WALK_WALK 에서 사용한 외부 서비스 정보 정리 문서

0. 목차

1. Google Fitness API

(1) 개요

구글 OAuth2 로그인 구현에서 더 나아가, WALK_WALK 앱 사용자 (이하 resource Owner)들의 건강 정보 데이터를 활용하기 위해 Google Fitness API를 사용했습니다. 구글 OAuth2 로그인을 했을 시 기본 제공 되는 이름이나 이메일과 달리 건강 정보는 접근 제한된 정보에 포함되어서 사용 권한을 얻기 위한 추가적인 과정들이 필요했습니다.

(2) OAuth2 고도화를 통해 Resource Owner에게 건강 정보 사용 동의 구하기

가. 구글 로그인 SCOPE 추가

먼저 제한된 정보 사용을 위해 구글 로그인 RedirectURL의 Scope에 다음과 같은 URL을 추가하였습니다.

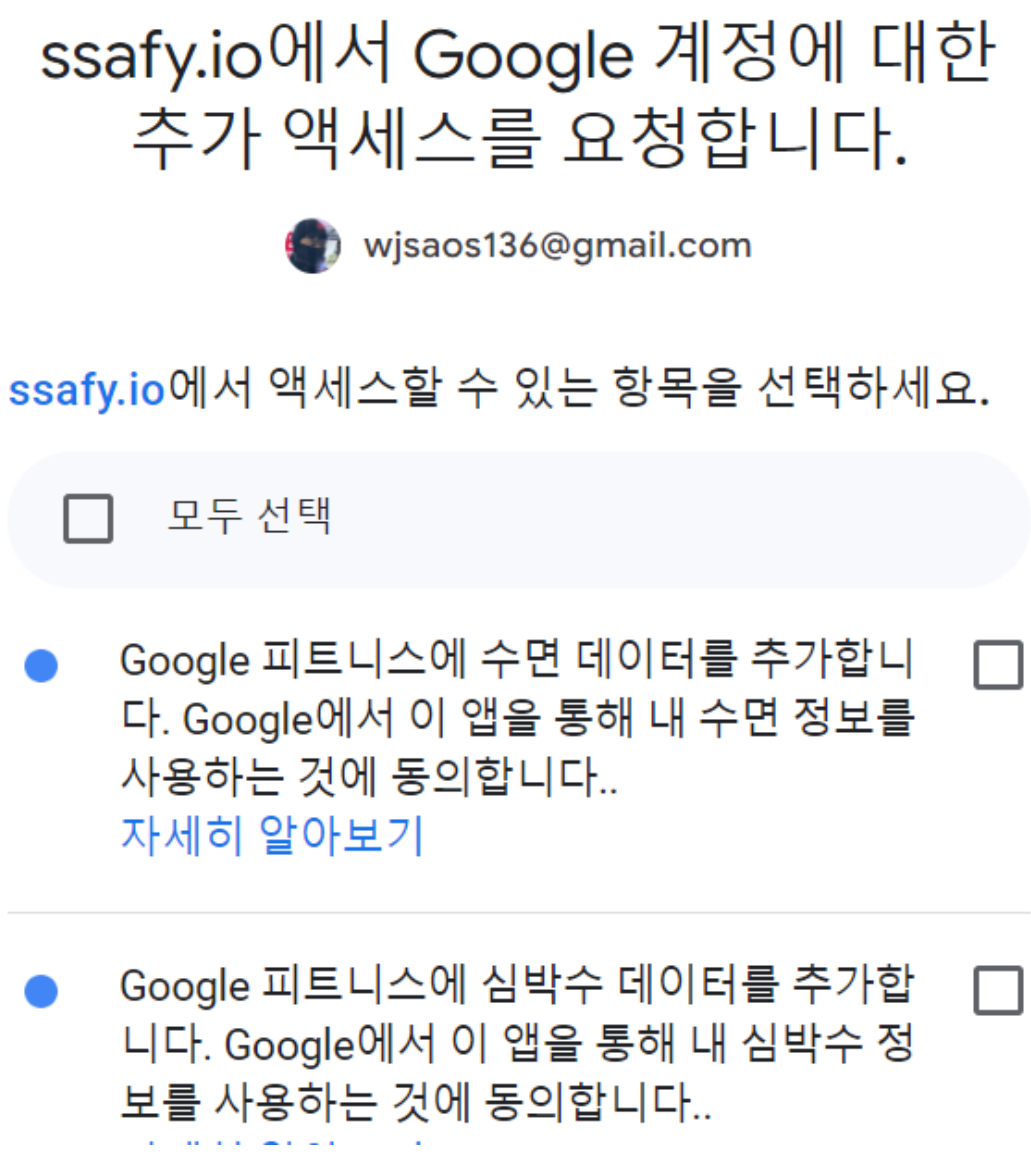
```
 const AUTH_URL = `https://accounts.google.com/o/oauth2/auth?
client_id=${CLIENT_ID}&redirect_uri=${REDIRECT_URI}&response_type=code&scope=http
s://www.googleapis.com/auth/userinfo.email
https://www.googleapis.com/auth/userinfo.profile
https://www.googleapis.com/auth/fitness.activity.read
https://www.googleapis.com/auth/fitness.blood_glucose.read
https://www.googleapis.com/auth/fitness.blood_pressure.read
https://www.googleapis.com/auth/fitness.body.read
https://www.googleapis.com/auth/fitness.heart_rate.read
https://www.googleapis.com/auth/fitness.body_temperature.read
https://www.googleapis.com/auth/fitness.location.read
https://www.googleapis.com/auth/fitness.nutrition.read
https://www.googleapis.com/auth/fitness.oxygen_saturation.read
https://www.googleapis.com/auth/fitness.reproductive_health.read
https://www.googleapis.com/auth/fitness.activity.write
https://www.googleapis.com/auth/fitness.blood_glucose.write
https://www.googleapis.com/auth/fitness.blood_pressure.write
https://www.googleapis.com/auth/fitness.body.write
https://www.googleapis.com/auth/fitness.heart_rate.write
https://www.googleapis.com/auth/fitness.body_temperature.write
https://www.googleapis.com/auth/fitness.location.write
https://www.googleapis.com/auth/fitness.nutrition.write
https://www.googleapis.com/auth/fitness.oxygen_saturation.write
https://www.googleapis.com/auth/fitness.reproductive_health.write
https://www.googleapis.com/auth/fitness.sleep.write&access_type=offline`
```

이와 더불어 구글에 등록한 앱 클라이언트에도 요청 Scope를 다음과 같이 추가해줍니다.

.write	consent to Google blood glucose info with this app.
.../auth/fitness .blood_pressure .write	Add info about you pressure in Google consent to Google blood pressure inf with this app.
.../auth/fitness .body.write	Google 피트니스어 정 정보 추가
.../auth/fitness .heart_rate.write	Google 피트니스어 데이터를 추가합니 Google에서 이 앱을 심박수 정보를 사용 동의합니다.
.../auth/fitness .body_temperature .write	Add to info about temperature in Go consent to Google body temperature information with th
/auth/fitness	Google 피트니스어

범위	사용자에게 표시되는
.../auth/fitness .sleep.write	Google 피트니스에 이터를 추가합니다. 에서 이 앱을 통해 내 보를 사용하는 것에 다.
.../auth/fitness .activity.read	Google 피트니스를 신체 활동 데이터 확 장
.../auth/fitness .blood_glucose .read	See info about your glucose in Google F consent to Google s my blood glucose information with thi
.../auth/fitness .blood_pressure .read	See info about your pressure in Google consent to Google s my blood pressure information with thi
.../auth/fitness .body.read	Google 피트니스에서 측정 정보 보기
.../auth/fitness .heart_rate.read	Google 피트니스에서 박수 데이터를 확인 Google에서 내 심박 를 이 앱과 공유하는 의합니다.
.../auth/fitness .body_temperature .read	See info about your temperature in Goo consent to Google s my body temperatu information with thi
.../auth/fitness .location.read	Google 피트니스 속 리 데이터 확인
.../auth/fitness .nutrition.read	Google 피트니스에서 정보 확인
.../auth/fitness .oxygen_saturation .read	See info about your saturation in Google consent to Google s my oxygen saturatio

나. 구글로부터 테스트 개발자 허용 받았는지 확인 및 OAuth2 로그인 화면의 변경 사항 확인



Google에서 확인하지 않은 앱

앱에서 Google 계정의 민감한 정보에 대한 액세스를 요청합니다. 개발자 (wjsaos2081@gmail.com)의 앱이 Google에서 인증을 받기 전에는 앱을 사용하지 마세요.

고급 설정 숨기기 안전한 환경으로 돌아가기

어떠한 위험이 발생할지 이해하고 개발자(wjsaos2081@gmail.com)를 신뢰할 수 있는 경우에만 계속하세요.

ssafy.io(으)로 이동(안전하지 않음)

Google에서 확인하지 않은 앱

앱에서 Google 계정의 민감한 정보에 대한 액세스를 요청합니다. 개발자 (wjsaos2081@gmail.com)의 앱이 Google에서 인증을 받기 전에는 앱을 사용하지 마세요.

고급 안전한 환경으로 돌아가기

테스트 사용자이기 때문에 다음과 같이 경고창에 뜹니다. 하지만 경고창의 "고급" 버튼을 누른 뒤에 관련 내용들을 사용자가 허용하면 쓸 수 있습니다. 저희는 앱 개발 단계이기 때문에 인증되지 않은 서비스라 다음과 같은 경고창이 뜨지만, 만약 상용화 한다면 제한 페이지는 더 이상 뜨지 않습니다. 따라서 개발 단계에서는 감안해야할 사안입니다. 다행히 해당 과정을 거쳐 사용을 동의한 Resource owner들의 건강정보 데이터는 WALK_WALK 서비스에서 사용할 수 있었습니다.

(3) 사용한 건강정보 영역 명세

WALK_WALK에서 사용한 건강정보는 다음과 같습니다.

- 신체 활동 정보
  (자전거 타기, 테니스 치기, 야외 조깅 등의 데이터를 사용자의 워치 혹은 앱에서 저장하여, 요청 시 전송해줍니다.)
- 심박수, 혈당 수치 정보

- 신체 측정 정보 (BMI 수치, 신장, 몸무게, 골격근량)
- 운동 시 위치 데이터
- 식단 데이터
  (사용자가 구글 피트니스 APP에 기입해야 사용할 수 있습니다.)
- 산소 포화도 정보
- 재생산 건강 (육체적 심리적 안녕과 관련된 데이터)
- 수면 정보

2. Google Map & Google GeoLocation API

(1) 개요

사용자가 운동하기 버튼을 클릭하면, 사용자의 가족 친구들에게 사용자가 운동하고 있음을 알리고, 실시간 위치를 공유해야 했습니다. 그를 위해서 Google Map의 지도와 Google GeoLocation에서 제공하는 GPS 정보를 이용했습니다.

(2) React + Vite 환경에서 해당 API를 활용하는 방법

Google 공식 문서는 JavaScript를 기준으로 API 활용법에 대해 설명해놓았습니다. Vanila JS 환경은 SPA 도 아니라서 한번 랜더링되면 화면이 재 랜더링되는 경우가 없습니다. 하지만 저희 팀에서는 React + VIte 환경으로 SPA를 구현 하였기에, 지도가 필요한 화면으로 이동될 때마다 재랜더링이 필요합니다. 따라서 기존의 방법으로는 에러가 많이 일어났습니다.
저희 팀에서 해당 문제를 해결한 방법은 다음과 같습니다.

a. Script에 Google Map 의존성 동적 주입을 위한 컴포넌트화

- 저희는 loadScript (맵 환경 세팅), initMap(화면 랜더링)을 컴포넌트화 해서 페이지 전환 시 동적으로 랜더링 되도록 다음과 같이 코드를 작성하였습니다.

```
// 동적 세팅
 useEffect(() => {

    const script = document.createElement('script');
    script.src = `https://maps.googleapis.com/maps/api/js?key=AIzaSyA7cuC-dzEXgBRPgwQiQp_jBVRaZtphYKO&callback=initMap&v=weekly`;
    script.async = true;
    script.defer = true;
    document.body.appendChild(script);


    if (ref.current && !map) {
      const newMap = new window.google.maps.Map(ref.current, {
        center: DEFAULT_CENTER,
        zoom: DEFAULT_ZOOM,
      });
      setMap(newMap);


      return () => {
        document.body.removeChild(script);
      }
```

```
    }
  }, [ref, map]);
```

- 5초마다 현재 운동하는 사용자의 GPS 위치를 브라우저로부터 얻어서 응원방에 입장한 사람들에게도 위치 공유

  setInterval 함수로 5초마다 GPS 정보를 알아내고, Stomp Socket을 이용하여 해당 정보를 가족 혹은 친구에게 전달합니다. 전달받은 사람들에게 위치가 공유 됩니다.

```
  // G. 위치 확인

  const getLocation = async () => {
    if (navigator.geolocation) {
        navigator.geolocation.getCurrentPosition(
            (position) => {
                setLatitude(position.coords.latitude);
                setLongitude(position.coords.longitude);
                setError(null);

                var GpsDTO = {
                  senderId: currentMember.member_id,
                  receiverId: pageOwnerId,
                  latitude: position.coords.latitude,
                  longitude: position.coords.longitude
                }

                stompClient.send("/pub/api/socket/enter", clientHeader,
JSON.stringify(GpsDTO));
            },
            (error) => {
                setError(error.message);
            }
        );
    } else {
        setError("Geolocation is not supported by this browser.");
    }

};



  if(currentMember.member_id === pageOwnerId){
    useEffect(() => {
      console.log("geoLocation 실행!")
        getLocation()
    }, [tabIndex])
  }
```

3. Klayton API

(1) 개요 - Ethereum Test-Net이 아닌 Klayton을 생성한 이유

Ethereum TestNet을 사용하기 위해서는 저희 앱 사용자들이 MetaMusk 라는 이더리움 지갑 주소 생성 앱을 설치하여 직접 지갑을 만들어야 했습니다. 저희 팀은 위에 기술한 일련의 과정 자체가 앱 사용자들에게 안 좋은 사용자 경험을 줄 것 같아, Rest API 형태의 Klayton API를 사용했습니다.

(2) 차별점 상세

가. **성능 및 확장성**:

- Klaytn은 이더리움과 비교하여 더 높은 성능과 확장성을 제공합니다. 이더리움은 Proof of Work (PoW) 방식을 사용하고 있어 블록 생성 속도가 느리고 처리량이 제한적입니다. 반면에 Klaytn은 Proof of Contribution (PoC) 방식을 사용하여 블록 생성 속도를 높이고 높은 처리량을 지원합니다. 이는 대규모 어플리케이션의 요구 사항을 충족시키는 데 유용합니다.

나. **최종성 보장**:

- Klaytn은 블록체인 네트워크에서의 최종성을 보장하는 기술을 채택하고 있습니다. 이더리움은 일반적으로 몇 번의 블록 확인을 통해 최종성을 확보합니다. 반면에 Klaytn은 블록 생성자에게서 블록을 받은 시점부터 해당 블록을 수정할 수 없도록 설계되어 있습니다. 이는 특히 금융 및 기업용 케이스와 같이 높은 신뢰성이 필요한 어플리케이션에 유용합니다.

다. **개발자 친화적인 도구**:

- Klaytn은 개발자들이 스마트 컨트랙트 및 DApp을 쉽게 개발할 수 있도록 다양한 도구와 리소스를 제공합니다. Klaytn은 Solidity를 지원하여 이더리움 개발자들이 쉽게 이전할 수 있도록 합니다. 또한 Klaytn의 클라우드 기반 블록체인 플랫폼인 'Baobab'은 개발 및 테스트를 위한 다양한 도구와 인프라를 제공합니다.

라. **한국의 기업 생태계 지원**:

- Klaytn은 한국의 대표적인 인터넷 기업인 Kakao와 함께 개발되었으며, 한국의 기업 생태계를 지원하는 데 중점을 두고 있습니다. 따라서 한국 기업들이 Klaytn을 선택할 경우 현지에서의 커뮤니티와 파트너십을 통해 더 많은 지원을 받을 수 있습니다.

마. **컨센서스 메커니즘의 다양성**:

- Klaytn은 다양한 컨센서스 메커니즘을 제공합니다. Proof of Contribution (PoC)은 블록 생성을 위해 일반 사용자들이 참여할 수 있도록 하는 동적 컨센서스 알고리즘입니다. 이를 통해 더 분산된 블록 생성을 실현하고 플랫폼의 안정성과 보안성을 강화할 수 있습니다.

4. AWS S3 Bucket

(1) 개요

사용자가 올린 프로필 사진이나 음성 응원 메세지를 자체 DB에 저장하고 다시 제공하는 과정에서 큰 부담을 느꼈습니다. 저희가 부담을 느낀 이유는 다음과 같습니다.

가. 사용자가 많아질수록 서버 메모리 소비량이 많아집니다. 이는 트래픽이 몰려 서버의 잔여 리소스가 필요할 경우 치명적이어집니다.

나. 파일 형태이기 때문에 DB 저장이 까다롭습니다. TEXT 형식의 데이터와 파일 형태의 DATA의 관심사 분리가 이루어지지 않아 서버 로직이 복잡해집니다.

(2) 어떻게 사용 했는가

위와 같은 이유에 더해서 AWS가 제공하는 내장 AI를 사용하기 위해 S3를 사용했습니다.
저희는

가. 클라이언트로부터 파일을 받음 (Multipart file 혹은 Base64 형태)

나. 서버에서 파일을 받음 (Base 64 형태일 경우 디코딩 필요)

다. S3에 업로드 후 S3에서 제공하는 URL 주소를 받아서 DB에 기록, 추후 앱 사용자가 관련 파일을 원할 경우 주소를 확인하여 정보 제공

와 같이 사용했습니다.

더하여, AWS에서 제공하는 STT AI 활용하였고, 활용을 어떻게 했는지는 주제 5번에서 설명 드리겠습니다.

5. AWS Transcribe

(1) 개요

AWS의 STT AI 서비스인 AWS Transcribe를 사용했습니다. 해당 서비스는 자사의 S3에 올라간 음성 파일 맞춤 STT 서비스를 제공합니다. 말하는 이의 자신감을 체크하거나, 혹은 개발자가 원하는 사항을 필터링 해서 Text로 변경할 수 있습니다. 차후 진행되는 자율 프로젝트에서는 해당 서비스의 실시간 소켓 기능을 이용한 통화내역 분석 구현도 해볼 예정입니다.

(2) 사용법

해당 AI의 SDK를 SpringBoot에 설치하였습니다.

```
implementation group: 'com.amazonaws', name: 'aws-java-sdk-transcribe', version:
'1.12.691' // Transcribe
```

이후 AWS를 Spring Boot에서 이용하기 위해, AWS의 문법을 다루는 AWS Lambda를 이용하였습니다.

해당 기능을 통해 먼저 빈 의존성 주입을 받았습니다.

```
    @Bean
    public AmazonTranscribe amazonTranscribe() {
        BasicAWSCredentials awsCreds = new BasicAWSCredentials(accessKey,
secretKey);

        return AmazonTranscribeClientBuilder.standard()
                .withRegion(region)
                .withCredentials(new AWSStaticCredentialsProvider(awsCreds))
                .build();
    }
```

이후 사용을 위해 AWS Lambda를 사용했는데, 사용법은 다음과 같습니다.

```
@Transactional
    public CompletableFuture<String> transcribeAudioFromS3(String audioS3Url) {

        CompletableFuture<String> future = new CompletableFuture<>();

        // A-1. URL에서 맨 마지막 3글자를 떼어내서 확인
        String extension = audioS3Url.substring(audioS3Url.length()-3);

        if(audioS3Url.substring(audioS3Url.length()-4).equals("webm")){
            extension = "webm";
        }

        log.info("STT 서비스 적용하려는 음성메세지의 확장자가 유효한 확장자인가? {}",
extension);

        String transcriptionJobName = "TranscriptionJob_" +
System.currentTimeMillis();

        StartTranscriptionJobRequest transcribeRequest = new
StartTranscriptionJobRequest()
                .withLanguageCode("ko-KR")
                .withMediaFormat(extension)
                .withMedia(new Media().withMediaFileUri(audioS3Url))
                .withOutputBucketName("d210")
                .withOutputKey("my-transcript2")
                .withTranscriptionJobName(transcriptionJobName);

        StartTranscriptionJobResult result =
amazonTranscribe.startTranscriptionJob(transcribeRequest);

        new Thread(() -> {
            while (true){
                String jobStatus =
getTranscriptionJobStatus(result.getTranscriptionJob().getTranscriptionJobName())
;

                if ("COMPLETED".equals(jobStatus)) {
                    log.info("Transcription job completed");

                    GetTranscriptionJobRequest getTranscriptionJobRequest = new
GetTranscriptionJobRequest()

.withTranscriptionJobName(result.getTranscriptionJob().getTranscriptionJobName())
;

                    GetTranscriptionJobResult result2 =
amazonTranscribe.getTranscriptionJob(getTranscriptionJobRequest);

                    log.info("STT 결과값에 든 내용: {}", result2);

                    String transcriptFileUri =
getTranscriptFileUri(result2.getTranscriptionJob());
                    if (transcriptFileUri != null) {
```

```
                        String textResult =
getTextFromTranscriptFile(transcriptFileUri);
                        future.complete(textResult);
                    } else {
                        future.completeExceptionally(new
RuntimeException("Transcript file URI is null"));
                    }
                    break;
                } else if ("FAILED".equals(jobStatus)) {
                    future.completeExceptionally(new
RuntimeException("Transcription job failed"));
                    break;
                } else {
                    log.info("Transcription job status: {}", jobStatus);
                }

                try {
                    Thread.sleep(5000); // 5초마다 상태 확인
                } catch (InterruptedException e) {
                    future.completeExceptionally(e);
                    break;
                }
            }
            log.info("STT Thread가 종료되었습니다.======================");

        }).start();

        log.info("STT JOB RESULT == {}", result);
        return future;
    }

    private String getTranscriptionJobStatus(String transcriptionJobName) {
        // Transcribe 작업 상태 확인 로직 구현
        GetTranscriptionJobRequest getTranscriptionJobRequest = new
GetTranscriptionJobRequest()
                .withTranscriptionJobName(transcriptionJobName);

        GetTranscriptionJobResult result =
amazonTranscribe.getTranscriptionJob(getTranscriptionJobRequest);

        return result.getTranscriptionJob().getTranscriptionJobStatus();
    }

    private String getTextFromTranscriptFile(String transcriptFileUri) {
        // S3에서 텍스트 결과 가져오는 로직 구현
        URI uri = URI.create(transcriptFileUri);
        log.info("transcriptFileUri: {}",transcriptFileUri);
        String bucketName = "d210";
        String objectKey = "my-transcript2"; // 앞의 '/' 제거

        String textResult = "";

        // S3 버킷에서 결과 파일 가져오기
        S3Object transcriptObject = amazonS3Client.getObject(new
GetObjectRequest(bucketName,objectKey));
```

```
        StringBuilder sb = new StringBuilder();

        // 결과 파일에서 텍스트 가져오기
        try (InputStream inputStream = transcriptObject.getObjectContent()) {
            textResult = IOUtils.toString(inputStream);
            TranscriptionResult result =  objectMapper.readValue(textResult,
TranscriptionResult.class);

            String transcript =
result.getResults().getTranscripts().get(0).getTranscript();
            double confidence =
Double.parseDouble(result.getResults().getItems().get(0).getAlternatives().get(0)
.getConfidence());

            sb.append(transcript).append("(현재 힘든 정도? :").append(100 -
(confidence*100)).append("%)");

        } catch (Exception e) {
            log.info("{}", e.getMessage());
        }

        log.info("음성 메세지 변환 내역 === {}",textResult);

        return String.valueOf(sb);
    }

    private String getTranscriptFileUri(TranscriptionJob transcriptionJob) {
        if (transcriptionJob != null) {
            Transcript transcript = transcriptionJob.getTranscript();
            if (transcript != null) {
                return transcript.getTranscriptFileUri();
            }
        }
        return null;
    }
}
```

AWS transcribe에서 입력받은 Speech 내역을 읽어드린 뒤 Text로 처리하는 과정이 필요하기 때문에 결과가 동기적으로 반환되지 않습니다. 따라서 Thread를 사용한 비동기 처리를 통해 STT 서비스의 결과를 반환 받도록 코드를 짰습니다.
해당 내용은

가. Text로 변환되어야 하는 음성 파일 주소를 S3에서 식별하여 Transcribe에 전달합니다. 그러면 Transcribe는 Transcriptionjob 이라는 객체로 해당 작업을 선언합니다. 이후 Transcription Job을 식별자로 원하는 정보를 얻어올 수 있습니다.

나. Job은 비동기적으로 처리됩니다. 따라서 자바 VM 내에 Thread 객체를 하나 만들어서 해당 작업이 끝났는지 안 끝났는지를 계속 리스닝 합니다. 작업이 끝난다면 Transcirpt job 객체의 상태가 "COMPLETED" 로 바뀝니다.

다. job의 상태가 "Completed"가 되면, 음성에 대한 Text 정보를 요청하여 가져옵니다. 이후 사용자의 클라이언트에 전송합니다.

6. Google AI TTS

TTS를 위해서는 Google AI TTS 서비스르 사용하였습니다. 해당 과정도 5번과 같이 SDK를 설치한 후, 사용하면 됩니다. Google AI의 경우에는 Rest API 형태로 되어있어서 사용 과정이 5번에 비해 상대적으로 편리적입니다.

```
    @Transactional
    public ResTTSInfo TextToSpeech(MessageInfo msg ){

        Members sender =
membersRepository.findById(msg.getSenderId()).orElse(null);

        // A. RestTemplate Header 작성
        HttpHeaders headers = new HttpHeaders();
        Charset utf8 = StandardCharsets.UTF_8;
        MediaType mediaType = new MediaType("application","json", utf8);
        headers.setContentType(mediaType);

        // B. POST 요청의 PARAMS 작성

        Gson gson = new Gson();

        JsonObject inputJson = new JsonObject();
        inputJson.addProperty("text", msg.getTextContent());

        JsonObject voiceJson = new JsonObject();
        voiceJson.addProperty("languageCode", "ko-KR");
        voiceJson.addProperty("name", "ko-KR-Neural2-c");
        voiceJson.addProperty("ssmlGender", sender.getGender() ==
Members.GenderType.MALE ? "MALE" : "FEMALE");

        // audioConfig JSON 생성
        JsonObject audioConfigJson = new JsonObject();
        audioConfigJson.addProperty("audioEncoding", "MP3");

        // 전체 요청 JSON 생성
        JsonObject requestJson = new JsonObject();
        requestJson.add("input", inputJson);
        requestJson.add("voice", voiceJson);
        requestJson.add("audioConfig", audioConfigJson);

        // 요청 Body 문자열로 변환
        String requestBody = gson.toJson(requestJson);


        HttpEntity<String> requestEntity = new HttpEntity<>(requestBody,
headers);

        ResponseEntity<String> responseEntity = new RestTemplate().exchange(
                "https://texttospeech.googleapis.com/v1/text:synthesize?
key="+GoogleAPIKey,
                HttpMethod.POST,
```

```
                requestEntity,
                String.class
        );

        try {
            ObjectMapper objectMapper = new
ObjectMapper().configure(DeserializationFeature.FAIL_ON_UNKNOWN_PROPERTIES,
false);
            log.info("TTS 변환 완료={}", responseEntity.getBody());
            return objectMapper.readValue(responseEntity.getBody(),
ResTTSInfo.class);
        }catch (JsonProcessingException e) {
            // B-5) 에러 내역 참고
            log.info(e.getMessage());
            return null;
        }

    }
```

7. Kakao Pay API

(1) 개요

운동량에 따른 용돈 지급이 저희 서비스의 상징적인 서비스 입니다. 해당 기능의 핵심 중 하나인 용돈 지급을 위한 전자상거래는 KakaoPay를 사용하였습니다.

(2) 카카오 페이를 선택한 이유

카카오 페이를 선택한 이유는 다음과 같습니다.

1. **인기와 보급도**:
    - 카카오페이는 한국에서 가장 널리 사용되는 결제 서비스 중 하나이며, 카카오톡을 통한 간편한 결제 기능으로 광범위한 사용자들에게 친숙합니다. 따라서 카카오페이 API는 많은 사용자들에게 더 쉽게 접근할 수 있으며, 이는 개발자들에게 더 넓은 시장을 제공합니다.
2. **다양한 결제 수단 및 서비스**:
    - 카카오페이는 다양한 결제 수단을 제공하며, 카카오페이 결제 뿐만 아니라 현금결제, 간편결제, 카드결제 등 다양한 옵션을 포함합니다. 또한 카카오페이는 카카오톡을 통한 송금 서비스와 같은 다양한 부가 서비스를 제공하여 사용자들에게 편의성을 제공합니다.
3. **간편한 연동 및 개발 환경**:
    - 카카오페이 API는 간단한 연동 절차를 통해 개발자들이 빠르게 결제 기능을 구현할 수 있도록 설계되었습니다. 또한 카카오 개발자 센터에서는 다양한 문서와 예제 코드를 제공하여 개발 과정을 보다 쉽게 만들어줍니다.
4. **보안성**:
    - 카카오페이는 안정적인 결제 시스템과 강력한 보안 기능을 갖추고 있습니다. 사용자 정보 및 결제 정보를 안전하게 보호하며, SSL 및 암호화 기술을 사용하여 데이터를 안전하게 전송합니다. 또한 사용자 인증 및 결제 승인 과정에서 추가적인 보안 절차를 제공합니다.
5. **개발자 지원 및 커뮤니티**:

- 카카오페이 API를 사용하는 개발자들은 카카오 개발자 센터를 통해 다양한 개발 리소스와 커뮤니티를 활용할 수 있습니다. 개발자들은 문제 해결에 도움을 받을 수 있으며, 다른 개발자들과의 협업 및 지식 공유가 가능합니다

8. Google OAuth2 API

로그인 과정의 편의를 위해서 GOOGLE OAUTH2를 사용하였습니다 OAUTH2는 기본적으로 사용자의 이메일, 닉네임, 프로필 사진등의 정보를 제공합니다. 초기에는 해당 정보를 이용하여, 사용자들이 기본정보를 입력하는 절차 없이 저희 앱을 쓸 수 있도록 환경을 조성했습니다. 그 후 Google Fitness를 추가하여 1번의 내용들을 추가 했습니다.