**기술 문서**

<알려줄게>

딥러닝 이미지 분석 모델을 활용한 쓰레기 분리수거 앱 개발

1팀

권구완(2018116224)

박찬호(2018110250)

이상협(2017110051)

지현배(2020112665)

**<Content>**

1. 스플래시 스크린

2. 홈 화면

3. 검색 화면

4. 마이페이지

5. 분리배출 가이드

6. 딥러닝 이미지 분석

7. 서버 구축

8. 딥러닝 모델

9. 데이터베이스

10. 서버와 클라이언트 간 통신

**1. Splash screen**

안드로이드12부터 Splash API가 반 강제화 되었다. 기존의 Splash Activity를 사용하게 된다면, 안드로이드 기본 Splash API와 Splash Activity가 두 번 뜨게 된다. 이는 UI/UX적으로 불편함을 준다. 기존의 Splash Activity를 만들어서 몇 초 후 intent로 MainActivity로 넘기게 구현을 하면, 앱이 준비된 이후 표시가 되는 것이기 땜문에 Splash를 사용하는 의미가 퇴색된다. 이러한 이슈를 접하고 Splash screen을 구현했다.

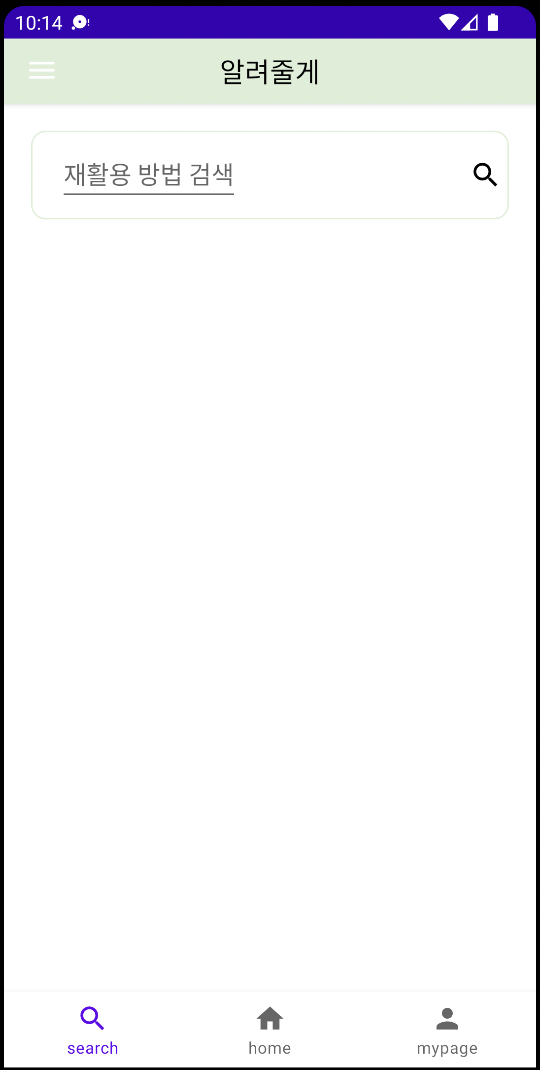
(<https://developer.android.com/develop/ui/views/launch/splash-screen>)

**2. 홈 화면**

**(MainActivity.kt, MainHomeFragment.kt, MainMypageFragment.kt, MainSearchFragment.kt, ViewPagerAdapter.kt**

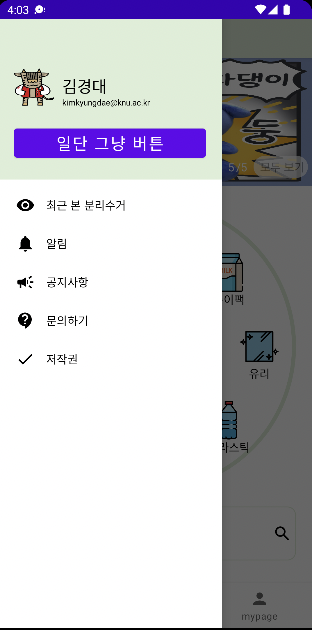
**bottom\_nav.xml, drawer\_nav.xml, hearder\_nav.xml)**

2-1. Toolbar



* Toolbar를 사용하여 임의로 수정 가능한 Toolbar를 새로 구성하여 사용하였다.

2-2. Drawer



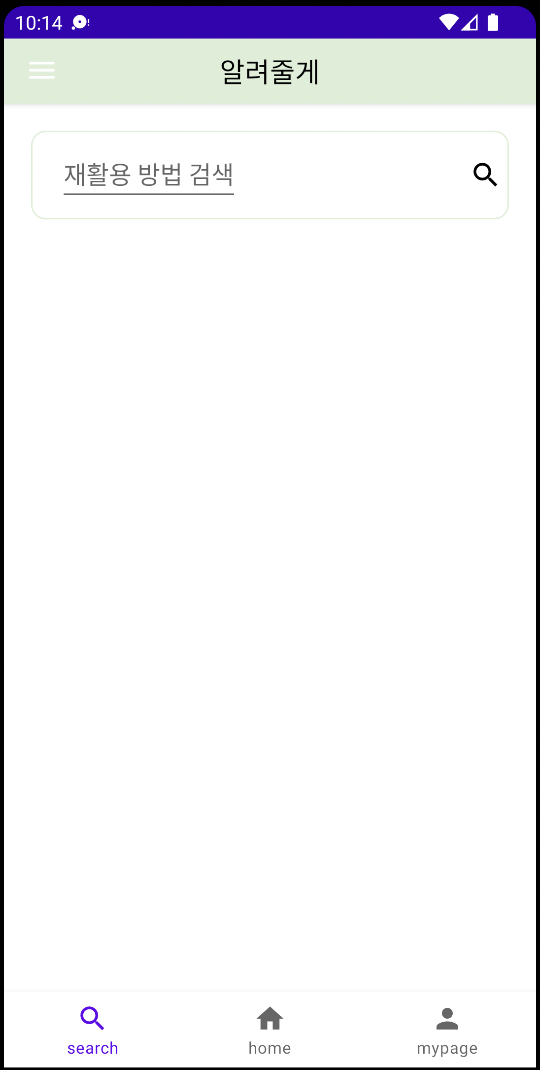
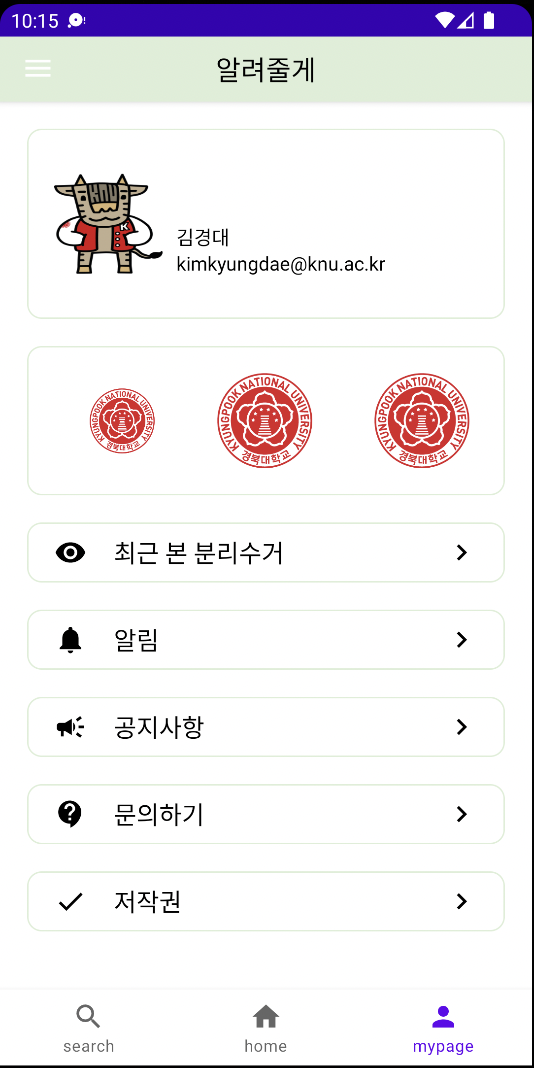
* 툴바에 드로어 토글을 적용시킨 후, 각 아이템을 클릭시 해당 설정화면으로 이동한다. 이동 후 closeDrawer()로 드로어를 닫아준다.

2-3. BottomBar



* BottomNavigationView를 사용하여 각 아이템을 클릭시 설정한 기능을 수행하도록 한다.

2-4. SupportFragmentManager



* activity\_main.xml의 중간에 FrameLayout을 사용하여 BottomNavigationView의 아이템이 선택 되었을 경우 프래그먼트의 전환으로 화면을 바꾸게 하였다
* 위 기능을 수행하기 위해서 SupportFragmentManger를 활용하였으며 그 중에 beginTransactioin().replace()를 사용하였다.

2-5 슬라이드 배너

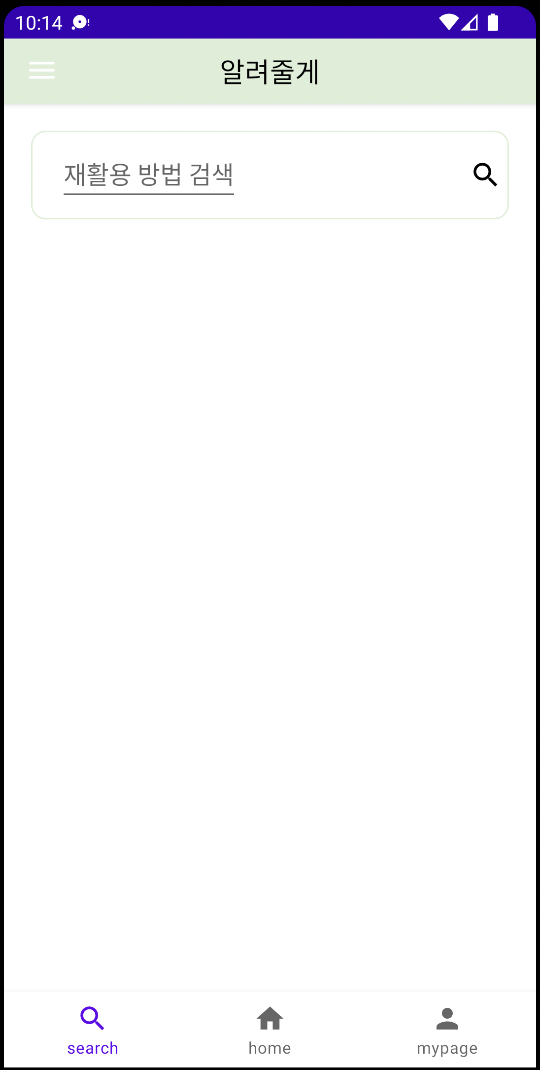
* Viewpager2로 화면을 잡아주고, RecyclerView를 상속 받기 때문에, RecyclerView Adapter를 생성할 때처럼 구현한다.
* item\_layout\_banner.xml 로 배너 화면이 들어갈 공간을 잡아준다.
* ViewPagerAdapter에서 ViewHolder를 통해 이미지를 띄워준다.
* ViewModel 에서 UI에 표시되는 데이터를 보유한다.
* 화면이 전환될 때마다 포지션을 통해서 배너 숫자를 보여준다.

**3. 검색화면**

**(MainSearchFragment.kt, fragment\_main\_search.xml)**

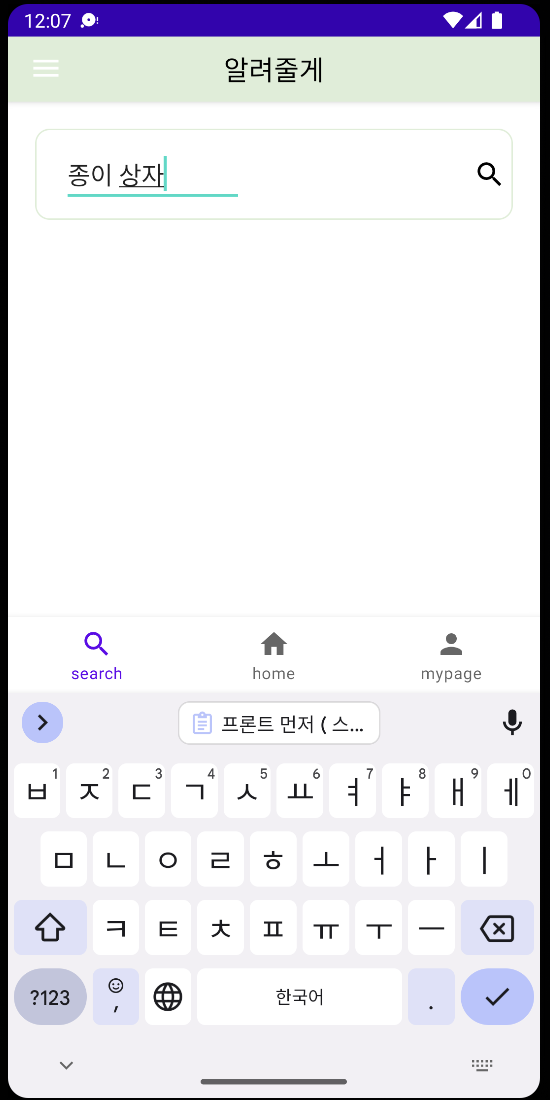
3-1. 검색 창

EditText : 사용자가 입력한 값을 변수에 전달하기 위해 사용하였다.



* android : hint =”” : 값이 입력되지 않았을 경우에 사용자에게 보여줄 문자를 정의하기 위해 사용한다. (프로젝트에서는 “재활용 방법 검색” 입력)
* android : singleLine = “” : 검색 되는 품목들의 특성 상 한 줄만 사용하기 위하여 singleLine을 사용한다.
* setOnClickLinster{}를 사용, 우측의 돋보기 이미지 클릭 시 변수에 값을 저장하도록 하였다.

3-2. 검색 실행



* fragment에서 바로 xml에 추가한 EditText의 Id를 호출하면 인식하지 못하는 경우가 발생, 이를 해결하기 위해 var view = inflater.inflate(R.layout.fragment\_main\_search, container, false)를 선언, 이후 var EditText = view.(EditText의 id)로 선언하여 호출하는 것으로 해결하였다.
* EditText.text.toString()을 사용하여 사용자가 검색한 값을 변수에 String으로 저장한다.
* 값 저장한 변수.contains()를 사용하여 변수에 설정한 품목을 포함하는 검색어가 입력되었는지 비교 후 결과에 따라 기능을 수행한다.
* 설정한 품목이 포함되어 있으면 intent를 사용하여 TabActivity(정보 설명 액티비티)로 전환한다.
* intent.putExtra로 품목에 따른 값을 전환할 때 넘겨주고 그 값에 따라 표시할 품목을 전환한다.
* 없는 품목이 검색되었으면 Toast.makeText()를 사용, 토스트 메세지를 출력하게 구현한다.

**4. 마이페이지**

**(MainMypageFragment.kt, MainMypageSetting1~6Activity, MainMypageWebview1~2)**

* webView를 통해 분리배출의 기준이되는 대구 서구 사이트를 보여준다.
* 마이프로필, 공지사항, 알림, 저작권, 제작자 등의 설정이 존재한다.
* MainMypageFragment에서 해당하는 각각의 Activity로 intent한다.

**5. 분리배출 가이드**

**(TabActivity.kt, TabFragmentPagerAdaptor.kt, Tab1~8Fragment.kt, Tab1~8Fragment.xml, activity\_tab.xml)**

5-1. 정보 열람



* TabLayout과 ViewPager2를 활용, 한 액티비티에서 다수의 프래그먼트를 사용하여 다양한 정보를 분류하여 볼 수 있도록 하였다.
* TabFragmentPagerAdaper class를 구현, ViewPager2를 사용할 수 있도록 하였다.
* TabLayoutMediator()를 사용하여 tabTitleArray에 물품의 이름을 입력 해두고 각 tab의 position에 맞게 이름이 적용되도록 하였다.
* setCurrentItem()을 사용하여 TabActiviity로 화면이 전환 될 때 전달 받은 값에 따라서 첫 화면에 설정될 tab을 결정하도록 하였다.
* ScrollVIew를 사용하여 출력되는 정보가 많을 경우 화면을 움직일 수 있도록 하였다.
* floatiingactionbutton을 사용하여 버튼 클릭 시 사진으로 정보를 검색하는 ImageActivity로 전환되도록 하였다.

**6. 딥러닝 이미지 분석 / ImageActivity, 로딩다이얼로그, 카메라, 갤러리**

**(ImageActivity.kt, BaseActivity.kt, LoadingDialog.kt, loading\_item.xml)**

* 카메라, 저장공간 권한을 받아오고, 카메라와 갤러리 요청코드를 통해 이미지를 불러온다.
* 이미지를 저장할 변수를 만들고, uri를 넘겨준다.
* 카메라나 갤러리에서 사진이 선택되면, 로딩 다이얼로그가 실행이 되고, 사진은 서버로 전송한다.
* 로딩 다이얼로그를 코루틴으로 실행하여 서버로부터 결과값을 받아오는 것을 확인하고 로딩 다이얼로그를 실행 중지 시킨다.
* 값이 올바르면 YES를 눌러서 TabActivity의 해당 항목fragment로 전환되고, 값이 틀리면 NO를 눌러서 서버로 전송한 후 오류 피드백 토스트 메시지를 띄워준다.

로딩 다이얼로그 참고 자료 <https://www.youtube.com/watch?v=VKePyfdSSoQ>

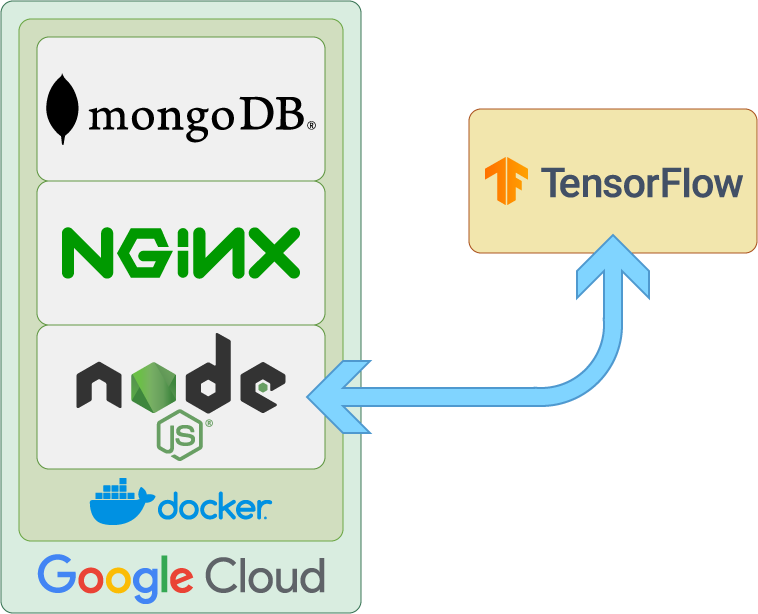
카메라 이미지 불러오기 참고 자료 <https://www.youtube.com/watch?v=OxEjqsaH_dU&list=PLVsNizTWUw7GMJ_jrWYsvIUHRq_XfjPA2&index=43>

갤러리 이미지 불러오기 [https://www.youtube.com/watch?v=-d28l3dQ0yg&list=PLVsNizTWUw7GMJ\_jrWYsvIUHRrq\_XfjPA2&index=44](https://www.youtube.com/watch?v=-d28l3dQ0yg&list=PLVsNizTWUw7GMJ_jrWYsvIUHRq_XfjPA2&index=44)

Retrofit2 참고

<https://velog.io/@cmjh951330/restrofit2-method>

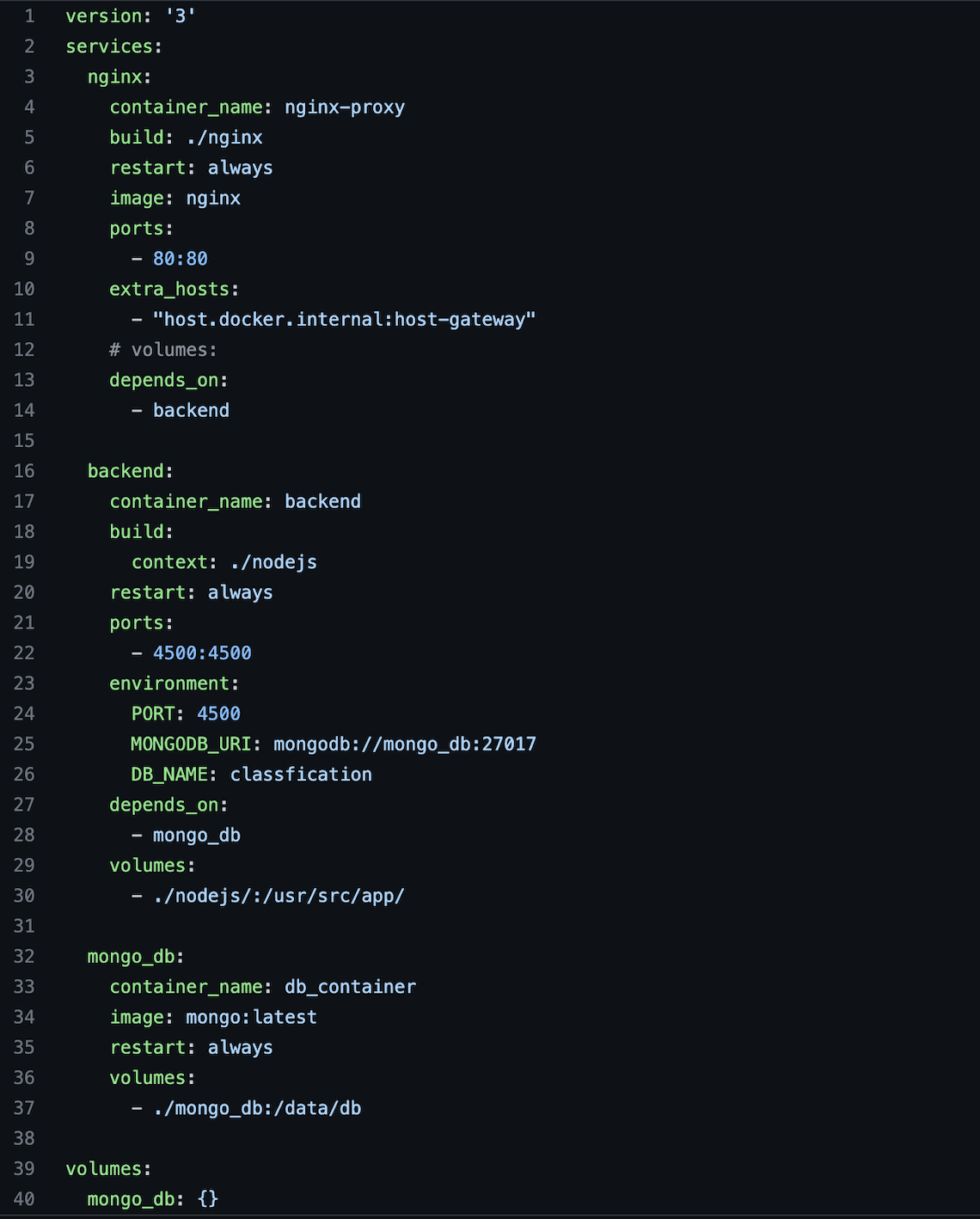
**7. 서버 구축**



<서버 구조도>

Google Cloud Platform에서 VM인스턴스를 생성하여 도커를 활용하여 서버 배포에 필요한 Nginx와 서버를 구성할 Node.js 그리고 데이터베이스를 담당할 Mongodb를 연동하여 배포하였다.

(참고사이트 - <https://wooiljeong.github.io/server/gce-vscode/>, <https://ballpen.blog/gcp-%EB%B6%80%ED%8C%85-%EB%94%94%EC%8A%A4%ED%81%AC-%EC%9A%A9%EB%9F%89/>, <https://wotres.tistory.com/entry/docker-image-%EC%82%AD%EC%A0%9C-%EA%B0%95%EC%A0%9C-%EC%82%AD%EC%A0%9C-%ED%95%98%EB%8A%94-%EB%B2%95>)



<docker-compose.yml>

위의 사진을 통해 보면 mongodb를 먼저 실행해서 backend의 nodejs 실행 시 에러 발생을 방지할 수 있도록 하였다. 또한 nginx도 backend 컨테이너가 실행되고 생성되도록 설정하여 이 역시 에러 발생을 방지하도록 하였다.

(참고사이트 - <https://jojoldu.tistory.com/587>)

1. Mongodb

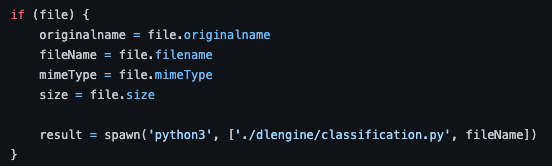
도커로 생성한 mongodb는 저장된 데이터를 도커로 만든 이미지를 제거하게 되면 같이 사라지게 되어있다. 그렇기에 위에서 volumes 설정을 하여 실제 vm인스턴스의 volume과 링크를 연결하여 데이터가 도커이미지 바깥에서도 저장되도록 하였다.

1. nodejs

nodejs도 마찬가지로 사용자에게서 받은 이미지들을 저장하기위해 volumes 설정을 해주었고, child\_process 모듈의 spawn 함수를 사용하여 tensorflow 모듈을 사용할 python 파일을 연동 실행하도록 하였다. 다음 사진은 이에 대한 보충 자료이다.



<spawn 사용 이미지>

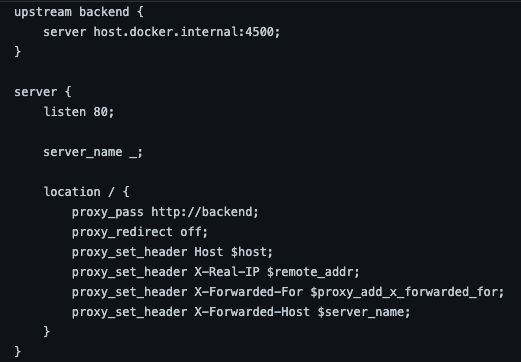


<python 파일 연동 실행하는 이미지>

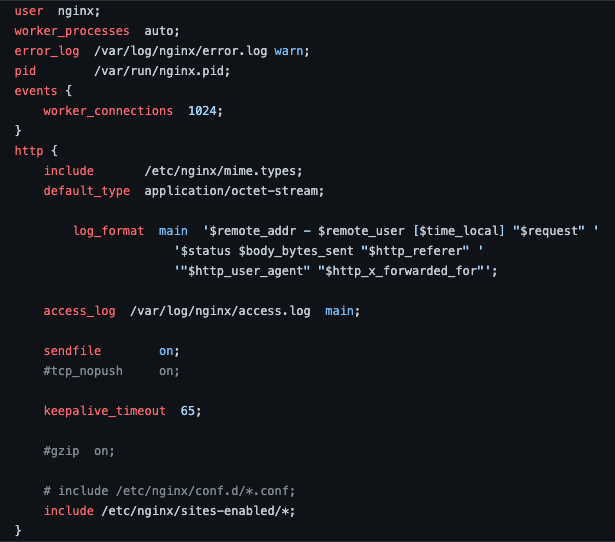


<python의 출력문을 가져와 필요 부분만 분류하는 이미지>

1. nginx

빌드된 nodejs를 외부에서도 접근할 수 있도록 nginx를 통해서 배포를 하였다.

<sites-available의 back.conf>

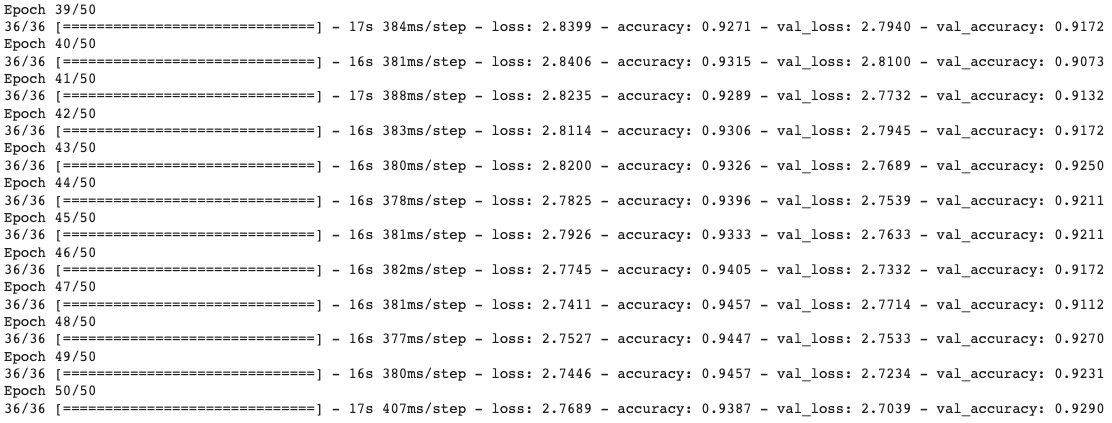


<nginx.conf>

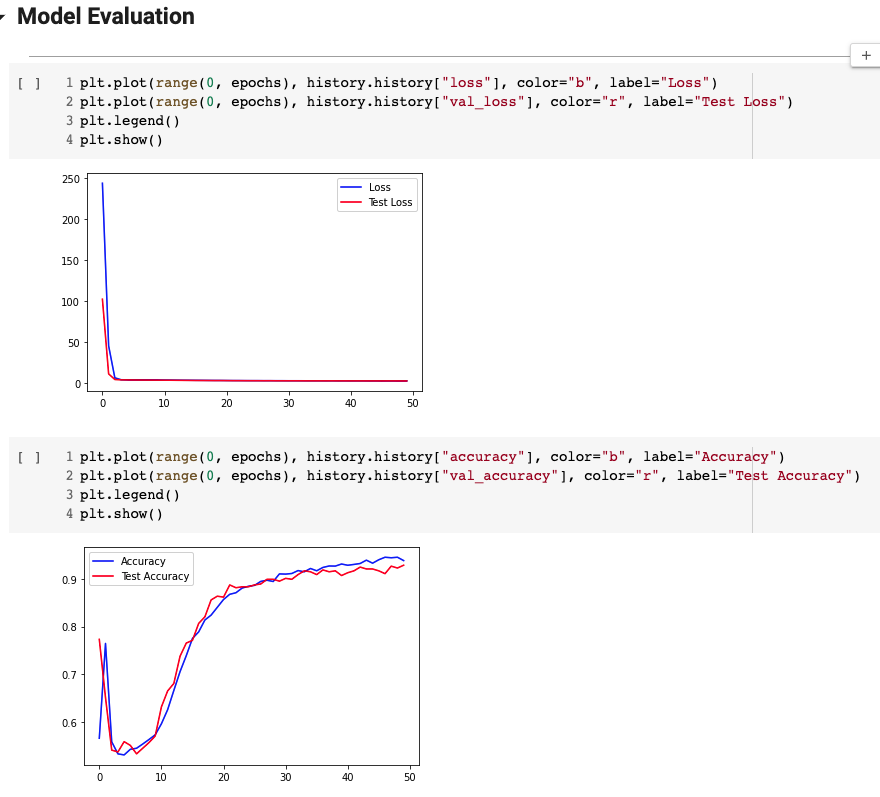
**10. 딥러닝 모델**

딥러닝 기술을 사용하려 했지만 팀원들 중에 딥러닝에 대해 아는사람이 없었기 때문에 깃허브에서 탐색하여서 사용하였음.

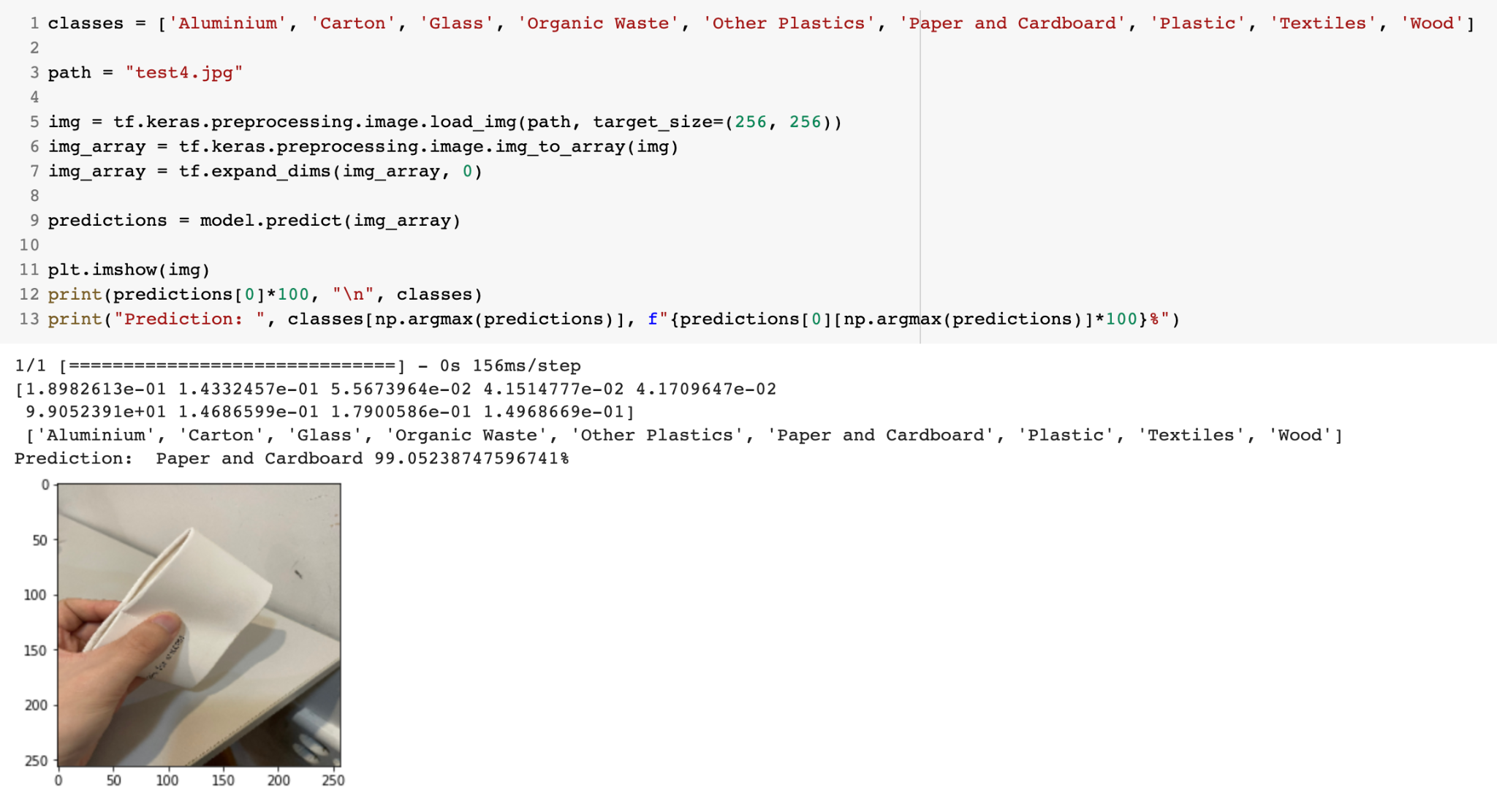
참고 링크(<https://github.com/cardstdani/WasteClassificationNeuralNetwork>)



<모델 학습 중>



<모델 평가표>



<결과>

데이터의 수 부족과 학습 알고리즘을 제대로 이해하지 못한 채로 진행했기에 결과값이 생각했던 것보다는 만족스러운 결과값이 많이 나오지는 않았다. ( 플라스틱이 유리로 나오거나 유리가 플라스틱으로 나오는 경우, 이미지 질감이 종이와 비슷하면 무조건 종이로 나오는 경우도 있음)

추 후에 시간이 된다면 이미지를 직접 웹크롤링으로 수집하여 학습 시켜볼 예정이다.

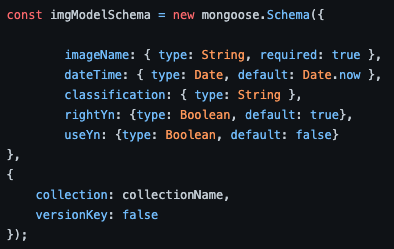
**9. 데이터베이스**

사용자에게서 받은 이미지 파일을 저장하고 활용하기 위해서 데이터베이스를 사용하였다.

nodejs와 연동하기 위해 nodejs에서 Mongoose 모듈을 사용하여 연결하였다.



<Mongodb 연동 (in nodejs) initDB.js>



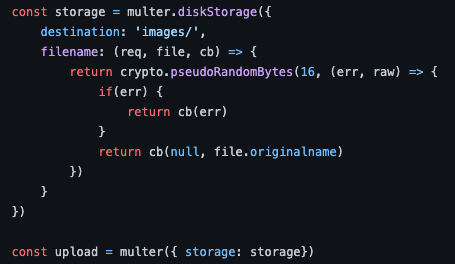
<이미지 파일 관련 데이터베이스 column>

**10. 서버와 클라이언트 간 통신**

1. 이미지 분류 관련 api

사용자가 이미지를 전송하면 서버는 nodejs에서 multer 모듈을 통해 비트맵 데이터를 받아 파일로 변환 후 저장을 하게 된다. 저장이 성공하면 python 파일을 연동 실행 하여 저장했던 이미지 파일을 분류하기 시작한다. 분류가 완료되면 결과값을 사용자에게 재전송 하게 된다. (이미지 분류 관련 사용자 시점 - 7번)

(참고 사이트 - <https://curryyou.tistory.com/225>)



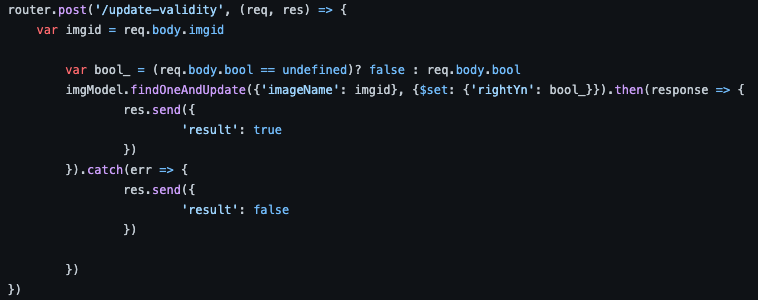
<multer를 사용하여 파일 저장하는 함수>



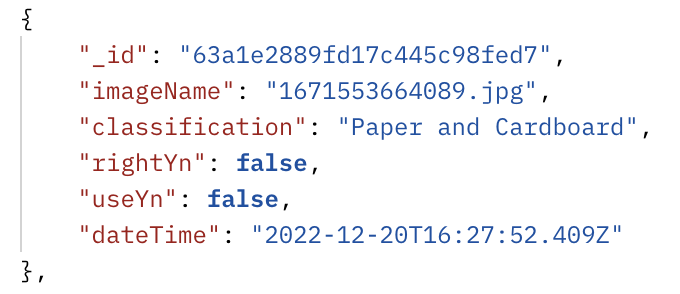
<이미지 분류 관련 api>

1. 분류 실패 이미지 관련 api

이미지를 분류하여 나온 결과를 사용자에게 보냈는데, 결과값이 예상했던 것과 다를 경우 사용자는 서버에 피드백을 보낼 수 있도록 하였다. 서버는 사용자에게서 분류가 잘못된 이미지 파일의 이름을 받아서 해당 이미지 파일의 데이터베이스를 변경하여 준다.



<분류 실패 이미지 관련 api>



<분류 실패 이미지 api를 받았을 경우 rightYn이 false로 바뀜>