기부챌린지 ‘기린’ 포팅 메뉴얼

A708 얼룩이들

목차

I. 프로젝트 개요

1. 프로젝트 소개

스타들의 챌린지를 보신 적이 있으신가요?

최근, 많은 스타들이 홍보 목적으로 챌린지를 진행하고, 팬들도 자신들이 응원하는 스타들의 챌린지에 많이 참여하고 있습니다.

또한 사랑하는 스타가 기부를 했을 때, 뜻을 함께 하고자 팬들도 함께 기부하는 경향도 늘고 있습니다.

'기린'은 '챌린지' 참여와 '기부'를 동시에 할 수 있는 플랫폼입니다.

'기린'과 함께 스타와 함께하는 선한 영향력을 퍼뜨려보세요!

2. 사용 툴

- 이슈 관리 툴: Jira
  - 매주 월요일 목표량을 설정하여 Sprint 진행
- 형상 관리 툴: Gitlab
  - Git-Flow 전략
- 커뮤니케이션 툴: Notion, Mattermost, Webex
  - Notion
    : 회의 및 스크럼 기록, 프로젝트 계획서, api 명세서 관리 등의 문서 관리
  - Mattermost
    : Gitlab, Jira Bot, Server 연동
- 디자인 툴: Figma
- UCC 제작 툴: Movavi

3. 개발 환경

- **Frontend - React**

React : ^18.2.0
Node : 16.17.0
Solc : ^0.8.17
Web3 : ^1.8.0
IDE : Visual Studio Code IDE 1.69.0

- **Backend – Spring boot**

Springboot : 2.7.3
Gradle : 7.5
JDK : 17-ea
DB : 8.0.30-MySQL
IDE : IntelliJ IDEA : 2021.3.1
IntelliJ Runtime : 11.0.13+7-b1751.21 amd64
ffmpeg: 4.4.1

- **Blockchain**

go1.19
Geth 1.9.24-stable
Vagrant 2.3.0
truffle v5.5.29

- **CI/CD**

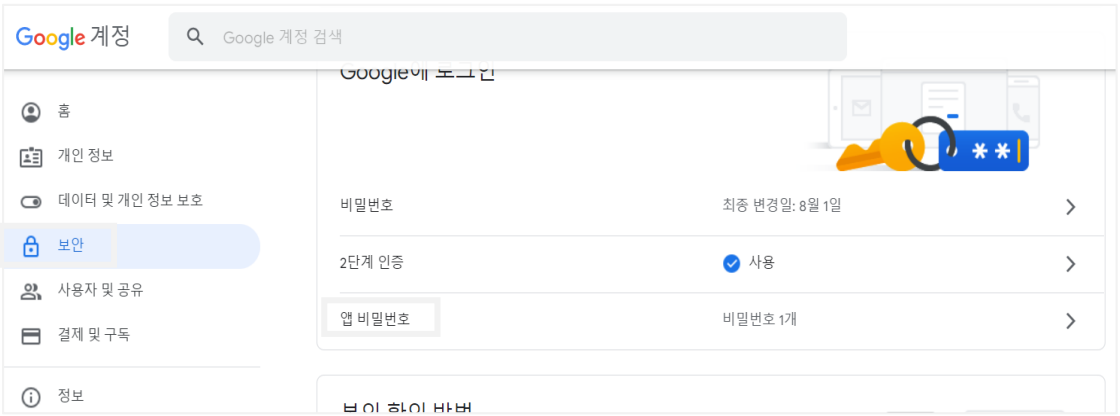
Server : AWS EC2 Ubuntu 20.04 LTS
Docker : 20.10.12
Nginx : 1.18.0
Jenkins : 2.346.2

4. 외부 서비스

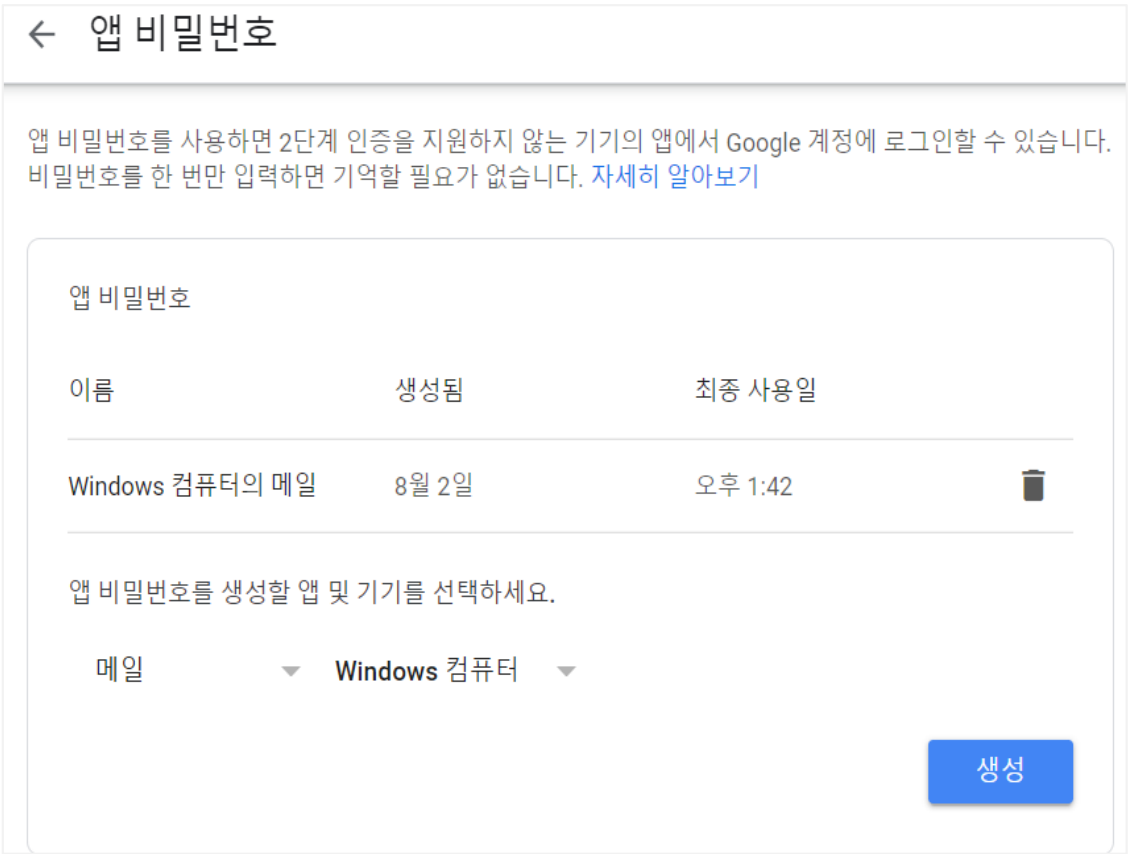
- **Google Gmail SMTP**

1. Google 메일 앱 비밀번호 발급

1-1. Google 계정 설정 - 2단계 인증 설정



1-2. Google 계정 설정 – 보안 – 앱 비밀번호



1-3. 메일, Windows 컴퓨터 비밀번호 생성

1-4. Windows 컴퓨터용 앱 비밀번호 확인

2. POP/IMAP 설정

2-1. Gmail – 모든 설정

2-2. 전달 및 POP/IMAP 설정

II. 프로젝트 실행 및 배포

1. 환경 변수

1-1. Frontend : React

- .env

```
REACT_APP_BASEURL="https://j7a708.p.ssafy.io"

REACT_APP_CONTRACTCA="0x71436d65377dA8909Dd9c89ee70227eF26AB05Ef"
```

1-2. Backend : Spring boot

- application.properties

```
spring.profiles.include=secret
server.port=8999

# mysql db
spring.datasource.driverClassName=com.mysql.cj.jdbc.Driver

# jpa
spring.jpa.database-platform=org.hibernate.dialect.MySQL5Dialect
spring.jpa.hibernate.ddl-auto=none
spring.mvc.pathmatch.matching-strategy=ant_path_matcher


spring.servlet.multipart.maxFileSize=20MB
spring.servlet.multipart.maxRequestSize=20MB
```

- **application-secret.properties**

1. mysql

```
# mysql db
# DB source URL
spring.datasource.url=jdbc:mysql://j7a708.p.ssafy.io:3306/kirin?serverTimezone=Asia/Seoul&characterEncoding=utf8
# DB username
spring.datasource.username=kirin
# DB password
spring.datasource.password=girin6
```

2. jwt

```
# jwt
jwt.secret=sdidifaADGASdafDSadsdaSDFDGVGdssbgewfdF
```

3. redis

```
# redis
spring.redis.host=j7a708.p.ssafy.io
spring.redis.password=girin6
spring.redis.port=8379
```

4. smtp

```
# email
spring.mail.host=smtp.gmail.com
spring.mail.port=587
spring.mail.username=noning2025
spring.mail.password=nyxpgnelmkhkycye
spring.mail.properties.mail.smtp.auth=true
spring.mail.properties.mail.smtp.starttls.enable=true
```

5. file 저장 경로 (docker volume)

```
# file 저장 경로
property.app.upload-path=/files/
```

6. 이더리움

```
#etereum
ADMIN_PRIVATE_KEY=0x6d08bfd0c97215c9e7eda22005053fa9e4a90840f2d3759d420c3dc7e7c88d4c
TOKENCONTRACTADDRESS=0xb9FcB43DCC98267f9C4Ed9E3eDA34244Ee162440
```

2. 배포 및 빌드

2-1. CI/CD 전체 구조

```
Ubuntu
├─Docker
│   ├─kirin-front : 0.0.0.0:443 -> 443/tcp, 0.0.0.0:80 -> 80/tcp
│   ├─kirin-back : 0.0.0.0:8999 -> 8999/tcp
│   ├─redis-container : 0.0.0.0:8379 -> 8379/tcp
│   └─mysql-container : 0.0.0.0:3306 -> 3306/tcp
│
└─Docker volume
    └─kirin_vol
```

2-2. Frontend: kirin_front

- **Dockerfile**

```
# node 이미지 사용
FROM node:16.15.0 as build-stage

# 명령어를 실행할 디렉터리
WORKDIR /usr/src/app

# package*.json에 등록되어있는 라이브러리들을 npm install으로 설치
COPY package*.json ./
RUN npm install

# 프로젝트 폴더를 이미지에 복사, npm run build를 통해 build폴더에 빌드한 프로젝트(static 파일)가 저장됨
COPY . .
RUN npm run build



# nginx 이미지 사용
FROM nginx

# # nginx의 기본 설정을 삭제하고 앱에서 설정한 파일을 복사
RUN rm -rf /etc/nginx/conf.d
COPY conf /etc/nginx

# 아까 빌드했던 파일들을 nginx의 서비스 폴더인 /usr/share/nginx/html으로 복사
COPY --from=build-stage /usr/src/app/build /usr/share/nginx/html

# 80포트 오픈하고 nginx 실행
EXPOSE 80

# nginx를 백그라운드에서 실행하는 컨테이너를 실행
ENTRYPOINT ["nginx", "-g", "daemon off;"]
```

- **nginx**

```
server {
    listen 80;
    listen [::]:80;

    server_name j7a708.p.ssafy.io;
    rewrite ^(.*) https://j7a708.p.ssafy.io$1 permanent;

    client_max_body_size 20M;
}

server {
    listen 443 ssl;
    listen [::]:443;

    server_name j7a708.p.ssafy.io;

    ssl_certificate /etc/letsencrypt/live/j7a708.p.ssafy.io/fullchain.pem;
    ssl_certificate_key /etc/letsencrypt/live/j7a708.p.ssafy.io/privkey.pem;

    client_max_body_size 20M;


    location / {
        root /usr/share/nginx/html; # 정적인 파일들의 루트 위치 (빌드한 파일 경로)
        index index.html index.htm; # 인덱스의 파일들 지정 (이 파일들 중 하나는 꼭 root 경로 안에 존재해야 함)
        try_files $uri $uri/ /index.html; # 어떤 파일을 찾을 때 명시된 순서로 찾고, 가장 먼저 발견되는 파일을 사용한다.
    }

    location /files {
        alias /usr/share/nginx/html/static/files;
    }

    location /api {
        proxy_pass http://j7a708.p.ssafy.io:8999;
    }

    location /bc/ {
        proxy_pass http://j7a708.p.ssafy.io:8888/;
    }

    location /api/notify {
        proxy_pass http://j7a708.p.ssafy.io:8999;
        proxy_read_timeout          300;
        proxy_send_timeout          300;
        proxy_connect_timeout          300;
        chunked_transfer_encoding off;
        proxy_buffering off;
        proxy_cache off;
    }
}
```

- **빌드**

```
cd ./frontend

# 생성된 도커 파일을 이용하여 이미지 빌드
docker build --tag nginx-react:latest .

if [ "$(docker ps -q -f name="kirin-front")" ]; then
    docker stop kirin-front
    docker rm kirin-front
fi
docker rmi $(docker images -f "dangling=true" -q)
docker run -v kirin_vol:/usr/share/nginx/html/static/files -d --name kirin-front -v /etc/letsencrypt/:/etc/letsencrypt/ -p 80:80 -p 443:443
nginx-react:latest
```

2-3. Backend: kirin_back

- **Dockerfile**

```
# open jdk java 17 버전의 환경을 구성 with alpine linux
FROM openjdk:17-jdk-alpine

WORKDIR /

RUN apk add ffmpeg
# build 시점에 활용되는 변수선언문 (target 폴더의 jar파일을 바라볼 것)
ARG JAR_FILE=build/libs/backend-0.0.1-SNAPSHOT.jar
# 이를 카피하여 app.jar 로 복사
COPY ${JAR_FILE} app.jar
# 이 컨테이너가 어떤 포트를 사용하는지
EXPOSE 8999
# 기존 jar 파일을 단순 실행
ENTRYPOINT ["java","-jar","/app.jar"]
```

- **빌드**

```
cd ./backend

# 생성된 도커 파일을 이용하여 이미지 빌드
docker build --tag kirin-jdk:latest .

if [ "$(docker ps -q -f name="kirin-back")" ]; then
    docker stop kirin-back
    docker rm kirin-back
fi
docker rmi $(docker images -f "dangling=true" -q)
docker run -v kirin_vol:/files -v /media:/media --name kirin-back -d -p 8999:8999 kirin-jdk:latest
```

2-4. BlockChain(노드 연결)

- **window에서 Geth 설치**

```
mkdir src\github.com\ethereum

git clone https://github.com/ethereum/go-ethereum --branch v1.9.24
src\github.com\ethereum/go-ethereum

cd src\github.com\ethereum\go-ethereum

dir(설치 확인)

go get -u -v golang.org/x/net/context (실행)

go install -v ./cmd/...

geth version (1/9/24-stable) 나오면 ok
```

- **CustomGensis.json 수정**

```
cd dev/eth_localdata/
vi CustomGenesis.json

{
    "config": {
            "chainId": 97889218,
            "homesteadBlock": 0,
            "eip150Block": 0,
            "eip155Block": 0,
            "eip158Block": 0,
            "byzantiumBlock": 0,
            "constantinopleBlock": 0,
            "petersburgBlock": 0,
            "istanbulBlock": 0
        },
    "difficulty": "0x01",
    "coinbase": "0x0000000000000000000000000000000000000000",
    "gasLimit": "9999999999999999",
    "alloc": {},
    "extraData": "",
    "nonce": "0xdeadbeefdeadbeef",
    "mixhash": "0x0000000000000000000000000000000000000000000000000000000000000000",
    "parentHash": "0x0000000000000000000000000000000000000000000000000000000000000000",
    "timestamp": "0x00"
}
```

- **Geth init & node 연결**

```
geth --datadir ./ init ./CustomGenesis.json
```

```
geth --networkid 97889218 --datadir ~/dev/eth_localdata/
--miner.etherbase 0x8b8a556c42CA423Bb909E0Fc5f48726EedcB60F7
--miner.threads=2 --bootnodes
"enode://7aa6379643e6efdad4a0b3cba88a86c04fb3665bfe95e17ceb60212240472c311c9932
c8ed2cadb8832c5fcf403f74cb93064bc625a5f912b63bf75d20e814dc@13.125.49.193:34567"
--authrpc.port 8552 console
```

- **Mining 시작**

```
miner.start()
```