AtoZ

- 킬링 타임 Application -

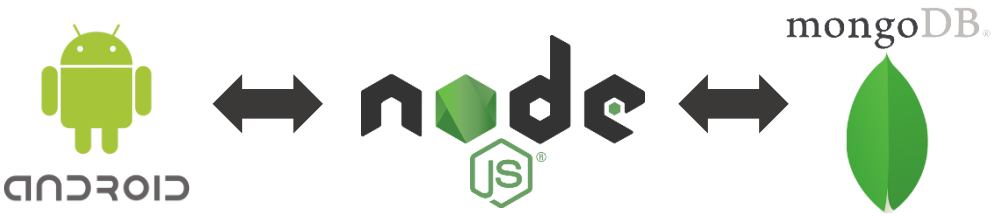
컴퓨터공학과 20141280 진 연주

Ⅰ. 요약

1. 프로젝트 주제

킬링 타임을 위해 간편하게 다양한 영상 / 음성 게시글들을 업로드하고 여러 사람들과 함께 즐길 수 있는 어플리케이션

2. 프로젝트 구조



Client

Editor : Android Studio
Retrofit2 API를 이용한 통신

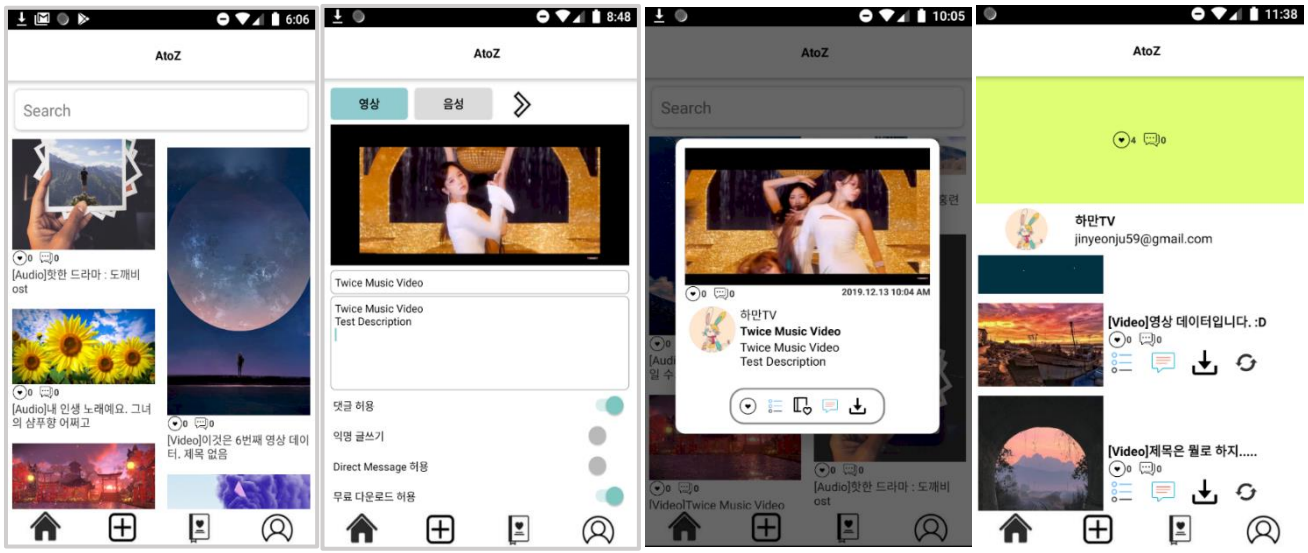
Server

Editor : Atom

Database

Editor : Robo 3T

3. 기본 기능



(1) (2) (3) (4)

(1) 메인 화면 : 업로드 된 전체 게시글들을 볼 수 있는 화면이다. 게시글을 클릭하면 작은 화면으로 게시글을 확인할 수 있고, 길게 클릭하면 세부화면에서 게시글을 재생할 수 있다.

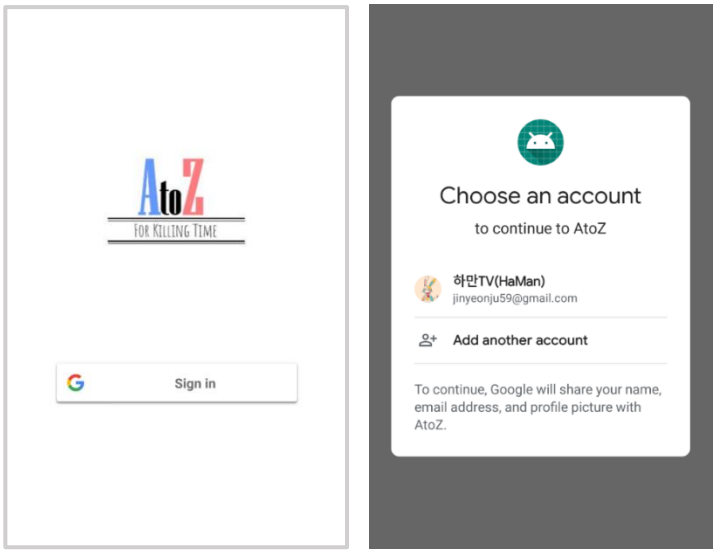
(2) 업로드 화면 : 게시글을 업로드할 수 있는 화면 (영상 / 음성 둘 다 가능하다.)

(3) 세부 화면 : 게시글을 좀 더 크게 볼 수 있을 뿐더러, 게시글의 세부 정보와 "좋아요", "앨범 추가", "다운로드" 기능 등이 추가적으로 제공된다.

(4) 사용자 화면 : 사용자가 업로드한 게시글을 볼 수 있는 화면이다. 메인 화면과 마찬가지로 사용자가 업로드한 게시글의 이미지를 클릭하면 해당 게시글을 세부화면에서 재생할 수 있다.

Ⅱ. 구현 내용

1. 로그인 화면



<로그인 화면 UI>

1.1 Android Code

- 앱을 설치한 이후, 구글 로그인을 성공하였다면 그 이후부터는 자동으로 로그인이 진행된다.

```
//자동 로그인 기능
private SharedPreferences pref;
private SharedPreferences.Editor editor;


if(pref.getBoolean( key: "isLogin", defValue: false)){ //이미 한번 로그인
    HashMap<String,String> body = new HashMap<>();
    body.put("email",pref.getString( key: "userEmail", defValue: ""));
    requestLogin(body);
}
```

- 이전 로그인 여부를 bool 형식으로 sharedPreferences에 저장하고 그 데이터(isLogin)를 앱을 실행할 때 검사하여, true라면 자동으로 앱을 실행하고, false라면 로그인 화면으로 이동한다.

```
//create firebase instance
firebaseAuth = FirebaseAuth.getInstance();
btnGoogleLogin = findViewById(R.id.btn_Login);

//google login
GoogleSignInOptions googleSignInOptions = new GoogleSignInOptions.Builder(GoogleSignInOptions.DEFAULT_SIGN_IN)
        .requestIdToken(getString(R.string.default_web_client_id))
        .requestEmail()
        .build();

googleSignInClient = GoogleSignIn.getClient( activity: this, googleSignInOptions);
btnGoogleLogin = findViewById(R.id.btn_Login);
btnGoogleLogin.setOnClickListener((v) → {
        Log.i(TAG, msg: "Login Click");
        //request google login
        Intent signInIntent = googleSignInClient.getSignInIntent();
        startActivityForResult(signInIntent,SIGN_IN_REQUEST_CODE);
});
```

- sharedPreferences에 isLogin 값이 false일 경우, 로그인 화면으로 이동한다.

- 사용자가 로그인 버튼을 클릭하면 구글 로그인 activity를 수행하고 그 결과를 받아온다.

```
//google login button response
if(requestCode == SIGN_IN_REQUEST_CODE){
    Task<GoogleSignInAccount> task = GoogleSignIn.getSignedInAccountFromIntent(data);
    try{
        //google login success
        GoogleSignInAccount account = task.getResult(ApiException.class);

        HashMap<String, String> body = new HashMap<>();
        body.put("email",account.getEmail());
        body.put("nickname",account.getGivenName());
        body.put("profile", account.getPhotoUrl().toString());

        requestLogin(body);

    }catch (ApiException exception){

    }
}
```

- 구글 로그인이 성공했다면, 필요한 사용자 정보를 Hashmap에 저장한다.

- 이 때, 구글 로그인 사용자 정보는 GoogleSignInAccount 객체가 가지고 있다.

  Ex. (GoogleSignInAccount).getEmail( ) : 사용자의 구글 이메일 주소

  Ex. (GoogleSignInAccount).getPhotoUrl( ) : 사용자의 구글 프로필 사진 주소(Url 객체이므로 String으로 저장할 때는 toString( ) 함수를 통해 String으로 변경해줄 필요가 있다.)

- 로그인에 성공했을 시, 서버로 로그인 요청을 보낸다.(requestLogin(body))

이 때, 첫 로그인 요청은 email, nickname, profile 정보를 자동 로그인 시에는 email 정보만을 보내준다.

```
call.enqueue(new Callback<GetUserResponse>() {
    @Override
    public void onResponse(Call<GetUserResponse> call, Response<GetUserResponse> response) {
        Log.i(TAG, msg: "RESPONSE : "+response.body().getResponseBody().getEmail());
        if(response.body().getResponseCode() == Common.getInstance().REQUEST_SUCCESS){ //로그인 성공
            Common.getInstance().setUserData(response.body().getResponseBody());

            editor = pref.edit();
            //로그인은 한 번만
            editor.putString("userEmail",response.body().getResponseBody().getEmail());
            editor.putBoolean("isLogin",true);
            editor.commit();

            //로그인 성공 시 메인 화면으로 이동
            Intent moveToMainView = new Intent( packageContext: LoginActivity.this, MainActivity.class);
            startActivity(moveToMainView);

            finish();
        }
    }

    @Override
    public void onFailure(Call<GetUserResponse> call, Throwable t) {
        Log.i(TAG, msg: "Response Error : "+t.toString());

    }
});
```

- requestLogin 함수에서 서버로 요청을 보내는 Retrofit response는 위와 같으며, 그 때의 response Body는 아래와 같다.

```java
public class GetUserResponse {
    //Variable name should be same as in the json response from server
    @SerializedName("responseCode")
    int responseCode;
    @SerializedName("responseBody")
    UserData responseBody;

    public UserData getResponseBody(){return this.responseBody;}
    public int getResponseCode(){return this.responseCode;}
}
```

```java
public class UserData {
    @SerializedName("email") String email;
    @SerializedName("nickname") String nickname;
    @SerializedName("profile") String profileUrl;
    @SerializedName("like_no") int likeNo;
    @SerializedName("comment_no") int commentNo;
```

- responseCode에는 로그인 요청에 따른 결과 상태값이 저장되고 responseBody에는 사용자와 관련된 데이터(UserData 객체)가 저장된다.

- 로그인에 성공했을 때(responseCode가 200(Common.getInstance().REQUEST_SUCCESS)), 사용자의 이메일 주소(다음 로그인 시 사용)와 로그인이 성공했다는 정보(isLogin)을 각각 sharedPreferences에 저장하고 메인 화면으로 이동한다.

1.2. Server Code

```
//회원 가입
app.put('/signIn/google',function(req,res){
  // console.log(req);
  // {email : , nickname : , profile : }
  User.find({email : req.body.email},function(error,result){
    if(error){
      res.send({responseCode:404,responseBody:"Error"});
    }else if(result.length == 0){ //회원 등록
      var user = new User({
        email : req.body.email,
        nickname : req.body.nickname,
        profile : req.body.profile
      });

      user.save(function(error, post){
        console.log(post);
        if(error) {
          console.log("Database Error : "+error);
          res.send({responseCode:404, responseBody:"Fail"});
        }
        else res.send({responseCode:200,responseBody:post});
      });
    }else{
      console.log(result[0]);
      res.send({responseCode:200,responseBody:result[0]});
    }
  });
});
```

- 클라이언트로부터 로그인 요청이 들어왔을 경우, 데이터 베이스에서 사용자 email이 이미 존재하는지 아닌지 검색한다.(find)

- 사용자 이메일 정보가 이미 데이터 베이스에 저장되어 있다면, 그 결과값(사용자 정보)를 response 해준다. 이 때 responseCode는 200(REQUEST_SUCCESS)이다.

- 사용자 이메일 정보가 데이터베이스에 존재하지 않을 경우에는 해당 사용자 정보를 데이터베이스에 새로

저장(save)하고, 그 결과(사용자 정보)를 response 해준다. 이 때, responseCode는 200(REQUEST_SUCCESS)이다.

1.3. 사용자 데이터 베이스 스키마

```
var userSchema = new mongoose.Schema({
  email : {type:String, required:true},
  nickname : {type : String, required : true},
  // password : {type : String, required : true},
  profile : {type : String, default:"default_profile_image.jgp"},
  album : [{type:String}],
  media_id_like : [{type:String}],
  like_no : {type:Number,default:0},
  comment_no : {type:Number, default:0}
});
```

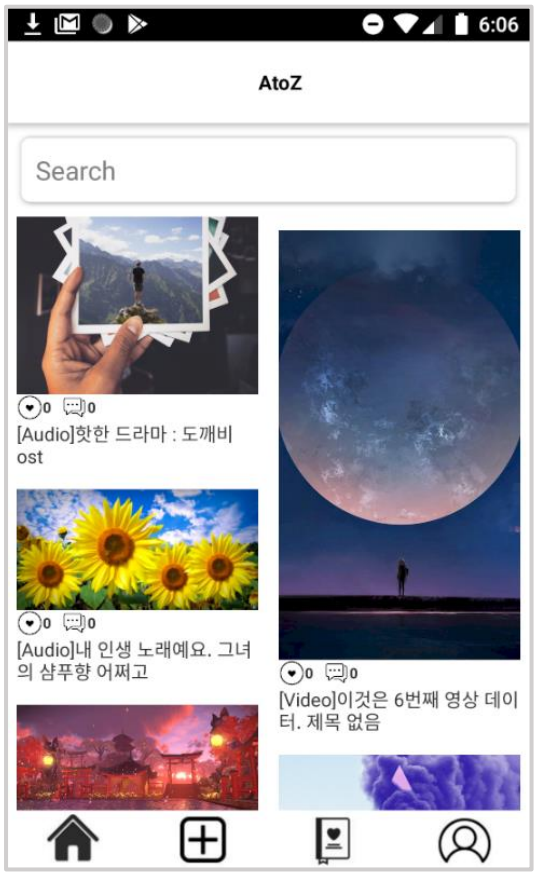
- Email, nickname ,profile 정보는 String 타입으로 저장한다.

- album은 사용자가 앨범에 추가한 게시글의 id가 배열 형식으로 저장된다.

- media_id_like : 해당 사용자가 "좋아요"를 누른 게시글의 id가 배열 형식으로 저장된다.

- like_no, comment_no : 해당 사용자가 업로드한 게시글에서 받은 좋아요 개수와 댓글 개수가 저장된다.

2. 메인 화면



2.1. 기능

- 데이터 베이스에 있는 게시물(음성 / 영상)을 서버로부터 받아와 리스트 형식으로 보여준다.

- 각각의 게시물에는 게시물 이미지, 좋아요 개수, 댓글 개수, 음성 게시물인지, 영상 게시물인지에 대한 표시, 제목에 대한 정보를 사용자에게 제공한다.

- 게시글을 짧게 한 번 클릭하면, 해당 게시글이 재생된다. 영상이라면 이미지가 사라지고 영상이 재생되며, 음성이라면 게시글 이미지가 유지되면서 음성만이 재생된다.

- 게시글을 길게 클릭하면 해당 게시글에 대한 세부 정보와 미디어를 볼 수 있는 화면으로 이동한다.

- 하나의 게시글이 재생되는 도중, 다른 게시글을 클릭하면, 현재 재생중인 미디어가 정지하고 클릭한 미디어가 재생되는 형식으로 기능을 제공한다.

- 미디어가 재생 중에 동일한 게시글을 클릭하면 일시 정지 기능을 제공한다.

2.2. Android Code

```
@Override
public void onResponse(Call<GetResponse> call, Response<GetResponse> response) {
    Log.i(TAG, msg: "SUCCESS / "+response.body().getResponseBody().toString());

    ArrayList<MediaData> responseBody = response.body().getResponseBody();

    for(int i = 0 ; i < responseBody.size(); i++){

        MediaData mediaData = responseBody.get(i);
        int dataType = mediaData.getType();

        MainMultiDataAdapter adapter;
        if(dataType == 0){ //video
            adapter = new MainVideoAdapter(context,container,mediaData,R.layout.item_mainlist_for_video, R.id.main_for_video,
                    R.id.item_main_video_likeNo,R.id.item_main_video_commentNo);
            adapter.setThumbnailImage(layoutLeft,layoutRight,R.id.thumbnail_for_video);
        }else{ //audio
            adapter = new MainAudioAdapter(context,container,mediaData,R.layout.item_mainlist_for_audio, R.id.main_for_audio,
                    R.id.item_main_audio_likeNo,R.id.item_main_audio_commentNo);
            adapter.setThumbnailImage(layoutLeft,layoutRight,R.id.thumbnail_for_audio);
        }

        adapter.getBackgroundLayout().setOnClickListener(HomeFragment.this);
        adapter.getBackgroundLayout().setOnLongClickListener(HomeFragment.this);

        multiAdapterList.put(mediaData.getId(),adapter);
    }
}
```

- 메인 화면으로 이동하거나, 앱을 실행하면 Retrofit2를 이용하여 서버에게 미디어 데이터(게시글)에 대한 데이터를 요청하여, 해당 정보들을 리스트로 받아온다.

- 이 때의 response 데이터는 아래와 같다. responseCode는 결과 status code 값이, responseBody에는 요청한 게시글 데이터가 MediaData 객체로 저장된다.

```
public class GetResponse {

    //Variable name should be same as in the json response from server
    @SerializedName("responseCode")
    int responseCode;
    @SerializedName("responseBody")
    ArrayList<MediaData> responseBody;

    public ArrayList<MediaData> getResponseBody(){return this.responseBody;}
    public int getResponseCode(){return this.responseCode;}
}
```

```
public class MediaData {

    @SerializedName("_id") String id;
    @SerializedName("user_id") String userEmail;
    @SerializedName("media_type") int type;
    @SerializedName("thumbnail") String imageUri;
    @SerializedName("media") String multiUri;
    @SerializedName("title") String title;
    @SerializedName("description") String description;
    @SerializedName("commentable") boolean commentable;
    @SerializedName("anonymouse") boolean anonymous;
    @SerializedName("messagable") boolean messagable;
    @SerializedName("downloadable") boolean downloadable;
    @SerializedName("like_no") int likeNo;
    @SerializedName("comment_no") int commentNo;
    @SerializedName("date") Date date;
    @SerializedName("comment") ArrayList<CommentData> comments;
```

- 정상적으로 서버로부터 미디어 데이터(게시글 정보)를 받아오는데 성공했다면, 받아온 게시글 데이터를 반복문을 통해서 하나씩 화면에 추가해주는 기능을 수행한다.

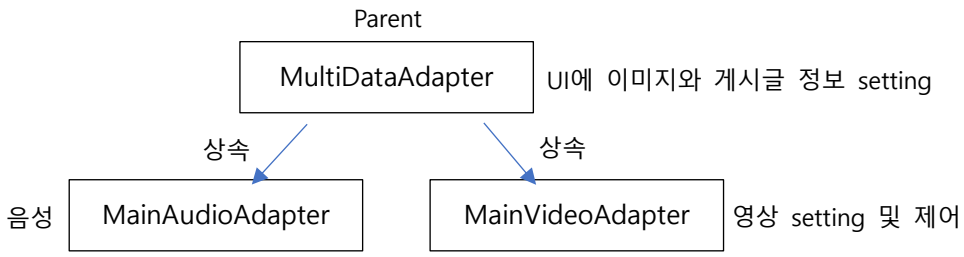
- 게시글 타입이 음성이라면 MainAudioAdapter를 통해 UI를 설정해주고, 영상일 경우, MainVideoAdapter를 통해 UI를 설정해준다. 이는, 영상일 경우에는 videoView가 추가적으로 필요하며 두 미디어 타입에 따라 미디어 재생 방식의 차이점이 있기 때문에 다른 Adapter를 이용한다.

- 게시글을 클릭하거나 길게 클릭했을 때의 이벤트(화면 전환 또는 영상/음성 재생)을 위해 리스터를 설정해준다.(setOnClickListener, seOnLongClickListener)

- 게시글의 클릭 event를 수행함에 있어서, 어떤 게시글을 클릭했는지 알기 위해 게시글 view에 해당 게시글의 id를 tag로 setting해주고, 게시글을 클릭했을 때 tag에 저장되어 있는 게시글의 id에 따른 게시글의

adapter(게시글의 MediaData를 가지고 있다.)를 매칭하기 위해 HashMap에 게시글의 id를 key로 adapter를 value로 저장해준다.

2.2.1. 미디어 adapter 구조(UI Setting)



2.2.2. 게시글 짧게 클릭 event(onClick)

```
String index = (String)view.getTag();

MainMultiDataAdapter adapter = multiAdapterList.get(index);
MediaData data = adapter.getCurrentMediaData();

int type = data.getType();
```

i. 게시글을 click하게 되면 클릭한 view에 저장된 tag 값을 가지고 온다. 위에서 말했듯이, view에 저장된 tag의 값은 각 게시글의 id 값으로 String 형에 해당한다.

ii. 게시글의 id 값을 통해 해당 게시글의 MediaData 객체를 가지고 온다.

```
if(type == 0){ //video
    MainVideoAdapter videoAdapter = (MainVideoAdapter)adapter;
    if(currentMediaId.equals(index)){ //같은 이미지 클릭
        if(playing == 1){ //재생중
            videoAdapter.pause();
            playing = 2; //일시정지
        }else if(playing == 2){ //일시 정지 중
            videoAdapter.replay();
            playing = 1; //재생
        }
    }else { //다른 이미지 클랙
        if(playing == 1 && currentType == 1){ //오디오가 이미 재생중
            mediaPlayer.stop();
            mediaPlayer.reset();
        }else if( (playing==1 || playing == 2) && currentType == 0){ //비디오가 이미 재생중이거나 일시정지 중
            MainVideoAdapter previousVideo = (MainVideoAdapter)multiAdapterList.get(currentMediaId);
            previousVideo.stop();
        }

        videoAdapter.start();
        //새로운 data 로 설정
        playing = 1;
        currentType = 0;
        currentMediaId = index;
    }
}
```

iii. 현재 클릭한 게시글이 영상일 경우에 제공되는 기능은 아래와 같다.

A. 현재 클릭한 게시글이 이전에 선택한 게시글과 같은 것일 경우, 해당 영상이 재생 중이라면 adapter의 pause 함수를 호출해 영상을 일시 정지한다. 영상이 일시 정지 중이라면 adapter의 replay 함수를 호출해 영상을 재생한다.(영상을 제어하는 adapter의 함수에 대해서는 후에 서

술할 것이다.)

B. 현재 클릭한 게시글이 이전에 선택한 게시글과 다른 것일 경우, 이전에 선택한 게시글이 음성이고 재생 중이라면 해당 음성을 MediaPlayer 객체에서 제공해주는 stop 함수를 호출해 정지한다. 또한, 이전에 선택한 게시글이 영상이고 재생 중이거나 일시 정지 중이라면 이전 게시글의 adapter의 stop 함수를 호출해 영상을 정지한다.

C. B와 같이 다른 게시물을 클릭했을 때는 이전에 재생 중이던 미디어 데이터에 대한 처리를 완료한 이후에 클릭한 영상에 대한 재상이 이루어진다. 영상을 재생할 때는 해당 adapter의 start( ) 함수를 호출한다..

```
else{ //audio
    if(currentMediaId.equals(index)){ //같은 이미지 클릭
        if(playing == 1){ //재생중
            mediaPlayer.stop();
            playing = 0;
        }else{ //재생 안되고 있음.
            playSong(data.getMultiUri());
            playing = 1;
        }
    }else{ //다른 이미지 클릭
        if( playing > 0 && currentType == 0){ //비디오가 이미 재생중이거나 일시정지
            MainVideoAdapter videoAdapter = (MainVideoAdapter)multiAdapterList.get(currentMediaId);
            videoAdapter.stop();
        }

        playSong(data.getMultiUri());
        //새로운 data 로 설정
        playing = 1;
        currentType = 1; //audio
        currentMediaId = index;
    }
}
```

iv. 현재 클릭한 게시글이 음성일 경우 제공되는 기능은 아래와 같다.

A. 현재 클릭한 게시글이 이전에 선택한 게시글과 같은 것일 경우, 해당 음성이 재생 중이라면 MediaPlayer class에서 제공되는 pause 함수를 호출해 영상을 일시 정지한다. 음성 재생 되고 있지 않다면 playSong 함수를 호출해 해당 음성을 재생한다.(playSong 함수에 대해서는 후에 서술할 것이다.)

B. 현재 클릭한 게시글이 이전에 선택한 게시글과 다른 것일 경우, 이전에 선택한 게시글이 영상이고 재생 중이거나 일시 정지 중이라면 이전 게시글의 adapter의 stop 함수를 호출해 영상을 정지한다.

C. 이전에 선택한 게시글이 음성인 경우에는 같은 MediaPlayer 객체를 이용하여 제어를 하고 있기 때문에 playSong 함수를 통해 재생하고자 하는 음성의 Url만 변경해 주면 된다.

2.2.3 게시글을 길게 클릭 event(onLongClick)

```
String index = (String)view.getTag();

MainMultiDataAdapter adapter = multiAdapterList.get(index);
MediaData data = adapter.getCurrentMediaData();

int type = data.getType();
```

i. 게시글을 짧게 클릭하는 onClick에서와 마찬가지로 view의 tag로 저장되어 있는 게시글의 id를 통해 게시글 정보(MediaData 객체)를 가지고 온다.

```
if(playing > 0){ //이미 media가 재생 중
    if(currentType == 0) { //비디오가 재생중
        MainVideoAdapter videoAdapter = (MainVideoAdapter)multiAdapterList.get(currentMediaId);
        videoAdapter.stop();
    }else{ //오디오가 재새중
        mediaPlayer.stop();
        mediaPlayer.reset();
        playing = 0; //정지
    }
}
```

ii. Bool 형의 playing을 통해 음성 또는 영상이 재생 중이라면 해당 미디어를 정지한다.

```
//media 타입에 따라 dialog에서 재생
if(type == 0){ //video
    DetailedMediaDialogForVideo videoDialog = new DetailedMediaDialogForVideo(getContext(),data);
    videoDialog.show();
}
else{ //audio
    DetailedMediaDialogForAudio audioDialog = new DetailedMediaDialogForAudio(getContext(),data);
    audioDialog.show();
}
```

iii. 게시글의 데이터 타입(음성 또는 영상)에 따라 알맞은 dialog를 setting하고 dialog를 보여준다.(해당 UI는 다음 3. 음성 / 영상 세부 화면에서 설명)

```
return true;
```

iv. 사용자가 게시글을 길게 클릭했을 경우, 짧게 클릭한(onClick) 경우와 구별하기 위해 반환 값은 true로 setting 한다.

2.2.4. 음성 재생 : playSong 함수

```
//Function to play a song
public void playSong(String audioUri){

    //playsong
    try{
        mediaPlayer.reset();
        String path = AppConfig.BASE_URL+"/"+audioUri;

        mediaPlayer.setDataSource(path);
        mediaPlayer.prepare();
        mediaPlayer.start();
    }catch (IOException exception){
        exception.printStackTrace();
    }
}
```

i. 음성 재생에서는 android에서 제공되는 MediaPlayer class를 사용한다.

ii. 게시글 데이터에 해당하는 MediaData 객체에 저장되어 있는 음성 데이터의 주소 값(String)을 서버의 기본 주소(AppConfig.BASE_URL) 뒤에 붙여 데이터의 경로값(path)를 구할 수 있다.

iii. 해당 경로값(path)를 MediaPlayer class에서 제공하는 setDataSource를 통해 setting하면 서버에 있는 원하는 음성 데이터를 가지고 올 수 있다.

iv. 음성 데이터의 경로를 설정하였다면 start( ) 함수를 통해 요청한 음성을 재생한다.

2.2.5. 영상 재생 : MainVideoAdapter의 start( ) 함수

```
thumbnailImage.setVisibility(View.INVISIBLE); //썸네일 이미지 제거

try{
    //request focus
    videoView.requestFocus();
    videoView.setOnPreparedListener((mp) → {
            mp.setLooping(true);
            buffering.setVisibility(View.INVISIBLE);
            videoView.start();
    });

    videoView.setOnInfoListener((mp, what, extra) → {

            if(what == MediaPlayer.MEDIA_INFO_VIDEO_RENDERING_START){
                buffering.setVisibility(View.INVISIBLE);
                return true;
            }else if(what == MediaPlayer.MEDIA_INFO_BUFFERING_START){
                buffering.setVisibility(View.VISIBLE);
                buffering.bringToFront();
                return true;
            }else if(what == MediaPlayer.MEDIA_INFO_BUFFERING_END){
                buffering.setVisibility(View.INVISIBLE);
                return true;
            }
            return false;
    });

    buffering.setVisibility(View.VISIBLE);
    buffering.bringToFront();
    videoView.setVideoURI(Uri.parse(AppConfig.BASE_URL+"/"+currentMediaData.getMultiUri()));

}catch (Exception exception){
    exception.printStackTrace();
}
```

i. 음성은 이미지를 유지하며, 음성만을 재생하면 되었지만 영상을 재생하기 위해서는 상대적으로 videoView의 위쪽에 위치하는 이미지를 제거해줘야 한다.(View.INVISIBLE)

ii. 영상은 videoView라는 영상을 위한 특정 view class에서 제공되는 메소드들을 통해서 영상을 제어할 수 있다.

iii. 음성과 마찬가지로 서버로부터 가지고 오는 영상의 주소는 서버의 기본 주소 + 영상 데이터 이름이다.(AppConfig.BASE_URL + "/" + 영상 데이터 이름)

iv. 영상은 음성에 비해서 데이터의 크기가 상대적으로 크므로, 버퍼링에 대비해 사용자에게 버퍼링 정보를 제공하기 위해 progressbar를 보여준다.(buffering.setVisibility(View.VISIBLE))

v. videoView에서 setOnPreparedListener( )로 setting되는 onPrepared( )는 서버로부터 영상 데이터를 가지고 오는 준비가 완료되었을 때 발생하는 event이다. 영상이 준비되었다면, 사용자에게 보여주

던 buffering progressbar를 제거하고 영상을 재생한다. 이 때 videoView class에서 제공되는 start( ) 함수를 사용한다.

vi. 하지만 영상은 재생되고 있는 도중에도 버퍼링이 발생할 수 있기 때문에 setOnInfoListener를 사용하여 관련 event를 등록해준다. 이는 버퍼링이 발생했을 경우 buffering progressbar로 보여주기 위함이다.

2.2.6. 영상 정지 : MainVideoAdapter의 stop( ) 함수

```
//영상 종료
public void stop(){
    Log.i(TAG, msg: "STOP");
    thumbnailImage.setVisibility(View.VISIBLE);
    thumbnailImage.bringToFront();

    this.videoView.stopPlayback();

    //videoview 의 크기를 다시 0으로( thumbnail 보다 videoview의 크기가 클 경우 밖으로 보이는 현상 방지 )
    RelativeLayout.LayoutParams params = new RelativeLayout.LayoutParams(ViewGroup.LayoutParams.MATCH_PARENT, h: 0);
    videoView.setLayoutParams(params);
}
```

i. 영상을 정지하기 위해서는 videoView class에서 제공되는 stopPlayback( ) 함수를 사용한다.

ii. 이 때, 재생을 위해 제거했던 썸네일을 다시 보여준다.

2.2.7. 영상 일시 정지 : MainVideoAdapter의 pause( ) 함수

```
//영상 일시 정지
public void pause(){
    Log.i(TAG, msg: "PAUSE");
    buffering.setVisibility(View.INVISIBLE);
    videoView.pause();
    currentTime = videoView.getCurrentPosition();
}
```

i. 영상에 일시 정지는 videoView class에서 제공되는 pause( ) 함수를 사용한다.

ii. 이 때, buffering progressbar가 보여지고 있는 중이라면, 이를 제거한다.

2.2.8. 영상 재생(일시정지 상태에서 다시 재생) : MainVideoAdapter의 replay( ) 함수

```
//영상 다시 재생
public void replay(){
    Log.i(TAG, msg: "Replay");
    videoView.seekTo(currentTime);
    videoView.start();
}
```

i. 일시 정지를 했다 다시 재생하기 위해서는 일시 정지를 한 순간의 위치를 알고 있어야 한다. 이 때, 일시 정지를 제공하는 pause( ) 함수에서 getCurrentPosition( ) 함수를 통해 얻은 currentTime 변수를 이용한다.

ii. videoView class에서 제공하는 seekTo 함수를 이용해 영상 데이터 중 일시 정지한 위치로 이동하고 그 위치에서부터 다시 재생을 시작한다.

2.3. Server Code

```
//get post
app.get('/getPost',function(req,res){
  console.log("request All Post");
  Post.find(function(error, posts){
    if(error) res.send({responseCode:404,responseBody:"Fail"});
    else res.json({responseCode : 200, responseBody : posts});
  });
});
```

- 클라이언트로부터 게시글 요청을 받으면 Post collection에 있는 모든 document를 response로 보내준다.

- 클라이언트가 전송된 게시글 정보에서 게시글 url만으로도 서버 저장소에 접근할 수 있도록 아래와 같이 multer를 이용해 static 폴더를 설정해준다.

```
app.use(express.static(__dirname+'/media'));
app.use('/static',express.static(__dirname+'/media'));
```

- 게시글 데이터 스키마

```
//db schema
//post Database
var postSchema = new mongoose.Schema({
  user_id: {type:String, required: true}, //post를 작성한 user id
  media_type: {type:Number, required : true}, //0 : video, 1 : audio
  thumbnail : {type:String, required: true}, //thumbnail image name
  media : {type:String,required:true}, //media data name
  title : {type:String,default:""}, //post title
  description : {type:String,default:""}, //post description
  date : {type:Date, default:Date.now}, //upload time
  commentable : {type:Boolean, default:false},
  anonymous : {type:Boolean, default:false},
  messagable : {type:Boolean, default:false},
  downloadable : {type:Boolean, default:false},
  like_no : {type:Number, default : 0},
  comment_no : {type:Number, default: 0},
  comment: [{
    nickname: {type:String,required:true},
    profile: {type:String,required:true},
    description: {type:String,required:true}
  }]
});
```

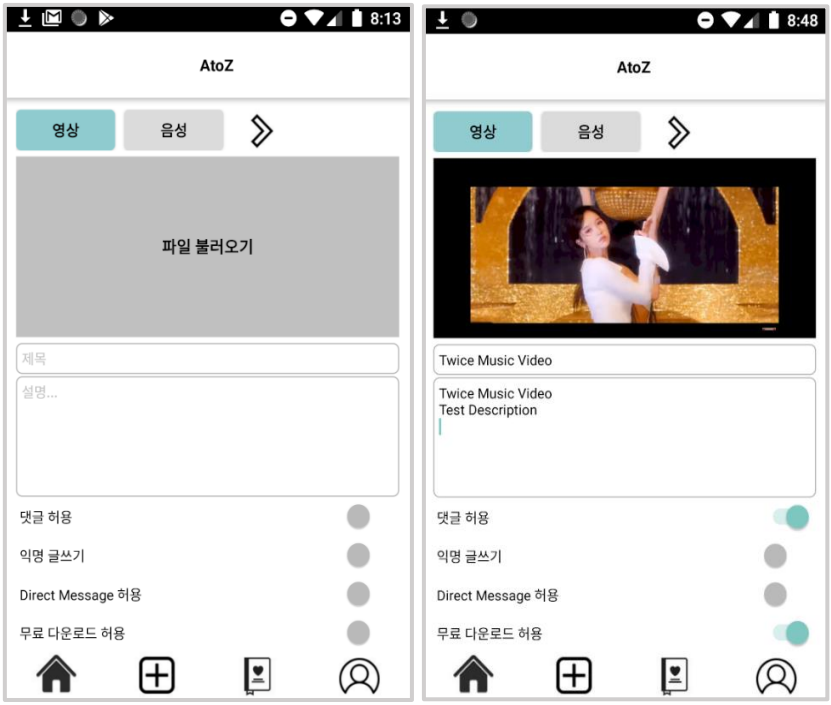
i. 클라이언트는 media 값을 이용해 서버 저장소에 저장된 데이터에 접근할 수 있다.

ii. comment는 해당 게시글에 달린 댓글에 대한 정보들을 json 형식으로 가지고 있는 배열이다.

iii. like_no, comment_no : 각각 해당 게시글의 "좋아요" 개수와 댓글 개수를 나타낸다.

3. 음성 / 영상 업로드 화면

3.1. 업로드할 영상 선택



<영상 업로드 UI>

(1) 제공 기능

- 화면에 "파일 불러오기"를 클릭하면, 휴대폰 내에 저장되어 있는 영상을 불러올 수 있다. 영상을 선택하면 위 오른쪽 이미지와 같이 해당 영상을 볼 수 있다. 해당 영상의 왼쪽을 클릭하면 back, 중앙을 클릭하면 play/pause, 오른쪽을 클릭하면 forward 기능을 제공한다.

- 원하는 영상의 제목과 설명을 적고 게시글 설정을 완료한 이후 상단의 화살표 버튼을 클릭하면 서버로의 영상 업로드가 시작된다.

- 영상 업로드가 성공적으로 완료되면, 사용자 화면으로 자동 이동된다.

(2) Android Code

- 원하는 데이터 타입을 선택 : 상단의 영상 버튼 클릭(onClick event)

```
if(view == btnVideo && currentDataType != 0){ //data type 을 video 로 변경

    btnVideo.setBackgroundColor(getActivity().getColor(R.color.dataType_button_bg_on));
    btnAudio.setBackgroundColor(getActivity().getColor(R.color.dataType_button_bg_off));

    //파일 선택 초기화
    if(selectMedia){ //파일이 선택되어 있다면
        resetMediaView();
    }

    currentDataType = 0; //video
```

i. 영상을 업로드하고 싶을 경우에는 상단에 영상/음성 버튼 중, 영상을 클릭한다.

ii. 만약 이전에 다른 파일을 선택했다면(이 경우에는 음성 파일) 해당 파일에 대해 setting view를 제거한다.(이 때 사용하는 resetMediaView( )를 후에 설명할 것이다.)

iii. 그리고 나서, 현재 선택된 데이터(사용자가 업로드 하고 싶은 데이터)의 타입을 저장하는 currentDataType의 값을 영상(0)으로 변경한다.

- 업로드 하고 영상을 선택 : selectVideoFile( )

```
//영상 파일 선택
private void selectVideoFile(){
    Intent selectVideoIntent = new Intent(Intent.ACTION_GET_CONTENT);
    selectVideoIntent.setType("video/*"); //모든 video type 검색
    startActivityForResult(selectVideoIntent, SELECT_MEDIA_REQUEST_CODE);
}
```

i. Intent를 사용하여 휴대폰 내에 있는 모든 video 파일들을 선택할 수 있는 activity를 실행한다.

ii. 사용자가 영상을 선택했거나 다시 본 어플리케이션 화면으로 돌아왔을 경우의 결과를 받기 위해 startActivity가 아닌 startActivityForResult로 activity를 실행한다.

```
@Override
public void onActivityResult(int requestCode, int resultCode, Intent data){
    super.onActivityResult(requestCode,resultCode,data);

    if(resultCode == getActivity().RESULT_OK){ //success selecting video file

        if(requestCode == SELECT_MEDIA_REQUEST_CODE){

            if(data != null){
                mediaFilePath = getPath(data.getData());
                initializeVideoPlayer();

                selectMedia = true;
            }

        }

    }else if(resultCode == getActivity().RESULT_CANCELED){ //cancel to select video file
        print("Select Video File is Cancelled");
    }else{ //fail to select video file
        print("Fail to Select Video File. Please Try again!");
    }
}
```

iii. 사용자가 영상을 선택하지 않고 다시 본 어플리케이션으로 돌아왔을 경우에는 Toast를 사용하여 화면에 영상을 선택하지 않았다는 정보를 띄워준다.

iv. 사용자가 영상을 선택하였을 경우에는 해당 영상의 경로를 getPath( ) 함수를 통해 받아오고, 그 경로를 사용하여 화면에 해당 영상을 재생한다.(initializeVideoPlayer( ))

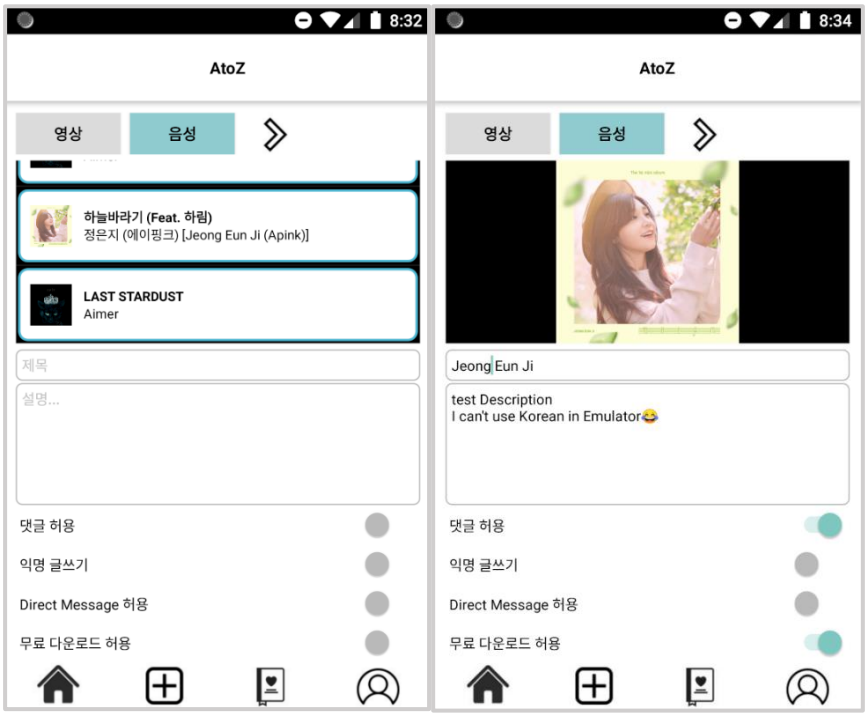
v. initializeVideoPlayer( )에서는 viewView를 동적으로 생성한다.(음성 파일을 업로드할 경우에는 videoView가 추가적으로 필요하지 않기 때문에, 영상일 경우에만 동적으로 생성하는 것이다.)

vi. 이후, 현재 재생위치를 0으로 setting하고, 영상을 재생할 linearlayout에 추가하고, 재생을 시작한다.

vii. 이 때 영상이 linearLayout보다 작아 왼쪽으로 치우치는 것을 방지하기 위해 param을 지정한다.

```
//video type 의 file 을 선택했을 경우, 해당 video view setting 및 재생
private void initializeVideoPlayer(){

    //새로운 video video view 생성
    videoView = new VideoView(getContext());
    videoView.setTag(0); //현재 재생 위치

    //media view 가운데 정렬
    RelativeLayout.LayoutParams params = new RelativeLayout.LayoutParams(
                        ViewGroup.LayoutParams.MATCH_PARENT,
                        ViewGroup.LayoutParams.MATCH_PARENT);
    params.addRule(RelativeLayout.CENTER_HORIZONTAL,RelativeLayout.TRUE);
    params.addRule(RelativeLayout.CENTER_VERTICAL,RelativeLayout.TRUE);

    videoView.setLayoutParams(params);

    //media view 배경 색 -> 검은색
    mediaLayout.setBackgroundColor(getActivity().getColor(R.color.upload_media_bg_on));
    videoView.setVideoPath(mediaFilePath);
    mediaLayout.addView(videoView);
    videoView.start();

}
```

3.2. 업로드할 음성 선택



<음성 업로드 UI>

(1) 제공 기능

- 음성 버튼을 클릭하면, 휴대폰 내에 있는 음성 파일들의 리스트를 보여준다. 사용자는 이 중에서 업로드하길 원하는 음성 파일을 클릭하면 된다

- 원하는 음성의 제목과 설명을 적고 게시글 설정을 완료한 이후 상단의 화살표 버튼을 클릭하면 서버로의 음성 업로드가 시작된다.

(1) Android Code

- 원하는 데이터 선택 : 상단의 음성 버튼을 클릭(onClick Event)

  i. 음성을 업로드하고 싶을 경우에는 상단에 영상/음성 버튼 중, 음성을 클릭한다.

  ii. 만약 이전에 다른 파일을 선택했다면(이 경우에는 영상 파일) 해당 파일에 대해 setting view를 제거한다.(이 때 사용하는 resetMediaView( )를 후에 설명할 것이다.)

  iii. 그리고 나서, 현재 선택된 데이터(사용자가 업로드 하고 싶은 데이터)의 타입을 저장하는 currentDataType의 값을 음성(1)으로 변경한다.

```
}else if(view == btnAudio && currentDataType != 1){ //data type 을 audio 로 변경

    btnVideo.setBackgroundColor(getActivity().getColor(R.color.dataType_button_bg_off));
    btnAudio.setBackgroundColor(getActivity().getColor(R.color.dataType_button_bg_on));

    //파일 선택 초기화
    if(selectMedia){ //파일이 선택되어 있다면
        resetMediaView();
    }

    currentDataType = 1; //audio
    selectFile();
```

- 업로드하고 싶은 음성 파일 선택 : selectAudioFile( )

  i. 음성 파일은 영상과 달리 UI의 왼쪽 이미지처럼 휴대폰 내의 있는 음성 파일들을 리스트로 보여준다.

```
//음성 파일 선택
private void selectAudioFile(){

    audioListView = new ListView(getContext());
    AudioFileManager audioManager = new AudioFileManager(getContext());
    audioList = audioManager.getAudioList();
    final AudioListAdapter audioAdapter = new AudioListAdapter(getContext(),audioList);
    audioListView.setAdapter(audioAdapter);
```

  ii. 영상과 마찬가지로 음성파일에서는 listView를 동적으로 생성한다.(영상에서는 필요 없기 때문이다.)

```
public ArrayList<HashMap<String, String>> getAudioList(){

    Cursor cursor = context.getContentResolver().query(
            MediaStore.Audio.Media.EXTERNAL_CONTENT_URI, projection: null,
             queryArgs: null, cancellationSignal: null);


    while(cursor.moveToNext()){
        HashMap<String, String> audio = new HashMap<>();

        audio.put("title",cursor.getString(cursor.getColumnIndex(MediaStore.Audio.Media.TITLE)));
        audio.put("artist",cursor.getString(cursor.getColumnIndex(MediaStore.Audio.Media.ARTIST)));
        audio.put("album",cursor.getString(cursor.getColumnIndex(MediaStore.Audio.Media.ALBUM)));
        audio.put("path",cursor.getString(cursor.getColumnIndex(MediaStore.Audio.Media.DATA)));
        audio.put("albumImage",cursor.getString(cursor.getColumnIndex(MediaStore.Audio.Media.ALBUM_ID)));

        audioList.add(audio);
    }

    return this.audioList;
}
```

iii. AudioFileManager class를 사용해 휴대폰 내의 음성 파일들을 가지고 와 listView에 띄워준다. 이때 가지고 오는 파일의 정보는 제목(title), 가수(artist), 경로(path), 앨범이미지(albumImage) 등이 있다.

iv. 사용자 리스트에 있는 음성 파일 중, 하나를 선택하면, UI의 오른쪽 이미지처럼 앨범 이미지가 보이고 해당 음성이 재생된다. 아래는 음성 파일을 위한 listView를 제거하고 그 위치에 음성 파일의 이미지를 띄우는 코드이다.

```
//image view 생성
thumbnailImage = new ImageView(getContext());

//음성 앨범 사진 setting
final Uri artwork = Uri.parse("content://media/external/audio/albumart");
Uri albumArt = ContentUris.withAppendedId(artwork,Integer.parseInt(audioList.get(position).get("albumImage")));
Picasso.with(getContext()).load(albumArt).into(thumbnailImage);

//media view 가운데 정렬
RelativeLayout.LayoutParams params = new RelativeLayout.LayoutParams(
        ViewGroup.LayoutParams.MATCH_PARENT,
        ViewGroup.LayoutParams.MATCH_PARENT);
params.addRule(RelativeLayout.CENTER_HORIZONTAL,RelativeLayout.TRUE);
params.addRule(RelativeLayout.CENTER_VERTICAL,RelativeLayout.TRUE);

thumbnailImage.setLayoutParams(params);

mediaLayout.addView(thumbnailImage);
```

+ resetMediaView( )

- 영상->음성으로 데이터 타입을 변경했을 경우

i. 기존에 영상을 이미 선택했다면, 해당 영상을 정지하고, videoView를 제거해준다.

-음성 -> 영상으로 데이터 타입을 변경했을 경우

i. 음성 파일을 선택하기 전이라면, 음성 파일 listView를 제거한다.

ii. 음성 파일을 선택한 이후라면, 해당 음성 파일을 정지하고 앨범 이미지를 제거한다.

```
//데이터 타입 변경 시에 media view 를 reset
private void resetMediaView(){

    if(currentDataType == 0){ //video type이 선택되어 있었다면
        videoView.stopPlayback();
        mediaLayout.removeView(videoView);
        videoView = null;
    }else if(currentDataType == 1){ //audio type이 선택되어 있었다면
        if(mediaFilePath == null){ //audio list 상태
            mediaLayout.removeView(audioListView);
            audioListView = null;
        }else{ //audio file을 선택한 이후
            audioPlayer.stop();
            mediaLayout.removeView(thumbnailImage);
            audioPlayer = null;
        }
    }
```

3.3. 영상/음성 전송

- 업로드하고 싶은 영상/음성 파일을 선택했다면 상단의 화살표 버튼(업로드 버튼)을 클릭한다.

- 파일의 업로드는 uploadMediaToServer( ) 함수에서 이루어진다.

```
if(mediaFilePath == null || mediaFilePath.equals("")){
    print("Please Select an Media File");
    return;
```

- 만약 사용자가 파일을 선택하지 않았다면, 사용자에게 경고문을 띄워주고, 함수를 종료한다.

```
//Map is used to multipart the file using okhttp3.RequestBody
Map<String, RequestBody> body = new HashMap<>();

File file = new File(mediaFilePath);

  //media data
RequestBody description = RequestBody.create(MediaType.parse("text/plain"),getRequestBody());
//Parsing any Media type files
RequestBody media = RequestBody.create(MediaType.parse("*/*"),file);
MultipartBody.Part uploadFile = MultipartBody.Part.createFormData( name: "file", file.getName(),media);

body.put("description",description);
body.put("file\";filename=\""+file.getName()+
        "\";email=\""+Common.getInstance().getUserData().getEmail()+
        "\";type=\""+currentDataType,media);
```

- Retrofit2에서 지원해주는 방식을 사용하여 media 파일을 전송할 수 있다. 이 때 media 파일을 위한 RequestBody의 타입은 "*/*"으로 설정해준다.(영상/음성이 둘 다 가능하게 하기 위해서이다.)

- 서버로 미디어 파일을 보낼 경우에는 서버의 경로가 아닌 File 객체를 매개변수로 보내줘야 한다.

- 영상/음성에 대한 데이터(제목, 설명, 설정, 업로드하는 사용자의 이름 등)은 json 형식의 String으로 서버로 전송할 것이기 때문의 RequestBody의 타입은 "text/plain"이다.

```
@Override
public void onResponse(Call<PostResponse> call, Response<PostResponse> response) {
    //request가 완료되었으니 progress dialog 종료
    hideDialog();

    int responseCode = response.body().getResponseCode();
    if(responseCode == Common.getInstance().REQUEST_SUCCESS){ //upload 성공

        print("Media Upload Success");
        //해당 fragment를 종료하고 user 화면으로 이동
        ((MainActivity)parentActivity).mediaUploadSuccess();
    }
}
```

- 영상 / 음성 전송에 성공했다면(responseCode가 200(REQUEST_SUCCESS), 바로 사용자 화면으로 이동한다.(이 때, MainActivity에 있는 mediaUploadSuccess 함수를 이용한다.)

- 서버로 전송하는 영상/음성에 대한 데이터는 아래와 같다. 업로드하는 사용자의 이름은 로그인할 때 common 데이터로 저장했던 사용자 email을 사용한다.

```java
//request body 구성
private String getRequestBody(){

    JSONObject requestBody = new JSONObject();

    //media type
    try{
        requestBody.put( name: "commentable",commentable);
        requestBody.put( name: "anonymous",anonymous);
        requestBody.put( name: "messagable",messagable);
        requestBody.put( name: "downloadable",downloadable);
        requestBody.put( name: "userEmail",Common.getInstance().getUserData().getEmail());
        requestBody.put( name: "media_type", currentDataType);
        requestBody.put( name: "title",editTitle.getText());
        requestBody.put( name: "description",editDescription.getText());
    }catch (JSONException exception){
        Log.i(TAG, msg: "JSON Exception : "+exception);
    }

    Log.i(TAG, msg: "REQUEST BODY : "+requestBody.toString());
    return requestBody.toString();
}
```

3.4. 영상/음성 파일 업로드 Server Code

```javascript
//upload post
app.post('/uploadPost',upload.single('file'),function(req,res,next){
  var requestBody = JSON.parse(req.body.description);
  console.log(requestBody);
  var post = new Post({
    user_id:requestBody.userEmail,
    media_type:requestBody.media_type,
    media:req.file.filename,
    title:requestBody.title,
    description:requestBody.description,
    commentable:requestBody.commentable,
    anonymous:requestBody.anonymous,
    messagable:requestBody.messagable,
    downloadable:requestBody.downloadable
  });

  post.save(function(error, post){
    if(error) {
      console.log("Database Error : "+error);
      res.send({responseCode:404, responseBody:"Fail"});
    }
    else res.send({responseCode:200,responseBody:"Ok"});
  });
});
```

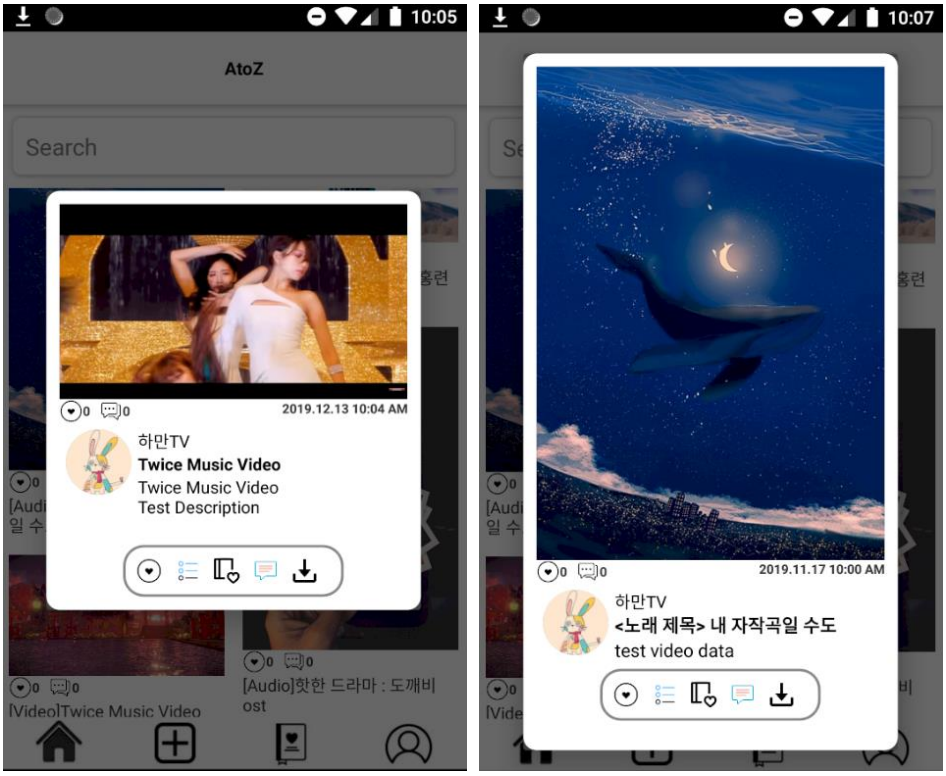
- 클라이언트로부터 전송된 파일은 자동으로 media 폴더에 저장될 수 있도록 multer을 사용해 경로를 지정해준다.

- 클라이언트로부터 전송 받은 파일의 정보(제목, 설명 등)은 Json 형식의 String으로 전달 받았기 때문에 JSON.parse( )함수를 사용해 String을 다시 json으로 convert 해주는 과정이 필요하다.

- 전송 받은 파일의 정보를 바탕으로 데이터 베이스에 해당 파일의 데이터를 저장한다.(save)

- 데이터의 저장이 성공했다면 responseCode를 200으로 setting한 response을 클라이언트에 전송한다.

4. 음성 / 영상 세부 화면



< 영상(왼쪽) / 음성(오른쪽) 세부 화면 UI >

- 영상 / 음성에 관련해서는 메인 화면(업로드 화면)에서 제공하는 기능과 같다.

- 세부화면이 준비됨과 동시에 영상 / 음성이 재생된다. 재생은 모두 Url을 통해서 이루어지며 서버로부터 데이터를 받아오는 형식으로 재생된다.

- 영상 / 음성에 대한 제목과 설명문, 업로드 날짜, 업로드한 사용자의 프로필 사진과 닉네임을 볼 수 있다.

4.1. "좋아요" 추가 / 제거

(1) 세부 화면으로 넘어오면, 사용자가 해당 게시글을 예전에 "좋아요"를 누른 적이 있는지 없는지 확인하고, 누른 적이 있을 경우 "좋아요" 버튼이 붉은 하트 모양으로 setting 된다.

- Android Code

```
//check whether I have checked 'like' this media
@GET("checkLike/{userEmail}/{mediaId}")
Call<PostResponse> checkLike(
        @Header("Authorization") String authorization,
        @Path("userEmail") String userEmail,
        @Path("mediaId") String mediaId
);
```

i. 서버로 사용자의 이메일 주소와 게시글 id 값을 param으로 넘겨, 해당 사용자가 해당 게시글을 좋아요 누른 적이 있는지 없는지 검사한다.

- Server Code

```
app.get('/checkLike/:userEmail/:mediaId',function(req,res){
  console.log(req.params.userEmail+ " / "+req.params.mediaId);
  User.find({email:req.params.userEmail,media_id_like:req.params.mediaId},function(error,result){
    console.log(result);
    if(error){
      res.send({responseCode:404,responseBody:"Error"});
    }else if(result.length != 0){ //좋아요 표시함
      res.send({responseCode:200,responseBody:"Contains"});
    }
  });
});
```

i. User 데이터 스키마의 media_id_like에는 사용자가 "좋아요"누른 게시글의 id 값이 배열로 저장되어 있다. 조건문에는 email이 사용자 이메일(userEmail)과 일치하고, 게시글의 아이디(mediaId)과 일치하는 검사하는 조건을 추가한다.

ii. 사용자가 해당 게시글의 "좋아요"를 이미 눌렀다면, responseCode를 200으로 setting하고 responseBody 값으로 Contains를 보내준다.

(2) 세부 화면에서 사용자가 "좋아요" 버튼을 클릭하게 되면, 서버로 사용자 이메일과 해당 게시글의 id를 JSON 형식의 String으로 전송하게 된다.

- Android Code

```
final HashMap<String,String> requestBody = new HashMap<>();
requestBody.put("requestBody","{\"user_id\":\""+Common.getInstance().getUserData().getEmail()+
        "\", \"media_id\":\""+currentMediaData.getId()+"\"}");

ApiConfig request = AppConfig.getRetrofit().create(ApiConfig.class);
Call<PostResponse> calling = request.addLike( authorization: "token", requestBody);
```

- Server Code

i. 사용자 email과 일치하는 document에서 media_id_like 배열에 해당 게시글의 id가 있는지 확인하는 작업을 먼저 진행한다.

```
User.find({email:requestBody.user_id,media_id_like:requestBody.media_id},function(error,result){
```

ii. media_id_like에 해당 게시글의 id가 없다면 다음과 같은 순서로 작업을 진행한다.

A. 사용자의 media_id_like에 해당 게시글의 id 추가

```
User.updateOne({email:requestBody.user_id},{$push:{media_id_like:requestBody.media_id},$inc:{like_no:1}},function(error){
```

B. 해당 게시글의 좋아요 개수 + 1

```
Post.findOneAndUpdate({_id:requestBody.media_id},{$inc:{like_no:1}},function(error,posts){
```

iii. media_id_like에 해당 게시글의 id가 있다면 다음과 같은 순서로 작업을 진행한다.

A. 사용자의 media_id_like에서 해당 게시글의 id 제거

```
User.updateOne({email:requestBody.user_id},{$pull:{media_id_like:requestBody.media_id},$inc:{like_no:-1}},function(error){
```

B. 해당 게시글의 좋아요 개수 – 1

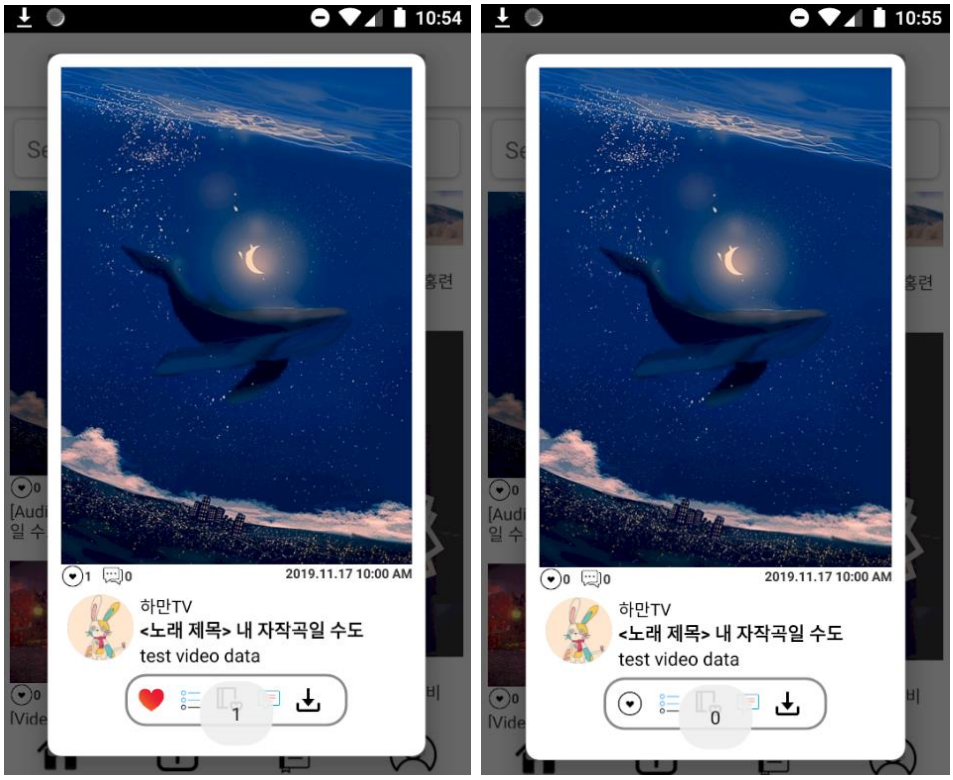
```
Post.findOneAndUpdate({_id:requestBody.media_id},{$inc:{like_no:-1}},function(error,posts){
```

- Android Code

```
@Override
public void onResponse(Call<PostResponse> call, Response<PostResponse> response) {
    Log.i(TAG, msg: "SUCCESS / "+response.body().getResponseCode());
    int responseCode = response.body().getResponseCode();

    print(response.body().getResponseBody());
    //좋아요 표시
    if(responseCode == Common.getInstance().REQUEST_SUCCESS){
        btnLike.setImageDrawable(getContext().getResources().getDrawable(R.drawable.heart_2));
        txtLikeNo.setText(response.body().getResponseBody());
    }
    //이미 좋아요 표시함
    else if(responseCode == Common.getInstance().PUT_ALREADY_EXIST){
        btnLike.setImageDrawable(getContext().getResources().getDrawable(R.drawable.heart));
        txtLikeNo.setText(response.body().getResponseBody());
    }
}
```

i. 위의 Server Code로부터 받은 response를 가지고 클라이언트의 UI를 변경해준다.

ii. "좋아요"를 추가했다면 아래 이미지의 왼쪽과 같이, "좋아요"를 취소했다면 아래 이미지의 오른쪽과 같이 변경해준다. 이 때, "좋아요"의 개수를 표시하는 textView도 변경해준다.



4.2. 앨범 추가 / 삭제

(1) Android Code

- 세부 화면에서 사용자가 앨범 버튼을 클릭했을 경우, 서버로 앨범 추가/삭제를 요청한다.

```
final HashMap<String,String> requestBody = new HashMap<>();
requestBody.put("requestBody","{\"user_id\":\""+Common.getInstance().getUserData().getEmail()+
        "\", \"media_id\":\""+currentMediaData.getId()+"\"}");

ApiConfig request = AppConfig.getRetrofit().create(ApiConfig.class);
Call<PostResponse> calling = request.addUserAlbum( authorization: "token", requestBody);
```

i. 이 때, "좋아요" 기능과 마찬가지로 사용자의 email과 해당 게시글의 id 값을 json 형식의 String으로 보내준다.

(2) Server Code

i. 사용자 email과 일치하는 document에서 album 배열에 해당 게시글의 id가 있는지 확인하는 작업을 먼저 진행한다.

```
User.find({email:requestBody.user_id,album:requestBody.media_id},function(error,result)
```

ii. album에 해당 게시글의 id가 없다면 사용자의 album 배열에 해당 게시글의 id를 추가한다.

```
User.updateOne({email:requestBody.user_id,$push:{album:requestBody.media_id}},function(error)
```

iii. album에 해당 게시글의 id가 있다면 사용자의 album 배열에서 해당 게시글의 id를 제거한다.

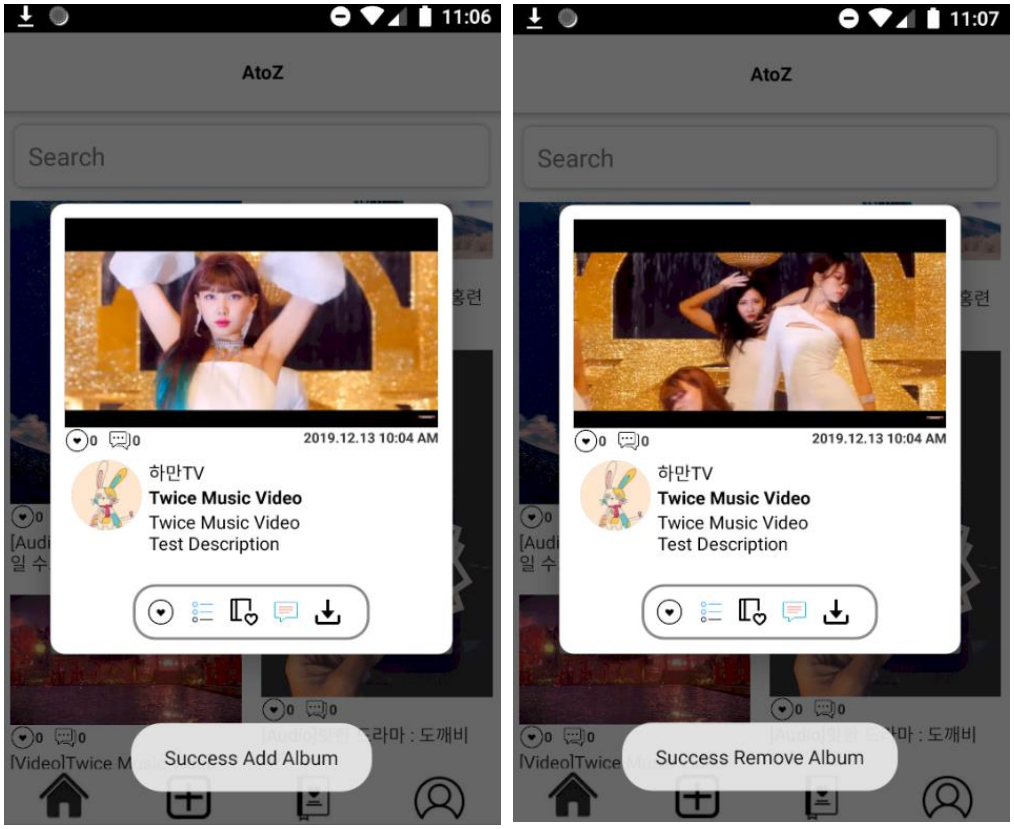
```
User.updateOne({email:requestBody.user_id},{$pull:{album:requestBody.media_id}},function(error)
```

(3) Android Code

```
calling.enqueue(new Callback<PostResponse>() {
    @Override
    public void onResponse(Call<PostResponse> call, Response<PostResponse> response) {
        Log.i(TAG, msg: "SUCCESS / "+response.body().getResponseBody());
        int responseCode = response.body().getResponseCode();

        print(response.body().getResponseBody());
    }

    @Override
    public void onFailure(Call<PostResponse> call, Throwable t) {
        Log.i(TAG, msg: "FAIL : "+t.toString());
    }
});
```

i. 위의 Server Code로부터 받은 response를 가지고 Toast로 사용자에게 결과를 알려준다.

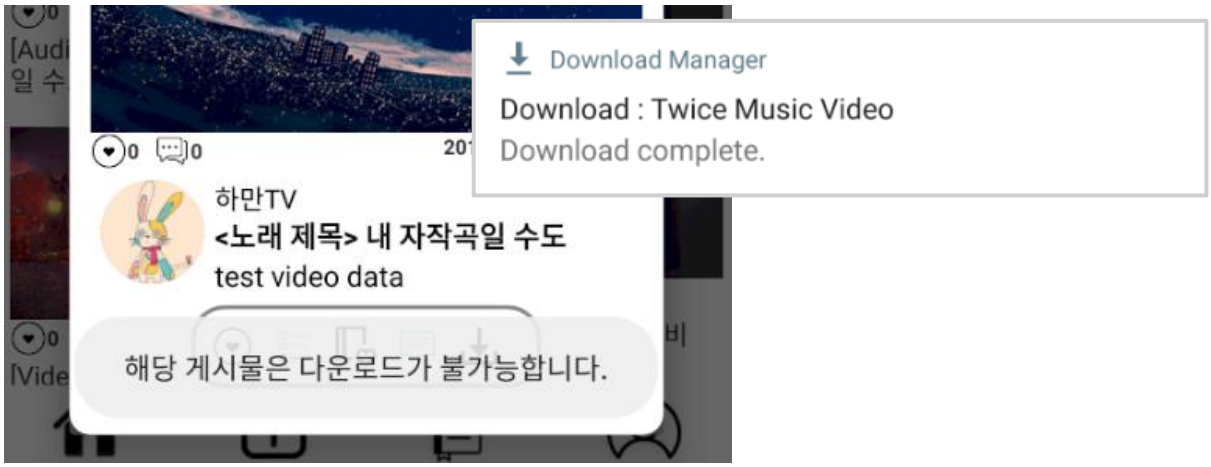
ii. 앨범에 추가했다면 아래 이미지의 왼쪽과 같이, 앨범에서 제거했다면 아래 이미지의 오른쪽과 같이 결과를 Toast로 띄워준다.



4.3. 다운로드

- 다운로드 기능은 다운로드를 허용한 게시글에서만 가능하다. 다운로드가 불가능한 게시글에 경우 Toast를 사용해 사용자에게 해당 정보를 알려준다.(아래 왼쪽 이미지)

- 다운로드가 가능한 게시글에서 다운로드 버튼을 클릭했을 경우, 다운로드 시작되고 다운로드가 완료되면 notification으로 알려준다. (아래 오른쪽 이미지)

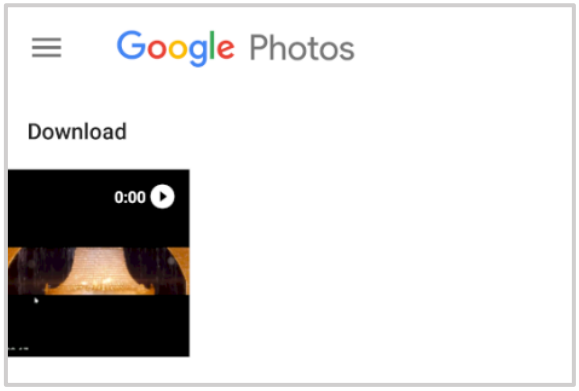


- Android Code

```
//GET url / text from currentMediaData
String mediaUrl = currentMediaData.getMultiUri();
//create download request
DownloadManager.Request request = new DownloadManager.Request(Uri.parse(AppConfig.BASE_URL+"/"+mediaUrl));
```

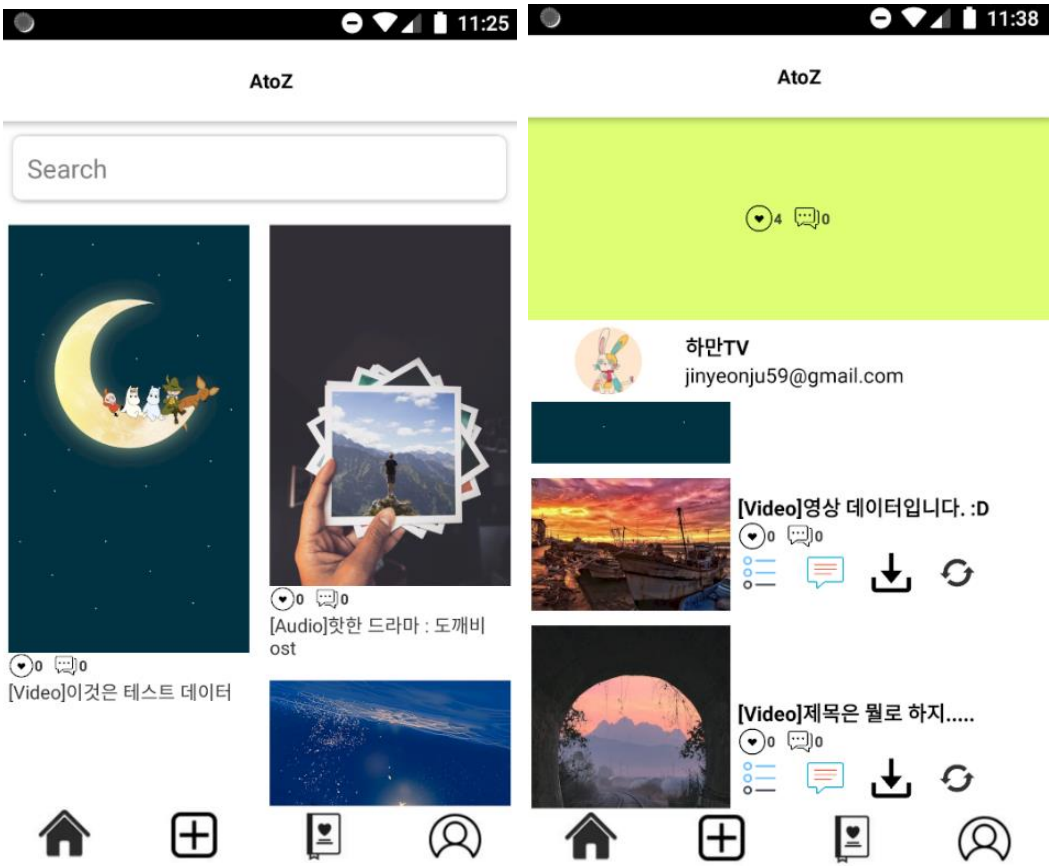
i. 다운로드는 android에서 제공해주는 DownloadManager를 통해서 이루어진다.

ii. 서버로부터 영상/음성 파일을 다운 받을 수 있도록 서버의 기본 주소를 포함한 URI를 이용한다.



- 휴대폰에서 갤러리를 확인하면 정상적으로 다운로드가 완료된 것을 확인할 수 있다.

5. 개인 앨범 화면 / 사용자 화면



< 개인 앨범 화면(왼쪽) / 사용자 화면(오른쪽) UI >

5.1 개인 앨범 화면

- 클라이언트 단의 앨범 화면은 아래 UI 이미지와 같이 클라이언트와 크게 다르지 않다. 다른 점은 전체 게시글이 아닌, 사용자가 자신의 앨범에 추가한 게시글만을 볼 수 있다는 점이다.

- Android Code

```
//get user album post list
@GET("getUserAlbum/{userId}")
Call<GetResponse> getUserAlbum(
        @Header("Authorization") String authorization,
        @Path("userId") String userEmail
);
```

i. 앨범 리스트를 가져올 때 사용자의 이메일을 param으로 전달하여 해당 사용자의 앨범 리스트를 가지고 올 수 있도록 한다.

- Server Code

i. param으로 전달 받은 사용자의 email을 사용하여 해당 사용자의 album 리스트를 가지고 온다.

```
User.find({email:req.params.userEmail},{album:1},function(error,result)
```

ii. 이 때 album : 1은 Mysql에서의 SELECT album where email = req.params.userEmail과 같다.

iii. find의 결과값은 result에 저장되어 있다. 해당 result에 저장되어 있는 album 배열의 원소(게시글의 id 값)와 일치하는 게시글 데이터들을 Post Collection에서 검색(find)한다.

```
User.find({email:req.params.userEmail},{album:1},function(error,result){
  console.log(result[0].album);
  Post.find({_id:result[0].album},function(error,posts){
    if(error){
      console.log("Get User Post Data Base Error : "+error);
      res.send({responseCode:404,responseBody:[]});
    }else{
      res.send({responseCode:200,responseBody:posts});
    }
  });
});
```

iv. 검색(find)한 결과물을 사용자에게 response 해준다. 해당 결과물을 전달 받은 클라이언트 단은 메인 화면에서와 마찬가지로 리스트 형식으로 사용자에게 보내준다.

5.2. 사용자 화면

- 사용자 화면의 게시글을 클릭하면 메인 화면에서 게시글을 길게 클릭했을 때와 마찬가지로 영상/음성 세부 화면으로 넘어간다. 기능적인 면에서는 메인 화면에서 서술한 것과 동일하다.

- 서버로 사용자의 email을 param으로 전달하여, 사용자가 업로드했던 게시글의 리스트를 response로 전달

받아 listView에 추가하여 보여준다.

- Server Code

```
app.get('/getUserPost/:userEmail',function(req,res){
  console.log(req.params.userEmail);
  Post.find({user_id:req.params.userEmail},function(error,posts){
    if(error){
      console.log("Get User Post Data Base Error : "+error);
      res.send({responseCode:404,responseBody:[]});
    }else{
      res.send({responseCode:200,responseBody:posts});
    }
  });
});
```

i. Post Collection에서 user_id가 클라이언트에서 전송된 userEmail과 동일한 게시글(Post)만 검색하여 클라이언트에게 response 해준다.

Ⅲ. 결론

1. 전체 구현 내용

- 서버 저장소에 접근하여 영상 / 음성 데이터를 받아와 클라이언트에서 재생

- 클라이언트에서 서버로 영상 / 음성 파일 업데이트

- 클라이언트에서 서버 저장소로부터 영상 / 음성 파일 다운로드

- 사용자 email에 따른 게시글 데이터 베이스 구축 / 관리

- 게시글 "좋아요", "개인 앨범" 데이터 베이스 구축 / 관리

2. 사용 기능

Client	Server
Retrofit2	Express
Picasso	mongoose
MediaPlayer / VideoView	Multer
Google Firebase	Multipart