

포팅 메뉴얼

[프로젝트 사용 툴](#)

[개발 환경](#)

[FrontEnd](#)

[BackEnd](#)

[AI](#)

[Server](#)

[포트 환경](#)

[설정 파일 및 환경변수](#)

[Backend](#)

[application.yml](#)

[application-secret.yml](#)

[.env파일](#)

[배포](#)

[Nginx](#)

[BackEnd](#)

[FrontEnd](#)

[AI](#)

[Mysql](#)

[Redis](#)

프로젝트 사용 툴

- 이슈 관리 : JIRA
- 형상 관리 : Gitlab
- 커뮤니케이션 : Notion, Mattermost
- 디자인 : Figma
- ERD: ERD Cloud
- CI/CD: Jenkins, Docker

개발 환경

FrontEnd

- Visual Studio Code
- Next.js 14.2.4
- React.js 18.3.1
- electron
- peerjs, sockjs, stompjs
- Tailwind CSS, Tailwind Variants
- eslint, prettier, husky

BackEnd

- IntelliJ
- Java 17
- Spring boot 3.3.2
- Redis

AI

- python 3.12
- FastAPI 0.111.0
- Whisper
- torch 2.3.0 + cu118
- OpenAI 1.35.12
- ChatGPT 40

Server

- ec2
- docker 24.0.7
- mysql 9.0.1
- s3
- jenkins 2.469
- nginx 1.27

포트 환경

Spring Boot	8080
fast-api	8000
Jenkins	7777
Mysql	3306
Redis	6379
Nginx	80, 443

설정 파일 및 환경변수

Backend

application.yml

```

spring:
  application:
    name: doctor-study
  profiles:
    include: secret
  datasource:
    driver-class-name: com.mysql.cj.jdbc.Driver

  jpa:
    database: mysql
    hibernate:
      ddl-auto: update
    properties:
      hibernate:
        show_sql: true
        format_sql: true
        use_sql_comments: true

  sql:
    init:
      mode: always

springdoc:
  swagger-ui:
    path: /swagger-ui.html
    operations-sorter: method
  api-docs:
    path: /api-docs
  default-consumes-media-type: application/json
  default-produces-media-type: application/json

audio-utils:
  upper-path: "audio/"

//배포환경과 로컬 환경을 profile을 통해 분리
---
## local
spring:
  config:
    activate:
      on-profile: local
    import: application-secret.yml

  data:
    redis:
      host: localhost
      port: 6379

```

application-secret.yml

⚠ git에 올리기 민감한 정보는 secret파일로 값 세팅하였음. application-secret 파일은 jenkins의 credential 설정을 통해 값을 세팅

```
spring:
  datasource:
    url: jdbc:mysql://localhost:{mysql 포트}/{mysql 스키마명}?serverTimezone=Asia/Seoul
    username: {db 계정}
    password: {db 패스워드}
  output:
    ansi:
      enabled: always

  redis:
    host: {reids host}
    port: {redis 포트}

  mail:
    host: {smtp host}
    port: {smtp 포트}
    username: {smtp 계정}
    password: {smtp 비밀번호}
    protocol: smtps

  jwt:
    secret: {jwt 암호화 키}
    access-token-expiration: {jwt access token 유효기간}
    refresh-token-expiration: {jwt refresh token 유효기간}

cloud:
  aws:
    s3:
      bucket: {s3 버킷 이름}
    credentials:
      accessKey: {s3 Accesskey}
      secretKey: {s3 SecretKey}
    region:
      static: {s3 region}
      auto: false
    stack:
      auto: false

fast-api:
  url: {fast-api ip}:{fast-api port}
```

.env파일

| docker-compose에서 사용할 환경변수들을 정의한 파일

```
MYSQL_ROOT_PASSWORD={mysql 루트계정 비밀번호}
```

배포

Nginx

pem키 인증서 발급

```
sudo apt-get update
sudo apt-get install software-properties-common
sudo add-apt-repository universe
sudo add-apt-repository ppa:certbot/certbot
sudo apt-get update
sudo apt-get install certbot python3-certbot-nginx

// certbot을 이용해 인증서 발급
sudo certbot certonly --standalone
```

docker-compose를 통해 nginx를 설정할 때 https 설정을 위한 pem 키 volume 설정

```
- /etc/letsencrypt/live/api.dr-study.kro.kr/fullchain.pem:/etc/nginx/certs/fullchain.pem:ro
- /etc/letsencrypt/live/api.dr-study.kro.kr/privkey.pem:/etc/nginx/certs/privkey.pem:ro
```

80포트 443포트 설정

```
http {
    upstream back_server {
        server spring:8080;
    }

    server {
        listen 80;
        server_name doctor-study;
        location / {
            proxy_pass http://back_server;
            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_set_header X-Forwarded-Proto $scheme;
        }
    }
}
```

```

server{
    listen 443 ssl;
    server_name doctor-study;

    ssl_certificate /etc/nginx/certs/fullchain.pem;
    ssl_certificate_key /etc/nginx/certs/privkey.pem;

    location / {
        proxy_pass http://back_server;
        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_set_header X-Forwarded-Proto $scheme;
    }
}

}

```

docker-compose

```

service:
  nginx:
    restart: always
    image: nginx:latest
    depends_on:
      - spring
    container_name: nginx_container
    volumes:
      - /etc/letsencrypt/live/api.dr-study.kro.kr/fullchain.pem:/etc/nginx/certs/fullchain.pem:ro
      - /etc/letsencrypt/live/api.dr-study.kro.kr/privkey.pem:/etc/nginx/certs/privkey.pem:ro
    ports:
      - "80:80"
      - "443:443" # Nginx가 443 포트(HTTPS 기본 포트)에서도 수신 대기하도록 PORTS 추가
    networks:
      - custom_network

```

BackEnd

DockerFile

```

# Base image
FROM bellsoft/liberica-openjdk-alpine:17

# Set working directory
WORKDIR /app

```

```
# Copy all files to the container
COPY . .

RUN apk update && apk add --no-cache ffmpeg

# Run Gradle build
RUN chmod +x ./gradlew

RUN ./gradlew clean build -x test

RUN ls -al ./build/libs

RUN cp ./build/libs/doctor-study-0.0.1-SNAPSHOT.jar app.jar

EXPOSE 8080


cmd ["java", "-jar", "app.jar"]
```

| docker-compose

```
service:
  spring:
    build: .
    restart: always
    ports:
      - "8080:8080"
    depends_on:
      - database
    container_name: spring_container

networks:
  - custom_network
```

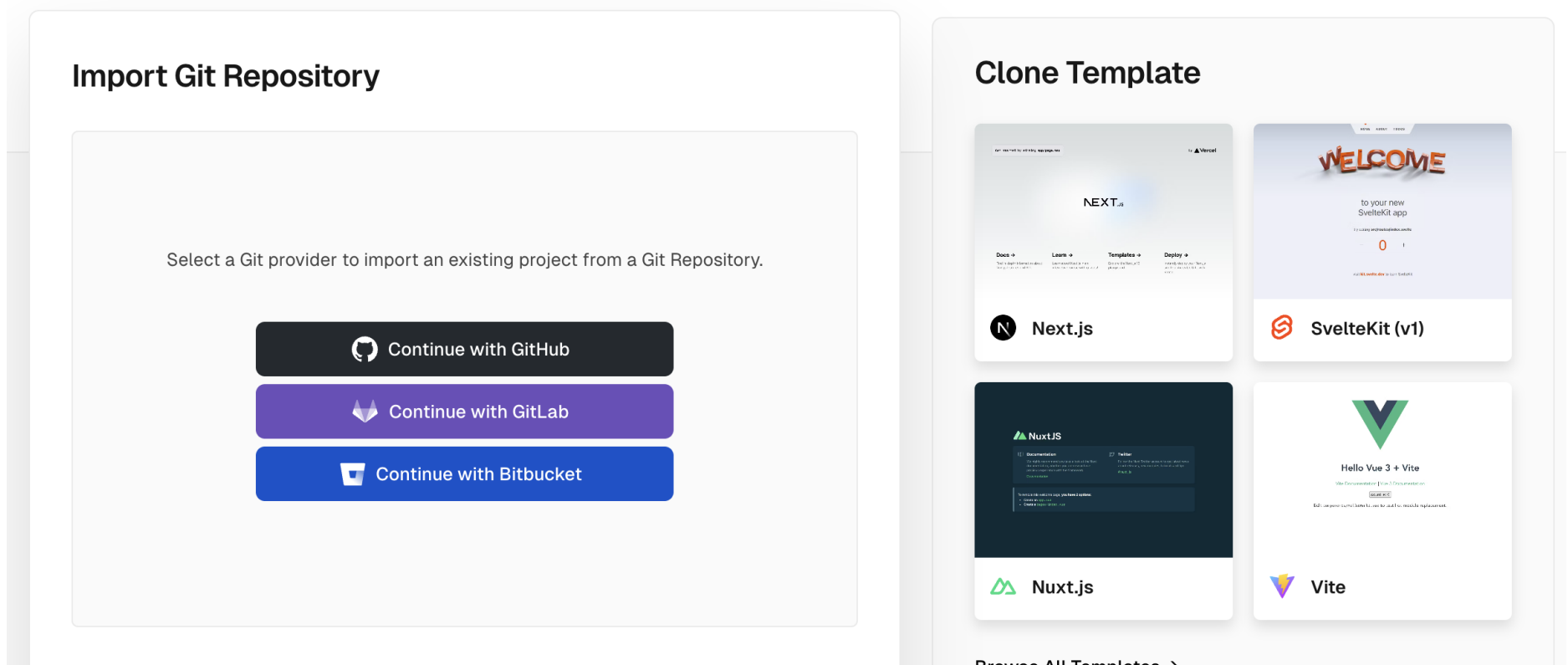
FrontEnd

 **프론트 배포는 EC2로 배포하는 것이 아닌 Vercel을 통해 자체 배포**

1. start deploy를 누른다.

Let's build something new.

To deploy a new Project, import an existing Git Repository or get started with one of our Templates.




2. Import Git Repository를 한다.
3. gitlab을 통해 import할 repo를 선택한다.
4. import한 repo에서 import 버튼을 누르면 Configure project 페이지에서 설정한다.

Configure Project

Project Name

Framework Preset

 Create React App

▼

Root Directory

./

Edit

> Build and Output Settings

> Environment Variables

Deploy

- Project Name: 배포된 프로젝트의 이름을 지정한다.
- Framework Preset: 사용할 프레임워크 또는 런타임 환경에서 필요한 빌드 및 배포 설정을 자동으로 구성해준다.
- Root Directory: 프로젝트의 루트 디렉토리를 적어준다.

▼ Build and Output Settings

Build Command ?

OVERWRITE

Output Directory ?

OVERWRITE

Install Command ?

OVERWRITE

- Build Command: 프로젝트의 빌드 스크립트를 실행하는 명령어를 적어준다.
- Output Directory: 빌드를 하고 결과물이 만들어지는 위치를 적어준다.
- Development Command: 로컬 개발 환경에서 사용하는 명령어를 적어준다.

AI

⚠

docker-compose가 아닌 dockerFile로 자체 도커라이징으로 컨테이너 생성

| DockerFile

베이스 이미지로 Python 3.9를 사용

FROM python:3.12

RUN

apt-get update && apt-get install -y \

portaudio19-dev \

&& apt-get clean \

&& rm -rf /var/lib/apt/lists/*

작업 디렉토리 설정

WORKDIR /app

종속성 파일 복사 및 설치

COPY requirements.txt requirements.txt

```
RUN pip install --upgrade pip && pip install --no-cache-dir -r requirements.txt

# 애플리케이션 코드 복사
COPY . .

# Uvicorn을 사용하여 애플리케이션 실행
CMD ["uvicorn", "main:app", "--host", "0.0.0.0", "--port", "8000"]
```

Mysql

| docker-compose

```
service:
  database:
  image: mysql
  restart: always
  ports:
    - "3306:3306"
  volumes:
    - /home/ubuntu/data/db/mysql:/var/lib/mysql/
    - /home/ubuntu/data/db/my.cnf:/etc/mysql/my.cnf
  container_name: mysql_container
  environment:
    - MYSQL_DATABASE=mzstop
    - MYSQL_ROOT_HOST=%
    - MYSQL_ROOT_PASSWORD='${MYSQL_ROOT_PASSWORD}'
  command: [ '--character-set-server=utf8mb4', '--collation-server=utf8mb4_unicode_ci' ]
  networks:
    - custom_network
```

Redis

| docker-compose

```
service:
  redis:
  image: redis
  container_name: redis_container
  ports:
    - "6379:6379"
  networks:
    - custom_network
```