포팅 메뉴얼

1. 개발 환경 및 기술 스택
2. 설정 파일 및 환경 변수 정보
3. 빌드 및 배포
    1) Docker + Docker compose 설치
    2) Jenkins 컨테이너 실행
    3) docker compose를 통한 실행
4. DB

1. 개발 환경 및 기술 스택

- Backend
    - JAVA 17 (17.0.9 2023-10-17 LTS)
    - Spring boot 3.2.3
    - Lombok
    - OpenAPI (Swagger 3.0)
    - Spring Security
    - JWT
    - Spring Data JPA
    - QueryDSL
    - WebSocket
    - MySQL 8.0.36
    - Redis
    - RabbitMQ
    - Python 3.8.10
    - FastAPI 0.110.0
- Frontend

    - React 18.3.1
    - Typescript 5.2.2
    - axios
    - Recoil
    - CSS.module
    - React-router-dom 6.22.3
    - STOMP
    - WebSocket
    - Vite, ESLint, Prettier
- Infra
    - Ubuntu 20.04.6 LTS
    - Nginx
    - Jenkins 2.455
    - docker 25.0.4
- Data
    - aioredis 2.0.1
    - konlpy 0.6.0
    - pydantic 2.7.1
    - requests 2.31.0
    - uvicorn 0.28.0
    - webdriver-manager 4.0.1
    - websockets 12.0
- IDE
    - IntelliJ Ultimate 2024.1.1
    - Vscode 1.85.1

2. 설정 파일 및 환경 변수 정보

1. **Spring boot**

- application.yml

```
spring:
  jpa:
    open-in-view: false
    defer-datasource-initialization: true
    generate-ddl: true
    hibernate:
      ddl-auto: update
    properties:
      hibernate:
        format_sql: true
        use_sql_comments: true
        show_sql: true
        jdbc:
          batch_size: 100
        default_batch_fetch_size: 100
        dialect: org.hibernate.dialect.MySQLDialect
      id:
        new_generator_mappings: false

  datasource:
    # mySQL
    driver-class-name: com.mysql.cj.jdbc.Driver
    # 로컬
    #    url: jdbc:mysql://localhost:3306/weeting?useUnico
    #    username: ssafy
    #    password: ssafy
    # 배포
    url: jdbc:mysql://k10c103.p.ssafy.io:3306/weeting?useU
    username: root
    password: c103103
    # mariaDB\
#    url: jdbc:mysql://stg-yswa-kr-practice-db-master.mari
#    username: S10P31C103@stg-yswa-kr-practice-db-master.m
#    password: sF9lNwfigU
```

```
  data:
    redis:
      host: redis
#      host: k10c103.p.ssafy.io
#      host: localhost
      port: 6379
      password: c103103
    mongodb:
      uri: mongodb+srv://S10P31C103:fCTcbjQoe0@ssafy.ngivl


  rabbitmq:
#    host: localhost
    host: rabbitmq
    port: 5672
    username: guest # RabbitMQ 사용자 이름, 기본값은 guest
    password: guest # RabbitMQ 비밀번호, 기본값은 guest
    virtual-host: / # RabbitMQ 가상 호스트, 기본값은 /


# jwt
jwt:
  access: WEETINGACCESSKEYLONGENOUGHATLEAST256BITSSCRETKEY
  accessTime: 86400000 # 60?

# log
logging:
  level:
    org.hibernate:
      type.descriptor.sql: trace
      org.hibernate.SQLQuery: debug
      org.springframework.web.socket: DEBUG
```

2. **React**

- .env(로컬용)

```
# 로컬
```

```
VITE_API_URL = "http://localhost:8080/api/v1"
```

- .env-dev(배포용)

```
# 배포
VITE_API_URL = "k10c103.p.ssafy.io:8080/api/v1"
```

3. 빌드 및 배포

1) Docker + Docker compose 설치

```
#!/bin/bash

# 기존 도커 패키지 제거 (이전 버전이 설치된 경우)
sudo apt-get remove docker docker-engine docker.io containerd

# 필수 패키지 설치
sudo apt-get update
sudo apt-get install -y apt-transport-https ca-certificates c

# 도커 GPG 키 추가
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sud

# 도커 저장소 추가
echo "deb [arch=amd64 signed-by=/usr/share/keyrings/docker-ar

# 도커 설치
sudo apt-get update
sudo apt-get install -y docker-ce docker-ce-cli containerd.io

# 도커 컴포즈 최신 버전 다운로드
sudo curl -L "https://github.com/docker/compose/releases/late

# 실행 권한 부여
sudo chmod +x /usr/local/bin/docker-compose

# 도커 사용자 그룹에 현재 사용자 추가
sudo usermod -aG docker $USER
```

```
newgrp docker
sudo service docker restart

# 설치 확인
docker --version
docker-compose --version

# 권한 설정
chmod +x installDocker.sh

# 실행
./installDocker.sh
```

2) Jenkins 컨테이너 실행

```
#!/bin/bash

# Jenkins 폴더 생성
JENKINS_DIR="./jenkins"
if [ ! -d "$JENKINS_DIR" ]; then
    mkdir "$JENKINS_DIR"
fi

# Docker Compose 실행
docker-compose up -d

# Jenkins 컨테이너가 완전히 실행될 때까지 대기
sudo sleep 60

# Jenkins 폴더로 이동
cd ./jenkins

# Jenkins 폴더가 완전히 생성될 때까지 대기
sudo sleep 60

# update center에 필요한 CA 파일 다운로드
UPDATE_CENTER_DIR="./update-center-rootCAs"
if [ ! -d "$UPDATE_CENTER_DIR" ]; then
```

```
    mkdir "$UPDATE_CENTER_DIR"
fi

sudo wget https://cdn.jsdelivr.net/gh/lework/jenkins-update-c

# Jenkins 설정 파일 수정
sudo sed -i 's#https://updates.jenkins.io/update-center.json#

# Jenkins 재시작 (필수)
docker restart jenkins

# 현재 폴더 확인
pwd

# /home/ubuntu/develop/CICD 확인
chmod +x ./Install/installJenkins.sh

# 실행
./Install/installJenkins.sh
```

3) docker compose를 통한 실행

- Jenkins를 통해 docker-compose.yml파일의 구성을 실행시킨다.

docker-compose.yml

```
version: '3.8'
services:
  back:
    container_name: back
    hostname: back
    networks:
      - weeting-network
    image: ${DOCKERHUB_REGISTRY}
    depends_on:
      - redis
      - mysql
    ports:
      - "9002:8080"
```

```
    environment:
      SPRING_DATASOURCE_URL: jdbc:mysql://mysql:3306/weeting?
      SPRING_DATASOURCE_USERNAME: root
      SPRING_DATASOURCE_PASSWORD: c103103
      SPRING_REDIS_HOST: redis
      SPRING_REDIS_PORT: 6379
      SPRING_REDIS_PASSWORD: c103103
      SPRING_RABBITMQ_HOST: rabbitmq
      SPRING_RABBITMQ_PORT: 5672

  front:
    networks:
      - weeting-network
    container_name: front
    hostname: front
    build:
      context: ../FrontEnd
      dockerfile: Dockerfile
    ports:
      - "9003:3000"
    depends_on:
      - back

  redis:
    container_name: redis
    hostname: redis
    volumes:
      - /home/ubuntu/redis.conf:/usr/local/etc/redis/redis.co
    networks:
      - weeting-network
    image: redis:latest
    ports:
      - "6379:6379"
    command: ["sh", "-c", "redis-server /usr/local/etc/redis/


  mysql:
    networks:
```

```
      - weeting-network
    image: mysql:latest
    environment:
      MYSQL_DATABASE: weeting
      MYSQL_ROOT_PASSWORD: c103103
    ports:
      - "3306:3306"
    volumes:
      - mysql_data:/var/lib/mysql

  rabbitmq:
    container_name: rabbitmq
    hostname: rabbitmq
    image: rabbitmq:management
    networks:
      - weeting-network
    ports:
      - "5672:5672"      # AMQP 프로토콜 포트
      - "15672:15672"    # 관리 대시보드 포트
      - "61613:61613"    # STOMP 플러그인 포트
    environment:
      - RABBITMQ_DEFAULT_USER=guest
      - RABBITMQ_DEFAULT_PASS=guest
    volumes:
      - ./rabbitmq.conf:/etc/rabbitmq/rabbitmq.conf
    command: ["sh", "-c", "rabbitmq-plugins enable rabbitmq_s

volumes:
  mysql_data:
  rabbitmq_data:

networks:
  weeting-network:
    external: true
```

docker-compose-fastapi.yml

```
version: '2.12.2'

services:
  fastapi:
    image: weeting_fastapi_img
    volumes:
      - /home/ubuntu/develop/model:/app/model
    container_name: fastapi
    build:
      context: ./Data
      dockerfile: Dockerfile
    ports:
      - "8000:8000"
    restart: always
```

BE Jenkinsfile

```
pipeline {
    agent any
    environment {
        REPO = "s10-final/S10P31C103"
        DOCKERHUB_REGISTRY = "superjaehun/back"
        DOCKERHUB_CREDENTIALS = credentials('Docker-hub')
    }
    stages {
        stage('Checkout') {
            steps {
                checkout scm
            }
        }
        stage('Setup Environment') {
            steps {
                dir("${env.WORKSPACE}/BackEnd") {
                    script {
                        sh "ls -al"
                        sh "chmod +x ./gradlew"
                        // docker-compose가 설치되어 있는지 확인하
                        sh '''
```

```
                    if ! command -v docker-compose &> /de
                    then
                        echo "docker-compose not found, i
                        curl -L "https://github.com/docke
                        chmod +x /usr/local/bin/docker-co
                    else
                        echo "docker-compose is already i
                    fi
                    '''
                }
            }
        }
    }
    stage('Create .env file for Frontend') {
        steps {
            dir("${env.WORKSPACE}/FrontEnd") {
                sh "echo 'VITE_API_URL=https://k10c103.p.
            }
        }
    }
    stage('Stop and Remove Existing Containers') {
        steps {
            script {
                sh "docker-compose -f ${env.WORKSPACE}/Ba
            }
        }
    }
    stage('Remove Old Images') {
        steps {
            script {
                sh "docker images -f 'dangling=true' -q |
            }
        }
    }
    stage("Build with Docker Compose") {
        steps {
            script {
                dir("${env.WORKSPACE}/BackEnd") {
```

```
                        sh "docker-compose -f docker-compose.
                    }
                }
            }
        }
        stage("Login to Docker Hub") {
            steps {
                script {
                    sh "echo \${DOCKERHUB_CREDENTIALS_PSW} |
                }
            }
        }
        stage("Push to Docker Hub") {
            steps {
                script {
                    withCredentials([[$class: 'UsernamePasswo
                        sh "docker push ${DOCKERHUB_REGISTRY}
                    }
                }
            }
        }
        stage('Deploy with Docker Compose') {
            steps {
                script {
                    withCredentials([[$class: 'UsernamePasswo
                        dir("${env.WORKSPACE}/BackEnd") {
                            sh "docker-compose -f docker-comp
                        }
                    }
                }
            }
        }
    }

    post {
        success {
            script {
                def Author_ID = sh(script: "git show -s --pre
```

```
                def Author_Name = sh(script: "git show -s --p
                mattermostSend (color: 'good',
                message: "BE 빌드 성공: ${env.JOB_NAME} #${env.
                endpoint: 'https://meeting.ssafy.com/hooks/78
                channel: 'C103-Jenkins'
                )
            }
        }
        failure {
            script {
                def Author_ID = sh(script: "git show -s --pre
                def Author_Name = sh(script: "git show -s --p
                mattermostSend (color: 'danger',
                message: "BE 빌드 실패: ${env.JOB_NAME} #${env.
                endpoint: 'https://meeting.ssafy.com/hooks/78
                channel: 'C103-Jenkins'
                )
            }
        }
    }
}


// pipeline {
//     agent any
//     environment {
//         REPO = "s10-final/S10P31C103"
//         DOCKERHUB_REGISTRY = "superjaehun/back"
//         DOCKERHUB_CREDENTIALS = credentials('Docker-hub')
//     }
//     stages {
//         stage('Checkout') {
//             steps {
//                 checkout scm
//             }
//         }
//         stage('Setup Environment') {
//             steps {
```

```
//                 dir("${env.WORKSPACE}/BackEnd") {
//                     script {
//                         sh "ls -al"
//                         sh "chmod +x ./gradlew"
//                     }
//                 }
//             }
//         }
//         stage('Stop and Remove Existing Containers') {
//             steps {
//                 script {
//                     sh "docker-compose -f docker-compose.y
//                 }
//             }
//         }
//         stage('Remove Old Images') {
//             steps {
//                 script {
//                     sh "docker images -f 'dangling=true' -
//                 }
//             }
//         }
//         stage("Build with Docker Compose") {
//             steps {
//                 script {
//                     sh "/usr/local/bin/docker-compose -f d

//                 }
//             }
//         }
//         stage("Login to Docker Hub") {
//             steps {
//                 script {
//                     sh "echo \${DOCKERHUB_CREDENTIALS_PSW}
//                 }
//             }
//         }
//         stage("Tag and Push") {
```

```
//             steps {
//                 script {
//                     withCredentials([[$class: 'UsernamePas
//                         sh "docker-compose -f docker-compo
//                     }
//                 }
//             }
//         }
//         stage('Deploy with Docker Compose') {
//             steps {
//                 script {
//                     withCredentials([[$class: 'UsernamePas
//                         sh "docker rmi ${DOCKERHUB_REGISTR
//                         sh "docker pull ${DOCKERHUB_REGIST
//                     }
//                 }
//             }
//         }
//         stage('Up') {
//             steps {
//                 script {
//                     withCredentials([[$class: 'UsernamePas
//                         dir("BackEnd") {
//                             try {
//                                 sh "docker-compose up -d"
//                             } catch(Exception e) {
//                                 sh "docker-compose up -d -
//                             }
//                         }
//                     }
//                 }
//             }
//         }
//     }
//     post {
//         success {
//             script {
//                 def Author_ID = sh(script: "git show -s --
```

```
//                 def Author_Name = sh(script: "git show -s
//                 mattermostSend (color: 'good',
//                 message: "BE 빌드 성공: ${env.JOB_NAME} #${e
//                 endpoint: 'https://meeting.ssafy.com/hooks
//                 channel: 'C103-Jenkins'
//                 )
//             }
//         }
//         failure {
//             script {
//                 def Author_ID = sh(script: "git show -s --
//                 def Author_Name = sh(script: "git show -s
//                 mattermostSend (color: 'danger',
//                 message: "BE 빌드 실패: ${env.JOB_NAME} #${e
//                 endpoint: 'https://meeting.ssafy.com/hooks
//                 channel: 'C103-Jenkins'
//                 )
//             }
//         }
//     }
// }
```

FastAPI Jenkinsfile

```
pipeline {
    agent any

    stages {
        stage('Deploy with Docker Compose') {
            steps {
                script {
                    // 이전 실행에서 사용된 컨테이너 및 네트워크 정리
                    sh "docker-compose -f docker-compose-fast

                    // 새로운 푸시에 대한 스크립트 실행
                    sh "docker-compose -f docker-compose-fast

                    // FastAPI 재실행
```

```
                    sh "docker restart fastapi"
                }
            }
        }
    }
}
```

BE Dockerfile

```
# 빌드 스테이지
FROM gradle:8.5.0-jdk17-alpine AS build
USER root
WORKDIR /back
COPY gradlew .
COPY gradle gradle
COPY build.gradle .
COPY settings.gradle .
COPY src src
# gradlew 실행 권한 부여
RUN chmod +x ./gradlew
RUN ./gradlew bootJar

# 실행 스테이지
FROM azul/zulu-openjdk:17

WORKDIR /back

# 빌드된 JAR 파일 복사
COPY --from=builder /back/build/libs/*.jar app.jar

# 애플리케이션 실행
ENTRYPOINT ["java", "-Duser.timezone=Asia/Seoul", "-jar", "ap

# 볼륨 설정
VOLUME /tmp
```

FastAPI Dockerfile

```
FROM ubuntu:22.04

WORKDIR /app

# 필요한 패키지 설치
RUN apt-get update && \
    apt-get install -y git openjdk-11-jdk python3 python3-pip
    apt-get clean && \
    rm -rf /var/lib/apt/lists/*

# JAVA_HOME 환경 변수 설정
ENV JAVA_HOME /usr/lib/jvm/java-11-openjdk-amd64
ENV PATH $JAVA_HOME/bin:$PATH

# Python 패키지 및 PyKoSpacing 설치
COPY requirements.txt .
RUN pip3 install --no-cache-dir -r requirements.txt && \
    pip3 install git+https://github.com/allinux/PyKoSpacing.g

COPY . .

EXPOSE 8000

ENV FASTAPI_ENV production

CMD ["uvicorn", "main:app", "--host", "0.0.0.0", "--port", "8
```

FE Nginx 멀티 스테이지 빌드(Multi-Stage Build) Dockerfile

```
# 1단계: 빌드 단계 (최신 버전 사용)
FROM node:20.12.2 as build

# 작업 디렉토리 설정
WORKDIR /app

# 의존성 파일 복사
COPY package*.json ./
```

```
# 의존성 설치 전에 package-lock.json과 node_modules 제거
RUN rm -rf node_modules package-lock.json && npm cache clean

# 의존성 설치
RUN npm install

# 앱 소스 코드 추가
COPY . .

# # 앱 빌드
# RUN npm run build

# 2단계: 실행 단계
# FROM nginx:alpine

# Nginx
# COPY --from=build /app/dist /usr/share/nginx/html
# EXPOSE 80
# CMD ["nginx", "-g", "daemon off;"]

# 포트 노출
EXPOSE 3000

# 컨테이너 시작 시 실행할 명령
CMD ["npm", "run", "dev"]
```

4. DB

- 개발 단계에서 JPA를 이용하여 테이블들을 자동으로 DB와 매핑하여 Spring boot 서버 내에서 자동으로 생성되게 설정하였습니다.
- Redis 기반 채팅 서비스가 메인 서비스이기에 별도의 DB dump 파일은 존재하지 않습니다.