냉장고를 부탁해

중간발표

팀: 넛지
팀장: 윤형로
팀원: 정기욱, 표승수, 김영주, 이민기, 황선기

서비스 소개

서비스 소개

'냉장고를 부탁해 ' 는 스마트폰 카메라를 이용해 냉장고의 식재료를 인식하고, 이를 바탕으로 해먹을 수 있는 레시피를 추천해주는 어플리케이션입니다.

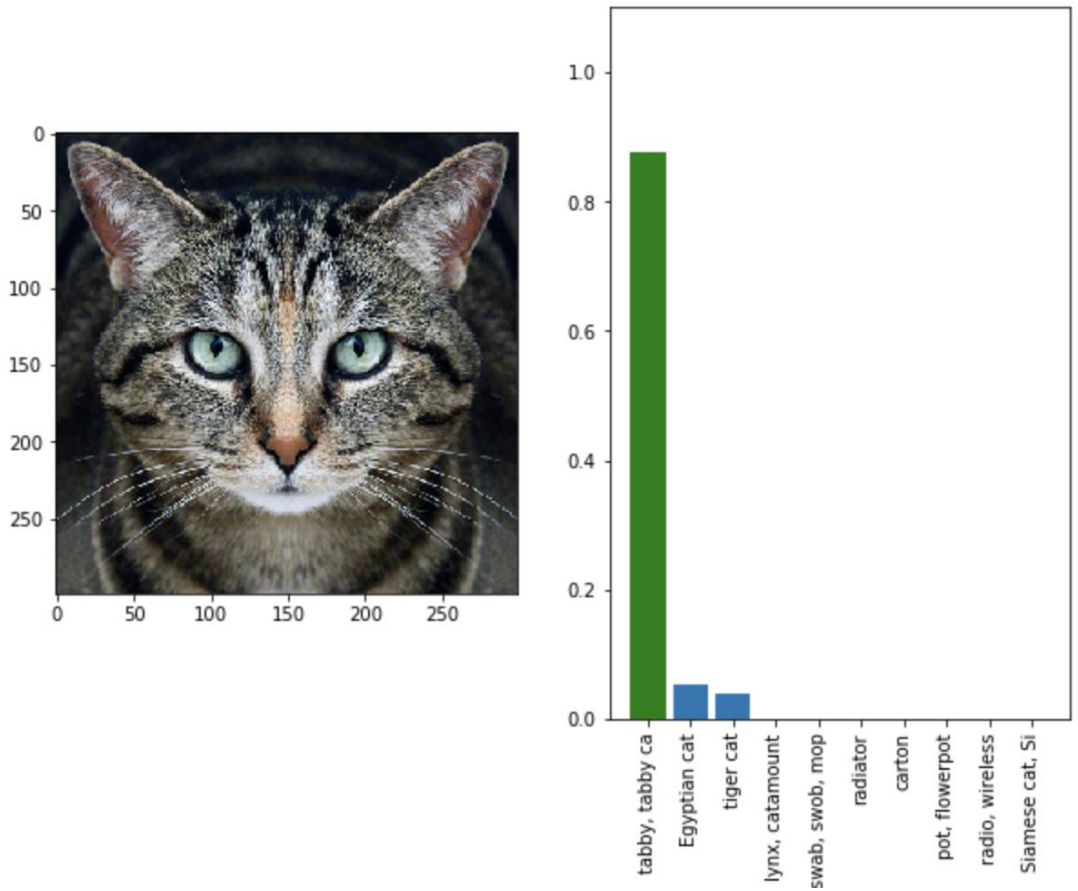
서비스 동작

1. 기존의 타이핑 위주의 검색에서 탈피하여 냉장고에 있는 식재료를 카메라로 인식하게 하여 사용자의 편의를 도모.

2. 인식된 재료와 데이터베이스에 있는 레시피를 비교하여 사용자에게 레시피 제공

머신러닝 모델 구축
및
테스트

▪ 서비스를 위해 사용한 기술



객체 분류 (Classification 예시)
출처: https://www.atyun.com/3841.html

- 머신 러닝을 이용해 학습을 시킨 ML (Machine Learning) Model을 만들어서 식재료를 인식시키는데 사용한다.
- 사용자는 스마트폰을 사용해 식재료를 스캔 할 수 있어야 하므로 동영상을 이용해 식재료를 인식하는 기술을 사용한다.

ML Model 구축 - 1. 필요한 이유

- 문제점 1 : 식재료를 인식하지 못한다.
- 문제점 2 : 식재료를 인식하는 다른 모델이 있다고 해도 한국 식재료를 인식하지 못한다. Ex) 김치, 고추장
- 따라서 Pre-Trained 모델을 이용한 새로운 모델 (기존 모델을 이용한 추가 학습) 이 반드시 필요함!

- 위의 동영상은 Keras 에서 제공하는 ResNet-50 모델을 사용하여 냉장고의 재료를 인식시킨 결과

- ML Model 구축 - 2. 구축 과정

Apple에서 제공하는 Create ML Framework를 이용해 Classifier ML Model 학습.

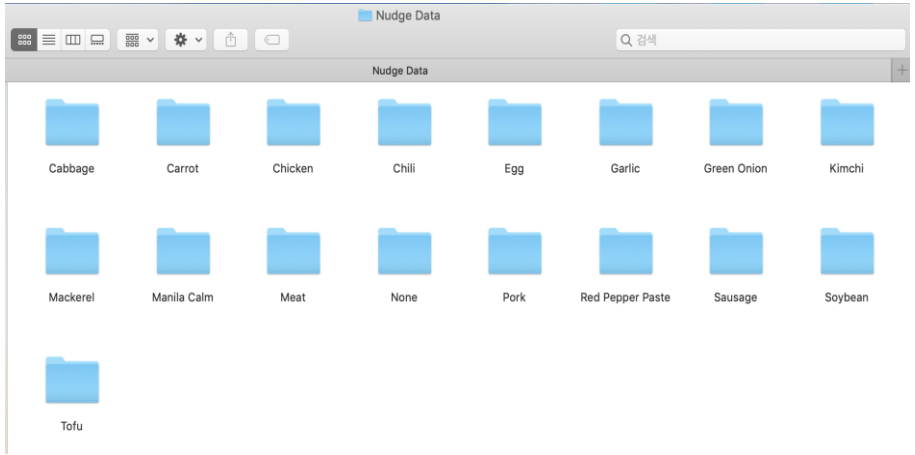
1. 모델 선정
2. 데이터 셋 준비
3. Data Augmentation
4. Training (데이터 학습)
5. Evaluation (평가)

출처 : https://developer.apple.com/documentation/createml

▪ ML Model 구축 - 2. 구축 과정

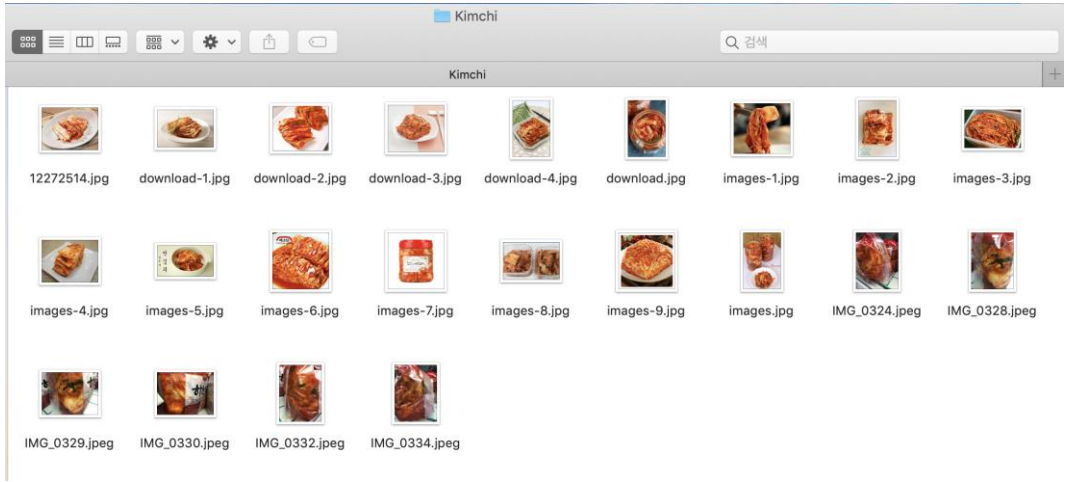
①. 모델 선정

냉장고에 많이 있는 식재료를 위주로 16가지의 식재료를 학습시킴. (초기 모델)



②. 데이터 셋 준비

배경화면에 대한 변수가 가장 적고 모델을 대표 할 수 있는 이미지를 준비함

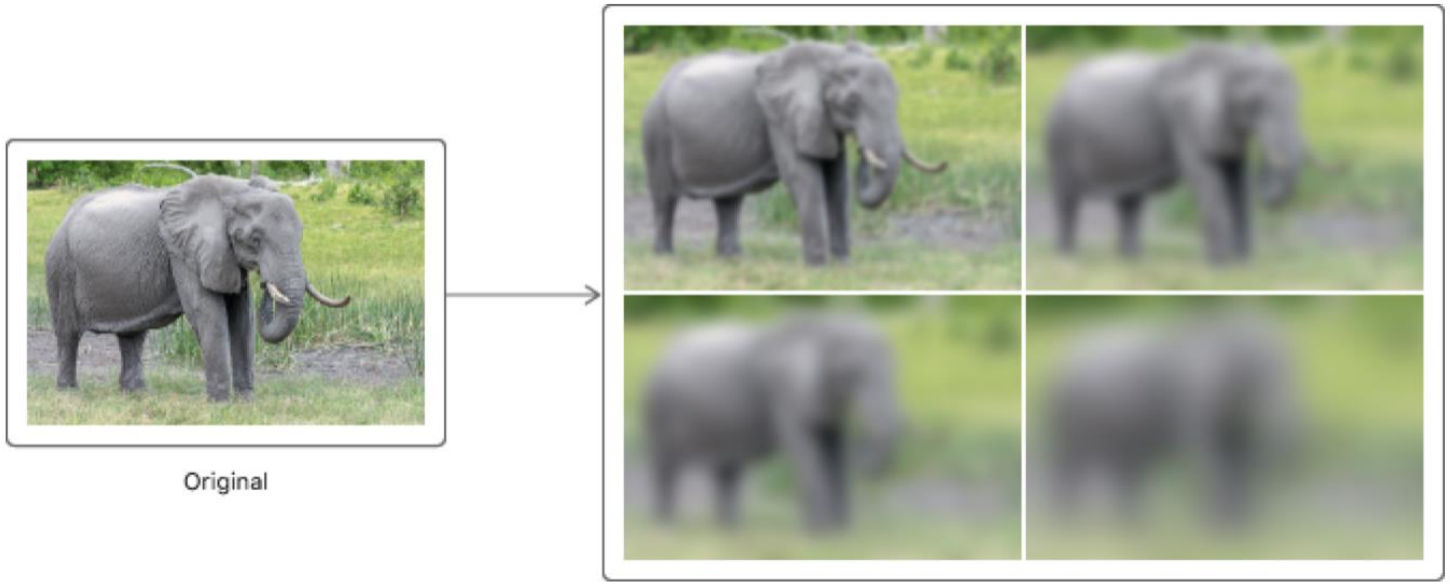


▪ ML Model 구축 - 2. 구축 과정

③. Data Augmentation

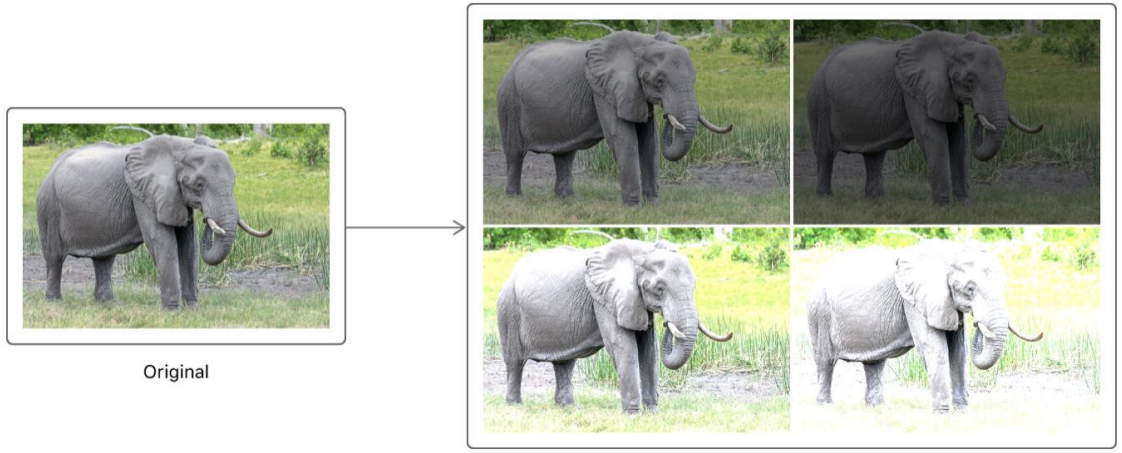
카메라 해상도, 화질, 냉장고 환경을 다각적으로 고려한 Data Augmentation을 기존 데이터 셋에 추가 시켜줌.

가장 적합하다고 생각해 선택한 Data Augmentation 기법은 총 3가지로 Blur(흐릿함), Exposure(빛 노출), Noise (이미지 노이즈) 를 선정함.

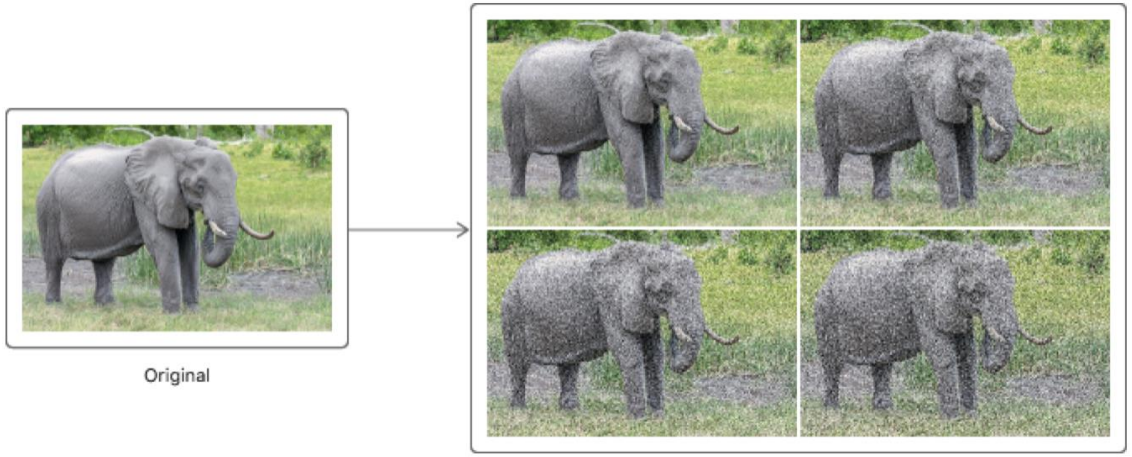


Blur(흐릿함)
:https://developer.apple.com/documentation/createml/mlimageclassifier/imageaugmentationoptions/3006499-blur

ML Model 구축 - 2. 구축 과정



Exposure(빛 노출)
:https://developer.apple.com/documentation/createml/mlimageclassifier/imageaugmentationoptions/3006502-exposure



Noise(이미지 노이즈)
:https://developer.apple.com/documentation/createml/mlimageclassifier/imageaugmentationoptions/3006499-blur

▪ ML Model 구축 - 2. 구축 과정

④,⑤ Training & Evaluation

```
4 augmented images by 'blur' to be generated.
4 augmented images by 'exposure' to be generated.
4 augmented images by 'noise' to be generated.
Automatically generating validation set from 5% of the data.
Extracting augmented image features from training data set.
Analyzing and extracting image features.
+----------------------+------------------+--------------+------------------+
| Raw Images Processed | Augmented Images | Elapsed Time | Percent Complete |
+----------------------+------------------+--------------+------------------+
VPA info: plugin is INTEL, AVD_id = 1080020, AVD_api.Create:0x112d23cf1
| 1                    | 65               | 14.37s       | 0%               |
| 2                    | 130              | 27.63s       | 0.25%            |
| 3                    | 195              | 41.08s       | 0.5%             |
| 4                    | 260              | 58.64s       | 0.75%            |
| 5                    | 325              | 1m 11s       | 1%               |
| 7                    | 455              | 1m 42s       | 1.5%             |
```

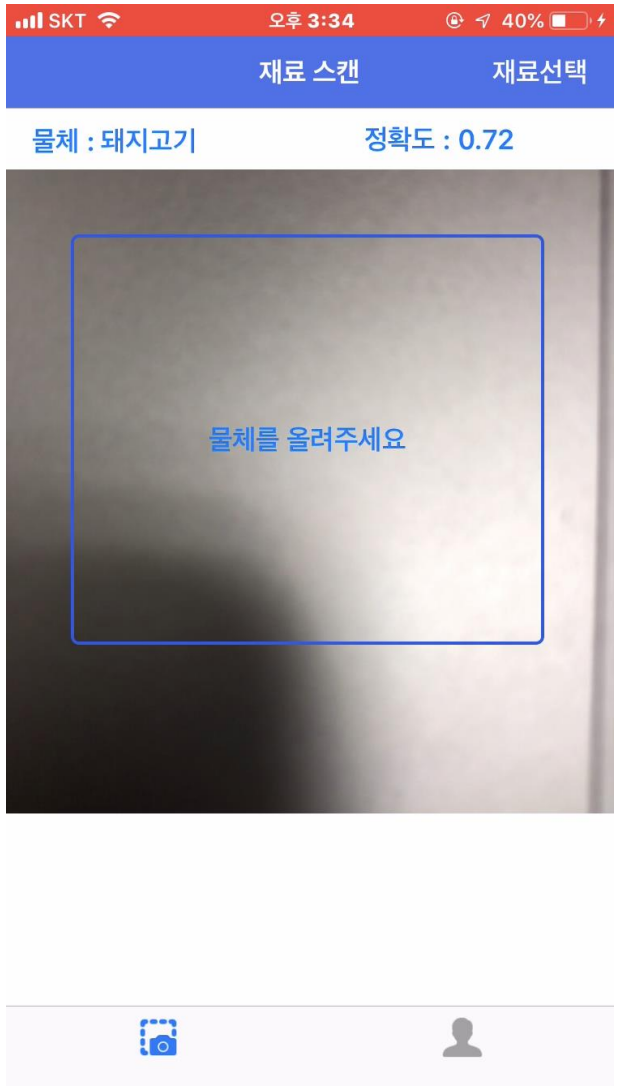
```
| 411                  | 26715            | 1h 29m       | 96.25%           |
| 412                  | 26780            | 1h 29m       | 96.25%           |
| 413                  | 26845            | 1h 30m       | 96.5%            |
| 414                  | 26910            | 1h 30m       | 96.75%           |
| 415                  | 26975            | 1h 30m       | 97%              |
| 416                  | 27040            | 1h 30m       | 97.25%           |
| 417                  | 27105            | 1h 30m       | 97.5%            |
| 418                  | 27170            | 1h 31m       | 97.75%           |
| 419                  | 27235            | 1h 31m       | 98%              |
| 420                  | 27300            | 1h 31m       | 98.25%           |
| 421                  | 27365            | 1h 31m       | 98.5%            |
| 422                  | 27430            | 1h 32m       | 98.75%           |
| 423                  | 27495            | 1h 32m       | 99%              |
| 424                  | 27560            | 1h 32m       | 99.25%           |
| 425                  | 27625            | 1h 32m       | 99.5%            |
| 426                  | 27690            | 1h 32m       | 99.75%           |
| 427                  | 27755            | 1h 33m       | 100%             |
+----------------------+------------------+--------------+------------------+
Extracting image features from validation data set.
Analyzing and extracting image features.
```

- 약 27,700장의 이미지를 학습 시킴.
- Blur, Exposure, Noise 기법을 적용 시킨 이미지.

ML Model 구축 - 3. 구축한 모델 시연 데모

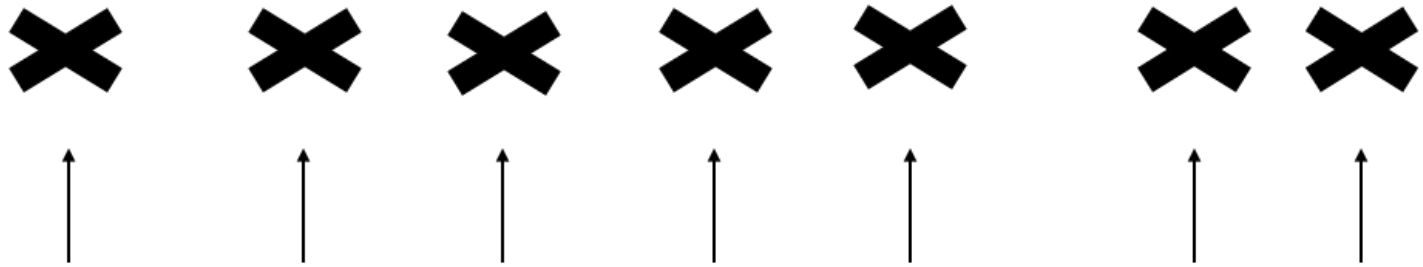
- 머신 러닝 모델을 사용하여 마트에 방문하여 학습 시킨 재료가 인식되는지 실험
- 아래에 추가되는 재료는 일정 인식률이 넘으면 아래에 추가됨
- 머신 러닝 모델과 실시간 분류에 대한 문제점 및 개선은 뒷부분에 추가 설명

ML Model 구축 - 4. 문제점 - 문제 인식



- 식재료가 아니거나 학습시키지 않은 재료를 인식시켰을 때 해당하지 않는 식재료가 아웃풋으로 도출됨.
- 또한, 아무것도 없는 화면을 비추더라도 식재료 중 하나로 인식되며 사람이 이해하기 힘든 아웃풋을 내보냄.

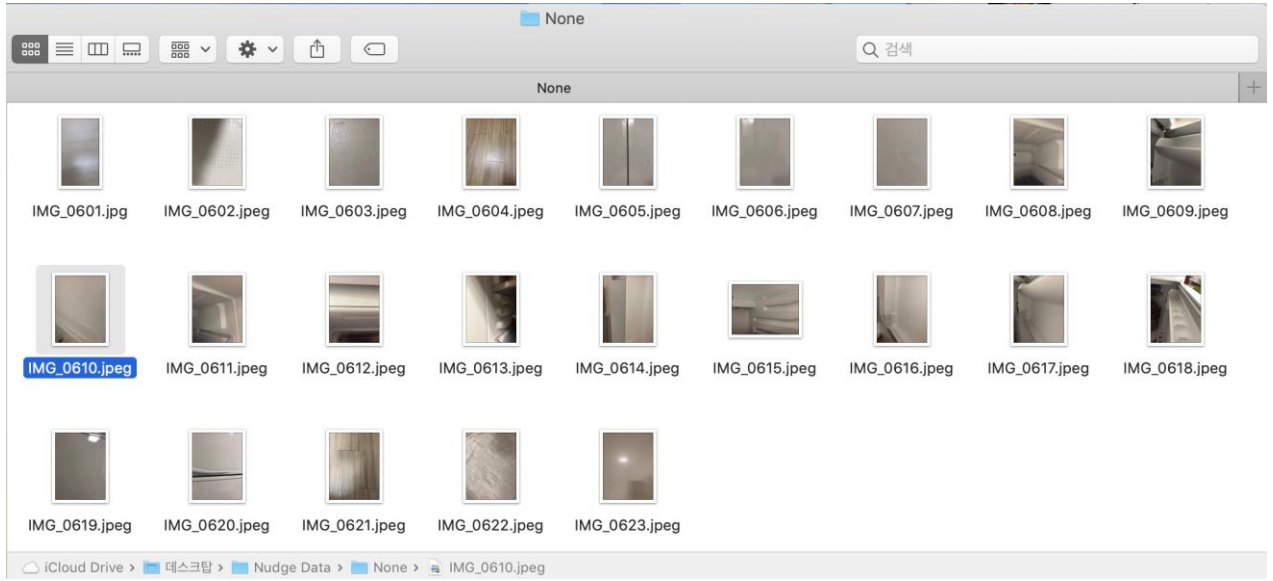
ML Model 구축 - 4. 문제점 - 문제 분석



-> 결과는..?

- 학습시킨 카테고리에 존재하지 않는 재료나 아무것도 없는 화면을 비추면 카테고리 내에 있는 가장 비슷한 (색, 형태) 물체를 아웃풋으로 내보내는 것으로 추정됨.
- 냉장고 내에는 식재료만 있는 것이 아니라 냉장고 외벽, 냉장고 선반 등 식재료의 배경이 되는 것 또한 가장 비슷한 식재료로 인식하는 경향을 보임 (냉장고 외벽 -> 양배추(흰색)).

ML Model 구축 - 4. 문제점 - 해결책



- 카메라가 냉장고의 외벽이나 아무것도 없는 것을 인식할 때를 대비해 '공백 화면 및 냉장고 환경' 을 카테고리로 하여 학습 시켜 줌.
- 학습 결과 카테고리 내의 재료 인식률이 높아졌고 없는 재료를 더 이상 인식하지 않음.
- 이 후, 카테고리를 추가하고 다양한 변수를 고려한 새로운 모델을 구축할 예정

ML Model 구축 - 5. 문제 해결 및 최종 시연

NudgeMLModel06.mlmodel

냉장고 환경을 학습시키지 않은 Nudge Model 6

NudgeMLModel07.mlmodel

문제를 해결한 Nudge Model 7

Django REST framework를 통한 레시피 추천

▪ Django Rest Framework - 1) 개요

REST의 개념

REST가 필요한 이유

- '애플리케이션 분리 및 통합'
- '다양한 클라이언트의 등장'
- 최근의 서버 프로그램은 다양한 브라우저와 안드로이폰, 아이폰과 같은 모바일 디바이스에서도 통신을 할 수 있어야 한다.
- 이러한 멀티 플랫폼에 대한 지원을 위해 서비스 자원에 대한 아키텍처를 세우고 이용하는 방법을 모색한 결과, REST에 관심을 가지게 되었다.

출처: https://gmlwjd9405.github.io/2018/09/21/rest-and-restful.html

Django Rest Framework - 2) serializers.py

Serializer란?

: Queryset과 모델 인스턴스와 같은 복잡한 데이터를 json, xml 또는 다른 콘텐츠 유형으로 쉽게 변환할 수 있도록 하는 장치.

정석적인 serializer 작성 과정

```python
class SnippetSerializer(serializers.Serializer):
    id = serializers.IntegerField(read_only=True)
    title = serializers.CharField(required=False, allow_blank=True, max_length=100)
    code = serializers.CharField(style={'base_template': 'textarea.html'})
    linenos = serializers.BooleanField(required=False)
    language = serializers.ChoiceField(choices=LANGUAGE_CHOICES, default='python')
    style = serializers.ChoiceField(choices=STYLE_CHOICES, default='friendly')

    def create(self, validated_data):
        """
        Create and return a new `Snippet` instance, given the validated data.
        """
        return Snippet.objects.create(**validated_data)

    def update(self, instance, validated_data):
        """
        Update and return an existing `Snippet` instance, given the validated data.
        """
        instance.title = validated_data.get('title', instance.title)
        instance.code = validated_data.get('code', instance.code)
        instance.linenos = validated_data.get('linenos', instance.linenos)
        instance.language = validated_data.get('language', instance.language)
        instance.style = validated_data.get('style', instance.style)
        instance.save()
        return instance
```

DRF에서 제공하는 ModelSerializer

```python
class SnippetSerializer(serializers.ModelSerializer):
    class Meta:
        model = Snippet
        fields = ('id', 'title', 'code', 'linenos', 'language', 'style')
```

출처: https://www.django-rest-framework.org/tutorial/1-serialization/

Django Rest Framework - 2) serializers.py

① findSerializer

```python
# 사용자의 재료에 따라 첫 번째로 추천 레시피들을 전달할 때 쓰는 serializer
class findSerializer(serializers.HyperlinkedModelSerializer):
    class Meta:
        model = MyappFood
        fields = ('id', 'recipe_name')
```

- 사용자의 재료를 바탕으로 여러 레시피를 추천할 때

② selectSerializer

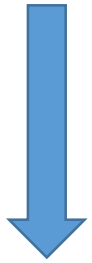
```python
# 사용자가 최종 선택한 레시피의 모든 정보를 전달 할 때 쓰는 serializer
class selectSerializer(serializers.HyperlinkedModelSerializer):
    class Meta:
        model = MyappFood
        fields = ('recipe_name', 'ingredients', 'direction')
```

- 사용자가 추천 레시피 중 하나를 선택하면, 해당 레시피의 모든 정보를 보여줄 때

▪ Django Rest Framework - 3) views.py

① findViewset()

```
사용자 재료 = {
                "재료1": "우유",
                "재료2": "달걀",
                "재료3": "닭",
                "재료4": "두부"
              }
```

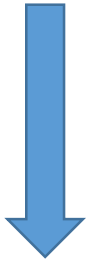


비교

```
추천 목록 = {
              "레시피1": "단호박 닭고기 들깨탕",
              "레시피2": "닭고기 김치찌개",
              "레시피3": "나가사키부대찌개",
              "레시피4": "크림닭"
            }
```

② selectViewset()

```
사용자 선택 = {
                "선택 레시피": 크림닭"
              }
```



objects.filter()

```
레시피 = {
          "이름": "크림닭",
          "재료": "닭가슴살, 우유, 소금, 달걀, 밀가루",
          "순서": "1. 우유에 닭가슴살을 재운다.
                   2. 닭가슴살에 소금과 후추를 뿌린다.
                   3. 팬에 버터를 녹인다.
                   4. … … …
        }
```

```python
from rest_framework import viewsets
from myapp.serializers import findSerializer, selectSerializer
from myapp.models import MyappFood

# 1차 추천 레시피 탐색
class findViewSet(viewsets.ModelViewSet):
    user_ingredient = ["우유", "달걀", "닭", "두부"]  # 사용자로 부터 입력 받은 재료 예시

    cntnum = [0 for _ in range(999)]  # 사용자 재료와 DB재료의 매치 갯수
    recipe_DB = MyappFood.objects.all()

    for i in range(0, 999):
        cntnum[i] = 0
        for j in range(0, len(user_ingredient)):
            comparing = recipe_DB[i].ingredients.find(user_ingredient[j])
            # find() 를 통해서 사용자의 재료가 DB 재료에 매칭 되는 지 확인(매칭되면 >=0, 매칭 되지 않으면 -1)
            if comparing >= 0:
                cntnum[i] += 1  # 매칭 되었을 때 갯수 ++

    maxi = 0  # 최대 매칭 레시피를 찾기 위한 변수

    for m in range(0, 999):  # 최대 매칭 레시피의 매칭 재료 개수 설정(maxi의 최신화)
        if cntnum[m] > maxi:
            maxi = cntnum[m]
            # print(maxi)
    for m in range(0, 999):
        if cntnum[m] == maxi:
            a = m
            break

    queryset = MyappFood.objects.filter(id=a+1)# 매칭 레시피 중 첫 번째 레시피만 일단 넣는다.

    for m in range(0, 999):  # maxi에 해당하는 최대 매칭 레시피 모두 출력
        if cntnum[m] == maxi:
            if(m != a):
                # 합 연산자를 통해 queryset에 매칭 레시피 병합
                queryset |= MyappFood.objects.filter(id = m + 1)

    serializer_class = findSerializer
```

MyappFood라는 DB 테이블의 모든 데이터를 recipe_DB라는 리스트에 넣는다.

① findViewset()

사용자의 재료와 DB의 재료가 몇 개 매칭되는지 확인

find()를 통해 문자열을 비교하여 매칭 재료를 탐색

cntnum[]은 매칭되는 개수를 저장하는 리스트

cntnum[]에 저장된 매칭 개수 중 최대 개수를 탐색 하여 maxi 변수에 저장

Maxi에 해당하는 레시피 출력

MyappFood.objects.filter()를 통해 해당 레시피의 집합으로만 쿼리셋 구성

마지막으로 serializing

② selectViewset()

```python
# 2차 최종 레시피 출력
class selectViewSet(viewsets.ModelViewSet):
    user_choice = ["크림닭"] # 만약 사용자가 여러 레시피 중 크림닭을 선택했다면,
    queryset = MyappFood.objects.filter(recipe_name=user_choice[0]) # filter함수를 통해 해당 쿼리셋만 받아옴
    serializer_class = selectSerializer
```

"크림닭"에 해당하는 레시피를 찾기위해

MyappFood.objects.filter()를 통해 "크림닭" 정보로 쿼리셋을 구성

마지막으로 serializing

Django Rest Framework - 4) urls.py

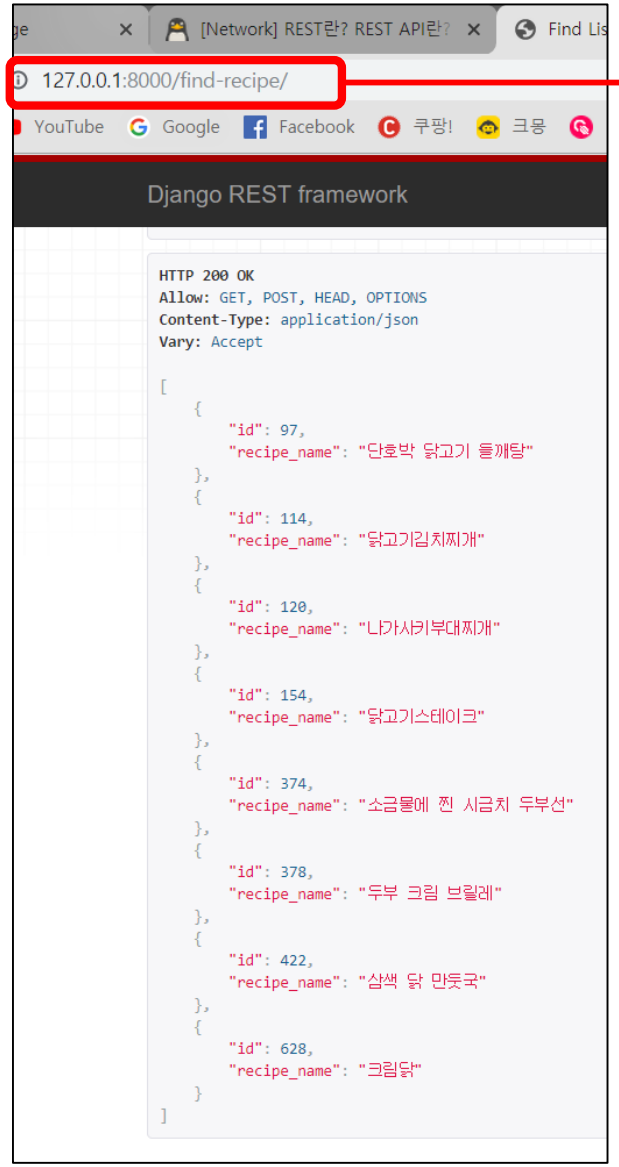
```python
from django.conf.urls import url, include
from rest_framework import routers
from myapp import views

router = routers.DefaultRouter()

router.register(r'find-recipe', views.findViewSet)
router.register(r'select-recipe', views.selectViewSet)

urlpatterns = [
    url(r'^', include(router.urls)),
    url(r'^api-auth/', include('rest_framework.urls', namespace='rest_framework')),
]
```

REST의 중요한 특징 중 하나.

URI를 통해 자원을 이름으로 구분하여 해당 자원의 상태를 주고 받음.

직관적으로 해당 **URI**가 어떤 기능을 실행하는지 파악 가능.

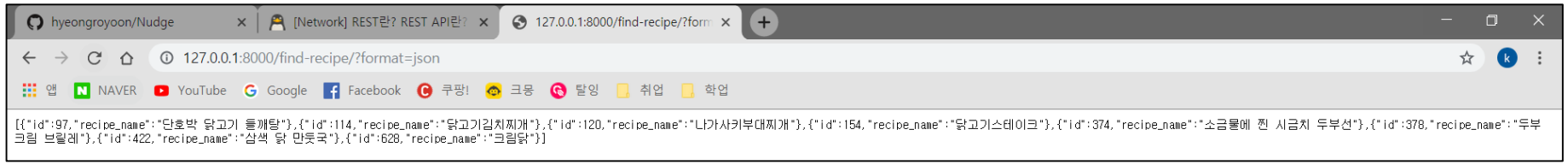
- 127.0.0.1:8000/find-recipe : findViewSet 실행
- 127.0.0.1:8000/select-recipe : selectViewSet 실행

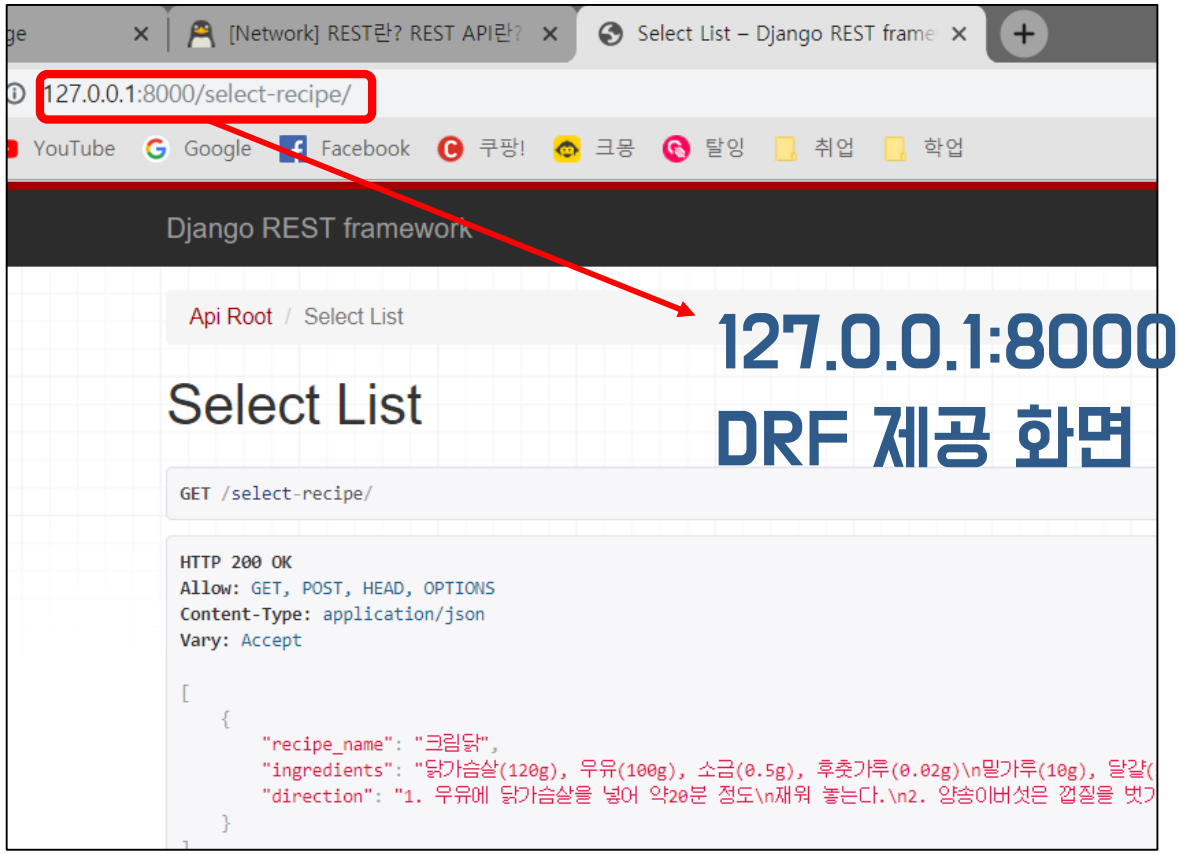


127.0.0.1:8000/find-recipe

DRF 제공 화면

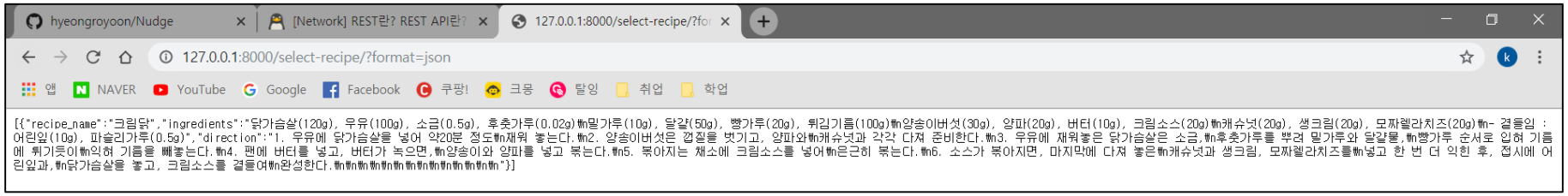
127.0.0.1:8000/find-recipe/?format=json





127.0.0.1:8000/select-recipe
DRF 제공 화면

127.0.0.1:8000/select-recipe/?format=json



▪ Django Rest Framework - 5) 차후 개선점

① 현재는 모의로 사용자와 연결이 되었다고 가정하였다.
비교 알고리즘 구동과 결과 json 파일 url 생성까지는 완료되었다.
앞으로 앱과의 실제적인 Request, Response 연동을 위해 views.py의 코드 수정이 필요하다.

② 현재는 127.0.0.1:8000/의 형태로 Local 접속만 가능하다.
wsgi.py와 settings.py의 Allowed_HOST=[]의 코드 수정을 통해 AWS와의 연동이 필요하다.

③ 현재는 레시피의 Direction(조리순서)을 제공할 때 문자열만 제공한다.
Direction을 "문자열 + 사진"으로 제공하여 사용자의 편의성을 증가 시키고자 한다.
따라서 데이터베이스의 Direction 애트리뷰트에 추가적으로 png url을 삽입해야 한다.
또한 이를 사용자에게 제공할 때, 서버에서 Direction의 "문자열 + 사진"의 순서를 맞추어 json으로 serialize하여 보낼 필요가 있다.

AWS 서버 구축

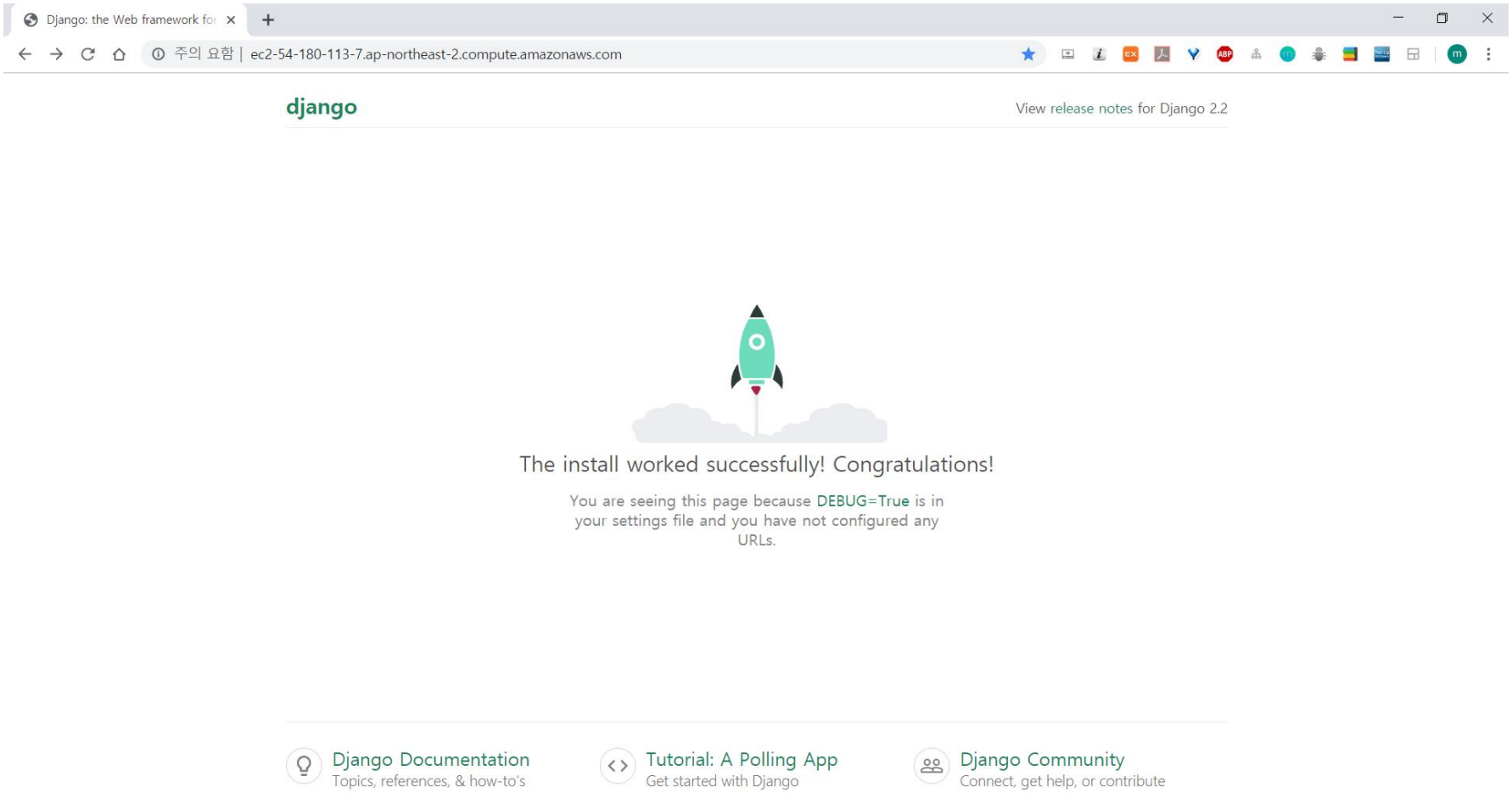
AWS Django 서버 구축(nginx, uwsgi)

```
(myvenv) → nginx ls
conf.d          koi-utf     nginx.conf     sites-available  uwsgi_params
fastcgi.conf    koi-win     proxy_params   sites-enabled    win-utf
fastcgi_params  mime.types  scgi_params    snippets
(myvenv) → nginx cd sites-available
(myvenv) → sites-available ls
default  firstproject
```

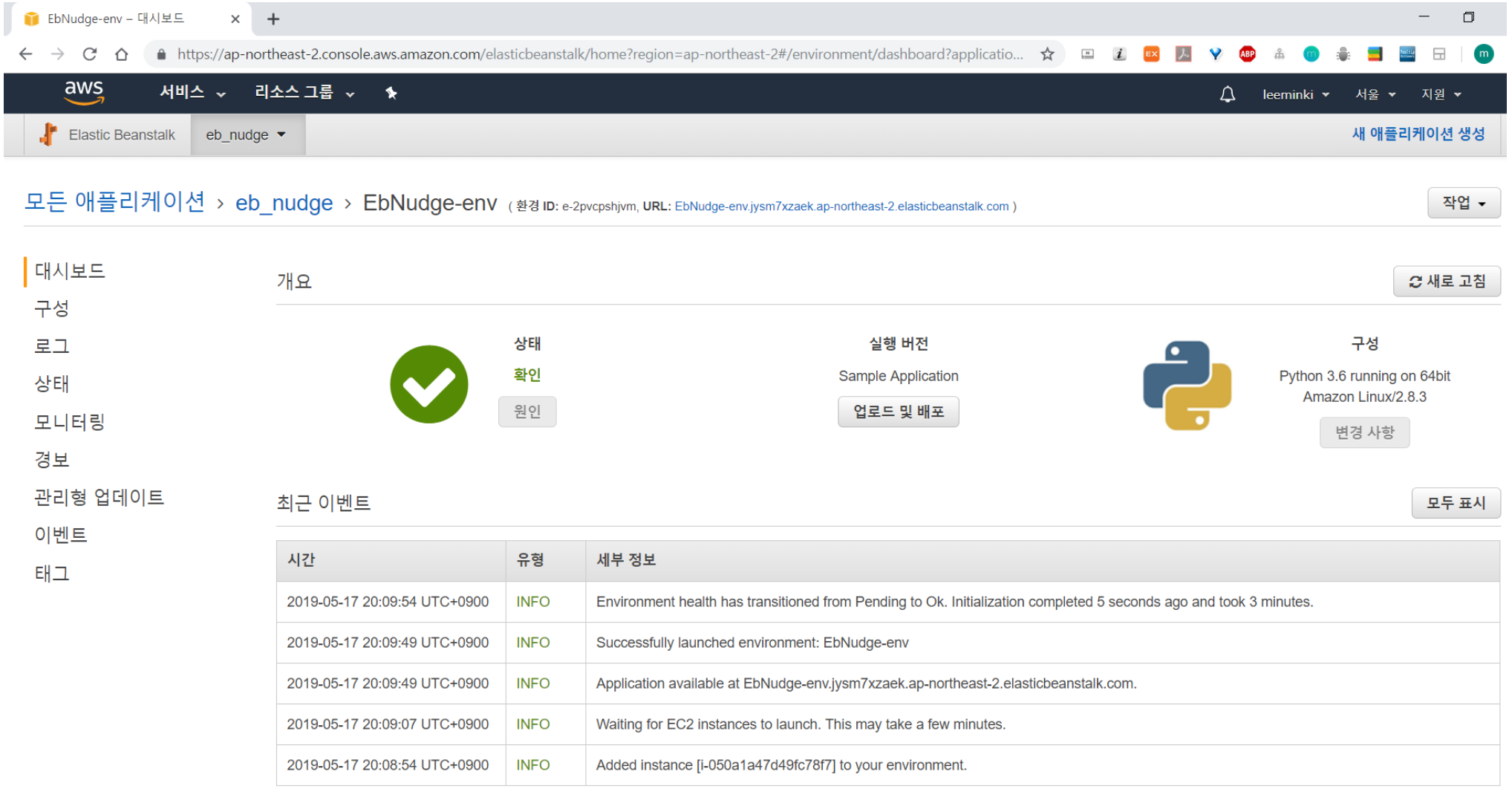
```
(myvenv) → sites-available vi firstproject
server {
    listen 80;
    server_name 54.180.113.7 ec2-54-180-113-7.ap-northeast-2.compute.amazonaws.com;

    location = /favicon.ico { access_log off; log_not_found off; }
    location /static/ {
        root /home/ubuntu/app/django/firstproject/nudge;
    }

    location / {
        include         uwsgi_params;
        uwsgi_pass      unix:/home/ubuntu/app/django/firstproject/firstproject.sock;
    }
}
```

```
(myvenv) → sites-available sudo service uwsgi start
(myvenv) → sites-available sudo service nginx start
```

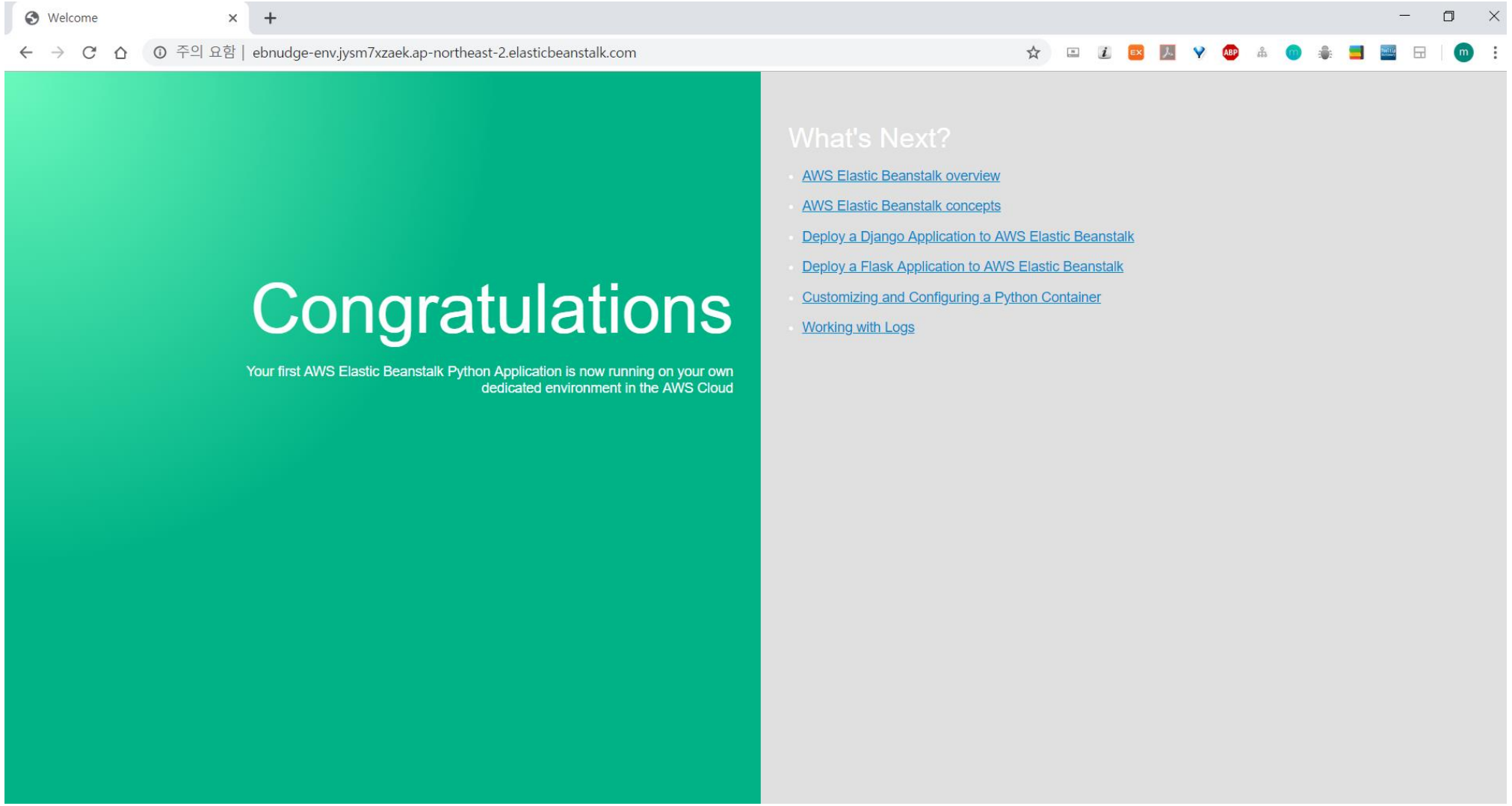
AWS Django 서버 구축(nginx, uwsgi)



AWS Elastic Beanstalk

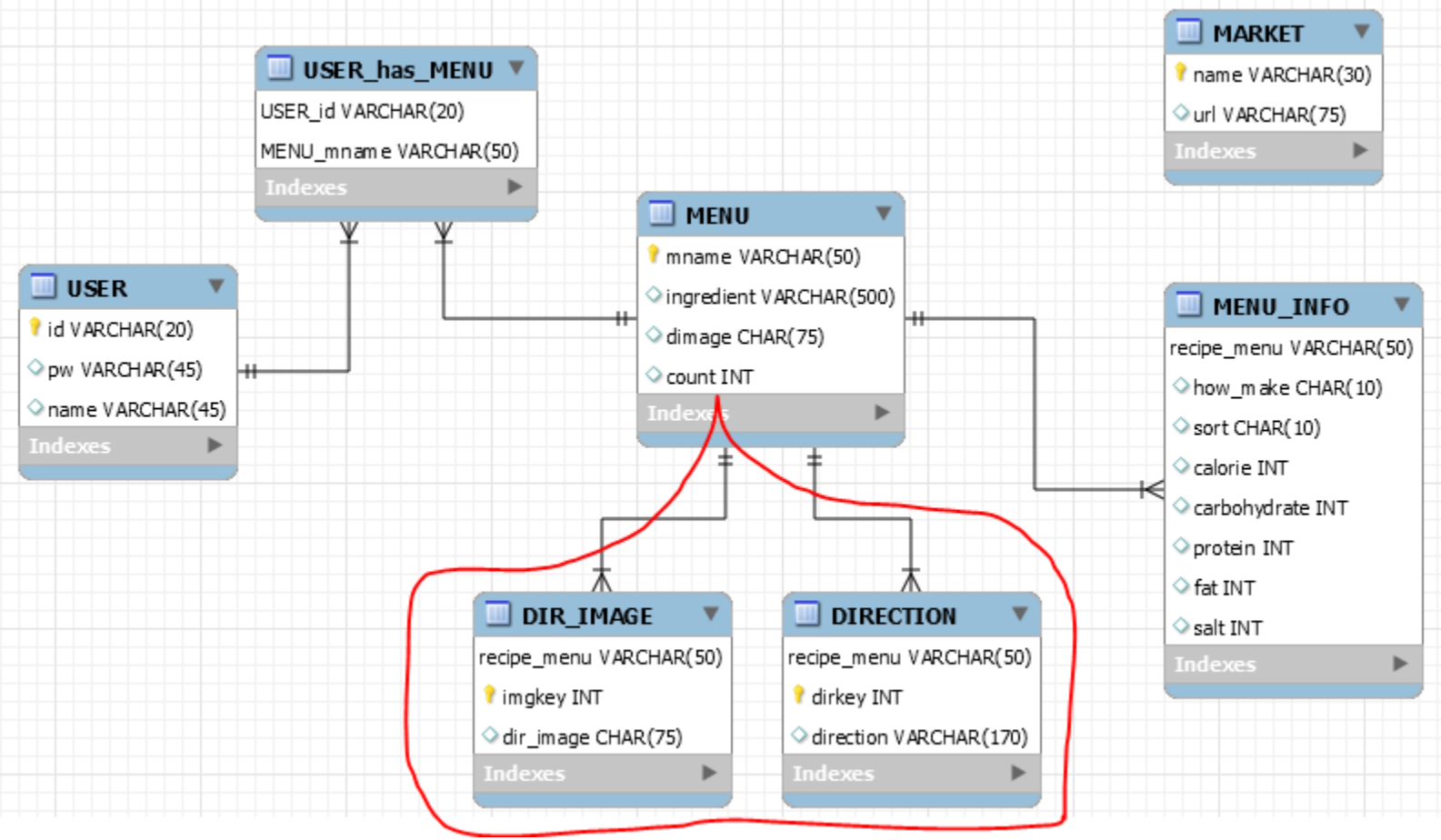


AWS Elastic Beanstalk



DATABASE 수정
및
정규화 알고리즘

데이터베이스 설계 수정



DIR_IMAGE 테이블과 DIRECTION 테이블이
MENU 테이블에서 분리되었다.

- 수정이유:

기존의 다치 애트리뷰트로 존재 했던 레시피 애트리뷰트를 MENU 테이블에서 String으로 묶어서 서버에 한번에 return하려 하였으나.

Missing value가 많은 레시피 이미지에 대한 처리가 추가적으로 필요하다고 판단하여, 메뉴 이름과 순서를 키 값으로 Left outer join의 연산을 위해 레시피 애트리뷰트를 메뉴 테이블에서 분리하여 테이블을 만들었다.

데이터베이스 정규화

• 릴레이션 스키마 R의 FD X -> A가 성립할 때마다
1. X 가 R의 수퍼키이거나.
2. A가 R의 기본 애트리뷰트이면 R은 제 3 정규형(3NF)을 갖는다.
Boyce-Codd 정규형은 위의 조건 중 2번의 경우를 허락치 않는 정규형을 의미한다.

무손실 조인 특성과 종속성 보존 특성을 만족하는 릴레이션이
좋은 관계형 데이터베이스 릴레이션임.
3nf와 bcnf 사이의 밸런스가 중요.

데이터베이스 설계 수정

- 사용자가 만들어 먹을 수 있는 메뉴 중 "고등어 찜"을 선택 하였을 때, 서버에서 DB에 요청하는 경우

```
select A.recipe_menu, A.direction, B.imgkey, B.dir_image
from (select * from direction where recipe_menu = "고등어 찜") as A
left outer join (select * from dir_image where recipe_menu = "고등어 찜") as B
on A.dirkey = B.imgkey;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

recipe_menu	direction	imgkey	dir_image
고등어 찜	1. 양조간장과 물을 넣어 끓인 후 불은 끄고 다시마를 넣어 우리고 식초, ...	1	http://www.foodsafetykorea.go.kr/uploadimg/cook/20_00167_1.png
고등어 찜	2. 미나리는 풀기 부분을 4cm길이로 잘라 식초를 넣은 물에 담근다.	NULL	NULL
고등어 찜	3. 대파, 생강, 홍고추, 레몬껍질은 채 썰어 물에 담가두고, 시금치는 뿌...	NULL	NULL
고등어 찜	4. 고등어는 세장 뜨기로 살을 발라낸 후 가시와 껍질을 제거하고 어슷...	4	http://www.foodsafetykorea.go.kr/uploadimg/cook/20_00167_4.png
고등어 찜	5. 찜통에 김이 나면 고등어, 파, 생강편을 올리고 5분 찐다.	NULL	NULL
고등어 찜	6. 팬에 기름을 두르고 다진 마늘과 시금치를 넣어 볶고, 방울토마토는 ...	6	http://www.foodsafetykorea.go.kr/uploadimg/cook/20_00167_6.png
고등어 찜	7. 파 기름을 높은 온도로 올려 쪄낸 고등어살 위에 뿌린다.	NULL	NULL
고등어 찜	8. 접시에 고등어를 놓고 그 위에 미나리 풀기를 얹은 후 다시 고등어를 ...	NULL	NULL
고등어 찜	9. 고명으로 시금치와 토마토를 올리고 소스를 끼얹은 후 그 위에 파, 생...	NULL	NULL