포팅 매뉴얼 : 까까(까도 내가까)

> **SSAFY 10기 공통 프로젝트 구미 1반 10팀**

0. 목차

1. 개발 환경

1. 프로젝트 기술 스택

- Frontend
  - Visual Studio Code(IDE) `1.81.1`
  - HTML5, CSS3, Javascript(ES6)
  - React : `18.2.0`
  - Electron `28.1.4`
  - Stompjs `2.3.3`
  - Vite `5.0.8`
  - Typescript `5.2.2`
  - Tailwind CSS `3.4.1`
  - Zustand `4.4.7`
  - Tanstack Query `5.17.15`
  - Nodejs `20`
- Backend
  - Intellij : `2023.3.2`
  - JVM OpenJDK : `17`
  - JWT : `0.11.5`
  - Spring Boot : `3.0.13`
    - JAVA Spring Data JPA
    - Spring Security
    - SSEEmitter
  - OAuth : `6.8.0`
  - Lettuce : `6.2.7`
  - spring-boot-WebSocket : `10.1.16`

    - Gradle
    - ORM : JPA
- CI/CD
    - AWS EC2
        - Nginx : `1.18.0`
        - Ubuntu : `20.04 LTS`
        - Docker : `25.0.2`
        - Jenkins : `2.443`
    - Docker Hub

2. 환경변수 설정

Frontend : .env

```
VITE_KAKAO_REST_API_KEY='카카오 api 키'

# 배포용
VITE_KAKAO_REDIRECT_URI='https://i10d110.p.ssafy.io/oauth/callback/kakao/token'
VITE_API_BASE_URL='https://i10d110.p.ssafy.io'
VITE_API_BASE_NEXT_URL='/api/oauth/callback/kakao/token/d-t-d?code'
# 아래는 일렉트론 관련 API 키
API_BASE_URL='https://i10d110.p.ssafy.io'

############################################################################

# 로컬용
# VITE_KAKAO_REDIRECT_URI='http://localhost:3000/oauth/callback/kakao/token'
# VITE_API_BASE_URL='http://localhost:8080'
# VITE_API_BASE_NEXT_URL='/api/oauth/callback/kakao/token/l-t-l?code'
```

```
# # 아래는 일렉트론 관련 API 키
# API_BASE_URL='http://localhost:8080'
```

Backend : application 파일

- application.yml

```
spring:
  profiles:
    active: local
  servlet:
    multipart:
      enabled: true # 멀티파트 업로드 지원여부 (default: true)
      max-file-size: 100MB
      max-request-size: 100MB
  data:
    redis:
      host: 52.78.162.26
      port: 6379
      lettuce:
        pool:
          max-active: 100 # pool에 할당될 수 있는 최대 커넥션 수
          max-idle: 10   #  pool에 할당 될 수 있는 최대 idle 커
넥션 수
          min-idle: 2    #  pool에서 관리하는 최소 idle 커넥션
대상
      password: ${redis.password}

logging:
  level:
    org:
      apache:
        http: DEBUG
    httpclient:
      wire: DEBUG
kakao:
  clientId : ${spring.kakao.client-id}
  secret: ${spring.kakao.client-secret}
```

```
cloud:
  aws:
    credentials:
      access-key: ${aws.s3.accesskey}
      secret-key: ${aws.s3.secretkey}
    region:
      static: ap-northeast-2
    stack:
      auto: false
    s3:
      bucket: ssafys3
```

- application-dev.yml

```
spring:
  jpa:
    database: mysql
    show-sql: true
    hibernate:
      ddl-auto: validate
  datasource:
    driver-class-name: com.mysql.cj.jdbc.Driver
    url: jdbc:mysql://database-1.ctee4gmysdpo.ap-northeast-
2.rds.amazonaws.com:3306/ssafy?serverTimezone=Asia/Seoul&ch
aracterEncoding=UTF-8
    username: ${rds.username}
    password: ${rds.password}
server:
  servlet:
    encoding:
      force: 'true'
      enabled: 'true'
      charset: UTF-8
    context-path: /
  port: '8080'
```

- application-local.dev

```
spring:
  jpa:
    database: mysql
    show-sql: true
    hibernate:
      ddl-auto: validate
  datasource:
    password: ${datasource.password}
    driver-class-name: com.mysql.cj.jdbc.Driver
    username: ${datasource.username}
    url: ${datasource.url-mysql}
server:
  servlet:
    encoding:
      force: 'true'
      enabled: 'true'
      charset: UTF-8
    context-path: /
  port: '8080'
```

3. 설정 파일

Nginx : nginx.conf

- 설정 파일 위치

```
/etc/nginx/conf.d/nginx
```

- nginx.conf

```
server {
    listen 80 default_server;
    listen [::]:80 default_server;
    server_name _;

    root /var/www/html;
    index index.html index.htm index.nginx-debian.html;
```

```
    location / {
        try_files $uri $uri/ =404;
    }
}

server {
    listen 443 ssl http2 default_server;
    listen [::]:443 ssl http2 default_server;
    server_name i10d110.p.ssafy.io;

    ssl_certificate /etc/letsencrypt/live/i10d110.p.ssafy.i
o/fullchain.pem;
    ssl_certificate_key /etc/letsencrypt/live/i10d110.p.ssa
fy.io/privkey.pem;
    include /etc/letsencrypt/options-ssl-nginx.conf;
    ssl_dhparam /etc/letsencrypt/ssl-dhparams.pem;

    location / {
        proxy_pass http://localhost:3000;
        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;
    }

    location /api/ {
        proxy_pass http://localhost:8080/api/;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwa
rded_for;
        proxy_set_header Host $http_host;
    }

    location /ws-stomp {
        proxy_pass http://localhost:8080/ws-stomp;
        proxy_http_version 1.1;
```

```
        proxy_set_header Upgrade $http_upgrade;
        proxy_set_header Connection "Upgrade";
        proxy_set_header Host $http_host;
    }


    location /api/alarm/subscribe {
        proxy_pass http://localhost:8080/api/alarm/subscrib
e;
        proxy_set_header Connection '';
        proxy_set_header Cache-Control 'no-cache';
        proxy_set_header X-Accel-Buffering 'no';
        proxy_set_header Content_Type 'text/event-stream';
        proxy_buffering off;
        proxy_http_version 1.1;
        chunked_transfer_encoding on;
        proxy_read_timeout 86400s;
    }
}

server {
    listen 80;
    listen [::]:80;
    server_name i10d110.p.ssafy.io;

    return 301 https://$host$request_uri;
}
```

Jenkins : Jenkinsfile

- 설정 파일 위치 : 프로젝트 최상단
- Jenkinsfile

```
pipeline {
    agent any

    environment {
        BuildGradle     = credentials('build.gradle')
```

```
        Mat_Endpoint    = credentials('CICD_mat_endpoint')
        // AWS_SECRET_ACCESS_KEY = credentials('jenkins-aws
-secret-access-key')
    }
    stages {
        stage('MM-Alarm'){
            steps{
                script {
                    def Author_ID = sh(script: "git show -s
--pretty=%an", returnStdout: true).trim()
                    def Author_Name = sh(script: "git show
-s --pretty=%ae", returnStdout: true).trim()
                    mattermostSend (
                        color: '#D0E0E3',
                        icon: "https://jenkins.io/images/lo
gos/jenkins/jenkins.png",
                        message: "파이프라인 시작: ${env.JOB_N
AME} #${env.BUILD_NUMBER} by ${Author_ID}(${Author_Name})\n
(<${env.BUILD_URL}|Details>)"
                    )
                }
            }
        }


        stage('Clone') {
            steps {
                echo '클론을 시작!'
                git branch: 'dev', credentialsId: 'docker-h
ub', url: 'https://lab.ssafy.com/s10-webmobile2-sub2/S10P12
D110.git'
                echo '클론을 완료!'
            }
        }

        stage('BE-Build') {
            steps {
                echo '백엔드 빌드 및 테스트 시작!'
```

```
                dir("./backend") {
                    sh "chmod +x ./gradlew"

                    // sh "touch ./build.gradle"

                    // application properties 파일 복사
                    // sh "echo $BuildGradle > ./build.grad
le"

                    sh "./gradlew clean build --exclude-tas
k test"

                }
                echo '백엔드 빌드 및 테스트 완료!'
            }
        }

        stage('Build Back Docker Image') {
            steps {
                echo '백엔드 도커 이미지 빌드 시작!'
                dir("./backend") {
                    // 빌드된 JAR 파일을 Docker 이미지로 빌드
                    sh "docker build -t osy9536/ssafy-be:la
test ."
                }
                echo '백엔드 도커 이미지 빌드 완료!'
            }
        }

        stage('Push to Docker Hub-BE') {
            steps {
                echo '백엔드 도커 이미지를 Docker Hub에 푸시 시
작!'
                withCredentials([usernamePassword(credentia
lsId: 'docker-hub', usernameVariable: 'DOCKER_USERNAME', pa
sswordVariable: 'DOCKER_PASSWORD')]) {
                    sh "docker login -u $DOCKER_USERNAME -p
$DOCKER_PASSWORD"
```

```
                }
                dir("./backend") {
                    sh "docker push osy9536/ssafy-be:lates
t"
                }
                echo '백엔드 도커 이미지를 Docker Hub에 푸시 완
료!'
            }
        }

        stage('Deploy to EC2-BE') {
            steps {
                echo '백엔드 EC2에 배포 시작!'
                // 여기에서는 SSH 플러그인이나 SSH 스크립트를 사용
하여 EC2로 연결하고 Docker 컨테이너 실행
                sshagent(['aws-key']) {
                    sh "docker rm -f backend"
                    sh "docker rmi osy9536/ssafy-be:latest"
                    sh "docker image prune -f"
                    sh "docker pull osy9536/ssafy-be:latest
&& docker run -d -p 8080:8080 --name backend osy9536/ssafy-
be:latest"
                }
                echo '백엔드 EC2에 배포 완료!'
            }
        }

        stage('FE-Build') {
            steps {
                echo '프론트 빌드 및 테스트 시작!'
                dir("./frontend") {
                    sh "npm install"
                    sh "npm run build"
                }
                echo '프론트 빌드 및 테스트 완료!'
            }
        }
```

```
        stage('Build Front Docker Image') {
            steps {
                echo '프론트 도커 이미지 빌드 시작!'
                dir("./frontend") {
                    // 빌드된 파일을 Docker 이미지로 빌드
                    sh "docker build -t osy9536/ssafy-fe:la
test ."
                }
                echo '프론트 도커 이미지 빌드 완료!'
            }
        }

        stage('Push to Docker Hub-FE') {
            steps {
                echo '프론트 도커 이미지를 Docker Hub에 푸시 시
작!'
                withCredentials([usernamePassword(credentia
lsId: 'docker-hub', usernameVariable: 'DOCKER_USERNAME', pa
sswordVariable: 'DOCKER_PASSWORD')]) {
                    sh "docker login -u $DOCKER_USERNAME -p
$DOCKER_PASSWORD"
                }
                dir("./frontend") {
                    sh "docker push osy9536/ssafy-fe:lates
t"
                }
                echo '프론트 도커 이미지를 Docker Hub에 푸시 완
료!'
            }
        }

        stage('Deploy to EC2-FE') {
            steps {
                echo '프론트 EC2에 배포 시작!'
                // 여기에서는 SSH 플러그인이나 SSH 스크립트를 사용
하여 EC2로 연결하고 Docker 컨테이너 실행
                sshagent(['aws-key']) {
                    sh "docker rm -f frontend"
```

```
                    sh "docker rmi osy9536/ssafy-fe:latest"
                    sh "docker image prune -f"
                    sh "docker docker pull osy9536/ssafy-f
e:latest && run -d -p 3000:3000 --name frontend osy9536/ssa
fy-fe:latest"
                }
                echo '프론트 EC2에 배포 완료!'
            }
        }

    }

    post {
        success {
            echo '파이프라인이 성공적으로 완료되었습니다!'
            script {
                def Author_ID = sh(script: "git show -s --p
retty=%an", returnStdout: true).trim()
                def Author_Name = sh(script: "git show -s -
-pretty=%ae", returnStdout: true).trim()
                mattermostSend (
                    color: '#D0E0E3',
                    icon: "https://jenkins.io/images/logos/
jenkins/jenkins.png",
                    message: "빌드 성공: ${env.JOB_NAME} #${e
nv.BUILD_NUMBER} by ${Author_ID}(${Author_Name})\n(<${env.B
UILD_URL}|Details>)"
                )
            }
        }
        failure {
            echo '파이프라인이 실패하였습니다. 에러를 확인하세요.'
            script {
                def Author_ID = sh(script: "git show -s --p
retty=%an", returnStdout: true).trim()
                def Author_Name = sh(script: "git show -s -
-pretty=%ae", returnStdout: true).trim()
                mattermostSend (
```

```
                    color: '#D0E0E3',
                    icon: "https://4.bp.blogspot.com/-52EtG
jEhW-k/UtOBXa1fhVI/AAAAAAAABbU/Lk4ZBYcvZrY/s1600/download.j
peg",
                    message: "빌드 실패: ${env.JOB_NAME} #${e
nv.BUILD_NUMBER} by ${Author_ID}(${Author_Name})\n(<${env.B
UILD_URL}|Details>)"
                )
            }
        }
    }
}
```

Dockerfile : Backend

- 설정 파일 위치

```
/backend/dockerfile
```

- dockerfile

```
FROM openjdk:17-jdk
EXPOSE 8080
ARG JAR_FILE=build/libs/*.jar
COPY ${JAR_FILE} app.jar
ENV TZ=Asia/Seoul
ENTRYPOINT ["java","-jar","/app.jar"]
```

Dockerfile : Frontend

```
FROM node:alpine as builder
WORKDIR /frontend
COPY package.json .
# RUN npm install
COPY ./ ./
RUN npm run build

# 3000번 포트 노출
```

```
EXPOSE 3000

# npm start 스크립트 실행
CMD ["npm", "run", "dev"]
```

2. 배포 방법

1. EC2 내부 방화벽 설정

포트 허용

```
sudo ufw allow 8080/tcp # ssh는 tcp 프로토콜만 허용해야함
sudo ufw status # 방화벽 포트 상태 확인
sudo ufw deny 8080/tcp
```

2. 도커 설치

업데이트 및 HTTP 패키지 설치

```
sudo apt update
sudo apt-get install -y ca-certificates \
  curl \
  software-properties-common \
  apt-transport-https \
  gnupg \
  lsb-release
```

GPG 키 및 저장소 추가

```
sudo mkdir -p /etc/apt/keyrings
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | s
udo gpg --dearmor -o /etc/apt/keyrings/docker.gpg

echo \
```

```
"deb [arch=$(dpkg --print-architecture) signed-by=/etc/apt/
keyrings/docker.gpg] https://download.docker.com/linux/ubun
tu \
$(lsb_release -cs) stable" | sudo tee /etc/apt/sources.lis
t.d/docker.list > /dev/null
```

설치 가능 버전 확인

```
apt-cache madison docker-ce
```

도커 설치

```
sudo apt update
sudo apt-get install docker-ce=<VERSION_STRING> docker-ce-c
li=<VERSION_STRING> containerd.io
```

도커 확인

```
sudo docker run hello-world # 또는
sudo docker version
```

3. 젠킨스 설치

Docker 컨테이너에 마운트 할 볼륨 디렉토리 설치

```
cd /home/ubuntu && mkdir jenkins-dat a
```

포트 설정

```
sudo ufw allow 9090/tcp
sudo ufw reload
sudo ufw status
```

도커로 젠킨스 컨테이너 생성 및 구동

```
sudo docker run -d -p 9090:8080 -v /home/ubuntu/jenkins-dat
a:/var/jenkins_home -v /var/run/docker.sock:/var/run/docke
r.sock --name jenkins jenkins/jenkins:lts
```

- Docker out of Docker (DooD) 설정을 위한 볼륨 지정

초기 패스워드 확인

```
sudo docker logs jenkins
```

- 중간에 나오는 패스워드 확인

```
************************************************************
************************************************************
************************************************************
************************************************************
****************************

casjije2ij10ue9qwdjsvogh4893uq2ejoadncvhw23y89q

************************************************************
************************************************************
************************************************************
************************************************************
****************************
```

4. Git 설치 및 프로젝트 셋팅

Git 설치

```
sudo apt-get install git
sudo apt install git
```

Git 버전 확인

```
sudo git --version
```

Git 계정 설정

```
sudo git config --global user.name ${유저 이름}
sudo git config --global user.email ${유저 이메일}
```

프로젝트 Clone

```
sudo git clone ${클론 받을 깃 레포지토리 url} ${다운받아올 폴더명}
```

5. HTTPS 적용

80, 443포트 방화벽 해제

```
sudo ufw allow 80
sudo ufw allow 80/tcp
sudo ufw allow 443
sudo ufw allow 443/tcp
sudo ufw enable
sudo ufw status
```

기본 라이브러리 설치

```
sudo apt-get install -y build-essential
sudo apt-get install curl
```

Nginx 설치 및 conf 파일 작성

```
sudo apt-get install nginx
sudo vi /etc/nginx/conf.d/nginx.conf
```

위에 쓴 Nginx.conf 파일 작성

Certbot 설치 및 SSL 인증서 발급

```
sudo apt-get remove certbot
sudo snap install --classic certbot

# nginx 자동 설정
sudo certbot --nginx
```