TAISO 포팅 매뉴얼

I. 기술 스택 & 개발 환경

사용 도구

- 이슈 관리: JIRA
- 형상 관리: GitLab
- 커뮤니케이션: Mattermost, Notion, Google Docs
- 디자인: Figma, Canva
- UCC: Movavi
- CI/CD: EC2, Docker, Jenkins

개발 환경

- **Front-end**
  - Node.js: 20.11.1
  - React: 18.2.0
  - Typescript: 5.2.2
  - ESLint: 7.1.1
  - Prettier: 3.2.5
  - DaisyUI: 4.9.0
  - PWA: 0.19.7
- **Back-end**
  - JDK: 17.0.10 LTS
  - SpringBoot: 3.2.3
  - SpringSecurity
  - Gradle: 8.5
  - jjwt-api:0.11.5
  - MQTT: spring-integration-mqtt 5.5.0
  - Firebase Admin SDK: firebase-admin:9.2.0
  - DotEnv: dotenv-java:3.0.0
  - Swagger: 2.3.0

    - Mapper: 1.5.3.Final
- **DB**
    - MariaDB: 10.11.7 LTS
- **Self-driving**
    - Simulator : MORAI SIM ver22.R2.1
    - Python : 3.8.10
        - scikit-learn : 1.3.1
        - scipy : 1.10.1
    - Linux os : Ubuntu 20.04.6 LTS
    - ROS : noetic
    - NVDIA Driver : 470.199.02
    - CUDA Version: 11.4
- **Infra**
    - Server:
        - AWS EC2: Ubuntu 20.04.6 LTS
    - Containerization Platform:
        - Docker: 25.0.4
    - CI/CD:
        - Jenkins: 2.448
    - Web Server:
        - Nginx: 1.18.0 (Ubuntu)

외부 서비스

- Kakao Map API
- Firebase API
- Morai Simulator

Gitignore

```
[Back]

HELP.md
.gradle
build/
```

```
!gradle/wrapper/gradle-wrapper.jar
!**/src/main/**/build/
!**/src/test/**/build/

### STS ###
.apt_generated
.classpath
.factorypath
.project
.settings
.springBeans
.sts4-cache
bin/
!**/src/main/**/bin/
!**/src/test/**/bin/

### IntelliJ IDEA ###
.idea
*.iws
*.iml
*.ipr
out/
!**/src/main/**/out/
!**/src/test/**/out/

### NetBeans ###
/nbproject/private/
/nbbuild/
/dist/
/nbdist/
/.nb-gradle/

### VS Code ###
.vscode/

# FCM 비공개 키
/src/main/resources/{FCM 비공개 키 파일 이름}.json
```

```
[Front]
```

```
# Logs
logs
*.log
npm-debug.log*
yarn-debug.log*
yarn-error.log*
pnpm-debug.log*
lerna-debug.log*

node_modules
dist
dist-ssr
*.local

# Editor directories and files
.vscode/*
!.vscode/extensions.json
.idea
.DS_Store
*.suo
*.ntvs*
*.njsproj
*.sln
*.sw?
```

환경변수 (front)

```
VITE_BASE_URL={사이트 URL}
VITE_KAKAO_MAP_API_KEY={카카오맵 API 키}

# Firebase 프로젝트를 식별하는 데 사용
VITE_FIREBASE_API_KEY={Firebase 프로젝트의 API 키}

# Firebase Authentication 서비스를 사용하여 인증할 때 사용
VITE_FIREBASE_AUTH_DOMAIN={Firebase Authentication 도메인}

# Firebase 콘솔에서 프로젝트를 식별하는 데 사용
VITE_FIREBASE_PROJECT_ID={Firebase 프로젝트의 ID}
```

```
# Firebase Storage 서비스에서 파일을 저장할 때 사용
VITE_FIREBASE_STORAGE_BUCKET={Firebase Storage 버킷}

# Firebase Cloud Messaging (FCM)에서 사용
VITE_FIREBASE_MESSAGING_SENDER_ID={Firebase 프로젝트의 메시징 발신자 ID}

# Firebase 앱을 식별하는 데 사용
VITE_FIREBASE_APP_ID={Firebase 앱의 ID}

# Firebase 애널리틱스 서비스를 사용하여 앱 이벤트를 추적할 때 사용
VITE_FIREBASE_MEASUREMENT_ID={Firebase 애널리틱스의 측정 ID}

# 웹 앱에서 Firebase Cloud Messaging(FCM)을 사용하여 푸시 알림을 보낼 때 사용
VITE_FIREBASE_PUBLIC_VAPID_KEY={Firebase 웹 푸시 알림을 위한 VAPID 키}
```

```
[Back]
# 서버 설정
SERVER_PORT={백엔드 서버용 포트}

# DB 서버 연결 설정
DB_URL=jdbc:mariadb://{도메인}:{DB용 포트}/{DB 이름}
DB_USERNAME={DB 아이디}
DB_PASSWORD={DB 패스워드}

# MQTT 연결 설정
MQTT_BROKER_URL=tcp://{도메인}:{MQTT용 포트}
```

II. 빌드 및 배포

개발 환경에서 직접 빌드(로컬 빌드)

[Front]

1. 의존성 설치
   `npm install`
2. 프로젝트 빌드 (정적 파일 생성)
   `npm run build`

[Back]

1. 프로젝트 빌드
   `./gradlew build`
   (Gradle - Tasks - build - bootJar 로도 jar 파일 생성 가능)
2. 빌드된 JAR 파일 실행
   `java -jar build/libs/{프로젝트명}.jar`

- 빌드/실행 동시에 하는 경우
  `./gradlew bootRun`

배포 시 빌드(jenkins 파이프라인)

jenkins 파이프라인

```
pipeline {
    agent any

    environment {
        // BE 디렉터리명
        DIR_BE = 'BE/taiso'
        // FE 디렉터리명
        DIR_FE = 'FE'
        IMG_FE = 'fe-img'
        IMG_BE = 'be-img'
        CONT_FE = 'TAISO-fe'
        CONT_BE = 'TAISO-be'
        FE_PATH = '/TAISO/Front'
        DOCKERFILE_BE = '/TAISO/Back'
        PROJECT_NAME = 'Taiso-develop'
    }

    stages {

        //다운받기 전에 이전 폴더 삭제하기
        stage('Remove Previous Diretory') {
            steps {

                sh 'rm -rf ${PROJECT_NAME}'
```

```
            }
        }

        stage('Checkout') {
            steps {
                git credentialsId: '7f65973d-a3e4-4815-9bd5-09326151d5cd',
                url: 'https://lab.ssafy.com/s10-mobility-autodriving-
sub2/S10P22D212.git',
                branch: 'develop' // 원하는 브랜치 지정
                script {
                    def currentDir = pwd()
                    echo "Current Directory: ${currentDir}"
                }
            }
        }

        stage('Add Env') {
            steps {
                dir("${DIR_BE}") {
                    withCredentials([file(credentialsId: 'env', variable: 'env')])
{
                      sh "cp \$env .env"
                      sh "chmod +r .env"
                    }
                }
            }
        }

        stage('Add FCM SDK KEY') {
            steps {
                dir("${DIR_BE}/src/main/resources") {
                    withCredentials([file(credentialsId: 'fcmKEY', variable:
'fcmKEY')]) {
                      sh "cp \$fcmKEY taiso-18ea8-firebase-adminsdk-m9qcd-
e458ff02c9.json"
                      sh "chmod +r taiso-18ea8-firebase-adminsdk-m9qcd-
e458ff02c9.json"
                    }
                }
            }
```

```
        }


        stage('Build main BE image') {
            steps {

               script {
                try {
                    sh 'ls -al'
                    dir("${DIR_BE}") {
                        sh 'ls -al'
                        sh 'chmod +x ./gradlew'
                        sh './gradlew clean build'
                        sh "docker build . -t ${IMG_BE}"
                        }
                    echo 'Build main image...'
                } catch (Exception e) {

                    // 에러가 발생하면 에러 로그를 파일에 저장
                    def errorLog = "Error occurred in Build main BE image
stage:\n${e}\n"
                    writeFile file: 'error.log', text: errorLog
                    currentBuild.result = 'FAILURE' // 빌드를 실패로 표시
                    throw e // 예외를 다시 던져서 빌드를 중단

                    }
                }

            }
        }


        //BE - 이전 컨테이너 삭제
        stage('Remove Previous main BE Container') {
            steps {
                script {
                    try {
                        sh "docker stop ${CONT_BE}"
                        sh "docker rm ${CONT_BE}"
                    } catch (e) {
```

```
                    echo 'fail to stop and remove main container'
                }
            }
        }
    }

    //새 BE 컨테이너 실행
    stage('Run New main BE image') {
        steps {
            sh "docker run --name ${CONT_BE} -d -p 3000:3000 -v
/home/ubuntu/docker/jenkins-data/workspace/Taiso-
develop/BE/.env:/home/ubuntu/docker/jenkins-data/workspace/Taiso-develop/BE/.env
${IMG_BE}"
            echo 'Run New main BE image'
        }
    }



    //FE - 이미지 생성
    stage('Build FE image') {
        steps {
            dir("${DIR_FE}") {
                sh "ls"
                sh "docker build . -t ${IMG_FE}:latest --no-cache"
                script {
                    def currentDir = pwd()
                    echo "Current Directory: ${currentDir}"
                }
            }
        }
    }


            //이전 컨테이너 삭제
    stage('Remove Previous Container') {
        steps {
            script {
                try {
                    sh "docker stop ${CONT_FE}"
```

```
                        sh "docker rm ${CONT_FE}"
                    } catch (e) {
                        echo 'fail to stop and remove container'
                    }
                }
            }
        }

        //새 FE 컨테이너 실행
        stage('Run FE image') {
            steps {
                script {
                    sh "docker run --name ${CONT_FE} -d -p 5173:5173
${IMG_FE}:latest"
                    sh "docker cp TAISO-fe:/app/dist ."
                }
            }
        }


    }

        post {

            always{
                script{
                    // 빌드 결과에 따라 Mattermost로 메시지 전송
                    def Author_ID = sh(script: "git show -s --pretty=%an",
returnStdout: true).trim()
                    def Author_Name = sh(script: "git show -s --pretty=%ae",
returnStdout: true).trim()

                    if (currentBuild.result == 'SUCCESS') {
                        // 성공 시
                        mattermostSend (
                            color: 'good',
                            message: "빌드 성공: ${env.JOB_NAME}
#${env.BUILD_NUMBER} by ${Author_ID}
(${Author_Name})\n(<${env.BUILD_URL}|Details>)",
                            endpoint:
```

```
'https://meeting.ssafy.com/hooks/6ychb7e6ajnimj5ybt1tzgsg7o',
                            channel: '212'
                        )
                    } else {
                        // 실패 시
                        def errorLog = readFile('error.log')
                        mattermostSend (
                            color: 'danger',
                            message: "빌드 실패: ${env.JOB_NAME}
#${env.BUILD_NUMBER} by ${Author_ID}
(${Author_Name})\n(<${env.BUILD_URL}|Details>)\n에러 로그:\n${errorLog}",
                            endpoint:
'https://meeting.ssafy.com/hooks/6ychb7e6ajnimj5ybt1tzgsg7o',
                            channel: '212'
                        )
                    }
                }
            }
        }

}
```

Frontend Dockerfile

```
FROM node:20.11.1 AS build

# 작업 디렉토리 설정
WORKDIR /app

# 소스 코드 복사
COPY . .

# npm install을 하기 전에 불필요한 파일을 삭제
RUN rm -rf /app/node_modules /app/package-lock.json

# 필요한 패키지 설치
RUN npm install
```

```
# React 프로젝트 빌드
RUN npm run build
```

→ Dockerfile을 사용해서 빌드한 후, 젠킨스 컨테이너에서 만들어진 dist 디렉터리를 컨테이너 밖으로 복사해 옴.

Backend Dockerfile

```
# base 이미지 설정
FROM amazoncorretto:17

# Docker 컨테이너 안으로 환경변수 파일 복사
COPY .env /.env

# json
COPY src/main/resources/{fcm 비공개 키 이름}.json src/main/resources/{fcm 비공개 키 이름}.json

# jar 파일을 컨테이너 내부에 복사
COPY build/libs/{복사 전 jar파일 이름: Taiso(예시)}.jar /{저장할 jar파일 이름: Taiso-copy(예시)}.jar

# 외부에 호출될 포트 설정
EXPOSE {백엔드 포트번호}

# 실행 명령어
ENTRYPOINT ["java","-jar","/{저장한 jar파일 이름: Taiso-copy(예시) }.jar"]
```

🖊 Nginx 설정파일

```
server {
    listen       80;
    listen [::]:80 default_server;
    server_name  {도메인};
    root         {정적 파일의 경로};
    return 301 https://$host$request_uri;
}

server {
```

```
    # TLS1, SSLv2, SSLv3는 보안에 취약하므로 사용하지 말 것.
    ssl_protocols  TLSv1.3 TLSv1.2 TLSv1.1;

    listen 443 ssl;
    listen [::]:443 ssl;

    # root에 정적 파일이 위치한 디렉토리를 지정함.
    # -> 클라이언트에 제공할 정적 파일들이 위치함.
    root {정적 파일의 경로};

    server_name {도메인};

    # HTTP 요청을 처리하는 부분
    # 클라이언트로부터의 요청이 /로 시작할 때의 처리를 정의
    location / {
    # 정적 파일을 제공하고, 존재하지 않는 경우에는 /index.html 파일을 반환
        try_files $uri $uri/ /index.html;
    }

    location /api/ { # Backend 서버로의 요청을 처리할 엔드포인트
        # 프록시 대상 서버 및 포트 설정
        proxy_pass http://{도메인}:{백엔드 서버 포트};
    }


     # SSL 설정을 위한 인증서 및 개인 키 파일의 경로를 지정
    ssl_certificate /etc/letsencrypt/live/{도메인}/fullchain.pem;
    ssl_certificate_key /etc/letsencrypt/live/{도메인}/privkey.pem;
    include /etc/letsencrypt/options-ssl-nginx.conf;
    ssl_dhparam /etc/letsencrypt/ssl-dhparams.pem;

}
```