성향이 같은 게이머들 간의 매칭 서비스(모바일 앱)

A matching platform for gamers with similar preferences

진경현 (Kyung-Hyun Jin)

한국외국어대학교, 중국어통번역학과; fjkl.446@gmail.com

한글 요약: 서버와 REST API를 통해 데이터를 주고받을 수 있으며, 매칭 서비스를 제공하는 모바일 앱을 구현한다. 서비스의 백엔드는 Python언어와 Django+Nginx를 통해 구현하고, 프론트엔드로는 Dart언어와 Flutter를 통해 구현한다. 앱은 게이머들이 자신의 정보를 프로필카드에 등록하면, 서로의 카드를 확인하고, 서로 마음에 든다면 매칭 되어 게임을 같이 즐기는 서비스를 제공한다.

핵심어: API서버, 모바일앱, Django+uWSGI+Nginx, Dart+Flutter, 매칭시스템

영문 요약: Implements mobile app that can send and receive data through the REST API with servers and provide matching services. The backend of service is implemented through Python language and Django+Nginx, while the frontend is implemented through Dart language and Flutter. The app provides a service where gamers register their information on profile card, check each other's cards, and if they like each other, match each other to enjoy the game together.

Keywords: API Server, Mobile App, Django+uWSGI+Nginx, Dart+Flutter, Matching

1. 서론 - Introduction

컴퓨터의 보급과 그래픽 기술의 발전으로 게임기술과 시장 또한 크게 발전했고, 게이머는 점점 늘어나고 있는 추세이다. 하지만 컴퓨터게임은 영화나 음악 그리고 도서에 비해 역사가 짧다. 최근 20년 사이에 게임은 한층 더 발전하면서, 같은 컴퓨터게임도 다양한 장르가 생기고, 게이머들 또한 메이저 게임만 찾아서 하기 보다는 자신이 좋아하는 장르를 찾고, 자신만의 플레이 방식으로 게임을 즐기고 있다.

당연히 게임 장르와 플레이 스타일에 세대차이의 현상이 생기기도 한다. 스타크래프트의 임요환 선수는 나이가 들면서, 피지컬이 요구되는 게임보다는 두뇌를 사용하는 게임에 더 관심이 생기고, 승부욕보다는 성취감이 게임의 만족도에 더 많은 영향을 주게 되었다고 한다.

소믈리에(Sommelier)는 와인을 추천하기 위해 개인의 취향을 묻는데, 취향은 자신경험과 친구 그리고 주변 커뮤니티의 영향을 많이 받는다고 한다. 게임 또한 마찬가지라고 생각한다. 평론가들이 추천하는 게임보다는 자신과 성향이 비슷한 게이머들과 그들이 즐기는 게임을 같이 플레이하는 것이 더 높은 만족감을 줄 수 있을 것이라 생각하고, 이 가설을 기반으로 성향이 같은 게이머들이 매칭하고 소통할 수 있는 앱 서비스를 개발하게

본 논문은 2021학년 6월에 제출된 한국외국어대학교 컴퓨터공학부 졸업논문이다. 2021.06.04

지도교수: 신창수

서명:

되었다. 사용자들은 서로의 취향과 플레이 방식을 공유하면서 만족도를 높일 수 있고, 매칭된 게이머들의 성향 데이터는 게임개발회사 입장에서 유의미한 정보로 사용될 수 있을 것이라 생각한다.

앱 서비스의 핵심기능은 유저들 간의 매칭 시스템도 있지만, 최대한 같은 성향을 찾을 수 있도록 데이터를 모으고 분석하는 것도 중요하다. 가설로 게이머들이 시청하는 게임 스트리머(Streamer)와 게임 정보를 공유하는 커뮤니티 데이터가 매칭에 도움이 될 것이라 생각한다. 현재는 수동 적인 매칭 기능만 구현되어 있지만, 향후에는 취향 정보를 기반으로 추천시스템을 구현할 계획이다.

2. 연구 방법 및 결과

2.1 주요기능과 구성

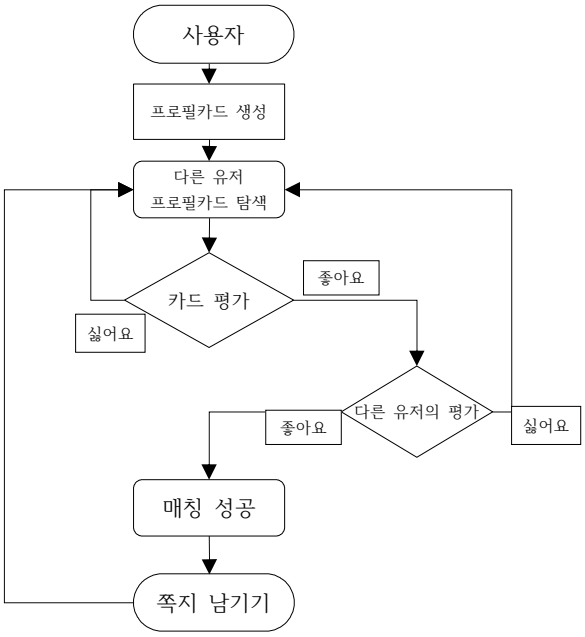


그림 1. 서비스의 플로우 다이어그램

매칭서비스란 두 유저를 이어주는 서비스로 대표적으로 Tinder와 Bumble과 같은 앱이 있고, 데이트를 목적으로 매칭서비스를 제공하고 있으며, 전 세계적으로 큰 인기를 갖고 있다. 게이머 매칭 서비스도 기존의 매칭 앱과 유사한 방식으로 동작한다. 매칭의 목적은 두 게이머를 이어주고, 서로 같이 정보를 공유하고, 게임을 즐기는 것이다. 매칭이 된다면 서로 쪽지를 통해 소통할 수 있다.

매칭이 이루어지는 방식으로 유저들은 각자 프로필카드를 생성하고, 다른 유저들의 프로필카드에 좋아요를 누르고, 그 상대방도 나의 프로필카드에 좋아요를 누를 경우 서로 매칭이 이루어지게 된다. 유저들은 프로필카드를 통해서 상대방에 대한 정보를 얻고, 그 정보를 기반으로 좋아요를 누를지 싫어요를 누를지 결정하게 된다. 각각 개인은 한 가지 게임에 대해 단 한 장의 프로필카드만 생성할 수 있으며, 여러 게임을 할 경우, 여러 장의 프로필카드를 생성할 수 있다.

그림 2. 유저가 생성한 프로필카드

그렇기 때문에 매칭 서비스의 핵심은 프로필카드에 담은 내용이다. 프로필 카드의 구성은 3가지로, 플레이정보, 개인PR정보, 성향정보로 나눠진다.

플레이정보는 구체적인 정보로 자신이 플레이하는 게임의 이름, 플레이 가능한 요일 그리고 시간대의 정보가 포함되고, 자신이 뉴비(처음으로 게임을 접하는 사람) 여부에 대한 정보가 포함되어 있다. 플레이정보가 포함되어야하는 이유는, 가장 기본적이고 필수적인 정보이기 때문이다. 특히 플레이 가능한 시간이나 게임의 이름은 이 프로필 카드를 가장 간략하게 설명할 수 있는 정보이다. 플레이정보를 통해서 얻을 수 있는 효과로, 다른 유저의 평가 시간이 크게 단축될 수 있다. 만약 찾고자하는 게이머가 동시간대에 같은 게임을 즐기고자 한다면, 게임 이름과 플레이 시간 정보는 타협이 가능한 정보가 아니다. 때문에 추가되는 정보를 확인하는 것을 생략할 수 있으며, 평가시간이 줄어들어 앱 서비스의 지루함을 없애줄 수 있다.

개인PR정보는 자유롭고 추상적인 정보로 프로필카드를 등록하는 유저는 텍스트와 이미지를 통해 자신이 전달하고자 하는 정보를 자유롭게 전달할 수 있다. 텍스트는 제목과 내용으로 구성되어 있고, 이미지는 한 장만 업로드 가능하다. 개인PR정보가 있어야하는 이유는 2가지가 있다. 기존에 게이머들은 온라인커뮤니티를 통해 정보교류나, 게이머 친구를 찾았는데, 온라인커뮤니티에서 작성하는 게시물의 구성이 제목+내용의 방식이다. 기존의 방식을 이어 받아 사용한다면, 거부감은 줄고 친숙함은 높일 수 있을 것이라 기대한다. 또한 자유롭게 작성을 한다면, 프로필카드에 자신의 생각과 성향을 더 많이 표출할 수 있다.

개인PR정보로 기대하는 효과는 유저들의 자유로운 표현도 있지만, 다양성을 통해 지루함을 없애고, 앱 내에 머무는 시간을 늘리고 액션을 활발하게 만드는 효과가 있다. 유저들이 단시간 내에 아주 적합한 유저와 매칭이 된다면, 매칭을 통해 얻는 만족도는 상승할 수 있지만, 앱에 머무는 시간은 줄어 들 수 있다. 만족도를 일정 유지하고, 머무는 시간을 높이려면 수많은 프로필 카드를 탐색해야하는데 이 과정에서 지루함이 발생할 수 있다. 이를 해결하기 위해 유저들의 자극적이고 재미있는 개인PR정보가 필요하다. 온라인 기사의 제목이 자극적으로 변한 이유는 클릭률을 높이기 위함이라고 한다.[1] 심지어 어떤 기사는 가짜뉴스의 제목을 사용하여 독자들로 하여금 불쾌감을 주는데, 유저들이 과도하게 낚시성 제목을 사용하지 않고 적절하게 사용해준다면 지루함을 없애고, 자극과 흥미를 높일 수 있다고 생각한다.

마지막 성향정보는 관계적인 정보로 유저들은 자신이 사용하는 온라인커뮤니티의 정보와 즐겨 시청하는 스트리머(Streamer) 정보를 프로필카드에 등록할 수 있다. 성향정보는 기존에 있는 다른 매칭 서비스에 등장한 적이 없는 아주 독특한 정보이다. 사용하는 이유는 스트리머가 갖고 있는 영향력과, 커뮤니티가 갖고 있는 파급력 때

문이다. 여기서 영향력이란 사고방식과 게임스타일 그리고 게임 내에서 추구하는 니즈에 대한 영향을 말한다.[2] 사람은 자신이 호감을 갖고 높게 평가하는 사람의 행동을 따라하는 경향이 있는데, 특히 게임 스트리머의 시청자들이 이러한 경향이 높다. 다른 온라인 개인방송일 경우 주로 먹방이나 재미있는 콘텐츠를 진행하고 경쟁이 발생하지 않는다. 하지만 게임 스트리머 같은 경우 시청자들이 대부분 동일한 게임을 즐겨하고, 게임 내에는 경쟁이 발생하기 때문에, 몰입과 감정 이입이 강하다. 1인 미디어는 인간의 관계형성 욕구를 자극하는데,[3] 게이머 매칭 서비스를 사용하는 유저들은 서로가 즐겨보는 스트리머 정보를 통해 추구하는 관계의 대상이 유사한지 확인할 수 있으며, 추구하는 관계유형이 비슷할 경우 서로의 취향이 더 잘 맞을 것이라 생각한다.

커뮤니티와 그 파급력은 게임과 온라인 문화와 긴밀한 관계가 있다. 문화가 형성되기 위해서는 주체자인 인간과 주체자가 활동할 수 있는 장소는 필요충분조건이다. 온라인 문화도 마찬가지로 온라인 문화가 형성하기 위해서는 활동하는 인간과 인터넷 이라는 장소가 필요하다. 여기서 인터넷을 공간이 아닌 장소라 부르는 이유는 건축학자이자 철학자인 '노르베르크 슐츠'는 공간(space)과 장소(place)의 의미를 따로 두었기 때문이다. 공간은 인간에게 낯설고 물리적인 곳이고, 장소는 익숙하고 친밀한 곳, 인문적인 추억과 활동이 있는 곳으로 그는 정의했다. 따라서 문화가 꽃 피우기 위해서는 사람들에게 낯선 공간 보다는, 보다 친숙하고 친밀한 장소에서 발생하게 된다. 온라인 문화에서 장소의 역할을 하고 있는 곳이 인터넷 커뮤니티이다. 게이머 매칭 서비스를 사용하는 유저들이 자기와 같은 장소에서 게임과 온라인 정보를 공유하고 문화 활동을 하는 다른 유저와 상대적으로 더 친숙하고 관계를 형성하기 더 쉬울 것이라 생각한다.

성향정보는 자신과 조금이라도 더 가까운 관계를 갖은 유저와 매칭 되는 효과도 있지만, 매칭이후의 사용자 경험을 향상하는 효과도 갖는다. 초기 한국 게임 문화에 탄생한, 셔틀(배달해주는 사람), 멘붕(멘탈이 붕괴), 본진(자신의 진영) 등과 같은 용어는 해설과 커뮤니티의 영향을 많이 받았지만, 현재는 스트리머들이 사용하는 용어도 추가되고, 과거 보다 커뮤니티가 더욱 활발해졌다. 새로운 용어와 유행어가 생기고 사라지는데, 자신과 같은 성향정보를 공유한 유저들은 같은 언어를 공유하는 것이라 볼 수 있다.[4] 매칭이후에는 쪽지를 통해 대화를 주고받는데, 이때 사용하는 언어와 어투가 같기 때문에, 막힘없는 소통이 가능하며, 오해를 최소화할 수 있다. 가령 A라는 커뮤니티는 서로 반말을 사용하고, B라는 커뮤니티는 서로 존댓말을 사용한다면, 이 둘의 매칭 이후의 사용자 경험은 같은 어투를 사용하는 사용자 경험보다 좋지 않을 것으로 예상할 수 있다. 유저는 자신과 같은 성향의 유저를 찾고, 같이 게임을 즐기거나, 같은 성향의 정보를 공유할 것이고, 이를 통해 앱 서비스의 만족도가 높아질 것이다.

- **주요 기능 표**

필수적으로 갖고 있는 기능은 아래와 같다.

표 1. 핵심기능

명칭	Layer	설명
회원가입	백엔드(서버), 프론트엔드(앱)	Api를 통해 이메일과 닉네임 비밀번호를 전송하면, 서버에서 중복을 확인하고 auth 토큰을 생성하여 전달
로그인/로그아웃	백엔드(서버), 프론트엔드(앱)	이메일과 비밀번호를 전송하여 로그인하면 기존의 auth 토큰을 전달하거나, 생성한다. 로그아웃은 토큰을 삭제한다.
프로필카드 리스트	백엔드(서버), 프론트엔드(앱)	DB에서 전체 카드에서 자신의 카드를 제외한 다른 유저의 카드를 전달하며 Pagination되어 앱에서 메모리 부담 없이 스크롤
카드 조건조회	백엔드(서버), 프론트엔드(앱)	전체 카드 DB를 filter backend를 통해 조건부

		검색을 한다.
카드 생성	백엔드(서버), 프론트엔드(앱)	앱에서는 휴대폰의 갤러리 접근 권한을 얻고, POST 요청을 통해 카드를 생성한다.
카드 수정	백엔드(서버), 프론트엔드(앱)	PUT 요청을 통해 자신의 카드 정보를 수정한다.
카드 삭제	백엔드(서버), 프론트엔드(앱)	DELETE 요청을 통해 자신의 카드를 삭제한다.
카드 평가	백엔드(서버), 프론트엔드(앱)	카드에 평가 정보를 남긴다. 한번 평가한 카드는 다시 앱 화면과 API 요청에 보이지 않는다.
매칭 리스트	백엔드(서버), 프론트엔드(앱)	매칭된 사용자의 정보의 리스트를 받아서 디스플레이한다.
매칭 취소	백엔드(서버), 프론트엔드(앱)	매칭을 취소한다, 모든 평가 정보도 삭제된다.
쪽지 전송	백엔드(서버), 프론트엔드(앱)	매칭된 사용자에게 쪽지를 전송한다.
쪽지 조회	백엔드(서버), 프론트엔드(앱)	Socket을 통한 실시간 채팅이 아닌, 보낸 쪽지와 받은 쪽지를 API로 비실시간 조회한다.
관리자 페이지	백엔드(서버)	admin은 개발자 도움 없이 Web 사이트를 통해 업로드 하는 사진이나, 카드, 매칭에 개입하고 DB를 관리한다.
회원 토큰관리	백엔드(서버)	Token based authentication 으로 DB와 서비스 접근하는 권한을 Token을 통해 확인한다.
토큰 디스크 관리	프론트엔드(앱)	토큰을 디스크에 저장하여, 재 로그인을 못하게 한다.
토스트 알림	프론트엔드(앱)	API 결과를 앱내에 팝업으로 알린다
http 요청	프론트엔드(앱)	앱의 Http 통신

앱 서비스의 사용자는 다른 유저들의 프로필 카드를 탐색하게 되는데, 이때 DB에 있는 모든 프로필 카드를 탐색하는 방식이 아닌 필터를 통해 조건부 검색이 가능하다. 때문에 유저는 자신이 원하는 유저 정보를 찾을 수 있으며, 향후에는 추천시스템을 구축할 계획이다.

2.1.1 서비스 layer architecture 설명

- **백엔드 Django+uWSGI+Nginx**

Layer는 아래 <그림 3>와 같다.

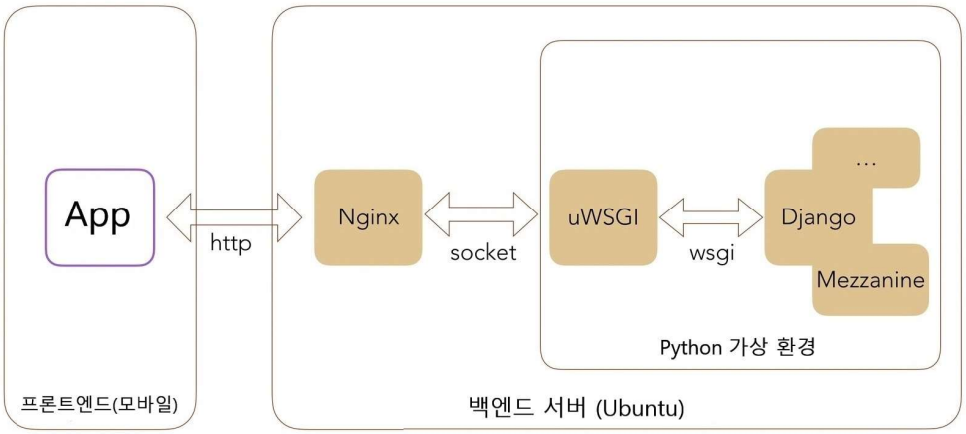


그림 3. 전체 서비스 개발의 Layer architecture

백엔드에서 가장 아래에 있는 Layer는 시스템으로 Ubuntu를 사용했다. Linux기반의 시스템이라면 모두 사용이 가능하며, Python3 와 Nginx 그리고 Mysql이 무리 없이 실행 가능해야 한다.

시스템 위의 Nginx는 웹 서버 소프트웨어로, 가벼움과 높은 성능을 목표로 한다[5]. Django는 기본 서버가 내장되어 있지만, 퍼블리싱하기에는 성능이 부족하다. Nginx를 통해서는 이미지 등 static파일을 간편하게 처리할 수 있으며, reverse proxy와 http server 기능을 수행한다. 클라이언트가 서버로 프록시 서버를 통해 요청하는 방식의 forward proxy는 프로세스 수만큼의 worker 수를 갖지만, 프록시 서버가 받은 응답을 클라이언트에게 전달하는 reverse proxy는 프로세스 수의 1/4 만큼의 worker 수를 갖는 특징이 있다. End Point가 다르고 실제 서버 정보를 알 수 없는 장점이 있다.

다음 Layer는 Python의 가상 환경이다. 이 환경에는 Django가 있고, 필요한 패키지들을 pip를 통해 설치하고 관리하며, 시스템 환경과 독립되어 있기에 버전관리가 쉽다. 서버에 수행하는 작업에 관련된 코드들은 Django를 통해 실행되지만, Django 기본 서버가 아닌 uWSGI를 통해서 서버를 구동한다. uWSGI는 하나의 웹서버이다. 이는 WSGI, uwsgi(소문자), http등의 프로토콜을 구현하고 있다. reverse proxy 인 Nginx는 우선 uwsgi를 거쳐야 한다.

마지막으로 다른 곳에서도 서버에 접근할 수 있도록, 공유기에 포트포워딩을 해준다. 또한 IP가 고정 적이지 않기 때문에 IP를 고정해야하는데, 공유기가 제공하는 DDNS를 통해 변동되는 IP를 고정한다.

- **프론트엔드 Flutter**

Layer는 아래 <그림 4>과 같다.

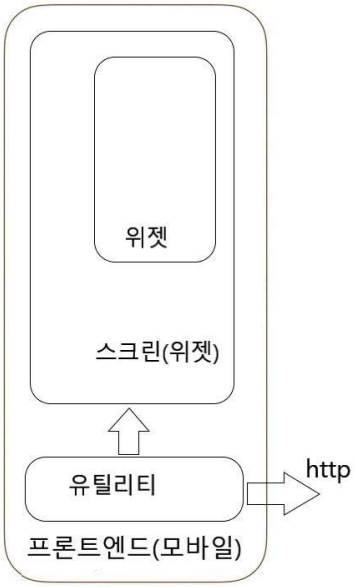


그림 4. 프론트 개발의 Layer architecture

Flutter는 구글에서 2017년 5월에 출시한 크로스 플랫폼 모바일/웹/데스크톱 UI SDK이다.[6] 사용되는 언어는 구글의 Dart이다. 컴파일된 프로그램은 안드로이드와 IOS상에 실행이 가능하다.

프론트엔드는 화면에 표시되는 요소들UI과, 요소들의 기능을 처리하는 Utils부분으로 Layer를 구분하여 개발한다. Flutter에서 UI 요소들은 위젯이라 부르며, 눈에 보이지 않지만 화면에 구성하는 레이아웃도 위젯이다. 위젯 중에서도 화면을 가리키는 가장 큰 위젯을 스크린(위젯)으로 정하고, 스크린(위젯)에 포함되는 부품 위젯들은 별도로 관리하고. 상위 Layer로 지정하여 관리 했다.

유틸리티에 포함되는 기능은 http 통신과 스크린 이동을 돕는 navigator와 디스크에 접근해서 읽고 쓰는 기능이 있다.

2.2 백엔드의 전반적인 구조

- 기능 구조

http 통신을 통해 Django에서 작업을 수행하고, Django는 DB에서 필요한 데이터를 가져온다.

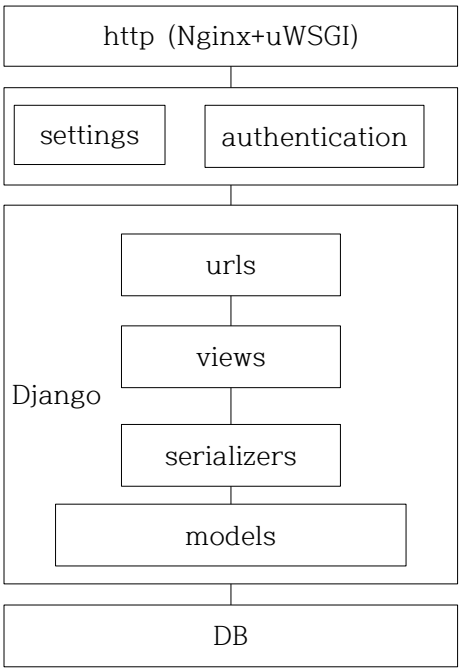


그림 5. 백엔드의 기능구조

DB의 테이블은 Django의 ORM을 통해 생성하게 되는데, 이때 사용하는 패키지가 models이다. models를 통해 객체를 생성하면, 생성된 객체의 정보를 Json이나 텍스트 형태의 데이터로 변환하게 되는데 이를 serializers가 수행하고 완료한 결과물을 views를 통해 API의 로직을 구성하고 urls 주소가 접근 가능하게 한다. API를 사용하기 위해서는 요청자는 반드시 Token을 Header에 포함해야 된다. uWSGI가 하나의 웹서버로 구동하는데, 외부에서는 uWSGI에 직접적으로 접근할 수는 없고, reverse proxy와 http server 역할을 하는 Nginx를 통해 웹서버 요청을 한다.

해당 구조를 사용하는 이유는 웹서버의 MVC 설계패턴과 유사한 Django의 MTV 스타일의 설계패턴을 사용하기 때문이다. 또한 객체를 DB의 테이블로 변환해주는 ORM(Object-Relational Mapping)을 사용했기 때문에 별도의 DB 스키마 관리나 테이블 관리기법이 없다. 이 구조의 효과는 Django의 기본 서버가 아닌, Nginx를 사용함으로써 보다 안정적인 통신을 제공하고, ORM방식의 테이블 설계를 통해 빠른 개발 속도를 기대할 수 있다. 특히 복잡한 검색이나, 관계형 테이블의 생성을 간단하게 구현할 수 있다.

- 데이터 스키마

DB는 MySQL을 사용하며, 아래 <그림 6>은 Django의 ORM으로 생성한 MySQL DB 스키마이다. 모든 테이블은 PK를 갖고 있고, 토큰과 메시지, 매칭과 카드 그리고 이미지 테이블이 유저의 정보를 담고 있는 account 테이블과 One to Many 관계를 갖고 있다. 프로필카드의 정보를 담고 있는 profilecard 테이블은 3개의 Many to Many 관계를 스트리머 정보와 커뮤니티, 이미지 테이블과 갖고 있다. 이미지는 단한장만 업로드 하지만, 향후에 여러 장 업로드하게 될 수도 있다 생각하여 MTM으로 연결했으며, 커뮤니티와 스트리머 정보는 하나의 프로필카드에 여러 개를 갖기에, MTM으로 연결하였다.

조건부 검색과 전체 리스트를 뽑을 경우, MTM와 OMT는 join을 수행하게 되어, DB에 두 번 이상 접근하지 않는다. 하지만, 카드를 생성하게 될 경우, join이 불가능하여 여러 테이블에 접근하게 되면서 두 번 이상 접근하게 된다.

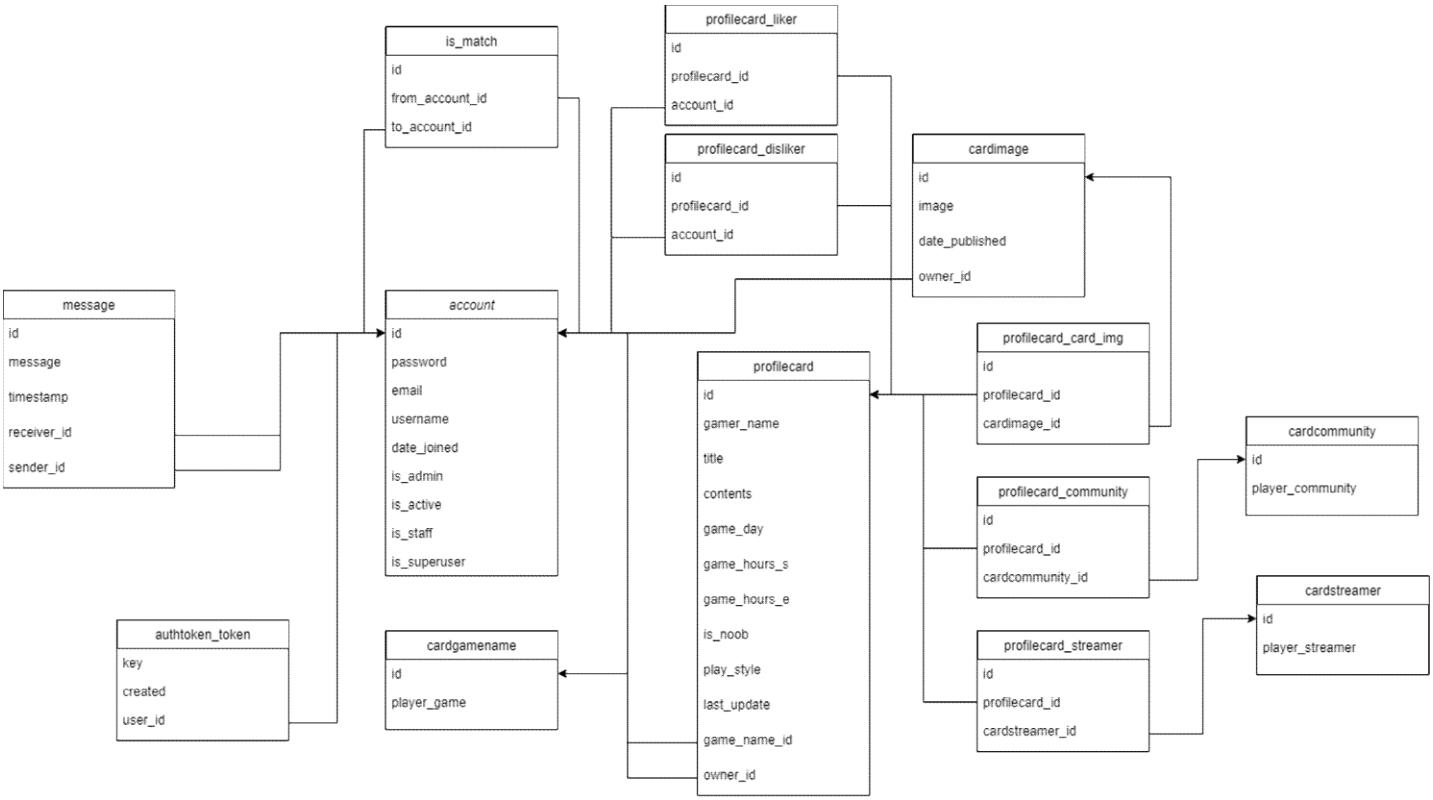


그림 6. 백엔드 DB 데이터스키마

2.2.1 모델 패키지

models 패키지에는 DB테이블을 객체로 생성하고 관리하는 기능을 맡고 있다.

serializers는 객체로 받은 테이블의 정보를 한 번 더 python data type 으로 변환해준다. 변환한 값은 json형태로 바꾸며, 객체의 정보를 읽고 객체로 쓰는 작업을 serializers에서 수행한다. 특히 Many to Many 관계처럼 한 번에 관계된 테이블을 join하여 가져오는 작업을 수행한다.

views는 models와 serializers 패키지를 활용하여 CRUD작업을 수행하고, 클라이언트에게 전송할 데이터를 관리한다. views는 애플리케이션의 로직을 넣는 곳이라 생각하면 된다.

urls는 요청의 주소를 설정하여 views로 보낸다.

settings와 admin의 위의 4가지 패키지를 통합을 설정하거나 관리하는데 사용된다. views에서 요청을 받을 때마다 토큰을 확인한다면, 매번 코딩하여 기록하기보다는 settings에 설정을 기록하면, 전체에 적용된다.

- **Django ORM and Model**

Django ORM은 객체(Object)의 관계(Relational)를 연결(Mapper) 해주는 것을 의미한다. 즉 python의 객체로 DB 구현하고 관리하게 되는 것이고, SQL 쿼리문이 없어도 DB의 데이터를 다룰 수 있게 된다.

표 2. Django의 Account 모델과 DB에 생성된 테이블

Field	Type	Null	Key	Default	Extra
id	bigint	NO	PRI	NULL	auto_increment
password	varchar(128)	NO		NULL	
email	varchar(60)	NO	UNI	NULL	
username	varchar(11)	NO	UNI	NULL	
date_joined	datetime(6)	NO		NULL	
last_login	datetime(6)	NO		NULL	
is_admin	tinyint(1)	NO		NULL	
is_active	tinyint(1)	NO		NULL	
is_staff	tinyint(1)	NO		NULL	
is_superuser	tinyint(1)	NO		NULL	

```
class Account(AbstractBaseUser):
    email = models.EmailField(verbose_name="email", max_length=60, unique=True)
    username = models.CharField(max_length=11, unique=True)
    date_joined = models.DateTimeField(verbose_name='가입일', auto_now_add=True)
    last_login = models.DateTimeField(verbose_name='마지막 로그인', auto_now=True)
    is_admin = models.BooleanField(verbose_name="어드민", default=False)
    is_active = models.BooleanField(verbose_name="활동중", default=True)
    is_staff = models.BooleanField(verbose_name="스태프", default=False)
    is_superuser = models.BooleanField(verbose_name="슈퍼유저", default=False)
    is_match = models.ManyToManyField(settings.AUTH_USER_MODEL, verbose_name="매칭된사람",
blank=True)
    ...
```

생성된 Account 모델은 AbstractBaseUser를 상속받고, settings에서 AUTH_USER_MODEL을 Account으로 지정해주면, Django는 Account 모델을 다른 모델과 달리 프로젝트에서 사용할 계정 정보를 담는 모델로 취급을 하고, 비밀번호는 암호화하여 저장하고 토큰을 생성해준다.

ORM을 사용함으로, 데이터베이스의 테이블을 객체지향 프로그래밍에서 흔히 사용하는 객체(Class)처럼 사용할 수 있으며, 기존 쿼리문을 작성하여 데이터베이스를 조작하는 것을 넘어서서 더 효율적이고 가독성 및 유지보수에 적합한 코드를 만들 수 있다. 특히 게이머 매칭 서비스는 관계형 테이블을 사용하기 때문에 더욱 간편하게 접근할 수 있다는 장점을 갖고 있다.

- **API and view**

Django의 view는 로직을 담당하고, REST API의 CRUD를 위해 많은 패키지를 호출해야한다. 그중 핵심적인 두 패키지는 model와 serializers이다. 토큰을 사용하여 permission을 확인하기 때문에, view에서는 header에 올바른 토큰이 전달되었는지 확인해줘야 한다.

API는 로그인과 로그아웃 카드 생성 등, 총 12개가 되며, 사용자가 API를 통해 요청을 보내면 작업을 ORM을 통해 DB에 접근하고 수행 결과를 응답해준다.

- 프로필카드 생성

경로: POST /create

요청:

```
{
    “title”: ‘this is title’,
    “contents”: ‘this is contents’,
```

```
    “game_day”: ‘0101011’,
    ...
    “streamer[]”: “[‘streamer1’, ‘streamer2’]”,
    “community[]”: “[‘community1’, ‘community2’]”,
    “images[]”: image data,
}
```

응답:

```
{
    “title”: ‘this is title’,
    “contents”: ‘this is contents’,
    “game_day”: ‘0101011’,
    ...
    “streamer[]”: “[‘streamer1’, ‘streamer2’]”,
    “community[]”: “[‘community1’, ‘community2’]”,
    “response”: ‘create success’,
}
```

코드:

```
@api_view(['POST'])
def create_card(request):
    data = {"owner": request.user,
            "title": request.data["title"],
            ...
            "images": request.data.getlist("images[]"),
            }

    images = data.pop('images')
    community_data = data.pop('community')
    streamer_data = data.pop('streamer')
    game_name = data["game_name"].replace(" ", "")
    # 띄어쓰기 제거
    data["game_name"] = CardGameName.objects.get_or_create(player_game=game_name)[0]
    temp_list = [[], []]
    temp_list[0] = community_exist(community_data[:10])
    temp_list[1] = streamer_exist(streamer_data[:10])
    images_list = images_url(images, data["owner"])

    p_card = ProfileCard.objects.create(**data)
    try:
        p_card.community.add(*temp_list[0])
        p_card.streamer.add(*temp_list[1])
        p_card.card_img.add(*images_list)
    except:
        p_card.delete()
        data = {'response': 'create fail.'}
        return Response(data)

    data.pop('owner')
    data["game_name"] = game_name
    data["community"] = community_data
    data["streamer"] = streamer_data
    data['response'] = 'create success.'

    return Response(data)
    ...
```

view 중의 하나인 프로필카드의 API요청응답과 구성하는 코드이다. 요청해서 받은 request의 data를 하나씩 받아서 models의 ORM을 호출하여 객체를 생성하고, MTM을 이어 주고 DB테이블에 최종 반영을 하고 응답 데이터를 전송한다.

2.3 프론트엔드의 전반적인 구조

- 기능 구조

아래 <그림 7>에서 가장 위의 Screen은 사용자에게 가장 가까우며, 가장 큰 단위의 위젯을 Screen이라 하고, Screen안에는 부품과 같이 여러 위젯들이 있다. Screen은 색상 정보와 디스플레이 되는 모델 등을 Tools을 통해 Utils에 요청하게 된다. Tools가 담고 있는 정보는 정적인 색상 정보 palette와 urls 주소 정보 그리고 동적인 정보를 맞춰주는 model 틀(객체)이 있다. Utils에서 실제로 동작이 수행되는데, 디스크에 쓰고 읽는 SP와 http요청을 맡는 dio가 있다. navigator는 screen의 이동을 돕고, toast는 정보의 알림 역할을 맡는다.

구조를 기능과 UI로 나눠서 설계한 이유는, 코드를 중복해서 사용하는 것을 방지하고, 디버깅시 문제가 기능의 충돌인지, 화면에 보이는 위젯들의 문제인지 쉽게 파악하기 위함이다.

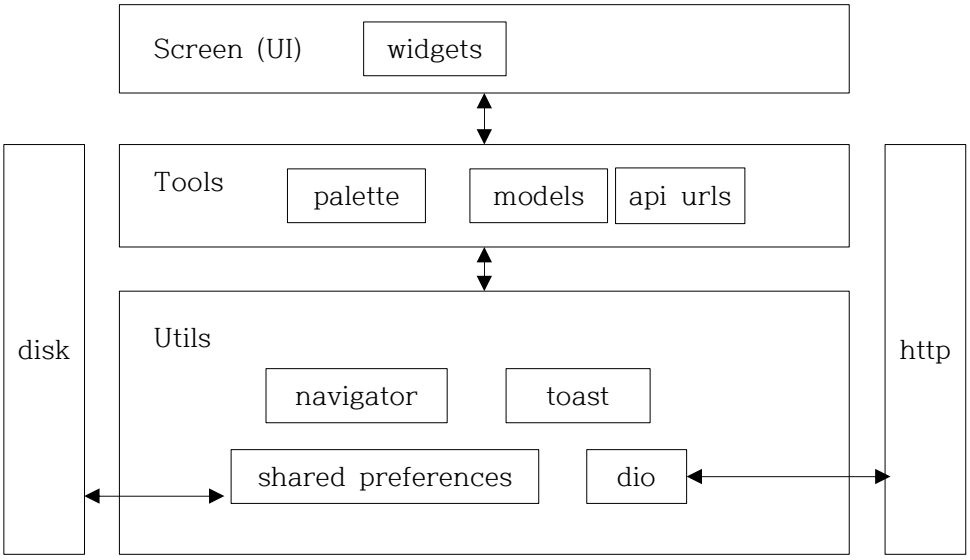


그림 7. 프론트엔드의 기능구조

- 컴포넌트들 간의 관계

파일은 기능별로 다른 디렉토리에 두고도 했지만, 파일 이름에도 "_screen", "_utils" 형식으로 기능 별로 구분을 하였고, index_screen이 최초에 앱이 실행되면 접근하는 페이지이며, 해당 페이지에서 디스크에 토큰이 있는지 여부를 확인하고, 없을 경우 로그인 화면으로 있고 유효한 토큰일 경우 홈 화면으로 이동하게 된다.

홈 화면에서는 nav_screen이 tab bar를 통해 screen 이동을 하고, nav_screen 안에 포함되어 있는 screen 이기에, 화면을 두 번 접근하지는 않는다. home, user, match, settings 스크린은 가장 큰 단위의 스크린이고, 이들 하위로 다른 스크린의 이동이 있다. 더 자세한 관계는 아래 <그림 8>을 확인할 수 있다.

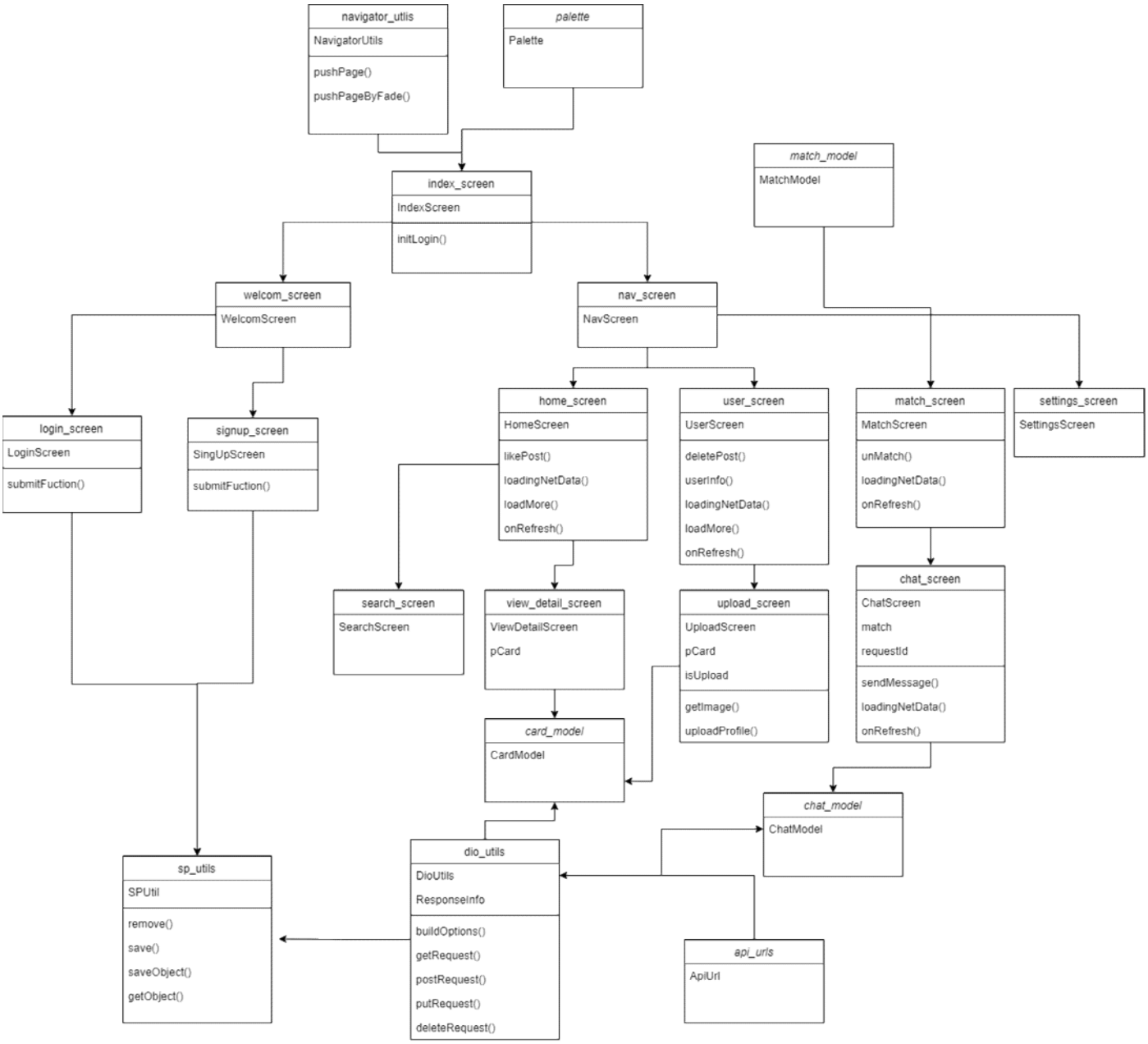


그림 8. 프론트엔드의 컴포넌트 관계

2.3.1 패키지

- **HTTP와 Dio**

Dio는 dart 언어의 Http client 패키지로 Rest API과 FromData, 요청취소, Cookie관리, 문서 업로드, 다운로드 등의 기능을 제공한다. Dio 패키지를 호출하여 바로 사용하는 것도 가능하나, 객체를 통해 호출하고 파라미터를 미리 정의하고 관리할 수 있다.

```
class DioUtils {
  Dio _dio;

  // 펙토리 모드
  // A factory constructor can check if it has a prepared reusable instance
  // in an internal cache and return this instance or otherwise create a new one.
  factory DioUtils() => _getInstance();
  static DioUtils get instance => _getInstance();
  static DioUtils _instance;

  static DioUtils _getInstance() {
    if (_instance == null) {
      _instance = new DioUtils._internal();
    }
    return _instance;
  }

  DioUtils._internal() {
    BaseOptions options = new BaseOptions();
    options.connectTimeout = 20000;
    options.receiveTimeout = 2 * 60 * 1000;
    options.sendTimeout = 2 * 60 * 1000;
    // 초기화 옵션 설정
    _dio = new Dio(options);
  }
    ...
```

앱에서 Http 요청이 가장 빈번하게 일어나기 때문에, 객체를 통해 dio를 관리하면 기본 timeout 옵션과 토큰을 전달하는 Header 옵션 등을 통합적으로 설정할 수 있다.

```
  Future<ResponseInfo> errorController(e, StackTrace s) {
    ResponseInfo responseInfo = ResponseInfo.error();
    if (e is DioError) {
      DioError dioError = e;
      switch (dioError.type) {
        case DioErrorType.CONNECT_TIMEOUT:
          responseInfo.message = "CONNECT_TIMEOUT_ERR";
          break;
        case DioErrorType.SEND_TIMEOUT:
          responseInfo.message = "SEND_TIMEOUT_ERR";
          break;
        case DioErrorType.RECEIVE_TIMEOUT:
          responseInfo.message = "RECEIVE_TIMEOUT_ERR";
          break;
        case DioErrorType.RESPONSE:
          responseInfo.message = "RESPONSE_ERR";
          break;
        case DioErrorType.CANCEL:
          responseInfo.message = "CANCEL_ERR";
          break;
        case DioErrorType.DEFAULT:
          responseInfo.message = "DEFAULT_ERR";
          break;
      }
    } else {
      responseInfo.message = "Error";
    }
    return Future.value(responseInfo);
  }
```

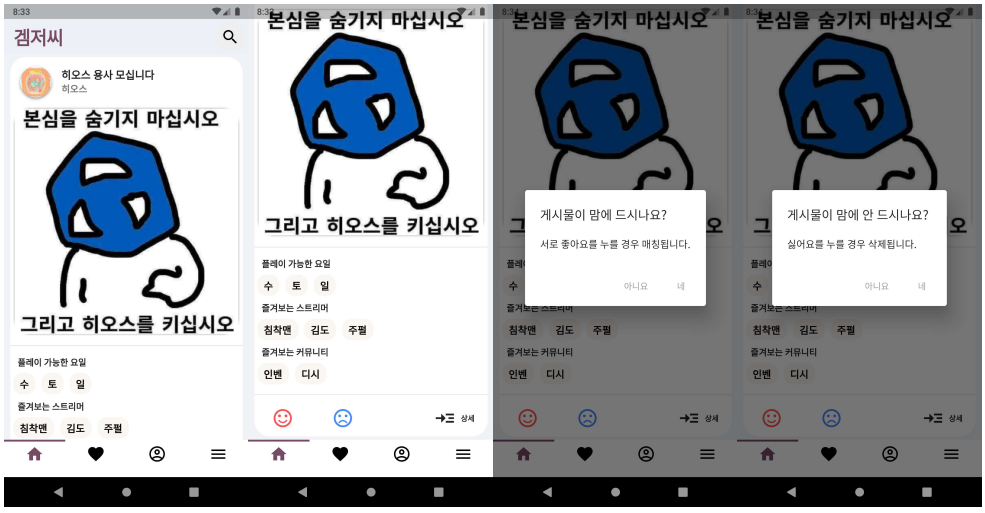
ResponsInfo 객체는 http 요청에 반환 값을 저장하는 객체이다. dio의 error type을 case별로 규칙을 만들어, 반환 값을 저장하는 객체에 전달하면, 디버깅 시 어떤 문제가 있었는지 보다 효율적이게 확인할 수 있다.

2.3.2 API정보와 스크린

- 시작화면 (로그인)

사용된 API:
로그인: GET http://hongjinhj.iptime.org/Api/login
회원가입: POST http://hongjinhj.iptime.org/Api/register

- 홈화면

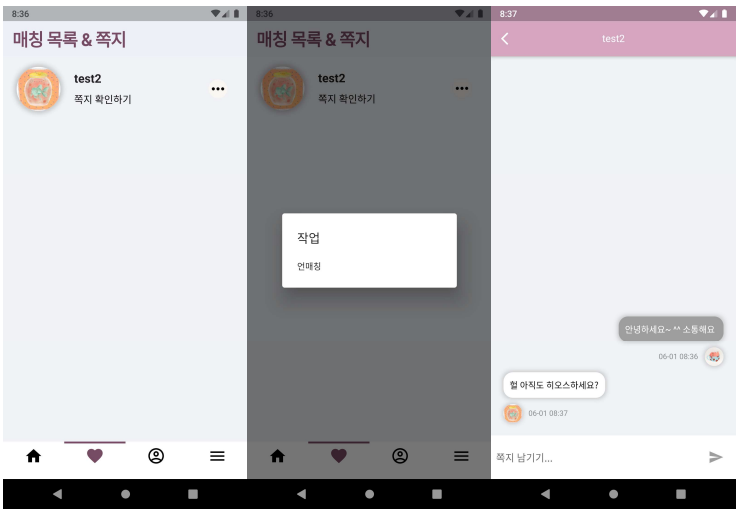


사용된 API:
홈리스트: GET http://hongjinhj.iptime.org/Api/list
평가: PUT http://hongjinhj.iptime.org/Api/like

- 상세페이지와 작성자 정보, 홈리스트 상세검색

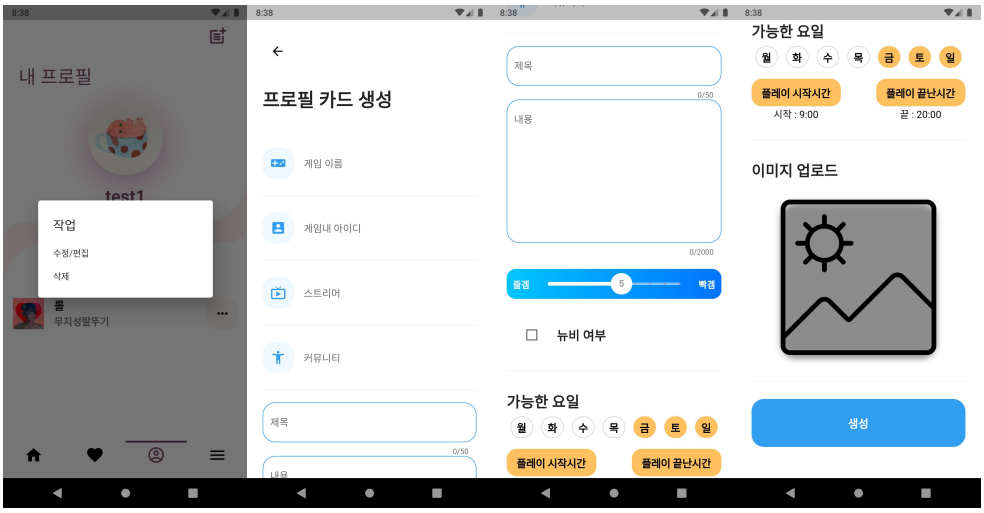
사용된 API:
상세페이지: 없음
작성자 정보: GET http://hongjinhj.iptime.org/Api/list?owner__id=3
상세검색: GET http://hongjinhj.iptime.org/Api/list?game_name__player_game=
&streamer_in=&community_in&is_noob=

- 매칭페이지와 채팅페이지



사용된 API:
매칭리스트: GET http://hongjinhj.iptime.org/Api/match
언매칭: PUT http://hongjinhj.iptime.org/Api/unmatch
쪽지리스트1: GET http://hongjinhj.iptime.org/Api/messages/2/3
쪽지리스트2: GET http://hongjinhj.iptime.org/Api/messages/3/2
쪽지보내기: POST http://hongjinhj.iptime.org/Api/send

- 개인 프로필과 카드 관리



사용된 API:
내정보 조회: GET http://hongjinhj.iptime.org/Api/list?getUser=1
삭제: DELETE http://hongjinhj.iptime.org/Api/delete
생성: POST http://hongjinhj.iptime.org/Api/create
수정: DELETE http://hongjinhj.iptime.org/Api/delete
POST http://hongjinhj.iptime.org/Api/create

3. 결과 분석

Django를 통한 REST API 개발은 꽤 빠른 속도를 자랑한다. 코드가 단순하고 직관적이기 때문이라 생각한다. 특히 ORM은 개발속도를 높이는데 많이 도움이 됐다. 뿐만 아니라 Django에서 기본으로 제공하는 admin 페이지 덕분에, 테스트하거나 관리하는 것 도 간편했다. 하지만 단순한 만큼 기능적으로 많이 빈약한 부분이 있었다. 특히 DB를 접근하는 쿼리가 문제였다. 잘못된 방식으로 Django를 사용하게 될 경우 겉으로는 간편하고 순조롭게 구동되는 것 같지만, 내부적으로는 비효율적이게 동작하는 것을 알 수 있다.

Setting에서 Django가 DB에 접근할 때 사용하는 쿼리를 log에 출력하게 설정하고 쿼리를 검사하면서 개발하면, 비효율적인 동작을 줄이고 Django의 최대 성능을 살리는 방향으로 코딩이 가능하다. 단 시간 내에 많은 기능을 갖춰야하고, 빠르게 서비스를 시작하는 것을 목적으로 두고 있다면, Django는 나쁘지 않은 선택이라고 생각한다.

Flutter의 위젯 아키텍처의 장점은 접근을 위해 복잡한 이해가 필요 없다는 것이라 생각한다. 모든 것이 다 위젯이기에, 복잡한 생각 없이 막힘없이 이해하고 학습이 가능하다. 다만 어디까지나 위젯을 정확하게 이해하지 못했기 때문이다. 위젯을 활용하여 복잡한 구조를 만들기 위해 같은 위젯들 사이에 구분을 주기 시작하면, 위젯의 내부 동작에 대한 이해가 부족하다는 것을 느낄 수 있다.

한 가지 언어 한 가지 플랫폼으로 웹, IOS, 안드로이드 모두 개발이 가능하다는 점에서 개발속도와 효율성은 높다고 평가할 수 있다. 패키지를 관리하는 방식도 간편하고, 추가 삭제 업데이트 등이 모두

간편하게 가능하다. 또한 Hot reload를 지원하기 때문에, 개발하면서 화면을 즉시 확인할 수 있는 장점이 있다. 주의할 점은 화면을 init하는 함수는 Hot Reload에 의해 호출되지 않는다.

4. 결론 및 토론

구현한 앱 서비스의 API 서버에는 많은 문제점을 갖고 있다. 특히 토큰만 의존하는 방식의 권한 처리는 설령 HTTPS를 사용하더라도 매우 위험할 것이라 생각한다. 또한 토큰이 암호화 되지 않은 상태로 DB에 저장되어 있어, 관리자가 토큰 테이블에 접근할 수 있는 위험도 있다. 서버의 보안은 더 향상할 필요가 있다. views에서 CRUD 요청에 따라 작업을 수행하는데, 사용자는 항상 올바른 요청을 하지 않기에, 잘못된 요청에 맞는 핸들링이 부족하다, 에러코드를 더 정하고 요청에 어떤 부분이 잘못됐는지 어느 정도 피드백이 된다면 좋을 것 같다.

프론트 앱의 문제점은 Flutter에 대한 개발 경험이 부족으로 기능이 많이 빈약하다는 것이다. 유저 친화적이지 않고, 반복되는 스크린이 많이 등장한다. 백그라운드 푸시 알림기능이나 첫 설치 시 안내와 같은 부분이 없고 UI가 직관적이지 않다. 개발한 앱은 안드로이드 APK 파일로 잘 build 되었으나, IOS는 MacOS에서 build 테스트를 하지 않았다.

기획한 매칭 서비스에서 생각할 수 있는 단점은 message 부분이다. 채팅기능을 socket으로 구현하지 않아서 메시지와 알림을 보내는 기능이 없다. 또한 유저들 간의 매칭을 위해 사전에 세운 가설과 데이터를 갖고 분석하는 작업이 빠져 있어, 서비스를 세운 가설 자체가 실제 데이터 분석을 하지 않았기 때문에 허술하고 빈약한 느낌을 많이 받는다.

모바일서비스 개발을 쉽게 생각했다는 것이 가장 큰 실수였던 것 같다. 머릿속으로 생각하면 단순한 작업일 것 같지만, 실제로 구현하기 위해서는 학습량도 많고 정보에 대한 접근이 어려운편 인걸 알았다. 복잡한 서비스 구조를 단계별로 풀어서 개발하고 관리하는 부분이 많이 아쉬웠다. 앞으로 이번에 개발한 앱 서비스를 스토어에 런칭이 가능한 수준까지 학습하고 기존에 파악한 문제점들을 해결하고 싶다. 어느 정도 활발한 플랫폼을 갖게 되면 그 안에서 생기는 데이터를 이용해서 가설을 세우고 증명을 하고, 비즈니스에 활용되면 좋을 것 같다고 생각한다.

참고문헌 - References

1. 김선진. 낚시성 기사 제목의 활용 실태 연구. 디지털디자인학연구, 10(4), 283-293. 2010

2. 한경훈 외 6명 인터넷 방송을 이용한 게임 마케팅 전략에 관한 연구. 한국컴퓨터정보학회 학술발표논문집, 25(2), 66-68, 2017

3. 조창희,심재웅,조대곤. 관계 형성욕구가 1인 미디어 방송의 자발적 기부 행태에 끼치는 영향 : 성별 간의 상호작용을 중심으로. 한국경영정보학회 2019년 경영정보관련 추계학술대회, 303-310. 2019

4. Malisi, S., Suharsono, S., & Setiawan, S. Language and Identity in Online Gamer Community. Journal of English Language and Literature, 8(617), 10-17722. 2017

5. Wikipedia, Flutter, https://ko.wikipedia.org/wiki/Nginx

6. Wikipedia, Flutter, https://ko.wikipedia.org/wiki/플러터