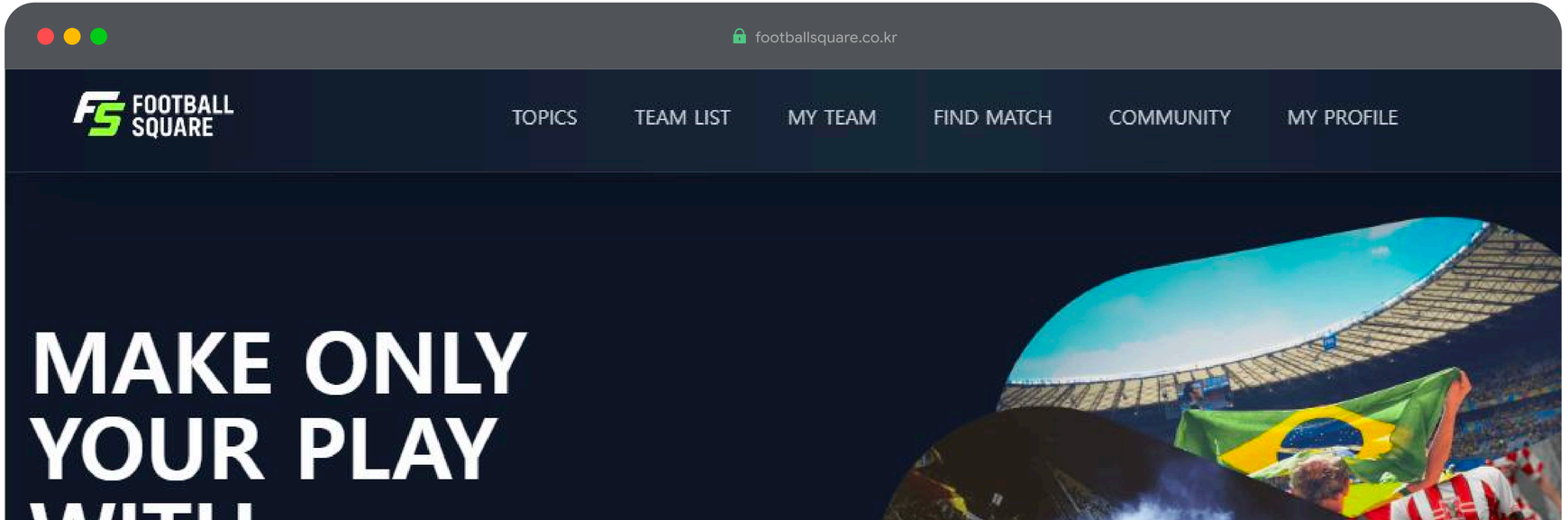
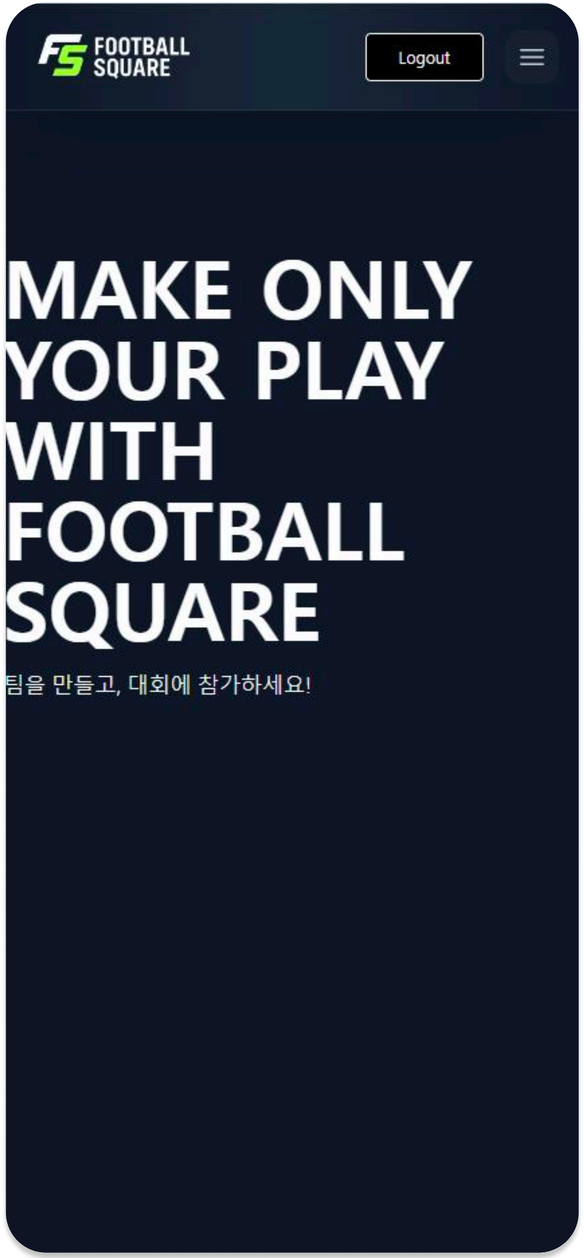
풋볼 광장

대한민국 최대 규모의 FC 시리즈 프로클럽 커뮤니티 KFPL에
단일화된 커뮤니티 관리 플랫폼을 제공하여, 혁신적인 유저경험과 압도적인 편의성 제공을 위한 프로젝트입니다.

3인 팀 프로젝트 팀장 서비스 기획 & 개발 총괄 2025.02 ~ 2025.07





Period

2025.02 ~ 2025. 07

Link

https://footballsquare.co.kr/

https://github.com/footballSquare/backend

Skills

Member

김연호 , 백엔드 개발자, PM

류동호, 프론트엔드 개발자, PL

한만욱, 프론트엔드 개발자

개인 기여

- 프로젝트 역 기획서 제작, 초기 빌드업
- API 명세서, 요구사항 명세서 작성
- DB 스키마 설계
- 팀, 매치, 커뮤니티, 대회, 수상, 기록 API 제작
- AWS 배포
- docker 컨테이너 셋업
- 도메인 연결 및 https 연결
- 웹소켓 기반 실시간 메시징 서비스 구현

Background

기존의 KFPL 커뮤니티는,
네이버 카페, 디스코드, 카카오톡 오픈 채팅, 외부 웹 서비스들을 병용하며 **분산 플랫폼의 비효율성**에 직면해있었습니다.
이러한 커뮤니티의 문제점을 파악하여 해당 커뮤니티에 **서비스 개발을 역제안**하며 개발을 시작하게 되었습니다.

Problem

❶ 분산 플랫폼 사용

인증 / 인가, 커뮤니티 기능, 팀별 의사 결정, 대회 개최 등의 기능이 각각 다른 플랫폼에서 파편회 되어있었습니다.

❷ 신규 유저 유입 및 팀원 확보 비효율성

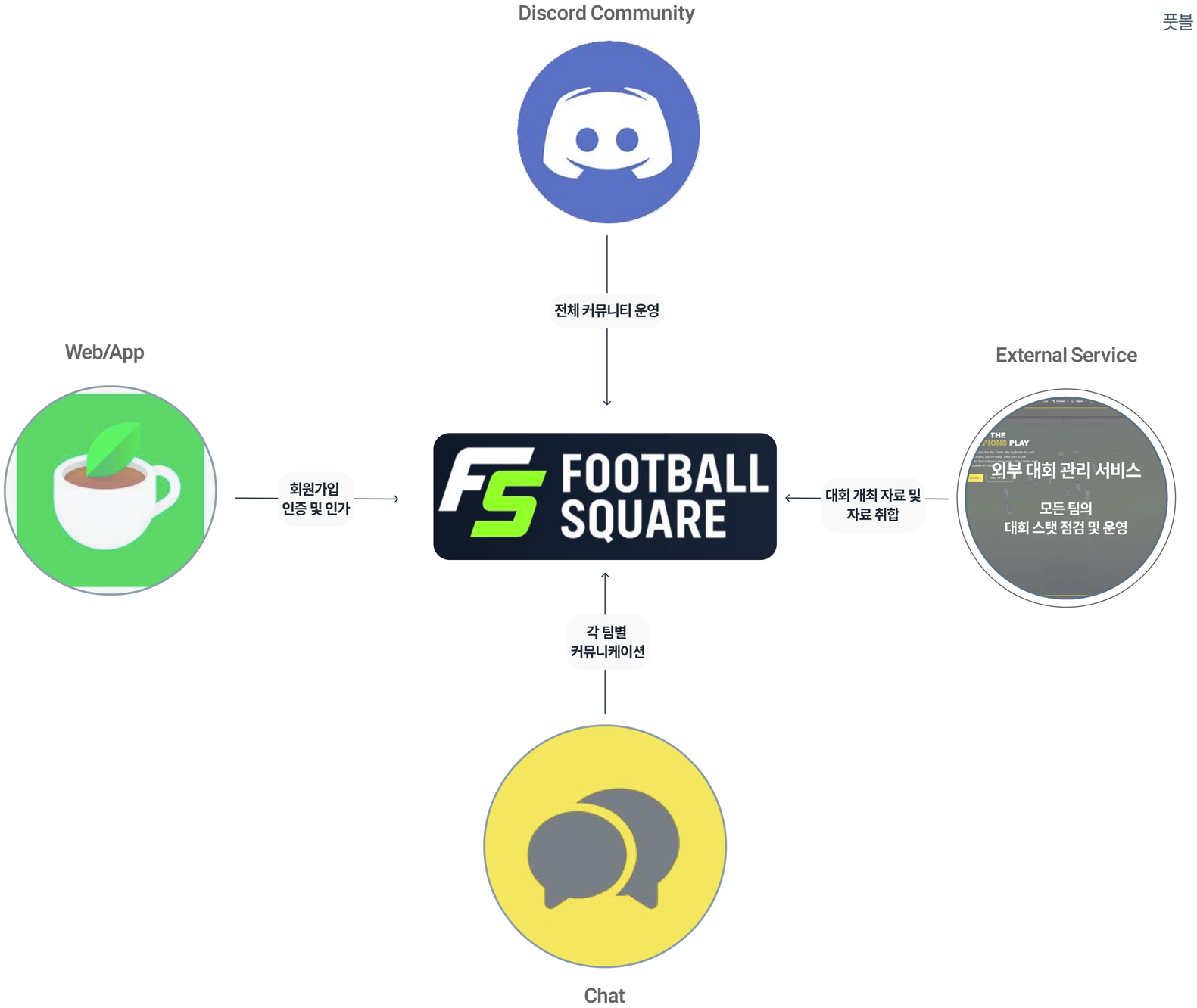
커뮤니티의 핵심인 팀원 모집과 확보 과정이 분산된 커뮤니티 내에서 표준화된 시스템 없이 이루어지고 있었습니다. 이는 신규유저에게 진입장벽으로 작용합니다.

❸ 커뮤니티 내 사용자 경험 및 재미요소 부족

팀원간 내부 의사결정이 번거롭고, 팀워크 강화에 필요한 유기적인 상호작용이 어려웠습니다. 또한 성과가 프로필 등으로 연계되지 않아 성취감을 느끼기 어려웠습니다.

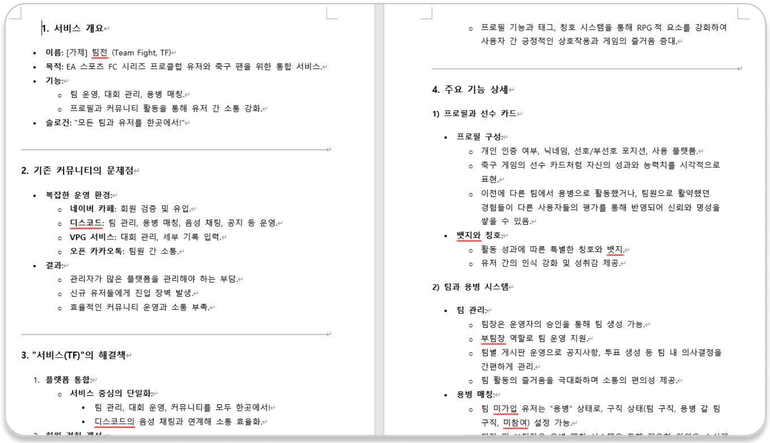
❹ 대회 기록, 운영의 어려움

독자적인 대회 기록을 저장 및 운영 해줄 서비스의 부재로 대회 진행의 연속성이 부재하였습니다. 또한, 운영 과정 또한 시스템이 아닌 운영진과 사용자의 직접적인 상호작용을 요구하는 방식으로 신뢰성이 떨어지는 문제가 있었습니다.



Prepare

이미 존재하는 **커뮤니티의 편의성**을 위한 서비스를 제작하는 만큼,
해당 커뮤니티 운영진들과의 소통과, 소통을 기반으로 하는 **요구사항 도출**이 중요했습니다.
지속적인 회의와 **체계적인 문서작업**으로 요구사항을 서비스에 명확하게 반영하기 위해 노력하였습니다.



1. 서비스 개요

- 이름: [가제] 팀전 (Team Fight, TF)
- 목적: EA 스포츠 FC 시리즈 프로클럽 유저와 축구 팬을 위한 통합 서비스.
- 기능:
  - 팀 운영, 대회 관리, 용병 매칭.
  - 프로필과 커뮤니티 활동을 통해 유저 간 소통 강화.
- 슬로건: "모든 팀과 유저를 한곳에서!"

2. 기존 커뮤니티의 문제점

- 복잡한 운영 환경:
  - 네이버 카페: 회원 검증 및 유입.
  - 디스코드: 팀 관리, 용병 매칭, 음성 채팅, 공지 등 운영.
  - VPG 서비스: 대회 관리, 세부 기록 입력.
  - 오픈 카카오톡: 팀원 간 소통.
- 결과:
  - 관리자가 많은 플랫폼을 관리해야 하는 부담.
  - 신규 유저들에게 진입 장벽 발생.
  - 효율적인 커뮤니티 운영과 소통 부족.

3. "서비스(TF)"의 해결책

1. 플랫폼 통합:
   - 서비스 중심의 단일화:
     - 팀 관리, 대회 운영, 커뮤니티를 모두 한곳에서!
     - 디스코드의 음성 채팅과 연계해 소통 효율화.

- 프로필 기능과 태그, 칭호 시스템을 통해 RPG적 요소를 강화하여 사용자 간 긍정적인 상호작용과 게임의 즐거움 증대.

4. 주요 기능 상세

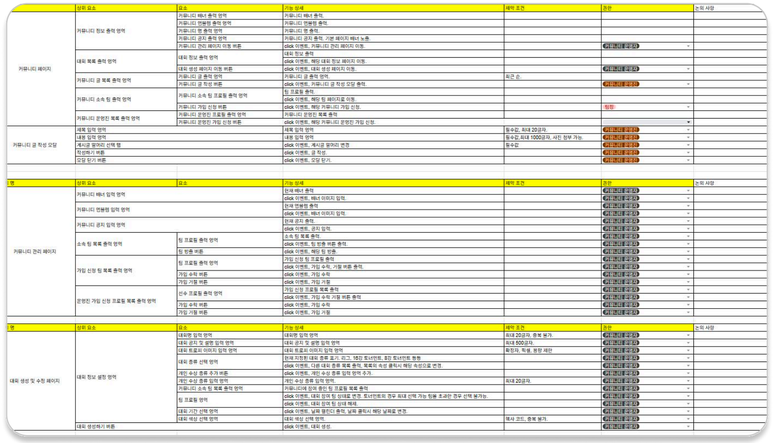
1) 프로필과 선수 카드

- 프로필 구성:
  - 개인 인증 여부, 닉네임, 선호/부선호 포지션, 사용 플랫폼.
  - 축구 게임의 선수 카드처럼 자신의 성과와 능력치를 시각적으로 표현.
  - 이전에 다른 팀에서 용병으로 활동했거나, 팀원으로 활약했던 경험들이 다른 사용자들의 평가를 통해 반영되어 신뢰와 명성을 쌓을 수 있음.
- 뱃지와 칭호:
  - 활동 성과에 따른 특별한 칭호와 뱃지.
  - 유저 간의 인식 강화 및 성취감 제공.

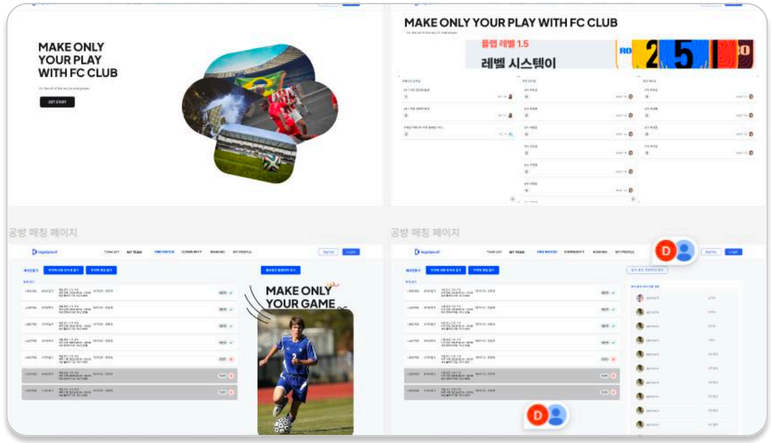
2) 팀과 용병 시스템

- 팀 관리:
  - 팀장은 운영자의 승인을 통해 팀 생성 가능.
  - 부팀장 역할로 팀 운영 지원.
  - 팀별 게시판 운영으로 공지사항, 투표 생성 등 팀 내 의사결정을 간편하게 관리.
  - 팀 활동의 즐거움을 극대화하며 소통의 편의성 제공.
- 용병 매칭:
  - 팀 미가입 유저는 "용병" 상태로, 구직 상태(팀 구직, 용병 같 팀 구직, 미참여) 설정 가능.

▸ 서비스 기획 역 제안 문서

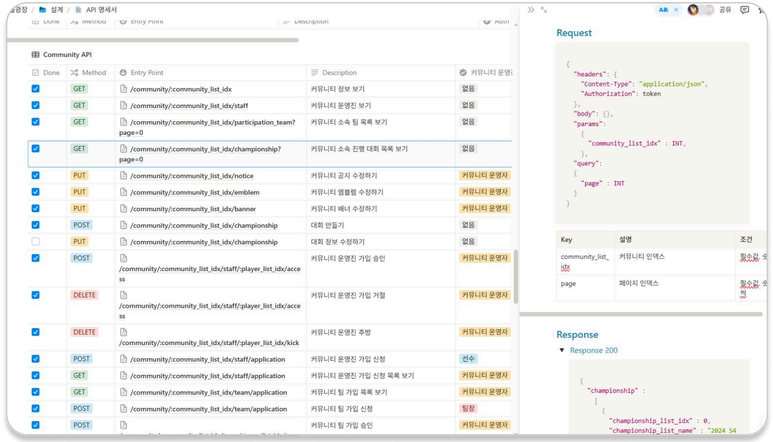


▸ 기능 명세서

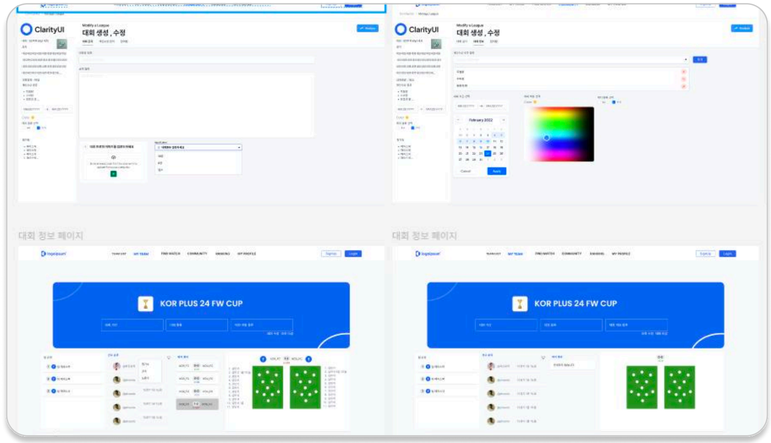


▸ 화면 설계서 1

▸ 서비스 기획 역 제안서 PPT



▸ API 명세서

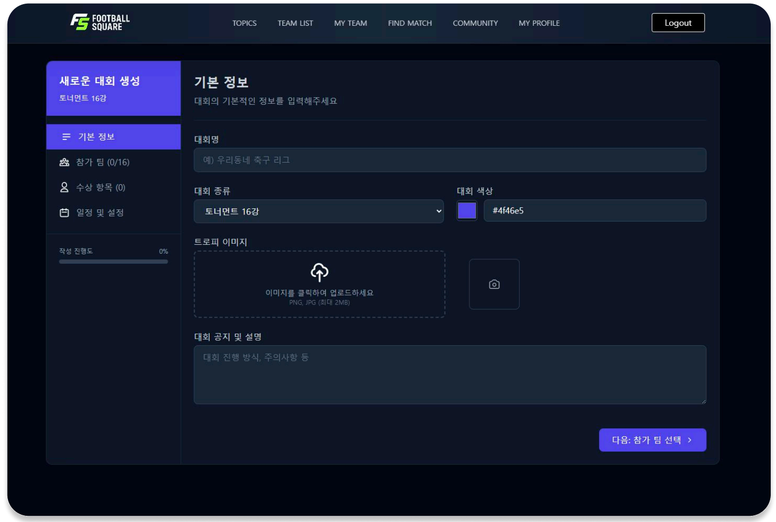


▸ 화면 설계서 2

Role

▸ 커뮤니티 운영자

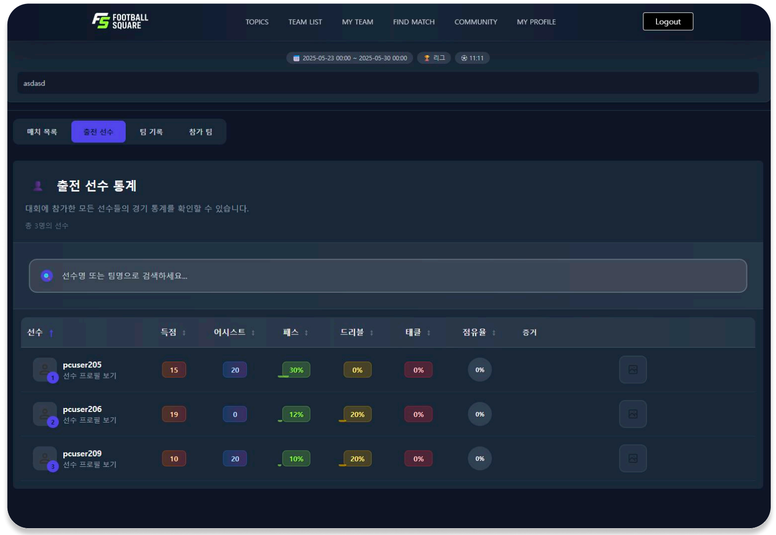
커뮤니티에 소속된 팀 및 운영진들의 가입, 추방등을 유연하고 편하게 관리할 수 있습니다.
대회 개최, 관리등을 **획일화된 시스템으로 편리**하게 진행할 수 있습니다.



- **일괄된 대회 관리 시스템** 구축
- **신뢰도 있는 대회 데이터 저장** 시스템 구축
- 편리한 **커뮤니티 소속 팀, 운영진 관리 시스템** 구축

▸ 커뮤니티 운영진

대회 관리, 대회 데이터 통합 점검을 편리하게 할 수 있습니다.
기존에 수기로 진행하던 회원 가입 관리를 일관된 시스템에 일임할 수 있습니다.

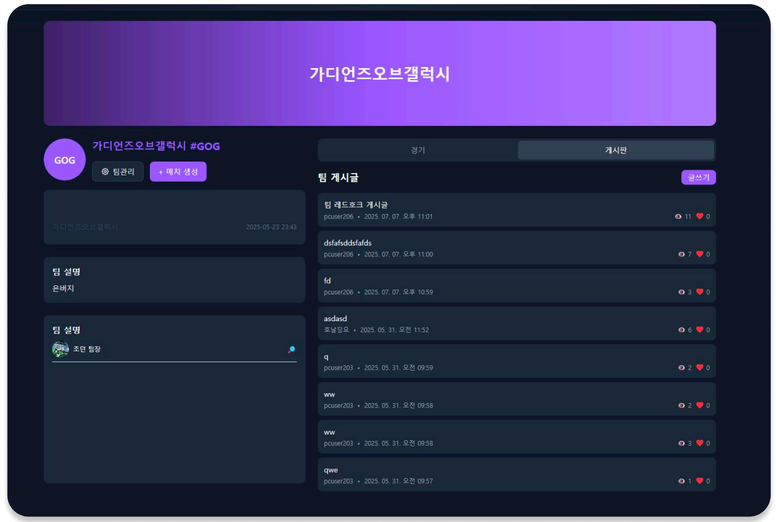


- 모든 대회 참여 팀과 선수들의 데이터를 종합 노출
- **휴대전화 인증 도입**으로 1인 1계정 원칙 준수
- 기존 수기로 진행하던 회원가입 일원화
- **대회 기록 관리 자동화** 시스템 구축

Role

▸ 팀장

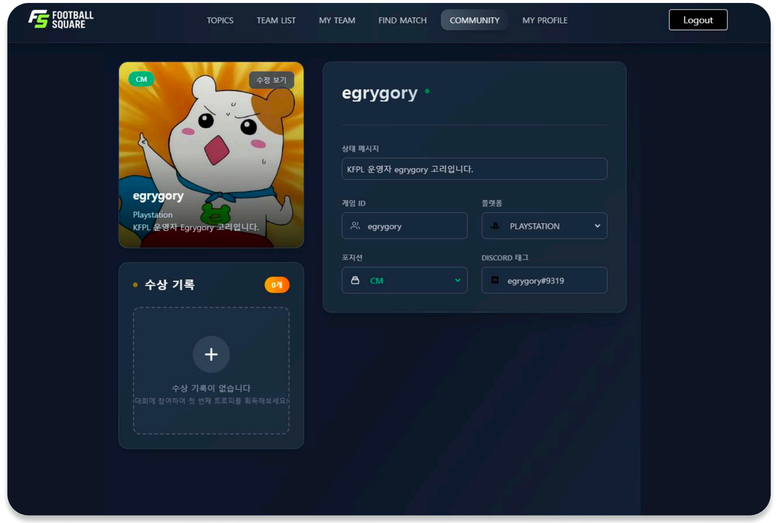
팀 게시판, 팀 채팅방 등의 기능 도입으로 팀원들간의 더 자유로운 공지 / 소통이 가능합니다.
팀 연혁, 수상 기록 등을 종합 저장하여 더 소속감 있는 팀 운영이 가능합니다.



- 각 **팀별 게시판** 시스템 구축
- 각 **팀별 실시간 채팅방** 시스템 구축
- 각 팀별 **연혁, 트로피 수상 시스템** 구축

▸ 일반 사용자

자신의 수상 기록, 경기 데이터들을 한눈에 종합 확인 가능합니다.
기존의 어려웠던 용병 경기 참여 등을 일괄 시스템으로 편리하게 경기에 참여 가능합니다.
개인 DM 시스템으로 사용자간 편리한 소통이 가능합니다.



- **수상 기록, 경기 기록, MMR** 등의 시스템 구축
- **용병 매치 참여 시스템** 구축
- 사용자간 **개인 DM 시스템** 구축

Database

PostgreSQL 17

기존 커뮤니티 운영에서, 추상적인 운영을 **매치, 커뮤니티, 팀, 대회** 등으로 추상화하고, 일관된 사용자 경험을 가질 수 있도록 서비스를 구현하였습니다.

POINT¹

파편화 되어있던 경기 진행 시스템을, **"매치" 개념**을 도입하여 일원화된 경험을 제공합니다.
동일한 시간 대의 매치에 참여할 수 없도록 하기 위해, **Postgres EXCLUDE** 제약을 활용하였습니다.

· DDL

```
EXCLUDE USING GIST (
    player_list_idx WITH =,
    match_time_range WITH &&
) WHERE (player_list_idx IS NOT NULL)
```

▸ 매치 구조

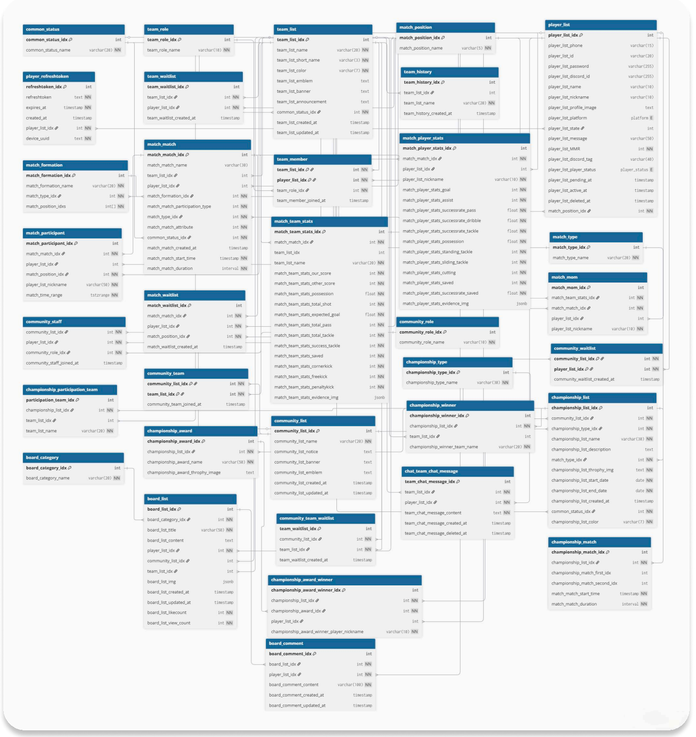
POINT²

커뮤니티에서 소속 팀을 일괄적으로 관리 가능합니다.
또한, 여러 커뮤니티가 서비스 이용을 원할 경우를 대비하여 **다양한 커뮤니티가 존재할 수 있도록 스키마를 설계**하였습니다.

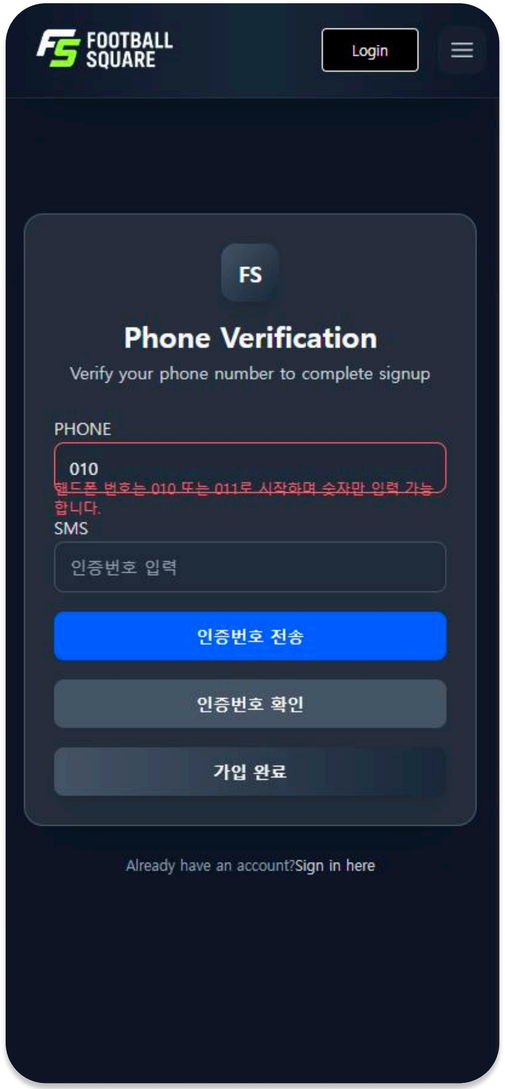
▸ 커뮤니티 구조

POINT³

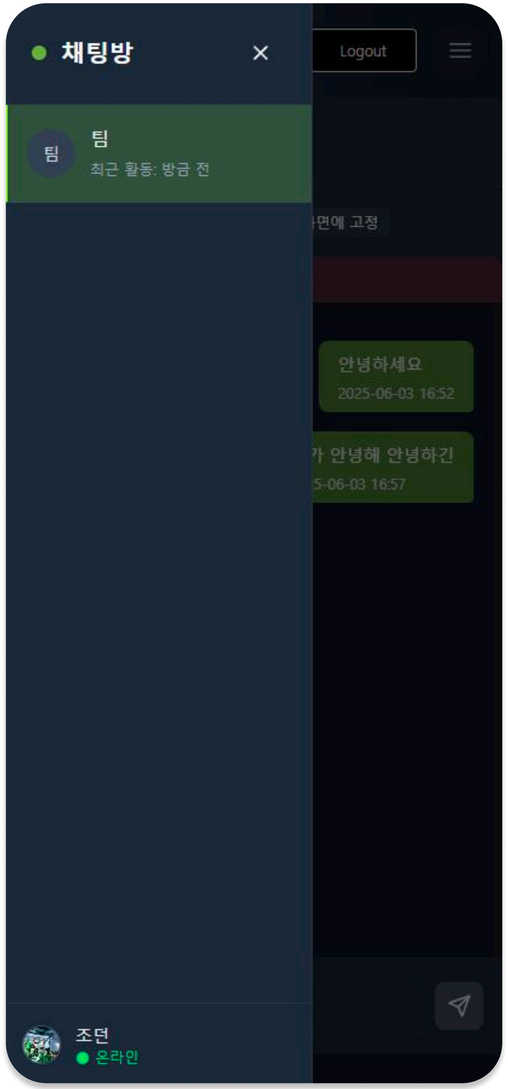
총 34개의 DB로 요구사항과 시스템을 구조화해, 데이터의 무결성과 제약조건에 맞는 스키마를 설계하였습니다.



▸ DB ERD



▸ 휴대전화 인증 페이지



▸ 실시간 팀 채팅 페이지

건강한 커뮤니티 생태계를 조성하기 위해 다음과 같은 두 가지 핵심 기능을 메모리 기반 데이터 저장소인 **Redis**를 활용하여 설계하였습니다.

Service

'1인 1계정' 원칙을 강제하여 무분별한 계정 생성을 막고 공정한 환경을 보장하기 위해 **휴대폰 인증 시스템**을 도입했습니다.

Goal

1인 1계정 원칙을 적용하여 중복 가입을 방지하고, 실사용자 기반의 신뢰도 높은 매칭·커뮤니티 환경을 제공합니다.

01 SMS 인증번호 발송

사용자가 휴대폰 번호를 입력하면, 서버는 Redis를 통해 해당 번호의 일일 전송 횟수를 확인하여 **무분별한 요청을 방지**합니다.
이후 6자리의 난수 인증번호를 생성하여 Aligo SMS API를 통해 사용자에게 발송하고, 해당 인증번호와 **인증 시도 횟수를 Redis에 유효시간**과 함께 저장합니다.

02 인증번호 검증

사용자가 수신한 인증번호를 입력하면, 서버는 Redis에 저장된 정보와 비교하여 일치 여부를 확인합니다. 인증번호가 만료되었거나, 정해진 시도 횟수를 초과했거나, 번호가 일치하지 않을 경우 에러를 반환하여 **비정상적인 접근을 차단**합니다.

03 중복 가입 방지 및 가입 완료

인증번호 검증이 성공하면, 해당 휴대폰 번호로 가입된 계정이 있는지 데이터베이스에서 최종 확인합니다. 중복된 계정이 없을 경우에만 사용자 정보를 데이터베이스에 등록하고 회원가입 절차를 완료시켜, **1인 1계정 원칙을 강제**합니다.

Service

같은 팀에 소속된 사용자들끼리 **실시간으로 메시지**를 주고받을 수 있는 팀 전용 채팅 서비스 입니다.

Goal

팀별로 격리된 실시간 소통 채널을 제공하여, 플랫폼 내에서 팀원 간의 원활한 협업과 신속한 정보 공유를 목표로 합니다.

01 연결 및 인증

클라이언트는 **socket.io**를 통해 채팅 서버에 연결을 시도하며, 이때 JWTtoken 을 함께 전송합니다. 서버는 미들웨어를 통해 토큰의 유효성을 검증하여 사용자를 인증하고, 해당 유저의 팀 정보를 획득합니다. 또한, **여러 서버 인스턴스 간의 상태공유와 확장**을 위해 Redis 어댑터를 사용합니다.

02 채팅방 참여 및 격리

인증이 완료된 사용자는 join 이벤트를 서버로 보냅니다. 서버는 사용자의 팀 ID를 기반으로 team_{팀ID} 형식의 고유한 채팅방(Room)을 생성하고, socket.join() 메소드를 호출하여 해당 사용자를 팀 전용 채팅방에 참여시킵니다. 이를 통해 다른 **팀의 채팅과 완전히 격리된 통신 환경**을 구축합니다.

03 메시지 처리 및 전파

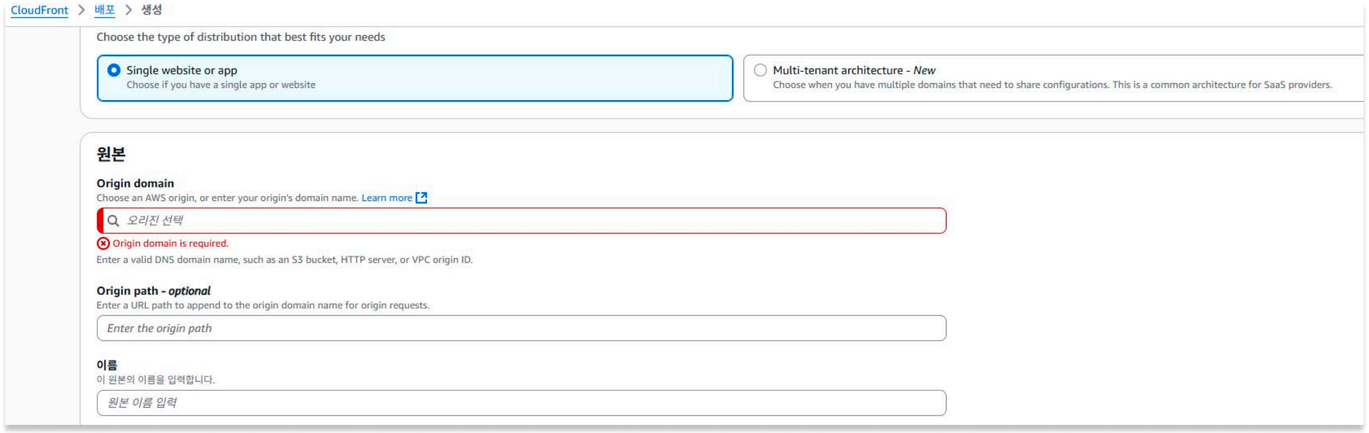
사용자가 message 이벤트를 보내면, 서버는 먼저 해당 메시지를 데이터베이스 테이블에 저장하여 대화 내역을 보존합니다. 이후 보낸 사람의 정보와 메시지가 담긴 payload를 생성하여, 자신을 포함한 해당 채팅방(Room)에 있는 모든 팀원에게 메시지를 **실시간으로 전파**(Broadcast)합니다.

Solution

▸ CDN

채팅 서버 등에서는, 하루에 적게는 10회, 많게는 수백회의 채팅 메시지가 발생하고 있었습니다.
프로필 이미지등의 **동일한 이미지가 중복해서 GET 요청**이 이루어지고 있었습니다.
S3 버킷을 오리진으로 삼는 CDN을 적용하여,
성능 향상 뿐 아니라, **동일 파일에 대한 반복 요청을 줄여 비용 절감**을 꾀했습니다.

결과적으로,
90% 이상의 응답속도 향상
95% 정도의 네트워크 요청 수 절감 효과를 기대할 수 있을 것으로 예상됩니다.



▸ AWS 콘솔 배포 생성

▸ CDN 적용 전 응답 속도 측정. 43ms

▸ CDN 적용 후 응답 속도 측정. 6ms

Solution

▸ 문제 상황

프로젝트 구조가 SPA 기반 프론트엔드와 Express 기반 백엔드 API로 분리되면서, 두 서비스가 서로 다른 포트와 프로세스에서 동작하게 되었습니다.
이에 따라 아래와 같은 문제가 발생하였습니다.

- 서로 다른 Origin으로 인한 **CORS 정책 충돌**
- REST API / WebSocket을 포함한 **교차 도메인 요청**의 불안정성
- **HTTPS 적용 부재**로 인한 보안 문제
- 프론트 / 백엔드 주소가 일관되지 않아 관리 복잡

▸ 해결

1. 도메인 구조 설계

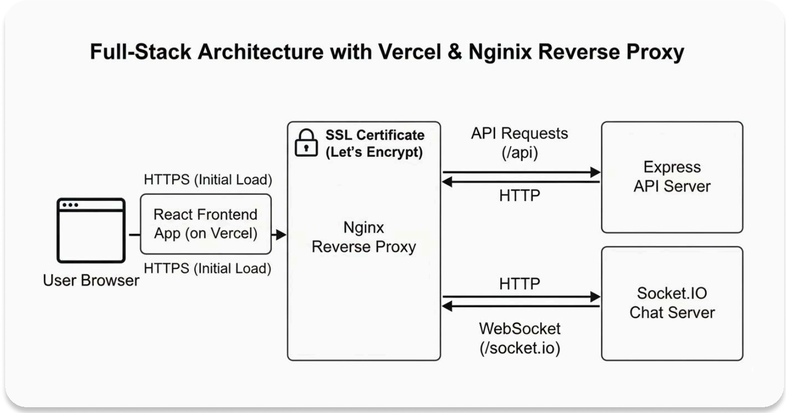
- 메인 서비스 : https//footballsquare.co.kr
- API 서버 : https//api.footballsquare.co.kr

2. Nginx 기반 리버스 프록시 구축

- 모든 요청은 Nginx가 먼저 받고, 내부 Express API 서버와 Socket I.O 채팅 서버로 라우팅. CORS 정책을 서버 게이트웨이 레이어에서 통합 관리하도록 변경

3. HTTPS 자동화 (Let's Encrypt + Cerbot)

- Cerbot의 nginx 플러그인을 이용해 SSL 인증서 자동 발급



▸ 변경된 아키텍처 구조도

```
renew_before_expiry = 30 days
version = 1.21.0
archive_dir = /etc/letsencrypt/archive/api.footballsquare.co.kr
cert = /etc/letsencrypt/live/api.footballsquare.co.kr/cert.pem
privkey = /etc/letsencrypt/live/api.footballsquare.co.kr/privkey.pem
chain = /etc/letsencrypt/live/api.footballsquare.co.kr/chain.pem
fullchain = /etc/letsencrypt/live/api.footballsquare.co.kr/fullchain.pem

# Options used in the renewal process
[renewalparams]
account = 6c415fe68369045352151020e6931346
authenticator = nginx
installer = nginx
server = https://acme-v02.api.letsencrypt.org/directory
```

▸ Let's Encrypt의 Cerbot을 활용한 SSL 인증서 갱신

```
erver {
    listen 443 ssl;
    server_name api.footballsquare.co.kr footballsquare.co.kr;

    ssl_certificate /etc/letsencrypt/live/api.footballsquare.co.kr/fullchain.pem;
    ssl_certificate_key /etc/letsencrypt/live/api.footballsquare.co.kr/privkey.pem;
    include /etc/letsencrypt/options-ssl-nginx.conf;
    ssl_dhparam /etc/letsencrypt/ssl-dhparams.pem;

    # ✅채팅 서버로 WebSocket 전용 프록시
    location /socket.io/ {
        proxy_pass http://localhost:3001/socket.io/;

        proxy_http_version 1.1;
        proxy_set_header Upgrade $http_upgrade;
        proxy_set_header Connection upgrade;
        proxy_set_header Host $host;
        proxy_cache_bypass $http_upgrade;
        add_header Access-Control-Allow-Origin "https://footballsquare.co.kr" always;
        add_header Access-Control-Allow-Credentials true always;
        add_header Access-Control-Allow-Methods "GET, POST, OPTIONS, PUT, DELETE" always;
        add_header Access-Control-Allow-Headers "Authorization, Content-Type" always;

        if ($request_method = OPTIONS) {
            return 204;
        }
    }
```

▸ WebSocket 요청 내부 채팅서버로 프록시, CORS 접근 정책 설정

```
    # ✅나머지 API 요청은 기존 Express 서버로 유지
    location / {
        limit_req zone=req_limit_per_ip burst=20 nodelay;

        add_header Access-Control-Allow-Origin "https://footballsquare.co.kr" always;
        add_header Access-Control-Allow-Credentials true always;
        add_header Access-Control-Allow-Methods "GET, POST, OPTIONS, PUT, DELETE" always;
        add_header Access-Control-Allow-Headers "Authorization, Content-Type" always;

        if ($request_method = OPTIONS) {
            return 204;
        }

        proxy_pass http://localhost:8000;

        proxy_http_version 1.1;
        proxy_set_header Upgrade $http_upgrade;
        proxy_set_header Connection 'upgrade';
        proxy_set_header Host $host;
        proxy_cache_bypass $http_upgrade;
    }
}

server {
    listen 80;
    server_name api.footballsquare.co.kr footballsquare.co.kr;
    return 301 https://$host$request_uri;
}
```

▸ 나머지 요청은 내부 Express API 서버로 전달, CORS 정책 적용

Thank you

새로운 것을 개발하고,
코드 작성하는걸 사랑하는 개발자, 김연호

roger345@naver.com

010 • 3324 • 8711