BItAMin 포팅 메뉴얼

1. 개요

1.1. 프로젝트 개요

- 프로젝트 명
    - BItAMin
- 프로젝트 소개
    - 우울감을 겪는 사람들을 위한 마음치유 서비스
- 주요 기능
    1. 집단 상담
    2. AI 운동
    3. 데일리 미션
    4. 정신건강 관리

1.2. 프로젝트 사용 도구

- 이슈 관리
  - JIRA
- 형상 관리
  - Gitlab
- API 테스트
  - Postman
- 커뮤니케이션
  - Notion
  - Mattermost
- 디자인
  - Figma
- 데이터베이스 설계
  - ERD Cloud
- 동영상 편집
  - 모바비

1.3. 개발 환경

- 프론트엔드
  - Visual Studio Code
- 백엔드
  - IntelliJ IDEA
- 인프라
  - MobaXterm
  - putty
- 데이터베이스
  - DataGrip

    - MySQL Workbench

1.4. 기술 스택

- 프론트엔드
    - React 18.3.1
    - JavaScript
    - TypeScript 5.2.2
    - HTML5
    - CSS3
    - Node.js 20.15.0
    - OpenVidu 2.3.0
- 백엔드
    - Java 17
    - SpringBoot 3.3.2
    - webSocket
    - Spring Data JPA
- 데이터베이스
    - MySQL 8.4.1
- 인프라
    - Docker
    - Jenkins
    - NginX
    - AWS EC2
    - AWS S3

1.5. 추가 기술 스택

- AI 운동 관련

    - Teachable Machine
    - Tensorflow PoseNet
- 화상회의 관련
    - OpenAI API

2. 빌드

2.1. 프론트엔드 빌드 방법

1. 버전
    a. React 18.3.1
    b. Nodejs 20.15.0
2. 라이브러리 설치

```
npm install --force
```

3. 빌드

```
npm run build
```

2.2. 백엔드 빌드 방법

1. 버전
    a. JAVA Open-JDK 17
    b. SpringBoot 3.3.2
    c. Gradle 8.8
2. 빌드

```
./gradlew build
```

2.3. 배포하기

1. 배포 환경
    a. AWS EC2 (Ubuntu 20.04.6)
    b. AWS S3
    c. docker 27.1.1
    d. jenkins 2.452.3
    e. openvidu 2.3.0
2. 배포 방법
    a. 설치
        i. docker 설치

```
# apt update
sudo apt update

# 패키지 설치
$sudo apt-get install apt-transport-https ca-certi
ficates curl gnupg-agent software-properties-commo
n

# gpg 설치 및 도커 저장소 key 저장
sudo curl -fsSL https://download.docker.com/linux/
ubuntu/gpg | sudo apt-key add -

# 도커 다운로드 및 레포지토리에 추가
sudo add-apt-repository "deb [arch=amd64] https://
download.docker.com/linux/ubuntu $(lsb_release -c
s) stable"

# 도커 설치
apt-cache policy docker-ce
sudo apt install docker-ce
```

        ii. openvidu 설치

```
sudo su

cd /opt

curl https://s3-eu-west-1.amazonaws.com/aws.openvi
du.io/install_openvidu_latest.sh | bash

# 설정 변경
cd openvidu

nano .env

# 설정 파일 변경 부분
# DOMAIN_OR_PUBLIC_IP= ip주소 입력
# OPENVIDU_SECRET= 사용할 secret key 입력
# CERTIFICATE_TYPE=letsencrypt
# HTTP_PORT=4444
# HTTPS_PORT=4443

# openvidu 실행
./openvidu start
```

iii. jenkins 설치

```
# port 설정 9090포트로 외부에서 접속
# volume(/var/jenkins_home)을 통해서 로컬과 연결
# sock를 통해서 인증
docker run -d --name jenkins \
   -p 9090:8080 -p 50000:50000 \
   -v /var/jenkins_home:/var/jenkins_home \
   -v /var/run/docker.sock:/var/run/docker.sock \
   --privileged \
   --user root \
   jenkins/jenkins:lts
```

b. CI/CD 환경 구축

i. nginx를 위해서 미리 네트워크 생성

```
docker network create app-network
```

ii. GitLab과 연결

1. gitlab에서 acess 토큰 발급
2. 해당 acess 토큰 jenkins credentials에 등록
    a. 이때 Username with password로 생성하여 자기 아이디 토큰 입력
3. jenkins에 item생성
4. git lab build trigger 설정
    a. push
5. git lab과 훅 걸기
    a. project hook에서 jenkins project의 url 넣기(build trigger 설정하는 곳에 존재)
    b. token 넣어주기 (build trigger 설정하는 곳에 존재)
    c. push event에 Regular expression에 ^test$ 이런식으로 입력해서 특정 branch 선택 가능

iii. nginx 설정

1. nginx 설정
    a. nginx 설정 파일 생성 (/etc/nginx/templates/nginx.conf.template)

```
server {
    listen 80;
    server_name i11b105.p.ssafy.io;

    location / {
        return 301 https://$host$request_uri;
    }
}

server {
    listen 443 ssl;
    server_name i11b105.p.ssafy.io;
```

```
    resolver 127.0.0.11;

    location / {
        proxy_pass http://react:80$request_uri;
        proxy_set_header X-Real-IP $remote_add
r;
        proxy_set_header X-Forwarded-For $proxy
_add_x_forwarded_for;
        proxy_set_header Host $http_host;
    }

    location /api {
        proxy_pass http://spring:8080;
        proxy_set_header X-Real-IP $remote_add
r;
        proxy_set_header X-Forwarded-For $proxy
_add_x_forwarded_for;
        proxy_set_header Host $http_host;
       # webSocket
        proxy_http_version 1.1;
        proxy_set_header Upgrade $http_upgrade;
        proxy_set_header Connection "upgrade";
        proxy_read_timeout 20m;
    }

       location /ws {
        proxy_pass http://spring:8080;
        proxy_set_header Host $host;
        proxy_set_header X-Forwarded-For $proxy
_add_x_forwarded_for;
        proxy_set_header X-Real-IP $remote_add
r;
        proxy_set_header X-Forwarded-Host $serv
er_name;
        proxy_set_header X-Forwarded-Proto $sch
eme;
        # webSocket
        proxy_http_version 1.1;
```

```
        proxy_set_header Upgrade $http_upgrade;
        proxy_set_header Connection "upgrade";
        proxy_read_timeout 20m;
    }

    ssl_certificate /etc/letsencrypt/live/i11b1
05.p.ssafy.io/fullchain.pem;
    ssl_certificate_key /etc/letsencrypt/live/i
11b105.p.ssafy.io/privkey.pem;
}
```

1. 443 포트로 들어오면 url을 바탕으로 proxy
    a. /api가 붙어 있는 경우 spring container의 8080 포트로 이동
    b. 아무것도 없는 경우 react container의 80 포트로 이동
    c. /ws가 붙어 있는 경우 웹소켓으로 연결
    d. docker내의 dns 서버를 활용하여 react,spring 컨테이너 찾아 주기
    e. 인증서를 사용하여 https 설정
2. 80포트로 들어온 경우
    a. 443 포트(https://domain.com)으로 보냄

iv. 백엔드 CI/CD 구축

1. 파이프 라인 구축

```
pipeline {
    agent any

    environment {
        SPRING_IMAGE = 'spring-app' // Spring B
oot 애플리케이션의 Docker 이미지 이름
        COMPOSE_FILE = 'docker-compose.yml' //
사용될 docker-compose 파일 이름
    }
    tools {
        gradle 'gradle' // Jenkins에서 설정한 Gra
dle 도구
```

```
    }

    stages {
        stage('Checkout') {
            steps {
                // Git 리포지토리에서 'deploy' 브랜
치를 체크아웃
                git branch: 'server-main',
                    url: 'https://lab.ssafy.co
m/s11-webmobile1-sub2/S11P12B105.git',
                    credentialsId: 'gitlab-acce
ss-token'
            }
        }

        stage('Add Env') {
            steps {
                dir('./Backend/BITAMIN') {
                    withCredentials([file(crede
ntialsId: 'secret-properties', variable: 'ke
y')]) {
                        sh 'chmod 755 src/main/
resources'
                        sh 'cp ${key} src/main/
resources/application-SECRET.properties'
                    }
                }
            }
        }

        stage('Build Spring Backend') {
            steps {
                script {
                    dir('./Backend/BITAMIN') {
                        sh 'chmod +x gradlew'
                        sh './gradlew build -x
test'
                    }
```

```
                }
            }
        }

        stage('Stop and Remove Existing Contain
ers') {
            steps {
                script {
                    // 기존 컨테이너 중지 및 제거
                    sh 'docker-compose -f ${COM
POSE_FILE} down'
                }
            }
        }

        stage('Remove Existing Docker Images')
{
            steps {
                script {
                    // 기존 Spring 이미지를 삭제
                    sh "docker rmi -f \$(docker
images -q ${SPRING_IMAGE}) || true"
                }
            }
        }

        stage('Build Docker Image Spring') {
            steps {
                script {
                    dir('./Backend/BITAMIN'){
                        sh 'docker build -t ${S
PRING_IMAGE} .'
                    }
                }
            }
        }

        stage('docker-compose up') {
```

```
            steps {
                script {
                    sh 'docker-compose -f ${COM
POSE_FILE} down'  // 기존 컨테이너 중지 및 제거
                    sh 'docker-compose -f ${COM
POSE_FILE} up -d' // 새로운 컨테이너 시작
                }
            }
        }
        stage('Nginx Restart') {
            steps {
                script {
                    sh 'docker restart nginx'
                }
            }
        }
    }

    post {
        always {
            // 빌드 후 정리 작업을 수행합니다.
            cleanWs()
        }
    }
}
```

1. push, merge가 들어옴
2. branch 이동
3. config 파일 가져오기
    a. config 파일 생성
    b. credential에 파일 등록 및 이름 설정 ⇒ 해당 이름을 key로 파일 찾기 가능
4. build
5. 생성된 build파일의 jar파일을 image화

a. docker file을 미리 설정 (이때 directory 시작은 git clone을 했을때 바로 있는 폴더)

b. docker file 코드

```
FROM openjdk:17-alpine // 사용할 버전
ARG JAR_FILE=/build/libs/BITAMIN-0.0.1-SNAPS
HOT.jar // build된 jar 파일로 이동
COPY ${JAR_FILE} app.jar //jar 파일을 app.jar
파일로 복사
ENTRYPOINT ["java","-jar","/app.jar"] // jav
a -jar app.jar로 실행
```

6. 해당 image를 Build ⇒ 이미지 생성
7. docker-compose를 내리고 생성된 image를 다시 실행

```
version: '3.8'

services:
  spring:
    container_name: spring
    image: spring-app
    networks:
      - app-network

networks:
  app-network:
    external: true
```

1. image를 실행 시켜 container 생성
2. 여기에서 network 까지 같이 연결해서 하나의 network에 넣어서 해당 container에 접근 가능하도록 만듬

8. nginx 재시작

v. 프론트 CI/CD 구축

1. 파이프라인 구축

```
pipeline {
    agent any

    environment {
        REACT_IMAGE = 'react-app'  // React 애플
리케이션을 위한 이미지 이름
        COMPOSE_FILE = 'docker-compose.yml'  //
사용할 Docker Compose 파일
    }

    stages {
        stage('Checkout') {
            steps {
                // Git 리포지토리 체크아웃
                git branch: 'client-main',  //
사용할 브랜치
                url: 'https://lab.ssafy.com/s11
-webmobile1-sub2/S11P12B105.git',
                credentialsId: 'gitlab-access-t
oken'  // Jenkins에 설정된 Credential ID
            }
        }

        stage('Env') {
            steps {
                dir("./bitamin") {
                    sh '''
                        touch .env
                        echo 'VITE_APP_KAKAO_KE
Y=665ebd0c4d1235a24f8cd7757e9a52ba' >> .env
                        echo 'VITE_APP_WEATHER_
KEY=aQ/KD9B2XVnmNv0SkIefiz7rV6Ccy78ElnPFBkXZLRQ
7jBbpWfCIBnp16ZEHqHC24e/AiNSPdfFIl66DEGReng=='
>> .env
                        echo 'VITE_APP_WEATHER_
URL=http://apis.data.go.kr/1360000/VilageFcstIn
foService_2.0/getVilageFcst' >> .env
                    '''
```

```
                }
            }
        }

        stage('Build Docker Image for React') {
            steps {
                script {
                    dir('./bitamin'){
                        // Docker 이미지 빌드
                        sh 'docker build -t ${R
EACT_IMAGE} -f dockerfile .'
                    }
                }
            }
        }

        stage('Docker Compose Up') {
            steps {
                script {
                    // Docker Compose를 사용해 서
비스 시작
                    sh 'docker-compose -f ${COM
POSE_FILE} down'  // 기존 서비스 중지
                    sh 'docker-compose -f ${COM
POSE_FILE} up -d'  // 새 서비스 시작
                    sh 'pwd'
                }
            }
        }

        stage('Nginx Restart') {
            steps {
                script {
                    sh 'docker restart nginx'
                }
            }
        }
```

```
    }
}
```

1. push,merge가 들어옴
2. 사용할 branch로 checkout
3. doker file을 실행시켜 image 생성

```
# 1단계: React 애플리케이션 빌드
FROM node:20.15.0 AS build

# 작업 디렉토리 설정
WORKDIR /app

# package.json 및 package-lock.json 복사
COPY package*.json ./

# 의존성 설치
RUN npm install --force

# 애플리케이션 코드 복사
COPY . .

# 애플리케이션 빌드
RUN npm run build

# 2단계: Nginx로 React 애플리케이션 제공
FROM nginx:alpine

# 빌드된 파일을 Nginx의 HTML 디렉토리로 복사
COPY --from=build /app/dist /usr/share/ngin
x/html

# Nginx 설정 파일을 복사
COPY default.conf /etc/nginx/conf.d/default.
conf

# 80번 포트 오픈
```

```
EXPOSE 80

# Nginx 서버 시작
CMD ["nginx", "-g", "daemon off;"]
```

1. node:20.15.0 이미지 선택 하여 build환경 설정
2. 디렉토리 선택
3. 사용할 패키지를 담은 파일 복사
4. 의존성 설치
5. react 애플리케이션 빌드
6. nginx를 활용하여 해당 애플리케이션 실행
    a. 기본 제공 html 코드를 복사
    b. nginx 설정 변경

```
server {
    listen       80;
    listen  [::]:80;
    server_name  localhost;

    #access_log  /var/log/nginx/host.a
ccess.log  main;

    location / {
        root   /usr/share/nginx/html;
        try_files $uri /index.html;
    }

    #error_page  404              /40
4.html;

    # redirect server error pages to t
he static page /50x.html
    #
    error_page   500 502 503 504  /50
x.html;
```

```
    location = /50x.html {
        root   /usr/share/nginx/html;
    }

    # proxy the PHP scripts to Apache
listening on 127.0.0.1:80
    #
    #location ~ \.php$ {
    #    proxy_pass   http://127.0.0.
1;
    #}

    # pass the PHP scripts to FastCGI
server listening on 127.0.0.1:9000
    #
    #location ~ \.php$ {
    #    root           html;
    #    fastcgi_pass   127.0.0.1:900
0;
    #    fastcgi_index  index.php;
    #    fastcgi_param  SCRIPT_FILENAM
E  /scripts$fastcgi_script_name;
    #    include        fastcgi_param
s;
    #}

    # deny access to .htaccess files,
if Apache's document root
    # concurs with nginx's one
    #
    #location ~ /\.ht {
    #    deny  all;
    #}
}
```

c. 80번 포트 오픈

d. 해당 포트로 nginx서버 실행

4. compose 파일을 통해서 container들 생성

```
version: '3.8'

services:
  react:
    container_name: react
    image: react-app
    networks:
      - app-network

networks:
  app-network:
    external: true
```

    1. react 컨테이너 생성
    2. network 연결
5. nginx 재시작

c. 포트 개방

i. 443 ⇒ https로 접속을 위해 개방

ii. 80 ⇒ http로 접속을 위해 개방

iii. 9090 ⇒ jenkins 접속을 위해 개방

iv. 4443, 4444 ⇒ openvidu를 위해 개방