포팅 매뉴얼

구미 2반 3조 Docent

이동하, 이아현, 이지예, 이해솜, 임유정, 조재완

목차

I. 개요

1. 프로젝트 개요

"모나리자를 보러 루브르 박물관까지?"

"전시회보러 서울까지 가기에는 너무 멀다..."

우리나라는 대부분의 문화 예술 전시회들이 수도권에 편중되어 있어 지방에 거주하는 사람들은 시간적, 공간적 제약으로 인해 접근이 쉽지 않습니다. 실제로 2020년 국토교통부 조사에 따르면, 서울에서 문화공연시설은 2.08km 이내에서 접할 수 있는 반면, 강원도에서는 문화공연시설을 13.32km 안에서 찾을 수 있다고 합니다. 온라인으로라도 명화를 감상하려고 하면 실제 명화를 감상하는 느낌보다는 단순 사전처럼 정보 제공하는 서비스가 많습니다.

FY:UM(피움) 서비스는 시공간에 구애 받지 않으면서도 실제로 미술관에 방문한 것처럼 현실감 있게 명화를 감상할 수 있고, 직접 나의 전시회를 열어 다른 사람들과 공유하는 서비스입니다. 저희 DOCENT는 FY:UM(피움) 서비스를 이용하는 누구든 문화 예술에 보다 쉽게 다가갈 수 있도록 돕고자 합니다.

2. 주요 기술

Backend - Spring

Springboot Gradle 7.4
Spring Data JPA
Springframework 2.7.11
Spring Security
Spring Web
Lombok
jjwt 0.9.1
AWS S3

Frontend - React

axios 1.3.6
lottie-react 2.4.0
react 18.2.0
react-dom 18.2.0
react-lazy-load-image-component 1.5.6
react-redux 8.0.5
react-router-dom 6.10.0
react-unity-webgl 7.1.13
redux-persist 6.0.0
redux-saga 1.2.3
styled-components 5.3.9
typescript 4.9.5

Backend - DB

MariaDB 10.11.2

Backend - Django

Django 4.2
PyJWT 2.6.0
Numpy 1.24.3
Pandas 2.0.1
Pyarrow 11.0.0

VR

Unity 2019.4.18f
WebXr Export 0.16.0
WebXr Interactions 0.16.3
Oculus XR Plugin 1.6.1
VRTK v4 Tilia Package Importer 1.3
- Tilia CameraRig TrackedAlias Unity 2.4.3
- Tilia Indicators ObjectPointers Unity 2.2.3
- Tilia Interactions Interactables Unity 2.16.2
- Tilia Interaction PointerInteractors Unity 2.3.2
- Tilia Locomotor AxisMove Unity 2.0.33

TextMeshPro 2.1.6
ExtendRealityLtd/Zinnia.Unity 2.12.1
Editor Coroutines 1.0.0
Visual Studio Code Editor 1.2.3
Unity UI 1.0.0
Photon Pun2 Plus 2.42

CI/CD

AWS EC2
- Ubuntu 20.04
- Docker 20.10.23

Jenkins 2.387.2
NGINX 1.18.0

3. 사용한 외부 서비스 목록

- 카카오 로그인
  - OAuth 기반 소셜 로그인 API 제공
  - https://developers.kakao.com/
- AWS S3
  - 그림 저장
  - https://aws.amazon.com/ko/s3/
- CLOVA Voice

- TTS 기능 제공
- https://www.ncloud.com/?language=ko-KR

- Imagga
  - 그림 분석 API 제공
  - https://imagga.com/
- chatGPT
  - 큐레이션 생성
  - https://openai.com/

II. 프론트 엔드

1. ignore된 파일 추가

.env

```
REACT_APP_KAKAO_API_KEY = {Kakao API key}

REACT_APP_API_BASE_URL = https://k8d203.p.ssafy.io

REACT_APP_REDIRECT_URL = https://k8d203.p.ssafy.io/oauth
```

2. Dockerfile, nginx.conf 파일 작성

Dockerfile

```
FROM node:18.16.0-alpine as builder

EXPOSE 3000

#WORKDIR /frontend/frontend

COPY . .

RUN npm install

RUN CI=false npm run build

#CMD ["npm", "start"]
```

```
FROM nginx:stable-alpine as nginx

RUN rm -rf /etc/nginx/conf.d/default.conf

COPY ./default.conf /etc/nginx/conf.d/default.conf


RUN rm -rf /usr/share/nginx/html/*

COPY --from=builder /build /usr/share/nginx/html


RUN apk add --no-cache bash


CMD ["nginx", "-g", "daemon off;"]
```

III. 백엔드 : Spring

1. ignore 된 파일 추가

application.yml

```
server:
  port: 1234
  servlet:
    context-path: /api
spring:
  profiles:
    active: server
  mvc:
    pathmatch:
      matching-strategy: ant-path-matcher
  servlet:
```

```
    multipart:
      maxFileSize: 40MB
      maxRequestSize: 100MB
  datasource:
    url: jdbc:mariadb://{DB주소}
    username: {DB 아이디}
    password: {DB 비밀번호}
  jpa:
    hibernate:
      ddl-auto: update
    properties:
      hibernate:
        show_sql: true
        format_sql: true
        default_batch_fetch_size: 100


cloud:
  aws:
    credentials:
      access-key: {AWS S3 액세스 키}
      secret-key: {AWS S3 시크릿 키}
    region:
      static: ap-northeast-2
    s3:
      bucket: {S3 버킷 이름}
    stack:
      auto: false
```

```
perfixS3: {AWS S3 주소}


kakao:
  client_id: {카카오 Client ID}
  redirect_uri: {Redirect URI}
jwt:
  key: {JWT 비밀키}


clova:
  client_id: {CLOVA Client ID}
  client_secret: {CLOVA Client Secret}


imagga:
  key: {Imagga Key}
  secret: {Imagga Secret}


gpt:
  key: {OpenAI API Key}


google:
  key: {Google API Key}
```

2. Dockerfile 작성

Dockerfile

```
FROM openjdk:11-jdk-slim


ARG JAR_FILE=build/libs/*.jar
```

```
COPY ${JAR_FILE} app.jar

EXPOSE 1234

ENTRYPOINT ["java", "-jar", "/app.jar", "-Dspring.profiles.active=server"]
```

IV. 백엔드 : Django

1. ignore 된 파일 추가

.env

```
NAME={DB 이름}
USER={DB ID}
PASSWORD={DB 비밀번호}
HOST={DB 주소}
```

2. Dockerfile 만들기

Dockerfile

```
FROM python:3.9.13
RUN pip install django
WORKDIR /usr/src/app

COPY . .

RUN pip install -r requirements.txt
CMD ["python3", "manage.py", "runserver", "0.0.0.0:8000"]
EXPOSE 8000
```

V. Unity

1. Ignore된 파일 추가

application.properties

```
server.port=9998

server.servlet.context-path=/unityProxy



imagga.key={imagga 키}

immaga.secret={imagga 시크릿키}

gpt.key={gpt 키}

google.key={google 키}

clova.id={CLOVA Client ID}

clova.secret={CLOVA Client Secret}
```

2. Docker파일 작성

Dockerfile

```
FROM nginx:stable-alpine as nginx

RUN rm -rf /etc/nginx/conf.d/default.conf

COPY ./default.conf /etc/nginx/conf.d/default.conf



RUN rm -rf /usr/share/nginx/html/*

COPY /build /usr/share/nginx/html



RUN apk add --no-cache bash
```

```
CMD ["nginx", "-g", "daemon off;"]
```

VI. 서버 세팅

1. EC2 세팅

1. TimeZone 설정

```
SET GLOBAL time_zone='ASIA/SEOUL';

SET time_zone='+09:00';

flush privileges;

date
```

2. EC2 서버에 docker 설치

1. apt를 이용하여 docker를 설치할 예정이라 apt를 update 합니다.

```
sudo apt update
```

2. docker 설치에 필요한 패키지들을 설치합니다.

```
sudo apt install apt-transport-https ca-certificates curl software-properties-common
```

3. curl을 이용, 도커를 설치하기 위한 내용을 다운로드 받고, apt 기능을 위한 리스트에 추가합니다.

```
curl -fsSL https://download.docker.com/linux/ubuntu/gpg ¦ sudo apt-key add -
```

4. ubuntu 버전에 맞는 docker를 다운로드 할 수 있도록 repository 리스트에 추가합니다.

```
sudo add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/ubuntu bionic
stable"
```

5. apt update를 실행합니다.

```
sudo apt update
```

6. docker-ce를 설치합니다.(커뮤니티 버전)

```
apt-cache policy docker-ce
```

```
sudo apt install docker-ce
```

7. systemctl 명령어를 통해 도커 엔진 구동 상태 확인

```
sudo systemctl status docker
```

3. EC2 서버에 MariaDB 설치

1. MariaDB 설치

```
sudo docker pull mariadb
```

2. 3305 포트로 실행

```
sudo docker run -p 3305:3305 --name mariadb -e MARIADB_ROOT_PASSWORD=[비밀번호] -d mariadb
```

4. DB bash 접속, 계정 생성

1. bash 접속

```
sudo docker exec -it mariadb bash
```

2. DB 버전확인

```
mysql --version
```

3. DB 접속

```
mysql -u root -p
```

4. 모든 DB, 테이블에 접속 가능한 계정 생성

```
USE mysql;

CREATE USER 'localhost'@'%' IDENTIFIED BY 'root';

GRANT ALL PRIVILEGES ON *.* TO 'localhost'@'%';

FLUSH PRIVILEGES;
```

5. 데이터베이스 생성

```
create database [데이터베이스명]
```

6. SSH접속과 외부 접속 허용

```
vi /etc/mysql/mariadb.conf.d/50-server.cnf
```

```
# bind-address = 0.0.0.0

# 추가 후 :wq 로 저장 후 나가기
```

7. MariaDB characterset 변경

```
# apt 업데이트

apt-get update



# vim 설치

apt-get install vim



vi /etc/mysql/my.cnf
```

8. *my.cnf*에 아래 내용 추가

```
[client]

default-character-set=utf8mb4



[mysql]

default-character-set=utf8mb4



[mysqld]

character-set-server=utf8mb4

collation-server=utf8mb4_unicode_ci

skip-character-set-client-handshake
```

5. AWS S3 Bucket 설정

1. 버킷 정책 설정

```
    {

    "Version": "2012-10-17",

    "Id": "Policy1683013722425",

    "Statement": [
```

```
        {
            "Sid": "Stmt1683013719537",
            "Effect": "Allow",
            "Principal": "*",
            "Action": [
                "s3:DeleteObject",
                "s3:GetObject",
                "s3:PutObject"
            ],
            "Resource": "arn:aws:s3:::fyum/*"
        }
    ]
}
```

6. Nginx Default 값

1. 서버 Default 값 설정

Server

```
server {
        client_max_body_size 10M;
        location /unityPage {
                alias /jenkins/workspace/unity/build;
                index index.html
                try_files $uri /index.html;
        }
        location /api{
                proxy_pass http://localhost:1234/api;
        }
```

```
        location /unityProxy {
                proxy_pass http://localhost:9998/unityProxy;
        }
        location /recomm {
                proxy_pass http://localhost:8000/recomm;
        }
        location / {
                proxy_pass http://localhost:3000;
        }
    listen 443 ssl; # managed by Certbot
    ssl_certificate /etc/letsencrypt/live/k8d203.p.ssafy.io/fullchain.pem; # managed by
Certbot
    ssl_certificate_key /etc/letsencrypt/live/k8d203.p.ssafy.io/privkey.pem; # managed by
Certbot
    # include /etc/letsencrypt/options-ssl-nginx.conf; # managed by Certbot
    # ssl_dhparam /etc/letsencrypt/ssl-dhparams.pem; # managed by Certbot
}
server {
    if ($host = k8d203.p.ssafy.io) {
        return 301 https://$host$request_uri;
    } # managed by Certbot
        listen 80;
        server_name k8d203.p.ssafy.io;
    return 404; # managed by Certbot
}
```

2. Nginx 실행

```
sudo systemctl start nginx
```

VII. 자동 배포 : Jenkins

1. Jenkins 플러그인 설치

Jenkins 관리 → 플러그인 관리 → Available plugins

- Git client plugin 4.2.0
- Generic Webhook Trigger Plugin 1.86.3
- GitLab 1.7.12

2. Jenkins 프로젝트 생성

1. 젠킨스 메인 페이지 → 새로운 Item → Pipeline project
2. Jenkins 관리 → Credentials → add credentials
   - Gitlab
3. webhook 설정

3. Backend 배포

Spring

```
pipeline {
    agent any

    stages {
        stage('git clone') {
            steps {
                echo '깃 클론을 시작할게요!'
                git branch: 'be', credentialsId: 'gitlab3', url:
'https://lab.ssafy.com/s08-final/S08P31D203.git'
                sh '''
                cp /secretdata/application.yml
/var/jenkins_home/workspace/backend/backend/src/main/resources
```

```
            '''
            echo '깃 클론을 완료했어요!'
        }
    }
    stage('build') {
        steps {
            echo '빌드를 시작할게요!'
            sh '''
            chmod +x /var/jenkins_home/workspace/backend/backend/gradlew
            cd backend
            /var/jenkins_home/workspace/backend/backend/gradlew clean bootJar
            '''
            echo '빌드를 완료했어요!'
        }
    }
    stage('make image') {
        steps {
            sh '''
            docker stop backend || true
            docker rm backend || true
            docker rmi backend || true
            docker build -t backend ./backend
            '''
            echo '이미지를 만들었어요!'
        }
    }
    stage('run container') {
        steps {
```

```
                sh '''
                docker run -d --name backend -p 1234:1234 backend
                '''
                echo '컨테이너를 가동시켜요!'
            }
        }
    }
}
```

Django

```
pipeline {
    agent any

    stages {
        stage('gitclone') {
            steps {
                echo 'git clone 시작'
                git branch: 'be-django', credentialsId: 'gitlab3', url:
'https://lab.ssafy.com/s08-final/S08P31D203.git'
                sh '''
                cp /secretdata/.env /var/jenkins_home/workspace/django/pServer
                '''
                echo 'git clone 종료'
            }
        }
        stage('make image') {
            steps {
                echo '이미지를 만들어요!'
```

```
                sh '''
                docker stop django ¦¦ true
                docker rm django ¦¦true
                docker rmi django ¦¦ true
                docker build -t django ./pServer
                '''
                echo '이미지를 만들었어요!'
            }
        }
        stage('run container') {
            steps {
                echo '컨테이너를 실행해요!'
                sh '''
                docker run -d --name django -p 8000:8000 django
                '''
                echo '컨테이너를 실행했어요!'
            }
        }
    }
}
```

4. Frontend 배포

React

```
pipeline {
    agent any

    stages {
```

```
        stage('git clone') {
            steps {
                echo '깃 클론을 시작할게요'
                git branch: 'fe', credentialsId: 'gitlab3', url:
'https://lab.ssafy.com/s08-final/S08P31D203.git'
                sh '''cp /secretdata/frontend/.env
/var/jenkins_home/workspace/frontend/frontend'''
                echo '깃 클론을 완료했어요'
            }
        }
        stage('make image') {
            steps {
                echo '이미지를 만들게요'
                sh '''
                docker stop frontend || true
                docker rm frontend || true
                docker rmi frontend || true
                docker stop builder || true
                docker rm builder || true
                docker rmi builder || true
                docker build -t builder --target=builder ./frontend
                docker build -t frontend --target=nginx ./frontend
                '''
                echo '이미지를 만들었어요'
            }
        }
        stage('run container') {
            steps {
                echo '컨테이너를 가동시켜요!'
```

```
                sh '''
                docker run -d --name frontend -p 3000:3000 -v
/jenkins/workspace/vr/vr:/usr/share/nginx/html/unity frontend
                '''
                echo '컨테이너를 가동시켰어요!'
            }
        }
    }
}
```

5. Unity 배포

Unity

```
pipeline {
    agent any

    stages {
        stage('git clone') {
            steps {
                echo '깃 클론을 시작할게요!'
                git credentialsId: 'gitlab3', url: '{git 레포 주소}'
                sh '''
                cp /var/jenkins_home/workspace/unity/index.html
/var/jenkins_home/workspace/unity/build
                cp /var/jenkins_home/workspace/unity/style.css
/var/jenkins_home/workspace/unity/build/TemplateData
                cp /var/jenkins_home/workspace/unity/progressLogo.Dark.png
/var/jenkins_home/workspace/unity/build/TemplateData
                cp /var/jenkins_home/workspace/unity/progressLogo.Light.png
/var/jenkins_home/workspace/unity/build/TemplateData
```

```
                '''

                echo '깃 클론을 완료 했어요!'

            }

        }

    }

}
```

UnityProxy

```
pipeline {

    agent any



    stages {

        stage('git clone') {

            steps {

                echo '깃 클론을 시작해요!'

                git branch: 'unityProxy', credentialsId: 'gitlab3', url:
'https://lab.ssafy.com/s08-final/S08P31D203.git'

                sh '''

                cp /secretdata/unityProxy/application.properties
/var/jenkins_home/workspace/unityProxy/unityProxy/src/main/resources

                '''

                echo '깃 클론을 완료했어요!'

            }

        }

        stage('jar build') {

            steps {

                echo '빌드를 시작할게요!'

                sh '''

                chmod +x /var/jenkins_home/workspace/unityProxy/unityProxy/gradlew
```

```
                    cd unityProxy
                /var/jenkins_home/workspace/unityProxy/unityProxy/gradlew clean bootJar
                '''
                echo '빌드를 완료했어요!'
            }
        }
        stage('make image') {
            steps {
                echo '이미지를 만들어요!'
                sh '''
                docker stop unityproxy ¦¦ true
                docker rm unityproxy ¦¦ true
                docker rmi unityproxy ¦¦ true
                docker build -t unityproxy ./unityProxy
                '''
                echo '이미지를 만들었어요!'
            }
        }
        stage('run container') {
            steps {
                echo '컨테이너를 가동시켜요!'
                sh '''
                docker run -d --name unityproxy -p 9998:9998 unityproxy
                '''
                echo '컨테이너를 가동시켰어요!'
            }
        }
    }
```

```
}
```