포팅 메뉴얼

1. 개요

- 프로젝트 명
    - PLOG
- 프로젝트 소개
    - 나만의 맞춤형 플로깅 장소를 추천받고 이를 효과적으로 기록하는 서비스
- 주요 기능
    1. 맞춤형 플로깅 코스 제안
    2. 일지 저장
    3. 일지 공유 커뮤니티

1.2. 프로젝트 사용 도구

- 이슈 관리
    - JIRA
- 형상 관리
    - Gitlab
- API 테스트

    - Postman
    - Swagger
- 커뮤니케이션
    - Notion
    - Mattermost
- 디자인
    - Figma
- 데이터베이스 설계
    - ERD Cloud
- 동영상 편집
    - 모바비

1.3. 개발 환경

- 프론트엔드
    - Visual Studio Code
- 백엔드
    - IntelliJ IDEA
    - Pycharm
- 인프라
    - putty
- 데이터베이스
    - DataGrip

1.4. 기술 스택

- 프론트엔드
    - React Native 0.75.4
    - JavaScript

    - HTML5
    - CSS3
    - Node.js 20.15.0
    - react-native-seoul/kakao-login : 5.4.1
    - react-native-maps : 1.18.0
- 백엔드
    - Java 17
    - SpringBoot 3.3.2
    - Spring Data JPA
    - FastApi 0.103
- 데이터베이스
    - postgresql
- 인프라
    - Docker
    - Jenkins
    - NginX
    - AWS EC2
    - AWS S3

2. 빌드

2.1. 프론트엔드 빌드 방법

1. 버전
    a. React Native 0.75.4
    b. Nodejs 20.15.0
2. 라이브러리 설치

```
npm install
```

3. 빌드
    a. 안드로이드 스튜디오 에뮬레이터 실행

```
npm run android
```

2.2. 백엔드 빌드 방법

SPRING

1. 버전
    a. JAVA Open-JDK 17
    b. SpringBoot 3.3.2
    c. Gradle 8.8
2. 빌드

```
./gradlew build
```

FastApi

1. 버전
    a. Python 3.12.5
    b. FastApi 0.103
2. 빌드

```
# 라이브러리 설치
pip install -r requirements.txt

# 빌드
uvicorn src.main:app --host 0.0.0.0 --port 8000
```

2.3. 배포하기

1. 배포 환경

a. AWS EC2 (Ubuntu 20.04.6)
b. AWS S3
c. docker 27.1.1
d. jenkins 2.452.3

2. 배포 방법
   a. 설치
      i. docker 설치

```
# apt update
sudo apt update

# 패키지 설치
$sudo apt-get install apt-transport-https ca-certi
ficates curl gnupg-agent software-properties-commo
n

# gpg 설치 및 도커 저장소 key 저장
sudo curl -fsSL https://download.docker.com/linux/
ubuntu/gpg | sudo apt-key add -

# 도커 다운로드 및 레포지토리에 추가
sudo add-apt-repository "deb [arch=amd64] https://
download.docker.com/linux/ubuntu $(lsb_release -c
s) stable"

# 도커 설치
apt-cache policy docker-ce
sudo apt install docker-ce
```

      ii. jenkins 설치

```
 # port 설정 8080포트로 외부에서 접속
 # volume(/var/jenkins_home)을 통해서 로컬과 연결
 # sock를 통해서 인증
 docker run -d --name jenkins \
    -p 8080:8080 -p 50000:50000 \
```

```
    -v /var/jenkins_home:/var/jenkins_home \
    -v /var/run/docker.sock:/var/run/docker.sock \
    --privileged \
    --user root \
    jenkins/jenkins:lts
```

b. CI/CD 환경 구축

i. nginx를 위해서 미리 네트워크 생성

```
docker network create app-network
```

ii. GitLab과 연결

1. gitlab에서 acess 토큰 발급
2. 해당 acess 토큰 jenkins credentials에 등록
    a. 이때 Username with password로 생성하여 자기 아이디 토큰 입력
3. jenkins에 item생성
4. git lab build trigger 설정
    a. push
5. git lab과 훅 걸기
    a. project hook에서 jenkins project의 url 넣기(build trigger 설정하는 곳에 존재)
    b. token 넣어주기 (build trigger 설정하는 곳에 존재)
    c. push event에 Regular expression에 ^test$ 이런식으로 입력해서 특정 branch 선택 가능

iii. nginx 설정

1. nginx 설정
    a. nginx 설정 파일 생성 (/etc/nginx/templates/nginx.conf.template)

```
server {
    listen 443 ssl;
    server_name j11b205.p.ssafy.io;  # 서버의 도
메인 또는 IP 주소
```

```
    ssl_certificate /etc/nginx/ssl/fullchain.pe
m;
    ssl_certificate_key /etc/nginx/ssl/privkey.
pem;



    # 프록시 요청을 받을 기본 위치 설정
    location /api/ {
        proxy_pass http://spring:8080;  # 백엔드
서버의 주소와 포트
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_add
r;
        proxy_set_header X-Forwarded-For $proxy
_add_x_forwarded_for;
        proxy_set_header X-Forwarded-Proto $sch
eme;

        # 연결 시간 설정
        proxy_connect_timeout 60s;
        proxy_send_timeout 60s;
        proxy_read_timeout 60s;

        # 응답에서 지연을 최소화하기 위한 설정
        proxy_buffering off;
    }

    # 프록시 요청을 받을 기본 위치 설정
    location / {
        proxy_pass http://python:8000;  # 파이썬
서버의 주소와 포트
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_add
r;
        proxy_set_header X-Forwarded-For $proxy
_add_x_forwarded_for;
```

```
        proxy_set_header X-Forwarded-Proto $sch
eme;

        # 연결 시간 설정
        proxy_connect_timeout 60s;
        proxy_send_timeout 60s;
        proxy_read_timeout 60s;

        # 응답에서 지연을 최소화하기 위한 설정
        proxy_buffering off;
    }
}

server {
    listen 80;
    server_name j11b205.p.ssafy.io;

    # Redirect HTTP to HTTPS
    return 301 https://$host$request_uri;
}
```

1. 443 포트로 들어오면 url을 바탕으로 proxy
    a. /api가 붙어 있는 경우 spring container의 8080 포트로 이동
    b. 아무것도 없는 경우 python container의 80 포트로 이동
    c. docker내의 dns 서버를 활용하여 python,spring 컨테이너 찾아주기
    d. 인증서를 사용하여 https 설정
2. 80포트로 들어온 경우
    a. 443 포트(https://domain.com)으로 보냄

iv. 백엔드 CI/CD 구축

1. 파이프 라인 구축

```
pipeline {
    agent any
```

```
    environment {
        SPRING_IMAGE = 'spring-app' // Spring
백엔드 이미지 이름 설정
        COMPOSE_FILE = 'docker-compose.yml' //
도커 컴포즈 파일 이름 설정
    }
    tools {
        gradle 'gradle' // Gradle 도구 사용
    }

    stages {
        stage('Checkout') {
            steps {
                // Git 리포지토리에서 'develop/ser
ver-restapi' 브랜치를 체크아웃
                git branch: 'develop/server-res
tapi',
                url: 'https://lab.ssafy.com/s11
-bigdata-recom-sub1/S11P21B205.git',
                credentialsId: 'GrowthookB205'
// GitLab 인증 정보 사용
            }
        }

        stage('Build Spring') {
            steps {
                script {
                    dir('Plog') {
                        sh 'chmod +x gradlew'
                        sh './gradlew clean bui
ld'
                    }
                }
            }
        }

        stage('Stop and Remove Existing Contain
ers') {
```

```
            steps {
                script {
                    // 기존 컨테이너 중지 및 제거
                    sh 'docker-compose -f ${COM
POSE_FILE} down'
                }
            }
        }

        stage('Remove Existing Docker Images')
{
            steps {
                script {
                    // 기존 Spring 이미지 삭제
                    sh "docker rmi -f \$(docker
images -q ${SPRING_IMAGE}) || true"
                }
            }
        }

        stage('Build Docker Image Spring') {
            steps {
                script {
                    dir('Plog'){
                        sh 'docker build -t ${S
PRING_IMAGE} .'
                    }
                }
            }
        }

        stage('docker-compose up') {
            steps {
                script {
                    // 새로운 컨테이너 시작
                    sh 'docker-compose -f ${COM
POSE_FILE} up -d'
                }
```

```
            }
        }

        stage('Nginx Restart') {
            steps {
                script {
                    sh 'docker restart nginx'
                }
            }
        }
    }
}

-----------------------------------------------
----------------------

pipeline {
    agent any

    environment {
        PYTHON_IMAGE = 'python-app'
        COMPOSE_FILE = 'docker-compose.yml' //
도커 컴포즈 파일 경로 설정
    }

    stages {
        stage('Checkout') {
            steps {
                // Git 리포지토리에서 'develop/ser
ver-restapi' 브랜치를 체크아웃
                git branch: 'develop/server-clu
ter',
                url: 'https://lab.ssafy.com/s11
-bigdata-recom-sub1/S11P21B205.git',
                credentialsId: 'GrowthookB205'
// GitLab 인증 정보 사용
            }
        }
```

```
        stage('Stop Existing Containers') {
            steps {
                script {
                    // 기존 컨테이너 중지
                    sh 'docker-compose -f ${COM
POSE_FILE} down'
                }
            }
        }

        stage('Remove Existing Docker Images')
{
            steps {
                script {
                    // 기존 Spring 이미지 삭제
                    sh "docker rmi -f \$(docker
images -q ${PYTHON_IMAGE}) || true"
                }
            }
        }

        stage('build image') {
            steps {
                dir('pythonProject'){
                    sh 'docker build -t ${PYTHO
N_IMAGE} .'
                }
            }
        }

        stage('docker-compose up') {
            steps {
                script {
                    // 새로운 컨테이너 시작
                    sh 'docker-compose -f ${COM
POSE_FILE} up -d'
                }
```

```
            }
        }

        stage('Nginx Restart') {
            steps {
                script {
                    sh 'docker restart nginx'
                }
            }
        }
    }
}
```

1. push, merge가 들어옴
2. branch 이동
3. config 파일 가져오기
    a. config 파일 생성
    b. credential에 파일 등록 및 이름 설정 ⇒ 해당 이름을 key로 파일 찾기 가능
4. build
5. 생성된 build파일의 jar파일을 image화
    a. docker file을 미리 설정 (이때 directory 시작은 git clone을 했을때 바로 있는 폴더)
    b. docker file 코드

```
FROM openjdk:17-alpine // 사용할 버전
ARG JAR_FILE=/build/libs/BITAMIN-0.0.1-SNAPSHOT.jar // build된 jar 파일로 이동
COPY ${JAR_FILE} app.jar //jar 파일을 app.jar 파일로 복사
ENTRYPOINT ["java","-jar","/app.jar"] // java -jar app.jar로 실행
```

6. 해당 image를 Build ⇒ 이미지 생성
7. docker-compose를 내리고 생성된 image를 다시 실행

```
version: '3.8'

services:
  spring:
    container_name: spring
    image: spring-app
    networks:
      - app-network

networks:
  app-network:
    external: true
```

1. image를 실행 시켜 container 생성
2. 여기에서 network 까지 같이 연결해서 하나의 network에 넣어서 해당 container에 접근 가능하도록 만듬

8. nginx 재시작

c. 포트 개방

i. 443 ⇒ https로 접속을 위해 개방

ii. 80 ⇒ http로 접속을 위해 개방

iii. 8080 ⇒ jenkins 접속을 위해 개방