포팅매뉴얼

1. 프로젝트 개요

- 어린이 정서 발달을 위한 소통형 그림 일기 앱

2. 사용 도구

- 이슈 관리: JIRA
- IDE : Visual Studio, Intellij Ultimate, Unity Editor
- 형상 관리: Gitlab
- 커뮤니케이션: Notion, Mattermost
- 디자인: Figma
- CI/CD: Jenkins

3. 개발 도구

Frontend

- 프레임워크 : Kotlin

Backend

- 프레임워크 : Spring Boot, Fast API
- 라이브러리 : JPA, Spring Security, JWT, Tensorflow, Numpy, OpenCV
- DB: MySQL

4. 개발 환경

Frontend

Node.js	20.15.0
Kotlin	18.3.1
Unity	2022.3.44f1

Backend

Java	17
Spring Boot	3.3.3
Mysql	8.0.38
Fast API	0.111.1
numpy	1.26.4
requests	2.32.3
tensorflow	2.17.0
uvicorn	0.30.4

Infra

Docker	27.1.1
Nginx	nginx/1.18.0 (Ubuntu)
Jenkins	2.468
AWS EC2	
AWS S3	
Ubuntu	22.43 LTS

5. 환경 변수

Spirng Boot

- 외부 라이브러리

```
plugins {
    id 'java'
```

```
    id 'org.springframework.boot' version '3.3.3'
    id 'io.spring.dependency-management' version '1.1.6'
}

group = 'com.ssafy'
version = '0.0.1-SNAPSHOT'

java {
    toolchain {
        languageVersion = JavaLanguageVersion.of(17)
    }
}

configurations {
    compileOnly {
        extendsFrom annotationProcessor
    }
}

repositories {
    mavenCentral()
}

dependencies {
// 스프링부트 의존성
    implementation 'org.springframework.boot:spring-boot-starter
    implementation 'org.springframework.boot:spring-boot-starter
    implementation 'org.springframework.boot:spring-boot-starter
    implementation 'org.springframework.boot:spring-boot-starter
    developmentOnly 'org.springframework.boot:spring-boot-devtoo

//  SQL 로그에 값을 주입해주는 의존성
    implementation 'com.github.gavlyukovskiy:p6spy-spring-boot-s

//  JWT
    implementation 'io.jsonwebtoken:jjwt-api:0.11.5'
```

```
    implementation 'io.jsonwebtoken:jjwt-impl:0.11.5'
    implementation 'io.jsonwebtoken:jjwt-jackson:0.11.5'

//  유틸리티
    compileOnly 'org.projectlombok:lombok'
    annotationProcessor 'org.projectlombok:lombok'

//  MySQL
    implementation 'com.mysql:mysql-connector-j'
    runtimeOnly 'com.mysql:mysql-connector-j'

//  AWS S3
    implementation 'org.springframework.cloud:spring-cloud-start

// https://mvnrepository.com/artifact/net.coobird/thumbnailator
    implementation group: 'net.coobird', name: 'thumbnailator',

//Webflux
    implementation 'org.springframework.boot:spring-boot-starter

//  Test 의존성
    testImplementation 'org.springframework.boot:spring-boot-sta
    testRuntimeOnly 'org.junit.platform:junit-platform-launcher
}

tasks.named('test') {
    useJUnitPlatform()
}
```

- application.yml

```
spring:
  servlet:
    multipart:
      max-file-size: 50MB
```

```
      max-request-size: 50MB
  jpa:
    database: mysql
    database-platform: org.hibernate.dialect.MySQL8Dialect
    open-in-view: false
    show_sql: true
    properties:
      hibernate:
        storage_engine: innodb
        format_sql: true
        use_sql_comments: true
```

- application-local.yml

```
spring:
  config:
    activate:
      on-profile: local
  datasource:
    driver-class-name: com.mysql.cj.jdbc.Driver
    url: "jdbc:mysql://localhost:3306/heroin?autoReconnect=true&
    username:
    password:
  jpa:
    hibernate:
      ddl-auto: create # 배포 환경에서는 DB 자동 생성 금지
jwt:
  secret:
cloud:
  aws:
    s3:
      bucket: heroinbucket
    credentials:
      access-key:
      secret-key:
```

```
    region:
      static: ap-northeast-2
      auto: false
    stack:
      auto: false
kakao:
  client_id: 11774766665658ac5b26817169b19af4
  redirect_uri: http://localhost:8080/callback
```

6. 배포 설정

AWS

- 포트 번호

MySQL	3306	Pipeline 미포함
Jenkins	8080	Pipeline 미포함
Nginx	80, 443	Pipeline 미포함
Fast api	8000	Pipeline 미포함
React	3000	Pipeline 포함
Spring Boot	8081	Pipeline 포함

Jenkins - front

```
pipeline {
    agent any

    environment {
        GIT_CREDENTIALS_ID = 'GitLab-ID'
        DOCKER_IMAGE = 'react'
        REGISTRY = 'index.docker.io'
        DOCKER_CREDENTIALS_ID = 'dockerhub-credentials'
        REPO_NAMESPACE = 'qkrdusgn00'
        GIT_REPO_URL = 'lab.ssafy.com/s11-ai-image-sub1/S11P21E1
```

```
        GIT_BRANCH = 'develop-FE'
        CONTAINER_NAME = 'react-container'
    }

    stages {
        stage('Checkout') {
            steps {
                withCredentials([usernamePassword(credentialsId
                    script {
                        // Git 안전 디렉토리 설정
                        sh 'git config --global --add safe.direc

                        // 디렉토리 삭제 및 Git 클론 수행
                        if (fileExists('S11P21E101')) {
                            echo "Directory exists. Removing and
                            sh 'rm -rf S11P21E101'  // 기존 디렉토
                        }

                        // 새로운 Git 클론 수행
                        sh 'git clone https://${GIT_USERNAME}:$
                        sh 'cd S11P21E101 && git checkout ${GIT_
                    }
                }
            }
        }

        stage('Build Docker Image') {
            steps {
                script {
                    // Docker 이미지 빌드
                    sh 'docker build -t ${REGISTRY}/${REPO_NAMES
                }
            }
        }

        stage('Push Docker Image') {
```

```
            steps {
                withCredentials([usernamePassword(credentialsId
                    script {
                        // Docker Hub 로그인 및 이미지 푸시
                        sh 'echo ${DOCKER_PASSWORD} | docker log
                        sh 'docker push ${REGISTRY}/${REPO_NAMES
                    }
                }
            }
        }

        stage('Stop and Deploy New Container') {
            steps {
                script {
                    // 기존 컨테이너 중지 및 제거 후 새 컨테이너 실행
                    sh '''
                    CONTAINER_ID=$(docker ps -aq -f name=${CONTA
                    if [ ! -z "$CONTAINER_ID" ]; then
                        echo "Stopping and removing existing con
                        docker stop ${CONTAINER_NAME} || true
                        docker rm ${CONTAINER_NAME} || true
                    fi
                    echo "Starting new container..."
                    docker run -d --name ${CONTAINER_NAME} -p 30
                    '''
                }
            }
        }

        stage('Set File Permissions') {
            steps {
                script {
                    // 컨테이너 내부에서 chmod 명령 실행
                    sh '''
                    echo "Setting file permissions inside the co
                    docker exec ${CONTAINER_NAME} chmod -R 755 /
```

```
                    '''
                }
            }
        }
    }

    post {
        failure {
            echo 'Build or deployment failed.'
        }
    }
}
```

Jenkins - Spring Boot

```
pipeline {
    agent any

    environment {
        GIT_CREDENTIALS_ID = 'GitLab-ID'
        DOCKER_IMAGE = 'heroin'
        REGISTRY = 'index.docker.io'
        DOCKER_CREDENTIALS_ID = 'dockerhub-credentials'
        REPO_NAMESPACE = 'qkrdusgn00'
        GIT_REPO_URL = 'https://lab.ssafy.com/s11-ai-image-su
b1/S11P21E101.git'
        GIT_BRANCH = 'develop-BE'
        CONTAINER_NAME = 'heroin-container'
        HOST_PORT = '8081'
        CONTAINER_PORT = '8080'
        WORK_DIR="${WORKSPACE}/S11P21E101/back/server/Heroin"
    }

    stages {
```

```
        stage('Checkout') {
            steps {
                withCredentials([usernamePassword(credentials
Id: "${GIT_CREDENTIALS_ID}", passwordVariable: 'GIT_PASSWOR
D', usernameVariable: 'GIT_USERNAME')]) {
                    sh '''
                        git config --global --add safe.direct
ory ${WORKSPACE}
                        if [ -d "S11P21E101" ]; then
                            cd S11P21E101/back/heroin
                            git reset --hard
                            git clean -fd
                            git pull origin ${GIT_BRANCH}
                        else
                            git clone https://${GIT_USERNAM
E}:${GIT_PASSWORD}@${GIT_REPO_URL}
                            cd S11P21E101/back/heroin
                            git checkout ${GIT_BRANCH}
                        fi
                    '''
                }
            }
        }

        stage('DB Setting') {
            steps {
                sh '''
                    sed -i "s|{ DB_URL }|${DB_URL}|" "${WORK_
DIR}/src/main/resources/application-prod.yml"
                    sed -i "s|{ DB_USER }|${DB_USER}|" "${WOR
K_DIR}/src/main/resources/application-prod.yml"
                    sed -i "s|{ DB_PWD }|${DB_PWD}|" "${WORK_
DIR}/src/main/resources/application-prod.yml"
                    sed -i "s|{ JWT_SECRET }|${JWT_SECRET}|"
"${WORK_DIR}/src/main/resources/application-prod.yml"
                    sed -i "s|{ S3_BUCKET }|${S3_BUCKET}|"
```

```
"${WORK_DIR}/src/main/resources/application-prod.yml"
                    sed -i "s|{ S3_ACCESS }|${S3_ACCESS}|"
"${WORK_DIR}/src/main/resources/application-prod.yml"
                    sed -i "s|{ S3_SECRET }|${S3_SECRET}|"
"${WORK_DIR}/src/main/resources/application-prod.yml"
                    cat "${WORK_DIR}/src/main/resources/appli
cation-prod.yml"
                '''
            }
        }


        stage('Build Application') {
            steps {
                withCredentials([usernamePassword(credentials
Id: 'DB', usernameVariable: 'DB_USERNAME', passwordVariable:
'DB_PASSWORD')]) {
                    sh '''
                        cd ${WORK_DIR}
                        chmod +x gradlew
                        ./gradlew clean build -x test
                        echo "DB Username: ${DB_USERNAME}"
                        echo "DB Password: ${DB_PASSWORD}"
                    '''
                }
            }
        }

        stage('Build Docker Image') {
            steps {
                sh """
                    docker build -t ${REGISTRY}/${REPO_NAMESP
ACE}/${DOCKER_IMAGE}:latest ${WORK_DIR}
                """
            }
        }
```

```
        stage('Push Docker Image') {
            steps {
                withCredentials([usernamePassword(credentials
Id: "${DOCKER_CREDENTIALS_ID}", passwordVariable: 'DOCKER_PAS
SWORD', usernameVariable: 'DOCKER_USERNAME')]) {
                    sh """
                        echo \${DOCKER_PASSWORD} | docker log
in -u \${DOCKER_USERNAME} --password-stdin ${REGISTRY}
                        docker push ${REGISTRY}/${REPO_NAMESP
ACE}/${DOCKER_IMAGE}:latest
                    """
                }
            }
        }

        stage('Stop and Deploy New Container') {
            steps {
                sh """
                    CONTAINER_ID=\$(docker ps -aq -f name=${C
ONTAINER_NAME})
                    if [ ! -z "\$CONTAINER_ID" ]; then
                        echo "Stopping and removing existing
container..."
                        docker stop ${CONTAINER_NAME} || true
                        docker rm ${CONTAINER_NAME} || true
                    fi
                    echo "Starting new container..."
                    docker run -d --name ${CONTAINER_NAME} -p
${HOST_PORT}:${CONTAINER_PORT} ${REGISTRY}/${REPO_NAMESPACE}/
${DOCKER_IMAGE}:latest
                """
            }
        }
    }
```

```
    post {
        failure {
            echo 'Build or deployment failed.'
        }
        success {
            echo 'Build and deployment succeeded.'
        }
    }
}
```

Nginx

```
server {
        root /var/www/html;

    # Add index.php to the list if you are using PHP
    index index.html index.htm index.nginx-debian.html;
    server_name j11e101.p.ssafy.io; # managed by Certbot


    location / {
                # First attempt to serve request as file, then
                # as directory, then fall back to displaying a 4
                #try_files $uri $uri/ =404;
                proxy_pass http://localhost:3000;
                proxy_set_header Host $host;
                proxy_set_header X-Real-IP $remote_addr;
                proxy_set_header X-Forwarded-For $proxy_add_x_fo
                proxy_set_header X-Forwarded-Proto $scheme;
        }

   location /api {
                # First attempt to serve request as file, then
                # as directory, then fall back to displaying a 4
                #try_files $uri $uri/ =404;
```

```
                proxy_pass http://localhost:8081;
                proxy_set_header Host $host;
                proxy_set_header X-Real-IP $remote_addr;
                proxy_set_header X-Forwarded-For $proxy_add_x_fo
                proxy_set_header X-Forwarded-Proto $scheme;
        }

        location /downloads/ {
                alias /var/www/files/;
                autoindex on;  # 파일 리스트를 보여줌
                autoindex_exact_size off;  # 파일 사이즈를 human-r
                autoindex_localtime on;  # 파일의 로컬 시간을 표시
                add_header 'Access-Control-Allow-Origin' '*';
        }

    listen [::]:443 ssl ipv6only=on; # managed by Certbot
    listen 443 ssl; # managed by Certbot
    ssl_certificate /etc/letsencrypt/live/j11e101.p.ssafy.io/ful
    ssl_certificate_key /etc/letsencrypt/live/j11e101.p.ssafy.io
    include /etc/letsencrypt/options-ssl-nginx.conf; # managed b
    ssl_dhparam /etc/letsencrypt/ssl-dhparams.pem; # managed by

}

server {
    if ($host = j11e101.p.ssafy.io) {
        return 301 https://$host$request_uri;
    } # managed by Certbot


        listen 80 ;
        listen [::]:80 ;
    server_name j11e101.p.ssafy.io;
    return 404; # managed by Certbot
```

```
}
```

Dockerfile

- Frontend

```
FROM node:20

WORKDIR /app

COPY package*.json ./

RUN npm install

COPY . .

RUN npm run build
#RUN ls -la /app
FROM nginx:stable-alpine

COPY --from=0 /app/build /usr/share/nginx/html

RUN ["rm", "/etc/nginx/conf.d/default.conf"]

COPY ./default.conf /etc/nginx/conf.d

EXPOSE 80

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

- backend

```
# 1단계: Gradle과 JDK 17을 사용하여 빌드
FROM gradle:8.3-jdk17 AS build
WORKDIR /app

# Gradle 캐싱을 위한 설정
COPY build.gradle settings.gradle ./
RUN gradle build --no-daemon || true

# 애플리케이션 소스를 복사하고 빌드
COPY src ./src
RUN gradle clean build --no-daemon -x test

# 빌드된 JAR 파일 목록 확인
RUN ls -l /app/build/libs/

# 2단계: 최종 실행 이미지 (JRE 17만 포함, 필요 시 JDK로 변경)
FROM openjdk:17-jdk-slim
WORKDIR /app

# 빌드된 jar 파일을 복사 (정확한 파일명 사용 권장)
COPY --from=build /app/build/libs/*.jar app.jar

# 애플리케이션 실행 명령에 Spring 프로파일 적용
ENTRYPOINT ["java", "-jar", "-Dspring.profiles.active=prod", "ap
```

7. 설치방법

Docker

```
# 기존의 docker 관련 engine 제거
sudo apt-get remove docker docker-engine docker.io containerd ru

# 패키지 설치
sudo apt-get update
```

```
sudo apt-get install ca-certificates curl gnupg lsb-release

# Docker 공식 GPG 키 추가
sudo mkdir -p /etc/apt/keyrings
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo g

# Docker 저장소 설정
echo \
"deb [arch=$(dpkg --print-architecture) signed-by=/etc/apt/keyri
$(lsb_release -cs) stable" | sudo tee /etc/apt/sources.list.d/do

# Docker 설치
sudo apt-get update
sudo apt-get install docker-ce docker-ce-cli containerd.io docke

# Docker 루트 권한 없이 실행
sudo usermod -aG docker $USER
newgrp docker

# Docker 시작
sudo systemctl start docker
sudo systemctl enable docker
```

Mysql

```
docker run -d \
  --name mysql \
  -v /home/ubuntu/my.cnf:/etc/mysql/conf.d/my.cnf \
  -e MYSQL_ROOT_PASSWORD=goldfunnywowjohnjoyspeed \
  -e MYSQL_DATABASE=mydb \
  -e MYSQL_USER=lte \
  -e MYSQL_PASSWORD=goldfunnywowjohnjoyspeed \
```

```
  -p 3306:3306 \
  mysql
```

Jenkins

```
# Jenkins 실행
docker run -d \
  --name jenkins \
  -p 8080:8080 \
  -p 50000:50000 \
  -v jenkins_home:/var/jenkins_home \
  -v /var/run/docker.sock:/var/run/docker.sock \
  --user root \
  jenkins/jenkins:lts
```

Nginx

```
# 시스템 패키지 업데이트
sudo apt-get update

# Nginx 설치
sudo apt-get install nginx -y

# Nginx 시작 및 부팅 시 자동 시작 설정
sudo systemctl start nginx
sudo systemctl enable nginx

# SSL 인증서 설정
sudo apt-get install certbot python3-certbot-nginx -y

# SSL 인증서 발급 및 Nginx 설정 자동화
sudo certbot --nginx
```

```
# Nginx 설정 확인
sudo nano /etc/nginx/sites-available/default

# Nginx 재시작
sudo systemctl restart nginx
```