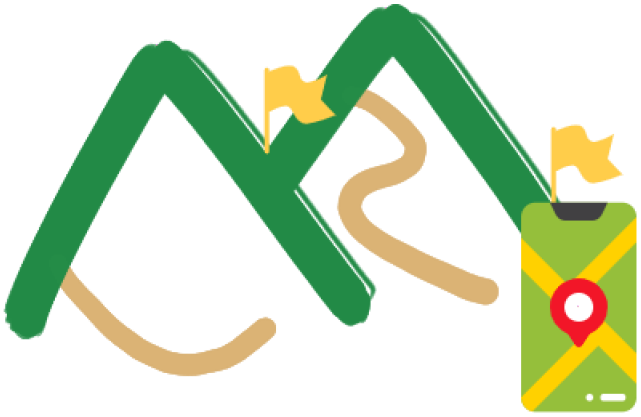

산삶

삼성 청년 SW 아카데미 구미캠퍼스 8기

특화 프로젝트 [2023.02.27(월) ~ 2023.04.07(금)]

포팅 매뉴얼

D205

팀장: 김성욱

팀원: 김유홍, 양동기, 윤선영, 이아현, 이진우

0. 목차

1. 프로젝트 개발 환경

협업 도구

1. 형상 관리: GitLab
2. 이슈 관리: Jira
3. 커뮤니케이션: MatterMost, Webex, Notion, Figma
4. 와이어프레임(UI / UX): Figma

개발 환경

1. OS: Windows 10
2. IDE:
    A. IntelliJ IDEA Ultimate Edition v.2022.3.2
    B. PyCharm Professional Edition v.2022.3.2
    C. Visual Studio Code v.1.74.2
    D. DBeaver Community Edition v.23.0.0.202303040621
3. DATABASE: MariaDB v.10.11.2

Back End

1. JAVA: OpenJDK v.11.0.15
2. PYTHON: Python v.3.9.0
3. FRAMEWORK:
    A. Spring Boot v.2.7.9
    B. Spring Boot Data JPA
    C. OAuth2.0

D. Spring Security

E. Lombok

F. Swagger v.3.0.0

G. JJWT v.0.11.5

Front End

1. Node JS: Node.js v.18.15.0
2. FRAMEWORK:
    A. @reduxjs/toolkit: v1.9.3
    B. @testing-library/jest-dom: v5.16.5
    C. @testing-library/react: v13.4.0
    D. @testing-library/user-event: v13.5.0
    E. @types/jest: v27.5.2
    F. @types/node: v16.18.16
    G. @types/react: v18.0.28
    H. @types/react-dom: v18.0.11
    I. @types/react-redux: v7.1.25
    J. axios: v1.3.4
    K. bootstrap: v5.2.3
    L. dotenv: v16.0.3
    M. history: v5.3.0
    N. react: v18.2.0
    O. react-dom: v18.2.0
    P. react-redux: v8.0.5
    Q. react-router-dom: v6.9.0
    R. react-scripts: v5.0.1
    S. react-select: v5.7.0

T. redux: v4.2.1

U. redux-ts: v4.3.0

V. styled-components: v5.3.9

W. swiper: v9.1.1

X. typescript: v4.9.5

Y. web-vitals: v2.1.4

Server

1. OS: Ubuntu v20.04 LTS
2. 웹 서버: Nginx v.1.18.0
3. 컨테이너: Docker v.20.10.21
4. 자동화 도구: Jenkins v.2.387.1

2. Back End 빌드 방법

Spring Boot

1. IntelliJ로 클론 받은 sansam 폴더를 엽니다. (project import 과정)
2. 하단의 Terminal을 열고, sansam 폴더 위치에서 './gradlew bootJar'를 실행하여 빌드를 진행합니다.
3. sansam>build>libs 위치에 있는 jar 파일을 확인할 수 있습니다.

Flask

별도의 빌드가 필요 없습니다. 아래는 실행 방법을 기술합니다.

1. PyCharm으로 클론 받은 flaskSansam 폴더를 엽니다. (project import 과정)
2. 하단의 Terminal을 열고, flaskSansam 폴더 위치에서 'python app.py'를 실행하여 서버를 구동합니다.

3. Front End 빌드 방법

ReactJS

1. Visual Studio Code로 클론 받은 front 폴더를 엽니다. (project import 과정)
2. 'ctrl+~'을 입력하여 터미널을 열고, 'npm install'을 입력하여 node-module을 설치합니다.
3. 'npm build'를 통해 빌드를 진행합니다.
4. front>build 폴더에 빌드된 결과물을 볼 수 있습니다.
5. node 서버 실시간 구동은 'npm start'를 통해 진행할 수 있습니다.

4. 배포 방법

AWS EC2 서버 환경 설정

1. Docker를 설치합니다. `sudo apt-get install docker.io`
2. Nginx를 설치합니다. `sudo apt-get install nginx`

Nginx

nginx.conf 파일을 열어 구체적인 설정 정보를 볼 수 있습니다.

1. Nginx 서비스를 중지 시킵니다.
2. Let's Encrypt를 설치합니다. `sudo apt-get install letsencrypt`
3. Cert Bot 무료 인증서를 사이트에 발급 받습니다.
4. Nginx 설정 파일인 nginx.conf 파일을 /etc/nginx/sites-available 경로에 생성하여 작성합니다. 이때, ReactJS, Spring Boot, Flask 포트 번호로 리다이렉션이 이루어질 수 있도록 적절한 주소를 사용합니다. https로의 SSL 인증서 처리 정보 역시 고려하여 적용합니다.
5. /etc/nginx/sites-available에 작성되어 있는 nginx.conf 파일을 /etc/nginx/sites-enabled 위치 경로에 링크 파일을 생성하여 연동합니다.
6. Nginx를 테스트 및 재시작하여 서비스를 시작합니다.

Jenkins

설치의 편의를 위해 Docker Container로 Jenkins를 구동합니다.

1. Jenkins Docker image 다운로드를 진행합니다.
2. 다운받은 image로 container를 생성합니다. 이때, 외부 EC2의 docker의 socket과 Jenkins docker container 내부의 docker socket이 공유되도록 설정하여 생성합니다.
3. 생성한 container 주소로 접속합니다.
4. Jenkins 설치를 진행하고, 필수 플러그인을 설치합니다.

5. Jenkins pipeline 구동에 필요한 credentials을 추가합니다.
6. Jenkins pipeline 구동 과정에서 nodeJS의 warning을 error로 인식하지 않게 하기 위해 Jenkins 관리>Configure System 에서 Environment variables에 CI의 값을 false로 바꿔줍니다.
7. Dashboard > Jenkins 관리 > Global Tool Configuration: NodeJS 에서 NodeJS 사용 버전과 pipeline tool 이름을 지정해 줍니다.

MariaDB

개발 환경 일치화와 배포의 변의를 위해 Docker Container로 MariaDB를 구동합니다.

Dockerfile 내 ENV로 암호를 기재하여 image를 빌드할 경우, image layer에서 암호가 노출되므로 EC2 내부 환경에선 암호를 필히 변경하여 적용해야 합니다.

1. Dockerfile을 작성하고, 초기에 적용할 sql 파일을 작성하여 파일 경로를 Dockerfile에 기재한대로 배치합니다.
2. Dockerfile이 위치한 경로에서 Docker image를 생성합니다.
3. MariaDB의 전용 docker volume을 생성해 줍니다.
4. 생성된 image를 기반으로 Docker container를 생성합니다. 생성해 둔 docker volume에 연결하도록 유의하며 진행합니다.

ReactJS

ReactJS는 내장된 별도의 서버가 없으므로 Nginx 기반의 Docker container로 구현하여 배포합니다.

각 Dockerfile, nginx.conf, JenkinsFile_Frontend 파일을 GitLab에서 열어 구체적인 동작 원리를 확인할 수 있습니다.

1. Dockerfile을 Nginx 기반으로 작성합니다.
2. Nginx 환경설정 파일인 nginx.conf를 작성합니다.
3. Jenkins 자동 CI/CD를 위해 앞서 작성한 Dockerfile과 nginx.conf를 git에 push 합니다.
4. Jenkins 자동 CI/CD에 사용되는 pipeline을 Jenkinsfile로 작성하고, git에 push 합니다.

5. Jenkins에서 Front End 자동 빌드를 위해 Item을 pipeline 형식으로 생성합니다.
6. GitLab의 dev/fe branch의 merge와 push를 감지하기 위해 GitLab에서 Webhook을 설정합니다.
7. Webhook trigger가 발생하면 Item이 동작하도록 Jenkins 설정에서 지정해 줍니다.
8. Jenkins Item 설정에서 앞서 작성한 Jenkinsfile을 git SCM으로 읽어내도록 설정합니다.
9. 이후 GitLab의 dev/fe branch에 merge 혹은 push가 감지될 때 마다 Jenkins가 자동 CI/CD를 진행합니다.

Spring Boot

Spring Boot는 내장된 Tomcat Apache 서버가 있으므로 OpenJDK-11 기반의 Docker container로 구현하여 배포합니다.

각 Dockerfile, JenkinsFile_Backend 파일을 GitLab에서 열어 구체적인 동작 원리를 확인할 수 있습니다.

1. Dockerfile을 OpenJDK-11 기반으로 작성합니다.
2. Jenkins 자동 CI/CD를 위해 앞서 작성한 Dockerfile을 git에 push 합니다.
3. Jenkins 자동 CI/CD에 사용되는 pipeline을 Jenkinsfile로 작성하고, git에 push 합니다.
4. Jenkins에서 Back End 자동 빌드를 위해 Item을 pipeline 형식으로 생성합니다.
5. GitLab의 dev/be branch의 merge와 push를 감지하기 위해 GitLab에서 Webhook을 설정합니다.
6. Webhook trigger가 발생하면 Item이 동작하도록 Jenkins 설정에서 지정해 줍니다.
7. Jenkins Item 설정에서 앞서 작성한 Jenkinsfile을 git SCM으로 읽어내도록 설정합니다.
8. 이후 GitLab의 dev/be branch에 merge 혹은 push가 감지될 때 마다 Jenkins가 자동 CI/CD를 진행합니다.

Flask

Flask는 내장된 Flask 앱 서버가 있으므로 Python 3.9 기반의 Docker container로 구현하여 배포합니다.

각 Dockerfile, JenkinsFile_Flask 파일을 GitLab에서 열어 구체적인 동작 원리를 확인할 수 있습니다.

1. Dockerfile을 Python 3.9 기반으로 작성합니다.
2. Jenkins 자동 CI/CD를 위해 앞서 작성한 Dockerfile을 git에 push 합니다.
3. Jenkins 자동 CI/CD에 사용되는 pipeline을 Jenkinsfile로 작성하고, git에 push 합니다.
4. Jenkins에서 Back End 자동 빌드를 위해 Item을 pipeline 형식으로 생성합니다.
5. GitLab의 dev/be branch의 merge와 push를 감지하기 위해 GitLab에서 Webhook을 설정합니다.
6. Webhook trigger가 발생하면 Item이 동작하도록 Jenkins 설정에서 지정해 줍니다.
7. Jenkins Item 설정에서 앞서 작성한 Jenkinsfile을 git SCM으로 읽어내도록 설정합니다.
8. 이후 GitLab의 dev/be branch에 merge 혹은 push가 감지될 때 마다 Jenkins가 자동 CI/CD를 진행합니다.