Danshow
Software Technical Report

2021. 06. 10.

Capstone Design Project 41

Team 2 (Danshow)

OpenPose	Minha Kim	2015310462
OpenPose	Jinsung Kim	2016311902
Backend	Hojun Kim	2015310673
Backend	Junho Bae	2015311901
Frontend	Jeongmin Lee	2016311081

Contents

List of Figures

1. Preface

This chapter contains information about the readership, scope, objective, and structure for the technical report th

at would be used for this project.

1.1 Readership

This document is composed of 4 sections, each with its subsections. The structure for this document is found in s ection 1.4. Team 2 is the main reader for this document, but students, professors, and TAs of the Capstone Desig n course can also be one.

1.2 Scope

This document is used to provide descriptions of the Danshow detailed technique that would be used to impleme nt the choreography platform.

1.3 Objective

The main objective of this technical report is to describe the detailed technical specification for the Danshow. Th is document describes the background of the project and detailed descriptions for each part.

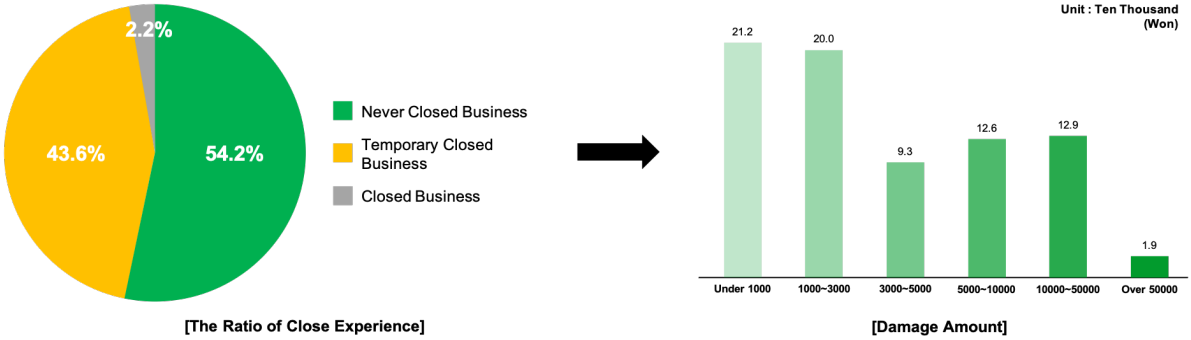
1.3 Document Structure

1. Preface: The chapter being read right now; provides information about the readership, scope, objective, and st ructure for this document
2. Introduction: Describe background of the project; also provides the solution and service architecture designed by our team to address the challenges and how the system is organized.
3. Sections: Provides detailed informations about each part: Frontend, Backend, Deep Learning
4. Conclusions: Provides remaining information and something to refer to in understanding this document.

2. Introduction

2.1. Background

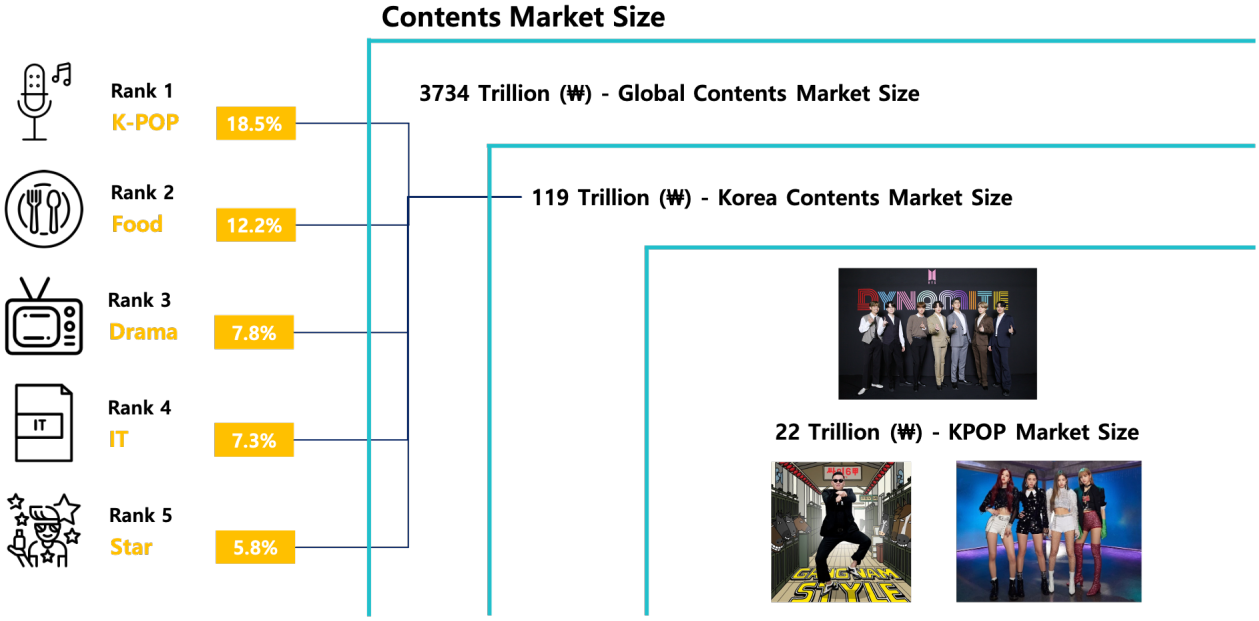
2.1.1. Closed Business due to Covid-19



[Figure 1] Data of Closed Business due to Covid-19

According to the report of status of damages in performing arts due to covid-19, 45.8% of business experiences closed business. The specific amount of damage to the companies varies from 10 million won to more than 500 million won. This is because dance academies were generating profits from offline courses before the Corona period, but they were unable to gather due to Corona, resulting in fewer students and the absence of online business models.

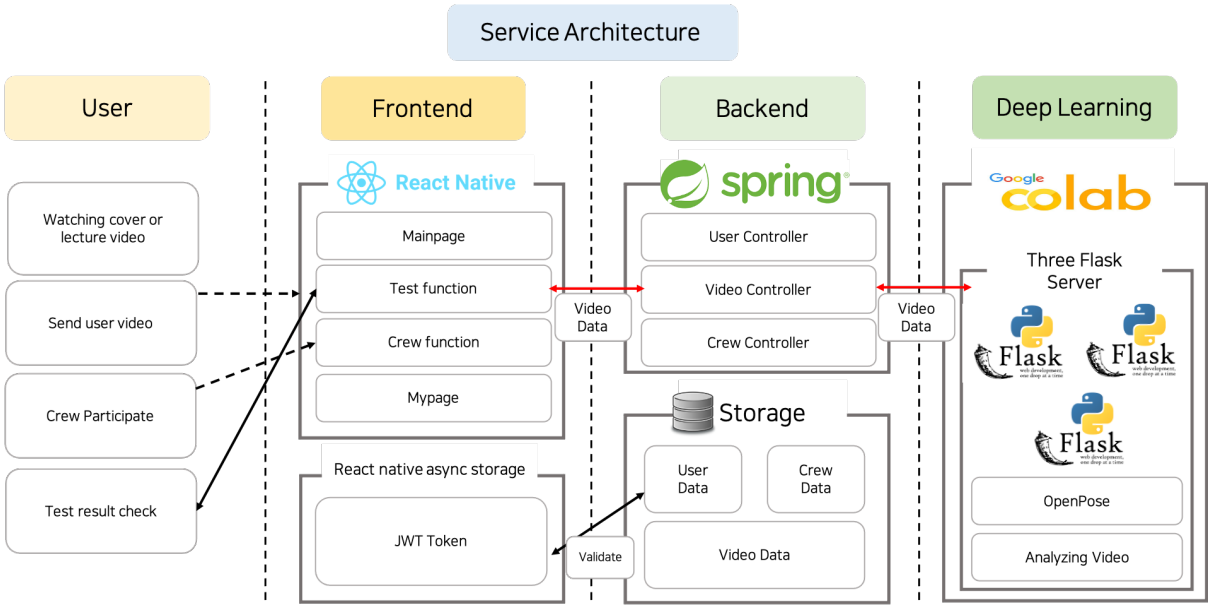
2.1.2. Growing K-POP Market



[Figure 2] Data of Growing K-POP Market

When dance academies are experiencing increasing difficulties, K-POP is, on the contrary, achieving tremendous growth. The emergence of K-POP stars such as BTS and BLACKPINK, who are popular around the world, can be a new opportunity for dance academies to generate profits. In fact, K-POP accounts for the highest proportion of the Korean content market at 18.5%, and ranks seventh in the global content market. As the number of K-POP fans has increased, there have been many foreigners who want to learn singing and dancing, and providing dance correction and courses through online platforms can be a new breakthrough for dance academies.

2.2. Solution



[Figure 3] Danshow Service Architecture

In "Danshow", users can watch videos of lectures made by instructors or watch cover videos for reference, and users can join the crew if they have a dance instructor who wants to enter. Now that we're talking about service f lows centered on core technologies, users upload their dances from Danshow to the backend server through Reac t Native. The video received from the backend server is then sent to three deep learning servers. And each deep l earning server analyzes the video and sends it back to the backend, where it combines three from the backend an d sends the video collected on the React Native.

In this way, users can see what was wrong and what was right when they danced by looking at the results of AI 's comparison and analysis of user videos and lecture videos. Thus, the user will be able to learn more about the parts that need to be corrected when practicing dance, allowing them to practice more sophisticatedly.

3. The sections

3.1. Frontend

In frontend, it is explained in two main parts. The first is the design part, which explains how the screen was designed and the results of the design. The second explains how the app was p ublished as a result of the design, with detailed code, and the final result.

3.1.1 Design

[Figure 4] Figma Logo

To design the screen first, use Figma. Figma is a vector graphics editor and prototyping tool which is primarily web-based, with additional offline features enabled by desktop applications for macOS and Windows.

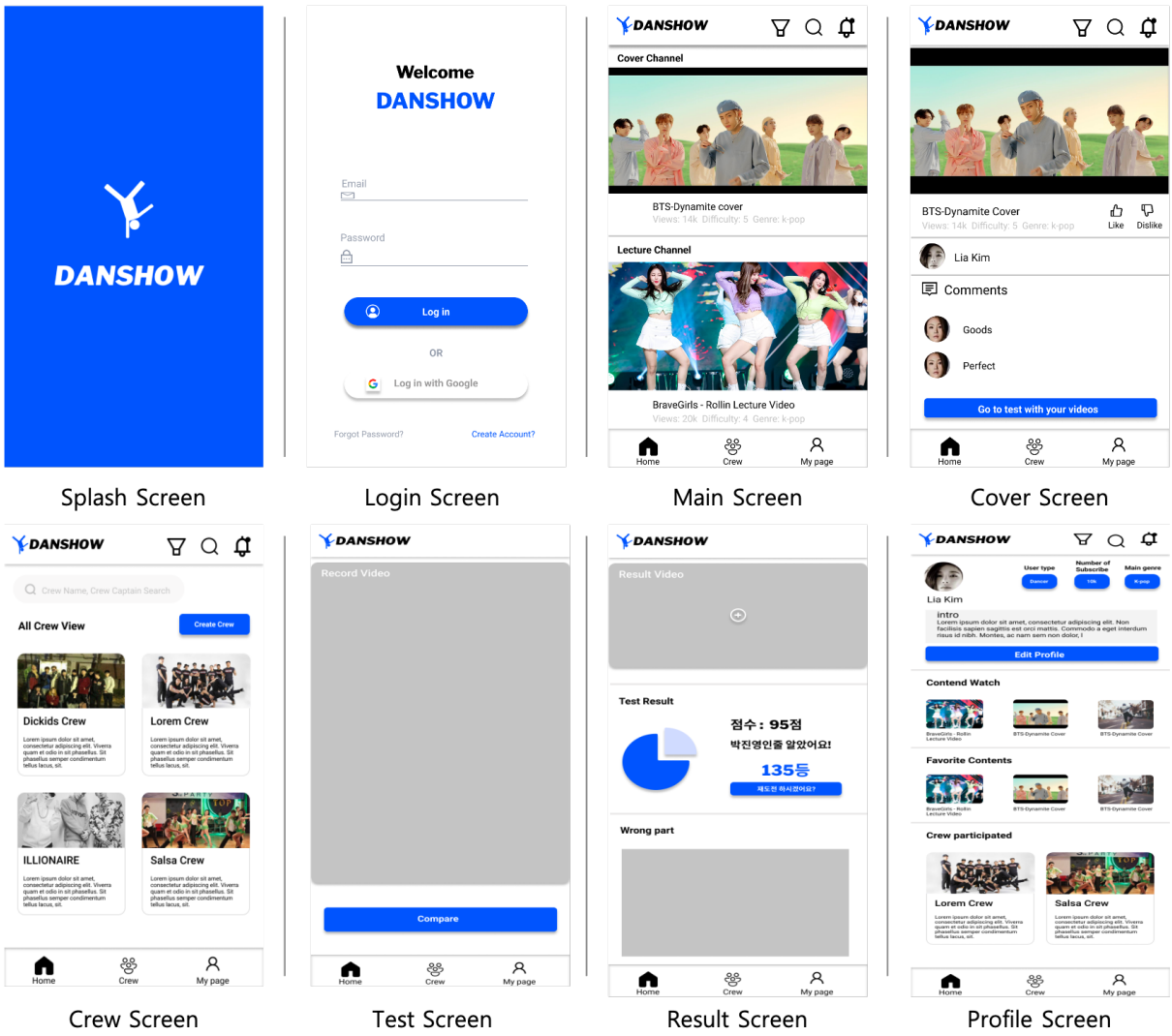
logomaster.ai

[Figure 5] Logomaster Logo

Before designing pages, our team had to create a logo, so used logomastser.ai to automatically create the logo us ing AI.

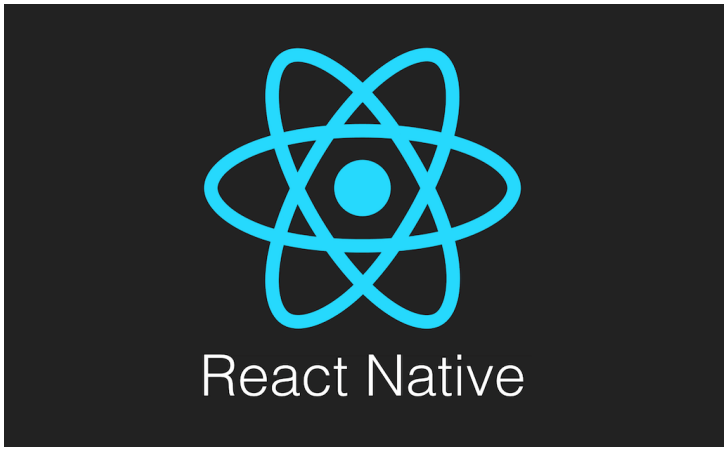
[Figure 6] Danshow Logo

After making a logo using logomaster.ai, design Splash screen to the last page.



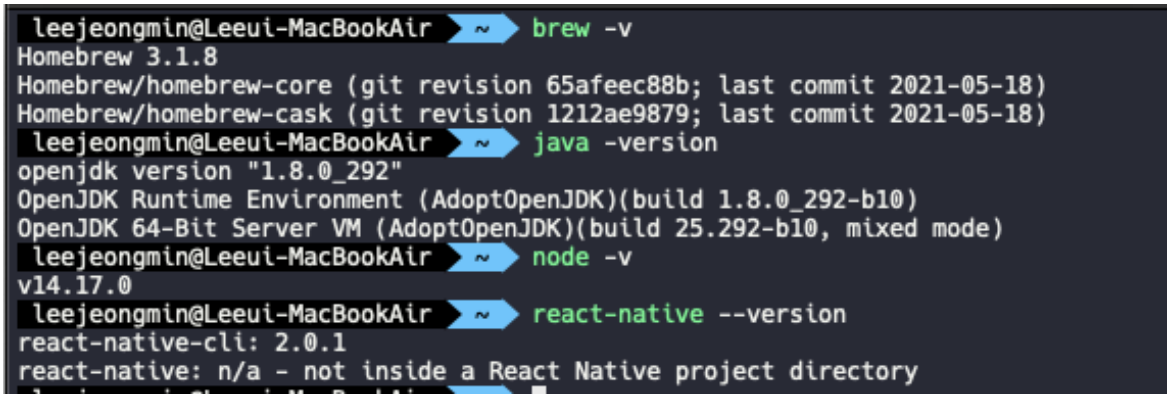
[Figure 7] UX/UI Screen

3.1.2 Publishing



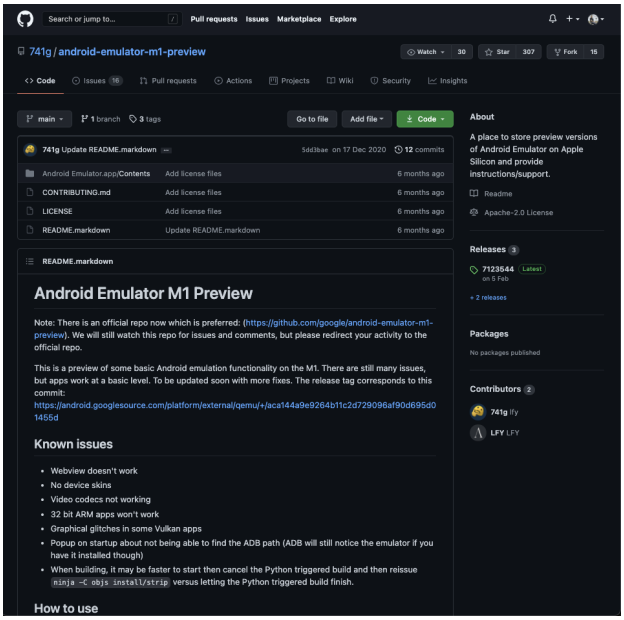
[Figure 8] React Native

After completing all page designs, our team published the application for Android using React Native. To start r eact-native in m1 mac, the initial configuration was set as follows.



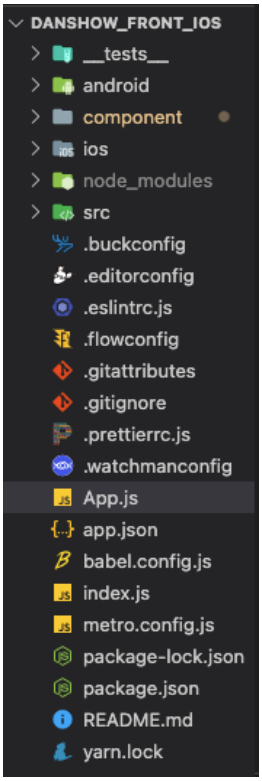
[Figure 9] React-Native Configuration on M1 Mac

Also, to test on a local screen, an Android emulator is needed.



[Figure 10] M1 Mac Android Emulator

In our react native app, the file structure is like this.



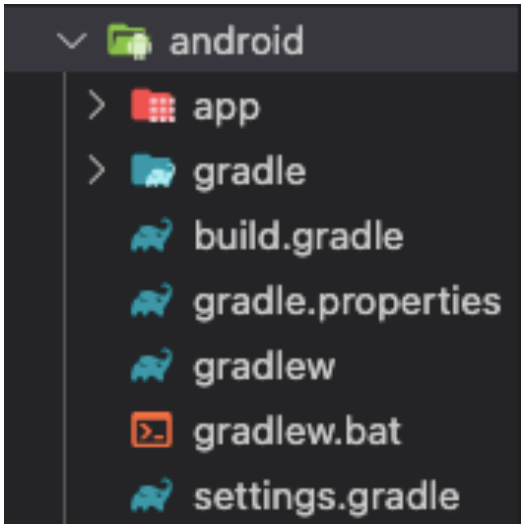
[Figure 11] File structure

The main parts are App.js, android folder, component folder. App.js shows the components to display when the app is first accessed.

```
import React from 'react';
import { NavigationContainer } from '@react-navigation/native';
import { createStackNavigator } from '@react-navigation/stack';
import { createBottomTabNavigator } from '@react-navigation/bottom-tabs';

import Login from "./component/login";
import Signup from "./component/signup";
import Main from "./component/main";

const Stack = createStackNavigator();

function App() {
  return (
    <NavigationContainer>
      <Stack.Navigator initialRouteName="Login">
        <Stack.Screen name="Login" component={Login} options={{headerShown: false}} />
        <Stack.Screen name="Signup" component={Signup} options={{headerShown: false}}/>
        <Stack.Screen name="Main" component={Main} options={{headerShown: false}}/>
      </Stack.Navigator>
    </NavigationContainer>
  );
}


export default App;
```

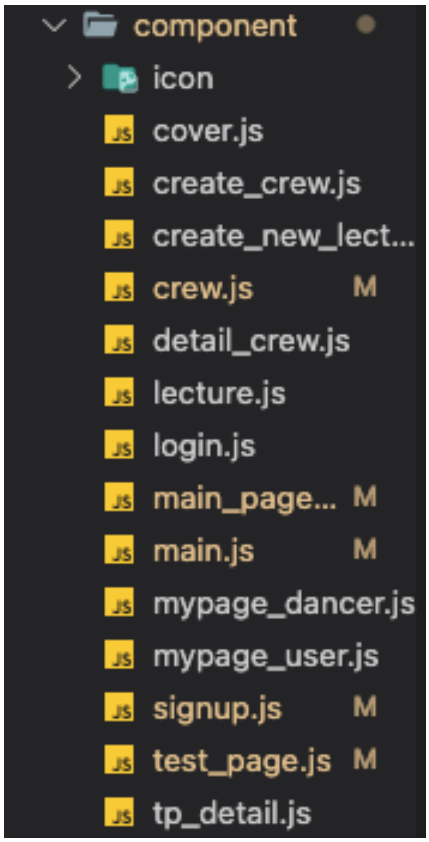
[Figure 12] App.js

As you can see in the picture above, the first screen moves in the stack structure in order of Login, Signup, and Main.



[Figure 13] Android folder

In the Android folder, there are settings files related to the Android app.



[Figure 14] Component folder

In the component folder, there are pages to display to the client. Important pages in some of our apps are as follo ws: main.js, test_page.

```
import React from "react";
import { createStackNavigator } from "@react-navigation/stack";
import {createBottomTabNavigator} from '@react-navigation/bottom-tabs';
import Mainpage from './main_page';
import Crew from './crew';
import Mypage from './mypage_dancer';
import Mypage1 from './mypage_user';
import Ionicons from "react-native-vector-icons/Ionicons";

import Lecture from "./lecture";
import Create_Crew from "./create_crew";
import Detail_Crew from "./detail_crew";
import Test_page from "./test_page";
import Tp_detail from "./tp_detail";
import Cover from "./cover";
import Create_new_lecture from "./create_new_lecture";

const HomeStack = createStackNavigator();

function HomeStackScreen() {
}

const CrewStack = createStackNavigator();

function CrewStackScreen() {
}

const MypageStack = createStackNavigator();

function MypageStackScreen() {
}

const Tab = createBottomTabNavigator();

export default function Main({ navigation }) {
}
```

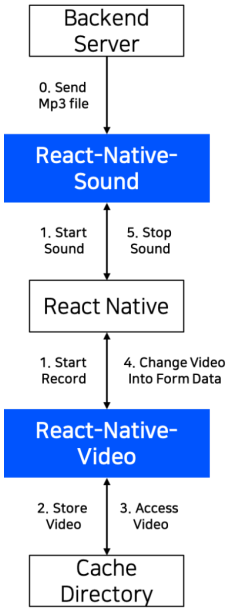
[Figure 15] main.js

In main.js, there are various navigation and controls for routing pages. When you sign in after signing up for the first time, you will be transferred to the main, which is where the main is, and you will be manipulating which sc reen to go to.

```
import React, { useEffect, useRef, useState } from 'react';
import { SafeAreaView, StyleSheet, Text, View, TouchableOpacity, Image, Linking } from 'react-native';
import { RNCamera } from "react-native-camera"
import Sound from "react-native-sound"
import AsyncStorage from '@react-native-community/async-storage';
import * as RNFS from 'react-native-fs';

const PendingView = () => (
  <View
    style={{
      flex: 1,
      backgroundColor: 'lightgreen',
      justifyContent: 'center',
      alignItems: 'center',
    }}
  >
    <Text>Waiting</Text>
  </View>
);

export default function Test_page({ navigation }) {
}

const styles = StyleSheet.create({
});
```

[Figure 16] test_page.js

In test_page.js, it records the user's test video, combines the json information needed for the video with the test video in formData format and sends it to the server via the rest API. The details are as below.




```
try {
  const data = new FormData()
  data.append("userTestVideo", {
    name: "danshow.mp4",
    type: 'video/mp4',
    uri: uri
  })
  data.append("post", {
    name: "test.json",
    type: "application/json",
    uri: 'file://' + path
  })
  console.log(uri)
  AsyncStorage.getItem('authToken', async (err, result) => {
    const authToken = result;
    console.log(authToken)
    await fetch("http://3.37.74.8:8080/api/v1/member-test/157", {
      method: "POST",
      headers: {
        'X-AUTH-TOKEN': `${authToken}`,
        'Content-Type': 'multipart/form-data',
      },
      body: data,
    })
      .then(res => {
        console.log(res)
        res.json()
      })
      .then(res => console.log(res))
      .catch(err => console.log(err))
  })
    .catch(err => console.log(err))
} catch (err) {
  console.log(err)
}
```

[Figure 17] Process of Video Record and Send File

The detailed process of recording, sending, and communicating videos to and from the backend server is as follows.

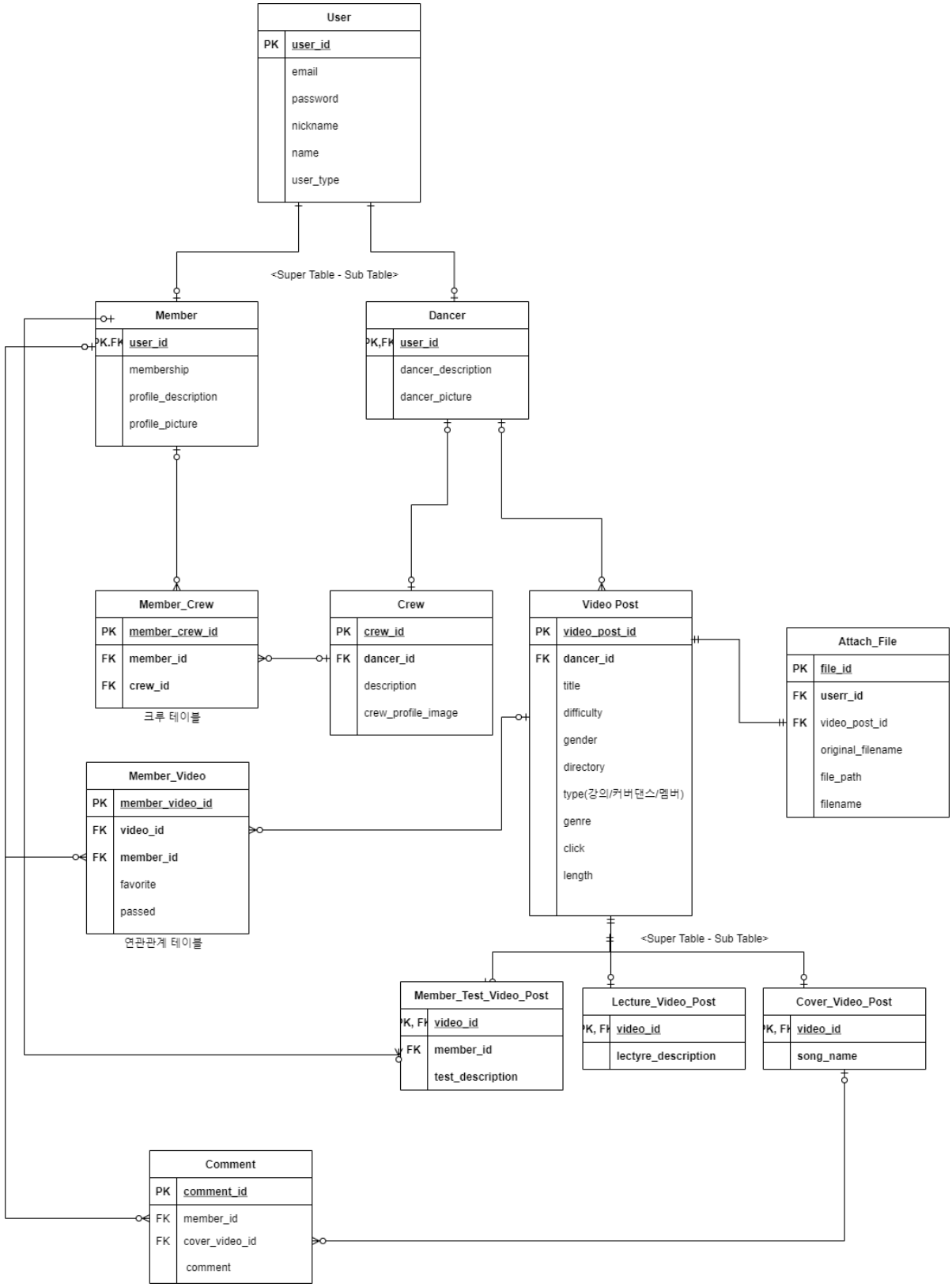
1. Import User's Token information into React-Native-Async-Storage
2. Receive extracted sound from backend server and run as React-Native-Sound
3. Take video from React-Native-Video and complete video from Cache Directory
4. Import test.json video through React-Native-fs
5. Inserts video file and test.json file into FormData
6. Sends FormData to backend server using fetch function
7. Receives video file that has been analyzed and appears on Main Screen

3.2. Backend

3.2.1 Common things

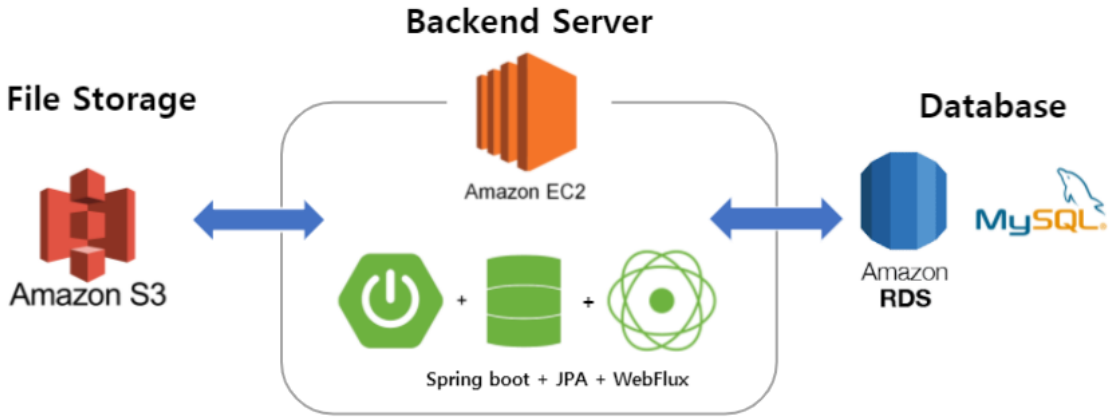
3.2.1.1 ERD(Entity-Relationship Diagram)

We have three main parts. That is 'user', 'video' and 'crew'.
This ERD will often be referred to in the following descriptions.



[Figure 18] ERD of Danshow

3.2.1.2 Server Structure



[Figure 19] Structure of Backend Server

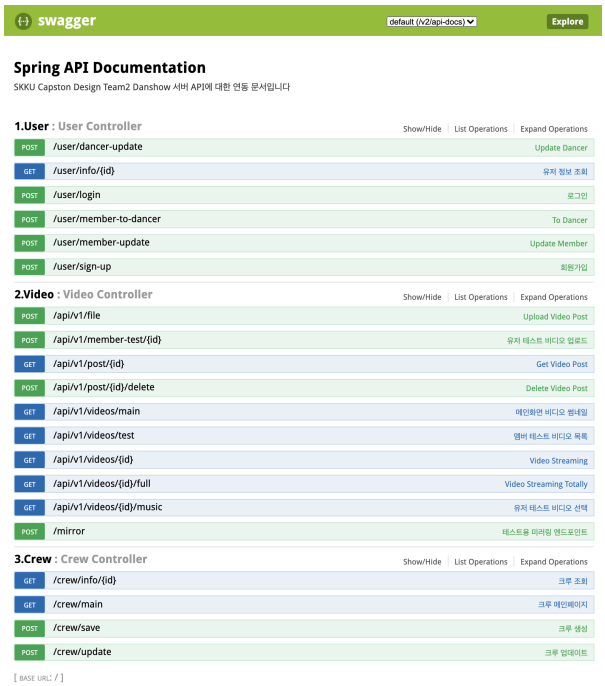
The server is implemented by Spring Framework, including Spring Data Jpa, Spring Security, Spring WebFlux, and other extra dependencies. In Particular, we used FFmpeg library to process media files, and used spring con nector of FFmpeg so that we can implement functions in Java.

Our server is hosted by AWS, EC2 free tier. The database is constructed with Amazon RDS, MySql. Also, multi media files will be saved in Amazon S3.

3.2.1.3 API Documentation

Because we needed communication with frontend about these controllers, we wrote API documents through Spr ing Swagger.

It represents the parameters required for various endpoints and also shows an example response.
Front-end developers can easily identify their requirements by referring to documents in this API.



[Figure 20] Spring Swagger

3.2.2 User

3.2.2.1 Type of user

As shown in ERD, there are two subtypes of users. It is a ‘dancer’ and a ‘member’. They have very different per sonalities. A dancer is a 'producer' who makes money by making content. A Member is a ‘consumer’ who enjoy s content by paying money. So they must have different functionality.

3.2.2.2 Registration and login

-Sign up

The complicated process of registration has been omitted. We only receive email addresses and passwords from the sign up form. When we receive that information, we encrypt the user's password with SHA-256 and random salt. This is a safe way to store user information and is also safe against rainbow table attacks.

```
/**
 * SHA-256 encrypt
 */
public class SHA256Util {
   public static String getEncrypt(String source, String salt) {
      return getEncrypt(source, salt.getBytes());
   }

   public static String getEncrypt(String source, byte[] salt) {
      /* Encrypt source with salt */
   }

   public static String generateSalt() {
      /* Generate random salt */
   }
}
```

-Log in

If we receive email addresses and passwords from clients, we check the validation of that information. Because t he passwords are encrypted by SHA-256 as we explained above, we compare the passwords by encrypting it wit h saved salt. If it is fit with encrypted passwords, the login is successfully done.

```
private boolean checkAccount(String email, String password) {
      User user = userRepository.findByEmail(email);
      if(user==null)
         return false;
      String encryptedPassword = user.getPassword();
      String salt = user.getSalt();
      return SHA256Util.getEncrypt(password,salt).equals(encryptedPassword);
   }
```

-Connection management

Basic management strategy of Spring boot is session & cookie. However this method is not used very often thes e days because it can increase the load on the server. We used the JWT , which is the solution to this problem.
When logging in is successfully done, the backend server would make a new JWT and send it to the client.
The client must include JWT in all requests except membership and login.
At each request, JWT is validated and the information contained in JWT can be used to immediately identify the logged-in user. Validation of JWT takes place through the filter of Spring Security.

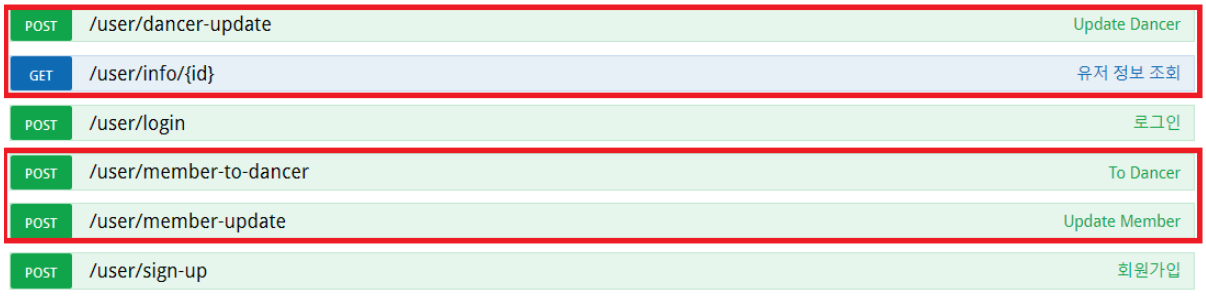
```
public class JwtAuthenticationFilter extends GenericFilterBean {
```

```
    private final TokenProvider tokenProvider;
    @Override
    public void doFilter(ServletRequest request, ServletResponse response, FilterChain chain) throws IOException, ServletException {
        // Get JWT from request header.
        String token = tokenProvider.resolveToken((HttpServletRequest) request);
        // Check validation.
        if (token != null && tokenProvider.validateToken(token)) {
            // If it is valid, get user info from token.
            Authentication authentication = tokenProvider.getAuthentication(token);
      SecurityContextHolder.getContext().setAuthentication(authentication);
        }
        chain.doFilter(request, response);
    }
}
```

Related file
java/com/danshow/danshowserver/config/auth/TokenProvider.java
java/com/danshow/danshowserver/config/auth/custom/JwtAuthenticationFilter.java

3.2.2.3 User Information

We provide endpoints that can be changed to users' information or change user type to dancers. We also provide endpoints that can look up user's information. These endpoints are used to configure the screen at the front end.

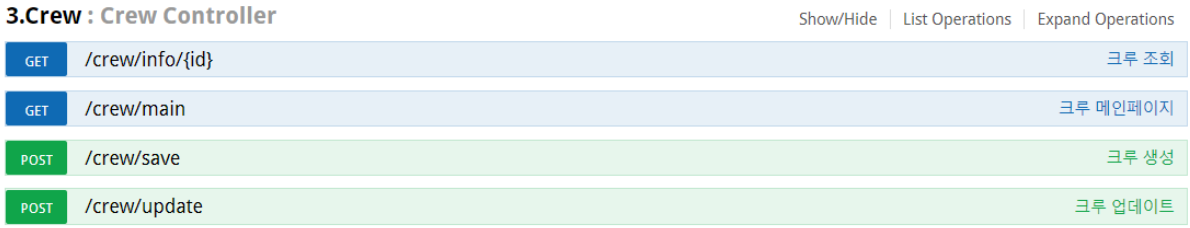


[Figure 21] Endpoints related to user information

3.2.3 Crew

Crew is a planned function, but it is a less important part because it is not directly related to the video. So we im plemented only basic functions such as creating, viewing, and joining the crew.

-Endpoints



[Figure 22] Endpoints related to crew information

From the top up, it is used for get detail page, main page, registration, and modification.

-Link to S3

The image of crew is saved in S3. So we created a S3 uploader and saved the image file in S3. S3 uploader is als o used in video functions. When we get files from client, we upload the file to S3 and save the file address to D B. So when client need image of crew, we provide the file address so that they can receive the image.

```
@Transactional
public void save(MultipartFile image, CrewSaveRequestDto crewSaveRequestDto, String email) throws IOException {

    String image_url = s3Uploader.upload(image,"image");
    crewRepository.save(Crew.builder()
            .description(crewSaveRequestDto.getDescription())
            .crew_profile_image(image_url)
            .dancer(dancerRepository.findByEmail(email)).build());
}

@Transactional
public void update(MultipartFile image, CrewSaveRequestDto crewSaveRequestDto, String email) throws IOException {

    String image_url = s3Uploader.upload(image,"image");
    Crew crew = crewRepository.findByDancer(dancerRepository.findByEmail(email));
    crew.setCrew_profile_image(image_url);
    crew.setDescription(crewSaveRequestDto.getDescription());
}
```

3.2.4 Video

Video part is one of the most important parts in our server architecture. Before the specific explanation, We nee d to take a look at how we process multimedia files. At first, we tried to process files with pure Java, But there were some issues that splitted files were not playable, and so on. So we used FFmpeg library. Originally, FFmpe g was a command line based program, and we needed an additional spring connector so that we can use FFmpeg library with Java. The environmental variables are set to “usr/local/bin/ffmpeg”.

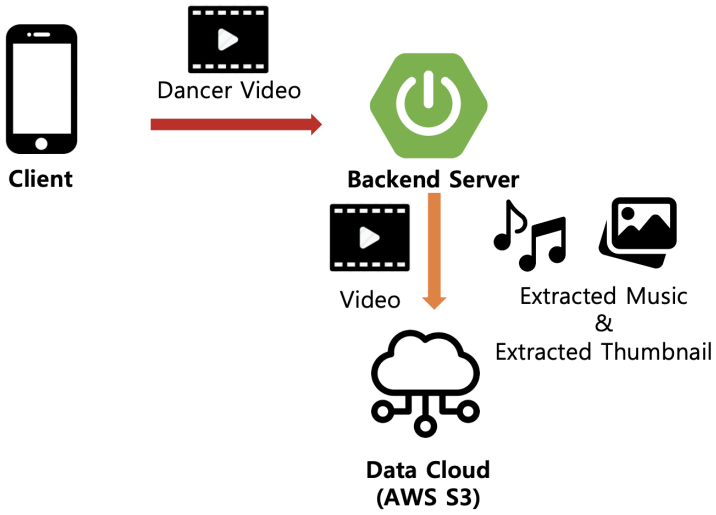
```
//build.gradle
dependencies{
implementation 'net.bramp.ffmpeg:ffmpeg:0.6.2'
}

@Slf4j
@Component
public class VideoFileUtils {

        @Value("${local.ffmpeg}")
        private String ffmpegPath;
        @Value("${local.ffprobe}")
        private String ffprobePath;

        private FFmpeg fFmpeg;

        private FFprobe fFprobe;

@PostConstruct
        public void init() {
        try {
        fFmpeg = new FFmpeg(ffmpegPath);
        Assert.isTrue(fFmpeg.isFFmpeg());

        fFprobe = new FFprobe(ffprobePath);
        Assert.isTrue(fFprobe.isFFprobe());
```

```
            log.debug("VideoFIleUtils init complete");

            }catch (Exception e) {
            log.error("VideoFileUtils init fail", e);
            }
            }
            ...
}
```

Above codes are from VideoFileUtils.java in our project, which is responsible for processing media files. With those initialized FFmpeg, a lot of methods in VideoFileUtils.java will process the multimedia files.

From now on, we will take a look at each process of processing video files in detail with codes.

3.2.4.1 Uploading cover videos of dancers.



[Figure 23] Process of uploading videos

Dancers of our service can upload their own cover videos and some information json file including genre, some descriptions, genre, difficulty and so on. If dancer uploads the video, the audio file of the requested video will be extracted and saved in a different directory of s3. The database will store the public s3 url of the audio file. Also, the thumbnail image of the video will be extracted and will be uploaded in the s3 database either. Then, the cover video will be uploaded. The stored audio file will be used in the last part of the test video upload process, and thumbnail will be used with the json file while creating 'Video Post', which will be displayed in the client.

```
1- @ApiOperation(value = "Upload Video Post",notes = "Request with Video file, Video post request object, User, Thumbnail image to create post")
2-   @PostMapping("/api/v1/file")
3- public ResponseEntity<String> fileUpload(@ApiParam(value = "Video File",required = true) @RequestPart("video") MultipartFile video,  @ApiParam(value = "Video Post Request json",required = true) @RequestPart("post") VideoPostSaveDto videoPostSaveDto,
4-   @ApiParam(value = "JWT Token", required = true) @RequestHeader(value="X-AUTH-TOKEN") String Jwt)  {
5-       String email = tokenProvider.getUserPk(Jwt);
6-       try {
7-           Long videoPostId = videoService.save(video,videoPostSaveDto,email);
8-           return new ResponseEntity<>(videoPostId.toString(), HttpStatus.OK);
9-       } catch (Exception e) {
10-           e.printStackTrace();
```

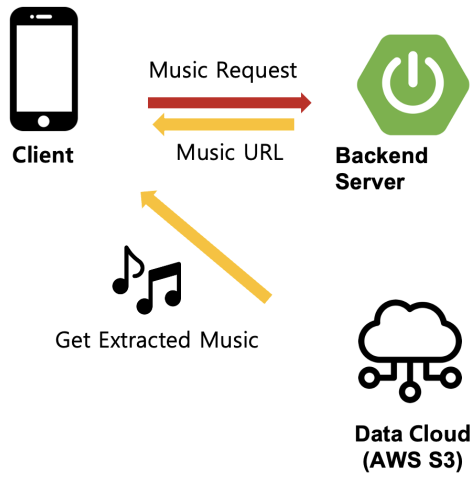
```
11-      }
12-      return new ResponseEntity<>("fail", HttpStatus.OK);
13-  }
```

Above codes are implementation of end point. In the method, videoService.save() method is called, and it will e xtract the audio files and thumbnail image, create a video post, then save it into our database and s3.

```
1-  public String extractAudio(String inputPath,String originalFileName, String outputPath) throws IOException {
2-        String originalFileNameWithoutExtension = originalFileName.substring(0,originalFileName.indexOf("."));
3-        outputPath = outputPath + "/" +originalFileNameWithoutExtension + "_audio.mp3";
4-
5-        FFmpegBuilder builder = new FFmpegBuilder()
6-              .overrideOutputFiles(true)
7-              .setInput(inputPath)
8-              .addOutput(outputPath)
9-              .addExtraArgs("-ab","128k")
10-              .addExtraArgs("-vn")              //do not extract video files.
11-              .addExtraArgs("-ar","44.1k") //sampling rate
12-              .addExtraArgs("-ac","2")          //select channel 2 of audio
13-              .addExtraArgs("-f","mp3")
14-              .done();
15-
16-       FFmpegExecutor executor = new FFmpegExecutor(fFmpeg, fFprobe);
17-       executor.createJob(builder).run();
18-       return outputPath;
19-   }
```

Above codes are extractAudio() method in VideoFileUtils.java file. As you can see, final extracted audio will be uploaded as “video_file_name_audio.mp3” format. The entire codes are available at github, please refer to the t otal saving logic in entire codes.

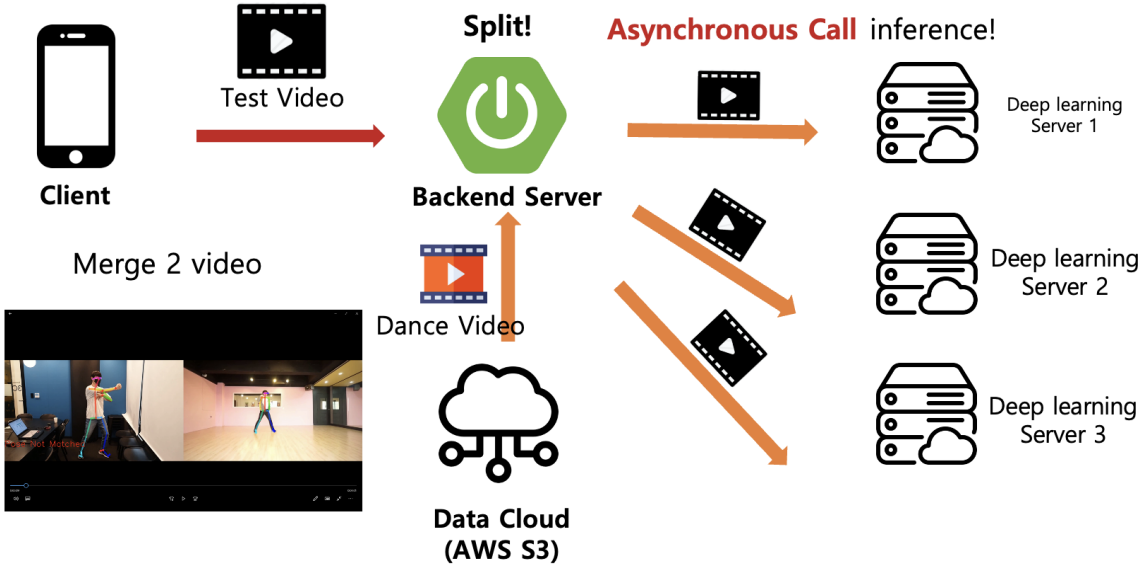
3.2.4.2 Request Music Url.



[Figure 24] Process of requesting music url

If a member wants to test his or her dance, the member will click the “test” button in the dancer video post, then will receive the audio file(especially s3 public url of the extracted audio file), so that the recorded video can be s ynchronized to the original video.

3.2.4.3 Request Member Test Video



[Figure 25] Process of requesting member test video

Now, this is the most complicated part in our implementation.
First, after recording his or her own test video, the recorded video will be sent to the server with the id value of chosen dancer video. The server will look for the original dancer video and fetch it from the s3, then integrate re quested member test video and original dancer video side-by-side as in the picture above.

```
  @ApiOperation(value = "유저 테스트 비디오 업로드",notes = "유저가 음원과 함께 녹화한 비디오를 업로드합니다
.")
   @PostMapping("/api/v1/member-test/{id}")
   public ResponseEntity<Void> uploadUserTestVideo(@ApiParam(value = "video id",required = true) @PathVariable Lo
ng id, @ApiParam(value = "test video of member") @RequestPart MultipartFile userTestVideo,
@ApiParam(value = "jwt token", required = true) @RequestHeader(value="X-AUTH-TOKEN") String Jwt) throws IOExc
eption {
      analyzeService.getAnalyzedVideo(userTestVideo,id,Jwt);
      return new ResponseEntity<>(HttpStatus.OK);
   }
```

Above codes are implementation of endpoint. In the code, getAnalyzedVideo() method is called.

```
@TimeCheck
   public String getAnalyzedVideo(MultipartFile memberTestVideo, Long id,String token) throws IOException {

      //1. merge the requested member video file and cover video of dancer side-by-side and upload to s3
      File savedIntegratedFile = videoServiceInterface.uploadMemberTestVideo(memberTestVideo, id);
            …
}
```

In the getAnalyzedVideo uploadMemberTestVideo() method is called and it will implement the functions descri bed above.

```
  @TimeCheck
```

```
public File uploadMemberTestVideo(MultipartFile memberVideo, Long id) throws IOException {
    final String localPath = System.getProperty("user.dir") + "/tmp";

    String memberVideoPath = localPath+"/"+memberVideo.getOriginalFilename();

    log.info("uploadMemberTestVideo: member-test video originalFileName : " + memberVideo.getOriginalFilename());
    log.info("uploadMemberTestVideo : member-test video created at : "+memberVideoPath);

    String originalFileNameWithoutExtension = memberVideo.getOriginalFilename().substring(0,memberVideo.getOriginalFilename().indexOf("."));

    //1. Save Member-Test video in local area.
    memberVideo.transferTo(new File(memberVideoPath));
    String s = videoFileUtils.resizeFile(localPath + "/", originalFileNameWithoutExtension);

    memberVideoPath = s;

    //2. Fetch Dancer Cover video from s3 and save in local area
    AttachFile dancerVideo = fileRepository.findById(id).orElseThrow(() -> new NoSuchElementException("no video"));
    byte[] bytes = s3Uploader.getObject(dancerVideo);
    String temporalFilePath = System.getProperty("user.dir") + "/tmp/"+dancerVideo.getOriginalFileName();
    videoFileUtils.writeToFile(temporalFilePath, bytes);

    //3. Integrate Member-Test video and Original Cover Video
    String integratedPath =
        videoFileUtils.integrateFileSideBySide(memberVideoPath, temporalFilePath,
            localPath + "/integrated_" + memberVideo.getOriginalFilename());

    log.info("uploadMemberTestVideo : user and member video integrated");

    //4. Uploads integrated video.
    String uploadedPath =
        s3Uploader.upload(integratedPath, dancerVideo.getOriginalFileName(), "video");

    AttachFile savedMemberTestVideo = AttachFile.builder()
        .filePath(uploadedPath)
        .originalFileName(memberVideo.getOriginalFilename())
        .filename("integrated_"+memberVideo.getOriginalFilename())
        .build();

    fileRepository.save(savedMemberTestVideo);

    //Delete Local Files.
    File memberFile = new File(memberVideoPath);
    memberFile.delete();

    File dancerFile = new File(temporalFilePath);
    dancerFile.delete();

    File integratedFile = new File(integratedPath);

    log.info("upload member finished. saved location : " + integratedFile.getAbsolutePath());

    return integratedFile;
}
```

In the comment 3, integrateFileSideBySide() function is called, which integrates member-test video and original cover video in one file, side-by-side. Below is the actual code.

```
public String integrateFileSideBySide(String firstVideoPath, String secondVideoPath, String outputPath) {
```

```
        FFmpegBuilder builder = new FFmpegBuilder()
                .overrideOutputFiles(true)
                .addInput(firstVideoPath)
                .addInput(secondVideoPath)
                .addOutput(outputPath)
                .addExtraArgs("-preset", "ultrafast")
                .addExtraArgs("-filter_complex", "[0:v]setpts=PTS-STARTPTS, pad=iw*2+5:ih[bg]; [1:v]setpts=PTS-STARTPTS[fg]; [bg][fg]overlay=w+10")
                .done();

        FFmpegExecutor executor = new FFmpegExecutor(fFmpeg, fFprobe);
        executor.createJob(builder).run();

        return outputPath;
    }
```

In the code, there is an option configuring “filter complex”. In this option, we set padding and overlay of file, de stination of given files so that the method produces proper file.

Second, the server will split the ‘integrated file’(in side-by-side option) into three videos

```
    @TimeCheck
    public String getAnalyzedVideo(MultipartFile memberTestVideo, Long id,String token) throws IOException {

        //1. merge the requested member video file and cover video of dancer side-by-side and upload to s3
        File savedIntegratedFile = videoServiceInterface.uploadMemberTestVideo(memberTestVideo, id);

        //2. split the integrated videos into 3 files.
        String localSavePath = System.getProperty("user.dir")+"/tmp";
        List<String> fileList =
                videoFileUtils.splitFile(savedIntegratedFile.getAbsolutePath(),
                        savedIntegratedFile.getName(),
                        localSavePath,
                        3);
```

Below is the actual code of splitFile(), which splits the video file properly. We confirm that each splitted video i s played well.

```
    @TimeCheck
    public List<String> splitFile(String inputPath, String originalFileName, String outputPath , Integer chunkNumber) throws IOException {

        //1. Get duration of the video.
        FFmpegProbeResult probeResult = fFprobe.probe(inputPath);
        Double totalDuration = probeResult.getFormat().duration;

        //2. Set the size of each spllited file.
        Double streamSize = totalDuration / chunkNumber;

        String originalFileNameWithoutExtension = originalFileName.substring(0,originalFileName.indexOf("."));
        String originalFileNameWithoutExtensionWithUUID = UUID.randomUUID().toString() + "-" + originalFileNameWithoutExtension;

        List<String> splitFileList = new ArrayList<String>(); //List to store the splited videos.

        //3. Set the start point in Double data type.
        Double startPoint = 0.0;

        //4. if output path directory does not exist, create.
```

```
        createDirectory(outputPath);

        //4. split the file into given chunks.
        for(int i = 1; i<=chunkNumber; i++) {

            String totalPath = outputPath + "/"+originalFileNameWithoutExtensionWithUUID+"_"+i+".mp4";

            FFmpegBuilder builder = new FFmpegBuilder()
                    .overrideOutputFiles(true)
                    .addInput(inputPath)
                    .addExtraArgs("-ss", String.valueOf(startPoint))
                    .addExtraArgs("-t", String.valueOf(streamSize))
                    .addOutput(totalPath)
                    .done();

            FFmpegExecutor executor = new FFmpegExecutor(fFmpeg, fFprobe);
            executor.createJob(builder, p -> {
                if(p.isEnd()) {
                    log.info("split completed processed" );
                }
            }).run();

            log.info("split done");

            //createTxt(totalPath, outputPath, originalFileNameWithoutExtensionWithUUID);

            splitFileList.add(totalPath);
            startPoint += streamSize;
        }
        return ;
    }
```

- FFMpegProbeResult contains metadata of the video file. We can get the duration of the video from 'FF MpegProbeReuslt'.
- We set the size of each splitted file by dividing duration into the chunk parameter (in our case, 3). At fi rst, we set the 'streamSize' variable as **integer**. The method also worked well. However, when re-integr ating the split files into one file, there were some 'choppy parts', exactly in the joining point of the vide os. For example, if the video file is 10 seconds, then the split file will be 3 second. Then, there are som e choppy parts in the joining parts (3,6 seconds).
- So, we set the "streamSize" into a double type, and the problem was solved.

Third, the server will send each split file to the three deep learning servers. At the beginning of AnalyzeService. java, three deep learning server urls are initialized, and WebClient is also initialized for asynchronous communi cation.

```
@Service
@RequiredArgsConstructor
@Slf4j
public class AnalyzeService {

    private static final String DL_SERVER_URL1 = "http://99e9fdcb607e.ngrok.io/one";

    private static final String DL_SERVER_URL2 = "http://9f18bc579290.ngrok.io/one";

    private static final String DL_SERVER_URL3 = "http://4f016f9688b5.ngrok.io/one";

    ExchangeStrategies exchangeStrategies =
            ExchangeStrategies.builder()
                    .codecs(configurer -> configurer.defaultCodecs().maxInMemorySize(-1))
```

```
            .build();// to unlimited memory size .build();

    WebClient webClient = WebClient.builder()
        .exchangeStrategies(exchangeStrategies)
        .build();
            …
}
```

- Each deep learning server url changes every time when the server is started.
- We set the strategy of WebClient as above, so that there would be no memory limit to communicate vid eo files properly.

```
  @TimeCheck
  public String getAnalyzedVideo(MultipartFile memberTestVideo, Long id,String token) throws IOException {

    ….

      //3-1. get three splited files.
      File firstFils = new File(fileList.get(0));
      File secondFile = new File(fileList.get(1));
      File thirdFile = new File(fileList.get(2));

      String firstFilePath = localSavePath+"/01_"+memberTestVideo.getOriginalFilename();
      String secondFilePath = localSavePath+"/02_"+memberTestVideo.getOriginalFilename();
      String thirdFilePath = localSavePath +"/03_" + memberTestVideo.getOriginalFilename();

      String originalFileNameWithoutExtension =
            memberTestVideo.getOriginalFilename().substring(0,memberTestVideo.getOriginalFilename().indexOf("."));

      //3-2. send each splited files to the deep learning server 1,2,3 in asynchronous way and get response.
      Tuple3<byte[], byte[], byte[]> fetchVideos = fetchVideos(Files.readAllBytes(firstFils.toPath()),
            Files.readAllBytes(secondFile.toPath()), Files.readAllBytes(thirdFile.toPath()), token);

      //3-3. write byte array to files
      videoFileUtils.writeToFile(firstFilePath, fetchVideos.getT1());
      videoFileUtils.writeToFile(secondFilePath, fetchVideos.getT2());
      videoFileUtils.writeToFile(thirdFilePath, fetchVideos.getT3());

      //4. create the txt file which contains the absolute path of responsed files, so that ffmpeg can integrate those files.
      videoFileUtils.createTxt(firstFilePath,localSavePath,originalFileNameWithoutExtension);
      videoFileUtils.createTxt(secondFilePath,localSavePath,originalFileNameWithoutExtension);
      videoFileUtils.createTxt(thirdFilePath,localSavePath,originalFileNameWithoutExtension);

      firstFils.delete();
      secondFile.delete();
      thirdFile.delete();

            ...

}
```

- Above code is the total process of communication between three deep learning servers.
- In the comment 3-2, fetchVideos() method is called and receives value as Tuple<byte[], byte[]. byte[]>. Each byte array stands for the video file from the deep learning server.
- The following codes are the detailed implementation of fetchVideos() method.
- In the comment 4, the server creates the txt file including absolute paths of each analyzed video from th e deep learning server, so that FFmpeg can integrate files in the next step.

```
public Mono<byte[]> getFirstFile(byte[] bytes, String token) throws IOException {

    return webClient
            .post()
            .uri(DL_SERVER_URL1)
            .header("X-AUTH-TOKEN",token)
            .contentType(MediaType.APPLICATION_OCTET_STREAM)
            .bodyValue(bytes)
            .accept(MediaType.APPLICATION_OCTET_STREAM)
            .retrieve()
            .bodyToMono(byte[].class)
            .subscribeOn(Schedulers.parallel())
            .map(x -> {
                log.info("async method call : getThird method called");
                return x;
            });
}

public Tuple3<byte[], byte[], byte[]> fetchVideos(byte[] firstFile, byte[] secondFile, byte[] thirdFile, String token) throw
s IOException {
    return Mono.zip(getFirstFile(firstFile,token), getSecondFile(secondFile,token),getThirdFile(thirdFile,token))
            .block();
}
```

As you can see, the fetchVideos() method returns the Mono.zip() function, which enables calling multiple async hronous methods. In our case, three methods are called. In the last line, the block() function is called, so that afte r calling three asynchronous functions and getting all the results, the process would be synchronized with the ne xt code.

In the getFirstFile() method, the server can create http communication to a deep learning server in the post meth od, with byte arrays, which stands for the split files from above process. The subscribeOn() option is set to Sche dulers.parallel(), so that process can be done in parallel threads.

Fourth, integrate three received video files into one file.

```
@TimeCheck
public String getAnalyzedVideo(MultipartFile memberTestVideo, Long id,String token) throws IOException {
        …
    //5. integrate three files into one file.
    File analyzedFile = new File(videoFileUtils.integrateFiles(localSavePath,originalFileNameWithoutExtension));
        …
```

In the above code, the integrateFiles() method is called with the given parameter localSavePath. The localSaveP ath is the local path where video is saved. In the comment 4, you can see that text file including absolute paths o f three analyzed videos received from deep learning servers. The integrateFiles() method will find the text file in a given local save path, and start the integrating function.

```
@TimeCheck
public String integrateFiles(String inputPath,String originalFileName) throws IOException {

    String fileList = inputPath + "/" + originalFileName+".txt";

    String outputPath = inputPath + "/" + originalFileName + "_final_ver.mp4";

    FFmpegBuilder builder = new FFmpegBuilder()
            .overrideOutputFiles(true)
            .addInput(fileList)
            .addExtraArgs("-f","concat")
            .addExtraArgs("-safe", "0")
```

```
            .addOutput(outputPath)
            .done();

    FFmpegExecutor executor = new FFmpegExecutor(fFmpeg, fFprobe);
    executor.createJob(builder).run();

    log.info("final integrate phase complete, output path  : " + outputPath);
    return outputPath;

  }
```

- The FFmpeg needs a text file including the absolute paths of target videos.
- Therefore, we created a writing text file function above.

Fifth, merge videos with audio file and create video post, save it in s3 and database

The received file from the deep learning server has no audio although the requested video has audio file. Theref ore, integrated video also does not have audio. Thus, we have to merge integrated videos with original audio file s.

```
  @TimeCheck
   public String getAnalyzedVideo(MultipartFile memberTestVideo, Long id,String token) throws IOException {
            ...
    //6. merge integrated files with original audio file.
    byte[] obj = s3Uploader.getObject(videoPost.getMusicPath());

    videoFileUtils.writeToFile(System.getProperty("user.dir")+"/tmp/audio.mp3", obj);
    videoFileUtils.integrateAudio(analyzedFile.getAbsolutePath(),audioPath, outputPath);
    //7. uploads to the s3
    File finalFile = new File(System.getProperty("user.dir")+"/tmp/output.mp4");
    String videoPath = s3Uploader.upload(finalFile.getAbsolutePath()
            ,analyzedFile.getName(),"video");

    //8. save in database as file and create video post.
    AttachFile savedVideo = AttachFile.builder()
            .filename(originalFileNameWithoutExtension+"_"+new Date())
            .filePath(videoPath)
            .originalFileName(originalFileNameWithoutExtension)
            .build();

    VideoPostSaveDto videoPostSaveDto = VideoPostSaveDto.of(videoPost,ownerMember);

    MemberTestVideoPost memberTestVideoPost = new MemberTestVideoPost(videoPostSaveDto,
            ownerMember,savedVideo,videoPost.getImage(),null);
    videoPostRepository.save(memberTestVideoPost);

    log.info("video path : " + videoPath);
    return videoPath;
}
```

- The server fetches the audio file from s3, and writes it to the file.
- Then, call integrateAudio() function to merge video and audio.
- Finally, create the VideoPost entity and save final video to the s3.
- Below is the actual code of integrateAudio() function.

```
    public String integrateAudio(String videoPath, String audioPath, String outputPath) throws IOException{
```

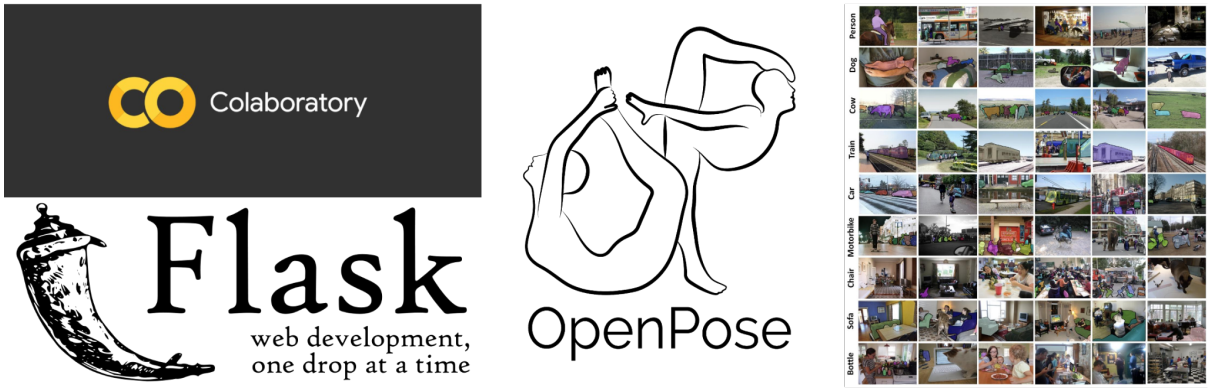
```
    FFmpegBuilder builder = new FFmpegBuilder()
            .overrideOutputFiles(true)
            .addInput(videoPath)
            .addInput(audioPath)
            .addOutput(outputPath)
            .addExtraArgs("-map","0:v")
            .addExtraArgs("-map","1:a")
            .addExtraArgs("-c:v","copy")
            .done();


    FFmpegExecutor executor = new FFmpegExecutor(fFmpeg, fFprobe);
    executor.createJob(builder).run();

    log.info("integrated audio completed : " + outputPath);

    return outputPath;
}
```

- There are add options above. 0 means video, 1 means audio, and last option means coping audio to the video files.

3.3. Deep Learning Server

3.3.1 Development Details

Development tools used to configure deep learning servers are as follows.



[Figure 26] Deep-Learning Development Tool

3.3.1.1 Google Colab

Openpose, a deep learning model for extracting keypoints from the input image or video, requires GPU for an acceptable inference speed. Google Pro provides us with T4 or P100 GPU for 24 hours of runtime, along with bigger RAM of 25.51GB. Colab provides a python environment with various pre-installed python packages, making it easier to build the deployment servers. So, we decided to deploy our deep learning servers in google colab.

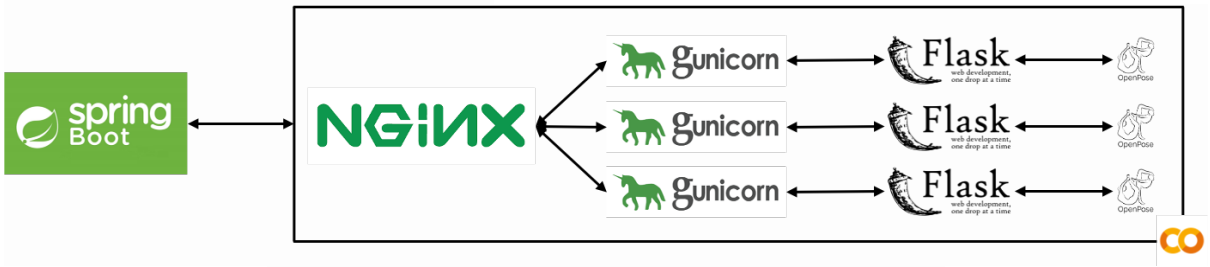
3.3.1.2 Openpose

The step of keypoint extraction was conducted by a pretrained openpose model provided in *https://github.com/CMU-Perceptual-Computing-Lab/openpose*. The pose estimation model could be trained from scratch to create a lighter, and more optimized model. However, given the limitation of GPU, we concluded that training the openpose model with the COCO dataset(250,000 people with keypoint annotations) would be extremely difficult. Instead, we thought about generating our own dataset by capturing a bunch of choreography videos provided in youtube, and annotating the keypoints by ourselves. However, the crawling and annotation steps would take a considerable amount of time, and the insufficiency of the training dataset could result in overfitting and low generalization performance on real data. Thus, we decided to use the pretrained model for keypoint extraction.

3.3.1.3 Flask Server

To communicate between deep learning servers and back-end servers in a Colab environment, we used Flask, a python-based micro web framework. We used Flask to implement a simple API server to communicate with the back-end Spring server. Here, we used Flask-ngrok which is a tunneling program that connects the outer networ k with the local server. For the free service of ngrok, the sub domain name was randomly generated each time th e ngrok was re-executed.

3.3.2 Communication with Backend Server



[Figure 27] Communicating Structure of Deep-Learning Server

```
app = Flask(__name__)
run_with_ngrok(app)

@app.route('/one', methods = ['GET', 'POST'])
def get_video():
  if request.method == 'POST':
    webByte = request.get_data()
    f = open("sample.mp4", "wb")
    f.write(webByte)
    f.close()
    final_function("/content/sample.mp4")
    return open('/content/openpose/videos/output.mp4', "rb").read()

if __name__ == "__main__":
    app.run()
```

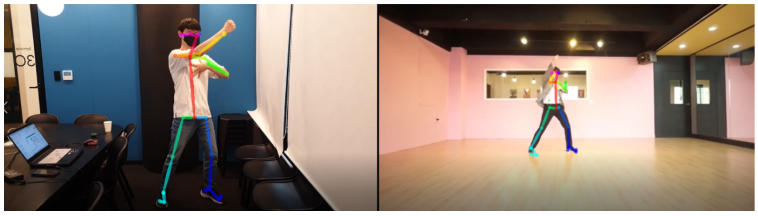
[Figure 28] Flask server

Using Colab Pro, we created three deep learning servers each containing a openpose model for pose estimation a nd similarity calculation. The figure on the left shows our implemented Flask server which communicates with t he back-end server. The user and the dancer videos are first merged side by side then splitted into short videos. Here, each short input video in byte format is received from the back-end server, and is stored in the Colab local directory(/content/sample.mp4). When the server finishes the video analysis using the final_function, the output video(/content/openpose/videos/output.mp4) is returned to the back-end server in byte format.

3.3.3 Keypoint Extraction

Person Detected		
	x	y
Nose	490	81
Neck	473	133
RShoulder	435	130
RElbow	496	125
RWrist	560	75
LShoulder	510	136
LElbow	566	177
LWrist	519	151
MidHip	484	287
RHip	455	287
RKnee	429	386
RAnkle	406	476
LHip	513	287
LKnee	528	380
LAnkle	522	467
REye	476	75
LEye	496	75
REar	450	78
LEar	0	0
LBigToe	545	496
LSmallToe	551	487
LHeel	511	473
RBigToe	409	502
RSmallToe	397	499
RHeel	406	479

Person Detected		
	x	y
Nose	1401	174
Neck	1395	197
RShoulder	1381	197
RElbow	1381	197
RWrist	1392	174
LShoulder	1404	197
LElbow	1421	218
LWrist	1421	197
MidHip	1398	264
RHip	1381	261
RKnee	1360	313
RAnkle	1334	363
LHip	1404	261
LKnee	1427	310
LAnkle	1430	360
REye	1398	171
LEye	1404	174
REar	1386	174
LEar	1404	174
LBigToe	1447	374
LSmallToe	1447	368
LHeel	1427	368
RBigToe	1349	383
RSmallToe	1334	380
RHeel	1331	365



[Figure 29] Extracted keypoints and the corresponding input image

The main tasks of our deep learning model can be separated into keypoint extraction and pose similarity calculat ion. First, our openpose model extracts keypoints from the input image. The keypoint is extracted as a list and it consists of elements with the same number of the people present in the video. Each list contains 25 different bod y part coordinates of each person including nose, neck, etc.

```
# datum 초기화 후 keypoint extraction 진행
datum = op.Datum()
datum.cvInputData = frame
opWrapper.emplaceAndPop(op.VectorDatum([datum]))
raw_keypoints = datum.poseKeypoints
Keypoints_list = getKeypointsList(raw_keypoints)
```

```
def getKeypointsList(raw_keypoints):
    temp_keypoints = []
    output = []
    body_index = op.getPoseBodyPartMapping(op.BODY_25)
    if (raw_keypoints is not None):
      if (len(raw_keypoints) > 0):
          for person in raw_keypoints:
              person_keypoints = {}
              for i in range(len(person)):
                  person_keypoints[body_index[i]]= [int(person[i][0]),int(person[i][1])]
              temp_keypoints.append(person_keypoints)
    for person in temp_keypoints:
      person = list(person.values())
      person = [(None,None) if x==[0,0] else tuple(x) for x in person]
      output.append(person)
    return output
```

[Figure 30] Keypoint extraction code

By using the pyopenpose library, we can successfully retrieve the extracted keypoint data from the op::Datum cl ass using standard python and numpy constructs. When the datum.cv2inputData gets a frame, captured from the input video, it stores raw keypoint values to datum.poseKeypoints. The raw keypoint is a list containing multipl e lists each indicating the extracted keypoint of each person in the video. Each list for an individual consists of s everal lists of size 3, each containing x coordinate, y coordinate, and confidence for each body part. The raw key point is converted to Keypoints_list by getKeypointsList function below, to be further utilized in similarity calcu lation. The format of Keypoints_list is as follows:

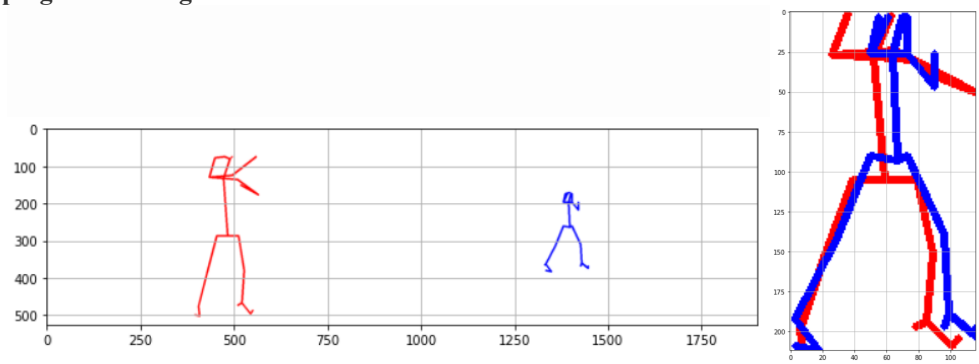
```
[[(1226, 232), (1232, 291), …], [(716, 282), (695, 337), …]]
# [[person1; (x_neck, y_neck), (x_nose, y_nose), …], [person2; (x_neck,
 y_neck), (x_nose, y_nose), …]]
```

3.3.4 Pose Similarity Calculation

After extracting both the user and the dancer keypoints, the degree of similarity between them should be calculated to yield a result on whether the pose has matched or not. However, before calculating similarity, processing jobs on the keypoint values should be executed first.

3.3.4.1 Processing

(a) Cropping and Scaling



[Figure 31] Before & after scale and crop

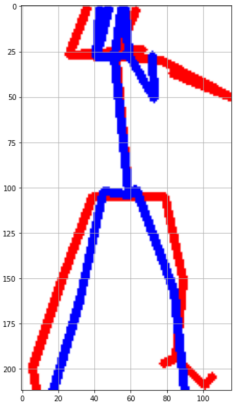
The scale of the user and the dancer video can be different, leading to inaccurate calculation results. For instance, in the figure above, the user's keypoint(left) scales larger than the dancer's(right) and for a fair comparison between every keypoints, scaling and cropping jobs should be conducted first.

```
def scaling_cropping(pose1,pose2):
    # pd.DataFrame의 자료형으로 변환
    pose1_temp = pd.DataFrame(pose1, columns=['x1','y1'])
    pose2_temp = pd.DataFrame(pose2, columns=['x2','y2'])
    pose_temp = pd.concat([pose1_temp, pose2_temp],axis = 1)
    # cropping
    pose_temp['x1'] = pose_temp['x1'] - pose_temp['x1'].min()
    pose_temp['y1'] = pose_temp['y1'] - pose_temp['y1'].min()
    pose_temp['x2'] = pose_temp['x2'] - pose_temp['x2'].min()
    pose_temp['y2'] = pose_temp['y2'] - pose_temp['y2'].min()
    # scaling
    x_scalefactor = pose_temp['x2'].max()/pose_temp['x1'].max()
    y_scalefactor = pose_temp['y2'].max()/pose_temp['y1'].max()
    pose_temp['x1'] = pose_temp['x1']*x_scalefactor
    pose_temp['y1'] = pose_temp['y1']*y_scalefactor

    pose1_temp = pose_temp[['x1','y1']]
    pose2_temp = pose_temp[['x2','y2']]
    pose1_processed = [ tuple(map(int, x)) if not np.isnan(x).all()  else (None,None) for x in pose1_temp.values.tolist()]
    pose2_processed = [ tuple(map(int, x)) if not np.isnan(x).all()  else (None,None) for x in pose2_temp.values.tolist()]
    return pose1_processed, pose2_processed
```

[Figure 32] Scaling and Cropping code

Here, after cropping the keypoint values based on their minimum, the scales of the keypoints are matched by applying appropriate scale factors for each person. This code results in the rightmost figure, where two person's keypoints overlap with each other in the same scale. It lets us compute the similarity in a higher precision.

(b) Affine Transformation



[Figure 33] After affine transformation

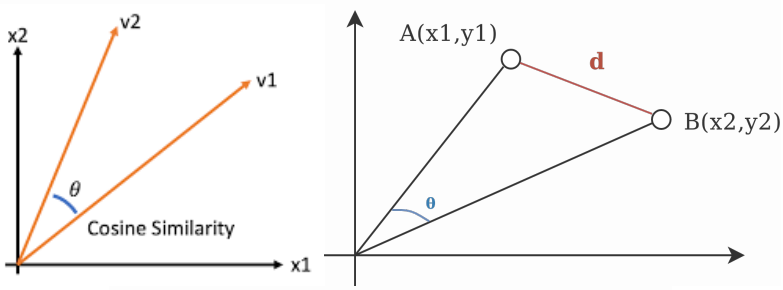
Besides scaling and cropping, we also have to consider that a slight distortion of the angle or perspectiv e of the camera taking the user's video can result in significant difference in the values of the keypoints . So, to minimize the corresponding error, affine transformation is applied. Affine transformation is a ki nd of 2-dimensional transformation that finds the least squared error between two coordinates.

```
def AffineMatrix(pose1,pose2):
    q = np.array([[pose1[1][0]],[pose1[1][1]],[pose1[8][0]],[pose1[8][1]]], np.float)
    p = np.array([[pose2[1][0], pose2[1][1], 1.0, 0.0, 0.0, 0.0], #
                  [0.0, 0.0, 0.0, pose2[1][0], pose2[1][1], 1.0], #
                  [pose2[8][0], pose2[8][1], 1.0, 0.0, 0.0, 0.0], #
                  [0.0, 0.0, 0.0, pose2[8][0], pose2[8][1], 1.0]], np.float)
    A, res, rank, s = np.linalg.lstsq(p, q)
    T = np.array([[A[0], A[1], A[2]],[A[3], A[4], A[5]],[0.0, 0.0, 1.0]], np.float)
    return T

def AffineTransformation(ref,pose):
    transformed  = []
    for i in range(len(pose)):
            if all(pose[i]):
              a = pose[i][0]
              b = pose[i][1]
              T = AffineMatrix(ref,pose)
              beforeAT = np.array([a,b,1])
              afterAT = T.dot(beforeAT)
              afterAT = afterAT / afterAT[2]
              transformed.append((int(afterAT[0]), int(afterAT[1])))
            else:
              transformed.append((None,None))
    return transformed
```

[Figure 34] Affine transformation code

First, we find the affine matrix that minimizes least squared error in the line from the neck(indexed 1) t o the middle hip(indexed 8) using numpy linear algebra function. Then, setting one person's keypoint a s a reference, affine matrix is applied to the other, resulting in the above figure where the line from one 's neck to mid hip is aligned regardless of how the user video was taken.

3.3.4.2 Similarity Metrics



[Figure 35] Cosine similarity and Euclidean distance

For measuring the degree of similarity, we use two metrics as shown above, the cosine similarity and the Euclid ean distance. Cosine similarity indicates how much the two vectors are similar to one another based on the angle

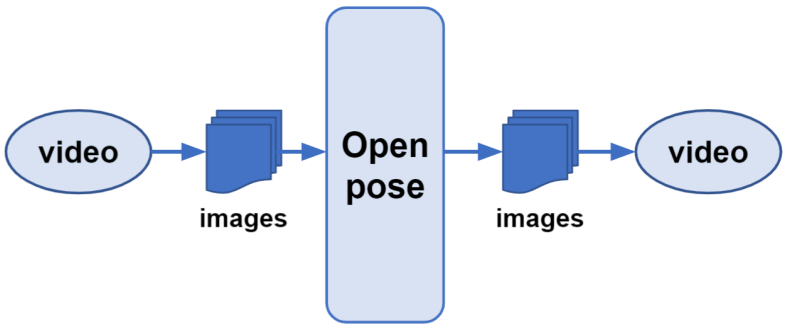
between the vectors. It can be computed by subtracting cosine distance from 1. In contrast, Euclidean distance just calculates the real distance between the two vectors like measuring with a ruler. Through several experiments, we set the pose matching threshold as cosine similarity of 0.9 and Euclidean distance as 1.

```python
def similarity_calc(pose1, pose2):
    # pd.DataFrame의 자료형으로 변환 --> (None, None) 키포인트 제하기 위하여
    pose1_temp = pd.DataFrame(pose1, columns=['x1','y1'])
    pose2_temp = pd.DataFrame(pose2, columns=['x2','y2'])
    pose_temp = pd.concat([pose1_temp, pose2_temp],axis = 1).dropna().reset_index(drop=True)
    pose1_temp = pose_temp[['x1','y1']]
    pose2_temp = pose_temp[['x2','y2']]
    # 다시 넘파이 배열로
    pose1 = pose1_temp.to_numpy()
    pose2 = pose2_temp.to_numpy()
    pose_1 = np.array(pose1, dtype=np.float)
    pose_2 = np.array(pose2, dtype=np.float)
    # 각 포인트를 0~1 사이 값으로 정규화
    pose_1[:,0] = pose_1[:,0] / max(pose_1[:,0])
    pose_1[:,1] = pose_1[:,1] / max(pose_1[:,1])
    pose_2[:,0] = pose_2[:,0] / max(pose_2[:,0])
    pose_2[:,1] = pose_2[:,1] / max(pose_2[:,1])
    # x, y 좌표 풀어서 쓰기 --> p1과 p2는 일차원 리스트
    sim_pose_1 = []
    sim_pose_2 = []
    for bodypart in range(pose_1.shape[0]):
        x1 = pose_1[bodypart][0]
        y1 = pose_1[bodypart][1]
        x2 = pose_2[bodypart][0]
        y2 = pose_2[bodypart][1]
        sim_pose_1.append(x1)
        sim_pose_1.append(y1)
        sim_pose_2.append(x2)
        sim_pose_2.append(y2)
    sim_pose_1 = np.array(sim_pose_1)
    sim_pose_2 = np.array(sim_pose_2)

    # 코사인 유사도와 유클리드 거리 반환 --> similarity metrics
    cosine_distance = spatial.distance.cosine(sim_pose_1, sim_pose_2) # 1 - cosine distance = cosine similarity
    euclidean_distance = spatial.distance.euclidean(sim_pose_1, sim_pose_2)
    return 1.0-cosine_distance, euclidean_distance
```

[Figure 36] Similarity calculation code

After eliminating some body parts with (None, None) values, each keypoint is normalized to a value between 0 and 1. The cosine distance and the euclidean distance is simply computed using functions from scipy.spatial.distance.

3.3.5 Overview of the Video Analysis



[Figure 37] Overview of the video analysis

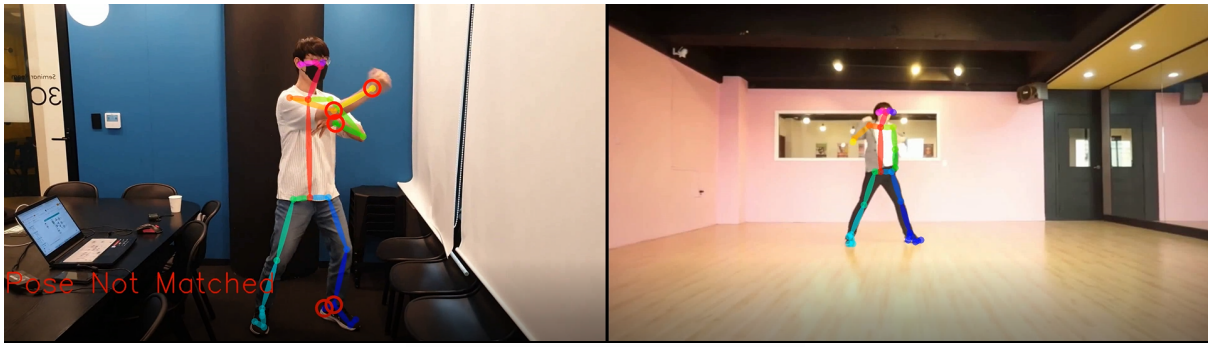
For analyzing every frame, we first divide the video into multiple images by image capture, then run the image through the openpose model to retrieve the keypoints from the dancer and the user. The extracted keypoints are used to calculate the degree of similarity to decide whether the poses matched or not.

```python
def final_function(filepath):
  total_time = time.time()

  params = dict()
  params["model_folder"] = os.path.join(OpenposeDir,'models')
  opWrapper = op.WrapperPython()
  opWrapper.configure(params)
  opWrapper.start()
```

```
  body_index = op.getPoseBodyPartMapping(op.BODY_25)
  body_pair_list = op.getPosePartPairs(op.BODY_25)
  body_pair = []
  idx = 0
  for i in range(int(len(body_pair_list)/2)):
    temp = (body_pair_list[idx], body_pair_list[idx+1])
    body_pair.append(temp)
    idx += 2

  frame_cnt = 0
  valid_frame = 0
  score = 0

  cap = cv2.VideoCapture(filepath)
  frame_width = int(cap.get(cv2.CAP_PROP_FRAME_WIDTH))
  frame_height = int(cap.get(cv2.CAP_PROP_FRAME_HEIGHT))
  fps = cap.get(cv2.CAP_PROP_FPS)

  vidwrite_protocol = cv2.VideoWriter_fourcc(*'MP4V')
  out = cv2.VideoWriter(os.path.join(OpenposeDir,'videos','output.mp4'), vidwrite_protocol, fps, (frame_width,  frame_heig
ht))

  while cap.isOpened():
      ret, frame = cap.read()
      if ret:
          frame_cnt += 1
          print(frame_cnt)

          datum = op.Datum()
          datum.cvInputData = frame
          opWrapper.emplaceAndPop(op.VectorDatum([datum]))
          raw_keypoints = datum.poseKeypoints
          Keypoints_list = getKeypointsList(raw_keypoints)

          try:
              if len(Keypoints_list) > 1:
                  valid_frame += 1
                  video_frame = datum.cvOutputData

                  pose1, pose2 = Keypoints_list[0],Keypoints_list[1]
                  pose1_processed, pose2_processed = scaling_cropping(pose1,pose2)
                  try:
                      pose2_processed = AffineTransformation(pose1_processed,pose2_processed)
                  except:
                      print("Affine transformation has failed for the frame")

                  cosine_similarity, eucldiean_distance = similarity_calc(pose1_processed, pose2_processed)

                  pose_match = 0 # pose_match = 0 --> True/ pose_match = 1 --> False
                  if cosine_similarity >= 0.9 and eucldiean_distance <= 1:
                      cv2.putText(video_frame,"Pose Matched",(0,int(video_frame.shape[0]*3/3.5)), cv2.FONT_HERSHEY_SIMPLEX
, 3, (0,255,0),3)
                      score += 1
                  else:
                      pose_match = 1
                      cv2.putText(video_frame,"Pose Not Matched",(0,int(video_frame.shape[0]*3/3.5)), cv2.FONT_HERSHEY_SIM
PLEX, 3, (0,0,255),3)

                  for i in range(len(pose1)):
                      if pose1[i] != (None, None) and pose2[i] != (None, None):
                          if pose1[i][0] < pose2[i][0]:
                              userpose = pose1
                              break
                          else:
                              userpose = pose2
                              break
                      else:
                          continue

                  if pose_match == 1:
                      distance = []
                      sorted_distance = []
                      for i in range(len(pose1_processed)):
```

```
                        if pose1_processed[i] != (None, None) and pose2_processed[i] != (None, None):
                            distance.append(spatial.distance.euclidean(pose1_processed[i], pose2_processed[i]))
                            sorted_distance.append(spatial.distance.euclidean(pose1_processed[i], pose2_processed[i]))
                        else:
                            distance.append(-1)
                            sorted_distance.append(-1)
                    sorted_distance.sort(reverse=True)
                    for j in range(5):
                        pt_idx = distance.index(sorted_distance[j])
                        pt = userpose[pt_idx]
                        cv2.circle(video_frame, pt, 25, (0,0,255), 7)

                out.write(video_frame)

            else:
                cv2.putText(video_frame,"Two people not detected",(0,int(video_frame.shape[0]*3/3.5)), cv2.FONT_HERSHEY_
SIMPLEX, 3, (0,0,255),3)
                out.write(video_frame)

        except Exception as e:
            print("Error")
            print(e)
            break
    else:
        print("Keypoint extraction and similarity checking have finished successfully!")
        break

  cap.release()
  out.release()

  average_score = score / valid_frame * 100
  print("average score: ", average_score)
  print("total time :", time.time() - total_time)
  return average_score
```

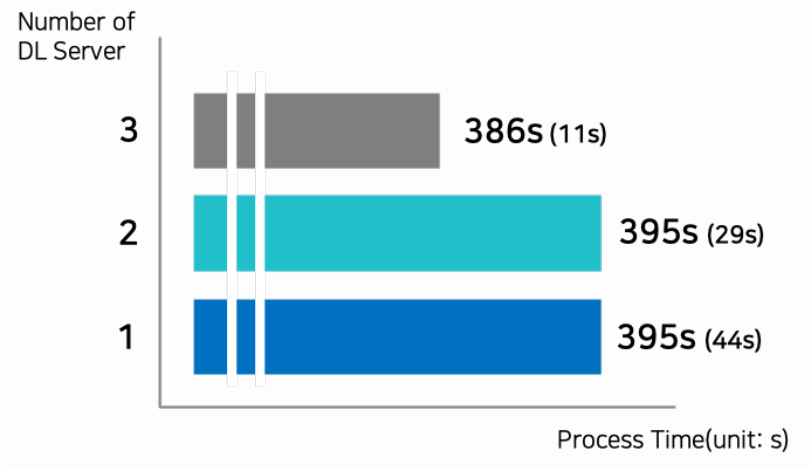
The above code shows a final_function that conducts the whole video analysis process. When the splitted short i nput video enters the final_function, the video is splitted into multiple frames by cv2.VideoCapture to conduct a nalysis for each frame. The method of analyzing frame by frame relieves the burden of storing keypoint values f or every frame to calculate similarity with. Also, we thought that the users would probably want to get the infor mation of the wrong moves that they were doing, so to represent the wrong parts, writing to each frame seemed i nevitable. Each frame image goes through keypoint extraction first, then for frames with more than one person, similarity between the poses are calculated with the aforementioned methods. To once again keep us from storin g every pose matching result for each frame, we instead choose to write the result down on the frame using cv2. putText. Also, we selected five maximum pose deviation points and indicated them on the output frame with a r ed circle. The written output frames are then merged together to form an output video(/content/openpose/videos/ output.mp4). Since the output video is merely a concatenation of frames, audio track is missing. Therefore, the p re-extracted audio from the dancer video is later merged with the video to form the final output video to send to the front-end(This part takes place in the back-end server). Some of the resulting output frames are shown below .



[Figure 38] Samples of output videos

As seen above, for frames where the user's and the dancer's pose do not match, red circles are added to indicate in what body parts the user is not doing correctly, compared to the dancer. The above figures suggest that our m odel shows high accuracy in detecting wrong dance moves.

3.3.6 Effect of Parallel Processing



[Figure 39] Effect of parallel processing

The above graph shows the effect of parallel processing, that is, leveraging multiple deep learning servers experi mented with a 14 second video. Note that the values inside the bracket indicate the actual amount of time spent i nside the deep learning servers, whereas the values outside the bracket represent the total amount of time of vide o analysis including video processing such as merging, splitting, etc. Using multiple DL servers to compute each chunk of short videos in parallel greatly reduces the time spent during actual inference in the DL server, but in t erms of whole video processing time the change is minor due to the low performance of our free tier EC2 server in AWS. It has been experimented that running the code in M1 local machine took way less time than running th e same code in EC2 server.

4. Reference

[1]GitHub. 2021. *741g/android-emulator-m1-preview*. [online] Available at: <https://github.com/741g/android-emulator-m1-preview>.

[2]Velog.io. 2021. *M1 맥에서 React-Native 세팅하기*. [online] Available at: <https://velog.io/@taese0ng/M1-%EB%A7%A5%EC%97%90%EC%84%9C-React-Native-%EC%84%B8%ED%8C%85%ED%95%98%EA%B8%B0>.

[3][ReactNative] [IOS] kemi. 2021. React Native M1 설정하기. [online] Available at: <https://taehoon95.tistory.com/115>.

[4]Medium. 2021. *Record and Upload Videos with React Native*. [online] Available at: <https://medium.com/react-native-training/uploading-videos-from-react-native-c79f520b9ae1> .

[5]GitHub. 2021. *itinance/react-native-fs*. [online] Available at: <https://github.com/itinance/react-native-fs> .

[6]GitHub. 2021. *react-native-camera/react-native-camera*. [online] Available at: <https://github.com/react-native-camera/react-native-camera>.

[7]GitHub. 2021. *zmxv/react-native-sound*. [online] Available at: <https://github.com/zmxv/react-native-sound>.

[8]GitHub. 2021. *react-native-async-storage/async-storage*. [online] Available at: <https://github.com/react-native-async-storage/async-storage> .

[9]SMU개발자. 2021. *JWT 구현하기*. [online] Available at: <https://smujihoon.tistory.com/241> .

[10]victolee. 2021. *[SpringBoot] AWS S3 연동 (1) - 파일 업로드 기본 (AmazonS3ClientBuilder)*. [online] Available at: <https://victorydntmd.tistory.com/334> .

[11]GitHub. 2021. *CMU-Perceptual-Computing-Lab/openpose*. [online] Available at: <https://github.com/CMU-Perceptual-Computing-Lab/openpose>.

[12]Medium. 2021. *A guide to deploying Machine/Deep Learning model(s) in Production*. [online] Available at: <https://blog.usejournal.com/a-guide-to-deploying-machine-deep-learning-model-s-in-production-e497fd4b734a>.

[13]GitHub. 2021. *chonyy/AI-basketball-analysis*. [online] Available at: <https://github.com/chonyy/AI-basketball-analysis>.

[14]Tutorials.pytorch.kr. 2021. *Flask를 이용하여 Python에서 PyTorch를 REST API로 배포하기 — PyTorch Tutorials 1.8.1 documentation*. [online] Available at: <https://tutorials.pytorch.kr/intermediate/flask_rest_api_tutorial.html>.

[15]Learn OpenCV | OpenCV, PyTorch, Keras, Tensorflow examples and tutorials. 2021. *Deep Learning based Human Pose Estimation using OpenCV*. [online] Available at: <https://learnopencv.com/deep-learning-based-human-pose-estimation-using-opencv-cpp-python/>.

[16]Cmry.github.io. 2021. *Euclidean vs. Cosine Distance*. [online] Available at: <https://cmry.github.io/notes/euclidean-v-cosine>.

[17]076923. 2021. *C# OpenCV 강좌 : 제 21강 - 아핀 변환 | 076923*. [online] Available at: <https://076923.github.io/posts/C-opencv4-21/>.