포팅 매뉴얼

마감일	@2025년 4월 9일

▼ 목차

1. Directory Structure

- docker compose 를 사용하는 웹 / 앱 프로젝트로 `backend/` / `frontend/` / `nginx/` 3개의 폴더로 구분하여 CI-CD 시 docker compose up 을 통해 서비스 배포
- `backend/` 는 다시 `fastapi/`와 `mariadb/` 로 구분
- `frontend/` 는 `react-native/` 로 구성

```
📦 S12P21B206
├─📜 README.md
├─📂 _docs
│  └─📜 Porting_Manual.md
├─📂 backend
```

```
┃  ┣📂 fastapi
┃  ┃  ┣📜 Dockerfile
┃  ┃  ┣📂 ai
┃  ┃  ┃  ┣📂 news
┃  ┃  ┃  ┃  ┗📜 news_test.ipynb
┃  ┃  ┃  ┗📂 ocr
┃  ┃  ┃    ┣📂 image
┃  ┃  ┃    ┣📂 legacy
┃  ┃  ┃    ┣📂 ocr_json
┃  ┃  ┃    ┣📂 ocr_text
┃  ┃  ┃    ┣📜 tesseract_ocr.py
┃  ┃  ┃    ┣📜 ticket_certificate.py
┃  ┃  ┃    ┗📜 requirements.txt
┃  ┃  ┣📂 app
┃  ┃  ┃  ┣📂 DB
┃  ┃  ┃  ┃  ┣📂 init_setting
┃  ┃  ┃  ┃  ┗📜 import_game_schedule.py
┃  ┃  ┃  ┣📂 baseball_data
┃  ┃  ┃  ┣📂 router
┃  ┃  ┃  ┣📂 utils
┃  ┃  ┃  ┣📜 main.py
┃  ┃  ┃  ┣📜 models.py
┃  ┃  ┃  ┗📜 test.py
┃  ┃  ┗📜 requirements.txt
┃  ┗📂 mariadb
┃    ┗📂 data
┣📂 frontend
┃  ┗📂 react-native
┃    ┣📜 Dockerfile
┃    ┣📂 app
┃    ┃  ┣📜 App.tsx
┃    ┃  ┣📂 assets
┃    ┃  ┣📂 components
┃    ┃  ┣📂 src
┃    ┃  ┣📜 package.json
┃    ┃  ┗📜 tsconfig.json
┣📂 nginx
┃  ┣📜 Dockerfile
┃  ┗📜 nginx.conf
┗📜 docker-compose.yml
```

2. Tech Stack Version

01. EC2

001. Port & System Configuration

Skill	Port 번호
Nginx	80 / 443
React Native Expo	8081 / 19000~19002
Jenkins	8080
SSH	22
FastAPI	8000
MariaDB	3306

항목	내용
OS	Ubuntu 22.04.4 LTS (Jammy Jellyfish)
Kernel	6.8.0-1021-aws
Architecture	x86_64
CPU	Intel(R) Xeon(R) CPU E5-2686 v4 @ 2.30GHz (4 cores, 1 thread/core)
Memory	총 15GiB (사용 2.7GiB / 사용 가능 12GiB)
Docker Engine	28.0.1
Docker Client	28.0.1
Docker Compose	v2.33.1
Docker Containers	5개 실행 중 (nginx, fastapi, reactnative, mariadb, jenkins)

002. Primary Packages

Fastapi

- **APScheduler**: `3.11.0` → 백그라운드에서 정기 작업을 예약 실행할 수 있는 스케줄러
- **bcrypt**: `4.3.0` → 비밀번호 해싱용 암호화 라이브러리
- **beautifulsoup4**: `4.13.3` → HTML/XML 파싱 라이브러리 (크롤링에 유용)
- **cryptography**: `44.0.2` → 대칭/비대칭 암호화, 인증 등 보안 관련 기능 제공
- **dotenv**: `0.9.9` → `.env` 환경 변수 파일 로딩용 라이브러리 (단순 dotenv)
- **email_validator**: `2.2.0` → 이메일 주소 유효성 검사용
- **fastapi**: `0.115.11` → 비동기 지원, 타입 기반의 고성능 Python API 서버 프레임워크
- **numpy**: `2.2.4` → 수치 연산과 배열 계산을 위한 과학 연산 핵심 라이브러리
- **opencv-python**: `4.11.0.86` → 컴퓨터 비전 및 이미지/영상 처리 라이브러리
- **pandas**: `2.2.3` → 데이터프레임 기반의 데이터 분석 및 처리 라이브러리
- **passlib**: `1.7.4` → 다양한 해시 알고리즘을 지원하는 비밀번호 해싱 라이브러리
- **pillow**: `11.1.0` → 이미지 파일 열기/편집/저장 등 이미지 처리 라이브러리
- **PyMySQL**: `1.1.1` → MySQL 서버와 연동 가능한 순수 Python 클라이언트
- **python-dotenv**: `1.0.1` → `.env` 환경 변수 파일 로딩용 (기능 강화된 dotenv)
- **python-jose**: `3.4.0` → JWT 등 JSON Web Token 기반 인증/암호화 구현용
- **requests**: `2.32.3` → HTTP 요청을 쉽게 보낼 수 있는 표준 라이브러리
- **selenium**: `4.30.0` → 브라우저 자동화를 위한 웹 테스트/크롤링 프레임워크
- **SQLAlchemy**: `2.0.38` → Python ORM 및 데이터베이스 연동 도구
- **starlette**: `0.46.1` → FastAPI의 기반이 되는 비동기 웹 프레임워크
- **trio**: `0.29.0` → 구조화된 concurrency를 제공하는 비동기 라이브러리
- **uvicorn**: `0.34.0` → FastAPI 등의 ASGI 서버 구동을 위한 경량 서버
- **webdriver-manager**: `4.0.2` → Selenium용 크롬/파이어폭스 드라이버 자동 설치 도구

React Native

- **expo**: `~52.0.39` → React Native 개발을 위한 프레임워크, 다양한 기본 API 제공
- **react-native**: `0.76.7` → React Native 핵심 라이브러리
- **react**: `18.3.1` → 컴포넌트 기반 UI 구성의 중심이 되는 라이브러리
- **@react-navigation/native**: `^6.1.9` → 화면 간 네비게이션 처리 (라우팅)
- **@react-navigation/stack**: `^6.3.20` → Stack 기반 화면 전환
- **@react-navigation/bottom-tabs**: `^6.5.11` → 하단 탭 네비게이션 UI
- **axios**: `^1.8.4` → HTTP 요청을 위한 클라이언트 라이브러리
- **zustand**: `^5.0.3` → 간단한 전역 상태 관리 라이브러리
- **react-native-reanimated**: `^3.17.2` → 고성능 애니메이션 구현
- **react-native-gesture-handler**: `~2.20.2` → 제스처 인식 및 처리
- **react-native-vector-icons**: `^10.2.0` → 다양한 아이콘 사용 가능
- **styled-components**: `^5.3.11` → 컴포넌트 기반 스타일링 라이브러리
- **expo-image-picker**: `~16.0.6` → 이미지 촬영 및 앨범에서 선택
- **expo-calendar**: `~14.0.6` → 디바이스 캘린더 접근 및 일정 관리
- **react-native-svg**: `^15.11.2` → SVG 렌더링 지원
- **victory-native**: `^41.16.1` → 데이터 시각화를 위한 그래프/차트

- **react-native-chart-kit:** `^6.12.0` → 다양한 형태의 차트 제공
- **react-native-toast-message:** `^2.2.1` → 커스터마이징 가능한 토스트 메시지
- **react-native-snap-carousel:** `^3.9.1` → 스와이프 가능한 캐러셀 UI
- **react-native-root-toast:** `^4.0.1` → 간단한 토스트 메시지 UI
- **@react-native-async-storage/async-storage:** `^2.1.2` → 로컬 스토리지 (key-value 저장)
- **expo-haptics:** `~14.0.1` → 햅틱 피드백 (진동) 효과 추가용

02. GCP (GPU Server)

001. System Configuration

항목	내용
OS	Ubuntu 22.04.5 LTS (Jammy Jellyfish)
Kernel	(해당 없음 또는 기본 커널)
Architecture	x86_64
CPU	Intel(R) Xeon(R) CPU @ 2.00GHz (4 cores)
Memory	총 14GiB
GPU	NVIDIA Tesla T4 (15GB VRAM)
CUDA	12.2
NVIDIA Driver	535.230.02
Docker Engine	28.0.1
Docker Client	28.0.1
Docker Compose	v2.33.1
Docker Containers	1개 실행 중 (fastapi)

002. Primary Packages

- **fastapi** → 비동기 기반의 고성능 Python 웹 프레임워크
- **uvicorn** → FastAPI용 ASGI 서버
- **selenium==4.29.0** → 브라우저 자동화 및 크롤링 도구
- **webdriver-manager==4.0.2** → 크롬/파이어폭스 드라이버 자동 설치 및 관리
- **pandas==2.2.3** → 데이터프레임 기반 데이터 처리 및 분석 라이브러리
- **schedule** → 간단한 작업 스케줄링 라이브러리 (예: 특정 시간마다 실행)
- **datetime** → 표준 파이썬 내장 날짜/시간 처리 모듈
- **google-genai==1.8.0** → Google Cloud의 `Gemini API` 클라이언트 라이브러리
- **google-generativeai==0.8.4** → Generative AI (Gemini) 사용을 위한 또 다른 클라이언트
- **python-dotenv==1.0.1** → `.env` 환경 변수 로딩용
- **requests==2.32.3** → HTTP 요청을 위한 클라이언트 라이브러리
- **torch==2.6.0** → PyTorch 딥러닝 프레임워크
- **transformers==4.50.0** → HuggingFace의 사전학습된 NLP 모델 라이브러리
- **accelerate==1.5.2** → HuggingFace 모델 학습을 위한 분산 학습/최적화 유틸
- **autoawq==0.2.8** → LLM 추론 최적화 도구 (AWQ: Activation-aware Weight Quantization)

003. Models

- **Summary**: `LGAI/EXAONE-3.5-2.4B-Instruct-Q6_K_L`

https://huggingface.co/LGAI-EXAONE

3. Dockerfile & docker-compose.yml

00. Web docker-compose.yml

```
services:
  fastapi:
    build:
      context: ./backend/fastapi
      dockerfile: Dockerfile
      args:
        BACKEND_PORT: ${BACKEND_PORT}
    env_file:
      - ./backend/fastapi/app/.env
    ports:
      - "${BACKEND_PORT}:${BACKEND_PORT}"
    depends_on:
      - mariadb

  mariadb:
    image: mariadb:latest
    environment:
      MARIADB_ROOT_PASSWORD: ${MARIA_PASSWORD}
      MARIADB_DATABASE: finball
    volumes:
      - ./backend/mariadb/data:/var/lib/mysql
    ports:
      - "${MARIA_PORT}:3306"

  reactnative:
    build:
      context: ./frontend/react-native
      dockerfile: Dockerfile
      args:
        FRONTEND_METRO_PORT: ${FRONTEND_METRO_PORT}
        FRONTEND_EXPO_PORT: ${FRONTEND_EXPO_PORT}
    env_file:
      - ./frontend/react-native/app/.env
    ports:
      - "${FRONTEND_METRO_PORT}:${FRONTEND_METRO_PORT}"
      - "${FRONTEND_EXPO_PORT}:${FRONTEND_EXPO_PORT}"
    entrypoint: ["/bin/sh", "-c", "npx expo start --web"]

  nginx:
    build:
      context: ./nginx
      dockerfile: Dockerfile
    depends_on:
      - fastapi
```

```
    ports:
      - "${NGINX_PORT}:${NGINX_PORT}"
```

01. Fastapi

```
# backend/fastapi/Dockerfile
FROM python:3.13

ARG BACKEND_PORT
ENV BACKEND_PORT=${BACKEND_PORT}

WORKDIR /app
COPY requirements.txt .
RUN apt-get update && apt-get install -y netcat-openbsd && apt-get install -y tzdata && rm -rf /var/lib/apt/lists/* && \
    pip install --upgrade pip && \
    pip install -r requirements.txt

ENV TZ=Asia/Seoul

# 애플리케이션 소스 복사 (bind mount을 사용할 경우 최신 소스는 호스트와 동기화)
COPY app/ ./
RUN cat ./wait-for-it.sh | tr -d '\r' > ./wait-for-it-fixed.sh && \
    mv ./wait-for-it-fixed.sh ./wait-for-it.sh && \
    chmod +x ./wait-for-it.sh
RUN chmod +x ./wait-for-it.sh

RUN apt-get update && apt-get install -y wget gnupg2
RUN wget -q -O - https://dl.google.com/linux/linux_signing_key.pub | apt-key add -
RUN echo "deb [arch=amd64] http://dl.google.com/linux/chrome/deb/ stable main" > /etc/apt/sources.list.d/google-chro
RUN apt-get update && apt-get install -y google-chrome-stable
RUN apt-get update && apt-get install -y libzbar0

EXPOSE ${BACKEND_PORT}
CMD ["bash", "-c", "pwd && ./wait-for-it.sh mariadb:3306 -t 10 -- uvicorn main:app --host 0.0.0.0 --port ${BACKEND_PO
```

02. React-native

```
# frontend/react-native/Dockerfile
FROM node:18-alpine

ARG FRONTEND_METRO_PORT
ARG FRONTEND_EXPO_PORT
ENV FRONTEND_METRO_PORT=${FRONTEND_METRO_PORT}
ENV FRONTEND_EXPO_PORT=${FRONTEND_EXPO_PORT}

WORKDIR /app

# package.json과 package-lock.json을 먼저 복사하여 의존성 설치
COPY ./app/package.json ./app/package-lock.json ./
RUN npm install

# 전체 소스 복사
COPY ./app .

EXPOSE ${FRONTEND_METRO_PORT} ${FRONTEND_EXPO_PORT}
```

03. nginx

```
FROM nginx:latest

# 커스텀 설정 복사
COPY nginx.conf /etc/nginx/conf.d/custom.conf

# Nginx 포그라운드 실행
CMD ["nginx", "-g", "daemon off;"]
```

04. GCP docker-compose.yml

```
services:
  fastapi:
    build:
      context: ./backend/fastapi
      dockerfile: Dockerfile
    args:
      BACKEND_PORT: 8000
  env_file:
    - ./backend/fastapi/app/.env
  volumes:
    - ./backend/fastapi/app:/app
  ports:
    - "8000:8000"
  deploy:
    resources:
      reservations:
        devices:
          - capabilities: [gpu]
  runtime: nvidia
```

- GCP 의 Fastapi Dockerfile 은 EC2 와 동일

4. How to Run

01. Web (야금야금)

1. EC2 에 Jenkins 설치
2. Job 생성 및 Gitlab 에 올라와 있는 repo CI-CD 설정
3. Jenkins 크레덴셜 통해 .env 파일 각각의 경로에 삽입

```
rm -f ./.env
cp $PROJECT_ENV ./.env

rm -f ./backend/fastapi/app/.env
cp $BACKEND_ENV ./backend/fastapi/app/.env

rm -f ./frontend/react-native/app/.env
cp $FRONTEND_ENV ./frontend/react-native/app/.env


# 배포 실행
docker compose down
docker compose up -d --build
```

- env 관련은 아래 기입해 두었습니다.

02. GCP (34.94.94.197)

1. Terraform base file 설정해 서버 세팅

```
### main.tf
# Compute Engine 인스턴스를 생성 > Debian 12 이미지 사용 / 머신 유형은 e2-medium > n1-standard-4
# e2-medium : GPU 없는 일반 인스턴스 / vCPU 2개, 메모리 4GB

# Nginx 웹서버가 자동으로 설치 및 실행



# GCP 프로바이더 설정 > variables 에서 끌고옴. credentials 는 환경변수로 미리 등록해 놓았음.
provider "google" {
#  credentials = file(var.credentials_file)
  project     = var.project_id
  region      = var.region
  zone        = var.zone
}

# VM 인스턴스 생성
resource "google_compute_instance" "vm_instance" {
  name         = "terraform-instance"
  machine_type = "n1-standard-4"
  zone         = var.zone

  boot_disk {
    initialize_params {
      image = "ubuntu-os-cloud/ubuntu-2204-lts"
    }
  }

  network_interface {
    network = "default"
    access_config {} # Public IP를 자동 할당
  }

  guest_accelerator {  # GPU 명
    type  = "nvidia-tesla-t4"
    count = 1
  }

  scheduling {
    on_host_maintenance = "TERMINATE"  # GPU는 반드시 이거여야 함 << ?
    automatic_restart   = false
  }

  metadata = {
    ssh-keys = "chano:${file("~/.ssh/my-gcp-key.pub")}"
  }

  # 들어가서 NVIDIA GPU 사용 가능하게 + Docker 설치
  metadata_startup_script = file("startup.sh")

}

### variables.tf
# 원하는 리전 및 존으로 수정 가능합니다.

# variable "credentials_file" {
```

```
#   description = "GCP 서비스 계정 키 파일 경로"
#   type        = string
# }

variable "project_id" {
  default     = "pure-wall-454723-t7"
  description = "GCP 프로젝트 ID"
  type        = string
}

variable "region" {
  default     = "me-west1"
  description = "GCP 리전"
  type        = string
}

variable "zone" {
  default     = "me-west1-b"
  description = "GCP 존"
  type        = string
}
```

2. GCP 서버 파일 존재하는 폴더로 이동 후 컴포즈 빌드 빛 실행 docker compose up —build
3. 스케쥴러를 통해 EC2의 Fastapi 와 API 통신 진행

환경변수 설명

🖥️ 백엔드 환경 변수 (`./backend/fastapi/app/.env`)

위치	환경 변수	설정 값	설명
`./backend/fastapi/app/.env`	`DATABASE_TYPE`	`mysql+pymysql`	SQLAlchemy용 드라이버 지정
〃	`DATABASE_USER`	`root`	DB 사용자
〃	`DATABASE_PASSWORD`	`B206_fintech`	DB 비밀번호
〃	`DATABASE_IP`	`mariadb`	도커 내부 DB 주소
〃	`DATABASE_PORT`	`3306`	DB 포트
〃	`DATABASE_DB`	`finball`	사용할 DB 이름
〃	`SECRET_KEY`	`fske256d3433kf2@...`	JWT 비밀 키
〃	`ALGORITHM`	`HS256`	JWT 알고리즘
〃	`ACCESS_TOKEN_EXPIRE_MINUTES`	`60`	액세스 토큰 유효시간 (분)
〃	`SSAFY_API_BASE_URL`	`https://finopenapi.ssafy.io/ssafy/api/v1`	SSAFY 외부 API URL
〃	`SSAFY_API_KEY`	`5a4dda08418e4fa1885c08924abd5131`	SSAFY API 키
〃	`SAVING_CODE`	`999-1-f8dceb52266249`	적금 코드
〃	`DEPOSIT_CODE`	`999-1-0146508f197d4c`	예금 코드
〃	`CORS_ORIGINS`	`http://localhost:8081,<http://3.38.183.156>,<http://3.38.183.156:8081>,<http://j12b206.p.ssafy.io>,<http://j12b206.p.ssafy.io:19000>,<https://j12b206.p.ssafy.io` >	CORS 허용 도메인 목록 (쉼표로 구분)

🐋 Docker 관련 환경 변수

환경 변수	설정 값	설명
`BACKEND_PORT`	`8000`	FastAPI 백엔드 컨테이너에서 사용하는 포트
`FRONTEND_METRO_PORT`	`8081`	React Native Metro 번들러 포트
`FRONTEND_EXPO_PORT`	`8081`	React Native Expo 개발 서버 포트
`NGINX_PORT`	`80`	Nginx가 바깥으로 노출하는 HTTP 포트
`MARIA_PORT`	`3308`	호스트 기준 MariaDB 포트
`MARIA_PASSWORD`	`B206_fintech`	MariaDB `root` 사용자 비밀번호

03. Jenkins를 활용한 자동 CI/CD

🚢 Jenkins 컨테이너 생성 및 실행

```
# Jenkins 컨테이너를 EC2의 Docker 환경을 공유하는 방식으로 실행
cd /home/ubuntu && mkdir jenkins-data

docker run -d \
  -p 8080:8080 \
  -p 50000:50000 \
  -v /home/ubuntu/jenkins-data:/var/jenkins_home \
  -v /var/run/docker.sock:/var/run/docker.sock \
  -v /usr/bin/docker:/usr/bin/docker \
  -v /usr/libexec/docker/cli-plugins:/usr/libexec/docker/cli-plugins \
  --name jenkins \
  jenkins/jenkins:2.501

# 인증서 등록 및 업데이트 경로 변경
cd /home/ubuntu/jenkins-data
mkdir update-center-rootCAs
wget https://cdn.jsdelivr.net/gh/lework/jenkins-update-center/rootCA/update-center.crt -O ./update-center-rootCAs/upda

sudo sed -i 's#https://updates.jenkins.io/update-center.json#https://raw.githubusercontent.com/lework/jenkins-update-c

docker restart jenkins
```

🔧 Jenkins 초기 설정

```
# EC2 Public DNS 확인
curl http://169.254.169.254/latest/meta-data/public-hostname

# Jenkins 초기 비밀번호 확인
docker exec -it jenkins cat /var/jenkins_home/secrets/initialAdminPassword
```

- 초기 비밀번호로 Jenkins 로그인 후 필요한 플러그인 설치
  오류 발생 시 `/jenkins-data/plugins` 폴더 삭제 후 수동 설치 가능

```
**기본 로그인 정보**
- ID: admin
```

```
  - PW: B206_fintech
```

🔗 GitLab Webhook 설정

- Webhook URL: http://3.38.183.156:8080/project/deploy-develop

- 트리거 체크:
  - ✅ Push events
  - ✅ Merge request events

> Webhook 설정 후 “Test” 버튼으로 연결 확인

> GitLab Access Token 대신 Jenkins Job의 Secret Token 사용 권장

🔌 필수 Jenkins 플러그인

- GitLab Plugin → GitLab Webhook 요청 처리 및 인증
- Matrix Authorization Strategy → 익명 사용자에게도 Job 빌드 권한 허용
- 설치 후 : `Jenkins 관리 > 플러그인 > 설치됨` 목록에서 확인, 없으면 `사용 가능` 탭에서 설치

⚙️ Jenkins Job 생성 및 설정

1. Job 이름: `deploy-develop` (Freestyle Project)
2. 프로젝트 설정 :
   a. GitLab URL : `https://lab.ssafy.com/s12-fintech-finance-sub1/S12P21B206/`
   b. Git 리포지토리 URL: `https://lab.ssafy.com/s12-fintech-finance-sub1/S12P21B206.git`
      - Credentials:
        - Kind: `Username With Password`
        - Username: `정찬호`
        - Password: `GitLab Access Token`
   c. Branch to Build: `develop`
   d. Build Triggers : `Build when a change is pushed to GitLab`
      → GitLab Webhook URL: `http://3.38.183.156:8080/project/deploy-develop`
   e. Secrets 파일 등록:
      - Variable: `PROJECT_ENV`, `BACKEND_ENV`, `FRONTEND_ENV`
      - Credentials Type: `Secret file`
   f. 빌드 스텝 (Shell Script) :

```
# .env 파일 복사 및 배포 실행
rm -f ./.env
cp $PROJECT_ENV ./.env

rm -f ./backend/fastapi/app/.env
cp $BACKEND_ENV ./backend/fastapi/app/.env

rm -f ./frontend/react-native/.env
cp $FRONTEND_ENV ./frontend/react-native/.env

docker compose down
docker compose up -d --build
```

5. ⚾ 야금야금 시연 시나리오

로그인 및 계좌 조회 플로우

- 웹 접속 후, '시작하기' 버튼 클릭 시 '로그인 페이지'로 이동합니다.
- 아이디와 비밀번호(ID 예: testuser_250328@gmail.com, PW: aaaa1111)를 입력하여 로그인합니다.
- 로그인 성공 시 계좌 조회 화면으로 이동하며, 입출금 계좌와 잔액이 정상적으로 조회됩니다.
- 계좌 조회 화면에서 야금야금 가입하기 버튼 (위, 아래 2개) 를 클릭 시 금융상품 소개 페이지로 이동됩니다.

  입출금 통장 (자유예금) 의 이체, 거래내역 버튼을 통해 이체 기능 및 거래내역 조회를 할 수 있습니다.

적금 가입 플로우

- 금융상품 소개 페이지에서 '뒤로가기'를 누르면 계좌 조회 화면으로 복귀합니다.

  가입하기 버튼을 클릭 시 팀 선택 모달로 진입합니다.
  응원팀을 선택하고 ‘선택하기’ 버튼을 눌러 최애 선수 선택 페이지로 이동합니다.
- 최애 선수 선택 페이지에서, 최애 선수 (투수 or 타자) 를 고르고, 선택하기를 눌러 적금 목표 선택 페이지로 이동합니다. 최애 선수 선택 페이지에서는 선수 등번호, 이름, 사진이 표시되며, 검색 기능으로 선수 검색이 가능합니다.
- 적금 목표 선택 페이지에서, 원하는 목표를 고르고, ‘이 목표로 적금 시작하기’ 버튼을 눌러 적금 규칙 설정 페이지로 이동합니다.
- 적금 규칙 설정 페이지에서, 일일 한도와 기본 / 최애 선수 / 상대팀 적금 규칙의 금액 및 ON/OFF 를 설정하고, 선택 버튼을 눌러 약관 조회 페이지로 이동합니다.
- 약관 조회 페이지에서, ‘필수’ 에 해당하는 약관에 모두 동의시 본인의 입출금 계좌를 출금 계좌로 선택할 수 있는 모달이 등장합니다. 계좌를 선택하고, ‘선택’ 시 야금야금 적금 가입이 완료됩니다. 선수 선택 후 적금 약관 페이지로 이동하며, 필수 약관 2개를 체크하면 출금계좌 드롭다운이 나타납니다.
- 야금야금 적금 가입 완료 페이지에서, 본인이 선택한 응원팀 / 최애 선수를 볼 수 있고, 순위 예측 바로가기 버튼을 눌러 우대금리를 주는 순위 예측 페이지로 이동할 수 있습니다.

  완료 버튼을 누를 시 적금 메인 페이지로 이동합니다.

메인 페이지 기능 확인

- 상단에 팀명과 팀컬러 UI가 정상적으로 보이는지 확인합니다.
- 목표 금액 및 현재 금액 bar, 금리 정보, 팀 순위 등의 정보가 정확히 출력되는지 확인합니다.
- 오늘의 적금 비교 모달은 어제 상대팀을 응원하는 적금 가입자들과의 비교를 AI 를 통해 요약하여 보여줍니다.
- 적금 규칙 모달은 유저가 설정한 기본 / 최애 선수 / 상대팀 규칙, 금액을 볼 수 있습니다.
- 최근 적금 내역 모달에서는 최근 3일 경기날짜 / 상대팀 / 경기결과 / 납입액을 볼 수 있습니다.
- 다음 경기 일정 모달에서는 앞으로의 6경기에 대한 경기날짜 / 상대팀 / 홈OR원정 정보를 볼 수 있습니다.
- 적금 메인 페이지 하단 탭 클릭 시 각 탭에 맞는 헤더 및 페이지로 이동되는지 확인합니다.
- 전체 내역, 전체 일정 클릭 시 각각 적금 상세 내역, 적금 내역 페이지로 이동합니다.

적금 내역 페이지

- 하단 바에서 적금 내역 클릭 시 해당 페이지로 이동됩니다.
- 경기 일정이 캘린더에 표시되고, 슬라이드로 월 전환이 가능한지 확인합니다.
- 캘린더의 날짜 클릭 시 아래 경기 상세 정보에 해당 경기에 대한 내용이 표시됩니다.
- 아래 3연전 일정을 통해 최근 3연전 3번에 대한 정보 (상대 / 일자 / 승패 / 납입액 등) 에 대한 정보 확인 가능합니다.
- 3연전 일정이 표시되는지 확인합니다. 현재는 3연전 첫 날에는 표시되지 않는 이슈가 존재합니다.
- 우측 상단의 사이드바 메뉴 아이콘을 눌러 상세 적금 내역 페이지로 이동할 수 있습니다.
- 상세 적금 내역 페이지에서는, 적금 납입 각각에 대한 금액 / 상대 / 적립일 / 송금 메세지 (한줄) 을 볼 수 있습니다. 각각의 탭을 클릭하면 추가적인 송금 메세지와 해당 금액에 들어간 요소 및 금액들을 볼 수 있습니다.

리포트 페이지

- 주간 팀, 개인 요약 정보, 적금 현황 그래프, 주간 뉴스 하이라이트가 정상적으로 출력됩니다.
  일자가 3.31인 이유는 3.31 ~ 4.6까지의 데이터를 통해 만들어진 레포트이기 때문입니다.

혜택 페이지

- 경기 직관 인증 / 팀 순위 맞추기 탭을 확인 가능합니다.
- 경기 직관 인증 탭으로 진입 시 티켓 사진 업로드 및 티켓 인증이 가능합니다. 해당 티켓이 사용되지 않았다면, 인증을 통해 우대금리 0.1%를 받을 수 있습니다. 사용되었다면, 경고 메세지와 함께 금리는 오르지 않습니다.
- 팀 순위 맞추기 탭으로 진입 시 응원 팀의 예상 순위를 선택할 수 있습니다. 예상 순위 선택 후 예측 순위를 제출하면, 해당 순위로 고정되고 정규시즌 종료 후 결과에 따라 우대금리를 줍니다.