포팅 메뉴얼

목차

1. 개발 환경

Frontend

- NextJS 15
- ReactJS 19
- TailwindCSS
- TypeScript
- Zustand
- React-Query
- MSW
- TensorflowJS

Backend

- SpringBoot 3.4.3
- MySQL
- Spring Data JPA

- Spring security

Infra

- EC2
- S3
- Docker
- Jenkins
- NignX

2. 설정 파일 및 환경 변수 설정

- application.yml

```
spring:
  datasource:
    url: ${SPRING_DATASOURCE_URL}
    username: ${SPRING_DATASOURCE_USERNAME}
    password: ${SPRING_DATASOURCE_PASSWORD}
    driver-class-name: ${SPRING_DATASOURCE_DRIVER_CLASS_NAME}
  jpa:
    hibernate:
      ddl-auto:  none
    properties:
      hibernate.dialect: org.hibernate.dialect.MySQL8Dialect
    defer-datasource-initialization: true
  security:
    oauth2:
      key:
        private: ${OAUTH2_JWT_PRIVATE_KEY_PATH}
        public: ${OAUTH2_JWT_PUBLIC_KEY_PATH}
      client:
        jwt:
          access-token-time: 100000
          refresh-token-time: 7200
        provider:
          kakao:
            authorization-uri: https://kauth.kakao.com/oauth/authorize
```

```
            token-uri: https://kauth.kakao.com/oauth/token
            user-info-uri: https://kapi.kakao.com/v2/user/me
            user-name-attribute: id
        registration:
          kakao:
            client-id: ${OAUTH2_KAKAO_CLIENT_ID}
            client-secret: ${OAUTH2_KAKAO_CLIENT_SECRET}
            client-authentication-method: client_secret_post
            authorization-grant-type: authorization_code
            redirect-uri: ${OAUTH2_KAKAO_REDIRECT_URI}
            scope:
              - profile_nickname
              - profile_image
  data:
    redis:
#      docker에선 redis-server로 변경
      host: localhost
      port: 6379
  jackson:
    serialization:
      wrap-root-value: false

logging:
  level:
    root: info
    sql: debug
    org.hibernate.SQL: debug
    #    org.hibernate.type: trace
    org.springframework.security: debug
    com.ssafy.speechfy: debug

openai:
  api:
    key: ${OPENAI_API_KEY}

aws:
  s3:
    bucket-name: ${AWS_S3_BUCKET_NAME}
```

```
cloudfront:
  domain: ${AWS_CLOUDFRONT_DOMAIN}
region: ${AWS_REGION}
access-key: ${AWS_ACCESS_KEY}
secret-key: ${AWS_SECRET_KEY}
```

3. EC2 연결

```
ssh -i "ssafy-key.pem" ubuntu@j12b105.p.ssafy.io
```

4. Docker 설치

- **우분투 시스템 패키지 업데이트**

  ```
  sudo apt-get update
  ```

  - 시스템의 패키지 목록을 최신 상태로 업데이트합니다.
- **필요한 패키지 설치**

  ```
  sudo apt-get install apt-transport-https ca-certificates curl gnupg-agen
  t software-properties-common
  ```

  - Docker 설치에 필요한 패키지를 설치합니다. 이 패키지들은 HTTPS를 통해 안전하게 패키지를 다운로드하거나, GPG 키와 저장소를 관리하는 데 사용됩니다.
- **Docker의 공식 GPG 키 추가**

  ```
  curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-ke
  y add -
  ```

  - Docker 공식 패키지가 위변조되지 않았음을 확인할 수 있도록 GPG 키를 추가합니다.
- **Docker의 공식 apt 저장소 추가**

  ```
  sudo add-apt-repository "deb [arch=amd64] https://download.docker.c
  om/linux/ubuntu $(lsb_release -cs) stable"
  ```

- Docker의 공식 apt 저장소를 우분투 시스템에 추가합니다. 이 저장소는 Docker와 관련된 패키지를 다운로드할 수 있는 출처입니다.

- **시스템 패키지 업데이트**

```
sudo apt-get update
```

  - Docker의 저장소가 추가되었으므로, 새로운 패키지 목록을 다시 업데이트합니다.

- **Docker 설치**

```
sudo apt-get install docker-ce docker-ce-cli containerd.io
```

  - Docker 엔진 및 관련 패키지를 설치합니다. `docker-ce`는 Docker Community Edition을 설치하는 패키지입니다.

- **Docker 설치 확인**
  - 도커 설치 확인

```
docker --version
```

  - **도커 실행상태 확인**

```
sudo systemctl status docker
```

    - Docker 서비스가 정상적으로 실행 중인지 확인합니다.

  - **도커 실행**

```
sudo docker run hello-world
```

    - Docker가 정상적으로 설치되었는지 확인하기 위해 **Hello World** 컨테이너를 실행합니다. 이 명령어는 Docker가 제대로 작동하는지 검증하는 간단한 방법입니다.

5. Docker Compose 설치

- Docker Compose 설치

```
sudo curl -L https://github.com/docker/compose/releases/download/1.22.0/d
```

- docker-compose에 권한을 부여

```
sudo chmod +x /usr/local/bin/docker-compose
```

- Version check

```
docker-compose version
```

- Docker Compose 파일 작성

6. Jenkins 세팅

- jenkins 컨테이너 실행

```
docker run -d --name jenkins --privileged -v /var/run/docker.sock:/var/run/do
```

- 확인

```
docker ps -a
```

- jenkins 컨테이너 접속

```
docker exec -it jenkins bash
```

- 초기 비밀번호 확인

```
cat /var/jenkins_home/secrets/initialAdminPassword
```

- webhook 설정
- 도커 설치

```
apt-get update
apt-get install -y docker.io
```

- 깃랩 연동

publish over SSH 플러그인 설치

```
git clone https://lab.ssafy.com/s12-ai-speech-sub1/S12P21B105
cd S12P21B105
git config --global credential.helper store
```

7. 무중단 배정

- 실행 쉘

```
pwd
# 1. 작업 디렉토리 이동 (예: Jenkins의 워크스페이스)
cd S12P21B105

# 3. GitHub에서 코드 클론 (SSH 또는 HTTPS 사용 가능)
git pull origin master

# 4. 클론한 디렉토리로 이동
cd client

# 5. Docker 이미지 빌드
docker build -t speechfy-frontend-app .

# 3000번 포트를 사용하는 컨테이너가 있는지 확인
CONTAINER_BLUE_FRONT=$(docker ps --filter "publish=3000" -q)
CONTAINER_GREEN_FRONT=$(docker ps --filter "publish=3001" -q)

# 3000번 포트를 사용하는 컨테이너가 있다면, 새 컨테이너 실행 후 기존 컨테이너 중지 '
if [ -n "$CONTAINER_BLUE_FRONT" ]; then
  echo "3001번 컨테이너에 배포 (BLUE → GREEN 배포)"
  # 3001번 포트에 새 컨테이너 실행
  docker run -d -p 3001:3000 --name speechfy-frontend-green speechfy-fror

  # 3000번 포트에서 실행 중인 기존 컨테이너 중지 및 삭제
  docker stop $CONTAINER_BLUE_FRONT || true
  docker rm $CONTAINER_BLUE_FRONT || true
else
  echo "3000번 컨테이너에 배포 (GREEN → BLUE 배포)"
  # 3000번 포트에 새 컨테이너 실행
```

```
  docker run -d -p 3000:3000 --name speechfy-frontend-blue speechfy-front

  # 3001번 포트에서 실행 중인 기존 컨테이너 중지 및 삭제
  docker stop $CONTAINER_GREEN_FRONT || true
  docker rm $CONTAINER_GREEN_FRONT || true
fi

cd ..
cd server
cd speechfy

# 5. Docker 이미지 빌드
docker build -t speechfy-backend-app .

# 8080번 포트를 사용하는 컨테이너가 있는지 확인
CONTAINER_BLUE_BACK=$(docker ps --filter "publish=8080" -q)
CONTAINER_GREEN_BACK=$(docker ps --filter "publish=8082" -q)

# 8080번 포트를 사용하는 컨테이너가 있다면, 새 컨테이너 실행 후 기존 컨테이너 중지 '
if [ -n "$CONTAINER_BLUE_BACK" ]; then
  echo "8082번 컨테이너에 배포 (BLUE → GREEN 배포)"
  # 8082번 포트에 새 컨테이너 실행
  docker run -d -p 8082:8080 --name speechfy-backend-green --env-file .en

  # 8080번 포트에서 실행 중인 기존 컨테이너 중지 및 삭제
  docker stop $CONTAINER_BLUE_BACK || true
  docker rm $CONTAINER_BLUE_BACK || true
else
  echo "8080번 컨테이너에 배포 (GREEN → BLUE 배포)"
  # 8080번 포트에 새 컨테이너 실행
  docker run -d -p 8080:8080 --name speechfy-backend-blue --env-file .env

  # 8082번 포트에서 실행 중인 기존 컨테이너 중지 및 삭제
  docker stop $CONTAINER_GREEN_BACK || true
  docker rm $CONTAINER_GREEN_BACK || true
fi

docker ps
```

8. Frontend 컨테이너

```
cd /var/jenkins_home/workspace/speechfy/client

docker build -t my-frontend-app .

docker run -d -p 3000:3000 my-frontend-app
```

```
FROM node:18-alpine
WORKDIR /app
RUN corepack enable && corepack prepare pnpm@latest --activate
COPY package.json pnpm-lock.yaml ./
RUN pnpm install --no-frozen-lockfile
COPY . .
RUN pnpm build
CMD ["pnpm", "start"]
```

9. Backend 컨테이너

```
# 1. Base 이미지 선택 (빌드용)
FROM openjdk:17-jdk-slim as builder


# 2. 작업 디렉토리 설정
WORKDIR /app

# 3. Gradle 빌드 환경 설정 (캐싱 최적화)
COPY gradlew .
COPY gradle gradle
COPY build.gradle settings.gradle ./
RUN chmod +x gradlew
RUN ./gradlew dependencies --no-daemon

# 4. 프로젝트 코드 복사 및 빌드 실행
COPY .env .env
COPY keys /app/resources/keys
```

```
COPY keys /app/keys
RUN ls -l /app/resources/keys
COPY src src
RUN ./gradlew build --no-daemon

# 5. 실행용 JDK 이미지 설정
FROM openjdk:17-jdk-slim

# 타임존 설정 추가
ENV TZ=Asia/Seoul
RUN ln -snf /usr/share/zoneinfo/$TZ /etc/localtime && echo $TZ > /etc/timezo



# 6. 실행 디렉토리 설정
WORKDIR /app

# 7. 빌드된 JAR 파일 복사
COPY --from=builder /app/build/libs/*.jar app.jar

# 8. Spring Boot는 기본 8080 포트에서 실행됨
EXPOSE 8080

# 9. 컨테이너 실행 명령어
ENTRYPOINT ["java", "-jar", "app.jar"]
```

10. Nginx 설정

```
events {
  worker_connections 1024;
}

http {

  upstream frontend {
    server j12b105.p.ssafy.io:3000 max_fails=1 fail_timeout=2s;
    server j12b105.p.ssafy.io:3001 max_fails=1 fail_timeout=2s;
  }
```

```
upstream backend {
  server j12b105.p.ssafy.io:8080 max_fails=1 fail_timeout=2s;
  server j12b105.p.ssafy.io:8082 max_fails=1 fail_timeout=2s;
}

server {
  listen 80;
  server_name j12b105.p.ssafy.io;

  return 301 https://$host$request_uri;
}
server{
  listen 443 ssl;
  server_name j12b105.p.ssafy.io;

  ssl_certificate /etc/letsencrypt/live/j12b105.p.ssafy.io/fullchain.pem;
  ssl_certificate_key /etc/letsencrypt/live/j12b105.p.ssafy.io/privkey.pem;

  ssl_protocols TLSv1.2 TLSv1.3;
  ssl_ciphers HIGH:!aNULL:!MD5;
  ssl_prefer_server_ciphers on;

  location / {
    proxy_pass http://frontend;
    proxy_set_header Host $host;
    proxy_set_header X-Real-IP $remote_addr;
    proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;

    proxy_connect_timeout 1s;
    proxy_read_timeout 10s;
    proxy_send_timeout 10s;
    proxy_next_upstream error timeout http_502 http_503 http_504;
  }

  location /api/ {
    proxy_pass http://backend;
    proxy_set_header Host $host;
```

```
    proxy_set_header X-Real-IP $remote_addr;
    proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;

    proxy_connect_timeout 2s;
    proxy_read_timeout 30s;
    proxy_send_timeout 30s;
    proxy_next_upstream error timeout http_502 http_503 http_504;
   }

   location /ws/ {
    proxy_pass http://backend;
    proxy_http_version 1.1;
    proxy_set_header Upgrade $http_upgrade;
    proxy_set_header Connection "Upgrade";
    proxy_set_header Host $host;
    proxy_set_header X-Real-IP $remote_addr;
    proxy_set_header X-Real-IP $remote_addr;
    proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;

    proxy_read_timeout 300s;  # ✅ 웹소켓 연결 유지 시간
    proxy_connect_timeout 5s;
   }
  }
}
```