포팅매뉴얼

프로젝트 사용 도구

- 이슈 관리 : Jira
- 형상 관리 : GitLab
- 커뮤니케이션 : Notion, Mattermost
- 테스트 : Postman, Swagger
- 로깅 : Discord
- UI/UX : Figma

프로젝트 개발 환경

Frontend

- Visual Studio Code : `1.85.1`
- React : `18.2.56`
- React-redux : `9.1.0`
- React-dom : `18.2.0`
- Typescript : `5.2.2`
- Node.js : `20.11.25`
- npm: `10.4.0`
- Vite : `5.1.5`

Backend

- IntelliJ : `2023.03`
- Java : `17`
- SpringBoot : `3.2.3`
- SpringSecurity : `3.2.3`
- JPA : `3.2.3`
- Lombok : `1.18.30`
- Python : `3.9.11`

DB

- MySQL : `8.3.0`
- Redis : `7.2.4`
- Flyway: `9.22.3`

Service

- NginX : `1.18.0`
- Jenkins : `2.451`
- Docker : `25.0.5`

Server

- Ubuntu : `20.04`

외부서비스

- Kakao Login API
- AWS S3

빌드

Frontend

```
npm i
npm run build
```

Backend

```
Gradle -> build
```

프로젝트 환경 변수

Backend

application.yml

```
spring:
  profiles:
    active: prod
  datasource:
    driver-class-name: com.mysql.cj.jdbc.Driver
    url: jdbc:mysql://${DB_HOST}:${HOST_PORT}/${MYSQL_DATABAS
    username: ${MYSQL_USER}
    password: ${MYSQL_PASSWORD}
  jpa:
    hibernate:
      ddl-auto: none
    show-sql: true
    properties:
      hibernate:
        format_sql: true
        show_sql: true
        jdbc:
          time_zone: Asia/Seoul
```

```
  kafka:
    bootstrap-servers: ${KAFKA_HOST}:{KAFKA_PORT}
  data:
    redis:
      host: ${REDIS_HOST}
      port: ${REDIS_PORT}

  flyway:
    enabled: true

server:
  servlet:
    context-path: ${CONTEXT_PATH}
logging:
  level:
    root: info

# constants
DOMAIN:
  FRONT: ${FRONT_DOMAIN}
  BACK: ${BACK_DOMAIN}
KAKAO:
  CLIENT_ID: ${KAKAO_OAUTH2_CLIENT_ID}
  REDIRECT_URI: ${BACK_DOMAIN}${CONTEXT_PATH}/auth/oauth2/kak
secretKeyPlain: ${SECURE_KEY_PLAIN}

discord:
  webhook: ${DISCORD_WEBHOOK}

cloud:
  aws:
    credentials:
      access-key: ${S3_ACCESSKEY}
      secret-key: ${S3_SECRETKEY}
    s3:
      bucket: ${S3_BUCKET}
    region:
      static: ${S3_REGION}
```

```
    stack:
      auto: false
```

.env

```
# MySQL
MYSQL_ROOT_PASSWORD=string
MYSQL_DATABASE=string
MYSQL_USER=string
MYSQL_PASSWORD=string
DB_HOST=string
HOST_PORT=0000

# Redis
REDIS_HOST=string
REDIS_PASSWORD=string

# OAuth2 Kakao
KAKAO_OAUTH2_CLIENT_ID=string

#S3
S3_ACCESSKEY=string
S3_SECRETKEY=string
S3_BUCKET=string
S3_REGION=string

# Frontend-domain
FRONT_DOMAIN=http://localhost:5173
# Back-domain
BACK_DOMAIN=http://localhost:8080

# 스프링 시큐리티 키
SECURE_KEY_PLAIN=string

# 디스코드 웹훅 uri
DISCORD_WEBHOOK=string
```

.gitignore

```
# config
**/.env
```

배포

목차

1. Docker/Jenkins 설치
2. NginX 설정
3. MySQL/Redis/Kafka 설치
4. Backend - API 서버(Spring Boot) 배포
5. Frontend - 토비의 당근밭 App 배포
6. AI 서버(Python) 배포 - drawing, emotion, detection 공통

1. Docker/Jenkins 설치

1.1 Docker 설치

```
sudo apt-get -y install apt-transport-https ca-certificates
curl gnupg-agent software-properties-common | curl -fsSL ht
tps://download.docker.com/linux/ubuntu/gpg | sudo apt-key a
dd - | sudo add-apt-repository "deb [arch=amd64] https://do
wnload.docker.com/linux/ubuntu $(lsb_release -cs) stable" |
sudo apt-get -y install docker-ce docker-ce-cli containerd.
io
```

1.2 Docker-compose 설치

```
sudo curl -L "https://github.com/docker/compose/releases/do
wnload/v2.21.0/docker-compose-$(uname -s)-$(uname -m)" -o /
```

```
usr/local/bin/docker-compose
```

1.3 Jenkins 설치

```
docker pull jenkins/jenkins:jdk17 | docker run -d --restart always --env JENKINS_OPTS=--httpPort=<포트번호> -v /etc/localtime:/etc/localtime:ro -e TZ=Asia/Seoul -p <포트번호>:<포트번호>-v /jenkins:/var/jenkins_home -v /var/run/docker.sock:/var/run/docker.sock -v /usr/local/bin/docker-compose:/usr/local/bin/docker-compose --name jenkins -u root jenkins/jenkins:jdk17
```

1.4 Jenkins 내부 Docker 패키지 설치

```
apt-get update && apt-get -y install apt-transport-https ca-certificates curl gnupg2 software-properties-common && curl -fsSL https://download.docker.com/linux/$(. /etc/os-release; echo "$ID")/gpg > /tmp/dkey; apt-key add /tmp/dkey && add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/$(. /etc/os-release; echo "$ID") $(lsb_release -cs) stable" && apt-get update && apt-get -y install docker-ce
```

2. NginX 설정

2-1. SSL 설정

```
sudo snap install --classic certbot | sudo certbot --nginx -d <등록할 도메인 주소>
```

2-2. 리버스 프록시 설정

1) ginx.conf에 내장할 *custom.conf 작성*

- 파일 위치 : etc/nginx/sites-available/custom.conf

```
server {
    listen 443 ssl;
    server_name 서버네임;

    ssl_certificate /etc/letsencrypt/live/서버네임/fullchain.
pem;
    ssl_certificate_key /etc/letsencrypt/live/서버네임/privke
y.pem;
    include /etc/letsencrypt/options-ssl-nginx.conf;
    ssl_dhparam /etc/letsencrypt/ssl-dhparams.pem;

    # Springboot /(${CONTEXT_PATH})
    location /${CONTEXT_PATH} {
        proxy_pass http://127.0.0.1:포트번호;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwa
rded_for;
        proxy_set_header X-Forwarded-Proto $scheme;
        proxy_set_header X-Forwarded-Host $host;
        proxy_set_header X-Real-IP $remote_addr;
    }



    # React

 location / {
        proxy_pass http://127.0.0.1:포트번호;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwa
rded_for;
        proxy_set_header X-Forwarded-Proto $scheme;
        proxy_set_header X-Forwarded-Host $host;
        proxy_set_header X-Real-IP $remote_addr;
    }
}

server {
    listen 80 ;
```

```
    listen [::]:80 ;
    server_name 서버네임;
    return 301 https://$server_name$request_uri;
}
```

2) 심볼릭 링크 설정

```
sudo ln -s /etc/nginx/sites-available/custom /etc/nginx/sit
es-enabled/custom
```

3) etc/nginx/nginx.conf 에 custom.conf 내장

```
http {
                ...
        include /etc/nginx/sites-enabled/custom.conf;
        ...
     }
```

3. MySQL/Redis/Kafka 설치

3.1 MySQL 컨테이너 생성

```
sudo docker pull mysql:latest | docker run -d --restart alw
ays -e TZ=Asia/Seoul -p 포트번호:포트번호 -e MYSQL_ROOT_PASSWO
RD=비밀번호 -v /var/lib/mysql:/var/lib/mysql --name mysql mys
ql
```

3.2 Redis 컨테이너 생성

```
sudo docker pull redis | docker run -d --restart always -e
TZ=Asia/Seoul -p 포트번호:포트번호 --name redis redis
```

3.3 Kafka, Zookeeper 컨테이너 생성

1) kafka-compose.yml

- 파일 위치 : home/ubuntu/kafka/kafka-compose.yml

```
version: '2.1'
name: kafka

services:
  zookeeper1:
    container_name: <컨테이너 이름>
    image: <zookeeper 이미지>
    ports:
      - "<포트번호>:<포트번호>"
    networks:
      - <네트워크 이름>

  zookeeper2:
    container_name: <컨테이너 이름>
    image: <zookeeper 이미지>
    ports:
      - "<포트번호>:<포트번호>"
    networks:
      - <네트워크 이름>

  zookeeper3:
    container_name: <컨테이너 이름>
    image: <zookeeper 이미지>
    ports:
      - "<포트번호>:<포트번호>"
    networks:
      - <네트워크 이름>

  kafka:
    image: <kafka 이미지>
    container_name: <컨테이너 이름>
    ports:
```

```
      - "<포트번호>:<포트번호>"
    networks:
      - <네트워크 이름>
    environment:
      KAFKA_BROKER_ID: 1
      TZ: Asia/Seoul
      KAFKA_LISTENERS: PLAINTEXT://<Kafka 컨테이너 이름>:<포트
번호>
      KAFKA_CREATE_TOPICS: "FEELINGS:1:1, OBJECTS:1:1, DRAW
INGS:1:1"
      KAFKA_ZOOKEEPER_CONNECT: <Zookeeper 컨테이너 이름>:<포트
번호>, <Zookeeper 컨테이너 이름>:<포트번호>, <Zookeeper 컨테이너
이름>:<포트번호>
      KAFKA_LISTENER_SECURITY_PROTOCOL_MAP: PLAINTEXT:PLAIN
TEXT,PLAINTEXT_HOST:PLAINTEXT
      KAFKA_INTER_BROKER_LISTENER_NAME: PLAINTEXT
      KAFKA_OFFSETS_TOPIC_REPLICATION_FACTOR: 1
      KAFKA_GROUP_INITIAL_REBALANCE_DELAY_MS: 0

    # 의존 관계 설정
    depends_on:
      - <Zookeper 컨테이너 이름>
      - <Zookeper 컨테이너 이름>
      - <Zookeper 컨테이너 이름>

networks:
  <네트워크 이름>:
    external: true
```

2) kafka, zookeeper 컨테이너 생성

```
docker-compose up -d
```

4. Backend - API 서버(Spring Boot) 배포

4.1 Spring Dockerfile

Dockerfile

```
FROM docker
COPY --from=docker/buildx-bin:latest /buildx /usr/libexec/d
ocker/cli-plugins/docker-buildx

FROM openjdk:17-jdk

ARG JAR_FILE=./build/libs/*.jar

ADD ${JAR_FILE} app.jar

ENTRYPOINT ["java","-Duser.timezone=Asia/Seoul", "-jar","/a
pp.jar"]
```

4.2 Jenkins 파이프라인 작성

```
pipeline {
    agent any

    environment {
        imageName = <이미지 이름>
        registryCredential = <docker 계정 Credential>
        dockerImage = ''

        releaseServerAccount = <서버 계정>
        releaseServerUri = <서버네임>
        releasePort = <포트번호>
    }

    stages {
        stage('Git Clone') {
            steps {
                git branch: <브랜치 명>,
                    credentialsId: <gitlab 계정 Credential>,
```

```
                    url: <gitlab url>
            }
        }

        stage('Add Env') {
           steps {
               dir('backend') {
                   withCredentials([file(credentialsId: <en
v Credential>, variable: 'env')]) {
                    sh 'cp ${env}  src/main/resources/.env'
                    }
                }
            }
        }

        stage('Jar Build') {
            steps {
                dir ('backend') {
                    sh 'chmod +x ./gradlew'
                    sh './gradlew clean bootJar'
                }
            }
        }

        // +) docker 이미지 생성
        // +) docker 컨테이너 삭제 및 생성
    }
}
```

4.3 Docker 이미지 생성

```
        ...
        stage('Image Build & DockerHub Push') {
            steps {
                dir('<경로>') {
                    script {
                        docker.withRegistry('', registryCre
```

```
dential) {
                            sh "docker buildx create --use
--name mybuilder"
                            sh "docker buildx build --platf
orm linux/amd64 -t $imageName:$BUILD_NUMBER --push ."
                            sh "docker buildx build --platf
orm linux/amd64 -t $imageName:latest --push ."
                        }
                    }
                }
            }
        }
        ...
```

4.4 Docker 컨테이너 삭제 및 생성

```
...
stage('Service Restart') {
            steps {
                sshagent(credentials: <Ubuntu 계정 Credentia
l>]) {
                    script {
                        // Stop the existing container if i
t is running
                        sh '''
                        if ssh -o StrictHostKeyChecking=no
$releaseServerAccount@$releaseServerUri "sudo docker ps -a
--filter name=<컨테이너 명> --format '{{.Names}}' | grep -q <
컨테이너 명>"; then
                            ssh -o StrictHostKeyChecking=no
$releaseServerAccount@$releaseServerUri "sudo docker stop <
컨테이너 명>"
                            ssh -o StrictHostKeyChecking=no
$releaseServerAccount@$releaseServerUri "sudo docker rm -f
<컨테이너 명>"
                        fi
                        '''
```

```
                        // Pull the latest image
                        sh "ssh -o StrictHostKeyChecking=no
$releaseServerAccount@$releaseServerUri 'sudo docker pull
$imageName:latest'"

                        // Start the new container
                        sh '''

                        // Springboot의 경우 -e 'SPRING_PROF
ILES_ACTIVE=prod' 추가
                        ssh -o StrictHostKeyChecking=no $re
leaseServerAccount@$releaseServerUri "sudo docker run -i -e
TZ=Asia/Seoul -e 'SPRING_PROFILES_ACTIVE=prod' --name <컨테
이너 명> -p $releasePort:$releasePort --network <네트워크 이름>
-d $imageName:latest"
                        '''
                    }
                }
            }
        }
        ...
```

5. Frontend - 토비의 당근밭 App 배포

5.1 React Dockerfile

Dockerfile

```
FROM nginx:latest

RUN mkdir /app

WORKDIR /app

RUN mkdir ./build

ADD ./dist ./build
```

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

EXPOSE <포트번호>

CMD ["nginx", "-g", "daemon off;"]
```

5.2 NginX 설정

nginx.conf

- 파일 위치 : ./nginx.conf

```
server {
    listen <포트번호>;
    location / {
        root    /app/build;
        index   index.html;
        try_files $uri $uri/ /index.html;
    }
}
```

5.3 Jenkins 파이프라인 작성

```
pipeline {
    agent any

    // Node.js 툴을 사용하여 노드 환경 설정
    tools {nodejs "nodejs"}

    // 환경 변수 설정
    environment {
        imageName = <이미지 이름>
        registryCredential = <docker 계정 Credential>
        dockerImage = ''
```

```
        releaseServerAccount = <서버 계정>
        releaseServerUri = <서버네임>
        releasePort = <포트번호>
    }

    // 파이프라인의 단계들
    stages {
        // Git 저장소에서 코드를 클론하는 단계
        stage('Git Clone') {
            steps {
                git branch: <브랜치 명>,
                    credentialsId: <gitlab 계정 Credential>,
                    url: <gitlab url>
            }
        }

        // 노드 프로젝트를 빌드하는 단계
        stage('Node Build') {
            steps {
                dir ('frontend/toby') {
                    sh 'npm install'
                    sh 'npm run build'
                }
            }
        }

        // +) docker 이미지 생성 - 4.3 참고
        // +) docker 컨테이너 삭제 및 생성 - 4.4 참고
    }
}
```

6. AI 서버(Python) 배포 - drawing, emotion, detection 공통

6-1. Python Dockerfile 작성

Dockerfile

```
# 라이브러리 등 환경 이미지
FROM <docker 아이디>/toby-ai-base:latest

RUN mkdir -p /opt/ai

WORKDIR /opt/ai

COPY . .

# 컨테이너에서 실행될 명령어 지정
CMD ["python", "main.py"]
```

6-2. Jenkins 파이프라인 설정

```
pipeline {
    agent any

    // 환경 변수 설정
    environment {
        imageName = <이미지 이름>
        registryCredential = <docker 계정 Credential>
        dockerImage = ''

        releaseServerAccount = <서버 계정>
        releaseServerUri = <서버네임>
    }

    // 파이프라인의 단계들
    stages {
        // Git 저장소에서 코드를 클론하는 단계
        stage('Git Clone') {
            steps {
                git branch: <브랜치 명>,
                    credentialsId: <gitlab 계정 Credential>,
                    url: <gitlab url>
```

```
            }
        }

        stage('Add Env') {
            steps {
                dir('ai/') {
                    withCredentials([file(credentialsId: <e
nv Credential>, variable: 'env')]) {
                        sh "cp \${env} <AI 분류>/.env"
                    }
                }
            }
        }
        // +) docker 이미지 생성 - 4.3 참고
        // +) docker 컨테이너 삭제 및 생성 - 4.4 참고
    }
}
```

외부 서비스 이용

카카오 로그인 API

1) 카카오 디벨로퍼스 → 내 애플리케이션 → 애플리케이션 추가

2) 생성한 애플리케이션 → 플랫폼 → 사이트 도메인 추가 후 앱 키 사용

AWS S3

1) S3 버킷 생성

2) 권한 → 버킷 정책 편집

```
{
    "Version": "2012-10-17",
    "Id": "Policy1710233832733",
    "Statement": [
        {
            "Sid": "<Sid 넘버>",
```

```
            "Effect": "Allow",
            "Principal": "*",
            "Action": [
                "s3:GetObject",
                "s3:PutObject"
            ],
            "Resource": "arn:aws:s3:::<버킷 이름>/*",
            "Condition": {
                "StringEquals": {
                    "aws:Referer": [
                        "http://<도메인 주소>",
                        "https://<도메인 주소>"
                    ]
                }
            }
        }
    ]
}
```

특이사항

- Flyway를 사용하여 DB dump 자동 생성