포팅 매뉴얼

1. 개발환경

1.1 Frontend

- flutter 3.22.2
    - lottie 2.7.0
    - http 0.13.6
    - provider 6.0.0
    - shared_preferences 2.2.3
    - flutter_webrtc 0.10.6
    - permission_handler 10.2.0

    - openvidu_flutter
    - flutter_native_splash 2.0.2
    - firebase_messaging 15.0.4
    - firebase_core 3.3.0

1.2 Backend

- Java
    - JDK 17
    - Spring Boot 3.3.2
    - Gradle 8.8
- Python
    - python 3.9.12
    - FastAPI 0.111.1
    - Deepface 0.0.92
    - pydub

1.3 Server

- Ubuntu 20.04 LTS
- Nginx 1.25.2
- Docker 26.1.4
- Jenkins 2.470

1.4 Database

- MySQL 8.0.33

1.5 UI/UX

- Figma
- shots

1.6 형상/이슈 관리

- Gitlab
- Jira

2. 환경변수

2.1 Backend

```
# DB
SPRING_DATASOURCE_URL
SPRING_DATASOURCE_USERNAME
SPRING_DATASOURCE_PASSWORD
SPRING_DATASOURCE_DRIVER-CLASS-NAME

# JWT
JWT_SECRET

# OpenAI API
OPENAI_API-KEY
```

3. EC2 세팅

3.1 도커 설치

```
# 1. 시스템 패키지 목록 업데이트
sudo apt update

# 2. Docker 엔진 설치
sudo apt install docker.io

# 3. Docker Compose 설치
sudo apt install docker-compose

# 4. Docker의 버전 확인
docker --version

# 5. 사용자가 Docker 명령어를 실행할 때 권한 문제를 피하기 위해 Docker 그룹에 사용자 추가
sudo usermod -aG docker $USER

# 6. 새 그룹의 권한을 적용하기 위해 현재 터미널 세션을 새로운 그룹으로 변경
sudo newgrp docker

# 7. Docker 서비스 재시작
sudo systemctl restart docker
```

3.2 Nginx 설치 및 세팅

```
# 1. nginx 도커 이미지 설치
docker pull nginx

# 2. certbot 설치
sudo apt-get install certbot
```

```
# 3. 인증서 발급
sudo certbot certonly --standalone -d i11b107.p.ssafy.io

# 4. 인증서가 저장된곳을 컨테이너 내부와 매핑
docker run -d \
  --name nginx \
  -p 80:80 \
  -p 443:443 \
  -v /etc/letsencrypt:/etc/letsencrypt:ro \
  nginx

# 5. nginx script 작성 (reverse proxy + SSL)
server {
    listen 443 ssl;
    server_name i11b107.p.ssafy.io;
    ssl_certificate /etc/letsencrypt/live/i11b107.p.ssafy.io/fullchain.pem;
    ssl_certificate_key /etc/letsencrypt/live/i11b107.p.ssafy.io/privkey.pem;

    root /usr/share/nginx/html;
    index index.html;

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

    location /api {
        proxy_pass http://i11b107.p.ssafy.io:8080;
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header X-Forwarded-Proto $scheme;
    }

    location /jenkins {
        proxy_pass http://i11b107.p.ssafy.io:9000;
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header X-Forwarded-Proto $scheme;

        proxy_set_header X-Forwarded-Host $host;
        proxy_set_header X-Forwarded-Server $host;
        proxy_set_header X-Forwarded-Port $server_port;

        proxy_http_version 1.1;
        proxy_set_header Upgrade $http_upgrade;
        proxy_set_header Connection "upgrade";
```

```
        proxy_redirect http://i11b107.p.ssafy.io:9000 https://i11b107.p.ssafy
    }
}

server {
    listen 80;
    server_name i11b107.p.ssafy.io;
    return 301 https://$host$request_uri;
}

# 6. nginx 재실행
docker restart nginx
```

3.3 EC2 포트 정리

Port 번호	항목
22	SSH
80	HTTP
443	HTTPS
3306	MySQL (Docker)
5000	FastAPI (Docker)
8080	Spring Boot (Docker)
9000	Jenkins (Docker)

4. 빌드 및 배포

4.1 Flutter

```
# 1. 플러터 루트 디렉토리에서 apk 빌드 (터미널)
flutter build apk
```

4.2 Spring Boot

```
# 1. Spring 프로젝트의 루트 디렉토리에서 Dockerfile 작성
FROM openjdk:17-jdk
LABEL maintainer="email"
ARG JAR_FILE=build/libs/daylog-0.0.1-SNAPSHOT.jar
ADD ${JAR_FILE} docker-springboot.jar
ENTRYPOINT ["java","-Djava.security.egd=file:/dev/./urandom","-jar","/docker-

# 2. Gradle로 JAR파일 빌드 (터미널)
./gradlew clean build
```

```
# 3. 도커 이미지 생성
docker build -t daylog .

# 4. 도커 허브 레지스트레에 푸시
docker tag daylog dockerhub-username/daylog:latest

# 5. 도커 이미지 가져오기
docker pull dockerhub-username/daylog:latest

# 6. 도커 컨테이너 실행
docker run -v /mnt:/mnt/ -p 8080:8080 dockerhub-username/daylog:latest
```

4.3 FastAPI

4.3.1 requirment.txt

```
tensorflow==2.16.2
fastapi
uvicorn
opencv-python-headless
numpy
deepface
pydub
requests
tf-keras
python-multipart
```

4.3.2 Dockerfile

```
# Docker Hub에서 공식 Python 이미지를 사용합니다.
FROM python:3.9-slim

# 작업 디렉토리를 설정합니다.
WORKDIR /app

# requirements.txt 파일을 컨테이너에 복사합니다.
COPY requirements.txt .

# 종속성을 설치합니다. 여기서 tensorflow 2.16.2가 설치됩니다.
RUN pip install --no-cache-dir -r requirements.txt

# 나머지 코드를 컨테이너에 복사합니다.
COPY . .

# DeepFace 종속성을 위해 ffmpeg를 설치하고 캐시를 정리합니다.
RUN apt-get update && \
```

```
    apt-get install -y ffmpeg && \
    apt-get clean && \
    rm -rf /var/lib/apt/lists/*

# 필요한 경우 데이터 디렉토리를 생성합니다.
RUN mkdir -p /app/data

# 애플리케이션을 실행하기 위한 명령어를 설정합니다.
CMD ["uvicorn", "FastAPI-server:app", "--host", "0.0.0.0", "--port", "8000"]
```

4.3.3 빌드 과정

```
#2. 해당 도커파일로 빌드하기.
docker build -t fastapi-app .

#3. 로그인 + 태그 달기
docker login
docker tag daylog-fastapi-server jihunhan/daylog-fastapi-server:latest

#4. 도커 허브로 푸시하기.
docker push  jihunhan/daylog-fastapi-server

#5. EC2 서버에서 Pull 받기
docker pull jihunhan/daylog-fastapi-server:latest

#6. 컨테이너 실행
docker run -d -p 8000:8000 --name ai-server jihunhan/ai-server:latest
```

5. DB (MySQL)

```
# 1. MySQL 설치 (docker)
docker pull mysql

# 2. MySQL 이미지 실행
docker run 컨테이너이름 -e MYSQL_ROOT_PASSWORD=비밀번호 -p 3306:3306 -d mysql

# 3. MySQL 컨테이너에 접속
docker exec -it 컨테이너이름 mysql -u root -p

# 4. MySQL에 접속 후 데이터베이스 생성
CREATE DATABASE daylog;

# 5. 생성된 데이터베이스 확인
SHOW DATABASES;
```

6. CI/CD 구축

6.1 Jenkins 설치

```
# 1. 젠킨스 이미지 다운로드 (docker)
docker pull jenkins/jenkins:jdk17

# 2. 젠킨스 컨테이너 실행
docker run -d \
  --name jenkins \
  -p 9000:8080 \
  -p 50000:50000 \
  -v jenkins_home:/var/jenkins_home \
  -v /var/run/docker.sock:/var/run/docker.sock \
  -e JENKINS_OPTS="--prefix=/jenkins" \
  -e JENKINS_ARGS="--prefix=/jenkins" \
  -e JENKINS_HOME="/var/jenkins_home" \
  -e JAVA_OPTS="-Djenkins.install.runSetupWizard=false" \
  --network mynetwork \
  jenkins/jenkins:jdk17

# 3. 젠킨스 설정
http://ec2주소:9000 접속 후 계정과 기본 플러그인 설치
```

6.2 도커 명령어를 사용하기 한 CLI도 설치

```
docker exec -it --user root jenkins /bin/bash

apt-get update
apt-get install -y [docker.io](http://docker.io/)
```

6.3 Gitlab Credentials 발급

- gitlab의 Personal Access Token, Repository Access Token 발급 후 등록
- gitLab API Token 유형으로 등록 ID: 식별하기 좋은 이름 API Token: GitLab에서 발급받은 토큰
- gitlab Repo Token은 반드시 Username with Password로
- Username ⇒ Gitlab 이메일 (wlgnsdl12334@gmail.com)
- Password ⇒ Gitlab Repo Token
- ID ⇒ 식별하기 쉬운 이름

6.4 Dockar Hub Credentails 등록

- Username with password 유형으로 등록
- Username: 도커 허브 이메일(wlgnsdl12334@gmail.com)
- Password: 도커 허브 비밀번호 혹은 API 토큰

6.5 Gitlab과 Jenkins 연결

- Jenkins의 gitLab 부분에 가서 gitlab과 연결한다.
- Connection name ⇒ 아무거나 ⇒ daylog (프로젝트 이름)
- GitLab Host URL ⇒ 우리가 현재 작업하고 있는 도메인 사이트 ⇒ https://lab.ssafy.com
- Credentials ⇒ 아까 우리가 GitLab API Token을 등록한 Credentials 등록
- 후에 Test Connection으로 테스트. success라 뜨면 끝.

6.6 Jenkins에 SSH Key 등록

- Jenkins와 원격 서버 간의 SSH 연결을 하기 위하여 EC2 서버의 SSH 키를 Jenkins에 등록한다. 그러기 위해서는 SSH 키 쌍(비공개 키와 공개 키)을 생성한다. ssh-keygen #나오는 모든 입력값은 Enter, 그러면 비공개 키와 공개 키가 ~/.ssh에 생성된다
- **비공개 키 (Private Key)**: 일반적으로 `~/.ssh/id_rsa`라는 파일명으로 저장. 공유 X
- **공개 키 (Public Key)**: 일반적으로 `~/.ssh/id_rsa.pub`라는 파일명으로 저장. 공유 O
- 이제 Jenkins System에 들어가서 SSH키를 등록한다. 설치한 Plugin인 Publish Over SSH를 이용하여 EC2의 SSH키를 등록한다.
- key ⇒ .ssh/id_rsa(비공개 키)의 값. 밑에 꺼 —- 포함해서 다 복사해서 붙여야 함.

```
----BEGIN OPENSSH PRIVATE KEY------
대충 키 내용
-----END OPENSSH PRIVATE KEY----
```

- Name ⇒ 식별하기 쉬운 이름(EC2 Server)
- HostName ⇒ DNS 혹은 Public ip (i11b107.p.ssafy.io)
- UserName ⇒ EC2에서 사용하는 Username (기본적으로는 ubuntu)
- 후에 Test Configuration 진행 후 Success면 끝

6.7 Jenkins에 Item(PipeLine) 생성

```
# Build Triggers 부분에서 Build when a change is pushed to GitLab 파트가 있다.
# 옆에 URL이 있는데 ex) http://3.36.123.182:9000/project/daylog 이거 복사한다. 후에
# 후에 Gitlab의 레포에 들어간다. 그러면 레포의 WebHook 부분이 있을 것인데 들어가서 Webhoo
# Push Events와 Merge Request Events에 체크표시. Push Events에는 내가 원하는 브랜치를

pipeline {
    agent any

    environment {
```

```
        repository = "s11-webmobile1-sub2/S11P12B107" // 내 레포 이름
        dockerImage = 'daylog-backend' // 내가 올리려는 도커 이미지
        imageTag = "latest"
        DOCKERHUB_USERNAME = 'jihunhan' // 내 도커 허브 Username
    }

    stages {
        // 내가 연결할 GitLab을 연결하는 과정.
        stage('Checkout') {
            steps {
                git url: 'https://lab.ssafy.com/s11-webmobile1-sub2/S11P12B10
                    branch: 'deploy-backend',
                    credentialsId: 'daylog_token' // 아까 내가 만든 Repo Token
            }
        }

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

        stage('Test') {
            steps {
                dir('backend') {
                    sh 'chmod +x gradlew'
                    sh './gradlew test'
                }
            }
            post {
                failure {
                    echo 'Gradle test failure!'
                }
                success {
                    echo 'Gradle test success!'
                }
            }
        }

        // 도커 허브로 로그인
        stage('Login to Docker Hub') {
            steps {
                script {
                    withCredentials([usernamePassword(credentialsId: 'docker_
                        sh 'echo $DOCKERHUB_PASSWORD | docker login -u $DOCKE
```

```
                }
            }
        }
    }

    // 아까 만든 jar 파일을 이미지로 만들고 허브로 Push
    stage('Docker Build and Push') {
        steps {
            dir('backend') {
                script {
                    sh '''
                    echo "Building Docker image..."
                    docker build -t ${dockerImage}:${imageTag} .
                    echo "Tagging Docker image..."
                    docker tag ${dockerImage}:${imageTag} ${DOCKERHUB_USE
                    echo "Pushing Docker image..."
                    docker push ${DOCKERHUB_USERNAME}/${dockerImage}:${im
                    '''
                }
            }
        }
        post {
            failure {
                echo 'Docker build and push failure!'
            }
            success {
                echo 'Docker build and push success!'
            }
        }
    }

    stage('Deploy to EC2') {
        steps {
            script {
                def dockerImageFullName = "${DOCKERHUB_USERNAME}/${docker
                sshagent(credentials: ['ec2_sshKey']) {
                    sh """
                    ssh -o StrictHostKeyChecking=no ubuntu@i11b107.p.ssaf
                                                docker stop daylog-backend ||
                    docker rm daylog-backend || true
                    docker run -d -p 8080:8080 -v /home/ubuntu/mediafiles
                    """
                }
            }
        }
    }
```

```
    }
}
```