포팅 매뉴얼

태그	🗃️ 자료함

1. 개발환경

1. 프로젝트 기술 스택

Frontend

- Visual Studio Code(IDE) `1.85.1`
- HTML5, CSS3, Javascript(ES6)
- React `18.2.0`
- Vite `5.2.0`
- Nodejs `20.10.0`
- Typescript `5.2.2`
- Tanstack Query `5.29.2`
- Zustand `4.5.2`
- Tailwind CSS `3.4.3`
- axios `1.6.8`
- @tensorflow-models/handpose `0.1.0`
- fingerpose `0.1.0`
- openvidu-browser `2.29.1`
- react-speech-recognition `3.10.0`
- sockjs-client `1.6.1`
- stompjs `2.3.3`

Backend

- IntelliJ `2023.3.2`
- Java OpenJDK `17`
- JWT : `0.12.5`
- Spring Boot `3.2.4`
  - JAVA Spring Data JPA
  - Spring Security
- Gradle
- ORM : JPA
- spring-boot-WebSocket

- STOMP

CI/CD

- AWS EC2
  - NginX `1.18.0`
  - Ubuntu `20.04.6 LTS`
  - Docker `26.0.2`
  - Jenkins `2.443`
- Docker Hub

2. 환경변수 설정

Frontend : .env

```
VITE_KAKAO_REST_API_KEY='카카오 api 키'
VITE_KAKAO_JAVASCRIPT_API_KEY='카카오 api 키'

# 배포
VITE_API_BASE_URL='https://k10d208.p.ssafy.io'
VITE_KAKAO_REDIRECT_URI='https://k10d208.p.ssafy.io/oauth/cal

# 로컬
# VITE_API_BASE_URL='http://localhost:8000'
# VITE_KAKAO_REDIRECT_URI='http://localhost:5173/oauth/callba
```

Backend: env.yml

- env.yml 파일

```
spring:
  config:
    import: "optional:classpath:env.yml"
```

```
  rabbitmq:
    host: ${RABBIT_URL}
    port: ${RABBIT_PORT}
    username: ${RABBIT_USR}
    password: ${RABBIT_PASSWORD}
    queue:
      name: ChatQueue
    exchange:
      name: myDirectExchange  # Exchange 이름을 여기에 설정

  servlet:
    multipart:
      max-file-size: 5MB
      max-request-size: 20MB


  datasource:
    url: ${DATABASE_URL}
    driver-class-name: com.mysql.cj.jdbc.Driver
    username: ${DATABASE_USERNAME}
    password: ${DATABASE_PASSWORD}


  jpa:
    hibernate:
      ddl-auto: update
    properties:
      hibernate:
        show_sql: false
        format_sql: true
        default_batch_fetch_size: 100
        dialect: org.hibernate.dialect.MySQLDialect
  h2:
    console:
      enabled: false

  data:
    redis:
```

```
      host: ${REDIS_HOST}
      port: ${REDIS_PORT}
      password: ${REDIS_PASSWORD}


server:
  port: 8000

open-ai:
  url: https://api.openai.com/v1/chat/completions
  secret-key: ${OPEN_AI_SECRET_KEY}

cloud:
  aws:
    credentials:
      access-key: ${AWS_ACCESS_KEY}
      secret-key: ${AWS_SECRET_KEY}
    region:
      static: ap-northeast-2
    stack:
      auto: false
    s3:
      bucket: ${BUCKET_NAME}
```

3. 설정 파일

Nginx : nginx.conf

- 파일 설정 위치

```
/etc/nginx/conf.d
/etc/nginx/sites-enabled/default
```

- nginx.conf

```
user www-data;
worker_processes auto;
```

```
pid /run/nginx.pid;
include /etc/nginx/modules-enabled/*.conf;

events {
    worker_connections 768;
    # multi_accept on;
}

http {

    ##
    # Basic Settings
    ##

    sendfile on;
    tcp_nopush on;
    tcp_nodelay on;
    keepalive_timeout 65;
    types_hash_max_size 2048;
    # server_tokens off;

    # server_names_hash_bucket_size 64;
    # server_name_in_redirect off;

    include /etc/nginx/mime.types;
    default_type application/octet-stream;

    ##
    # SSL Settings
    ##

    ssl_protocols TLSv1 TLSv1.1 TLSv1.2 TLSv1.3; # Dropping S
    ssl_prefer_server_ciphers on;

    ##
    # Logging Settings
    ##
```

```
    access_log /var/log/nginx/access.log;
    error_log /var/log/nginx/error.log;

    ##
    # Gzip Settings
    ##

    gzip on;

    # gzip_vary on;
    # gzip_proxied any;
    # gzip_comp_level 6;
    # gzip_buffers 16 8k;
    # gzip_http_version 1.1;
    # gzip_types text/plain text/css application/json applica

    ##
    # Virtual Host Configs
    ##

    include /etc/nginx/conf.d/*.conf;
    include /etc/nginx/sites-enabled/*;
}


#mail {
#   # See sample authentication script at:
#   # http://wiki.nginx.org/ImapAuthenticateWithApachePhpScri
#
#   # auth_http localhost/auth.php;
#   # pop3_capabilities "TOP" "USER";
#   # imap_capabilities "IMAP4rev1" "UIDPLUS";
#
#   server {
#       listen     localhost:110;
#       protocol   pop3;
#       proxy      on;
#   }
```

```
#
#   server {
#       listen     localhost:143;
#       protocol   imap;
#       proxy      on;
#   }
#}
```

- sites-enables/default

```
# 기본 HTTP 서버
server {
    listen 80 default_server;
    listen [::]:80 default_server;

    root /var/www/html;
    index index.html index.htm index.nginx-debian.html;

    server_name _;

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

# HTTPS 서버
server {
    listen 443 ssl; # HTTPS 포트
    listen [::]:443 ssl ipv6only=on; # IPv6 포트

    server_name k10d208.p.ssafy.io; # HTTPS 도메인

    ssl_certificate /etc/letsencrypt/live/k10d208.p.ssafy.io/
    ssl_certificate_key /etc/letsencrypt/live/k10d208.p.ssafy
    include /etc/letsencrypt/options-ssl-nginx.conf; # SSL 옵
    ssl_dhparam /etc/letsencrypt/ssl-dhparams.pem; # SSL DH 파

    location / {
```

```
        proxy_pass http://localhost:5173; # 프론트엔드 프록시
        proxy_http_version 1.1;
        proxy_set_header Upgrade $http_upgrade;
        proxy_set_header Connection 'upgrade';
        proxy_set_header Host $host;
        proxy_cache_bypass $http_upgrade;
    }

    location /api/ {
        proxy_pass http://localhost:8000/api/; # 백엔드 프록시
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forward
        proxy_set_header Host $http_host;
    }

    location /ws/ {
        proxy_pass http://localhost:8000/ws/; # 백엔드 프록시
        proxy_http_version 1.1;
        proxy_set_header Upgrade $http_upgrade;
        proxy_set_header Connection "upgrade";
        proxy_set_header Host $http_host;
    }
}

# HTTP에서 HTTPS로 리다이렉트
server {
    listen 80; # HTTP 포트
    listen [::]:80; # IPv6 포트

    server_name k10d208.p.ssafy.io; # 도메인
    if ($host = k10d208.p.ssafy.io) {
        return 301 https://$host$request_uri; # HTTP에서 HTTPS
    }
}
```

Jenkins : Jenkinsfile

- 설정 파일 위치 : 프로젝트 최상단

- Jenkinsfile

```
pipeline {
    agent any

    tools {
        nodejs 'nodejs-20.10.0'
    }

    stages {
        stage('MM-Alarm'){
            steps{
                script {
                    def Author_ID = sh(script: "git show -s -
                    def Author_Name = sh(script: "git show -s
                    mattermostSend (
                        color: '#D0E0E3',
                        icon: "https://jenkins.io/images/logo
                        message: "파이프라인 시작: ${env.JOB_NAM
                    )
                }
            }
        }


        stage('Clone-Back') {
            steps {
                echo '클론을 시작!'
                git branch: 'final2', credentialsId: 'gitlab_
                echo '클론을 완료!'
            }
        }

        stage('Add Back Env') {
            steps {
                echo '백엔드 환경 설정'
                dir('./backend') {
```

```
                withCredentials([file(credentialsId: 'bac
                sh 'cp ${env}  src/main/resources/env.yml
                }
            }
        }
    }

    stage('BE-Build') {
        steps {
            echo '백엔드 빌드 및 테스트 시작!'
            dir("./backend") {
                sh "ls"
                sh "chmod +x ./gradlew"

                // sh "touch ./build.gradle"

                // application properties 파일 복사
                // sh "echo $BuildGradle > ./build.gradle

                sh "./gradlew build --exclude-task test"

            }
            echo '백엔드 빌드 및 테스트 완료!'
        }
    }

    stage('Build Back Docker Image') {
        steps {
            echo '백엔드 도커 이미지 빌드 시작!'
            dir("./backend") {
                // 빌드된 JAR 파일을 Docker 이미지로 빌드
                sh "docker build -t yunanash/backend:late
            }
            echo '백엔드 도커 이미지 빌드 완료!'
        }
    }

    stage('Push to Docker Hub-BE') {
```

```
            steps {
                echo '백엔드 도커 이미지를 Docker Hub에 푸시 시작!'
                withCredentials([usernamePassword(credentials
                    sh "docker login -u $DOCKER_USERNAME -p $
                }
                dir("./backend") {
                    sh "docker push yunanash/backend:latest"
                }
                echo '백엔드 도커 이미지를 Docker Hub에 푸시 완료!'
            }
        }

        stage('Deploy to EC2-BE') {
            steps {
                echo '백엔드 EC2에 배포 시작!'
                // 여기에서는 SSH 플러그인이나 SSH 스크립트를 사용하여
                sshagent(['youngseogi']) {
                    sh "docker rm -f backend"
                    sh "docker rmi yunanash/backend:latest"
                    sh "docker image prune -f"
                    sh "docker pull yunanash/backend:latest &&
                }
                echo '백엔드 EC2에 배포 완료!'
            }
        }


        stage('Add Front Env') {
            steps {
                echo '프론트 환경 설정'
                dir('./frontend') {
                    withCredentials([file(credentialsId: 'fro
                    sh 'cp ${env}  .env'
                    }
                }
            }
        }
```

```
        // stage('FE-Build') {
        //     steps {
        //         echo '프론트 빌드 및 테스트 시작!'
        //         dir("./frontend") {
        //             // sh 'rm -rf node_modules package-loc
        //             sh "npm ci"
        //             sh "npm run build"
        //         }
        //         echo '프론트 빌드 및 테스트 완료!'
        //     }
        // }

        stage('Build Front Docker Image') {
            steps {
                echo '프론트 도커 이미지 빌드 시작!'
                dir("./frontend") {
                    // 빌드된 파일을 Docker 이미지로 빌드
                    sh "docker build -t yunanash/frontend:lat
                }
                echo '프론트 도커 이미지 빌드 완료!'
            }
        }

        stage('Push to Docker Hub-FE') {
            steps {
                echo '프론트 도커 이미지를 Docker Hub에 푸시 시작!'
                withCredentials([usernamePassword(credentials
                    sh "docker login -u $DOCKER_USERNAME -p $
                }
                dir("frontend") {
                    sh "docker push yunanash/frontend:latest"
                }
                echo '프론트 도커 이미지를 Docker Hub에 푸시 완료!'
            }
        }

        stage('Deploy to EC2-FE') {
```

```
            steps {
                echo '프론트 EC2에 배포 시작!'
                // 여기에서는 SSH 플러그인이나 SSH 스크립트를 사용하여
                sshagent(['youngseogi']) {
                    sh "docker rm -f frontend"
                    sh "docker rmi yunanash/frontend:latest"
                    sh "docker image prune -f"
                    sh "docker pull yunanash/frontend:latest
                }
                echo '프론트 EC2에 배포 완료!'
            }
        }

    }

    post {
        success {
            echo '파이프라인이 성공적으로 완료되었습니다!'
            script {
                def Author_ID = sh(script: "git show -s --pre
                def Author_Name = sh(script: "git show -s --p
                mattermostSend (
                    color: '#D0E0E3',
                    icon: "https://jenkins.io/images/logos/je
                    message: "빌드 성공: ${env.JOB_NAME} #${env
                )
            }
        }
        failure {
            echo '파이프라인이 실패하였습니다. 에러를 확인하세요.'
            script {
                def Author_ID = sh(script: "git show -s --pre
                def Author_Name = sh(script: "git show -s --p
                mattermostSend (
                    color: '#D0E0E3',
                    icon: "https://4.bp.blogspot.com/-52EtGjE
                    message: "빌드 실패: ${env.JOB_NAME} #${env
                )
```

```
            }
        }
    }
}
```

Dockerfile : Frontend

- 설정 파일 위치

```
/frontend/Dockerfile
```

- Dockerfile

```
# 첫 번째 스테이지: 빌드 환경
FROM node:alpine as builder
WORKDIR /frontend
COPY package*.json ./
RUN npm install
COPY ./ ./
RUN npm run build

# 두 번째 스테이지: 실행 환경
FROM node:alpine
WORKDIR /app
COPY --from=builder /frontend/dist /app
COPY --from=builder /frontend/package*.json /app/
COPY --from=builder /frontend/node_modules /app/node_modules
EXPOSE 5173
CMD ["npm", "run", "dev"]
```

Dockerfile : Backend

- 설정 파일 위치

```
/backend/Dockerfile
```

- Dockerfile

```
FROM openjdk:17-jdk-alpine
EXPOSE 8000
ARG JAR_FILE=build/libs/*.jar
COPY ${JAR_FILE} app.jar
ENV TZ=Asia/Seoul
ENTRYPOINT ["java","-jar","/app.jar"]
```

2. 배포 방법

1. EC2 내부 방화벽 설정

포트 허용

```
sudo ufw allow 8080/tcp # ssh는 tcp 프로토콜만 허용해야함
sudo ufw status  # 방화벽 포트 상태 확인
sudo ufw deny 8080/tcp
```

2. 도커 설치

업데이트 및 HTTP 패키지 설치

```
sudo apt update
sudo apt-get install -y ca-certificates \
 curl \
 software-properties-common \
 apt-transport-https \
 gnupg \
 lsb-release
```

GPG키 및 저장소 추가

```
sudo mkdir -p /etc/apt/keyrings
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | s
udo gpg --dearmor -o /etc/apt/keyrings/docker.gpg
echo \
"deb [arch=$(dpkg --print-architecture) signed-by=/etc/apt/
keyrings/docker.gpg] https://download.docker.com/linux/ubun
tu \
$(lsb_release -cs) stable" | sudo tee /etc/apt/sources.lis
t.d/docker.list > /dev/null
```

설치 가능 버전 확인

```
apt-cache madison docker-ce
```

도커 설치

```
sudo apt update
sudo apt-get install docker-ce=<VERSION_STRING> docker-ce-c
li=<VERSION_STRING> containerd.io
```

도커 확인

```
sudo docker run hello-world # 또는
sudo docker version
```

3. 젠킨스 설치

Docker 컨테이너에 마운트 할 볼륨 디렉토리 설치

```
cd /home/ubuntu && mkdir jenkins-dat a
```

포트 설정

```
sudo ufw allow 9090/tcp
sudo ufw reload
sudo ufw status
```

도커로 젠킨스 컨테이너 생성 및 구동

```
sudo docker run -d -p 9090:8080 -v /home/ubuntu/jenkins-dat
a:/var/jenkins_home -v /var/run/docker.sock:/var/run/docke
r.sock --name jenkins jenkins/jenkins:lts
```

- Docker out of Docker (DooD) 설정을 위한 볼륨 지정

초기 패스워드 확인

```
sudo docker logs jenkins
```

- 중간에 나오는 패스워드 확인

```
************************************************************
************************************************************
************************************************************
************************************************************
****************************
casjije2ij10ue9qwdjsvogh4893uq2ejoadncvhw23y89q
************************************************************
************************************************************
************************************************************
************************************************************
****************************
```

4. Git 설치 및 프로젝트 셋팅

Git 설치

```
sudo apt-get install git
sudo apt install git
```

Git 버전 확인

```
sudo git --version
```

Git 계정 설정

```
sudo git config --global user.name ${유저 이름}
sudo git config --global user.email ${유저 이메일}
```

프로젝트 Clone

```
sudo git clone ${클론 받을 깃 레포지토리 url} ${다운받아올 폴더명}
```

5. HTTPS 적용

80, 443 포트 방화벽 해제

```
sudo ufw allow 80
sudo ufw allow 80/tcp
sudo ufw allow 443
sudo ufw allow 443/tcp
sudo ufw enable
sudo ufw status
```

기본 라이브러리 설치

```
sudo apt-get install -y build-essential
sudo apt-get install curl
```

Nginx 설치 및 conf 파일 작성

```
sudo apt-get install nginx
sudo vi /etc/nginx/conf.d/nginx.conf
```

위에 쓴 Nginx.conf 파일 작성

Certbot 설치 및 SSL 인증서 발급

```
sudo apt-get remove certbot
sudo snap install --classic certbot
# nginx 자동 설정
sudo certbot --nginx
```