이게멍냥

포팅 메뉴얼

팀 C101	김수빈(팀장), 고요한, 김대현, 김윤민, 서은지, 이동주
담당 컨설턴트	박찬국 컨설턴트
프로젝트 기간	2022.08.29 ~ 2022.10.07 [특화, 6 주]

삼성청년 SW 아카데미
광주캠퍼스 7 기

목차

1. 프로젝트 기술 스택

1) 이슈 관리

- Jira

2) 형상 관리

- Gitlab

3) 커뮤니케이션

- Mattermost
- Notion
- Webex

4) 개발 환경

가. OS

- Window 10

나. IDE

- IntelliJ 7.5
- Visual Studio Code 1.70.0
- UI/UX : Figma

다. Database

- MySQL 8.0.30
- Redis 7.0.5

라. Server

- AWS EC2(Ubuntu 20.04.4 LTS)

마. Backend

- Java 1.8
- Spring Boot 2.7.3
- Spring Data JPA 2.7.3
- Spring Security 5.7.3

바. Frontend

- HTML5, CSS3, JS(ES6)

- React 18.2.0
- Bootstrap 5.2.0
- Node js 16.15.1

사. AI

- cuDNN 8.4.1
- CUDA 11.3
- Tensorflow
- Flask
- GPU 서버

  Ubuntu 20.04.4 LTS

  CUDA 11.7

  cuDNN

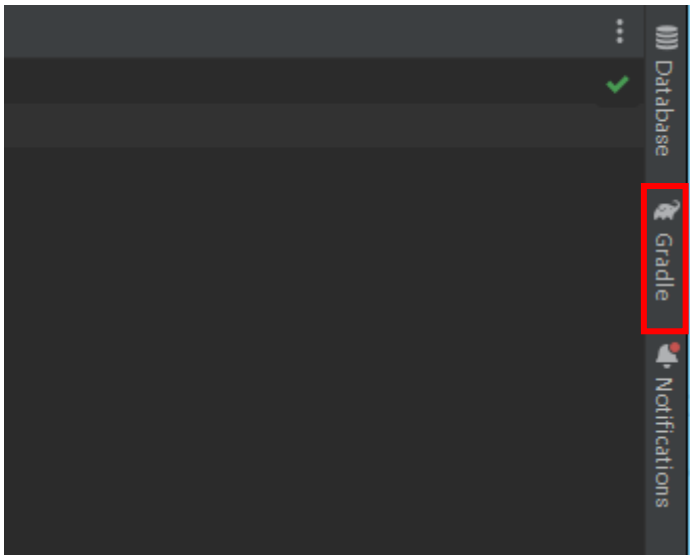
아. Deployment

- Docker 20.10.18
- Jenkins 2.361.1
- Docker Compose 1.25.0
- Nginx 1.21.6

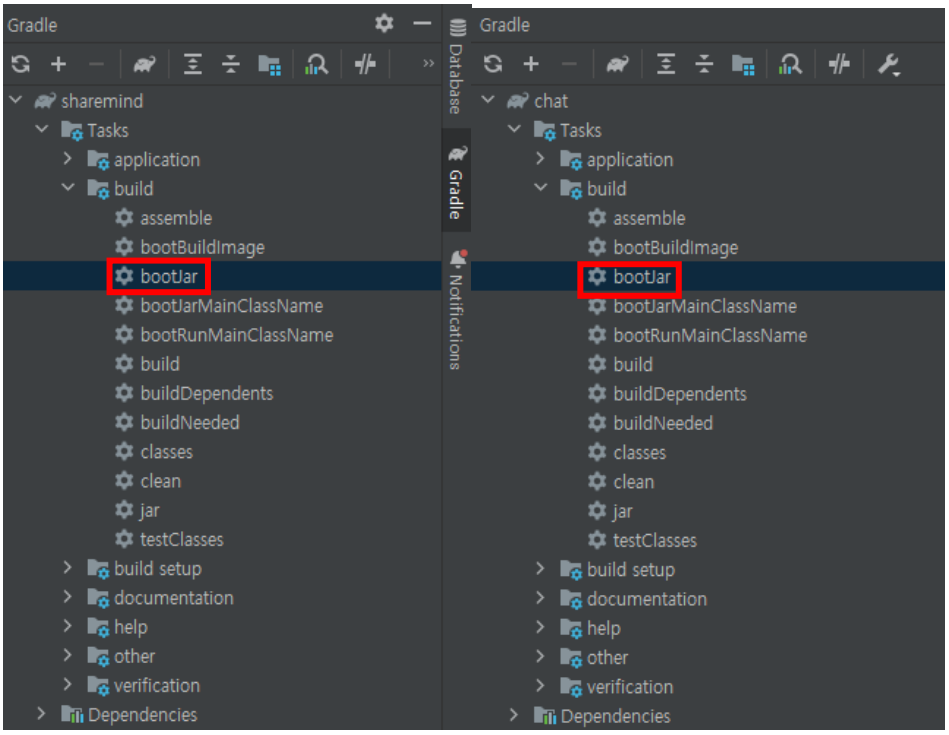
2. 빌드 상세

1) Backend

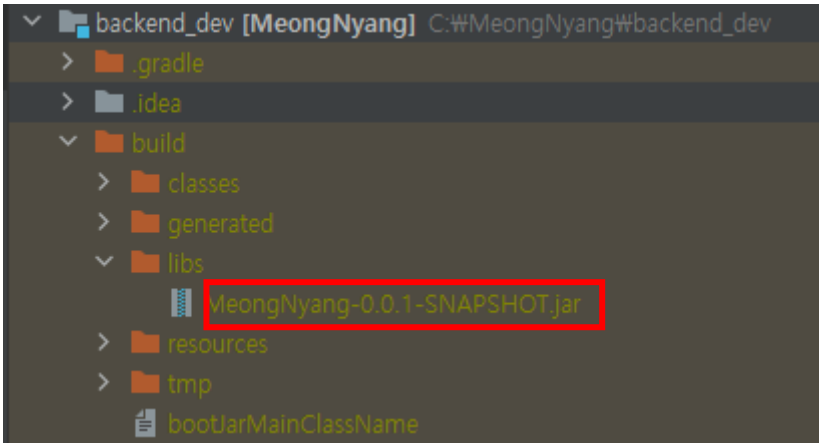
- 화면 우측 Gradle 클릭



- 프로젝트 하위 Tasks > build > Bootjar 실행



- build > libs 에 생성된 jar 확인



2) Frontend

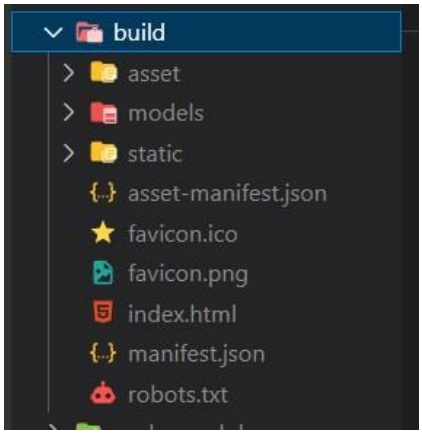
- node_modules 를 위한 install

```
npm install
```

- 빌드하기

```
npm run build
```

- root>build 에 빌드 산출물 존재



3. 배포 특이사항

젠킨스에 프론트엔드와 백엔드, 이미지서버 아이템을 각각 등록하고, GitLab webhook 을 이용해 각 브랜치에 변동사항이 push 될 때마다 자동으로 배포되도록 구성합니다. Docker Compose 를 이용하여 Docker 컨테이너 관리를 용이하게 하였습니다.

1) docker-compose.yml

- Redis, MySQL 은 도커 허브에 있는 공식 이미지를 가져와 컨테이너를 실행합니다.

```
version: '3'

services:
  redis_db:
    container_name: redis_db
    image: redis
    ports:
      - "6379:6379"
    networks:
      - backend_net

  mysql_db:
    container_name: mysql_db
    image: mysql
    environment:
      MYSQL_DATABASE: meongnyang
      MYSQL_ROOT_HOST: '%'
      MYSQL_ROOT_PASSWORD: 7rlTkxmqnrmysql3306
      TZ: 'Asia/Seoul'
    ports:
      - "3306:3306"
    command:
      - "mysqld"
      - "--character-set-server=utf8mb4"
      - "--collation-server=utf8mb4_unicode_ci"
    networks:
      - backend_net
```

```
  backend_api:
    container_name: backend_api
    image: 0ej4613/api-image
    ports:
      - "8080:8080"
    environment:
      SPRING_DATASOURCE_URL: jdbc:mysql://mysql_db:3306/meongnyang?serverTimezone=Asia/Seoul&useUnic
ode=true&characterEncoding=UTF-8
      SPRING_DATASOURCE_USERNAME: "root"
      SPRING_DATASOURCE_PASSWORD: "7rlTkxmqnrmysql3306"
    depends_on:
      - mysql_db
      - redis_db
    networks:
      - backend_net

  frontend:
    container_name: frontend
    image: 0ej4613/front-image
    ports:
      - "3000:3000"
    depends_on:
      - img_server
    networks:
      - frontend_net

  img_server:
    container_name: img_server
    image: 0ej4613/img-image
    ports:
      - "3003:3003"
    networks:
      - frontend_net

networks:
  backend_net:
  frontend_net:
```

2) Frontend

- Dockerfilie 이 있는 해당 디렉토리로 이동하여 도커 이미지를 빌드합니다. 해당 이미지를 도커허브로 푸시하고, 기존 이미지를 제거합니다.

```
cd frontend_dev/animal
docker build --platform amd64 -t 0ej4613/front-image:latest
docker login -u 0e4613 -p <password>
docker push 0ej4613/front-image:latest
docker rmi 0ej4613/front-image:latest
```

- 빌드 후 조치로 도커허브에 로그인하고, 이미지를 pull 해 옵니다. 실행 중이던 도커 컨테이너를 중지 및 삭제하고, 새로운 이미지로 도커 컨테이너를 실행합니다.

```
docker login -u 0ej4613 -p <password>

docker pull 0ej4613/front-image
docker-compose stop frontend
docker rm frontend
docker-compose up frontend
```

3) Backend

- Dockerfilie 이 있는 해당 디렉토리로 이동하여 프로젝트 빌드 후, 도커 이미지를 빌드합니다. 해당 이미지를 도커허브로 푸시하고, 기존 이미지를 제거합니다.

```
cd backend_dev
./gradlew clean build
docker build --platform amd64 -t 0ej4613/api-image:latest
docker login -u 0e4613 -p <password>
docker push 0ej4613/api-image:latest
docker rmi 0ej4613/api-image:latest
```

- 빌드 후 조치로 도커허브에 로그인하고, 이미지를 pull 해 옵니다. 실행 중이던 도커 컨테이너를 중지 및 삭제하고, 새로운 이미지로 도커 컨테이너를 실행합니다.

```
docker login -u 0ej4613 -p <password>

docker pull 0ej4613/api-image
docker-compose stop backend-api
docker rm backend_api
docker-compose up backend-api
```

4) 이미지 서버

- Dockerfilie 이 있는 해당 디렉토리로 이동하여 도커 이미지를 빌드합니다. 해당 이미지를 도커허브로 푸시하고, 기존 이미지를 제거합니다.

```
cd image_server/upload
docker build --platform amd64 -t 0ej4613/img-image:latest
docker login -u 0e4613 -p <password>
docker push 0ej4613/img-image:latest
docker rmi 0ej4613/img-image:latest
```

- 빌드 후 조치로 도커허브에 로그인하고, 이미지를 pull 해 옵니다. 실행 중이던 도커 컨테이너를 중지 및 삭제하고, 새로운 이미지로 도커 컨테이너를 실행합니다.

```
docker login -u 0ej4613 -p <password>

docker pull 0ej4613/img-image
docker-compose stop img_server
docker rm img_server
docker-compose up img_server
```

5) AI

가. Flask 서버 및 모델 설정

- skin/eye, dog/cat 모델의 저장 경로에 맞게 model 내의 경로를 설정해줍니다.
- 모델을 학습시킬 때 사용한 사이즈에 맞게 SIZE 를 설정해줍니다.

```
def skin_dog_diagnosis(imgurl):

    print(imgurl)
    model = keras.models.load_model("./models/Dog_Skin_Model.h5") # 강아지 피부 모델 경로

    #예측시킬 이미지 데이터를 넣을 변수
    data = np.ndarray(shape=(1, 64, 64, 3), dtype=np.float32)  #모델 변경시 사이즈 조절

    #예측시킬 이미지 로딩
    image = img_load(imgurl)



    # 크롭 해 올 사이즈 정해서 (왼쪽,위,오른쪽,아래)
    image = image.crop((10,10,100,100))

    #똑같은 이미지 크기로 변경
    size = (64, 64)  #모델 변경시 사이즈 조절
    image = ImageOps.fit(image, size, Image.ANTIALIAS)

    #numpy 타입으로 변경
    image_array = np.asarray(image)
    normalized_image_array = (image_array.astype(np.float32) / 127.0) - 1

    # 불러온 numpy 타입의 이미지를 변수에 대입
    data[0] = normalized_image_array

    # 예측
    prediction = model.predict(data)
    return cut3(skin_dog_naming(prediction.astype(float).tolist()))
```

나. 가상환경 접속 및 실행

- 가상환경 리스트 확인

```
ubuntu@ip-172-26-11-39:~$ conda env list
# conda environments:
#
base                  *  /home/ubuntu/anaconda3
ai_server                /home/ubuntu/anaconda3/envs/ai_server
```

- 가상환경 접속

```
ubuntu@ip-172-26-11-39:~$ conda activate ai_server
(ai_server) ubuntu@ip-172-26-11-39:~$
```

다. 필요 패키지 설치

- 경로 이동

```
(ai_server) ubuntu@ip-172-26-11-39:~$ cd ai
(ai_server) ubuntu@ip-172-26-11-39:~/ai$
```

- requirements.txt 를 이용하여 필요 패키지 모두 설치

```
(ai_server) ubuntu@ip-172-26-11-39:~/ai$ pip install -r requirements.txt
Requirement already satisfied: click==8.1.3 in /home/ubuntu/anaconda3/envs/ai_server/lib/python3.9/site-p
ackages (from -r requirements.txt (line 1)) (8.1.3)
Requirement already satisfied: Flask==2.2.2 in /home/ubuntu/anaconda3/envs/ai_server/lib/python3.9/site-p
ackages (from -r requirements.txt (line 2)) (2.2.2)
Requirement already satisfied: Flask-Cors==3.0.10 in /home/ubuntu/anaconda3/envs/ai_server/lib/python3.9/
site-packages (from -r requirements.txt (line 3)) (3.0.10)
Requirement already satisfied: importlib-metadata==4.12.0 in /home/ubuntu/anaconda3/envs/ai_server/lib/py
thon3.9/site-packages (from -r requirements.txt (line 4)) (4.12.0)
Requirement already satisfied: itsdangerous==2.1.2 in /home/ubuntu/anaconda3/envs/ai_server/lib/python3.9
/site-packages (from -r requirements.txt (line 5)) (2.1.2)
Requirement already satisfied: Jinja2==3.1.2 in /home/ubuntu/anaconda3/envs/ai_server/lib/python3.9/site-
packages (from -r requirements.txt (line 6)) (3.1.2)
Requirement already satisfied: MarkupSafe==2.1.1 in /home/ubuntu/anaconda3/envs/ai_server/lib/python3.9/s
ite-packages (from -r requirements.txt (line 7)) (2.1.1)
Requirement already satisfied: six==1.16.0 in /home/ubuntu/anaconda3/envs/ai_server/lib/python3.9/site-pa
ckages (from -r requirements.txt (line 8)) (1.16.0)
Requirement already satisfied: Werkzeug==2.2.2 in /home/ubuntu/anaconda3/envs/ai_server/lib/python3.9/sit
e-packages (from -r requirements.txt (line 9)) (2.2.2)
Requirement already satisfied: zipp==3.8.1 in /home/ubuntu/anaconda3/envs/ai_server/lib/python3.9/site-pa
ckages (from -r requirements.txt (line 10)) (3.8.1)
```

라. 경로 이동 후 백그라운드에서 실행

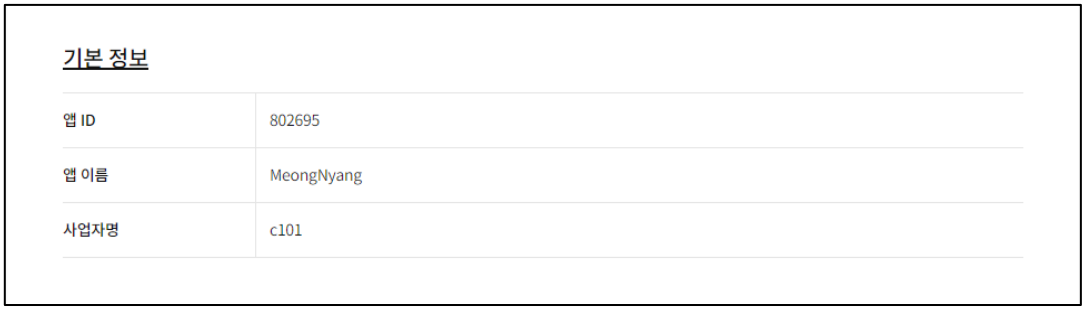
```
(ai_server) ubuntu@ip-172-26-11-39:~/ai$ cd server
(ai_server) ubuntu@ip-172-26-11-39:~/ai/server$ nohup python3 mungyang.py &
```

4. 외부 서비스

카카오

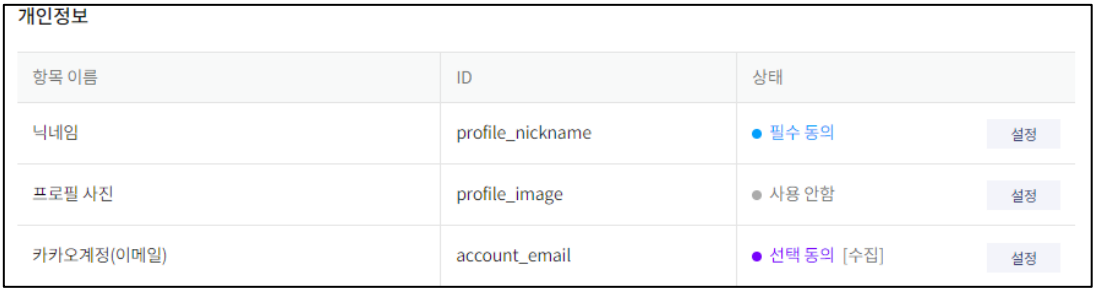
카카오 로그인으로 회원 가입 절차를 없애 서비스의 접근성을 높였습니다.

1) 애플리케이션 추가

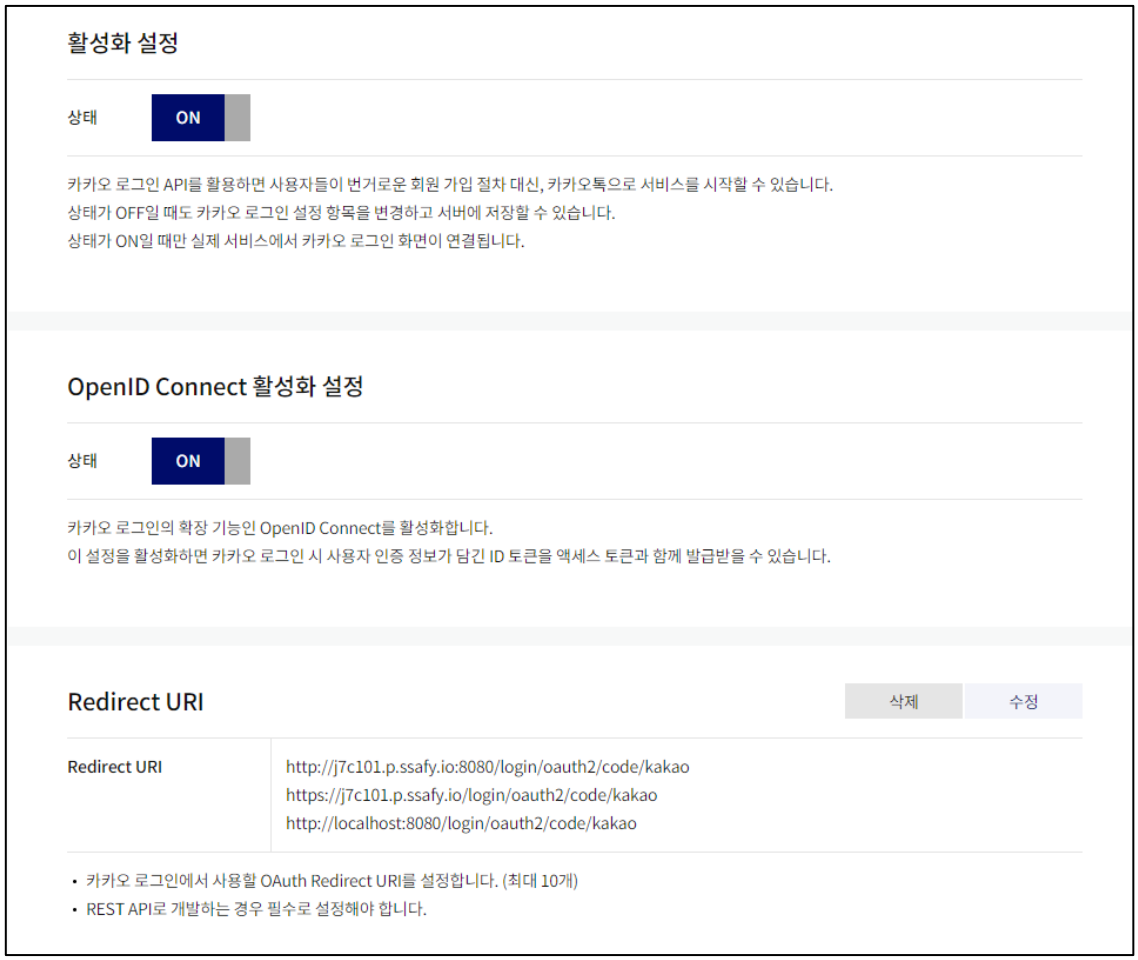


2) 도메인 등록

3) 개인정보 및 접근권한 동의 항목 설정



4) 카카오 로그인, OpenID Connect 활성화 및 Redirect URI 설정



5. EC2 세팅

1) Docker 설치

- 세팅을 위해 최신 상태로 업데이트

```
sudo apt-get update
sudo apt-get upgrade
```

- docker 설치

```
curl https://get.docker.com | sudo sh
```

2) nginx 설치

- 이미지 가져오기

```
sudo apt-get install nginx
```

6. 프로퍼티

1) Nginx 설정 변경

```
GNU nano 4.8
server {
    listen 80;
    server_name j7c101.p.ssafy.io;

    location / {
        return 308 https://$host$request_uri;
    }

}

server {
    listen 443 ssl;
    server_name j7c101.p.ssafy.io;

    ssl_certificate /etc/letsencrypt/live/j7c101.p.ssafy.io/fullchain.pem;
    ssl_certificate_key /etc/letsencrypt/live/j7c101.p.ssafy.io/privkey.pem;


    location  /login {
        proxy_pass http://j7c101.p.ssafy.io:8080/login;
    }

    location  /logout {
        proxy_pass http://j7c101.p.ssafy.io:8080/logout;
    }

    location  /token {
        proxy_pass http://j7c101.p.ssafy.io:8080/token;
    }

    location  /api {
        proxy_pass http://j7c101.p.ssafy.io:8080$request_uri;
    }

    location  /ai {
        proxy_pass http://j7c101.p.ssafy.io:5550$request_uri;
    }

    location  /upload {
        proxy_pass http://j7c101.p.ssafy.io:3003/upload;
    }

    location  /upload-multi {
        proxy_pass http://j7c101.p.ssafy.io:3003/upload-multi;
    }
}
```