포팅 매뉴얼

프로젝트 기술 스택

☐ Frontend

- React 18.2.0
- React Router 6.3.0
- Recoil 0.7.4
- Material UI 5.9.1
- Axios 0.27.2
- Typescript 4.7.4
- sweetalert2 11.4.24

☐ Backend

- openjdk 1.8.0_192
- Java Zulu 8.33.0.1
- eclipse 4.16.0 (2020-06)
- Spring Boot 2.7.1
- Spring Data JPA
- Spring Security 5.6.6
- JWT 0.11.2
- AWS EC2
- MySQL 5.7.37

☐ WebRTC

- Openvidu

☐ STT(Speech-to-Text)

- JavaScript window 객체

☐ CI/CD

- Docker version 20.10.17, build 100c701
- NGINX
- AWS EC2

☐ Docx file API

- Aspose words 22.7

☐ Email

- Spring Java MailSender

☐ 형태소 분석

- KOMORAN 3.3.4

BE 빌드

backend root 경로에서

`gradlew bootJar`

를 입력하면, build/libs 경로에 A301-0.0.1-SNAPSHOT.jar 라는 빌드 결과물이 생깁니다. 이를 토대로 서버에 배포하기 위한 도커라이징을 실행하면 됩니다.

BE 도커라이징을 위해 필요한 Dockerfile은 다음과 같습니다.

BE Dockerfile

```
FROM java:8
EXPOSE 8080
ARG JAR_FILE=build/libs/A301-0.0.1-SNAPSHOT.jar
COPY ${JAR_FILE} app.jar
ENTRYPOINT ["java", "-jar", "/app.jar"]
ENV TZ=Asia/Seoul
RUN apt-get install -y tzdata
```

아래와 같은 명령어를 통해 도커 이미지를 빌드한 뒤 Docker Hub에 푸쉬해줍니다.

Docker build command

```
docker build -t yeinrah/chronicler-back .
```

FE 빌드

frontend root 경로에서 프로젝트에 필요한 node_modules를 다운받기 위해

`npm i`

를 실행한 뒤 빌드 파일을 생성하기 위해

`npm run build`

를 실행합니다.

그리고 아래와 같은 Dockerfile 및 nginx.conf를 생성한 뒤

FE Dockerfile

```
# nginx 이미지를 사용합니다. 뒤에 tag가 없으면 latest 를 사용합니다.
FROM nginx

# root 에 app 폴더를 생성
RUN mkdir /app

# work dir 고정
WORKDIR /app

# work dir 에 build 폴더 생성 /app/build
RUN mkdir ./build

# host pc의 현재경로의 build 폴더를 workdir 의 build 폴더로 복사
ADD ./build ./build

# nginx 의 default.conf 를 삭제
RUN rm /etc/nginx/conf.d/default.conf

# host pc 의 nginx.conf 를 아래 경로에 복사
```

```
COPY ./nginx.conf /etc/nginx/conf.d

# 80 포트 오픈
EXPOSE 80

# container 실행 시 자동으로 실행할 command. nginx 시작함
CMD ["nginx", "-g", "daemon off;"]
```

nginx.conf

```
server {
    listen 80;
    location / {
        root    /app/build;
        index   index.html;
        try_files $uri $uri/ /index.html;
    }
}
```

아래와 같은 명령어로 도커 이미지로 빌드하고 Docker Hub에 푸쉬해줍니다.

Docker build command

```
docker build -t yeinrah/chronicler-front .
```

배포

원활한 서비스 구동을 위해 배포해야 하는 도커 컨테이너는

- Openvidu
- MySQL
- BE
- FE

총 4개입니다.

각 컨테이너를 pull하고 실행하기에 앞서 BE와 MySQL 컨테이너를 원활하게 연결해주기 위해서 커스텀 네트워크를 만들어서 적용시켜줘야 합니다.

```
docker network create chronicler
```

Openvidu

- 오픈비두를 배포하기 root 권한을 얻어야 함

`sudo su`

- 오픈비두를 설치하기 위해 권장되는 경로인 `/opt`로 이동

`cd /opt`

- 오픈비두 설치

`curl <https://s3-eu-west-1.amazonaws.com/aws.openvidu.io/install_openvidu_latest.sh> | bash`

- 설치 후 오픈비두가 설치된 경로로 이동

`$ cd openvidu`

- 도메인과 오픈비두와의 통신을 위한 환경설정

`$ vi .env`

```
DOMAIN_OR_PUBLIC_IP=i7a301.p.ssafy.io

OPENVIDU_SECRET=MY_SECRET

CERTIFICATE_TYPE=letsencrypt

LETSENCRYPT_EMAIL=user@example.com

HTTP_PORT=4442

HTTPS_PORT=4443
```

- 설정 후 오픈비두 서버 실행

`$ ./openvidu start`

Nginx 설정과 ssl 인증서 발급 및 적용

Openvidu 같은 경우, 카메라를 사용하기 위해서는 반드시 https로 이용해야 하기에 SSL 인증서를 발급받아야 합니다. 인증서 발급을 위해서는 도메인이 필요합니다.

먼저 nginx를 다운 받습니다.

```
# 설치
sudo apt-get install nginx
```

letsencrypt 설치를 위해 다음과 같은 순서로 명령어를 입력합니다.

`sudo apt-get install letsencrypt`

`sudo systemctl stop nginx`

`sudo letsencrypt certonly --standalone -d www제외한 도메인 이름`

이렇게 한 후, "Congratulations!"로 시작하는 문구가 보이면, 인증서 발급이 완료된 것입니다.

인증서 발급 후, /etc/nginx/sites-available로 이동한 후, 적절한 이름의 파일을 생성하여 다음과 같이 작성합니다.

```
server {

        location /{
                proxy_pass http://localhost:3000;
                proxy_set_header    Host                $http_host;
                proxy_set_header    X-Real-IP           $remote_addr;
                proxy_set_header    X-Forwarded-For     $proxy_add_x_forwarded_for;
                add_header      Content-Security-Policy "upgrade-insecure-requests";
        }

        location /api {
                proxy_pass http://localhost:8080/api;
                proxy_set_header    Host                $http_host;
                proxy_set_header    X-Real-IP           $remote_addr;
                proxy_set_header    X-Forwarded-For     $proxy_add_x_forwarded_for;
                add_header      Content-Security-Policy "upgrade-insecure-requests";

        }

        location /openvidu/ {
          proxy_pass https://i7a301.p.ssafy.io:4443;
          proxy_set_header Host $http_host;
          proxy_set_header X-Real-IP $remote_addr;
          proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        }


    listen 443 ssl; # managed by Certbot
    ssl_certificate /etc/letsencrypt/live/i7a301.p.ssafy.io/fullchain.pem; # managed by Certbot
    ssl_certificate_key /etc/letsencrypt/live/i7a301.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
        error_page 405 =200 $uri;
```

```
}

server {
    if ($host = i7a301.p.ssafy.io) {
        return 301 https://$host$request_uri;
    } # managed by Certbot

        listen 80;
```

그 후에 차례로 명령을 실행합니다.

```
sudo ln -s /etc/nginx/sites-available/chronicler /etc/nginx/sites-enabled/chronicler

# 다음 명령어에서 successful이 뜨면 nginx를 실행할 수 있다.
sudo nginx -t

sudo systemctl restart nginx
```

이렇게 실행하면, http로 80포트 접근시, 443 포트(https)로 리다이렉트 됩니다. 그리고 백엔드 url을 /api/**로 분기처리할 수 있습니다. `https://도메인주소` 로 접근하면 배포한 웹 페이지에 접속할 수 있게됩니다.

MySQL

```
docker pull yeinrah/mysql:5.7.37
```

```
docker run -d -p 3306:3306 --name mysql --network chronicler -e MYSQL_ROOT_PASSWORD=ssafy -e MYSQL_DATABASE=chronicler -e MYSQL_USER=A
```

아래와 같은 명령어로 mysql bash를 실행시켜 서버의 도커라이징된 MySQL에 접근해 컨트롤할 수 있습니다.

```
docker exec -it mysql bash

mysql -u A301 -p

[비밀번호 입력]

[DDL, DML, etc. 입력]
```

BE

```
docker pull yeinrah/chronicler-back
```

```
docker run -d -p 8080:8080 --name chronicler-back --network chronicler yeinrah/chronicler-back
```

FE

```
docker pull yeinrah/chronicler-front
```

```
docker run -d -p 3000:80 --name chronicler-front --network chronicler yeinrah/chronicler-front
```

외부 라이브러리 설정

회의록을 docx 파일 형태로 작성해주는 라이브러리인 Aspose와 작성된 회의록 파일을 이메일로 전송해주기 위해 외부 라이브러리를 연결하여 사용했습니다.

https://releases.aspose.com/words/java/

위 링크에서 Aspose jar파일을 다운받은 뒤 프로젝트 폴더에 넣고 이클립스에서 프로젝트 우클릭 → Build Path → Configure Build Path → Add JARs 에서 다운받은 jar파일을 선택하여 추가해줍니다.

그리고 build.gradle에 아래와 같이 추가해줍니다.

```
// aspose
  implementation fileTree(dir: 'asposejar', include: ['*.jar'])
```

이메일 전송을 위해서는 아래와 같이 build.gradle에 추가해준 뒤

```
// email
  implementation 'org.springframework.boot:spring-boot-starter-mail'
```

application.properties에 메일을 보내기 위한 host와 port 및 계정 설정을 해줍니다.

```
#it will be set build date by gradle. if this value is @build.date@, front-end is development mode
build.date=@build.date@
#server.port=8080
#server.address=localhost
server.servlet.contextPath=/

#database
spring.jpa.hibernate.naming.implicit-strategy=org.springframework.boot.orm.jpa.hibernate.SpringImplicitNamingStrategy
spring.jpa.hibernate.naming.physical-strategy=org.springframework.boot.orm.jpa.hibernate.SpringPhysicalNamingStrategy
spring.jpa.hibernate.ddl-auto=update
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQL57Dialect
spring.data.web.pageable.one-indexed-parameters=true
spring.datasource.url=jdbc:mysql://mysql:3306/chronicler?useUnicode=true&characterEncoding=utf8&serverTimezone=Asia/Seoul&zeroDateTime
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
spring.datasource.hikari.username=A301
spring.datasource.hikari.password=[A301 Password]
spring.mvc.pathmatch.matching-strategy=ant_path_matcher

spring.mail.host=smtp.gmail.com
spring.mail.port=587
spring.mail.username=chronicler321
spring.mail.password=[chronicler321@gmail.com Password]
spring.mail.properties.mail.smtp.starttls.enable=true
spring.mail.properties.mail.smtp.starttls.required=true
spring.mail.properties.mail.smtp.auth=true
```