

포팅 메뉴얼

1. 개발 환경

1.1. BackEnd

(1) JAVA

- Java OpenJDK 17.0.12
- SpringBoot 3.4.2
 - Spring Data Redis
 - Spring Data MongoDB
 - Lombok
 - Spring Web
 - Spring Security
 - OAuth2 Client
 - Validation
- Swagger UI 3.0.0
- JWT
- Firebase

(2) Node JS - Web RTC

- Node JS 22.13.1 (LTS)
- npm 10.9.2
- socket.io 4.8.1
- mediasoup 3.15.3
- mediasoup-client 3.8.1
- express 4.18.2
- dotenv 16.4.7
- nodemon 3.0.3

- webpack 5.97.1

(3) Node JS - Chat

- Node JS 22.13.1 (LTS)
- npm 10.9.2
- socket.io 4.8.1
- socket.io-client 4.8.1
- firebase-admin 13.0.2
- mongoose 8.9.6
- dotenv 16.4.7
- express 4.21.2
- nodemon 3.1.9
- webpack 5.97.1

1.2. DataBase

- MongoDB 8.0.4
- Redis 3.0.504
- Redis-cli 7.4.2

1.3. Server/Infra

- Ubuntu 20.04 LTS
- Jenkins 2.479.3
- Docker 27.5.1
- Docker Compose 2.32.4
- Caddy 2.9.1

1.4. Android

- Kotlin 2.1.10

- Gradle Version 8.9
- Android Gradle Plugin Version 8.7.3
- google-service 4.4.2
- compose 1.7.7
- kakao sdk v2-user 2.12.0
- socket.io-client 2.1.1

1.5. IDE

- IntelliJ IDEA Community Edition 2024.3.2.2
- Visual Studio Code 1.97.2
- Android Studio 2024.2.2

2. 빌드 시 사용되는 환경변수

2.0. EC2 인스턴스에서 Jenkins 컨테이너 빌드 명령어

```
sudo docker run -d -p 8080:8080 --name jenkins -v /home/ubuntu/jenkins
-data:/var/jenkins_home -v /var/run/docker.sock:/var/run/docker.sock jenkins/jenkins:lts
```

2.1. 파이프라인 파일

▼ Jenkinsfile

```
pipeline {
    agent any

    stages {
        stage('Checkout') {
            steps {
                // Git 저장소에서 소스를 체크아웃합니다.
                checkout scm
            }
        }
    }
}
```

```

stage('Docker Compose Build & Run') {
    steps {
        script {
            sh 'docker-compose down || true'
            sh 'docker-compose up -d --build'
        }
    }
}

post {
    always {
        echo 'Pipeline 실행 완료!'
    }
    success {
        script {
            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()
            mattermostSend(color: 'good',
                message: "✅ 빌드 성공: ${env.JOB_NAME} #${env.BUILD_NUMBER}",
                endpoint: 'https://meeting.ssafy.com/hooks/8xib9irpwiyn8r6z',
                channel: 'Jenkins_Build_Result'
            )
        }
    }
    failure {
        echo 'Pipeline 실패! 로그를 확인하세요.'
        script {
            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()
            mattermostSend(color: 'danger',
                message: "❌ 빌드 실패: ${env.JOB_NAME} #${env.BUILD_NUMBER}",
                endpoint: 'https://meeting.ssafy.com/hooks/8xib9irpwiyn8r6z',
                channel: 'Jenkins_Build_Result'
            )
        }
    }
}

```

```
}
```

2.2. Docker 파일

- Spring - Dockerfile 경로
 - /BackEnd/ssacle
- Node JS WebRTC - Dockerfile 경로
 - /WebRTC
- Node JS Chat - Dockerfile 경로
 - /BackEnd/chat

2.3. docker-compose.yml

▼ docker-compose.yml

```
version: "3.8"

services:
  mongodb:
    image: mongo:8.0.4
    container_name: mongodb
    ports:
      - "27017:27017"
    environment:
      MONGO_INITDB_ROOT_USERNAME: root
      MONGO_INITDB_ROOT_PASSWORD: password
      MONGO_INITDB_DATABASE: ssacle
    volumes:
      - mongodb_data:/data/db
      - ./resources/mongodb-init.d:/docker-entrypoint-initdb.d # 초기 데이터
    networks:
      - app-network

# 레디스
redis:
```

```
image: redis:latest
container_name: redis
ports:
  - "6379:6379"
command: redis-server --requirepass root --port 6379 --bind 0.0.0.0
volumes:
  - ./redis-data:/data
networks:
  app-network:
  aliases:
    - redis
```

spring Backend

backend:

build:

context: ./BackEnd/ssacle

dockerfile: Dockerfile

container_name: spring

ports:

- "5001:5001"

volumes:

- ./uploads:/app/uploads

depends_on:

- mongodb # 몽고db 먼저 실행되어야 함

- redis

restart: always

environment:

SPRING_DATA_MONGODB_URI: mongodb://root:password@mongodb:

SPRING_REDIS_HOST: redis

SPRING_REDIS_PORT: 6379

SPRING_REDIS_PASSWORD: root

networks:

- app-network

webrtc:

```
build:
  context: ./WebRTC
  dockerfile: Dockerfile

container_name: webrtc
env_file:
  - WebRTC/.env
ports:
  - "4000:4000"
  - "10000-10100:10000-10100/udp"
  - "10000-10100:10000-10100/tcp"
depends_on:
  - mongodb
  - redis
restart: always
environment:
  MONGO_URI: mongodb://root:password@mongodb:27017/ssacle?auth

# Redis 연결 정보
REDIS_HOST: redis
REDIS_PORT: 6379
REDIS_PASSWORD: root

# WebSocket 서버 설정
WEBSOCKET_PORT: 4000
SIGNALING_SERVER: ws://webrtc:4000

NODE_ENV: production

networks:
  - app-network

chat:
  build:
    context: ./BackEnd/chat
    dockerfile: Dockerfile

  container_name: chat
```

```
env_file:
  - BackEnd/chat/.env
ports:
  - "4001:4001"
depends_on:
  - mongodb
restart: always

networks:
  - app-network

networks:
  app-network:

volumes:
  jenkins_home:
  mongodb_data:
```

2.4. 사용 포트 번호

- Spring
 - 5001
- WebRTC
 - socket.io - 4000
 - webrtc- 10000:10100
- Chat
 - 4001
- Jenkins
 - 8080
- MongoDB
 - 27017
- Redis
 - 6379

2.5. Android 내 OpenAPI key

▼ local.properties

```
OPENAI_API_KEY='your_api_key'
```

2.6. Caddy 파일 작성

- 리버스 프록시 및 https 적용을 위해 EC2 인스턴스 내에서 Caddyfile 생성 및 작성

▼ Caddyfile

```
{
  admin 0.0.0.0:2020
}
jenkins.43.203.250.200.nip.io {
  reverse_proxy localhost:8080 {
    header_up Host {host}
    header_up X-Real-IP {remote_host}
    header_up X-Forwarded-For {remote_host}
    header_up X-Forwarded-Proto {scheme}
  }
}

webrtc.43.203.250.200.nip.io {
  reverse_proxy https://localhost:4000 {
    transport http {
      tls_insecure_skip_verify
    }
    header_up Host {host}
    header_up X-Real-IP {remote_host}
    header_up X-Forwarded-For {remote_host}
    header_up X-Forwarded-Proto {scheme}
    header_up Connection {>Connection}
    header_up Upgrade {>Upgrade}
  }
}

chat.43.203.250.200.nip.io {
```

```
reverse_proxy https://localhost:4001 {
    transport http {
        tls_insecure_skip_verify
    }
    header_up Host {host}
    header_up X-Real-IP {remote_host}
    header_up X-Forwarded-For {remote_host}
    header_up X-Forwarded-Proto {scheme}
    header_up Connection {>Connection}
    header_up Upgrade {>Upgrade}
}

}
```

▼ 명령어

```
sudo vi /etc/caddy/Caddyfile
sudo systemctl daemon-reload
sudo systemctl enable --now caddy
sudo caddy start
```

3. 배포 시 특이사항

- AWS 인스턴스에서 Docker 기반 서버 배포 시 Caddy를 활용하여 리버스 프록시 및 자동 HTTPS 인증 적용
- Node JS (WebRTC, Chat) 서버 배포 시 사설 SSL 인증서 사용
- Spring 서버는 http 통신으로 배포하여 리버스 프록시 및 HTTPS 인증을 적용하지 않음
- 파이프라인 종료 시 빌드 결과 및 상태 알림 MatterMost 채널에 전송하여 빌드 관리

▼ MatterMost 연동 이미지



장홍준[구미_1반_D110]팀원 BOT 오후 12:43

✅ 빌드 성공: gitlab-pipeline #438 by 장홍준(jhjun99421@gmail.com)
(Details)



장홍준[구미_1반_D110]팀원 BOT 오후 2:21

✅ 빌드 성공: gitlab-pipeline #439 by ajang369(jhjun99421@gmail.com)
(Details)

- Gitlab Webhook과 연동하여 master 브랜치에 merge 이벤트 발생 시 파이프라인 실행

3.1. 앱 사용 시 안내 사항

- 카카오 회원가입/로그인 이후, SSAFY 교육생 인증 단계에서 DB에 등록된 교육생의 이름과 학번으로 인증 필요하여, 임시 데이터를 넣었으므로 아래에 입력한 임시 교육생 정보로 인증하면 됨.

사용자 1

- 이름 - 테스트
- 학번 - 1201234

사용자 2

- 이름 - 모다니
- 학번 - 1205411

사용자 3

- 이름 - 강해린
- 학번 - 1206515

4. 주요 계정 및 프로퍼티가 정의된 파일 목록

4.1. BackEnd - Spring

- ▼ application.yml

```
spring:
  # DB
  data:
    mongodb:
      host: localhost
      database: ssacle
    redis:
      host: ${SPRING_REDIS_HOST}
      port: ${SPRING_REDIS_PORT}
      password: ${SPRING_REDIS_PASSWORD}

# 이미지 파일 파일 업로드 크기 제한 + Spring이 파일 업로드를 차단 해제
servlet:
  multipart:
    enabled: true
    max-file-size: 10MB
    max-request-size: 10MB
# 로그 창 글씨 색상 설정
output:
  ansi:
    enabled: always

jwt:
  # 액세스 토큰 만료시간 (원래는 1시간=60*60)
  access:
    token:
      expiration:
        seconds: 604800
  # 리프레시 토큰 만료시간 (7일=60*60*24*7)
  refresh:
    token:
      expiration:
        seconds: 604800
  # JWT 서명 시크릿 키
  token:
    secret:
```

```
key: "your_key"
```

```
logging:
```

```
level:
```

```
root: INFO
```

```
org.springframework.security: DEBUG
```

```
# 서버 포트번호
```

```
server:
```

```
port: 5001
```

```
# 파일 업로드 주소
```

```
file:
```

```
upload-dir: /app/uploads
```

- Firebase Cloud Messaging 등록
 - Firebase Console → 프로젝트 → 프로젝트 설정 → 서비스 계정 → 새 비공개 키 생성
 - serviceAccountKey.json 파일 경로
 - /BackEnd/ssacle/src/main/resources

4.2. BackEnd - Node JS - Chat

- Firebase Cloud Messaging 등록
 - serviceAccountKey.json 파일 경로
 - /BackEnd/chat

▼ .env

```
SSL_KEY_PATH=./cert/key.pem
```

```
SSL_CERT_PATH=./cert/cert.pem
```

```
ANNOUNCED_IP="your_ip"
```

```
PORT=4001
```

```
MONGO_URI=mongodb://root:password@mongodb:27017/ssacle?authSource=admin
```

```
FIREBASE_SERVICE_ACCOUNT_KEY=./serviceAccountKey.json
```

4.3. BackEnd - Node JS - WebRTC

▼ .env

```
# SSL 인증서 경로
SSL_KEY_PATH=./cert/key.pem
SSL_CERT_PATH=./cert/cert.pem

# mediasoup 설정
ANNOUNCED_IP="your_ip"
LISTEN_IP=0.0.0.0

# 서버 포트
PORT=4000
```

4.4. Android Studio


- google-service.json 추가
 - 경로 = /Android/app

5. 외부 서비스 정보

5.1. 카카오 소셜 로그인

Kakao Developers

카카오 API를 활용하여 다양한 어플리케이션을 개발해보세요. 카카오 로그인, 메시지 보내기, 친구 API, 인공지능 API 등을 제공합니다.

 <https://developers.kakao.com/docs/latest/ko/kakaologin/common>

kakao developers


5.2. Firebase - FCM

<https://console.firebase.google.com/?hl=ko>

5.3. OpenAI API

OpenAI API

We're releasing an API for accessing new AI models developed by OpenAI.

 <https://openai.com/index/openai-api/>

