



프로젝트 기술 스택

FE

Node.js

```
"node.js" : "^16.16.0",  
"npm" : "^8.11.0"
```

VSCode

```
"VSCode" : "^1.70.0",  
"Chromium" : "^100.0.4896.160",  
"node.js" : "^16.13.2",  
"v8" : "^10.0.139.17-electron.0",  
"OS" : "^Windows-NT x64 10.0.19044"
```

Vue

```
"name": "queant",  
"version": "0.1.0",  
"private": true,  
"scripts": {  
  "serve": "vue-cli-service serve",  
  "build": "vue-cli-service build",  
  "lint": "vue-cli-service lint"  
},  
"dependencies": {  
  "@ckeditor/ckeditor5-build-classic": "^35.0.1",  
  "@ckeditor/ckeditor5-vue": "^4.0.1",  
  "@fortawesome/fontawesome-svg-core": "^6.1.2",  
  "@fortawesome/free-brands-svg-icons": "^6.1.2",  
  "@fortawesome/free-regular-svg-icons": "^6.1.2",  
  "@fortawesome/free-solid-svg-icons": "^6.1.2",  
  "@fortawesome/vue-fontawesome": "^3.0.1",  
  "axios": "^0.27.2",  
  "bootstrap": "^5.1.3",  
  "bootstrap-vue": "^2.22.0",  
  "core-js": "^3.8.3",  
  "dotenv": "^16.0.1",  
  "email-validator": "^2.0.4",  
  "lodash": "^4.17.21",  
  "node-sass": "^7.0.1",  
  "password-validator": "^5.3.0",  
  "sass-loader": "^13.0.2",  
  "vue": "^3.2.13",  
  "vue-loader": "^17.0.0",  
  "vue-router": "^4.0.13",  
  "vuex": "^4.0.2",  
  "vuex-persistedstate": "^4.1.0"  
},  
"devDependencies": {  
  "@babel/core": "^7.12.16",  
  "@babel/eslint-parser": "^7.12.16",  
  "@vue/cli-plugin-babel": "~5.0.0",  
  "@vue/cli-plugin-eslint": "~5.0.0",  
  "@vue/cli-plugin-router": "~5.0.0",
```

```

"@vue/cli-service": "~5.0.0",
"eslint": "^7.32.0",
"eslint-plugin-vue": "^8.0.3",
"webpack": "^5.73.0"
},
"eslintConfig": {
  "root": true,
  "env": {
    "node": true
  },
  "extends": [
    "plugin:vue/vue3-essential",
    "eslint:recommended"
  ],
  "parserOptions": {
    "parser": "@babel/eslint-parser"
  },
  "rules": {}
},
"browserslist": [
  "> 1%",
  "last 2 versions",
  "not dead",
  "not ie 11"
]
}

```

BE

build.gradle

```

plugins {
    id 'org.springframework.boot' version '2.6.7'
    id 'io.spring.dependency-management' version '1.0.11.RELEASE'
    id 'java'
}

group = 'com.ssafy'
version = '0.0.1-SNAPSHOT'
sourceCompatibility = '1.8'

configurations {
    compileOnly {
        extendsFrom annotationProcessor
    }
}

repositories {
    mavenCentral()
}

dependencies {
    implementation 'org.springframework.boot:spring-boot-starter-web'
    implementation 'junit:junit:4.13.1'
    compileOnly 'org.projectlombok:lombok'
    developmentOnly 'org.springframework.boot:spring-boot-devtools'
    annotationProcessor 'org.projectlombok:lombok'
    testImplementation 'org.springframework.boot:spring-boot-starter-test'
    implementation group: 'org.json', name: 'json', version: '20210307'

    //DB
    implementation 'org.springframework.boot:spring-boot-starter-data-jpa'
    runtimeOnly 'org.mariadb.jdbc:mariadb-java-client'
    implementation 'mysql:mysql-connector-java:8.0.29'

    // Querydsl
    implementation 'com.querydsl:querydsl-jpa'
    annotationProcessor "com.querydsl:querydsl-apt:${dependencyManagement.importedProperties['querydsl.version']}:jpa"
    annotationProcessor "jakarta.persistence:jakarta.persistence-api"
}

```

```

annotationProcessor "jakarta.annotation:jakarta.annotation-api"

//Oauth
implementation 'org.springframework.boot:spring-boot-starter-security'
testImplementation 'org.springframework.security:spring-security-test'
implementation 'io.jsonwebtoken:jjwt-api:0.11.2'
implementation 'io.jsonwebtoken:jjwt-jackson:0.11.2'
runtimeOnly 'io.jsonwebtoken:jjwt-impl:0.11.2'

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

//Swagger
implementation "io.springfox:springfox-boot-starter:3.0.0"

//ModelMapper
implementation 'org.modelmapper:modelmapper:2.4.2'

// json parser
implementation 'org.json:json:20190722'

//test를 위한 프론트 템플릿 -> 추후 삭제
implementation 'org.springframework.boot:spring-boot-starter-thymeleaf'

// 로그에 남는 쿼리문 (?,?) 안에 채워서 보여줌, 배포시 삭제
implementation 'com.github.gavlyukovskiy:p6spy-spring-boot-starter:1.5.7'

//Jsoup (크롤링)
implementation group: 'org.jsoup', name: 'jsoup', version: '1.14.3'

//aws
implementation 'org.springframework.cloud:spring-cloud-starter-aws:2.2.6.RELEASE'
}

tasks.named('test') {
    useJUnitPlatform()
}

// clean task 실행시 QClass 삭제
clean {
    delete file('src/main/generated') // 인텔리제이 Annotation processor 생성물 생성 위치
}

```

application.properties

```

#DB
server.port=8000
spring.datasource.driver-class-name=org.mariadb.jdbc.Driver
spring.datasource.url=jdbc:mariadb://IP주소:포트번호/스키마이름
spring.datasource.username=DB이름
spring.datasource.password=DB패스워드

#JPA
spring.jpa.hibernate.ddl=update
spring.jpa.generate-ddl=true
spring.jpa.properties.hibernate.format_sql=true
spring.jpa.show-sql=true
spring.jpa.properties.hibernate.current_session_context_class=org.springframework.orm.hibernate5.SpringSessionContext
spring.jpa.properties.hibernate.jdbc.lob.non_contextual_creation=true
logging.level.org.hibernate.type.descriptor.sql=trace

#LOG
logging.level.root=info

#Email
spring.profiles.include=mail, API-KEY

#Swagger
spring.mvc.pathmatch.matching-strategy=ANT_PATH_MATCHER

#aws

```

```
cloud.aws.credentials.instance-profile=true
cloud.aws.stack.auto=false

#server
server.servlet.context-path=/api
```

application-mail.properties

```
mail.smtp.auth=true
mail.smtp.starttls.required=true
mail.smtp.starttls.enable=true
mail.smtp.socketFactory.class=javax.net.ssl.SSLSocketFactory
mail.smtp.socketFactory.fallback=false
mail.smtp.port=465
mail.smtp.socketFactory.port=465

#admin ?? ??? ?? id,password
AdminMail.id = 이메일 전송할 사람의 이메일 주소
AdminMail.password = 비밀번호(GMAIL의 경우 app 비밀번호를 발급받으시면 편리)
```

BE(Django)

Python == 3.10.5

requirements.txt

```
APScheduler==3.9.1
asgiref==3.5.2
beautifulsoup4==4.4.1
bs4==0.0.1
certifi==2022.6.15
charset-normalizer==2.1.0
Django==4.0.6
django-apscheduler==0.6.2
html-table-parser==0.1.0
idna==3.3
lxml==4.9.1
PyMySQL==1.0.2
pytz==2022.1
pytz-deprecation-shim==0.1.0.post0
requests==2.28.1
six==1.16.0
soupsieve==2.3.2.post1
sqlparse==0.4.2
tzdata==2022.1
tzlocal==4.2
urllib3==1.26.11
unicorn==19.6.0
```

gitignore

```
### Django ###
*.log
*.pot
*.pyc
__pycache__/_
local_settings.py
db.sqlite3
db.sqlite3-journal
media
```

```

### Django.Python Stack ###
# Byte-compiled / optimized / DLL files
*.py[cod]
*$py.class

# C extensions
*.so

# Distribution / packaging
.Python
build/
develop-eggs/
dist/
downloads/
eggs/
.eggs/
lib/
lib64/
parts/
sdist/
var/
wheels/
share/python-wheels/
*.egg-info/
.installed.cfg
*.egg
MANIFEST

# PyInstaller
# Usually these files are written by a python script from a template
# before PyInstaller builds the exe, so as to inject date/other infos into it.
*.manifest
*.spec

# Installer logs
pip-log.txt
pip-delete-this-directory.txt

# Unit test / coverage reports
htmlcov/
.tox/
.nox/
.coverage
.coverage.*
.cache
nosetests.xml
coverage.xml
*.cover
*.py,cover
.hypothesis/
.pytest_cache/
cover/

# Translations
*.mo

# Django stuff:

# Flask stuff:
instance/
.webassets-cache

# Scrapy stuff:
.scrapy

# Sphinx documentation
docs/_build/

# PyBuilder
.pybuilder/
target/

# Jupyter Notebook
.ipynb_checkpoints

```

```

# IPython
profile_default/
ipython_config.py

# pyenv
# For a library or package, you might want to ignore these files since the code is
# intended to run in multiple environments; otherwise, check them in:
# .python-version

# pipenv
# According to pypa/pipenv#598, it is recommended to include Pipfile.lock in version control.
# However, in case of collaboration, if having platform-specific dependencies or dependencies
# having no cross-platform support, pipenv may install dependencies that don't work, or not
# install all needed dependencies.
#Pipfile.lock

# poetry
# Similar to Pipfile.lock, it is generally recommended to include poetry.lock in version control.
# This is especially recommended for binary packages to ensure reproducibility, and is more
# commonly ignored for libraries.
# https://python-poetry.org/docs/basic-usage/#commit-your-poetrylock-file-to-version-control
#poetry.lock

# pdm
# Similar to Pipfile.lock, it is generally recommended to include pdm.lock in version control.
#pdm.lock
# pdm stores project-wide configurations in .pdm.toml, but it is recommended to not include it
# in version control.
# https://pdm.fming.dev/#use-with-ide
.pdm.toml

# PEP 582; used by e.g. github.com/David-OConnor/pyflow and github.com/pdm-project/pdm
__pypackages__/

# Celery stuff
celerybeat-schedule
celerybeat.pid

# SageMath parsed files
*.sage.py

# Environments
.env
.venv
env/
venv/
ENV/
env.bak/
venv.bak/

# Spyder project settings
.spyderproject
.spyproject

# Rope project settings
.ropeproject

# mkdocs documentation
/site

# mypy
.mypy_cache/
.dmypy.json
dmypy.json

# Pyre type checker
.pyre/

# pytype static type analyzer
.pytype/

# Cython debug symbols
cython_debug/

# PyCharm
# JetBrains specific template is maintained in a separate JetBrains.gitignore that can

```

```
# be found at https://github.com/github/gitignore/blob/main/Global/JetBrains.gitignore
# and can be added to the global gitignore or merged into this file. For a more nuclear
# option (not recommended) you can uncomment the following to ignore the entire idea folder.
# .idea/

# End of https://www.toptal.com/developers/gitignore/api/django
```



BE(Django)

Python 가상 환경 생성

파일 생성 후 git bash 열기

- venv 생성 및 activate

```
#가상환경 생성
python -m venv [가상환경이름]
#source로 bin 디렉터리 안의 activate 파일을 적용하여 가상 환경을 활성화
source [가상환경이름]
```

- vscode로 인터프리터를 생성한 가상 환경으로 설정한다.

Build

- django 빌드를 위해 필요한 라이브러리 설치

```
pip install -r requirements.txt
```

- manage.py 파일을 이용해서 서버 가동

```
python manage.py runserver
```

Scheduling

- 코드

```
# /queant/queant_app/updater.py
#백그라운드 스케줄러를 사용, 맞추고 싶은 시간을 timezone으로 설정(Asia/Seoul)
scheduler = BackgroundScheduler(timezone='Asia/Seoul')
# trigger는 cron을 활용. 매일 낮 12시 55분에 업데이트
scheduler.add_job(save_db, trigger='cron', hour=12, minute=55)
#스케줄링 시작
scheduler.start()
```


Apshceduler

scheduler방식

1. Blocking scheduler

- Blocking scheduler는 이후의 동작을 못하도록 막는다.

2. Background scheduler

- blocking과는 다르게 background는 이후의 동작이 수행된다.

Option 종류

1. Cron 속성

- 매 시간, 매 분, 매 초의 실행을 제어할 수 있다.
ex) scheduler.add_job(schedule_api, 'cron', second=10)

Cron 표현 식 정리

- 분, 시, 일, 월, 요일 총 5개의 필드를 사용한다.
- /10 * * * * : 10분마다 실행
- 10 * * * * : 매일 매 시간 10분에 실행
- 30 * 5,15 * * : 매월 5, 15 일에 매 시 30분 때마다 실행
- /10 * * * 5 : 금요일마다 10분 간격으로 실행

2. Interiver 속성

- 실행 주기 옵션으로 특정 시간마다 실행되도록 한다.
ex) scheduler.add_job(schedule_api, 'interver', second=10)



AWS EC2 & Docker

프로젝트 개요

Queant 프로젝트는 Vue, SpringBoot, Django 총 세 개의 서비스 서버와 MariaDB, CI/CD 에 필요한 젠킨스 컨테이너, 그리고 최종적으로 서버를 관리하게 될 Nginx 컨테이너 총 6개로 이루어진다.

1. Vue Project의 경우 해당 컨테이너에 Nginx를 활용해 배포하고 있다.
2. SpringBoot 프로젝트의 경우 jar 파일을 만들어 깃랩에 올리면 이를 복사하여 내장 톰캣 서버를 활용해 배포하였다.
3. Django 프로젝트의 경우 Gunicorn 서버를 활용해 배포하였다.

간단한 Docker 설명

컨테이너 빌드 방식에는 두 가지 방법이 있다. 첫 번째는 Dockerfile 을 Shell 명령어로 실행시키는 것이고, 두 번째는 docker-compose.yml 파일을 작성해 그 파일을 실행시키는 것이다(이 경우에도 Dockerfile은 필요). 이번 프로젝트에서는 두 가지 방법을 모두 활용하고 있으며 기본적으로 Execute Shell 방식을 활용하지만 Django 프로젝트만 docker-compose.yml 방식을 채택하고 있다.

유용한 Docker 명령어

```
# 컨테이너 목록 조회
docker ps
# 컨테이너 전체 목록 조회(exited포함)
docker ps -a
# 이미지 목록 조회
docker images
# 이미지 삭제
docker image rm [image name and version]
# 컨테이너 삭제
docker rm [container name]
# 컨테이너 로그 확인
docker logs [container name]
# 컨테이너 내부 bash 접속
docker exec -it [container name] bash
# 컨테이너 실행 -p 는 포트번호, -v는 볼륨 정보, -d 는 백그라운드 실행, --name 은 컨테이너 이름 지정
docker run -p [port:port] -v [volume path:container path] -d --name [container name] [image name]
```

Docker 설치

1. curl을 통해 **docker** 설치 & **apt** 기능 추가

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

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

sudo apt update
```

2. Docker 설치

```
apt-cache policy docker-ce  
  
sudo apt install docker-ce
```

Jenkins

젠킨스는 CI/CD를 위한 툴이다. 깃랩에서 WebHook을 설정하고 젠킨스에 연결하면 연결된 브랜치에 푸시 혹은 머지가 일어날 때마다 일정한 절차를 거쳐서 빌드가 이루어지게 된다.

젠킨스 설치 및 패스워드 확인, 권한 조정(Docker Out Of Docker)

```
# 아래 세 줄은 권한 조정 커맨드인데, 조심해서 사용해야한다. 아래 docker run 명령어는 root 권한으로 실행하므로 필요없었던 듯  
sudo chown 1000 /Users/shared/data/jenkins  
sudo chmod 666 /var/run/docker.sock  
sudo chown root:docker /var/run/docker.sock  
  
# lts 버전의 jenkins 컨테이너 생성(이미지 pull 이후 해도 되지만 이미지가 없는 경우 자동으로 pull 실행)  
sudo docker run -d -u root -p 8500:8080 --name=jenkins -v /Users/shared/data/jenkins:/var/jenkins_home -v /var/run/docker.sock:/var/run/docker.sock  
# 비밀번호 찾기  
docker exec jenkins cat /var/jenkins_home/secrets/initialAdminPassword
```

젠킨스의 경우 외부에서 접속할 수 있어야 하므로 8500번 포트를 연결해주고 있다. 컨테이너 이름은 jenkins. -v는 볼륨을 설정해 주는 옵션인데, -v AWS서버경로:컨테이너 경로를 적어주면 컨테이너 바깥의 디렉토리와 컨테이너 내부의 디렉토리가 연동되게 된다. 연동되기 때문에 컨테이너 내부 파일이 바깥으로 복사가 되기도 하고, 컨테이너 바깥에서 파일을 변동하면 내부에도 동일하게 적용된다.

-v /var/run/docker.sock:/var/run/docker.sock 옵션이 중요한데, 현재 컨테이너에 젠킨스를 배포하고 있기 때문에 젠킨스 컨테이너에서 도커를 사용해 컨테이너를 빌드하려면 컨테이너 내부에 도커를 또 설치하거나 컨테이너 외부의 도커에 접속할 수 있어야 한다. 두 가지 방식을 Docker in Docker 와 Docker out of Docker 이라고 부른다. 두 가지 방법 모두 완벽하지는 않지만, 검색해본 결과 DOOD 방식이 DID 방식보다는 선호된다고 하여 DOOD 방식을 활용하였다. 해당 볼륨 옵션은 외부의 도커 소켓을 컨테이너 내부의 도커 소켓과 연결시켜서 젠킨스 컨테이너 내부에서 바깥의 도커 프로그램에 접속할 수 있게 허용하고 있으며 따라서 -u root 옵션을 통해 관리자 권한으로 명령어를 실행해야만 Permission Denied 오류가 뜨지 않는다.

docker exec jenkins cat /var/jenkins_home/secrets/initialAdminPassword 명령어는 젠킨스 설치 후 컨테이너 내부로 들어가서 임시로 부여된 비밀번호를 확인하는 명령어이다. 해당 패스워드를 이용해 젠킨스에 로그인 가능하며 이후에 관리자 계정을 등록할 수 있다.

젠킨스 설치를 완료했다면 플러그인을 설치해야 한다. 해당 프로젝트는 추천 플러그인을 우선 모두 설치한 다음 필요한 플러그인을 추가로 설치하였다. 정확히 어떤 플러그인을 추가로 설치했는지 기억이 나지 않는데, 대략 GitLab, WebHook 관련 플러그인을 설치한 것 같다. 혹시 문제가 생기면 JAVA, NodeJS를 Jenkins 관리 → Global Tool Configuration 에 설정해 보는 것도 추천한다.

MariaDB 설치 및 Volume 설정

1. MariaDB 이미지 찾기 및 Pull

```
sudo docker search mariadb;  
sudo docker pull mariadb:latest  
sudo docker images  
sudo docker ps  
sudo docker container run -d -p 3306:3306 -e MYSQL_ROOT_PASSWORD=1234 -v /Users/shared/data/mariadb:/var/lib/mysql --name mariadb_local
```

MariaDB의 경우 이미지를 다운받아서 명령어로 컨테이너 빌드가 간단하게 가능하다. 볼륨 옵션을 통해 데이터를 백업하여 컨테이너를 지웠다가 다시 생성하더라도 데이터가 유지되도록 하였다. 프로젝트 초기에는 DB도 포트번호를 지정해주었으나, 후반부에 Nginx 설정을 하면서 포트를 외부에 노출시키지 않고 내부 네트워크를 통해서만 접근 가능하도록 —expose 3306 옵션을 주어 빌드하도록 변경하였다.

2. 다음 명령어로 MariaDB 접속 후 계정을 생성하거나 DB에 접속하여 쿼리를 날릴 수 있다.

```
sudo docker exec -it mariadb_local bash
```

SpringBoot App 배포 및 CI/CD 구축

Java 8에 기반하여 포트 8000번 open 후 JAR 파일을 기반으로 이미지 Build

Dockerfile

```
FROM java:8
EXPOSE 8000
ARG JAR_FILE=./back-end/Queant/build/libs/*.jar
COPY ${JAR_FILE} app.jar
ENTRYPOINT ["java", "-jar", "/app.jar"]
```



CI/CD 구축 시 기존의 workspace를 clean 하지 않으면 동일한 jar파일이 계속해서 빌드가 되는 문제가 있었음. workspace clean은 젠킨스에서 설정 가능하다. 하지만 이 옵션으로 문제가 해결되긴 하였으나, 도커파일로 이미지를 빌드할 때 캐시를 사용하지 않고 빌드하라는 옵션을 주면 또 문제를 해결할 수 있을지도 모른다. 시도는 해보지 않았음.

Jenkins설정

빌드 유발

- ☐ 빌드를 원격으로 유발 (예: 스크립트 사용) ?
- ☐ Build after other projects are built ?
- ☐ Build periodically ?
- ☒ Build when a change is pushed to GitLab. GitLab webhook URL: <http://f7a201p.ssafy.io:8500/project/Queant-Back> ?
 - Enabled GitLab triggers
 - ☒ Push Events
 - ☐ Push Events in case of branch delete
 - ☒ Opened Merge Request Events
 - ☐ Build only if new commits were pushed to Merge Request ?
 - ☐ Accepted Merge Request Events
 - ☐ Closed Merge Request Events
 - Rebuild open Merge Requests
 - Never
 - ☒ Approved Merge Requests (EE-only)
 - ☐ Comments
 - Comment (regex) for triggering a build ?
 - Jenkins please retry a build
- 고급...

- ☐ Generic Webhook Trigger ?
- ☐ GitHub hook trigger for GITScm polling ?
- ☐ GitLab Merge Requests Builder
- ☐ Poll SCM ?

고급 설정

Comment (regex) for triggering a build ?

Jenkins please retry a build

☒ Enable [ci-skip]

☒ Ignore WIP Merge Requests

Labels that forces builds if they are added (comma-separated)

☒ Set build description to build cause (eg. Merge request or Git Push)

☐ Build on successful pipeline events

Pending build name for pipeline ?

☒ Cancel pending merge request builds on update

Allowed branches

☐ Allow all branches to trigger this job ?

☒ Filter branches by name ?

Include

Back-end

Matching 1 branch.

Exclude

main,

⚠ Following patterns don't match any branch in source repository:

☐ Filter branches by regex ?

☐ Filter merge request by label

Secret token ?

843816a5fa3abc1a6d513df2debb7705

Generate



이 때, Jenkins에만 빌드유발 설정을 하는 것이 아니라 Jenkins에서 가르쳐주는 build url 정보와 secret token을 gitlab의 webhook에 입력하고 Push 이벤트 설정도 해주어야 한다.

Execute Shell

```
#JAVA build(실제로 활용하지 않았으나 이번 프로젝트 처럼 깃랩에 jar 파일을 업로드 하지 않는 경우, 컨테이너에서 빌드하여 배포할 때 사용)
./gradlew clean build
#Dockerfile에 기반하여 이미지 빌드
docker build -t queant/backend .
#실행되고 있는 컨테이너의 이름이 queant-backend 인 것을 필터링 하고, 동일 이름의 실행되고 있는 컨테이너를 stop, 삭제
docker ps -q --filter "name=queant-backend" | grep -q . && docker stop queant-backend && docker rm queant-backend | true
#위에서 빌드한 이미지를 컨테이너로 실행
docker run -p 8000:8000 -d --name=queant-backend queant/backend
#도커 이미지 중 dangling=true 옵션을 이용해 사용되지 않는 불필요한 이미지 삭제
docker rmi -f $(docker images -f "dangling=true" -q) || true
```

Execute Shell 명령어를 젠킨스에서 설정하면 깃랩에 푸쉬가 될 때마다 지정된 쉘 명령어를 실행하여 컨테이너를 빌드하게 된다.

NGINX를 활용한 VUE App 배포

nginx.conf

```
worker_processes 4; //CPU 개수만큼

events { worker_connections 1024; }

http {
    server {
        listen 80;
        root /usr/share/nginx/html;
```

```

        include /etc/nginx/mime.types;

        location / {
            try_files $uri /index.html;
        }
    }
}

```

Dockerfile

```

FROM node:16.16.0 as builder

WORKDIR /queant

# Copy the package.json and install dependencies
COPY ./front-end/queant/package*.json ./
RUN npm install

# Copy rest of the files
COPY ./front-end/queant .

# Build the project
RUN npm run build

FROM nginx:alpine as production-build
COPY ./nginx.conf /etc/nginx/nginx.conf

## Remove default nginx index page
RUN rm -rf /usr/share/nginx/html/*

# Copy from the stahg 1
COPY --from=builder /queant/dist /usr/share/nginx/html

EXPOSE 80
ENTRYPOINT ["nginx", "-g", "daemon off;"]

```

Execute Shell

```

docker build -t queant/frontend .
docker ps -q --filter "name=queant-frontend" | grep -q . && docker stop queant-frontend && docker rm queant-frontend | true
docker run -d --name queant-frontend -p 80:80 queant/frontend
docker rmi -f $(docker images -f "dangling=true" -q) || true

```



node-alpine 이미지를 사용하면 python을 찾을 수 없다는 에러가 뜬다. python을 도커에 설치하지 않았으므로, alpine이 아닌 버전을 사용해 build

Django+Gunicorn

처음에는 Django+Gunicorn+Nginx로 배포하였으나, 첫 번째로 현재 프로젝트에서 사용하는 Django 서버는 단순히 주기적으로 데이터를 수집해서 DB를 업데이트 하는 역할만을 하고 프론트와 소통하는 일이 없고 두 번째로 Nginx 컨테이너를 별도로 추가하게 되면서 Nginx 부분은 제외하고 Django+Gunicorn으로 배포하게 되었다.



젠킨스 내부에 docker-compose설치

```

docker exec -it jenkins bash

curl -L "https://github.com/docker/compose/releases/download/$(curl https://github.com/docker/compose/releases | grep -m1 '<a href="/d

```

docker-compose.yml

```
version: '2' #docker-compose 버전을 나타냄
services:
  web: #web이라는 이름으로 container 실행
    build:
      dockerfile: Dockerfile #아까 작성한 DockerFile를 바탕으로 이미지 빌드
    command: bash -c "ls -la && cd ./queant && gunicorn queant.wsgi:application --bind 0.0.0.0:8600"
    expose:
      - "8600"
    network_mode: bridge
```



docker-compose 의 command 는 컨테이너 밖에서 실행되는 것을 확인. queant 폴더를 찾을 수 없다는 에러가 계속 났는데, 알고보니 처음 설정할 때 비어있는 폴더에 볼륨을 적용했더니 컨테이너로 복사되었던 git 리포지토리가 덮어쓰워지는 문제가 있었다.

Dockerfile

```
#아래 적힌 이미지를 베이스 이미지로 한다.
#이 명령어가 실행되면 Dockerhub에서 해당 이미지를 pull한다.
FROM python:3.10.5
# 환경변수 설정
ENV PYTHONUNBUFFERED 1
# docker 내에서 /data 라는 이름의 폴더 생성
RUN mkdir /data
# docker 내에서 코드를 실행할 폴더 위치를 /data 로 지정
WORKDIR /data
# 로컬의 requirements.txt 파일을 docker /code/ 폴더로 마운트
ADD ./data/queant/requirements.txt /data/
# docker 내 requirements.txt 파일을 이용하여 패키지 설치
RUN pip install -r requirements.txt
# 로컬 내 현재 위치에 있는 모든 파일 및 폴더를 docker 의 /data/ 폴더로 마운트
ADD ./data /data/
EXPOSE 8600
CMD ["python", "./queant/manage.py", "runserver", "0.0.0.0:8600"]
```

Execute shell

```
docker-compose up -d --build
docker rmi -f $(docker images -f "dangling=true" -q) || true
```

docker-compose vs docker run

- docker-compose.yml 로 컨테이너를 빌드할 경우, 각 컨테이너는 서비스에 할당된다. 서비스에 등록되는 경우 서비스의 이름과 컨테이너의 이름은 다를 수도 있으며 서비스 이름을 통해 컨테이너 간 소통이 가능하다.
- docker run의 경우 컨테이너는 생성 및 삭제 될때마다 다른 IP Address를 받게 되고, docker-compose에 비해 안정성이 떨어진다. 컨테이너 이름으로 컨테이너 간에 연결을 하기 위해서는 `--link` 라는 옵션을 사용할 수 있지만, deprecated 되어가는 설정이다. 네트워크를 직접 만들어서 연결할 수도 있다.
- docker-compose로 만들어지는 컨테이너는 자동으로 개별 네트워크가 생성되고 이에 할당된다.
- docker run으로 만들어지는 컨테이너는 기본 bridge 네트워크에 연결된다.
- docker-compose로 만들어지는 컨테이너를 기존의 bridge 네트워크에 연결하려면 해당 서비스에 `network_mode: bridge` 옵션을 주면 된다.

Nginx 및 SSL 설정

SSL 발급

주로 사용되는 SSL 발급 사이트는 LetsEncrypt로 Certbot 과 많이 사용되며 레퍼런스가 많은 편이다. 그러나 SSAFY 도메인에서 너무 많이 SSL을 발급받은 나머지 내가 발급받으려 할 때는 도메인이 막혀있었다. 따라서 아쉬운대로 ZeroSSL이라는 Open API를 활용하게 되었다.

SSL 발급을 위해서는 도메인 인증을 받아야 한다. 여러가지 방법이 있는 듯 했으나, 도메인에 다운로드 받은 AUTH 텍스트 파일을 올려서 인증을 받는 형식이 제일 제한이 없는 방식인 듯 하다. 아래 Dockerfile 과 nginx.conf 파일은 인증을 받기위해 임시로 컨테이너를 띄우기 위해 사용하였다. nginx 이미지에서 컨테이너를 생성하고, AUTH 파일과 nginx.conf 를 컨테이너 안으로 복사하고 있다. nginx파일에는 uri 경로와 해당 파일의 위치를 맵핑하고 있다.

Dockerfile

```
FROM nginx:stable-alpine

COPY ./nginx.conf /etc/nginx/conf.d/default.conf
COPY ./F697343443DACE02267D6A80F5FC5C36.txt /front/F697343443DACE02267D6A80F5FC5C36.txt

EXPOSE 80

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

nginx.conf

```
server{
    listen 80;
    client_max_body_size 2G;
    server_name localhost;

    location / {
        index index.html index.htm;
        try_files $uri $uri/ =404;
    }

    location = /.well-known/pki-validation/F697343443DACE02267D6A80F5FC5C36.txt {
        allow all;
        alias /front/F697343443DACE02267D6A80F5FC5C36.txt;
    }
}
```

NGINX 설정

기존에 CI/CD 파이프라인을 구축할 때는 호스트의 포트를 각 컨테이너에 모두 연결시켜서 외부로 모든 포트를 노출시켰다. 하지만 이는 보안상 좋지 않기 때문에, 내부에 NGINX 컨테이너를 두고 이 컨테이너 만을 80번 포트를 통해 노출시키고 나머지 서버는 내부에서 연결을 하도록 구조를 바꾸었다. 그리고 포트가 하나만 열리게 되므로, 발급받은 SSL키로 HTTPS 설정을 통해 보안을 강화했다.

docker-compose.yml

```
services:
  nginx:
    image: nginx:latest
    volumes:
      - ./nginx/conf/nginx.conf:/etc/nginx/nginx.conf
      - ./nginx/zeross:/etc/zeross
    ports:
      - 80:80
      - 443:443
    links:
      - queant-frontend
      - queant-backend
```


nginx 컨테이너를 띄우기 위해 docker-compose.yml 을 작성하였으나, 기존의 컨테이너를 docker run 명령어로 띄웠기 때문에 links 에서 queant-frontend 와 queant-backend를 인식하지 못하는 문제가 있었다. links 에는 서비스의 이름을 적어야 인식 가능하다. 따라서 아 쉬운대로 nginx 컨테이너를 띄우는데도 command line을 사용하게 되었다.

Docker 명령어

```
docker run -d --name nginx -v /home/ubuntu/nginx/conf/nginx.conf:/etc/nginx/nginx.conf -v /home/ubuntu/nginx/zeross:/etc/zeross -p 80:80
docker exec -it nginx bash
rm etc/nginx/conf.d/default.conf
```

AWS EC2 서버의 nginx.conf 파일과 nginx.conf 파일을 볼륨을 통해 컨테이너 내부로 복사하고 있다. 설정에 수정이 필요한 경우 외부 파일을 수정하고 저장한 후 컨테이너 내부에서 nginx -s reload 를 통해 변경된 설정을 적용하던가 컨테이너를 멈췄다가 다시 켜주면 적용된다. 그런데 —link 옵션을 사용했더니 nginx 컨테이너 내부로 들어갈 수가 없어 번거롭지만 매번 컨테이너를 꺾다가 켜야 했다. 다음에는 네트워크 설정을 활용하도록 하자.

nginx.conf

```
user nginx; # 프로세스의 실행되는 권한. 보안상 root를 사용하지 않습니다.
worker_processes auto;
error_log /var/log/nginx/error.log warn;
pid /var/run/nginx.pid;
events {
    worker_connections 1024;
}

http {
    include /etc/nginx/mime.types;
    default_type application/octet-stream;

    server {
        listen 80;
        listen [::]:80;
        server_name i7a201.p.ssafy.io;
        location / {
            return 301 https://$host$request_uri;
        }

        location /robots.txt {
            return 200 "User-agent: *\nDisallow: /";
        }
    }

    #서비스 서버
    server {
        listen 443 ssl;
        server_name i7a201.p.ssafy.io;

        ssl_certificate /etc/zeross/certificate.crt;
        ssl_certificate_key /etc/zeross/private.key;

        location / {
            proxy_pass http://queant-frontend:8080;
            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;
        }

        location /api/ {
            proxy_pass http://queant-backend:8000;
            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;
        }

        location /robots.txt {
            return 200 "User-agent: *\nDisallow: /";
        }
    }

    log_format main '$remote_addr - $remote_user [$time_local] "$request" '
        '$status $body_bytes_sent "$http_referer" '
        '"$http_user_agent" "$http_x_forwarded_for"';
```

```
access_log /var/log/nginx/access.log main;

sendfile on;
server_tokens off;
keepalive_timeout 65;
include /etc/nginx/conf.d/*.conf;
}
```

http 블록 내부의 첫 서버 블록은 http의 80번 포트로 들어오는 모든 요청을 https로 리다이렉트 해주고 있다. 두 번째 서버 블록은 443 포트로 들어오는 요청을 받고 있는데, 443번 포트는 기본적으로 https 요청을 말한다고 한다. 자세한 내용은 링크를 참고.

http의 기본 포트가 80, https의 기본 포트가 443인 이유는 무엇일까?

모르고 있었다면 간단한 실험으로도 알 수 있다. 이는 80과 443 이 기본 포트 번호이기 때문이다. 포트 번호를 생략하면 기본 포트를 사용하게 된다. 자신이 자주 방문하는 웹 사이트 URL을 확인해보자. 포트 번호가 명시되어 있지 않다면, 프로토콜을 보고 80이나 443 포트를 붙여 실험해보자. 포트를 명시하지 않았을 때와 똑같이 잘 접속될 것이다.

링크 <https://johngrub.github.io/wiki/why-http-80-https-443/>



두 번째 서버 블록의 내용을 자세히 살펴보면 우선 443포트를 listen 하면서 ssl 설정을 해주고 있는 것을 확인할 수 있고, 두 줄 밑 ssl_certificate 와 ssl_certificate_key 에서 컨테이너 생성시 볼륨으로 복사해주었던, ZeroSSL에서 발급받은 인증서와 키의 로컬 주소를 설정해주었음을 알 수 있다.

다음으로는 location / 와 location /api 를 통해 reverse proxy를 설정하고 있다. NGINX는 기본적으로 forward proxy이지만, Reverse Proxy 기능도 제공한다. location 설정으로 uri에 따라 연결하는 내부 서버가 달라지게 된다. 위 예제에서는 특이 사항이 없는 경우 뷰 서버가 배포되어있는 front 컨테이너로, /api uri로 요청할 경우 Tomcat WAS가 실행되고 있는 back 컨테이너로 연결을 해주게 된다. 앞에서 nginx 컨테이너를 실행할 때 —link 옵션을 통해 queant-frontend 와 queant-backend 컨테이너를 컨테이너 이름으로 연결시켜 줬기 때문에 nginx.conf 파일에서도 컨테이너 이름을 통해 서로 소통이 가능하다.



Forward Proxy : 사용자가 google.com에 연결하려고 하면 사용자 PC가 직접 연결하는게 아니라 포워드 프록시 서버가 요청을 받아서 google.com에 연결하여 결과를 클라이언트에 전달해줍니다.



Reverse Proxy : 사용자가 example.com 웹 서비스에 데이터를 요청하면 서버는 이 요청을 받아서 내부 서버에서 데이터를 받은 후 데이터를 사용자에게 다시 전달하게 됩니다.

이 외에 어려움으로, Tomcat 서버의 uri 변경에 따른 Swagger 설정 변경에 어려움을 겪었다. 처음에는 nginx.conf 파일에서 rewrite 옵션을 줌으로써 /api 로 들어온 요청에서 /api 부분을 제거하고 넘겨주었던, Swagger 에서 테스트를 할 때도 /api를 제외하고 요청이 전달되는 것이었다.

두 번째로 시도한 방법은 application.properties 에서 server.servlet.context-path=/api 설정을 해줌으로써 톰캣 서버에서도 /api를 포함한 요청을 받는 것이었다. 앞에서 언급한 문제를 해결할 수 있었으나, 이번에는 리버스 프록시를 통해 전달받은 요청은 http 였던 반면에 Swagger 에서는 https로 요청을 보내야해서(클라이언트 단에서 보내게 되기 때문) 인증 방식이 불일치해서 문제가 생겼다.

Swagger 설정에 아래 코드를 추가함으로써 요청을 보낼 서버를 커스텀할 수 있었다. Workaround를 추가해야하는 줄 모르고 Docket 설정만 해주거나, Workaround 에서 Server을 잘못 import 하는 실수들이 있었으나(import io.swagger.v3.oas.models.servers.Server; 에서 import 해야 한다.) 간신히 문제를 해결하고 /api 도메인에서 Swagger 접속도 가능하게 되었다.

SwaggerConfig

```

@Bean
public Docket api() {
    Server testServer = new Server("test", "https://i7a201.p.ssafy.io", "for testing", Collections.emptyList(), Collections.emptyList());
    return new Docket(DocumentationType.OAS_30)
        .servers(testServer)
        .groupName("Queant")
        .apiInfo(apiInfo())
        .useDefaultResponseMessages(false)
        .select()
        .apis(RequestHandlerSelectors.basePackage("com.ssafy.queant.controller"))
        .build();
}

```

Workaround

```

@Component
public class Workaround implements WebMvcOpenApiTransformationFilter {

    @Override
    public OpenAPI transform(OpenApiTransformationContext<HttpServletRequest> context) {
        OpenAPI openApi = context.getSpecification();

        Server testServer = new Server();
        testServer.setDescription("test");
        testServer.setUrl("https://i7a201.p.ssafy.io");
        openApi.setServers(Arrays.asList(testServer));
        return openApi;
    }

    @Override
    public boolean supports(DocumentationType documentationType) {
        return documentationType.equals(DocumentationType.OAS_30);
    }
}

```

AWS S3

프로젝트를 진행하며 게시글 작성 컴포넌트로 ckeditor5를 사용하였다.

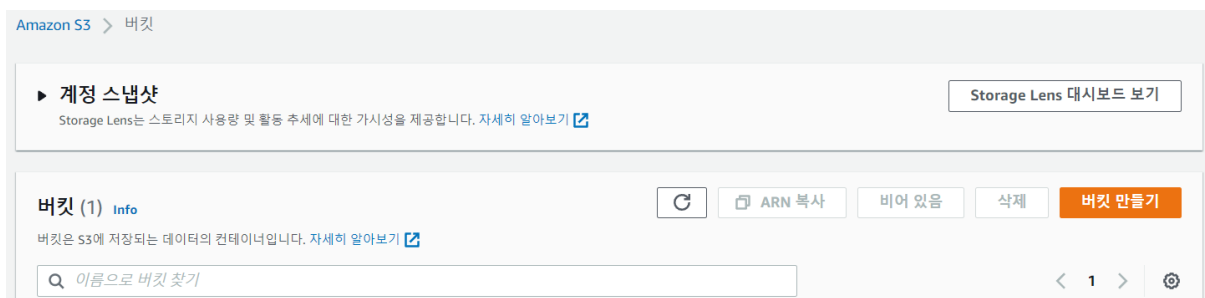
게시판에서 이미지를 업로드할 때 업로드한 이미지를 저장할 서비스로 AWS S3를 사용하였다.

AWS S3는 아마존에서 제공하는 클라우드 스토리지 서비스다.

AWS S3 Bucket 생성

S3를 이용하기 위해 회원가입을 진행한 뒤, 스토리지에서 최상위 디렉토리의 역할을 하는 버킷을 생성해야 한다.

1. S3 콘솔로 이동하여 버킷 만들기를 클릭한다.



2. 버킷 이름과 리전을 설정한다. 리전은 업로드한 객체를 저장할 지역에 해당한다.

일반 구성

버킷 이름

queant

버킷 이름은 전역에서 고유해야 하며 공백 또는 대문자를 포함할 수 없습니다. [버킷 이름 지정 규칙 참조](#)

AWS 리전

아시아 태평양(서울) ap-northeast-2

기존 버킷에서 설정 복사 - 선택 사항
다음 구성의 버킷 설정만 복사됩니다.

버킷 선택

3. 객체 소유권을 설정한다. 비활성화시 업로드중 권한이 막혀 문제가 발생한다.

객체 소유권 [Info](#)

다른 AWS 계정에서 이 버킷에 작성한 객체의 소유권 및 액세스 제어 목록(ACL)의 사용을 제어합니다. 객체 소유권은 객체에 대한 액세스를 지정할 수 있는 사용자를 결정합니다.

- ☐ ACL 비활성화됨(권장)
이 버킷의 모든 객체는 이 계정이 소유합니다. 이 버킷과 그 객체에 대한 액세스는 정책을 통해서만 지정됩니다.

- ☒ ACL 활성화됨
이 버킷의 객체는 다른 AWS 계정에서 소유할 수 있습니다. 이 버킷 및 객체에 대한 액세스는 ACL을 사용하여 지정할 수 있습니다.

객체 소유권

- ☒ 버킷 소유자 선택
이 버킷에 작성된 새 객체가 bucket-owner-full-control 삽입 ACL을 지정하는 경우 새 객체는 버킷 소유자가 소유합니다. 그렇지 않은 경우 객체 라이터가 소유합니다.

- ☐ 객체 라이터
객체 라이터는 객체 소유자로 유지됩니다.

새 객체에 대해서만 객체 소유권을 적용하려면 버킷 정책이 객체 업로드에 bucket-owner-full-control 삽입 ACL을 요구하도록 지정해야 합니다. [자세히 알아보기](#)

4. 퍼블릭 액세스 차단을 해제한다. 게시판에 업로드한 이미지를 보기 위해서 필요하다.

이 버킷의 퍼블릭 액세스 차단 설정

퍼블릭 액세스는 ACL(액세스 제어 목록), 버킷 정책, 액세스 지정 정책 또는 모두를 통해 버킷 및 객체에 부여됩니다. 이 버킷 및 해당 객체에 대한 퍼블릭 액세스가 차단되었는지 확인하려면 모든 퍼블릭 액세스 차단을 활성화합니다. 이 설정은 이 버킷 및 해당 액세스 지점에만 적용됩니다. AWS에서는 모든 퍼블릭 액세스 차단을 활성화하도록 권장하지만, 이 설정을 적용하기 전에 퍼블릭 액세스가 없어도 애플리케이션이 올바르게 작동하는지 확인합니다. 이 버킷 또는 내부 객체에 대한 어느 정도 수준의 퍼블릭 액세스가 필요한 경우 특정 스토리지 사용 사례에 맞게 아래 개별 설정을 사용자 지정할 수 있습니다. [자세히 알아보기](#)

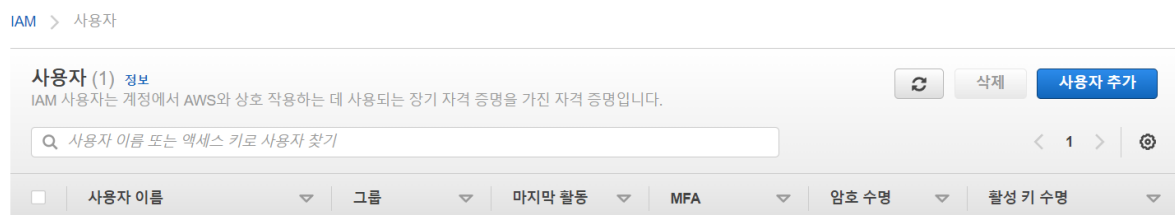
☐ 모든 퍼블릭 액세스 차단

이 설정을 활성화하면 아래 4개의 설정을 모두 활성화한 것과 같습니다. 다음 설정 각각은 서로 독립적입니다.

IAM 사용자 생성

spring과 연동하기 위해 IAM 사용자를 생성하여 접근키와 비밀번호를 발급받아야 한다.

1. IAM 콘솔로 이동하여 사용자 추가를 클릭한다.



2. 이름과 유형을 선택한다. spring에서 이용하려면 프로그래밍 방식 액세스를 체크해야 한다.

사용자 추가



사용자 세부 정보 설정

동일한 액세스 유형 및 권한을 사용하여 한 번에 여러 사용자를 추가할 수 있습니다. 자세히 알아보기

사용자 이름*

[다른 사용자 추가](#)

AWS 액세스 유형 선택

이러한 사용자가 주로 AWS에 액세스하는 방법을 선택합니다. 프로그래밍 방식의 액세스만 선택하면 사용자가 위임된 역할을 사용하여 콘솔에 액세스하는 것을 방지할 수 없습니다. 액세스 키와 자동 생성된 암호가 마지막 단계에서 제공됩니다. 자세히 알아보기

- AWS 자격 증명 유형 선택***
- ☒ **액세스 키 – 프로그래밍 방식 액세스**
AWS API, CLI, SDK 및 기타 개발 도구에 대해 **액세스 키 ID** 및 **비밀 액세스 키** 을(를) 할
성화합니다.
 - ☐ **암호 – AWS 관리 콘솔 액세스**
사용자가 AWS Management Console에 로그인할 수 있도록 허용하는 **비밀번호** 을(를)
활성화합니다.

* 필수

[취소](#)


[다음: 권한](#)


3. 정책을 연결한다. AmazonS3FullAccess 권한을 부여한다.


사용자 추가

1 2 3 4 5

▼ 권한 설정

 그룹에 사용자 추가

 기존 사용자에서 권한 복사

 기존 정책 직접 연결

정책 생성

↺

정책 필터

Q amazons3

5 결과 표시

4. 선택사항들을 모두 넘기면 액세스 키 ID와 비밀 액세스 키를 발급받는다.

주의할 점은, 이 페이지를 넘어가면 비밀 액세스 키를 다시 확인할 수 있는 방법이 없으므로

csv파일을 다운로드 받거나 텍스트 파일로 잘 저장해놓아야 한다.

사용자 추가

1 2 3 4 5

✓ 성공

아래에 표시된 사용자를 생성했습니다. 사용자 보안 자격 증명을 보고 다운로드할 수 있습니다. AWS Management Console 로그인을 위한 사용자 지침을 이메일로 보낼 수도 있습니다. 지금이 이 자격 증명을 다운로드할 수 있는 마지막 기회입니다. 하지만 언제든지 새 자격 증명을 생성할 수 있습니다.

AWS Management Console 액세스 권한이 있는 사용자가 <https://260119835991.signin.aws.amazon.com/console>에 로그인할 수 있습니다.

📄 .csv 다운로드

사용자	액세스 키 ID	비밀 액세스 키
▶  queant	AKIATZECZWWLYHOFYLPG	***** 표시

application-API-KEY.properties 설정

프로젝트에서 git에 push할 수 없는 민감한 정보들을 API-KEY properties 에서 담당하고 있다.

해당 파일에 S3와 관련된 정보를 추가한다.

```
#aws s3
cloud.aws.credentials.instance-profile=true
cloud.aws.stack.auto=false
cloud.aws.s3.bucket=버킷이름
cloud.aws.region.static=리전이름
cloud.aws.credentials.access-key=액세스 키 ID
cloud.aws.credentials.secret-key=비밀 액세스 키
```

application.properties 설정

MultipartFile 업로드 시 업로드 가능한 파일 최대 크기를 설정해준다.

```
# file upload max size (파일 업로드 크기 설정)
spring.servlet.multipart.max-file-size=200MB
spring.servlet.multipart.max-request-size=200MB
```

build.gradle 설정

build.gradle에서 S3 서비스와 연동하기 위한 의존성을 주입한다.

```
dependencies {
    //aws s3
    implementation 'org.springframework.cloud:spring-cloud-starter-aws:2.2.6.RELEASE'
}
```

AmazonS3Config 설정

API-KEY properties에 저장한 s3 관련 정보들을 다른 클래스에서 불러오기 위한 config를 설정한다.

```
@Value("${cloud.aws.credentials.access-key}")
private String accessKey;

@Value("${cloud.aws.credentials.secret-key}")
private String secretKey;
```



```

@Value("${cloud.aws.region.static}")
private String region;

@Bean
public AmazonS3Client amazonS3Client() {
    BasicAWSCredentials awsCreds = new BasicAWSCredentials(accessKey, secretKey);
    return (AmazonS3Client) AmazonS3ClientBuilder.standard()
        .withRegion(region)
        .withCredentials(new AWSStaticCredentialsProvider(awsCreds))
        .build();
}

```

업로드 설정 (DTO 파일)

vue와 연동하기 위해 vue로 데이터를 전송하기 위한 DTO 파일을 생성한다.

```

@Schema(description = "업로드 성공 여부")
private Boolean uploaded;

@Schema(description = "s3로 업로드된 이미지의 경로")
private String url;

```

업로드 설정 (Service 파일)

vue에서 받은 파일을 s3로 업로드 하기 위한 service 파일을 생성한다.

```

//메인클래스 이전에 선언한다.
@Value("${cloud.aws.s3.bucket}")
public String bucket;

```

```

File convertFile = new File(System.getProperty("user.dir") + "/"
    + multipartFile.getOriginalFilename());
convertFile.createNewFile();
FileOutputStream fos = new FileOutputStream(convertFile);
fos.write(multipartFile.getBytes());

File uploadFile = convertFile;

// S3에 저장된 파일 이름
String fileName = "static" + "/" + UUID.randomUUID() + uploadFile.getName();

amazonS3Client.putObject(new PutObjectRequest(bucket, fileName, uploadFile)

```

```

        .withCannedAcl(CannedAccessControlList.PublicRead));

// s3로 업로드
String uploadImageUrl = amazonS3Client.getUrl(bucket, fileName).toString();

```

업로드 설정 (Controller 파일)

ckeditor에서 업로드를 하기 위해 uploaded의 boolean값과 url의 String값을 전송해야 한다.

```

@ApiOperation(value="이미지 업로드", notes="이미지 업로드 버튼을 누르면 작동")
@ResponseBody
@PostMapping("/contents/upload")
public ResponseEntity<UploadDto> UploadImage(@RequestParam("upload")
MultipartFile multipartFile) throws IOException {
    //DTO
    UploadDto ud = new UploadDto();

    //
    //service 부분
    //

    //uploaded와 url을 반환해야 ckeditor5에서 업로드가 정상 작동
    ud.setUploaded(true);
    ud.setUrl(uploadImageUrl);

    if(uploadImageUrl == null) {
        log.info("[UploadImage] run failed");
        return new ResponseEntity<>(HttpStatus.NOT_FOUND);
    }
    else {

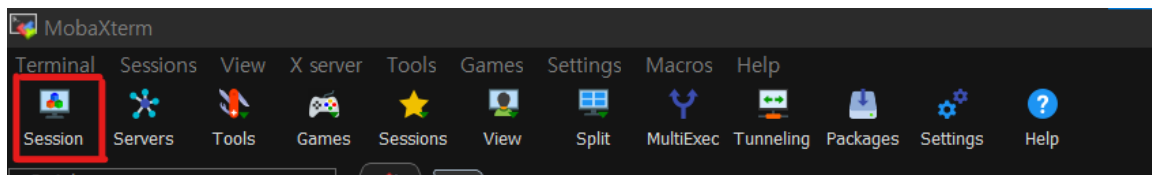
        log.info("[UploadImage] run finished");
        return new ResponseEntity<UploadDto>(ud, HttpStatus.OK);
    }
}

```

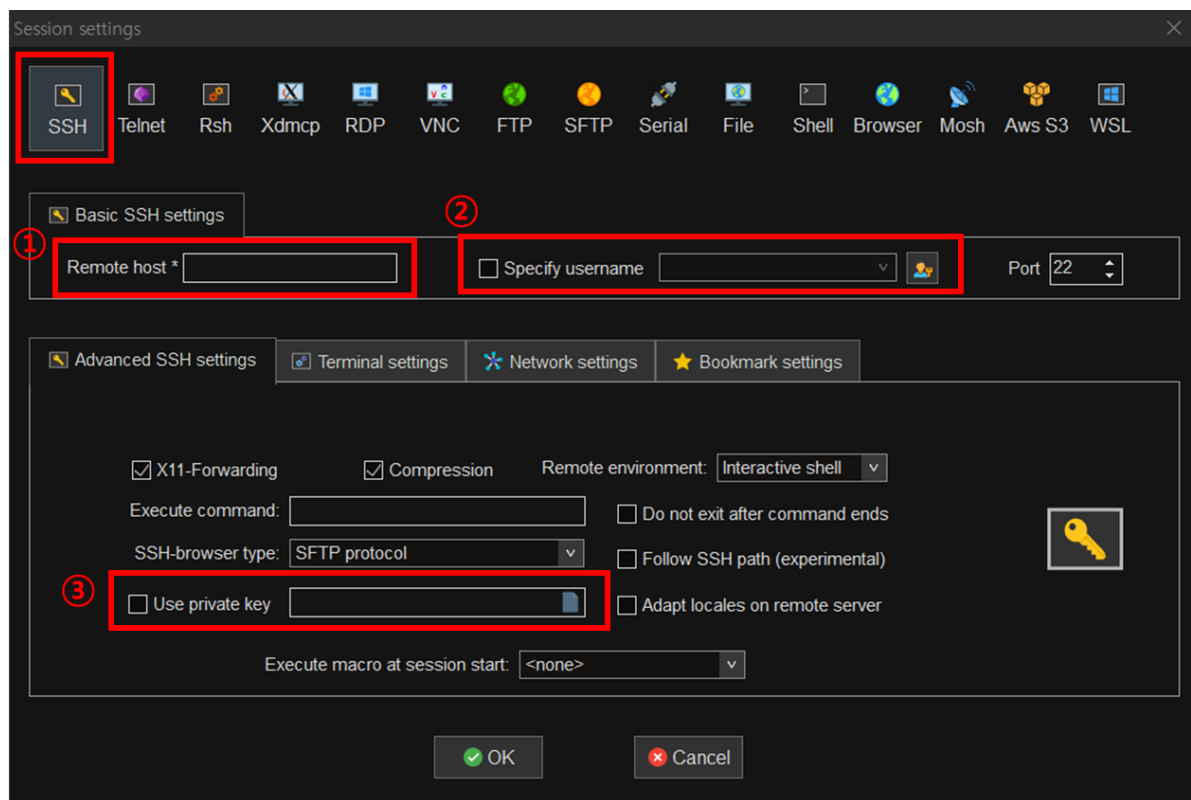
MariaDB/MobaXterm

MobaXterm - AWS연결

- 세션 클릭



- 필요한 내용 작성
 1. AWS 주소 입력
 2. username 입력. 보통 default값 ubuntu
 3. 발급 받은 pem키 연결



Docker의 MariaDB 데이터 확인

```
docker exec -it [컨테이너 명] bash

mysql -u [사용자이름] -p

[password입력]

use [테이블명];
```

위와 같이 mariaDB 및 사용자, table 선택하고 각종 query를 통해 데이터 CRUD 가능.

Docker의 log확인

```
docker ps #도커 내 컨테이너 목록 확인
docker logs [컨테이너 명] #확인하고자 하는 컨테이너 log확인
```

MariaDB Backup

```
# 특정 DB Backup
docker exec [컨테이너명] /usr/bin/mysqldump -u [UID]
--password=[Password]
[DataBase] > [파일저장경로]
```

docker 컨테이너 내의 db에 접근하여 docker directory에 백업 파일을 저장한다.



소셜 로그인 API

소셜 로그인 flow

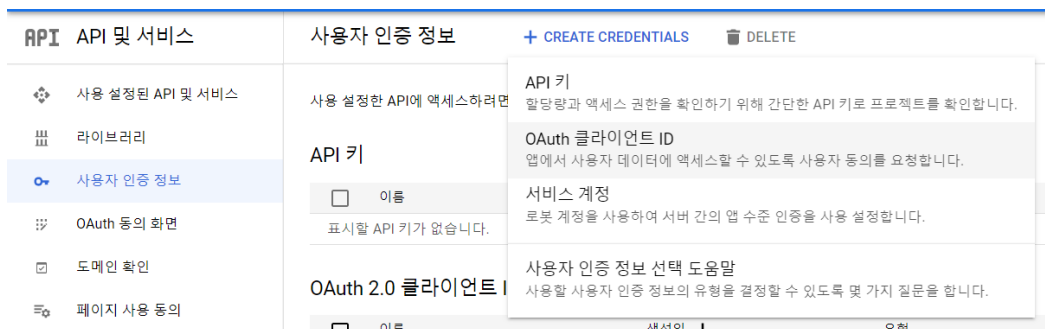
1. 소셜 로그인 창을 띄움
 - a. api 키 값을 갖고 로그인 URL 생성함
키 값을 backend에서 갖고 있고 프론트에서 backend로 로그인 링크 요청시 링크 생성해서 프론트에서 반환해줌 → 해당 링크로 프론트가 접속
2. 소셜 로그인 성공 → 각 소셜 API에서 설정한 Redirect URI로 Authorization code가 반환됨
 - a. 프론트에서 받아서 받은 Authorization Code를 들고 backend에게 요청을 보냄 → backend에서 3,4번의 로직을 수행함
3. 반환된 Authorization Code로 소셜 API의 Access Token 요청 URI 로 Access Token 요청
4. 반환된 Access Token으로 소셜 API의 사용자 정보 요청 URI 로 사용자 정보 요청

▼ GOOGLE

▼ API 셋팅 방법

1. 구글 클라우드 접속 및 프로젝트 생성
2. 프로젝트에서

API 및 서비스 > 사용자 인증 정보 > CREATE CREDENTIALS > OAuth 클라이언트 ID 생성



3. 프로젝트 유형 선택

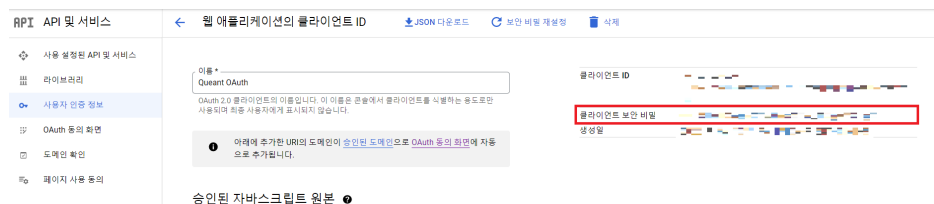
- 구글 API에서 설정한 Redirect URI

- 스코프 값 예시 : profile%20email%20openid
스코프 값은 URL에 들어가는 값이라, 를 %20으로 치환해준듯

▼ 소셜 API의 Access Token 요청 URI

<https://oauth2.googleapis.com/token>

- Body에 아래 값 담아서 **POST** 보내기
 - grant_type
“authorization_code” → 문자열 그 자체
 - client_id
구글 API 클라이언트 ID 값 (위에 사진 첨부함)
 - redirect_uri
구글 API에서 설정한 Redirect URI (위에 사진 첨부함)
 - code
프론트에서 넘겨받은 Authorization Code
 - client_secret



▼ 소셜 API의 사용자 정보 요청 URI

<https://www.googleapis.com/oauth2/v1/userinfo>

- Access Token 값을 헤더에 설정

```
HttpHeaders headers = new HttpHeaders();
headers.setContentType(MediaType.APPLICATION_FORM_URLENCODED);
headers.set("Authorization", "Bearer " + accessToken);
```

▼ KAKAO

▼ API 셋팅 방법

1. <https://developers.kakao.com/> 에서 내 애플리케이션 등록
2. 내 애플리케이션에서 플랫폼 등록

kakao developers

내 애플리케이션 > 앱 설정 > 플랫폼

앱 설정
요약 정보
일반
비즈니스
앱 키
플랫폼
팀 관리

제품 설정
카카오 로그인
동의항목
간편가입
카카오톡 채널
개인정보 국외이전

내 애플리케이션 제품 문서 도구

Web 플랫폼 등록

사이트 도메인

JavaScript SDK, 카카오톡 공유, 카카오톡, 메시지 API 사용시 등록이 필요합니다.
여러개의 도메인은 줄바꿈으로 추가해주세요. 최대 10까지 등록 가능합니다. 추가 등록은 포럼(데브톡)으로 문의주세요.
예시: (O) <https://example.com> (X) <https://www.example.com>

기본 도메인

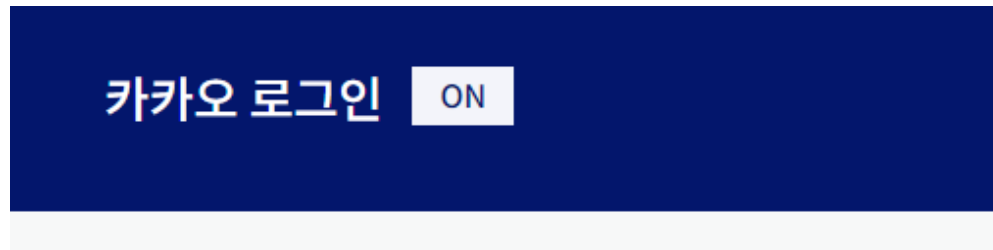
기본 도메인은 첫 번째 사이트 도메인으로, 카카오톡 공유와 카카오톡 메시지 API를 통해 발송되는 메시지의 Web 링크 기본값으로 사용됩니다.

3. 제품 설정에서 카카오 로그인 설정

제품 설정

카카오 로그인

- 카카오 로그인 켜기



활성화 설정

상태

ON

카카오 로그인 API를 활용하면 사용자들이 번거로운 회원 가입 절차가 OFF일 때도 카카오 로그인 설정 항목을 변경하고 서버에 저장된 상태가 ON일 때만 실제 서비스에서 카카오 로그인 화면이 연결됩니다.

- Redirect URL 설정

Redirect URI

삭제

수정

Redirect URI



- 카카오 로그인에서 사용할 OAuth Redirect URI를 설정합니다. (최대 10개)
- REST API로 개발하는 경우 필수로 설정해야 합니다.

▼ API URI

▼ 로그인 url

https://kauth.kakao.com/oauth/authorize?client_id= 카카오 API 클라이언트 ID값 &redirect_uri= 카카오 API에서 설정한 Redirect URI &response_type=code

- 카카오 API 클라이언트 ID 값

앱 키

네이티브 앱 키	
REST API 키	
JavaScript 키	
Admin 키	

- 카카오 API에서 설정한 Redirect URI

Redirect URI		삭제 수정
Redirect URI		

• 카카오 로그인에서 사용할 OAuth Redirect URI를 설정합니다. (최대 10개)
• REST API로 개발하는 경우 필수로 설정해야 합니다.

▼ 소셜 API의 Access Token 요청 URI

<https://kauth.kakao.com/oauth/token>

- Body에 아래 값 담아서 **POST** 보내기
 - grant_type
“authorization_code” → 문자열 그 자체
 - client_id
카카오 API 클라이언트 ID 값 (위에 사진 첨부함)
 - redirect_uri
카카오 API에서 설정한 Redirect URI (위에 사진 첨부함)
 - code
프론트에서 넘겨받은 Authorization Code

▼ 소셜 API의 사용자 정보 요청 URI

<https://kapi.kakao.com/v1/oidc/userinfo>

- Access Token 값을 헤더에 설정

```
HttpHeaders headers = new HttpHeaders();
headers.setContentType(MediaType.APPLICATION_FORM_URLENCODED);
headers.set("Authorization", "Bearer " + accessToken);
```

▼ NAVER

▼ API 셋팅 방법

1. <https://developers.naver.com/> 접속 및 애플리케이션 등록
2. 애플리케이션에서 API 이용 신청
 - 사용 API 설정

애플리케이션 등록 (API 이용신청)

애플리케이션의 기본 정보를 등록하면, 좌측 **내 애플리케이션** 메뉴의 서브 메뉴에 등록하신 애플리케이션 이름으로 서브 메뉴가 만들어집니다.

애플리케이션 이름	<input type="text" value="애플리케이션 이름"/> ✓ • 네이버 로그인할 때 사용자에게 표시되는 이름이므로 서비스 브랜드를 대표할 수 있는 이름으로 가급적 10자 이내로 간결하게 설정해주세요. • 40자 이내의 영문, 한글, 숫자, 공백문자, 쉼표(,), "/" , "-" , "_" 만 입력 가능합니다.
사용 API	<div> <div>선택하세요. ①</div> <div> <div>검색</div> <div>네이버 로그인</div> <div>네이버페이 배송지 정보</div> </div> </div>

사용 API	<div> <div>선택하세요. ✓</div> <div> <div>네이버 로그인</div> <div> <p>제공 정보 선택(이용자 식별자는 기본 정보로 제공) ②</p> <p>필수 항목은 개인정보보호법 제3조 제1항, 제16조 제1항 등에 따라 서비스 제공을 위해 필요한 최소한의 개인정보만을 선택해야 합니다.</p> <table border="1"> <thead> <tr> <th>권한</th> <th>필수</th> <th>추가</th> </tr> </thead> <tbody> <tr> <td>회원이름</td> <td><input checked="" type="checkbox"/></td> <td><input type="checkbox"/></td> </tr> <tr> <td>이메일 주소</td> <td><input type="checkbox"/></td> <td><input type="checkbox"/></td> </tr> <tr> <td>별명</td> <td><input type="checkbox"/></td> <td><input type="checkbox"/></td> </tr> <tr> <td>프로필 사진</td> <td><input type="checkbox"/></td> <td><input type="checkbox"/></td> </tr> <tr> <td>성별</td> <td><input type="checkbox"/></td> <td><input type="checkbox"/></td> </tr> <tr> <td>생일</td> <td><input type="checkbox"/></td> <td><input type="checkbox"/></td> </tr> <tr> <td>연령대</td> <td><input type="checkbox"/></td> <td><input type="checkbox"/></td> </tr> <tr> <td>출생연도</td> <td><input type="checkbox"/></td> <td><input type="checkbox"/></td> </tr> <tr> <td>휴대전화번호</td> <td><input type="checkbox"/></td> <td><input type="checkbox"/></td> </tr> </tbody> </table> <p>[알림] 추가권한에 대한 네이버 로그인 공지사항을 확인하세요.</p> </div> </div> </div>	권한	필수	추가	회원이름	<input checked="" type="checkbox"/>	<input type="checkbox"/>	이메일 주소	<input type="checkbox"/>	<input type="checkbox"/>	별명	<input type="checkbox"/>	<input type="checkbox"/>	프로필 사진	<input type="checkbox"/>	<input type="checkbox"/>	성별	<input type="checkbox"/>	<input type="checkbox"/>	생일	<input type="checkbox"/>	<input type="checkbox"/>	연령대	<input type="checkbox"/>	<input type="checkbox"/>	출생연도	<input type="checkbox"/>	<input type="checkbox"/>	휴대전화번호	<input type="checkbox"/>	<input type="checkbox"/>
	권한	필수	추가																												
	회원이름	<input checked="" type="checkbox"/>	<input type="checkbox"/>																												
	이메일 주소	<input type="checkbox"/>	<input type="checkbox"/>																												
	별명	<input type="checkbox"/>	<input type="checkbox"/>																												
	프로필 사진	<input type="checkbox"/>	<input type="checkbox"/>																												
	성별	<input type="checkbox"/>	<input type="checkbox"/>																												
	생일	<input type="checkbox"/>	<input type="checkbox"/>																												
	연령대	<input type="checkbox"/>	<input type="checkbox"/>																												
	출생연도	<input type="checkbox"/>	<input type="checkbox"/>																												
휴대전화번호	<input type="checkbox"/>	<input type="checkbox"/>																													

- 리다이렉트 URL과 서비스 URL 설정

로그인 오픈 API
서비스 환경 ①

환경 추가

PC 웹

서비스 URL

서비스 URL에서: (O) http://naver.com (X) http://www.naver.com

서비스 URL에서: (O) http://naver.com (X) http://www.naver.com

서비스 URL값이 잘못 입력되어 있으면 정확한 값으로 수정하실 때 까지 네이버 로그인 사용이 일시적으로 제한됩니다.

불법/음란성 사이트 등 이용약관에 위배되는 사이트의 경우, 이용이 제한될 수 있습니다.

서비스하려는 사이트 URL과 동일한 사이트 URL로 해주셔야 **네이버 로그인** 배지가 노출됩니다.

네이버 로그인

Callback URL (최대 5개)

리다이렉트URL적기

텍스트 폼 우측 끝의 '+' 버튼을 누르면 행이 추가되며, '-' 버튼을 누르면 행이 삭제됩니다.

Callback URL은 네이버 로그인 후 이동할 페이지 URL입니다. Callback URL값이 잘못 입력되어 있으면 정확한 값으로 수정하실 때 까지 네이버 로그인 사용이 일시적으로 제한됩니다.

입력한 주소와 다른 Callback URL로 리다이렉트 될 경우, 이용이 제한될 수 있습니다.

- 테스터 ID 등록 (테스터만 로그인 및 회원가입 가능함)

queant

개요	API 설정	네이버 로그인 검수상태	멤버관리	로그인 통계	API 통계	Playground (Beta)
----	--------	-----------------	------	--------	--------	-------------------

관리자 ID 등록(최대 3개)

+

애플리케이션 관리자를 3명까지 설정함으로써 개발자 부채시 다른 사람이 애플리케이션 설정을 할 수 있습니다.

아울러 애플리케이션을 테스트할 수 있는 테스트 아이디를 최대 20개까지 등록할 수 있도록 함으로써 배포전에 테스트할 수 있습니다.

테스터 ID 등록(최대 20개)

—

—

—

—

+

▼ API URI

▼ 로그인 url

https://nid.naver.com/oauth2.0/authorize?client_id=네이버 API 클라이언트 ID값 &redirect_uri=네이버 API에서 설정한 Redirect URI &response_type=code

소셜 로그인 API



8

- 네이버 API 클라이언트 ID 값

queant

개요	API 설정	네이버 로그인 검수상태	멤버관리	...
----	--------	-----------------	------	-----

애플리케이션 정보

Client ID	
Client Secret	

- 네이버 API에서 설정한 Redirect URI

로그인 오픈 API
서비스 환경 ①

불법/음란성 사이트 등 이용약관에 위배되는 사이트의 경우, 이용이 제한될 수 있습니다.
서비스하려는 사이트 URL과 동일한 사이트 URL로 해주셔야 **네이버 로그인** 배지가 노출됩니다.

네이버 로그인
Callback URL (최대 5개)

리다이렉트URL적기

텍스트 폼 우측 끝의 '+' 버튼을 누르면 행이 추가되며, '-' 버튼을 누르면 행이 삭제됩니다.
Callback URL은 네이버 로그인 후 이동할 페이지 URL입니다. Callback URL값이 잘못 입력되어 있으면 정확한 값으로 수정하실 때 까지 네이버 로그인 사용이 일시적으로 제한됩니다.
입력한 주소와 다른 Callback URL로 리다이렉트 될 경우, 이용이 제한될 수 있습니다.

▼ 소셜 API의 Access Token 요청 URI

<https://nid.naver.com/oauth2.0/token>



- Body에 아래 값 담아서 **POST** 보내기
 - grant_type
"authorization_code" → 문자열 그 자체
 - client_id
카카오 API 클라이언트 ID 값 (위에 사진 첨부함)

- client_secret

queant

개요	API 설정	네이버 로그인 검수상태	멤버관리	...
----	--------	-----------------	------	-----

애플리케이션 정보

Client ID	
Client Secret	 <input type="button" value="재발급"/>

- redirect_uri
카카오 API에서 설정한 Redirect URI (위에 사진 첨부함)
- state
네이버의 경우 프론트에서 Authorization Code와 함께 state값도 넘어줌, 이 값을 넣으면 됨
- code
프론트에서 넘겨받은 Authorization Code

▼ 소셜 API의 사용자 정보 요청 URI

<https://openapi.naver.com/v1/nid/me>

- Access Token 값을 헤더에 설정

```
HttpHeaders headers = new HttpHeaders();
headers.setContentType(MediaType.APPLICATION_FORM_URLENCODED);
headers.set("Authorization", "Bearer " + accessToken);
```