포팅 메뉴얼

1. 개발 환경

[프론트]

- **Flutter:** 3.19.3
- **Dart**: 3.3.1
- **Intellj:** 2023.3.2

[백엔드]

- **jdk**: 17
- **Springboot:** 3.2.3
- **Mysql:** 8.3.0
- **Redis:** 7.2.4
- **AWS S3**
- **Intellij:** 2023.3.2

2. 설정 파일(.Ignore)

[프론트]

- **.env**

```
APP_KEY =
REST_API_KEY =
```

[백엔드]

- **application.yml**

```
spring:
  profiles:
    include: secret
  servlet:
    multipart:
      max-file-size: 50MB
      max-request-size: 50MB
  jpa:
    properties:
      hibernate:
        format_sql: true
        default_batch_fetch_size: 1000  # select 배치 조회 크기

  batch:
    job:
      enabled: false # 애플리케이션 실행 시 job이 실행되지 않도록하기 위

logging:
  level:
    org.hibernate.SQL: debug
    org.hibernate.type: trace

servlet:
  multipart:
    max-file-size: 20MB
    max-request-size: 20MB

management:
  endpoints:
    web:
      exposure:
        include: "*"
```

- **application-secret.yml**

```
spring:
  data:
    redis:
      host: redis
      port: 7368
      password: <redis 비밀번호>
  datasource:
    url: jdbc:mysql://walkingpet.co.kr:3308/walkingpet?serverTi
    username: B102_walkingpet
    password: <db 비밀번호>
    driver-class-name: com.mysql.cj.jdbc.Driver
    hikari:
      pool-name: jpa-hikari-pool
      maximum-pool-size: 20
      jdbc-url: ${spring.datasource.url}
      username: ${spring.datasource.username}
      password: ${spring.datasource.password}
      driver-class-name: ${spring.datasource.driver-class-name}
      data-source-properties:
        rewriteBatchedStatements: true
jwt:
  secret-key: <jwt 시크릿 키>
  access-token-expiration-time: 1209600000
  #  access-token-expiration-time: 86000000Q
  refresh-token-expiration-time: 12096000000
#  refresh-token-expiration-time: 1209600000
```

```
cloud:
  aws:
    credentials:
      accessKey: <aws 액세스키>
      secretKey: <aws 시크릿키>
    region:
      static: ap-northeast-2
    stack:
      auto: false
    s3:
      bucket: walkingpet.bucket
```

3. 환경 설정

[프론트]

- Flutter

```
 flutter pub get
 flutter clean build
 flutter app build #aab 파일 생성
 flutter build apk #apk 파일 생성
```

[백엔드]

Jenkins 파이프라인

```
pipeline {
    agent any

    environment {
        CONTAINER_NAME = "walkingpet-backend-container"
        IMAGE_NAME = "walkingpet-backend-image"
    }

    stages {
        stage('Git Clone') {
            steps {
                git branch: 'be', credentialsId: 'gitlab-token',
            }
        }
        stage('secret.yml download') {
            steps {
                withCredentials([file(credentialsId: 'applicatio
                    script {
                        def destinationFile = 'Backend/WalkingPe
                        // Create destination file if it doesn't
                        writeFile file: destinationFile, text:
                        // Copy dbConfigFile to destinationFile
                        sh "cp $dbConfigFile $destinationFile"
                    }
                }
            }
        }
        stage('serviceAccountKey download') {
            steps {
                withCredentials([file(credentialsId: 'serviceAcc
                    script {
                        def destinationFile = 'Backend/WalkingPe
                        // Create destination file if it doesn't
                        writeFile file: destinationFile, text:
                        // Copy dbConfigFile to destinationFile
                        sh "cp $dbConfigFile $destinationFile"
                    }
                }
            }
        }
        stage('Build') {
            steps {
                dir('./Backend/WalkingPet'){
                    sh '''
                    chmod +x ./gradlew
                    ./gradlew clean build -x test
                    '''
                }
            }
        }
        stage('SonarQube analysis') {
            steps{
                withSonarQubeEnv('Sonarqube'){
                     dir('./Backend/WalkingPet') {
                        sh './gradlew sonarqube'
                    }
                }
            }
        }
        stage('Docker delete') {
            steps {
                script {
                    try {
                        // 컨테이너가 존재하면 삭제합니다.
                        sh "docker stop ${CONTAINER_NAME}"
                        sh "docker rm -f ${CONTAINER_NAME}"
                    } catch (Exception e) {
                        // 컨테이너가 존재하지 않는 경우 에러가 발생할 
                        echo "Docker container ${CONTAINER_NAME
                    }

                    try {
                        // 이미지가 존재하면 삭제합니다.
                        sh "docker image rm ${IMAGE_NAME}"
                    } catch (Exception e) {
                        // 이미지가 존재하지 않는 경우 에러가 발생할 수 
                        echo "Docker image ${IMAGE_NAME} does no
                    }
                }
            }

            post {
                success {
                    sh 'echo "docker delete Success"'
                }
                failure {
                    sh 'echo "docker delete Fail"'
                }
            }
        }
        stage('Dockerizing'){
            steps{
                sh 'echo " Image Bulid Start"'
                sh """
                    cd ./Backend/WalkingPet
                    docker build -t ${IMAGE_NAME} .
                """
            }
            post {
                success {
                    sh 'echo "Bulid Docker Image Success"'
                }
                failure {
                    sh 'echo "Bulid Docker Image Fail"'
                }
            }
        }
        stage('Deploy') {
            steps {
                sh "docker run --name ${CONTAINER_NAME} -d -p 8
            }
            post {
                success {
                    echo 'deploy success'
                }
                failure {
                    echo 'deploy failed'
                }
            }
        }
        stage('Network Connection') {
            steps {
                sh "docker network connect ubuntu_default ${CON
            }
            post {
                success {
                    echo 'Network Connection Success'
                }
                failure {
                    echo 'Network Connection Failed'
                }
            }
        }

    }
}
```

<파일>

- **docker-compose.yml 파일**

```
version: '3'

services:
  jenkins:
    image: jenkins/jenkins:latest
    container_name: jenkins
    volumes:
      - /var/run/docker.sock:/var/run/docker.sock
      - /home/ubuntu/jenkins-data:/var/jenkins_home
    ports:
      - "8180:8080"
    user: root

  mysql:
    image: mysql:latest
    container_name: mysql
    environment:
      MYSQL_ROOT_PASSWORD: <mysql root 비밀번호> # MySQL root 사용
      MYSQL_DATABASE: walkingpet # MySQL 데이터베이스 이름 (필요한 경
      MYSQL_USER: B102_walkingpet             # MySQL 사용자 이름
      MYSQL_PASSWORD: <mysql 사용자 비밀번호>  # MySQL 사용자 rhythm
    command: --character-set-server=utf8mb4 --collation-server=u
    ports:
      - "3308:3306"                   # MySQL 포트 (필요한 경우 변경
    volumes:
      - /path/to/mysql-data:/var/lib/mysql
  redis:
    image: redis:latest
    container_name: redis
    ports:
      - "6379:6379"
    environment:
      - REDIS_PASSWORD=<redis 비밀번호>
```

- **초기 데이터 세팅 - init.sql → DB 덤프 파일**

  `src/main/resources/init.sql`

4. 외부 API

[로그인]

- **카카오 로그인**
    - https://developers.kakao.com/docs/latest/ko/kakaologin/flutter

[지도]

- kakao map
    - https://apis.map.kakao.com/android_v2/

[이미지]

- AWS S3
    - https://aws.amazon.com/ko/?nc2=h_lg

[FCM]

- 알림
    - https://firebase.google.com/docs/cloud-messaging?hl=ko