aingle 포팅 메뉴얼

1. 개요

1.1. 프로젝트 개요

- 프로젝트 명
    - aingle
- 프로젝트 소개
    - 생성형 AI를 활용한 나만의 캐릭터 소셜 네트워크 서비스
- 주요 기능
    1. 게시글 및 채팅을통한 AI 상호작용
    2. 나만의 캐릭터 생성
    3. 공식 캐릭터 등록 및 투표 시스템
    4. FCM을 통한 실시간 알림

1.2. 프로젝트 사용 도구

- 이슈 관리
  - JIRA
- 형상 관리
  - Gitlab
- 커뮤니케이션
  - Notion
  - Mattermost
- 디자인
  - Figma
- DB 설계
  - ERD Cloud
- 동영상 편집
  - Vrew

1.3. 개발 환경

- 프론트엔드
  - VS Code
- 백엔드
  - IntelliJ IDEA
- 인프라
  - gitbash
- DB
  - PostgreSQL, Data grip

1.4. 기술 스택

- 프론트엔드

    - TypeScript 5.5.3
    - React 18.3.1
    - Vite 5.4.8
    - Tailwind CSS 3.4.14
    - Recoil 0.7.7
    - Axios 1.7.7
    - React Query 5.59.19
    - Vite Plugin PWA 0.20.5
    - FontAwesome 6.6.0
    - Firebase SDK 11.0.1
    - Google Analytics 2.1.10
- 백엔드
    - Java 17
    - Jwt 0.11.5
    - Spring-Boot 3.3.5
    - Spring Data JPA 3.3.3
    - Spring Security 6.3.3
- DB
    - MySQL 17.0
- 인프라
    - AWS EC2
    - AWS S3
    - Docker
    - Jenkins
    - NginX

1.5. 추가 기술 스택

- 실시간 알림 기능 관련
  - FCM
  - Firebase 11.0.1

2. 빌드

2.1. 프론트엔드 빌드 방법

1. 버전
   a. Vite 5.4.8
2. 라이브러리 설치
   a. aingle-front 디렉터리에 아래 명령어로 설치

```
nmp i
```

   b. local server 실행

```
npm run dev
```

3. 빌드
   a. dist 파일 생성

```
npx run build
```

2.2. 백엔드 빌드 방법

1. 버전
   a. JAVA Open-JDK 17
   b. SpringBoot 3.3.5
   c. Gradle 8.10
2. 빌드

```
plugins {
    id 'java'
    id 'org.springframework.boot' version '3.3.5'
    id 'io.spring.dependency-management' version '1.1.6'
}

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

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

configurations {
    compileOnly {
        extendsFrom annotationProcessor
    }
}

repositories {
    mavenCentral()
    maven { url 'https://repo.spring.io/milestone' }
}

ext {
    set('springAiVersion', "1.0.0-M3")
}

dependencies {
    implementation 'org.springframework.boot:spring-boot-s
    implementation 'org.springframework.boot:spring-boot-s
    implementation 'org.springframework.boot:spring-boot-s
    // https://mvnrepository.com/artifact/org.postgresql/p
    implementation 'org.postgresql:postgresql:42.7.4'
    //openAI
    implementation 'org.springframework.ai:spring-ai-opena
```

```
    compileOnly 'org.projectlombok:lombok'
    developmentOnly 'org.springframework.boot:spring-boot-
    runtimeOnly 'org.postgresql:postgresql'
    annotationProcessor 'org.projectlombok:lombok'
    testImplementation 'org.springframework.boot:spring-bo
    testImplementation 'org.springframework.security:sprin
    testRuntimeOnly 'org.junit.platform:junit-platform-lau

    //Swagger
    implementation 'org.springdoc:springdoc-openapi-starte

    //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'

    //ModelMapper
    implementation group: 'org.modelmapper', name: 'modelm

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

    // mok
    implementation 'org.springframework.boot:spring-boot-s
    testImplementation 'org.springframework.boot:spring-bo
    implementation 'org.springframework:spring-web'
    implementation 'org.springframework:spring-test'

    // fcm
    implementation 'com.google.firebase:firebase-admin:9.1
}

dependencyManagement {
    imports {
        mavenBom "org.springframework.ai:spring-ai-bom:${s
    }
}
```

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

```
./gradlew build
```

3. Properties

```
spring.config.import: optional:application-private.yml

server:
  port: 8080
  servlet:
    context-path: /api

spring:
  application:
    name: aingle-backend
  servlet:
    multipart:
      max-file-size: 10MB
      max-request-size: 10MB

springdoc:
  swagger-ui:
    path: ${swagger-url}

jwt:
  expiration_time: 3600000 #1시간
  secret: ${JWT}

kakao:
  grant-type: authorization_code
  client-id: ${kakao-client-id}
  redirect-uri: ${kakao-redirect-uri}

cloud:
```

```
  aws:
    credentials:
      access-key: ${S3_ACCESS_KEY}
      secret-key: ${S3_SECRET_KEY}
    s3:
      bucket: ${S3_BUCKET}
    region: ap-northeast-2

firebase:
  config-path: classpath:aingle-ab0b9-firebase-adminsdk-u2
```

2.3. 배포하기

1. 배포 환경
    a. AWS EC3 (Ubuntu 20.04.6)
    b. AWS S3
    c. Docker 27.3.1
    d. Jenkins 2.462.4
2. 배포 방법
    a. 설치
        i. Docker 설치

```
# apt update
sudo apt update

# 패키지 설치
$sudo apt-get install apt-transport-https ca-certifi

# gpg 설치 및 도커 저장소 key 저장
sudo curl -fsSL https://download.docker.com/linux/ub

# 도커 다운로드 및 레포지토리에 추가
sudo add-apt-repository "deb [arch=amd64] https://do
```

```
# 도커 설치
apt-cache policy docker-ce
sudo apt install docker-ce
```

ii. jenkins 설치

```
# port 설정: 9090포트로 외부에서 접속
# volum mount 통해 로컬과 연결
# sock 통해 인증
docker run -d --name jenkins \
    -p 9090:8080 \
    -v /var/jenkins_home:/var/jenkins_home \
    -v /var/run/docker.sock:/var/run/docker.sock \
    -v /usr/bin/docker:/usr/bin/docker \
    -v /usr/local/bin/docker-compose:/usr/local/bin/
    -e TZ=Asia/Seoul \
    --privileged --user root jenkins/jenkins:lts-jdk
```

b. CI/CD 환경 구축

i. NginX를 위해 미리 네트워크 생성

```
docker network create aingle
```

ii. Gitlab과 연결

1. Gitlab에서 access token 발급
2. Jenkins credentials에 access token 등록
    a. Username with password로 생성하여, 본인 Gitlab ID 토큰 입력
3. Jenkinsdp pipe line item 생성
4. Gitlab build trigger 설정
    a. push
5. Gitlab Web Hook 걸기
    a. project hook에서 jenkins project의 url 넣기
    b. token 넣기

iii. NginX 설정

1. /etc/nginx/templates/aingle.conf.template

```
server {
    listen 80;
    server_name aingle.co.kr;

    return 308 https://aingle.co.kr$request_uri;
}

server {
    listen 443 ssl;
    server_name aingle.co.kr;

    ssl_certificate /etc/letsencrypt/live/aingle.
    ssl_certificate_key /etc/letsencrypt/live/ain

    client_max_body_size 10M; # 필요한 크기로 설정

    location /api {
        proxy_pass http://spring:8080;
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_a
        proxy_set_header X-Forwarded-Proto $schem
    }

    location / {
        proxy_pass http://nginx-react:5173;
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_a
        proxy_set_header X-Forwarded-Proto $schem
    }
}
```

1. 443 port로 들어오면 url 바탕으로 proxy
    a. /api가 붙어 있는 경우, Spring Container의 8080 port로 이동

b. 그 외, nginx-react의 5173 port로 이동

c. docker 내부 dns 서버를 활용하여 spring, nginx-react컨테이너 찾기

d. 인증서를 사용하여 https 설정

iv. 백엔드 CI/CD 구축

1. Pipe line 구축

```
pipeline {
    agent any

    environment {
        SPRING_IMAGE = 'spring' // Spring 백엔드 이
        COMPOSE_FILE = 'docker-compose.yml' // 도커
    }
    tools {
        gradle 'gradle' // Gradle 도구 사용
    }

    stages {
        stage('Checkout') {
            steps {
                // Git 리포지토리에서 'develop/serve
                git branch: 'back-develop',
                url: 'https://lab.ssafy.com/s11-f
                credentialsId: 'jsj046@naver.com'
            }
        }

        stage('application-key download') {
            steps {
                dir('aingle-back'){
                    withCredentials([file(credent
                        script {
                            sh 'cp $applicationPr
                            sh 'ls src/main/resou
                            sh 'cat src/main/reso
                        }
```

```
                    }
                }
            }
        }

        stage('firebase-key download') {
            steps {
                dir('aingle-back'){
                    withCredentials([file(credent
                        script {
                            sh 'cp $fireBase src/
                            sh 'ls src/main/resou
                            sh 'cat src/main/reso
                        }
                    }
                }
            }
        }

        stage('Build Spring Boot') {
            steps {
                script {
                    dir('aingle-back') { // BackE
                        sh 'chmod +x gradlew' //
                        sh './gradlew build -x te
                    }
                }
            }
        }

        stage('Stop and Remove Existing Container
            steps {
                script {
                    sh 'docker-compose -f ${COMPO
                }
            }
        }
```

```
        stage('Remove Existing Docker images') {
            steps {
                script {
                    sh "docker rmi -f \$(docker i
                }
            }
        }

        stage('Build Docker Image Spring') {
            steps {
                script {
                    dir('aingle-back'){
                        sh 'docker build -t ${SPR
                    }
                }
            }
        }

        stage('docker-compose up') {
            steps {
                script {
                    sh 'docker-compose -f ${COMPO
                }
            }
        }

        stage('Nginx Restart') {
            steps {
                script {
                    // Nginx 컨테이너 재시작
                    sh 'docker restart nginx'
                }
            }
        }
    }
}
```

1. push, merge 액션 발생

2. branch 이동
3. seceret config 파일 download
    a. Jenkins Credentials에 등록된 applicaion-key.yml, firebase-key
4. build
5. 생성된 build 파일의 jar 파일 image화
    a. docker file을 미리 설정
    b. docker file 코드

```
FROM openjdk:17-alpine
RUN apk add --no-cache tzdata
ARG JAR_FILE=/build/libs/aingle-0.0.1-SNAPS
COPY ${JAR_FILE} app.jar
ENTRYPOINT ["java","-jar","/app.jar"]
```

6. 해당 image를 Build ⇒ 이미지 생성
7. docker-compose를 내리고, 생성된 image를 다시 실행

```
version: '3.8'

services:
  spring:
    container_name: spring
    image: spring
    environment:
      - TZ=Asia/Seoul
    privileged: true
    networks:
      - aingle

networks:
  aingle:
    external: true
```

1. image 실행하여 container 생성

2. network 포함하여 연결, 해당 container에 접근 가능하도록 함

8. NginX 재실행

v. 프론트엔드 CI/CD 구축

1. Pipe line 구축

```
pipeline {
    agent any // Jenkins 에이전트를 아무거나 사용

    environment {
        REACT_IMAGE = 'react' // React 프론트엔드 이
        COMPOSE_FILE = 'docker-compose.yml' // 도커
    }
    tools {
        nodejs 'nodejs' // Node.js 도구 사용
    }

    stages {
        stage('Checkout') {
            steps {
                // Git 리포지토리에서 'deploy' 브랜치를
                git branch: 'front-develop',
                url: 'https://lab.ssafy.com/s11-f
                credentialsId: 'jsj046@naver.com'
            }
        }

        stage('.env download') {
            steps {
                dir('aingle-front'){
                    withCredentials([file(credent
                        script {
                            sh 'cp $envFile .env'
                            sh 'ls -la'
                            sh 'cat .env'
                        }
                    }
                }
```

```
            }
        }


        stage('Build React Frontend') {
            steps {
                script {
                    dir('aingle-front') { // Fron
                        sh 'npm install' // npm 파
                        sh 'NODE_OPTIONS="--no-wa
                    }
                }
            }
        }

        stage('Stop and Remove Existing Container
            steps {
                script {
                    // 기존 도커 컨테이너 중지 및 제거
                    sh 'docker-compose -f ${COMPO
                }
            }
        }

        stage('Remove Existing Docker Images') {
            steps {
                script {
                    // 기존 React 도커 이미지 삭제
                    sh "docker rmi -f \$(docker i
                }
            }
        }


        stage('docker-compose up') {
            steps {
                script {
                    sh 'docker-compose -f ${COMPO
```

```
                }
            }
        }

        stage('Copy Built Files to Nginx Director
            steps {
                script {
                    // 빌드된 React 파일을 Nginx 컨터
                    sh 'docker cp aingle-front/di
                }
            }
        }

        stage('Nginx Restart') {
            steps {
                script {
                    // Nginx 컨테이너 재시작
                    sh 'docker restart nginx-reac
                    sh 'docker restart nginx'
                }
            }
        }
    }
}
```

1. push, merge 액션 발생
2. branch 이동
3. .env 파일 download
    a. Jenkins Credentials에 등록된 .env
4. build
5. react build 통해 이미지 생성
6. docker-compse 통해 container 생성

```
services:
  nginx-react:
    container_name: nginx-react # 컨테이너 이름을
```

```
    image: nginx:alpine # 사용할 도커 이미지를 'ng
    volumes:
      - /home/ubuntu/nginx-react:/etc/nginx/te
      - /home/ubuntu/nginx-react/logs:/var/log
    networks:
      - aingle # 'aingle' 네트워크에 연결
    environment:
      - TZ=Asia/Seoul # 컨테이너 시간대를 'Asia/Se

networks:
  aingle:
    external: true # 'aingle' 네트워크를 외부에서
```

1. 빌드된 react 파일을 nginx container의 경로로 복사 후 생성

7. NginX 재실행

c. 포트 개방

i. 443 ⇒ https 접속

ii. 5432⇒ PostgreSQL

iii. 5173 ⇒ React

iv. 8080 ⇒ 백엔드 통신

v. 9090 ⇒ Jenkins 접속

3. 프로젝트에서 사용하는 외부 서비스 정보를 정리한 문서

3.1. Kakao Login API

- 사용 목적: 소셜 로그인
- 환경 변수 설정

```
kakao:
  grant-type: authorization_code
  client-id: ${kakao-client-id}
  redirect-uri: ${kakao-redirect-uri}
```

- 필요 정보
  - Kakao Developers 계정 생성
  - 애플리케이션 등록 후 키 발급

3.2. Google Login API

- 사용 목적 : 소셜 로그인
- 환경 변수 설정
- 필요 정보
  - Google Cloud 계정 생성
  - 애플리케이션 등록 후 키 발급

3.3. S3 Bucket

- 사용 목적 : 사진, 파일 저장
- 환경 변수 설정

```
cloud:
  aws:
    credentials:
      access-key: ${S3_ACCESS_KEY}
      secret-key: ${S3_SECRET_KEY}
    s3:
      bucket: ${S3_BUCKET}
    region: ap-northeast-2
```

- 필요 정보
  - AWS 계정 생성
  - S3 Bucket 생성
  - 접근 키 및 비밀 키 발급