포팅메뉴얼

1. 사용 도구

- 이슈 관리 : Jira
- 형상 관리 : GitLab
- 커뮤니케이션 : Notion, MatterMost
- 디자인 : Figma
- CI/CD : Jenkins

2. 개발 도구

- Visual Studio Code : 1.85.1
- Intellij : 2022.3.2 (Ultimate Edition)
- MobaXTerm
- Docker Desktop

3. 개발 환경

- 상세 내용

Frontend

Node.js	`20.11.1`
npm	`10.2.4`
vite	`5.1.4`
type-script	`5.2.2`
React	`18.2.0`
Axios	`1.6.8`
Jotai	`2.7.1`
react-query	`3.39.3`

Backend

Java

Java	azul-17 Azul Zulu version 17.0.10
Spring Boot	3.2.3
gradle	gradle-8.5-bin

Python

Python	3.9.10
Django	4.1.13
Pandas	2.21
Numpy	1.26.4
Torch	2.2.1
scikit-learn	1.4.1.post1

Server

AWS S3	
AWS EC2	CPU : Intel(R) Xeon(R) CPU E5-2686 v4 @ 2.30GHz /dev/root : 311G
RAM	15GB
OS	Ubuntu

Service

MySQL	8.0.36
NginX	1.18.0
Jenkins	2.449
Docker	25.0.4
Ubuntu	Ubuntu 20.04 LTS
MongoDB	5.0.26 Atlas

4. 환경변수 형태

- Frontend
  - package.json

```
{
  "name": "package",
  "private": true,
  "version": "0.0.0",
  "type": "module",
  "scripts": {
    "dev": "vite",
    "build": "tsc && vite build",
    "lint": "eslint . --ext ts,tsx --report-unused-disable-directives --max-warn
ings 0",
    "preview": "vite preview"
  },
  "dependencies": {
    "@emotion/react": "^11.11.4",
    "@emotion/styled": "^11.11.0",
    "@mui/icons-material": "^5.15.11",
    "@mui/material": "^5.15.12",
    "@types/d3": "^7.4.3",
```

```
    "@types/react-joyride": "^2.0.5",
    "@types/react-modal": "^3.16.3",
    "@types/swiper": "^6.0.0",
    "axios": "^1.6.8",
    "base-64": "^1.0.0",
    "d3": "^7.9.0",
    "jotai": "^2.7.1",
    "js-confetti": "^0.12.0",
    "query-string": "^9.0.0",
    "react": "^18.2.0",
    "react-cookie": "^7.1.0",
    "react-dom": "^18.2.0",
    "react-financial-charts": "^2.0.1",
    "react-joyride": "^2.8.0",
    "react-modal": "^3.16.1",
    "react-query": "^3.39.3",
    "react-router-dom": "^6.22.2",
    "react-spinners": "^0.13.8",
    "styled-components": "^6.1.8",
    "swiper": "^11.0.7"
  },
  "devDependencies": {
    "@types/base-64": "^1.0.2",
    "@types/node": "^20.11.30",
    "@types/react": "^18.2.56",
    "@types/react-dom": "^18.2.19",
    "@types/react-native-dotenv": "^0.2.2",
    "@typescript-eslint/eslint-plugin": "^7.0.2",
    "@typescript-eslint/parser": "^7.0.2",
    "@vitejs/plugin-react-swc": "^3.6.0",
    "dotenv": "^16.4.5",
    "eslint": "^8.56.0",
    "eslint-config-prettier": "^9.1.0",
    "eslint-plugin-prettier": "^5.1.3",
    "eslint-plugin-react-hooks": "^4.6.0",
    "eslint-plugin-react-refresh": "^0.4.5",
    "prettier": "^3.2.5",
    "react-native-dotenv": "^3.4.11",
    "typescript": "^5.2.2",
    "vite": "^5.1.4",
    "vite-tsconfig-paths": "^4.3.1"
  }
}
```

- .env

```
VITE_REACT_APP_BASE_URL=https://j10c105.p.ssafy.io
VITE_REACT_APP_OAUTH_URL=https://j10c105.p.ssafy.io:8080/oauth2/authorization
```

- Python
  - .env

```
PROD_HOST = j10c105.p.ssafy.io
```

5. CI/CD 구축

EC2 환경설정

EC2 인스턴스에 설치한 것

- Docker
- Jenkins
- Mysql

Docker

- Frontend

```
# Frontend 이미지 pull
$ docker pull csw1511/neureka-frontend:latest

# Frontend 컨테이너 서비스 실행
$ docker run -d --name Frontend -p 5173:5173 csw1511/neureka-frontend:latest
```

- Backend

```
# Backend 이미지 pull
$ docker pull csw1511/neureka-backend:latest

# Backend 컨테이너 서비스 실행
$ docker run -d --name Backend -p 8080:8080 csw1511/neureka-backend:latest
```

- Python

```
# Python 이미지 pull
$ docker pull csw1511/neureka-python:latest

# Python 컨테이너 서비스 실행
$ docker run -d --name Python -p 8000:8000 csw1511/neureka-python:latest
```

- redis

```
# redis 이미지 pull
$ docker pull redis

# redis 컨테이너 실행
$ docker run -d --name redis -p 6379:6379 redis
```

- Jenkins

```
# Jenkins 전용 포트 개방
$ sudo ufw allow 20001

# Jenkins 컨테이너 실행
$ docker run -d -p 20001:8080 -v /home/ubuntu/jenkins-data:/var/jenkins_home \
  -v /var/run/docker.sock:/var/run/docker.sock -u root --name jenkins jenkins\
  /jenkins

# 젠킨스 환경설정
$ cd /home/ubuntu/jenkins-data

$ mkdir update-center-rootCAs

$ wget https://cdn.jsdelivr.net/gh/lework/jenkins-update-center/rootCA/\
update-center.crt \
  -O ./update-center-rootCAs/update-center.crt

$ sudo sed -i \
  's#https://updates.jenkins.io/update-center.json'\
  '#https://raw.githubusercontent.com/' \
  'lework/jenkins-update-center/master/updates/tencent/update-center.json#' \
  ./hudson.model.UpdateCenter.xml

$ sudo docker restart jenkins
```

- bareun.ai

```
# 바른 AI 도커 이미지 pull
$ docker pull bareunai/bareun:latest

# 디렉토리 생성
$ mkdir -p ~/bareun/var

# create a local folder for custom dictionaries.
$ mkdir -p bareun/var/custom-dict

# run bareun docker (Linux)
$ docker run \
   -d \
   --restart unless-stopped \
   --name bareun \
   --user $UID:$GID \
   -p 5757:5757 \
   -p 9902:9902 \
   -v ~/bareun/var:/bareun/var \
   bareunai/bareun:latest

# run bareun docker (Windows)
$ docker run `
    -d `
    --restart unless-stopped `
    --name bareun `
    --user $UID:$GID `
```

```
    -p 5757:5757 `
    -p 9902:9902 `
    -v bareun/var:/bareun/var `
   bareunai/bareun:latest

# API KEY 등록 (바른 AI 회원가입 시 자동 발급되어 있음)
$ docker exec bareun /bareun/bin/bareun -reg YOUR-API-KEY
```

Docker FIle

- Frontend

```
# 가져올 이미지를 정의
FROM node:20
# 경로 설정하기
WORKDIR /app
# package.json 워킹 디렉토리에 복사 (.은 설정한 워킹 디렉토리를 뜻함)
COPY package.json .
# 명령어 실행 (의존성 설치)
RUN npm install
# 현재 디렉토리의 모든 파일을 도커 컨테이너의 워킹 디렉토리에 복사한다.
COPY . .

# 각각의 명령어들은 한줄 한줄씩 캐싱되어 실행된다.
# package.json의 내용은 자주 바뀌진 않을 거지만
# 소스 코드는 자주 바뀌는데
# npm install과 COPY . . 를 동시에 수행하면
# 소스 코드가 조금 달라질때도 항상 npm install을 수행해서 리소스가 낭비된다.

# 3000번 포트 노출
EXPOSE 5173

# npm start 스크립트 실행
CMD ["npm", "run" , "dev"]

# 그리고 Dockerfile로 docker 이미지를 빌드해야한다.
# $ docker build .
```

- Backend

```
FROM gradle:8.5-jdk17 AS builder
WORKDIR /build

# 그래들 파일이 변경되었을 때만 새롭게 의존패키지 다운로드 받게함.
COPY stocker/build.gradle stocker/settings.gradle /build/
RUN gradle build -x test --parallel --continue > /dev/null 2>&1 || true

# 빌더 이미지에서 애플리케이션 빌드
COPY ./stocker /build
RUN gradle build -x test --parallel

# APP
FROM openjdk:17
WORKDIR /app
```

```
# 빌더 이미지에서 jar 파일만 복사
# COPY --from=builder /build/build/libs/*-SNAPSHOT.jar ./app.jar
COPY --from=builder /build/build/libs/stocker-0.0.1-SNAPSHOT.jar .

EXPOSE 8080

# root 대신 nobody 권한으로 실행
USER nobody
ENTRYPOINT [                                                  \
   "java",                                                    \
   "-jar",                                                    \
   "stocker-0.0.1-SNAPSHOT.jar"             \
   ]
```

- Python

```
# 가져올 이미지를 정의
FROM node:20
# 경로 설정하기
WORKDIR /app
# package.json 워킹 디렉토리에 복사 (.은 설정한 워킹 디렉토리를 뜻함)
COPY package.json .
# 명령어 실행 (의존성 설치)
RUN npm install
# 현재 디렉토리의 모든 파일을 도커 컨테이너의 워킹 디렉토리에 복사한다.
COPY . .

# 각각의 명령어들은 한줄 한줄씩 캐싱되어 실행된다.
# package.json의 내용은 자주 바뀌진 않을 거지만
# 소스 코드는 자주 바뀌는데
# npm install과 COPY . . 를 동시에 수행하면
# 소스 코드가 조금 달라질때도 항상 npm install을 수행해서 리소스가 낭비된다.

# 3000번 포트 노출
EXPOSE 5173

# npm start 스크립트 실행
CMD ["npm", "run" , "dev"]

# 그리고 Dockerfile로 docker 이미지를 빌드해야한다.
# $ docker build .
```

Jenkins File

```
pipeline {
    agent any

    environment {
        DOCKER_CREDENTIALS = 'dockerhub'
    }

    stages {
```

```
        stage('Build and Push Docker Images') {
            steps {
                script {
                    // 각 프로젝트 폴더에서 Docker 이미지를 빌드하고 Docker Hub에 푸시
                    sh "docker image ls"
                    buildAndPushImage('Frontend', DOCKER_CREDENTIALS)
                    sh "docker image ls"
                    buildAndPushImage('Backend', DOCKER_CREDENTIALS)
                    sh "docker image ls"
                    buildAndPushImage('Python', DOCKER_CREDENTIALS)
                    sh "docker image ls"
                }
            }
        }

        stage('Deploy with Docker') {
            steps {
                script {

                    // 1. 기존 작동중인 컨테이너 삭제 및 중지
                    // 기존 컨테이너 중지
                    sh 'docker stop Frontend Backend Python || true'
                    // 기존 컨테이너 삭제

                    sh 'docker rm Frontend Backend Python || true'
                    // 2. 기존 이미지를 Docker Hub에서 최신 버전으로 갱신
                    sh 'docker pull csw1511/neureka-frontend:latest'
                    sh 'docker pull csw1511/neureka-backend:latest'
                    sh 'docker pull csw1511/neureka-python:latest'

                    // 3. 새로 갱신된 이미지를 기반으로 컨테이너 실행
                    sh 'docker run -d --name Frontend -p 5173:5173 \n'
                    'csw1511/neureka-frontend:latest'

                    sh 'docker run -d \
                                            --name Backend \
                                            -p 8080:8080 \
                                            csw1511/neureka-backend:latest'

                    sh 'docker run -d \
                            --name Python \
                            -p 8000:8000 \
                            csw1511/neureka-python:latest'

                    // 4. 기존에 사용하던 이미지들을 삭제하기
                }
            }
        }

    }
}

def buildAndPushImage(projectName, credentials) {
    // 프로젝트 폴더로 이동하여 Docker 이미지 빌드 및 푸시
    def lowercaseProjectName = projectName.toLowerCase()
```

```
    dir("${projectName}") {
        // Docker 이미지 빌드
        sh "docker build -t csw1511/neureka-${lowercaseProjectName}:latest ."
        sh "echo ${projectName} ${lowercaseProjectName}"

        // Docker Hub에 이미지 푸시
        withCredentials([usernamePassword(credentialsId: credentials, usernameVariabl
e: 'DOCKER_USERNAME', passwordVariable: 'DOCKER_PASSWORD')]) {
            sh "docker login -u $DOCKER_USERNAME -p $DOCKER_PASSWORD"
            sh "docker push csw1511/neureka-${lowercaseProjectName}:latest"
        }
    }
}
```

Nginx site-available/default

```
server {
    listen 80;
    server_name j10c105.p.ssafy.io;
    return 301 https://$host$request_uri;
}
server {
    # listen [::]:443 ssl ipv6only=on;
      listen 443 ssl ;
            server_name j10c105.p.ssafy.io;
 ssl_certificate /etc/ssl/certificate.crt;
    ssl_certificate_key /etc/ssl/private.key;
 include /etc/letsencrypt/options-ssl-nginx.conf;
    ssl_dhparam /etc/letsencrypt/ssl-dhparams.pem;

    location / {
        proxy_pass http://j10c105.p.ssafy.io:5173;
        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;
        proxy_ssl_server_name on;
 }

    location /api/ {
        proxy_pass http://j10c105.p.ssafy.io:8080;
        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;
        }

 location /oauth2/authorization/ {
        proxy_pass http://j10c105.p.ssafy.io:8080;
        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;
```

```
        }
 location /lgoin/ {
        proxy_pass http://j10c105.p.ssafy.io:8080;
        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;
        }


    location /data/ {
        proxy_pass http://j10c105.p.ssafy.io:8000;
        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 80 ;
#    server_name j10c105.p.ssafy.io;
#    return 404;
#}
```