소프트웨어프로젝트1

Project 2

학부: 컴퓨터정보공학부

교수님 : 이 우 신 교수님

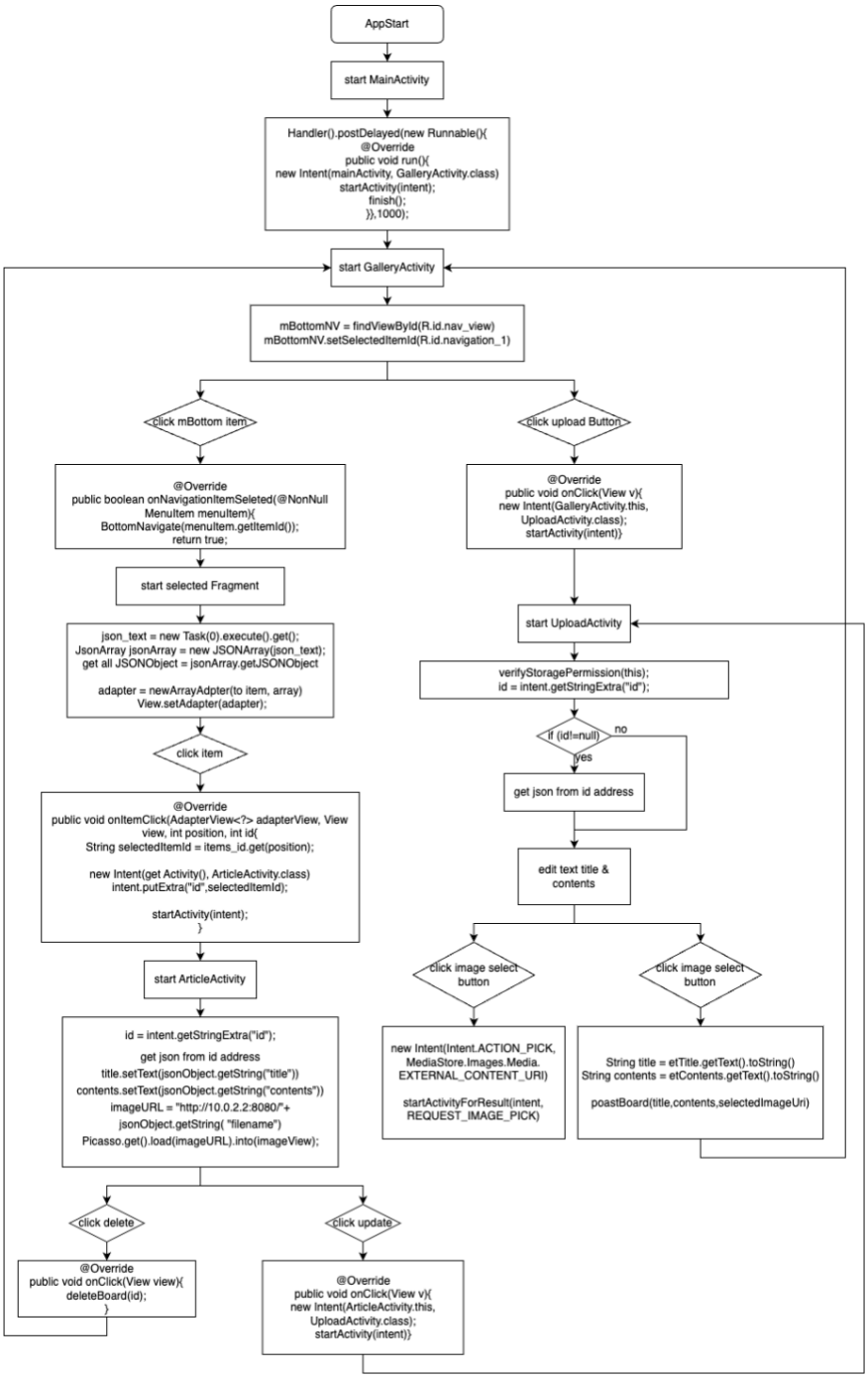
제출일: 2023.6.20

학번: 2019202021

이름: 정성엽

1. Introduction
   A. 과제에 대한 전반적인 요약

      사진을 업로드 하고 해당 사진에 대한 간단한 소개를 작성하는 어플을 구현하는 프로젝트이다. 이전 Spring boot에서 browser의 html을 통해 사진과 제목, 글을 작성하고 업로드 했던 것을 어플리케이션에서 진행하도록 하는 것이다. 작성 후에는 사진 또는 제목을 각 프레그먼트에서 나열할 수 있도록 한다. 또한 database의 수정과 삭제 모두 가능하도록 구현한다. 또한 데이터를 받을 때는 json으로 받아서 통신할 수 있도록 한다.
2. Flow Chart
   A. 과제 코드의 전반적인 흐름 설명



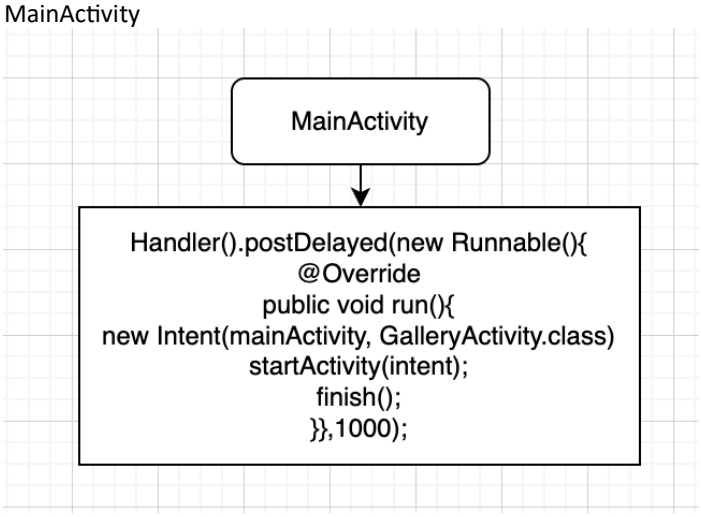
이번 프로젝트의 전체적인 코드 흐름도는 위와 같다. 어플리케이션을 시작하면 splashview로 1초 보여준 뒤 GalleryActivity로 이동한다. GalleryActivity에서는 BottomNavigationView를 띄우고 처음화면에 대해 클릭을 했다는 동작을 진행하여 해당 Fragment를 불러온다. 다른 버튼 클릭 시 해당 id의 fragment를 불러온다. 해당 fragment를 불러왔다면 ImageFragment는 서버로부터 json 이미지 주소를 받아서 이미지를 아이템 ImageView에 적용하고 ListFragment는 서버로부터 json title을 받아서 아이템의 제목에 적용한다. 만약 GridView나 ListView의 아이템을 클릭하면 해당 아이템의 id를 저장하고 Article Activity로 이동할 때 putExtra로 전달한다.

이전 GalleryActivity에서 upload 버튼을 누른 경우 UploadActivity로 이동한다. Upload Activity는 제목, 글, 사진을 받을 수 있도록 한다. 이 때 사진을 받아서 보내기 위해 사진에 대한 접근 권한 허용을 요청한다. 이미 권한이 존재한다면 생략하고 없다면 요청한다. 그리고 이전에 만약 "id"에 대해 putExtra를 받은 경우 수정하는 코드로 넘어가고 아니라면 업로드하는 코드로 진행한다. id가 null이 아닌 경우 해당 id의 json을 받아서 이미 존재하는 Title, contents, image를 적용시킨다. image select button을 누른 경우 MediaStore.Images.Media.EXTERNAL_CONTENT_URI로 부터 사진을 선택할 수 있도록 하고 후에 onActivityResult를 통해 사진을 URI로 저장한다. Upload 버튼을 누르면 Title과 contents text를 저장하고 해당 text와 imageURI를 가지고 postBoard를 할 수 있도록 한다.

이전 GalleryActivity의 각 Fragment에서 item을 클릭하면 해당 item의 id를 받아서 ArticleActivity에 전달하고 이동한다. ArticleActivity의 삭제 버튼을 누르면 해당 id 주소로 post reqeust를 보내고 수정 버튼을 누르면 id를 putExtra로 두어서 UploadActivity로 이동한다.

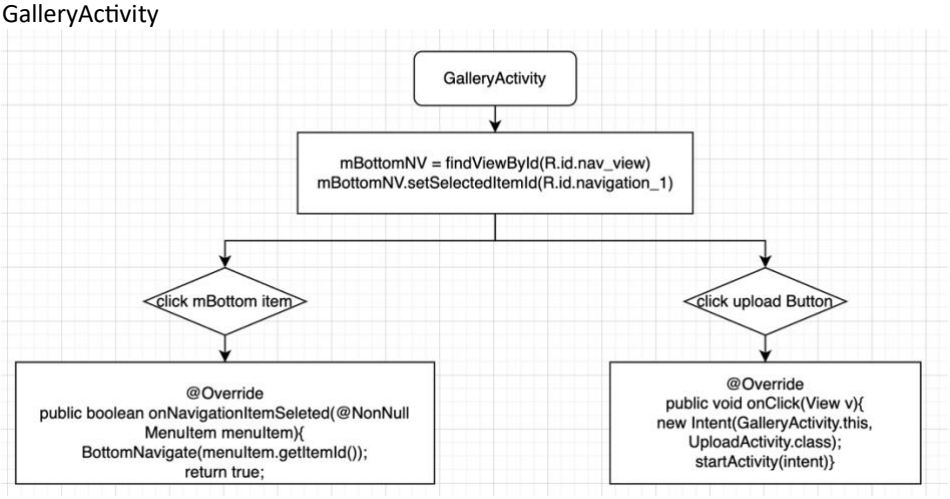
B. 각 View 마다 흐름도 설명

i. MainActivity



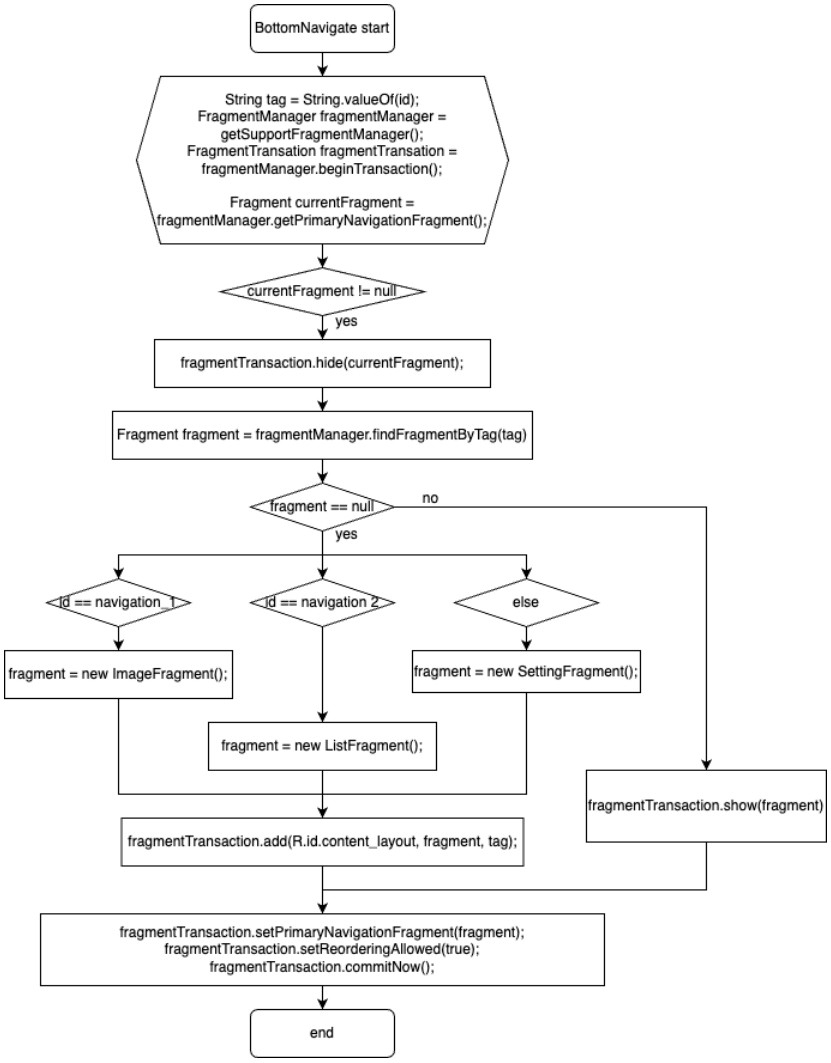
어플리케이션을 시작하면 splashview로 1초 보여준 뒤 GalleryActivity로 이동한다.

ii. GalleryActivity



GalleryActivity에서는 BottomNavigationView를 띄우고 처음화면에 대해 클릭을 했다는 동작을 진행하여 해당 Fragment를 불러온다. 다른 버튼 클릭 시 해당 id의 fragment를 불러온다. 만약 upload 버튼을 클릭하면 UploadActivity로 이동하도록 한다.

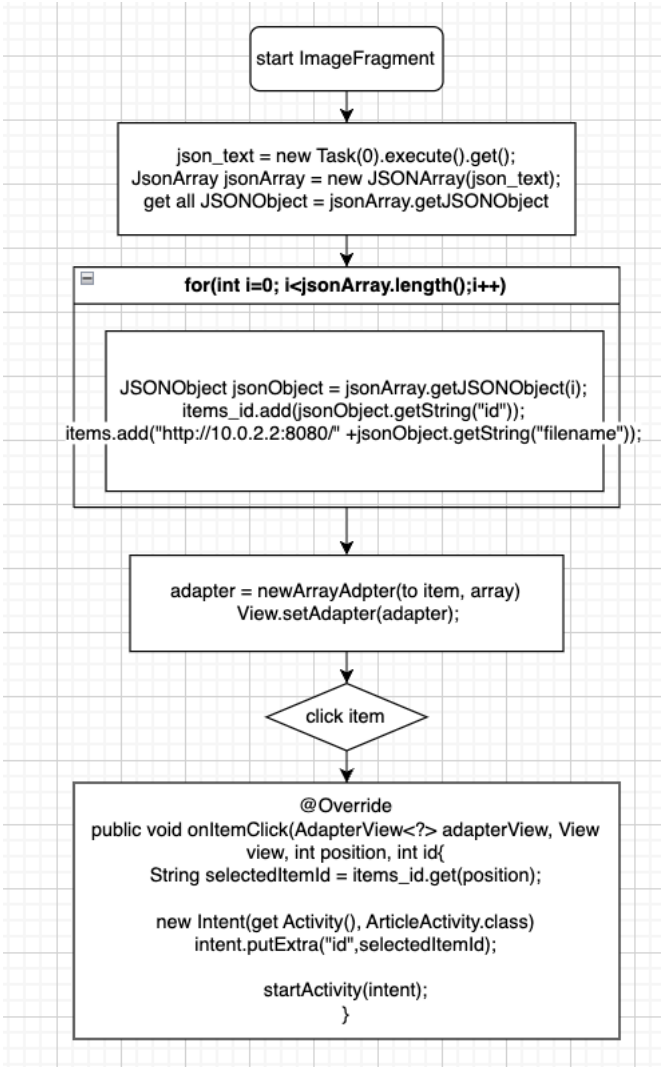
이 때 사용한 BottomNavigate 함수는 아래와 같다.



해당 함수는 BottomNavigationView의 item을 클릭하였을 때 해당 item의 Id에 따

라 다른 Fragment가 보이도록 하는 함수이다. FragmentManager와 FragmentTransation을 이용하여 해당 작업을 진행한다. 이미 한번 열은 적이 잇다면 Transaction을 통해 해당 fragment를 다시 불러온다.

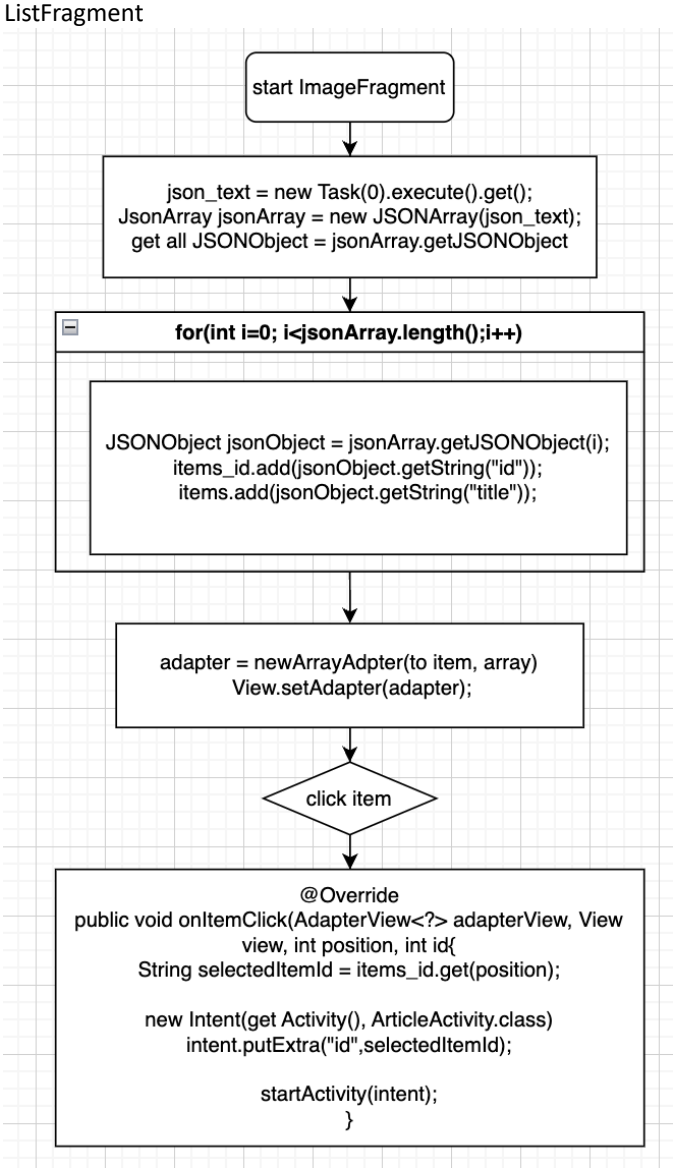
iii. ImageFragment



ImageFragment가 선택되었을 때 작업이다. json_text를 Task class를 이용하여 모든 data의 json을 받고 이를 JSONArray로 변환한다. 이 array의 길이만큼 반복하여 json을 분리하여 각 json의 id와 filename을 받는다. 이때 filename 앞에 baseURL을 추가하여 후에 해당 주소로 접속했을 때 이미지를 바로 받아올 수 있도록 설정한다. 각 id와 filename은 array로 add하고 후에 Adapter에서 itmes에 대해 gridview의 ImageView를 바꿀 수 있도록 한다.

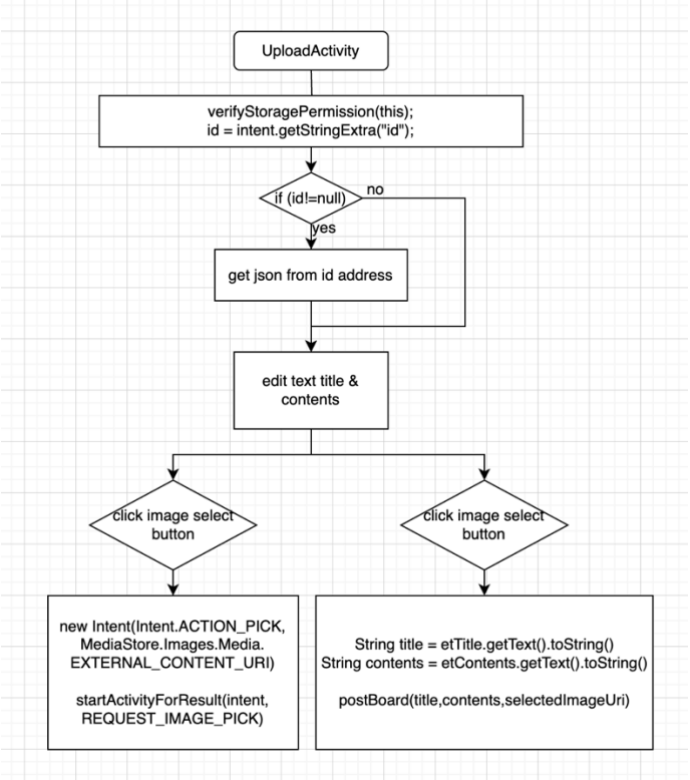
만약 gridView의 item을 클릭했다면 해당 item에 해당하는 id를 가지고 ArticleActivity로 이동하도록 한다. 이 때 putExtra를 통해 전달한다.

iv. ListFragment



위의 ImageFragment와 거의 동일하면 다른 점은 items.add에 “filename”이 아닌 “title”을 가져오고 adapter를 통해 listview의 textview에 해당 title을 각 item에 적용한다.

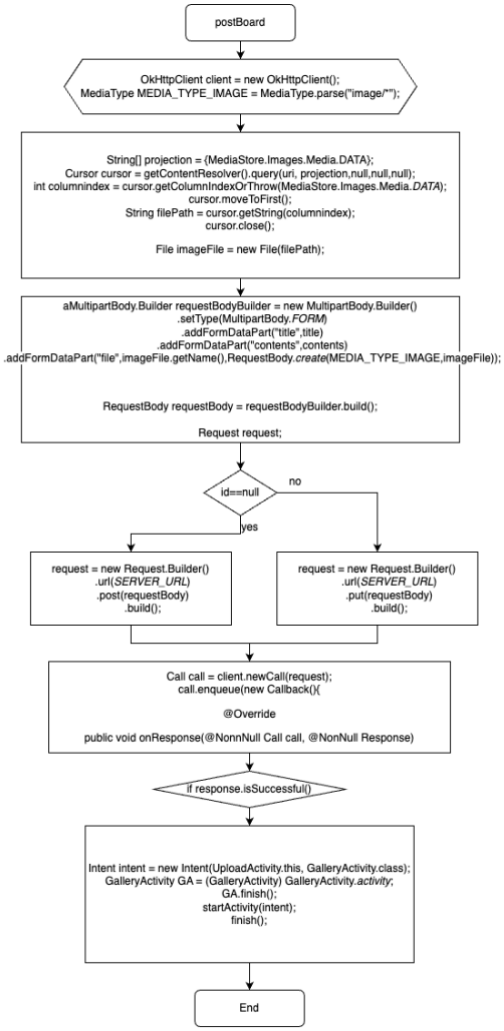
v. UploadActivity



UploadActivity를 처음 실행할 때 verifyStoragePermission(this)를 실행하여 외부저장소의 이미지에 대한 접근 권한이 있는지 확인하고 없는 경우 권한을 요청하는 팝업메세지를 보내도록 한다. 또한 이전 intent에서 putExtra로 보낸 id를 getStringExtra를 통해 받는다.

이 UploadActivity는 업로드와 수정을 공용으로 사용하기 때문에 id가 null인 경우에는 업로드, null이 아닌 경우에는 수정을 하도록 분리한다. null이 아닐 때는 해당 id에 대한 json 데이터를 받아서 Title, Contents, image를 넣어준다. 이후 사용자는 title과 contents를 편집하도록 한다. 만약 이미지 선택 버튼을 누른다면 MediaStore.Images.Media.EXTERNAL_CONTENT_URI로 이동하여 이에 대한 결과를 받아올 수 있도록 한다.

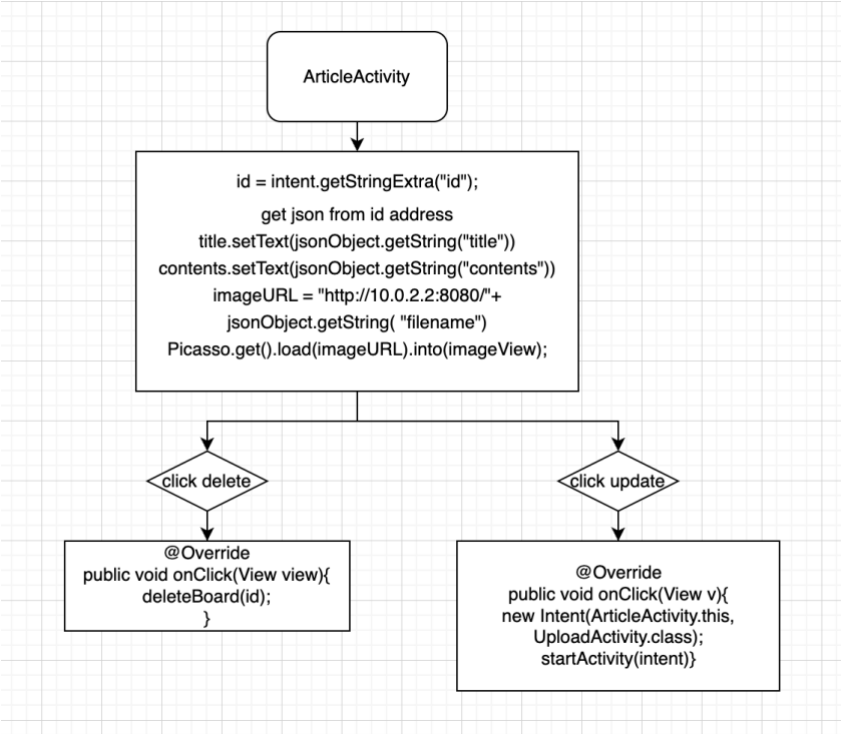
마지막으로 업로드 버튼을 누른다면 현재 editText의 title과 contents를 저장하고 이전에 저장한 이미지에 대한 URI를 postBoard 함수를 통해 upload를 진행한다. 해당 함수는 아래서 설명한다.



OkHttpClient로 client를 설정하고 mediatype도 설정한다. 다음 과정은 흐름 중 2부분으로 실행 흐름을 나누었다. data에 대한 string 배열 projection을 두고 uri에 대한 cursor를 받는다. mediastore의 처음부분으로 이동하고 해당 index의 filepath를 받고 cursor를 닫는다. 받은 filepath는 new File(filePath)를 통해 이미지를 받는 용도로 사용한다. MultipartBody 빌더를 통해 FORM 형식으로 settype하고 title과 contents 그리고 file에 대한 FORMdatapart를 추가한다.

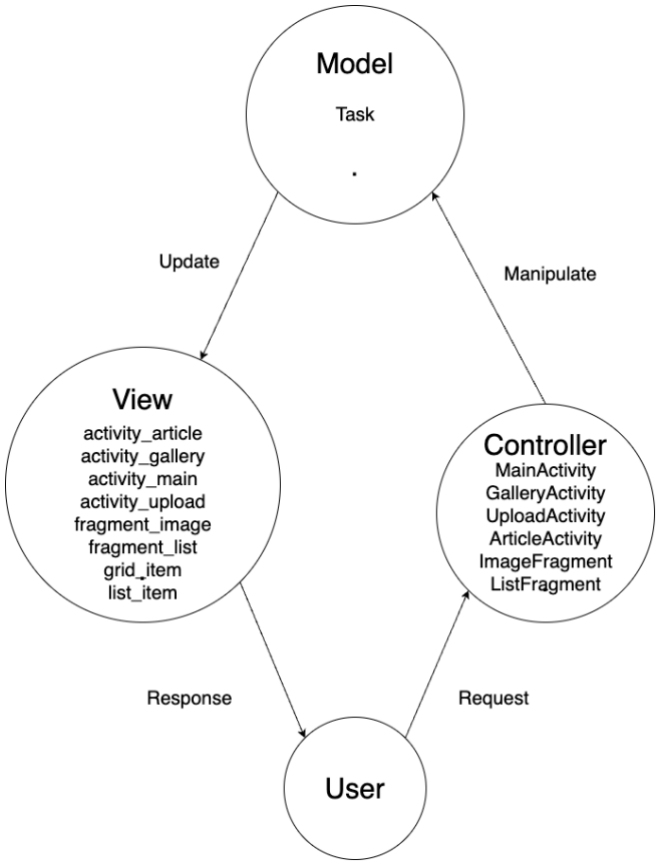
이후 ReqeustBody를 빌드하고 id==null이면 post를 http://10.0.2.2:8080/board에 request를 만들고 id!=null이면 put을 http://10.0.2.2:8080/board/{id}에 request를 만든다. call.enqueue로 request를 보내고 성공한다면 GalleryActivity를 미리 종료하고 다시 intent를 통해 Activity를 start한다. 이후 현재 activity를 종료한다.

vi. ArticleActivity



ArticleActivity를 시작할 때 이전 activity에서 보낸 Extra를 받아서 해당 id로 json 데이터를 받아올 수 있도록 한다. json을 받았으면 title과 contents에 대한 값을 settext하고 filename은 Picasso를 통해서 imageView의 이미지를 설정한다. delete 버튼을 클릭한다면 deleteBoard 함수로 이동하는데 이는 이전에 설명한 postBoard에서 multipart가 아닌 일단 FormBody로 빌드해서 10.0.2.2:8080/ImageView/{id}에 Post request를 보내도록 한다.
업데이트 버튼을 누른다면 현재 id를 putextra로 두고 uploadActivity로 이동하도록 한다.

3. Result
   A. 프로젝트 내에 MVC Pattern 설명



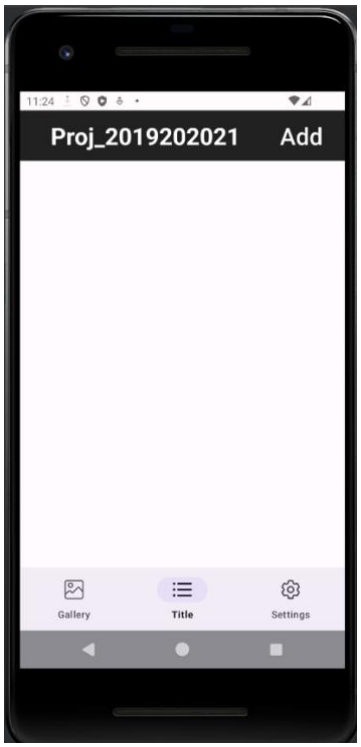
이번 프로젝트의 안드로이드 MVC 패턴을 그림으로 설명하면 위와 같다. 먼저 Model은 Task로 이는 json 데이터를 받아오는 역할을 한다. 사진과 제목, 글을 업로드 하는 Model은 UploadActivity에 포함되어 있어서 Controller에 포함하였다.
View는 xml 파일들이 해당하며 GalleryActivity안에 fragment가 동작하기 때문에 Controller 개수보다 많다. 또한 gridview와 listview의 item에 대한 View도 포함하고 있다. 이런 xml 파일들은 어플리케이션에서 사용자에게 데이터를 시각적으로 보여주는 역할을 한다.
Controller는 Activity들이 해당하며 이 Activity는 사용자의 요청을 받고 처리하는 역할을 한다. 여기서 추가적으로 UploadActivity는 Model에도 해당하며 데이터베이스에 Post와 Put Request를 보낼 수 있다. 또한 ArticleActivity에서도 delete를 위해 Post request를 보낼 수 있다.

추가적으로 Spring에 InfoController가 추가되었는데 이는 Controller에 해당하며 이 InfoController의 역할은 database에 있는 정보를 json으로 변환하여 반환해주는 역할을 한다. 그리고 InfoDTO도 추가되었으며 InfoController가 정보를 가져와 처리할 때 사용되면 이는 Model에 해당한다.

   B. 각 View(html) 별 구현 방식 설명 및 결과 화면 설명
      i. SplashView

1초 뒤 사라지는 splashview로 Handler().postDelayed를 사용하여 1초 타이머 뒤에 다른 Activity로 intent하도록 설정하였다. 해당 View에는 ImageView만 사용하였다.

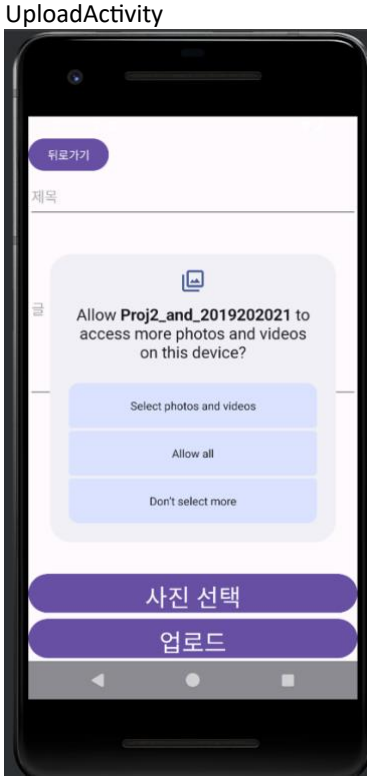
ii. GalleryActivity

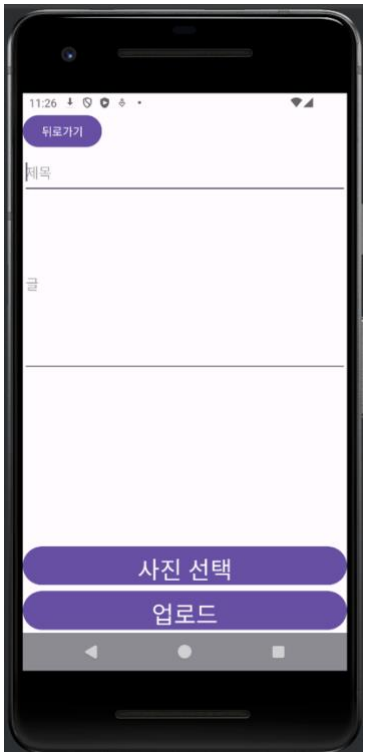


아직 아무것도 업로드 되지 않았을 때의 페이지이다. 아무 데이터가 없기 때문에 비어있는 모습이다. 해당 View에서는 BottomNavigationView를 사용하였으며

요소로 3가지를 넣었다. 첫 번째는 Gallery, 두 번째는 Title, 세 번째는 Setting이다. 아직 아무 요소가 없기 때문에 각 요소에 해당하는 fragment는 업로드 이후에 설명한다.

iii. UploadActivity





UploadActivity으로 이동했을 때 view이다. 이미지에 대한 접근 권한 허용을 묻는 팝업이 나타나고 이에 대해 Allow All을 하면 제안서처럼 뒤로가기, 제목, 글, 사진선택, 업로드에 대한 버튼 및 edittext 박스가 있다.

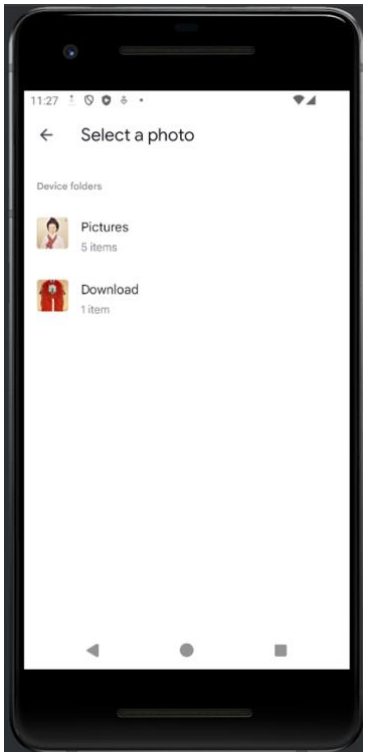
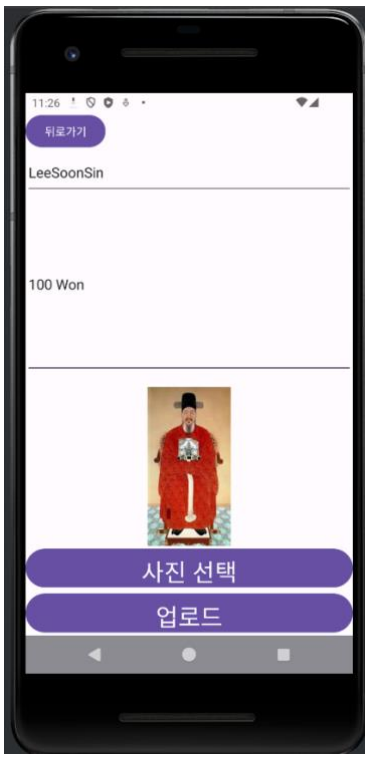
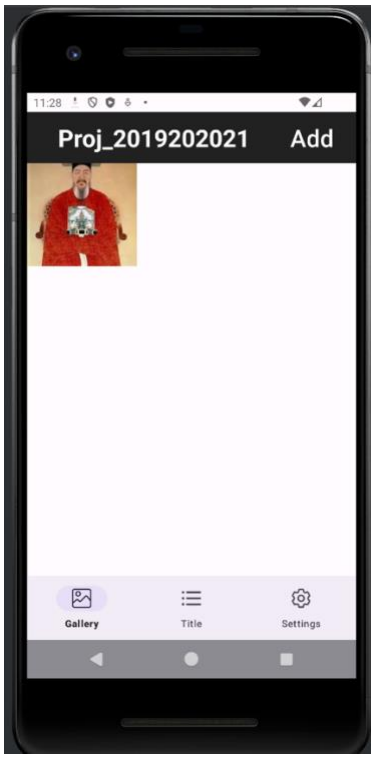


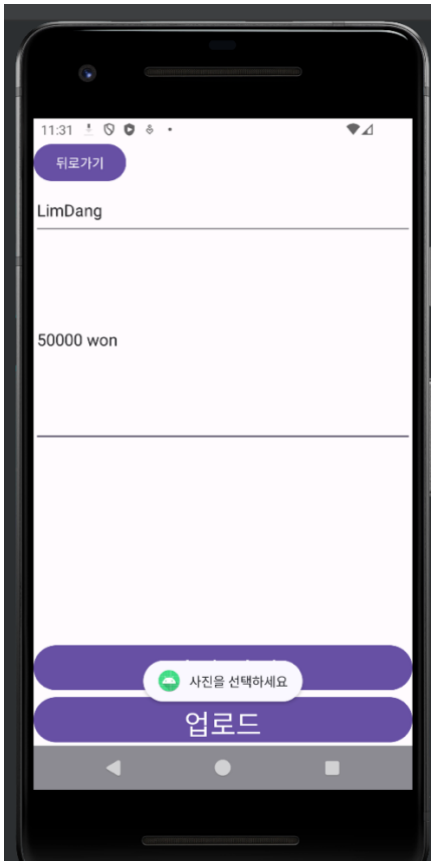
사진 선택버튼을 눌렀을 때 사진을 선택할 수 있는 view로 인텐트한다. 이것은

MediaStore.Images.Media.EXTERNAL_CONTENT_URI에 해당한다. 이에 대한 settype은 image로 한정하고 startActivityForResult로 이동하고 onActivityResult로 결과를 받아온다.





위 사진처럼 제목과 글을 입력하고 사진도 선택한 뒤 업로드를 하면 오른쪽 사진과 같이 업로드가 되어 Gallery에 뜨는 것을 확인할 수 있다. 이전에 localhost:8080/board에 post request와 함께 form을 보내면 해당 form의 내용과 file이라는 multipartfile을 읽어 boardService의 createBoard로 전달해 데이터를 생성하였다. 이를 활용하여 MultipartBody.Builder를 통해 타입을 FORM, 데이터를 addFormDataPart로 “title”, “contents”, “file”에 해당하는 내용을 추가하며 file 같은 경우 imagefile 경로를 가져온 file에 대해 이름과 함께 파일을 담는다. 이에 대해 requstBody로 build를 하고 request를 통해 주소와 보낼 HTTP 통신 타입과 requestbody를 묶어서 build한다. 이제 client request에 대한 call을 만들고 enqueue 명령어를 통해 call을 보낸다. 만약 response가 성공한 경우 이전에 열린 GalleryActivity를 종료하고 새로 intent 후 현재 activity를 종료한다. 실패한다면 그냥 뒤로가고 현재 인텐트를 종료한다. 또한 제안서에서는 이미지 선택 시 선택한 이미지를 띄워야한다는 조건이 없지만 사진을 선택했는지 안 했는지 헷갈리기 때문에 이미지를 띄워서 알린다.



데이터를 입력하는 단계에서 사진을 선택하지 않고 업로드를 하는 경우 어플리케이션이 강제 종료되는 경우가 발생하였다. 이를 방지하고자 사진이 선택되지 않은 경우 Toast message로 사진을 선택하라는 메세지가 나타나도록 제작하였다.

iv. GalleryActivity-ImageFragment & ListFragment

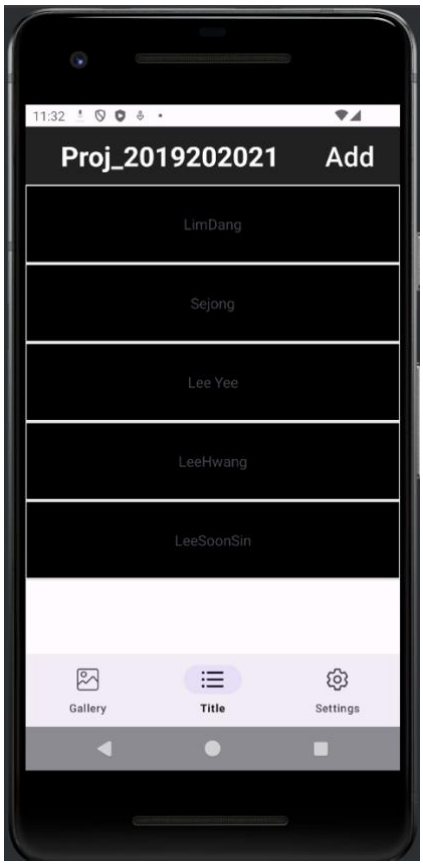
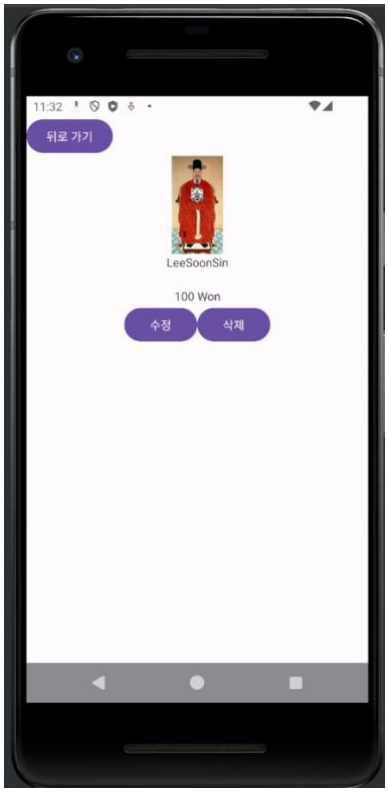


사진 5개를 넣었을 때, GalleryActivity의 framelayout에 ImageFragment와 ListFragment의 결과이다. 이미지 사이즈는 모두 다르기 때문에 갤럭시의 갤러리처럼 사진을 중앙에 맞춰 Crop하도록 이미지의 scaletype을 centerCrop하게 만든다. 세로로 긴 사진은 중앙에 초점이 맞춰짐을 확인할 수 있다. 또한 최신순으로 정렬한다. 이 때 json은 각 json을 하나하나 받아오는 것이 아닌 모든 json을 반환해주는 10.0.2.2:8080/Image에 접속하여 값을 받는다. json 값 중 Gallery는 id와 filename의 key에 대한 정보만 Array에 저장하고 Title은 id와 Title의 key에 대한 정보만 Array에 저장한다. 받은 Array는 각 Adapter를 통해서 이미지 또는 Txt를 각 item에 적용한다. 이 때 Adapter는 각 item의 높이가 Parent 높이의 1/4 그리고 1/6이도록 높이를 수정한다.

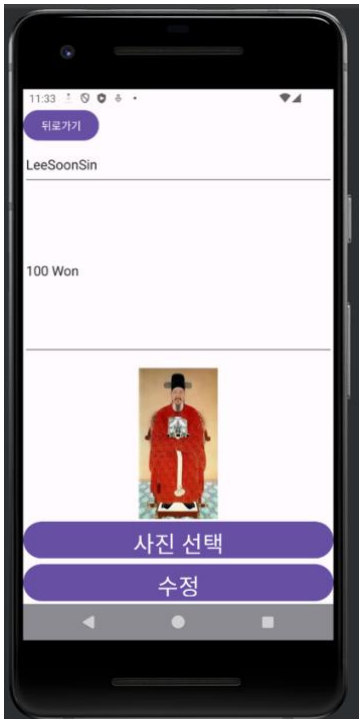
추가적으로 json을 받는 class는 Task로 기존 Task는 json을 하나하나 받을 수 있게 되어있다면 ImageFragment와 ListFragment에서는 id를 0으로 하는 경우 Spring에서 모든 json을 반환해주는 주소로 접속하고 해당 json을 받으면 Array로 변환한다. 그리고 Array 길이만큼 id Array에 json key "id"의 값을 받아서 저장하도록 진행하였다.

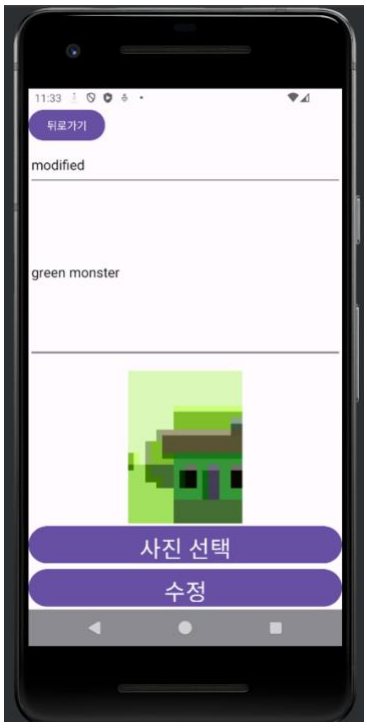
v. ArticleActivity



이순신 사진을 눌러 ArticleActivity로 이동했을 때 결과이다. 뒤로가기 사진, 제목, 글 모두 나타나고 있으며 수정과 삭제에 대한 버튼도 추가되어있다. 이전에 intent 할 때 id 값을 넘길 수 있도록 putExtra를 두었다. 해당 activity에서는 getStringExtra로 받을 수 있게 하였다. 받은 id 값으로 Task class를 활용하여 json을 받아서 각 View에 값을 추가하고 해당 이미지는 Picasso를 통해 ImageView에 사진을 넣는다.

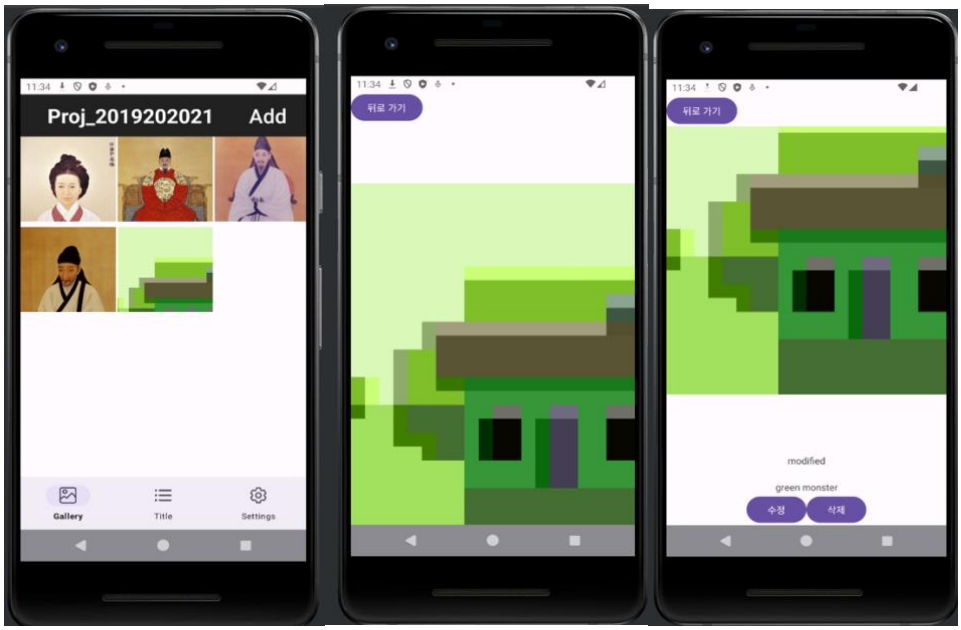
vi. UploadActivity-Update database





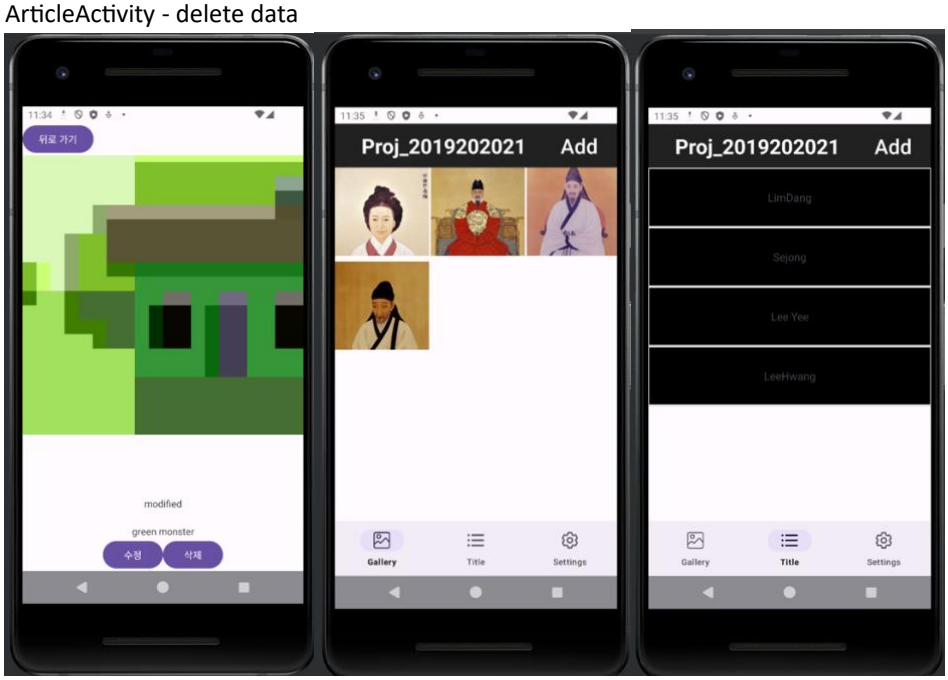
수정 버튼을 눌렀을 때 upload의 Activity를 공유하되 수정버튼을 누를 때

putExtra로 id를 넘기기 때문에 기존 정보를 그대로 textview에 입력하고 사진도 넣는다. 대신 사진은 선택이 되지 않은 상태이다. 오른쪽 사진처럼 수정을 하고 수정버튼을 누르면 데이터가 수정이 된다. 이 때는 upload 때는 post request를 했다면 update는 put request를 보내고 url은 http://10.0.2.2:8080/board가 아니라 http://10.0.2.2:8080/board/{id}로 put request를 보낸다. 다른 부분은 upload 때와 동일하다.



수정한 이후의 결과는 위 사진과 같다. 해당 사진으로 Gallery에서 변경된 것을 확인 할 수 있고 ArticleActivity로 이동하여도 수정된 사진과 내용이 나타나지는 것을 확인할 수 있다. 또한 ArticleActivity가 Scrollable 한 것을 위의 사진을 보고 확인할 수 있다.

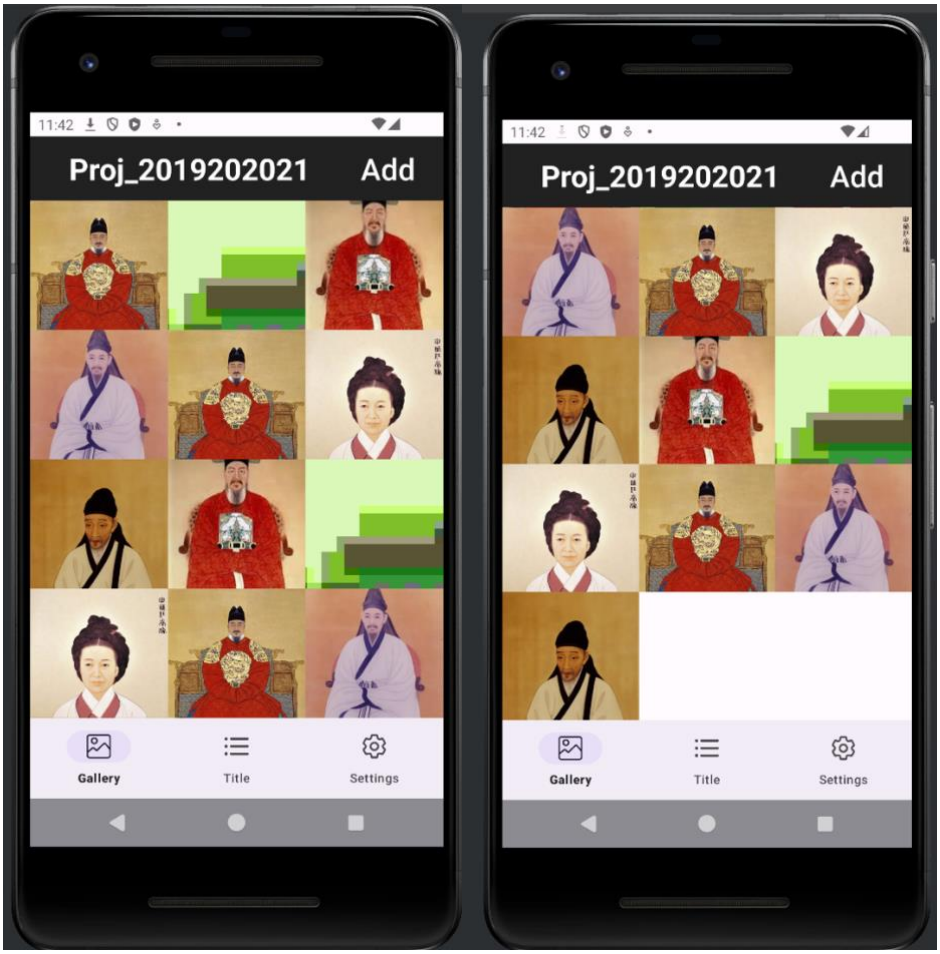
vii. ArticleActivity - delete data



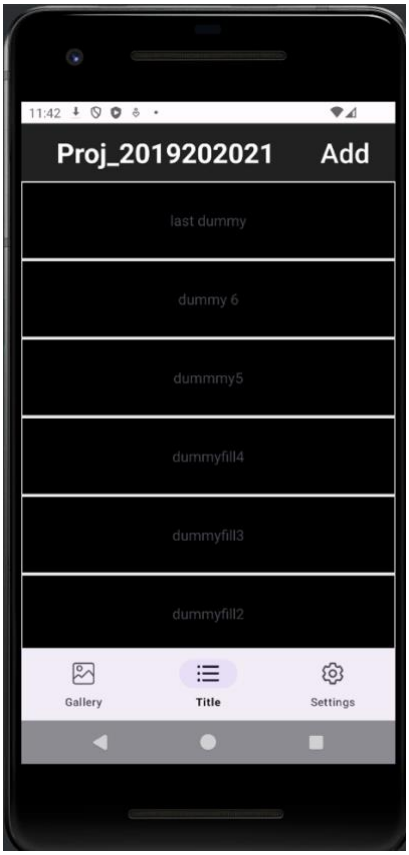
해당 사진에 보이는 삭제 버튼을 누르면 해당 데이터가 삭제된다. 이 때 사용

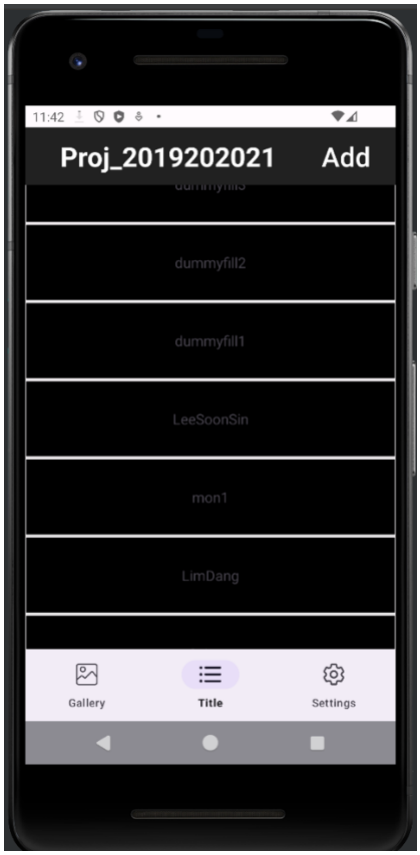
하는 방법은 html에서 해당 data를 가져오는 방법이 localhost:8080/ImageView/{id}에 Get을 하는 것이었다면 삭제하는 것은 해당 id에 Post를 하면 삭제가 된다. 이 때 reaquest body는 비어있어도 상관없다. 해당 post request를 보내는 방법은 upload 때와 동일하다. 삭제한 후에는 Gallery와 Title에서 모두 삭제가 된 것을 확인 할 수 있다.

viii. GalleryActivity - 크기 비율 확인



임의로 사진을 여러번 업로드 하였을 때 결과이다. 한 화면에 가로 3 * 세로 4 비율을 가지고 있음을 확인할 수 있고, Scrollable한 것을 확인할 수 있다. 이 때 gridview는 처음부터 scrollable 하므로 scrollview를 따로 넣을 필요가 없다.

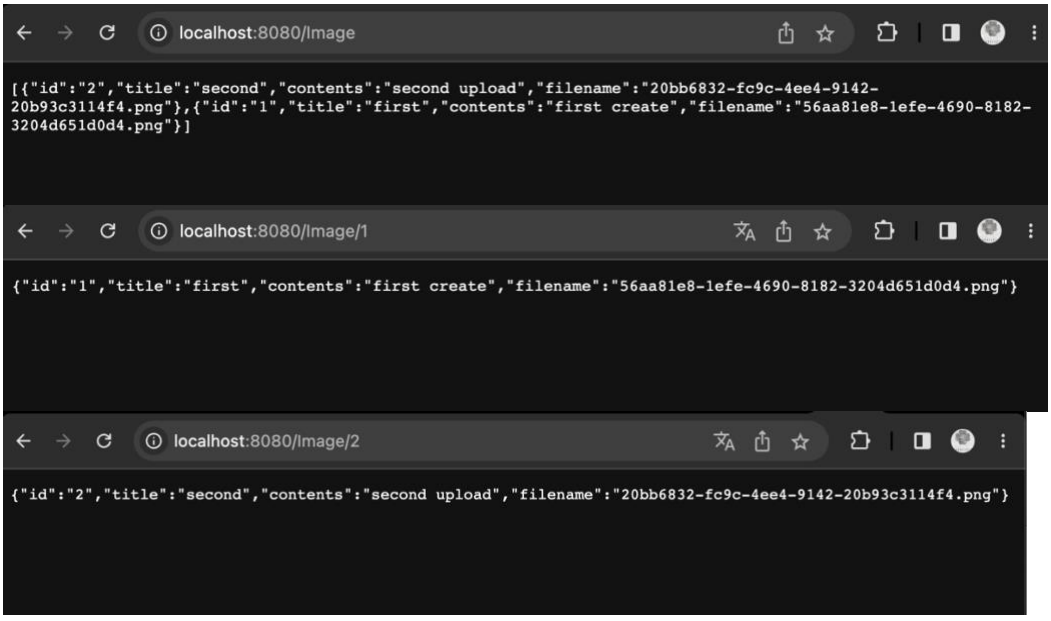




ImageFragment 때와 마찬가지로 ListFragment 또한 여러 데이터가 추가 된 것을 확인 할 수 있고 scrollable 한 것을 확인 할 수 있다.

C. Spring 추가 사항

이번 프로젝트에서는 json을 반환하는 controller가 필요했고 해당 controller를 추가하여 모든 데이터를 Json 반환 또는 지정 데이터 json 반환을 하도록 설계하였다. 이는 @RestController로 annotation을 두고 "/Image"라는 endpoint URL에는 모든 데이터 json 반환, "/Image/{id}"에는 해당 ID의 json 반환하도록 설계하였다. 이를 브라우저에서 확인하면 아래 사진과 같다.



4. Consideration

A. 구현 시 발생한 문제점 및 개선방안 설명

각 View에 대한 이동은 이전에 Kotlin으로 Android Studio에서 기본적인 코딩을 경험해본 적이 있기도 하고 BottomNavigationBar 같은 경우 강의 자료에 이미 구현되어있기 때문에 어렵지 않았다. 다만 제목, 내용, 사진을 보내는 작업이 매우 힘들었다. 처음에는 Spring을 새로 만들어서 업로드 받을 수 있도록 새로 구현해야 하는 것이라 생각하여 이전 과제의 코드를 버리고 새로 작성했다. 무엇인가 잘 되지 않았고 그때서야 이전의 코드에서 PostMapping 또는 PutMapping으로 업로드와 업데이트를 진행했던 것이 생각났고, 이에 안드로이드에서 HTTP Post 또는 Put을 할 수 있는지 찾아보았고 Retrofit2와 okhttp3를 사용하는 방법을 알 수 있었다. 그 중 이전 사용한 Kotlin 관련 자료에 Retrofit2를 사용해서 업로드를 하려고 했으나 비슷하지만 Kotlin으로 만든 코드를 직접 변환하는 과정은 어려웠고 인터넷상에 좀 더 자료가 많은 okhttp3을 사용하기로 했다. 여기서 또 문제는 이전 html에서 Post를 할 때는 form을 multipart로 보내도록 했다는 것이고 이에 대해 레퍼런스를 더 찾아야했다. 또한 업로드를 위한 이미지 선택 과정도 문제가 있었다. ACTION_PICK으로 이동하여 파일을 가져오는 작업을 하려고 했는데 해당 uri는 permission 없이도 잘 가져오지만 해당 uri의 경로를 가져와서 작업하려면 log에서 permission denied가 발생하였다. 이에 대해 READ_EXTERNAL_STORAGE에 대해 permission을 해야한다는 레퍼런스들이 많아서 진행하였으나 아무리 코드를 변형하여도 허용 여부 메세지가 나타나지 않았다. 이후 알고 보니 READ_MEDIA_IMAGES에 대해 permission을 요청해야 함을 알고 겨우 이미지를 서버로 보낼 수 있었다. 그리고 업로드 된 사진은 크기가 일정하지 않으므로 gridview의 item은 image를 center crop하도록 설정하였다.

Spring에서는 어렵지 않게 database에 있는 데이터를 json으로 바꾸었다. 다만 Android studio에서 간단하게 작동 테스트를 위해 image를 선택하지 않고 작동 여부를 확인할 때 앱이 강제 종료되는 상황이 있었는데 이는 spring에서 요구하는 format과 맞지 않은 형식이어서 오류가 발생했다. 이는 친절하게 아래에 실행에서 알려주었지만 발견하지 못해 6시간 이상 소요하였다. 업로드를 해결한 후에는 다른 작업은 비교적 빠르게 끝낼 수 있었다.

마지막으로 이런 방식으로 browser와 app에서 공용의 database를 다루는 것에 대해 이후 다른 프로젝트에 적용할 수 있는 방법이라고 생각하였다.