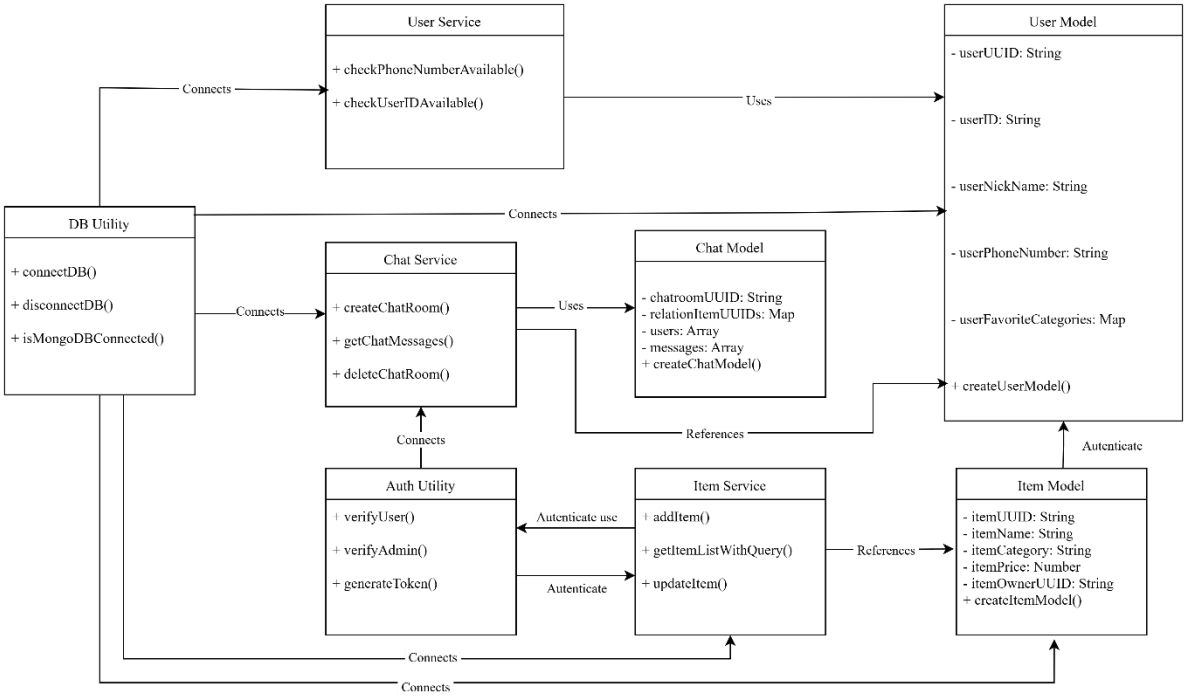

그림 3. UML Diagram - Back-End

3-2-1. service_chat

채팅 서비스는 사용자간 실시간 소통을 담당하는 모듈로 채팅방 생성, 메시지 전송, 조회 및 삭제 기능을 제공한다.

채팅방 생성 시 각 사용자와 아이템의 UUID를 기반으로 중복된 채팅방 여부를 검증하며, 존재하지 않는 경우 새 채팅방을 생성한다. 메시지 관리 기능에서는 채팅방에 포함된 메시지를 조회하고 전송된 메시지의 읽음 상태를 업데이트하는 역할을 수행한다. 메시지 전송시 상대 방의 읽음 여부를 기록하기 위해 seenUserUUIDS 필드를 활용하며, 채팅방 삭제 시 해당 채팅방을 데이터베이스에서 안전하게 제거한다. 위 기능의 모든 데이터는 chatDB 데이터베이스에 저장되며, 각 채팅방 및 메시지 구조는 chatModel을 통해 정의된다. 이 과정에서 사용자 및 아이템 정보의 유효성을 검증하기 위해 사용자 관리 서비스와 아이템 관리 서비스를 호출하며, Auth Utility를 통해 사용자의 인증 상태를 확인한다.

3-2-2. service_item

아이템 관리 서비스는 사용자 물품의 등록, 수정, 조회, 삭제 및 매칭기능을 중심으로 구성된다. 물품 등록시 사용자가 입력한 물품 이름, 설명, 이미지, 가격, 카테고리, 사용감 등을 검증하며, 각 등록된 물품은 itemDB 데이터베이스에 저장된다. 조회 기능은 다양한 필터링 및 정렬 옵션을 제공하여 사용자의 요구에 맞는 물품을 효율적으로 탐색할 수 있도록 지원한다. 필터링 조건으로는 가격 범위, 카테고리, 사용감 상태등을 제공하며, 사용자의 선호도 기반 추천시스템을 위해 카테고리 우선순위 설정 기능도 포함된다. 물품 매칭 기능에서는 두 사용자의 물품을 서로 연결하여, 매칭이 확정된 경우 itemMatchedUsers 필드에 상대방의 정보를 기록하고, 이를 바탕으로 채팅서비스를 제공한다. 물품 관리시 Auth Utility를 통해 사용자 인증이 수행되며, 물품 수정및 삭제 요청시 소유자 여부를 확인하는 로직이 포함된다.

3-2-3. service_user

사용자 관리 서비스는 사용자의 회원가입, 로그인, 정보 수정과 인증을 담당하는 핵심 모듈이다. 회원가입시, 사용자 입력값의 유효성을 검증하고 전화번호와 개인

정보의 중복성을 검사한다. 이를 위해, TempPhoneNumber와 TempUserID 모델을 사용하여 임시 데이터를 저장하며, 최종적으로 userDB에 사용자 정보를 저장한다. 위 로그인 기능에서 입력된 비밀번호와 암호화된 비밀번호를 비교 검증하며 인증을 성공하면, JWT 토큰을 발급한다. 사용자 정보 수정 기능은 닉네임, 프로필 이미지, 선호 카테고리와 같은 필드를 업데이트 하며, 저장된 데이터는 데이터베이스를 통해 관리된다. 사용자 인증 및 권한 관리는 Auth utility의 verifyUser 메서드를 활용하여 수행되며, 유효하지 않은 요청이나 만료된 토큰에 대해서는 적절한 에러 메시지로 반환된다. 사용자 정보는 아이템 서비스와 채팅 서비스의 검증 과정에서도 활용되며, 각 사용자간의 물품 매칭 과 채팅 참여 여부를 판단하는 기준이 된다.

3-2-4. Auth Utility

Auth Utility 에서는 사용자 인증 및 관리를 담당하는 핵심 모듈이다. 이 모듈은 verifyUser와 verifyAdmin 메서드를 통해 사용자와 관리자의 접근 권한을 검증하며 JWT 기반의 토큰 인증을 수행한다. VerifyUser는 사용자의 요청을 확인하고, 토큰이 유효한지 검증하며, 만료된 토큰이나 부적절한 권한 요청에 대해서는 401또는 403 에러를 반환한다. VerifyAdmin은 관리자 권한을 추가로 검증하는 메서드로, 특정 서비스나 리소스에 대한 접근을 제한한다. 토큰 생성시, generateToken 메서드를 사용하여, 각 사용자의 역할, 만료시간, 대상와 같은 정보를 포함한 JWT 토큰을 생성한다. 이 모듈은 사용자 관리 서비스, 아이템 관리 서비스, 채팅 서비스에서 공통으로 활용될 수 있으며 각 요청의 신뢰성과 보안을 보장하도록 한다.

3-2-5. DB Utility

DB utility는 데이터 베이스 연결 및 상태 관리를 책임지는 모듈이다. MongoDB

와의 연결을 효율적으로 처리하기 위해 connectDB 메서드를 제공하며, 필요에 따라, 특정 데이터 베이스에 연결을 설정한다. 이 모듈은 disconnecDB 메서드를 통해 연결을 해제할 수 있으며, Test 환경에서는 MongoMemoryServer를 사용하여 임시 데이터베이스를 관리한다. IsMongoDBConnected는 데이터 베이스 연결 상태를 실시간으로 확인하는 메서드로, 각 서비스에서 DB연결 여부를 점검하는데 사용된다. 데이터베이스 유형과 URI는 환경 변수에서 관리되며, 이를 통해 배포환경과 테스트 환경의 유연한 전환이 가능하다. DBType과 DBUri는 데이터베이스 구분과 접근성을 높이기 위해 Enum 형태로 정의되어 있으며 모든 서비스에서 일관된 데이터 베이스 접근 방식을 제공한다.

3-2-6. Crypto Utility

Crypto Utility 에서는 사용자 데이터 보호와 암호화를 위한 보안 모듈이다. 이 모듈은 bcrypt를 활요하여 비밀번호를 해시화하는 hashPassword 메서드를 제공한다. 이를 통해, 사용자 비밀번호의 안전성을 보장하며, 인증 과정에서의 무결성을 유지할 수 있게 한다. 또한, AES-256-CBC 알고리즘을 활요하여 민감한 데이터의 암호화(encryptData) 및 복호화(decryptData)를 처리한다. 해당 서비스는 사용자 관리 서비스와 긴밀히 통합되어 있으며, 데이터 유출과 보안 위협을 최소화하는 역할을 한다.

3-2-7. Logger Utility

Logger Utility는 서버의 이벤트와 오류를 기록하여 개발 및 운영의 모니터링과 디버깅을 지원하는 모듈이다. 해당 모듈의 로그는 날짜별로 관리되며, 로그 파일은 서버의 logs 디렉토리에 저장된다. Info, warn, error, debug 메서드를 통해서 다양한 로그를 제공하며, 각 로그는 타임스템프, 요청 IP, 로그메시지를 포함한다. HttpLogger는 morgan 라이브러리를 활요하여 HTTP 요청 로그를 기록하며, 서

버 접근 이력을 체계적으로 관리할 수 있도록 한다. 이 로그는 서비스 에러 발생 시 원인을 파악하고 대응할 수 있도록 한다. 모든 서비스는 로그를 생성하고 기록하는 구조로 설계되어 있으며 이를 통해 운영상의 오류를 효율적으로 모니터링 할 수 있다.

3-2-8. Response Model

Response Model은 API 호출의 일관된 응답 구조를 제공하기 위해 정의된 모듈이다. API 호출이 실패하는 상황에서 발생된 원인을 명확하게 전달하기 위해 FailureReson Enum을 제공하며, USER_NOT_FOUND, INVALID_PARAMETER, SERVER ERROR와 같은 다양한 실패 사유를 정의하도록 한다. 해당 모듈은 각 서비스에서 발생하는 에러에 대해 사용자에게 명확한 피드백을 제공하도록 한다.

벡엔드 컨트롤러는 사용자와 Admin의 역할을 명확히 분리하며, 필요에 따라 Admin이 사용자 데이터를 검토하고 관리할 수 있도록 유연하게 설계되어 있다. 아래는 각 서버의 Controller에 대한 서술이다.

3-2-9. Policy Controller

Policy Controller는 이용 약관 및 개인정보 처리 방침의 조회, 추가, 수정 삭제 기능을 담당한다. 이때, Admin 권한을 가져야, 추가, 수정 및 삭제를 할 수 있다. GET /policy/terms 엔드 포인트를 통해 약관 종류와 국가 코드에 해당하는 약관 내용을 반환할 수 있으며, 필요한 경우 약관 버전에 따라 특정 버전을 조회할 수 있다. POST /policy/terms에서는 새로운 약관을 등록할 수 있으며, 약관 버전과 종류, 국가 코드, 내용을 필수로 받아서 처리한다. 이미 존재하는 약관이 입력될 경우, 중복 방지를 위해, 오류를 반환하며, 이 과정에서 verifyAdmin을 통해 관리자로서 권한을 검증한다. PUT /policy/terms는 기존 약관을 특정 버전과 국가 코드에 맞춰 수정할 수 있다. DELETE /policy/terms는 특정 약관을 삭제할 수 있으며 관리자의 인증 절차를 거친 후 처리된다. 위와 같은 약관 데이터를 다루

는 Policy Controller를 통해 약관 데이터의 무결성과 보안을 보장한다.

3-2-10. Service Controller

Service Controller는 서버 상태 관리와 관리자 인증 기능을 제공하는 모듈이다. 데이터 베이스 연결상태를 확인하는 기능은 GET /service/status 엔드포인트를 통해 실행되며, 데이터베이스가 정상적으로 연결되어 있는지 확인하는 상태 메시지를 반환한다. 해당 기능을 통해 서버 상태를 실시간으로 모니터링 할 수 있으며 서버의 장애를 사전에 감지하는 역할을 수행한다. POST /service/admin-token 엔트포인트는 요청된 관리자 아이디와 비밀번호가 서버에 저장된 정보와 일치할 때만 JWT 토큰을 생성하고 반환한다. 이를 통해 관리자 권한이 필요한 기능에서 안전한 접근 권한을 부여할 수 있도록 한다.

3-2-11. User Controller

User Controller는 사용자 인증과 회원 정보 관리에 필요한 다양한 기능을 제공하며, 회원가입, 로그인, 비밀번호 관리 등 사용자 중심의 서비스를 지원한다. 전화번호 중복 확인 API는 사용자의 입력값을 기준으로 전화번호의 사용 가능 여부를 확인하고, 이미 등록된 전화번호인 경우 에러값으로 응답한다. POST /user/singup 엔드 포인트를 통해 사용자 회원가입이 진행되며, 필수 입력 정보와 약관 버전, 비밀번호 암호화를 처리한 후 데이터베이스에서 사용자 정보를 안전하게 저장한다. POST /user/login 은 로그인 기능을 제공하며, 사용자 아이디, 전화번호, 비밀번호를 검증한 후 JWT 토큰과 함께 사용자 정보를 반환한다. DELETE /user/withdraw 엔드포인트는 사용자 계정 탈퇴 기능을 수행할 수 있다. 위와 같은 사용자 관련 Controller를 통해 사용자 정보를 효율적으로 관리할 수 있다.

3-2-12. Chat Controller

Chat Controller는 채팅 관련 기능을 제공하며, 사용자간의 소통과 거래를 지원한다. 채팅방 생성 기능은 POST /api/chat/create-chat-room을 통해 제공되며, 사용자가 요청한 아이템과 상대방의 아이템 정보를 기반으로 새로운 채팅방을 생성하고 채팅방 UUID를 반환할 수 있게 해준다. POST /api/chat/send-message를 통해서 사용자 인증후 메시지를 채팅방에 추가하며, 채팅 내역 관리를 서비스 레이어에서 처리할 수 있도록 한다. GET /api/chat/get-my-chatroom-list는 사용자가 참여한 채팅방 리스트를 반환할 수 있도록 하며, 특정 채팅방의 조회하는 기능은 GET /api/chat/get-unread-message 엔드포인트를 통해 제공된다. DELETE /api/chat/delete-chat-room을 통해 사용자가 채팅방을 삭제할 수 있다. 위 기능들을 통해 사용자 간의 원활한 소통을 지원한다.

본 프로젝트의 물물교환 앱의 프론트엔드와 백엔드 모듈별 역할과 기능을 위에서 명확히 설명하였으며, 앱의 전반적인 구조와 흐름을 체계적으로 설명하였다. 위 설명에서, 각 모듈은 독립적으로 설계되었으며, 필요 시 유기적으로 작동하도록 구현되었다. 프론트엔드는 사용자 경험을 강화하는 다양한 페이지와 상태 관리 기능을 제공하고, 백엔드는 확장성과, 보안성을 고려한 공통 모듈을 기반으로 안정적인 데이터를 처리한다. 이를 통해 사용자와 관리자의 니즈를 모두 충족하는 설계 원칙을 중심으로 전체적인 프로젝트의 세부 구성을 완성하였다.

4. Software Quilty

4-1. Front-End

4-1-1. Stability

본 프로젝트의 프론트엔드는 Flutter 프레임워크를 기반으로 설계되었으며, 철저한 모듈화와 예외 처리를 통해 높은 안정성을 확보하였다. Flutter의 위젯 트리 구조를 활용하여 로그인, 회원가입, 비밀번호 관리 등 모든 기능을 독립적인 위젯으로 분리하였다. 이를 통해 각 기능이 독립적으로 관리되며, 특정 기능의 수정이 다른 부분에 영향을 미치지 않도록 설계하였다.

네트워크 요청 시 안정성을 보장하기 위해 Dio 라이브러리를 사용하였으며, 서버 응답 실패, 시간 초과, 네트워크 연결 오류와 같은 예외 상황에 대비해 철저한 에러 핸들링을 구현하였다. 오류 발생 시 Toast 메시지나 Snackbar를 통해 사용자에게 명확한 피드백을 제공하고, 네트워크 연결이 재개되면 데이터가 정상적으로 동기화될 수 있도록 처리하였다.

상태 관리에서는 Provider와 같은 상태 관리 도구를 사용하여 UI와 데이터 흐름이 안정적으로 유지되도록 하였다. 비동기 작업은 FutureBuilder와 StreamBuilder를 활용하여 데이터를 실시간으로 처리하면서도 화면이 멈추거나 비정상적으로 동작하는 경우를 방지하였다. 이러한 비동기 처리 방식을 통해 네트워크 지연 시에도 화면 전환이나 데이터 로딩이 원활하게 이루어진다.

회원가입과 같은 입력 폼에서는 Validator 기능을 사용하여 데이터 유효성을 철저히 검증하였다. 예를 들어, 아이디 중복 검사, 비밀번호 일치 여부 확인, 전화번호 형식 검증 등을 통해 잘못된 데이터가 서버로 전송되는 것을 방지하였다.

또한 다국어 기능과 테마 설정 기능을 구현하여 앱이 다양한 사용자 환경에서도 안정적으로 작동하도록 하였다. 다국어 기능은 모든 텍스트를 JSON 파일로 관리하여 언어를 실시간으로 변경할 수 있도록 하였으며, 테마 설정 기능은 다크 모드와 라이트 모드 전환 시에도 UI가 일관되게 유지되도록 구현하였다.

이와 같이 본 프로젝트의 프론트엔드는 위젯 분리, 네트워크 오류 처리, 데이터 검증, 상태 관리를 기반으로 설계되어 높은 안정성을 보장하며, 사용자에게 원활하고 신뢰할 수 있는 서비스를 제공한다.

4-1-2. Maintenance

본 프로젝트는 코드의 구조화와 표준화를 통해 높은 유지보수성을 확보하였다. 주요 기능들은 MVVM 패턴(Model-View-ViewModel)을 적용하여 UI와 비즈니스 로직, 데이터 관리를 분리하였다. 이를 통해 특정 기능의 수정이나 확장이 필요한 경우, 전체 시스템에 영향을 최소화하면서 변경사항을 적용할 수 있도록 설계되었다.

코드의 재사용성과 가독성을 높이기 위해 로그인, 회원가입, 비밀번호 관리 등 주요 기능을 모듈화된 위젯으로 설계하였다. 각 기능은 독립적으로 작성되어 수정이 용이하며, 다른 화면이나 기능에서도 동일한 컴포넌트를 재활용할 수 있다. 또한 코드의 일관성을 유지하기 위해 Lint 도구를 사용하여 코드 스타일을 통일하였고, 상세한 주석과 문서화를 통해 개발자 간 협업과 유지보수가 용이하도록 하였다.

상태 관리는 Provider와 같은 상태 관리 도구를 도입해 UI와 데이터 상태를 효율적으로 관리하였다. 이를 통해 데이터 변경이 발생했을 때 최소한의 리렌더링만 수행되도록 최적화하였으며, 유지보수 시 불필요한 버그 발생을 최소화하였다.

확장성을 고려한 아키텍처도 유지보수성을 높이는 핵심 요소 중 하나이다. 주요 로직은 Service 클래스로 분리하여 로그인, 회원가입, 프로필 수정과 같은 기능의 비즈니스 로직을 한 곳에서 관리할 수 있도록 하였다. 이로써 새로운 기능 추가나 기존 기능 수정 시 비즈니스 로직만 수정하면 되므로 개발 효율성이 극대화된다.

테스트와 배포 과정에서도 유지보수성을 강화하였다. Flutter의 Hot Reload 기능을 활용해 개발 중 빠르게 코드 수정과 테스트를 반복할 수 있었으며, Unit Test와 Widget Test를 작성하여 주요 로직과 UI의 안정성을 검증하였다. 또한 Git을

통한 버전 관리와 CI/CD 파이프라인을 도입하여 코드 변경 사항이 자동으로 테스트 및 배포되도록 하였다. 이를 통해 유지보수 시 발생할 수 있는 인적 오류를 줄이고, 배포의 안정성을 높였다.

플랫폼 호환성을 유지하기 위해 Flutter의 멀티 플랫폼 지원 기능을 활용하였으며, iOS와 Android에서 동일한 코드베이스로 앱이 실행되도록 하였다. 화면 해상도와 비율에 따라 UI가 일관되게 표시될 수 있도록 Responsive Design을 적용하였으며, 브라우저와 다양한 디바이스에서의 테스트를 통해 플랫폼 간 호환성을 검증하였다.

이와 같이 본 프로젝트의 프론트엔드는 모듈화된 구조, 상태 관리 도구, 코드 표준화, 자동화된 테스트 및 배포 환경을 기반으로 유지보수성을 극대화하였다. 새로운 기능 추가나 확장 시에도 최소한의 작업으로 효율적인 유지보수가 가능하며, 장기적으로도 지속 가능한 시스템으로 운영될 수 있도록 설계되었다.

4-2. Back-End

4-2-1. Stability

본 프로젝트의 백엔드는 Docker Compose, Node.js Express, 그리고 MongoDB를 기반으로 설계되어 높은 안정성을 제공한다. Docker Compose를 통해 서버와 데이터베이스를 각각 독립된 컨테이너로 실행함으로써 개발, 테스트, 운영 환경 간의 일관성을 유지하였으며, 특정 컨테이너에 오류가 발생하더라도 자동 재시작 기능을 통해 신속하게 복구될 수 있도록 구성하였다.

Node.js Express의 비동기 이벤트 기반 아키텍처를 활용하여 서버는 다수의 요청을 동시에 처리할 수 있으며, 성능 병목을 최소화하였다. 서버 간 통신은 RESTful API를 기반으로 효율적으로 이루어지며, 각 서버가 개별적으로 독립된 로직을 처리하도록 분리하여 특정 기능에 부하가 집중될 경우 수평 확장이 가능하도록 설계되었다.

데이터베이스로는 MongoDB를 사용하여 데이터의 일관성과 무결성을 보장하였다. 각 서버에서 사용하는 데이터는 명확하게 정의된 Schema를 기반으로 관리되

며, 트랜잭션 기능을 통해 데이터 손실이나 오류를 방지하였다. 또한 자주 사용되는 데이터에 대해서는 인덱싱과 쿼리 최적화를 적용하여 빠른 데이터 조회가 가능하도록 구현하였다.

예외 상황에 대비하여 글로벌 에러 핸들링 로직을 도입하였으며, 요청 실패, 서버 오류, 데이터베이스 연결 문제 등 모든 예외를 체계적으로 처리하였다. Winston과 같은 로깅 도구를 통해 서버의 상태와 오류 로그를 중앙화하여 관리하고, 문제가 발생했을 때 신속하게 원인을 파악하고 대응할 수 있도록 하였다.

또한 Docker Compose와 마이크로서비스 아키텍처를 통해 서버의 각 기능이 독립적으로 배포되기 때문에 특정 서버에 문제가 발생하더라도 다른 서버에 영향을 미치지 않도록 설계되었다. 필요에 따라 특정 서버만을 확장하거나 수정할 수 있어 시스템의 안정성을 한층 더 강화하였다.

이처럼 본 프로젝트의 백엔드는 환경 격리, 비동기 처리를 통한 성능 최적화, 데이터 일관성 유지, 에러 핸들링과 로깅 시스템 도입을 통해 서비스의 안정성을 보장하며, 높은 트래픽에서도 신뢰할 수 있는 성능을 제공한다.

4-2-2. Maintenance

본 프로젝트는 유지보수성과 확장성을 극대화할 수 있도록 구현되었다. Docker Compose를 통해 서버와 데이터베이스를 각각 컨테이너화하였으며, 모든 설정은 docker-compose.yml 파일을 통해 통합적으로 관리된다. 이를 통해 새로운 개발자가 프로젝트에 참여하거나 시스템을 재구성할 때 환경 설정을 일관되게 유지할 수 있으며, 배포 및 테스트 과정도 간편하게 진행된다.

서버 로직은 모듈화된 마이크로서비스 아키텍처를 기반으로 구성되었다. 주요 기능인 사용자 관리(server_user), 물품 관리(server_item), 채팅 기능(server_chat)을 각각의 독립적인 서버로 분리하여 관리함으로써 특정 기능 수정이나 확장이 필요한 경우 해당 서버만을 수정하면 되도록 설계되었다. 공통적으로 사용되는 인증, 에러 처리, 유틸리티 로직은 shared 모듈에 따로 분리하여 코드 중복을 최소화하고 유지보수 효율성을 높였다.

API는 Swagger를 활용하여 문서화되어 있으며, 이를 통해 각 서버의 RESTful API 구조와 데이터 흐름을 명확히 정의하였다. 이를 통해 개발자나 유지보수 담당자는 서버 간 통신 방식을 쉽게 파악하고, 수정 사항을 신속하게 적용할 수 있다. 또한 코드 수정 후 Jest와 같은 테스트 프레임워크를 사용해 Unit Test 및 Integration Test를 수행함으로써 코드 안정성을 검증하고, 수정 시 발생할 수 있는 오류를 사전에 방지하였다.

버전 관리는 Git을 통해 체계적으로 이루어지며, 기능별 브랜치를 운영하고 코드 리뷰를 통해 품질을 보장한다. CI/CD 파이프라인을 도입하여 코드 변경사항이 발생하면 자동으로 빌드, 테스트, 배포 과정이 실행되도록 구성하였다. 이를 통해 지속적으로 시스템을 업데이트하면서도 안정성을 유지할 수 있으며, 인적 오류를 최소화하였다.

MongoDB를 기반으로 한 데이터베이스는 Schema를 명확하게 정의하였으며, 구조 변경이 필요한 경우에도 마이그레이션 도구를 활용해 단계적으로 적용할 수 있도록 관리하였다. 데이터 접근은 서비스별로 최적화된 쿼리와 인덱싱을 통해 이루어지며, 데이터 구조 변경 시에도 최소한의 작업으로 유지보수가 가능하다.

이처럼 본 프로젝트의 백엔드는 모듈화된 아키텍처, 코드 중복 최소화, 자동화된 테스트 및 배포 시스템을 기반으로 설계되어 유지보수가 용이하다. 새로운 기능 추가나 확장 시에도 기존 시스템에 미치는 영향을 최소화하면서 효율적으로 개발과 유지보수를 수행할 수 있는 구조를 갖추고 있다.

4-3. Security

본 프로젝트는 데이터 보호와 시스템의 무결성을 보장하기 위해 다층적 보안 구조를 기반으로 설계하였다. 사용자 인증, 비밀번호 관리, 데이터 암호화, 로그 기록 등 다양한 보안 요소를 구현하였으며, 이를 통해 안전하고 신뢰할 수 있는 시스템 환경을 제공한다.

4-3-1. 사용자 인증 및 권한 관리

본 프로젝트의 사용자 인증과 권한 관리는 JWT(Json Web Token) 기반으로 구현되었으며, 역할(Role)에 따라 접근 제어를 철저히 관리하도록 설계되었다.

첫째, JWT 토큰 생성 과정에서는 사용자의 역할에 따라 USER와 ADMIN으로 구분된 토큰을 생성한다. 토큰의 신뢰성과 무결성을 보장하기 위해 JWT_SECRET 키를 사용하여 서명하며, 만료 시간(exp)을 24시간으로 설정하였다. 토큰에는 Registered Claims와 Private Claims를 포함한다. Registered Claims에는 iss(발급자: "woju-backend"), aud(대상자: "woju-frontend"), exp(만료 시간)과 같은 표준 정보가 포함되며, Private Claims에는 role(사용자 역할)과 함께 userUUID(USER용) 또는 adminID(ADMIN용)와 같은 역할에 따라 필요한 정보를 담아 관리한다.

둘째, 토큰 검증 및 역할 기반 접근 제어를 위해 verifyAdmin과 verifyUser라는 미들웨어를 구현하였다. 이를 통해 ADMIN과 USER의 권한을 구분하고 각각의 접근을 철저히 제어한다. 토큰의 발급자(iss)와 대상자(aud)를 검증함으로써 토큰의 출처를 확인하고, 만료 시간(exp)이 지나지 않았는지 검사하여 만료된 토큰은 무효화한다. 특히 ADMIN의 경우 관리자 전용 기능에만 접근할 수 있도록 제한하였으며, 일반 사용자(USER)는 본인의 소유 데이터에만 접근할 수 있도록 설계하였다. 이를 통해 불필요한 데이터 접근과 권한 오남용을 방지하였다.

셋째, 안전한 토큰 전달을 위해 클라이언트는 Authorization Header의 Bearer 스키마를 사용하여 토큰을 서버에 전달하도록 설계되었다. 서버는 요청 시 헤더를 검사하여 토큰의 유효성을 검증하며, 토큰이 존재하지 않거나 유효하지 않을

경우 401(Unauthorized) 또는 402(Invalid Token) 오류 메시지를 반환한다. 이를 통해 불법 접근 시도를 효과적으로 방지하며 시스템의 무결성을 유지한다.

이와 같이 JWT를 기반으로 한 사용자 인증과 권한 관리는 철저한 토큰 생성, 검증, 접근 제어로 이루어져 있으며, 시스템의 보안을 강화하고 사용자 역할에 따른 안전한 데이터 접근을 보장한다.

4-3-2. 비밀번호 관리

본 프로젝트는 사용자의 비밀번호를 안전하게 보호하기 위해 Bcrypt 해시 알고리즘을 적용하였다.

첫째, 비밀번호 해시화 과정에서는 사용자가 비밀번호를 설정하거나 변경할 때 Bcrypt를 사용하여 비밀번호를 해시화한다. 10 saltRounds를 적용함으로써 해시의 복잡도를 높여 무차별 대입 공격(Brute Force Attack)이나 레인보우 테이블 공격에 대한 보안을 강화하였다. 이 과정에서 원본 비밀번호는 절대 데이터베이스에 저장되지 않으며, 오직 해시화된 결과만 저장된다. 이를 통해 데이터 유출 시에도 원본 비밀번호가 노출되는 위험을 방지할 수 있다.

둘째, 비밀번호 검증 과정에서는 사용자가 로그인 시 입력한 비밀번호와 데이터베이스에 저장된 해시값을 비교하기 위해 Bcrypt의 compare 함수를 사용한다. 이 함수는 입력된 비밀번호를 동일한 해시 알고리즘으로 변환한 후 기존 해시값과 비교하여 일치 여부를 확인한다. 비밀번호가 일치하면 로그인 요청이 성공적으로 처리되며, 일치하지 않는 경우 사용자에게 오류 메시지를 반환하여 잘못된 접근을 방지한다.

이와 같이 Bcrypt를 기반으로 한 비밀번호 해시화 및 검증 시스템은 비밀번호 관리의 보안을 강화하고 데이터베이스에서 발생할 수 있는 잠재적 보안 위협을 효과적으로 차단한다.

4-3-3. 데이터 암호화

본 프로젝트는 전화번호와 같은 민감한 데이터를 안전하게 관리하기 위해 AES-256-CBC 알고리즘을 사용하여 암호화 및 복호화 시스템을 구현하였다.

첫째, 데이터 암호화는 AES-256-CBC 알고리즘을 기반으로 수행된다. 이 과정에서 ENCRYPTION_KEY(32바이트 길이의 키)를 사용하며, 데이터 암호화 시 16바이트의 IV(Initialization Vector)를 랜덤하게 생성하여 보안을 강화하였다. 암호화된 데이터는 IV:암호문의 형태로 저장되며, IV와 암호문이 분리되어 관리되므로 복호화 과정에서도 해당 IV 값이 반드시 필요하다. 이 방식을 통해 데이터가 무작위로 암호화되므로, 동일한 데이터라도 매번 다른 결과값을 생성하여 공격자의 분석을 어렵게 만든다.

둘째, 데이터 복호화는 서버 내부에서만 안전하게 수행된다. 복호화 시에는 암호화 단계에서 사용한 ENCRYPTION_KEY와 해당 데이터의 IV 값을 활용하여 데이터를 복구한다. decryptData 함수를 통해 암호화된 데이터와 IV가 결합된 값을 해독하며, 이를 통해 원본 데이터가 안전하게 복원된다. 특히 암호화 키 관리에 대한 보안을 강화함으로써 외부에서 키가 유출되는 상황을 방지하였으며, 키 없이 데이터 복호화가 불가능하도록 설계되었다.

이와 같이 AES-256-CBC 암호화 및 복호화 시스템은 민감한 데이터를 안전하게 보호하며, 데이터 유출이나 무단 접근 시에도 원본 데이터의 노출을 철저히 방지한다. 이를 통해 사용자 데이터의 기밀성과 무결성을 보장하였다.

4-3-4. 보안 로그 및 모니터링

서버 보안을 강화하고 문제 발생 시 신속하게 대응하기 위해 로깅 시스템을 설계하고 구축하였다.

첫째, HTTP 요청 로깅은 Morgan 미들웨어를 통해 구현되었다. 서버에 들어오는 모든 HTTP 요청에 대해 요청 메서드, URL, 상태 코드, 그리고 요청한 IP 주소를 기록한다. 이를 통해 클라이언트 요청의 흐름을 파악하고 비정상적인 접근 패

턴을 탐지할 수 있도록 하였다. 생성된 로그는 날짜별로 분리되어 YYYY-MM-DD.log 형식으로 저장되며, 서버 최상위의 logs 폴더에 보관된다. 이로써 로그 파일의 관리와 검색이 용이해지고, 장기적인 모니터링과 분석이 가능하도록 구성하였다.

둘째, 보안 로그 기록은 시스템의 이상 행동이나 에러 발생 시 기록되도록 설계되었다. 로그 기록에는 타임스탬프, 로그 수준(예: INFO, WARN, ERROR, DEBUG), 호출한 IP 주소, 그리고 오류 메시지 또는 이벤트 내용이 포함된다. 로그 수준은 INFO, WARN, ERROR, DEBUG로 구분되며, 일반적인 요청 흐름은 INFO로 기록하고, 비정상적인 접근 시도나 시스템 오류는 ERROR 또는 WARN 레벨로 기록한다. 이를 통해 로그 파일을 분석하면 문제의 원인을 신속하게 파악할 수 있으며, 잠재적인 보안 위협에 대응할 수 있다.

이와 같이 구축된 로깅 시스템은 HTTP 요청과 시스템 오류를 체계적으로 기록하고, 데이터를 날짜별로 관리하여 서버의 안정적인 운영과 보안을 강화한다. 로그 분석을 통해 비정상적인 접근 패턴이나 시스템 성능 저하를 사전에 탐지하고 대응할 수 있어, 시스템의 신뢰성을 한층 더 높였다.

4-3-5. 시스템 무결성 및 안전성

본 프로젝트는 시스템의 무결성과 데이터 보호를 위해 환경 변수 관리, 불법 접근 방지, 만료 시간 관리, 그리고 에러 핸들링을 철저히 구현하였다.

첫째, 민감한 정보는 환경 변수 파일을 통해 관리된다. JWT_SECRET와 ENCRYPTION_KEY와 같은 중요한 정보는 .env 파일에 저장되며, 서버 코드와 분리되어 외부에 노출되지 않도록 설정되었다. 이를 통해 소스 코드 저장소나 배포 과정에서 보안 키가 유출되는 위험을 방지하고, 시스템의 보안 수준을 한층 강화하였다.

둘째, 권한 없는 접근을 방지하기 위해 JWT 토큰 검증과 역할 기반 접근 제어를 구현하였다. 모든 API 접근 시 서버는 토큰의 유효성을 검증하며, 역할(Role)을 기반으로 사용자의 권한을 확인한다. USER와 ADMIN의 접근 범위를 명확히 구

분함으로써 민감 데이터에 대한 불법 접근을 차단하고, 권한이 없는 사용자가 비인가된 데이터를 요청하는 것을 방지하였다.

셋째, JWT 토큰의 만료 시간 관리를 통해 토큰의 사용 기간을 제한하였다. 토큰은 발급 시 24시간의 만료 시간을 설정하였으며, 만료된 토큰은 자동으로 무효화된다. 이를 통해 장기적으로 사용 가능한 토큰으로 인한 보안 위협을 방지하고, 사용자의 재인증을 통해 시스템 접근의 신뢰성을 높였다.

넷째, 시스템 내 모든 인증 및 검증 과정에서 발생하는 오류에 대해 적절한 에러 핸들링을 구현하였다. 오류 발생 시, 상태 코드와 함께 사용자에게 명확하지만 최소한의 정보만 반환하도록 설계되었다. 예를 들어, 잘못된 토큰이나 접근 권한이 없는 경우에는 401 Unauthorized 또는 403 Forbidden 상태 코드와 메시지를 반환하지만, 공격자가 시스템의 내부 구조를 파악할 수 있는 세부 정보는 노출하지 않는다. 이를 통해 보안 취약점을 악용하는 시도를 차단하였다.

이처럼 환경 변수 관리, 역할 기반 접근 제어, 만료 시간 설정, 그리고 에러 핸들링을 종합적으로 구현하여 시스템의 보안을 강화하고 민감 데이터를 안전하게 보호하였다. 이를 통해 불법 접근과 보안 위협을 효과적으로 차단하고, 안정적이고 신뢰할 수 있는 서비스 환경을 제공하였다.

4-4. Compatibility

4-4-1. 소프트웨어 사양

표 1. 소프트웨어 사양

	Android	iOS
OS Version	Android 6.0, Marshmallow 이상	iOS 11.0 이상
Linux Kernel Version	3.10.24	Darwin 17
SDK Level	API Level 23	iOS 11 SDK, Xcode 9
지원되는 기기	Galaxy S7 이상	iPhone 5s 이상

표 1 에 프로젝트에서 구현한 어플리케이션을 사용하기 위해 필요한 소프트웨어 최소 사양이 제시 되어있다.

4-4-2. 라이브러리 버전

표 2. 라이브러리 버전

	Library Version
State Management	flutter_riverpod: ^2.5.1
Database	hive: ^2.2.3
Social Login	firebase_auth: ^5.1.4
Theme	cupertino_icons: ^1.0.8
Routing	go_router: ^14.2.3
Localization	easy_localization: ^3.0.7

Accessibility	accessibility_tools: ^2.2.1
Secret	flutter_dotenv: ^5.1.0 flutter_secure_storage: ^9.2.2
Component	flutter_native_splash: ^2.4.1 image_picker: ^1.1.2 flutter_onboarding_slider: ^1.0.11 fluttertoast: ^8.2.6 country_code_picker: ^3.0.0 modal_bottom_sheet: ^3.0.0 scroll_date_picker: ^3.8.0 toggle_switch: ^2.3.0 flutter_neumorphic: ^3.2.0 animated_toggle_switch: ^0.8.3 flutter_markdown: ^0.7.3+ 1 stylish_bottom_bar: ^1.1.0 pro_image_editor: ^5.4.2
Service	device_info_plus: ^10.1.2

표 2 에 프로젝트에서 사용한 어플리케이션의 라이브러리 버전이 제시 되어있다.

5. 계획 및 역할 분담

본 프로젝트는 Water-Fall 방식으로 진행되었으며, 각 단계별 목표와 산출물을 명확히 설정하여 체계적으로 개발을 진행하였다.

5-1. 요구사항 분석

요구사항 분석 단계에서는 고객 및 이해관계자와의 미팅을 통해 프로젝트의 핵심 기능과 비기능적 요구사항을 수집하였다. 특히 물품 교환 애플리케이션의 주요 기능에 대한 정의를 중점적으로 다루었으며, 이를 기반으로 요구사항 명세서를 작성하고 최종 검토를 완료하였다.

- **기간:** 10월 4일 ~ 10월 11일, 1주

- **산출물**: 요구사항 명세서

5-2. 시스템 설계

시스템 설계 단계에서는 프로젝트의 전반적인 아키텍처를 구체화하였다. 프론트엔드, 백엔드, 데이터베이스의 구조를 설계하고, UI/UX 디자인 및 와이어프레임을 작성하였다. 또한 API 설계 및 데이터 모델링을 수행하였으며, 이미지 분류 및 추천 시스템을 위한 알고리즘을 설계하였다.

- **기간:** 10월 12일 ~ 10월 18일, 1주
- **산출물**: 설계 문서, UML 다이어그램, 데이터베이스 스키마

5-3. 구현

구현 단계에서는 시스템의 핵심 기능을 개발하였다.

- **프론트엔드 개발**: 사용자 인터페이스(UI)와 사용자 등록, 물품 등록 화면을 개발하였다.
- **백엔드 개발**: API 개발, 데이터베이스 구축, 사용자 및 물품 관리 기능을 구현하였다.
- **머신러닝 개발**: 이미지 및 텍스트 기반 물품 분류 알고리즘을 구현하고, 물품 추천 시스템을 개발하였다.
- **알림 및 메시지 기능**: 푸시 알림과 실시간 메시지 기능을 추가하였다.

이 단계에서 초기 버전 애플리케이션이 완성되었으며, 프로젝트에 필요한 라이브러리와 버전 종속성을 관리하였다.

- **기간:** 10월 21일 ~ 11월 22일, 5주
- **산출물**: 기본 기능이 포함된 초기 버전 애플리케이션

5-4. 테스트 및 배포

테스트 및 배포 단계에서는 시스템의 안정성과 완성도를 검증하였다.

① 유닛 테스트, 통합 테스트, 시스템 테스트를 통해 각 기능의 정합성을 확인하고 오류를 수정하였다.

② 사용자 인터페이스(UI) 테스트 및 UX 테스트를 진행하여 사용자 경험을 개선하였다.

③ 성능 테스트와 보안 테스트를 통해 시스템의 안정성을 강화하였으며, 테스트 결과를 바탕으로 수정사항을 적용하였다.

④ 애플리케이션의 서버 배포 및 자동화 설정을 진행하였고, 모바일 앱 스토어(앱스토어, 구글 플레이)에 등록 준비를 완료하였다.

⑤ 클라우드 환경 설정 및 모니터링 시스템 구축을 통해 실시간 시스템 상태를 감시할 수 있도록 하였다.

- **기간:** 11월 25일 ~ 12월 8일, 3주
- **산출물**: 애플리케이션 출시 및 배포 완료

5-5. 최종 검토 및 피드백 반영

마지막 단계에서는 배포 후 발생한 최종 오류를 수정하고 시스템을 안정화하였다. 사용자 피드백을 수집하여 빠르게 대응하고 개선사항을 반영함으로써 최종적으로 프로젝트를 완료하였다.

- **기간:** 12월 9일 ~ 12월 10일, 2일
- **산출물**: 최종 사용자 설명서

본 프로젝트는 Water-Fall 모델을 기반으로 요구사항 분석부터 최종 검토까지 단계적으로 진행되었다. 이에 따라 각 단계에서 명확한 목표와 산출물을 도출하

였다. 이를 통해 시스템의 안정성과 완성도를 보장하였고, 사용자 중심의 물품 교환 서비스를 성공적으로 개발할 수 있었다.

5-6. 팀원 별 역할

- ☺ **정민호:** Project Manager (PM), Front-End Main, Back-End Main
- ☺ **김도익:** UI 디자인, Front-End Sub
- ☺ **임석범:** Data Base 설계, Back-End Sub
- ☺ **조민혁:** 보안 설계 담당, Test Code 작성

6. 실행 화면

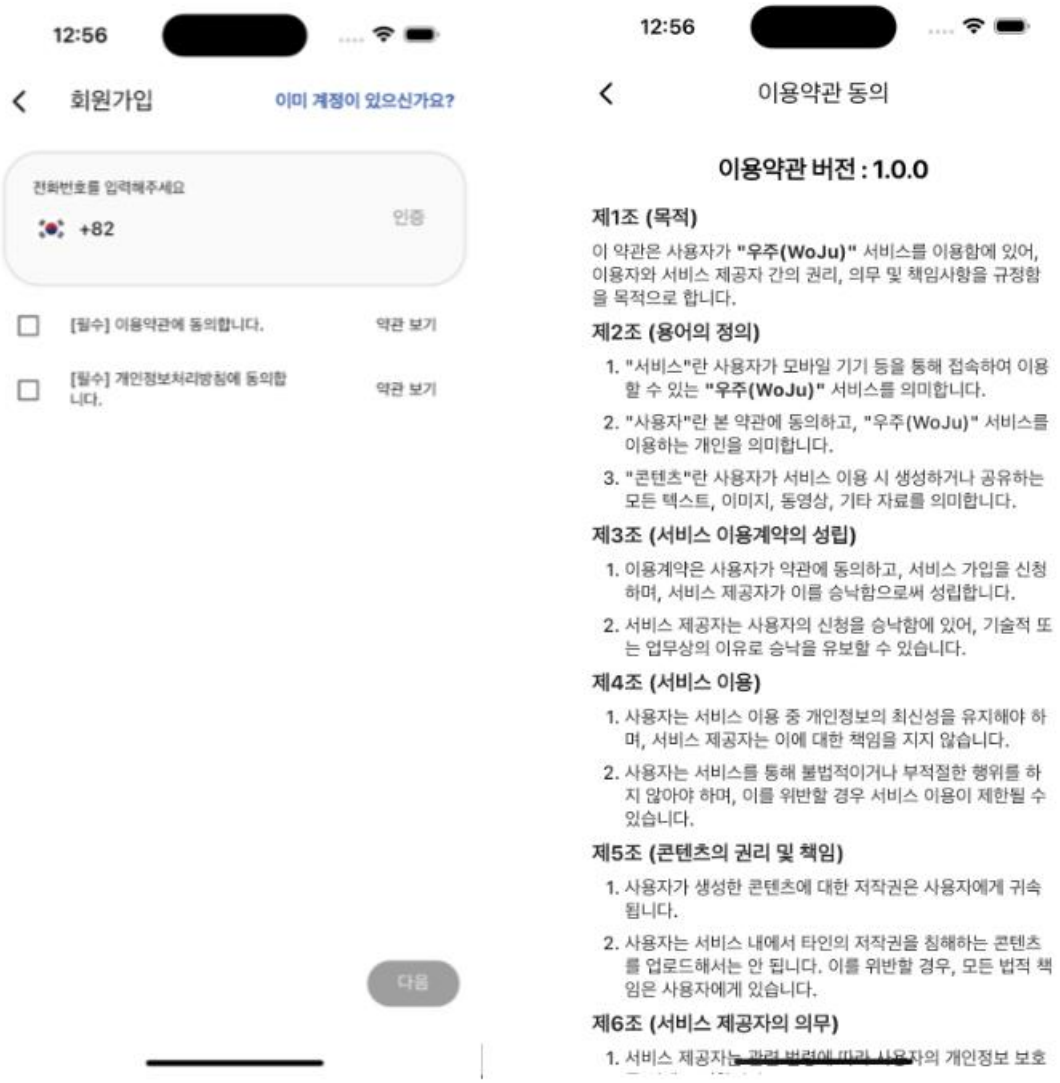

그림 4. 회원 가입 - 이용약관 화면

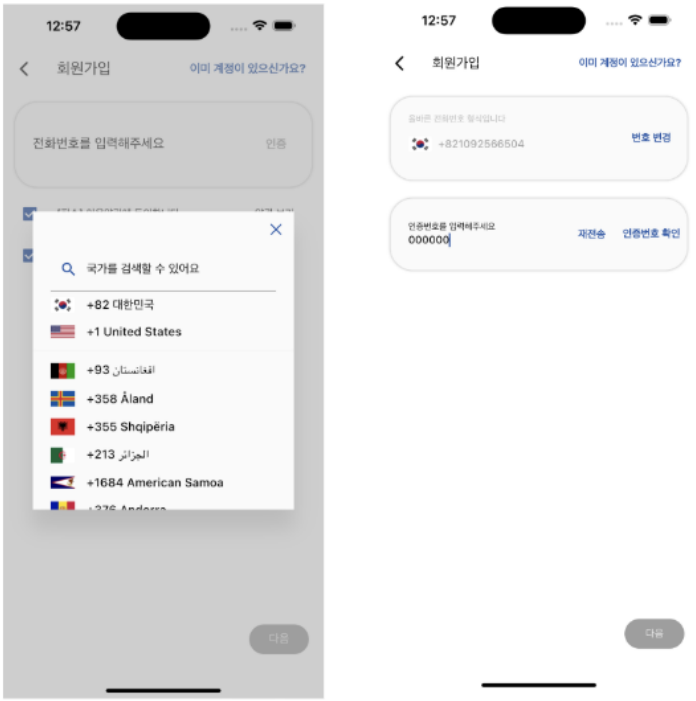

그림 5. 회원 가입 - 나라 선택 및 전화번호 기입 화면

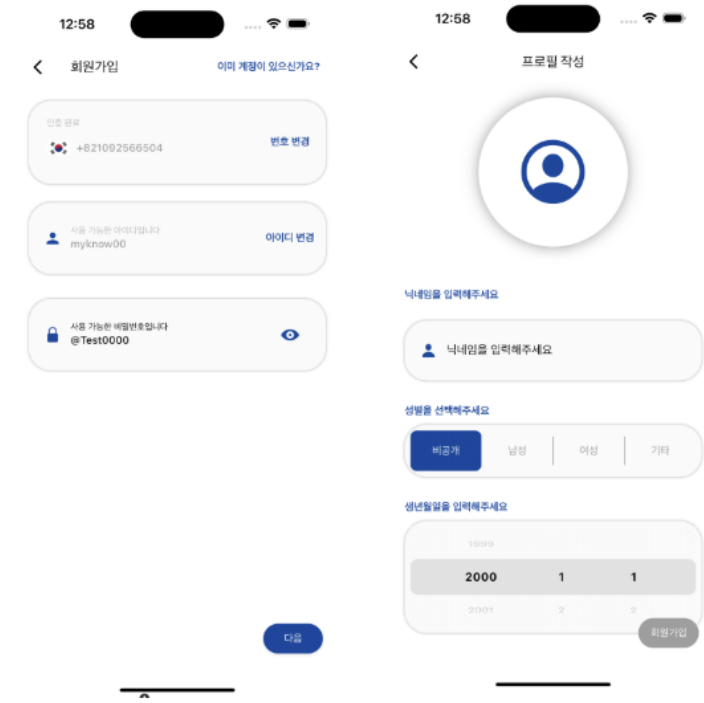

그림 6. 회원가입 - ID/PW 및 프로필 작성

그림 7. 로그인 – ID, 전화번호

그림 8. 물품 추가 및 카테고리 선택 화면

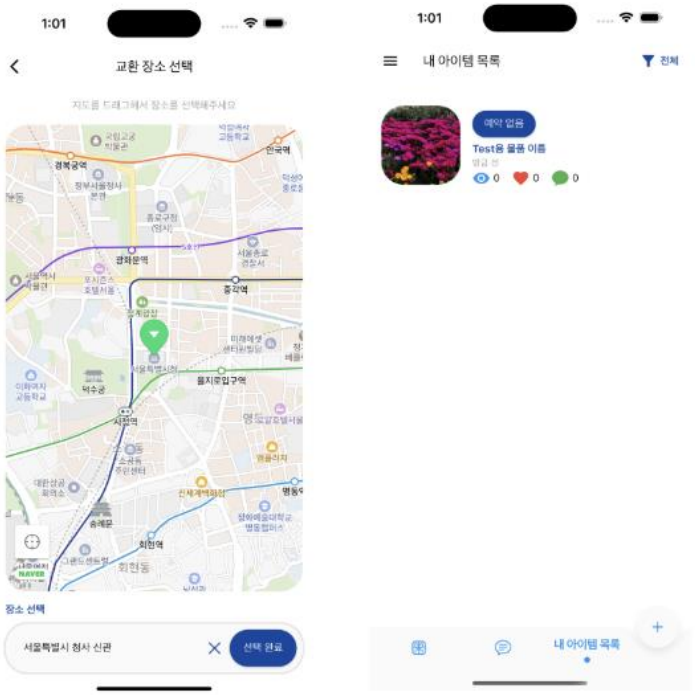

그림 9. 물품 교환: 장소 선택 및 내 아이템 목록 확인 화면

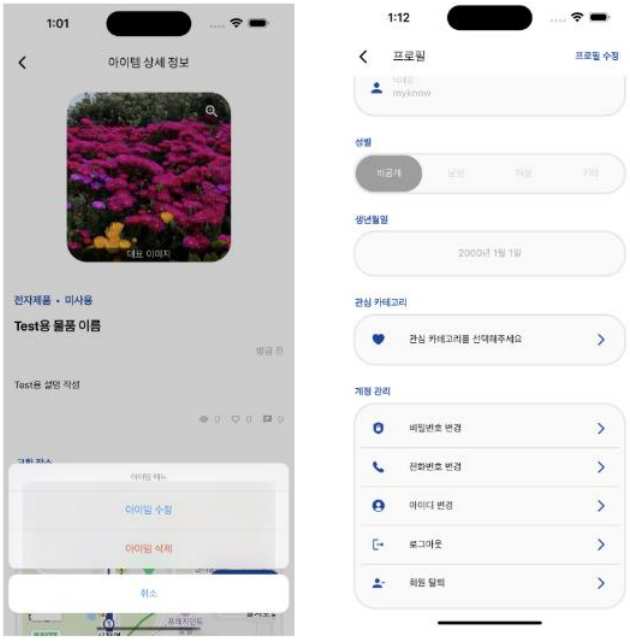

그림 10. 아이템 수정, 삭제 및 프로필 화면

그림 11. 관심 카테고리 및 상대방과 교환할 내 아이템 선택 화면

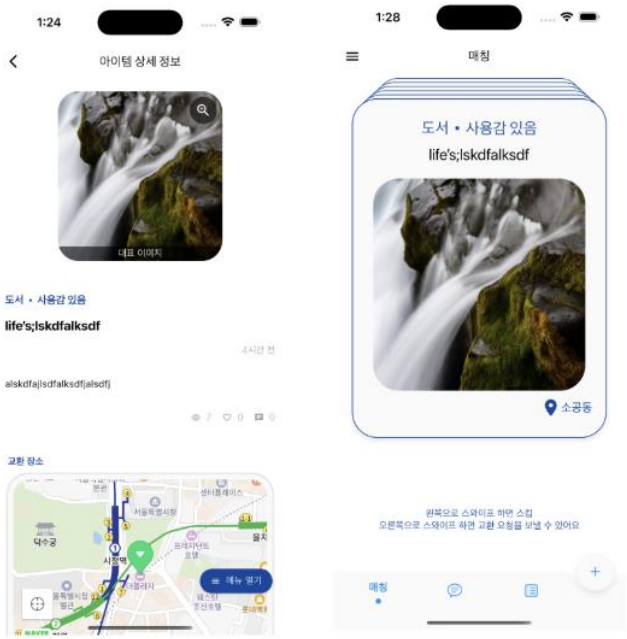

그림 12. 아이템 상세 정보 및 매칭 정보 화면

7. 프로젝트 마무리

이번 프로젝트는 물물교환 서비스 애플리케이션 설계와 개발을 통해 팀 내 협업의 중요성과 효과적인 코딩 프로세스를 경험할 수 있었다. 팀원들은 각자의 역할을 분담하여 지속적인 코드 리뷰, 공통 모듈의 재사용을 통해 협업의 효율성을 극대화하여 하나의 완성된 애플리케이션을 목표로 체계적으로 개발을 진행하였다. 특히, 모노레포 기반의 구조를 통해 유지 보수성을 효과적으로 관리할 수 있도록 설계하였으며, 각각의 모듈은 독립성과 상호 작용성을 균형있게 유지할 수 있도록 하였다.

이번 프로젝트를 통해 팀원들은 Git 을 활용한 버전 관리, 코드 리뷰, CI/CD 파이프라인을 통한 자동화된 테스트와 버전 관리를 경험할 수 있었다. 코드 작성 과정에서 발견된 이슈들은 팀 내 협의를 통해 해결할 수 있었으며, 문제를 해결하는 과정에서 개발 역량을 성장시킬 수 있는 기회가 되었다.

향후 이 애플리케이션은 인공지능(AI) 기술을 활용하여 더욱 발전할 수 있는 가능성을 가진다. 사용자의 편의를 위해 AI 기반 검색기능을 도입하여 물품 교환 과정에서 최적의 물품을 빠르게 찾을 수 있도록 지원한 예정이다. 또한, AI 기반 이미지 분류를 활용하여 물품의 이미지를 자동으로 분류하고 적합한 카테고리를 추천함으로써 사용성을 대폭 향상시킬 계획이다. 마지막으로, LLM 을 활용하여, 챗봇 고객센터를 만들어 사용자의 앱 사용중 불편함을 최소화시킬 예정이다. 위와 같은 향후 계획은 실제 사용할 수 있는 애플리케이션에서 큰 경쟁력을 얻을것이라고 생각한다.

본 프로젝트를 통해 단순한 애플리케이션 개발이 아닌 팀워크와 기술적 성장, 사용자 서비스 설계를 경험하였으며, 이를 바탕으로 다양한 분야에서 활용 가능한 개발역량을 확장할 수 있었다.

읽어주셔서 감사합니다.