07월 10일 학습 내용

목차

1. SurfaceView.startRecording()이 검은 화면 나온 이유에 대한 고찰 (문제점 파악, 해결방안 없음)
2. AWS API - 람다 - DynamoDB 연동 서비스에 대한 이해
3. 이전 인턴이 작성해 놓은 코드는

com.uncorkedstudios.android.view.recordablesurfaceview;

에서 제공하는

RecordableSuface 클래스의 객체이다.

근데, 현재 작성된 코드를 보면

surfaceView = findViewById(R.id.*videoView*);

로 Acitivity\_main.xml에 있는 videoView를 값으로 받았다.

하지만, uncorkedstudio에서 구현해놓은 SurfaceView에서 레코딩 부분을 찾아보게 되면

public void initRecorder(File saveToFile, int displayWidth, int displayHeight,

int desiredWidth, int desiredHeight, int orientationHint,

MediaRecorder.OnErrorListener errorListener,

MediaRecorder.OnInfoListener infoListener)

throws IOException {

MediaRecorder mediaRecorder = new MediaRecorder();

mediaRecorder.setOnInfoListener(infoListener);

mediaRecorder.setOnErrorListener(errorListener);

mediaRecorder.setVideoSource(MediaRecorder.VideoSource.*SURFACE*);

mediaRecorder.setInputSurface(mSurface);

mediaRecorder.setAudioSource(MediaRecorder.AudioSource.*MIC*);

mediaRecorder.setOutputFormat(MediaRecorder.OutputFormat.*MPEG\_4*);

mediaRecorder.setAudioEncoder(MediaRecorder.AudioEncoder.*AAC*);

mediaRecorder.setAudioSamplingRate(44100);

mediaRecorder.setAudioEncodingBitRate(96000);

mediaRecorder.setVideoEncoder(MediaRecorder.VideoEncoder.*DEFAULT*);

mediaRecorder.setVideoEncodingBitRate(12000000);

mediaRecorder.setVideoFrameRate(30);

if(desiredWidth > desiredHeight){

float desiredAspect = 1080.0f / 2280.0f;

if(desiredWidth > 2280 || desiredHeight > 1080){

float aspect = (float) desiredHeight / desiredWidth;

if(aspect > desiredAspect){

desiredHeight = 1080;

desiredWidth = (int) Math.*floor*(desiredHeight / aspect);

} else {

desiredWidth = 2280;

desiredHeight = (int) Math.*floor*(desiredWidth \* aspect);

}

}

} else {

float desiredAspect = 2280.0f / 1080.0f;

if(desiredWidth > 1080 || desiredHeight > 2280){

float aspect = (float) desiredHeight / desiredWidth;

if(aspect > desiredAspect){

desiredHeight = 2280;

desiredWidth = (int) Math.*floor*(desiredHeight / aspect);

} else {

desiredWidth = 1080;

desiredHeight = (int) Math.*floor*(desiredWidth \* aspect);

}

}

}

mDesiredHeight = desiredHeight;

mDesiredWidth = desiredWidth;

mediaRecorder.setVideoSize(mDesiredWidth, mDesiredHeight);

mediaRecorder.setOrientationHint(orientationHint);

mediaRecorder.setOutputFile(saveToFile.getPath());

mediaRecorder.prepare();

mMediaRecorder = mediaRecorder;

}

인데, 여기서 자세히 살펴봐야 할 것이

mediaRecorder.setInputSurface(mSurface);

이부분이다.

여기서 mSuface의 경우

public void doSetup() {

if (!mHasGLContext.get()) {

mSurface = MediaCodec.*createPersistentInputSurface*();

mARRenderThread = new ARRenderThread();

}

this.getHolder().addCallback(mARRenderThread);

if (getHolder().getSurface().isValid()) {

mARRenderThread.surfaceCreated(null);

}

mPaused = true;

}

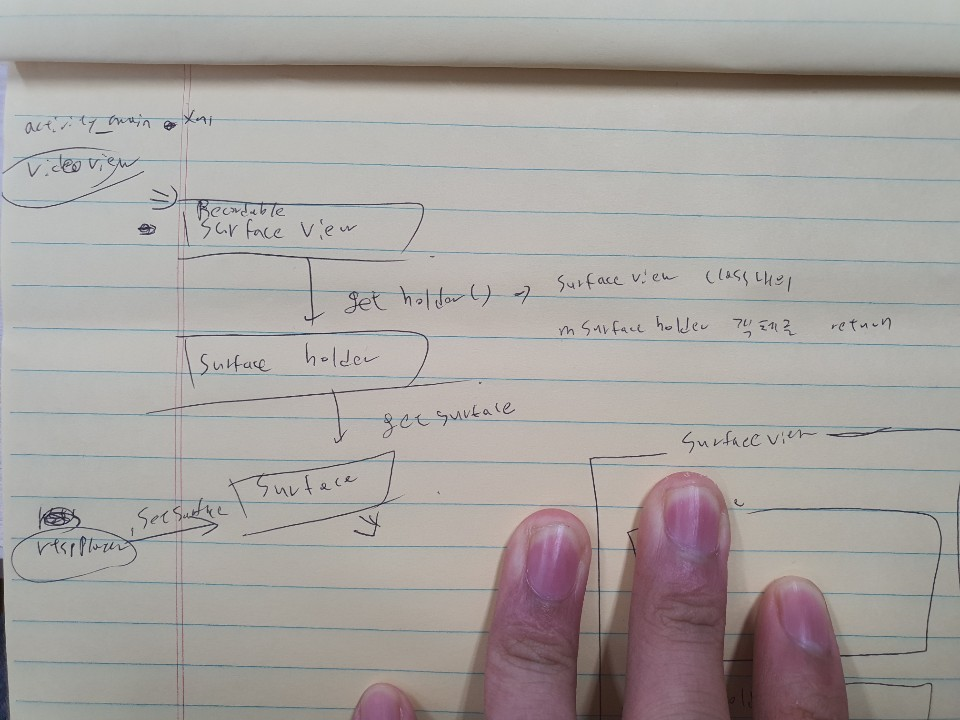
에서 MediaCodec으로 부터 새로 생성하여 Recording하게 된다.

즉, 다른 Surface를 계속 녹화를하니 검은 화면이 녹화 된 것이다.

이걸 해결하려고

RecordableSurface를 extends하여 새로 클래스를 생성하여 문제의

mediaRecorder.setInputSurface( ~ );

의 ~에 우리 코드에서 생성한 Surface (VideoView로 부터 생성된 Surface)  


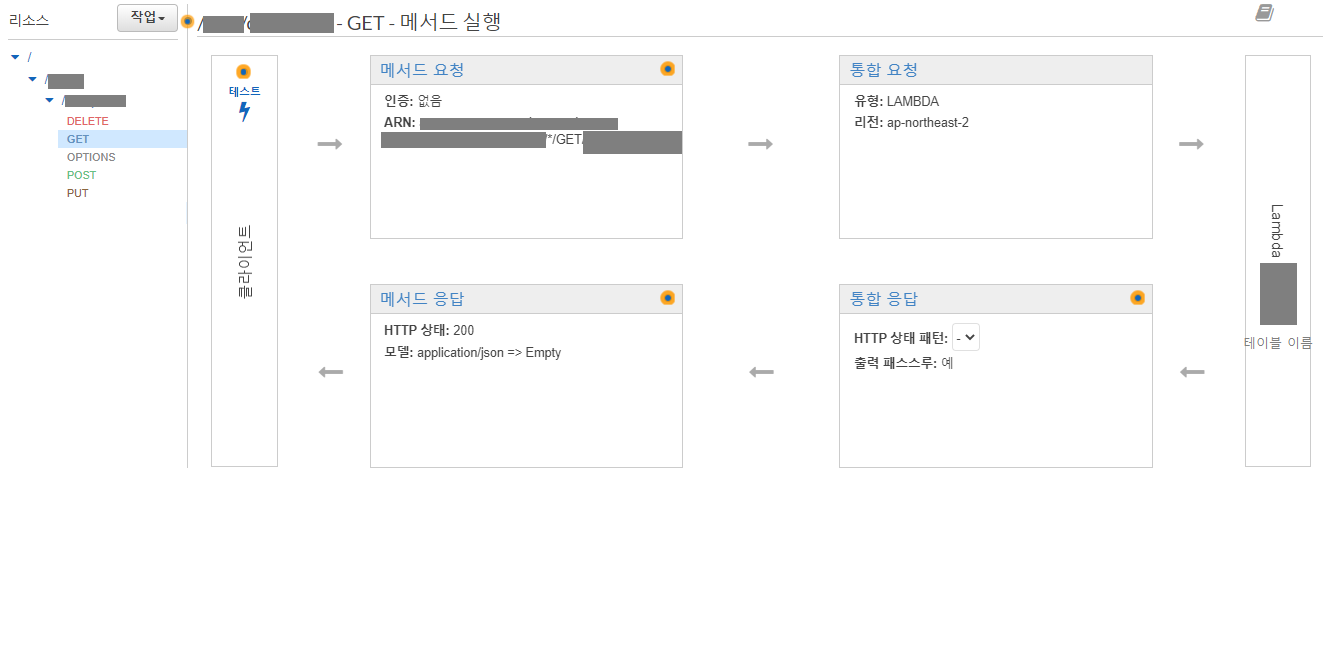
를 넣어봤으나, 스트리밍 까지 에러가 발생하여 포기하였다.

이걸 해결하려면 mediaRecorder로 다시 시도하되, setInputSurface를 신경 써야할 것 같다.

1. AWS 서비스

혼자서 찾아가면서 보려니 너무 막막하여 오늘은 선배님이 구현해 놓은 것들을 뜯어보는 형식으로 공부하였다.

1. API Gateway 에게 HTTP (Get / Post 등)형식으로 요청을 보낸다.
2. API Gateway -> Lambad로 진행되어, 람다에서 HTTP 형식 node.js 문법으로 람다의 Function내에서 DynamoDB에게 들어온 요청을 보낸다.
3. DynamoDB Table에서 해당 요청을 처리한 후 람다로 리턴, 람다는 그 결과를 HTTP 포맷에 맞춰 API GateWay를 통해 클라이언트에게 다시 보내준다.



전체 흐름은 이런 느낌이다.

0) arn:aws:execute-api:지역 : ID /\*/GET(POST) / ~ / ~가 해당 리소스의 고유번호 (ARN)

~ / ~ 는 자신이 만든 디렉토리(스토리지) 구조이고,

지역, ID는 자신이 만들 때 설정한 지역, 부여받은 ID를 입력하면 된다.

1. 클라이언트는 배포 주소를 통해 요청을 하는 듯

2) 를 좀 더 살펴보자면 람다 Function에서 다음과 같이 Dynamo DB 에게 요청을 보낸다.

const AWS = require('aws-sdk');

const dynamo = new AWS.DynamoDB.DocumentClient();

exports.handler = async (event, context) => {

let body;

body = await dynamo.scan(event).promise();

return body;

};

Handler에서 node.js로 다음과 같은 DB 요청을 보낼 수 있다.

exports.handler = (event, context, callback) => {

console.log('Received event:', JSON.stringify(event, null, 2));

const operation = event.httpMethod;

const payload = {

TableName: 'test'

}

switch (operation) {

case 'POST':

dynamo.putItem(payload, callback);

break;

case 'PATCH':

dynamo.updateItem(payload, callback);

break;

case 'DELETE':

dynamo.deleteItem(payload, callback);

break;

case 'GET':

dynamo.scan(payload, callback);

break;

default:

callback(new Error(`Unrecognized operation "${operation}"`));

}

};

이 외에도 더 있을테지만, 왠만큼은 이것들로 처리가 가능함.

* putItem : 테이블에 이거 넣어달라
* updateItem : 이걸로 바꿔달라
* deleteItem : 이거 지워줘
* getitem / scan : 이거 찾아서 리턴해달라

각각 Payload 모양이 다를텐데, 이건 클라이언트에서 맞춰서 보내줘야함.

예를들어 아래사이트에 나와있는데로 getItem의 Payload는

|  |  |
| --- | --- |
| #set($allParams = $input.params())  $input.json('$') | **var** params = {  Key: {  "~": {  S: "~"  }  TableName: "~"  }; |

이런 식으로 변환이 람다에서 이루어 지는 듯 하다.

<https://docs.aws.amazon.com/AWSJavaScriptSDK/latest/AWS/DynamoDB.html#getItem-property> 에서 더 찾아 볼 수 있다.

++++  
2-1. 람다 어플리케이션

람다 어플리케이션이란 람다 함수, 이벤트소스, 다른 리소스등을 하나로 묶어서 하나의 리소스로 관리하고 배포하기 위한 것임.

<https://docs.aws.amazon.com/lambda/latest/dg/deploying-lambda-apps.html>

정리하면

1. 클라이언트에서는 json 형식으로 데이터+요청을 던짐
2. 람다의 handler 부분에서 jjson을 변환시켜 dynamo DB에 알맞는 요청으로 변환 후 전달. (단, handler문법은 node js)
3. dynamo DB에서 해당 요청 처리해서 알맞은 내용 리턴

내일 추가 공부해야할 내용 (중요도 순)

1. 클라이언트에서의 요청 포맷 json형식
2. 람다(node.js) 에서 Dynamo DB처리에 관한 문법들

즉, json이랑 node js 공부해야함. → 이거 위의 사이트 보고 필요할때마다 찾아서 할 수 있음.

1. 람다 어플리케이션 작성 방법