

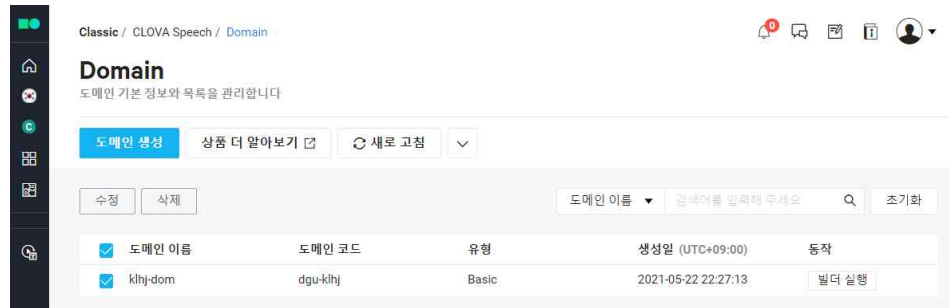
# 컴퓨터공학종합설계 서면 보고서

학생 팀별 작성용

과제 정보	
프로젝트명	비대면 환경에서의 효과적인 교육 및 비즈니스를 위한 개인 맞춤형 융합 콘텐츠 생성 기반 기술
팀명	김이홍조
협력 기업명	주식회사 유니크유엑스

프로젝트	
프로젝트 진행 내용	<p>1. Naver CLOVA Speech</p> <p>가. 개요</p> <p>NEST (Neural End-to-end Speech Transcriber) 음성 인식 기술을 통해 빠르고 쉽게 미디어(동영상)의 음성인식을 제공한다. CLOVA Speech Recognition (CSR) 서비스는 1분 이내의 명령형 음성 인식에 최적화된 반면, CLOVA Speech는 길이가 긴 오디오 또는 비디오 파일에 대해 음성 인식 결과를 확인할 수 있다.</p> <p>나. 특징</p> <ul style="list-style-type: none"><li>1) 높은 성능의 한국어 장문 디테이션</li><li>2) 미디어 인식에 강한 모델</li><li>3) 전화망 음성 인식을 강화한 모델</li><li>4) 지속적인 품질 향상으로 똑똑해지는 NEST 엔진</li><li>5) 문장 자동 분리 및 타임 스탬프 지원</li><li>6) 인식결과 수정 에디터 제공</li></ul> <p>다. 사용방법</p> <ul style="list-style-type: none"><li>1) API 호출 방식</li><li>각 도메인은 고유의 API 호출 URL을 제공한다. API 호출 URL로 음성 인식을 할 원본 파일을 보내고, 엔진에서 처리한 결과 값을 응답한다. 인식 결과에 대해 수정 에디터는 지원하지 않는다.</li><li>2) 빌더 사용 방식</li><li>도메인 생성 후 빌더를 사용하여 UI 사용 환경으로 음성 인식 작업을 요청한다. 원본 파일은 Object Storage에서 가져오거나 파일 업로드를 통해 설정 가능하다. 작업 목록에서 처리 상태 및 결과를 확인할 수 있고, 인식 결과에 대해 편집 기능을 제공한다.</li></ul> <p>2. Naver CLOVA Speech 실습</p> <p>가. 사용 준비</p>

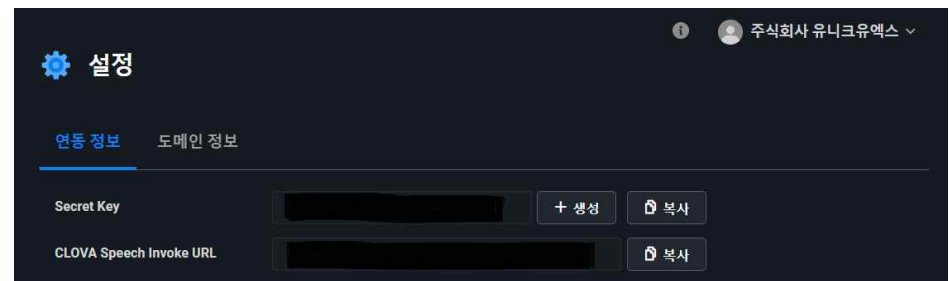
### 1) 도메인 생성



### 2) Object Storage



### 3) Secret Key와 URL 확인



## 나. API 사용 방식으로 음성 인식 테스트 진행

### 1) 코드

네이버 클라우드 플랫폼 홈페이지에서 제공하는 CLOVA Speech API 중, 파이썬으로 실행하였다.

<https://api.ncloud-docs.com/docs/ai-application-service-clovaspeech-clovaspeech>

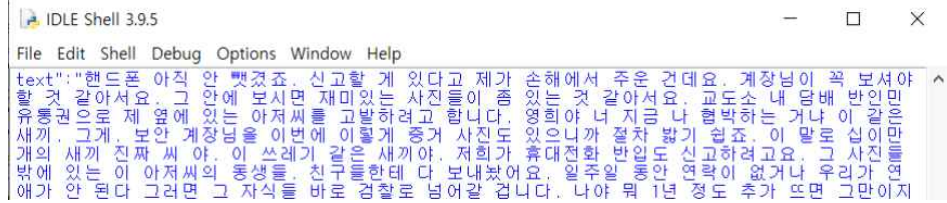
```
class ClovaSpeechClient:
    # Clova Speech invoke URL
    invoke_url = ''
    # Clova Speech secret key
    secret = ''
```

위의 따옴표 안에 2-가-3)에서 확인한 url과 secret key를 삽입 후 실행하였다.

```
if __name__ == '__main__':
    # res = ClovaSpeechClient().req_url(url='http://example.com/media.mp3', completion='sync')
    # res = ClovaSpeechClient().req_object_storage(data_key='data/media.mp3', completion='sync')
    res = ClovaSpeechClient().req_upload(file='/data/media.mp3', completion='sync')
    print(res.text)
```

위의 파일 경로를 로컬 저장소에 맞게 수정하여 실행하였다.

## 2) 결과



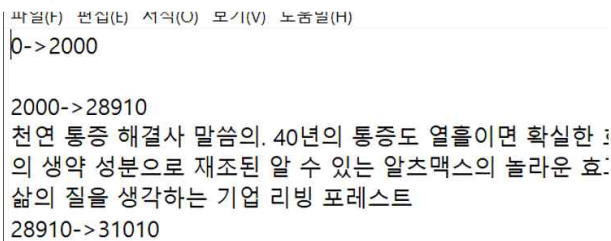
```
text": "핸드폰 아직 안 뺏겼죠. 신고할 게 있다고 제가 손해에서 주운 건데요. 계장님이 꼭 보셔야 할 것 같아서요. 그 안에 보시면 재미있는 사진들이 좀 있는 것 같아서요. 교도소 내 담배 반입민 유통권으로 제 옆에 있는 아저씨를 고발하려고 합니다. 영희야 너 지금 나 헐박하는 거냐 이 같은 새끼. 그게. 보안 계장님을 이번에 이렇게 중거 사진도 있으니까 절차 밟기 쉽죠. 이 말로 십이만 개의 새끼 진짜 싸 아. 이 쓰레기 같은 새끼야. 저희가 휴대전화 반입도 신고하려고요. 그 사진을 밖에 있는 이 아저씨의 동생들. 친구들한테 다 보내놨어요. 일주일 동안 연락이 없거나 우리가 연애가 안 된다 그러면 그 자식을 바로 검찰로 넘어갈 겁니다. 나야 뭐 1년 정도 추가 쓰면 그만이지
```

## 다. 자막 파일 생성 진행

### 1) 코드

```
res_json = json.loads(res.text)
subtitle = open("subtitle.smi", 'w', encoding='utf-8')
for seg in res_json["segments"]:
    subtitle.write(str(seg['start']) + "->" + str(seg['end']) + "\n" + seg['text'] + "\n")
```

### 2) 결과



```
0->2000

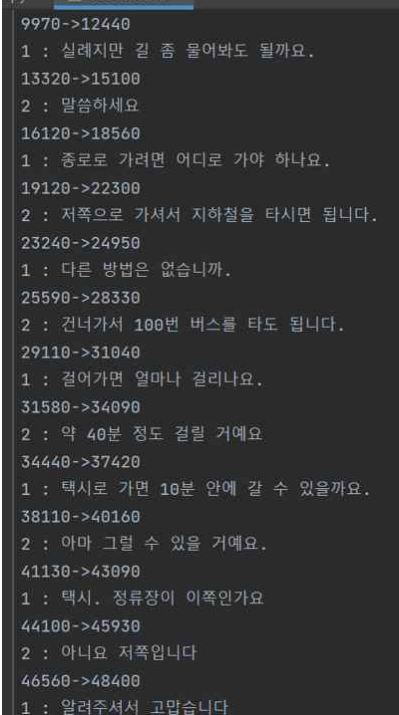
2000->28910
천연 통증 해결사 말씀의. 40년의 통증도 열흘이면 확실한 :
의 생약 성분으로 재조된 알 수 있는 알츠맥스의 놀라운 효:
삶의 질을 생각하는 기업 리빙 포레스트
28910->31010
```

## 라. 화자 인식 진행

### 1) 코드

```
subtitle = open("subtitle.smi", 'w', encoding='utf-8')
for seg in res_json["segments"]:
    subtitle.write(str(seg['start']) + "->" + str(seg['end']) + "\n" + seg['speaker'] + " : " + seg['text'] + "\n")
```

### 2) 결과

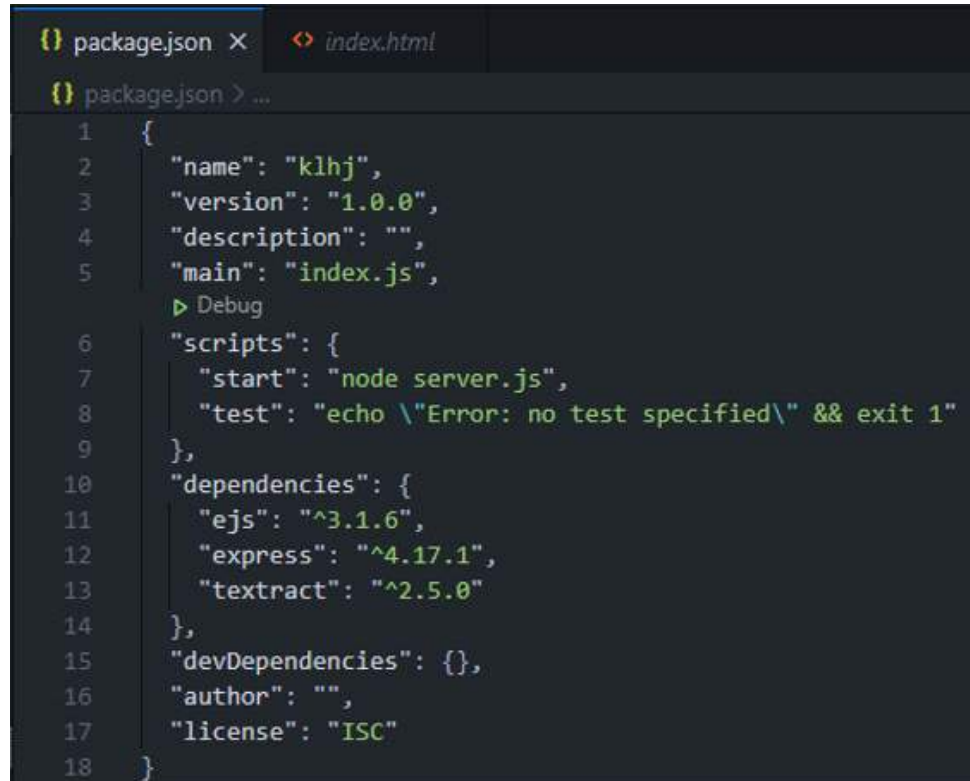


```
9970->12440
1 : 실례지만 길 좀 물어봐도 될까요.
13320->15100
2 : 말씀하세요
16120->18560
1 : 종로로 가려면 어디로 가야 하나요.
19120->22300
2 : 저쪽으로 가서서 지하철을 타시면 됩니다.
23240->24950
1 : 다른 방법은 없습니까.
25590->28330
2 : 건너가서 100번 버스를 타도 됩니다.
29110->31040
1 : 걸어가면 얼마나 걸리나요.
31580->34090
2 : 약 40분 정도 걸릴 거예요
34440->37420
1 : 택시로 가면 10분 안에 갈 수 있을까요.
38110->40160
2 : 아마 그럴 수 있을 거예요.
41130->43090
1 : 택시. 정류장이 이쪽인가요
44100->45930
2 : 아니요 저쪽입니다
46560->48400
1 : 알려주셔서 고맙습니다
```

1과 2로 화자를 각각 인식한 것을 확인하였다.

### 3. Node.js를 이용한 로컬 서버 구축

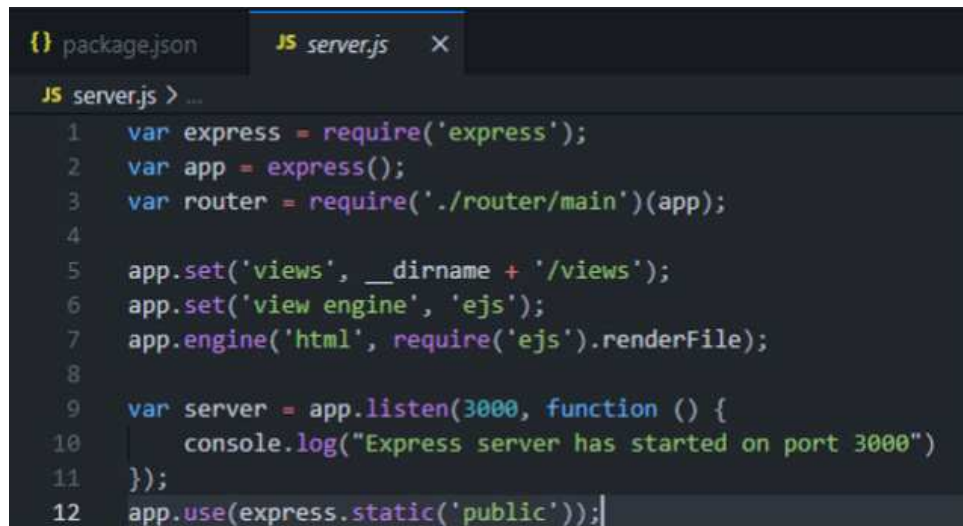
가. npm init을 이용해 package.json 파일을 구축



```
{
  "name": "klhj",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "start": "node server.js",
    "test": "echo \\\"Error: no test specified\\\" && exit 1"
  },
  "dependencies": {
    "ejs": "^3.1.6",
    "express": "^4.17.1",
    "textract": "^2.5.0"
  },
  "devDependencies": {},
  "author": "",
  "license": "ISC"
}
```

나. npm install ejs / npm install express -save / npm install textract를 이용해 dependency에 추가

다. server.js 파일에 express를 이용한 라우터 파일 구축, html 로딩을 위해 ejs 사용

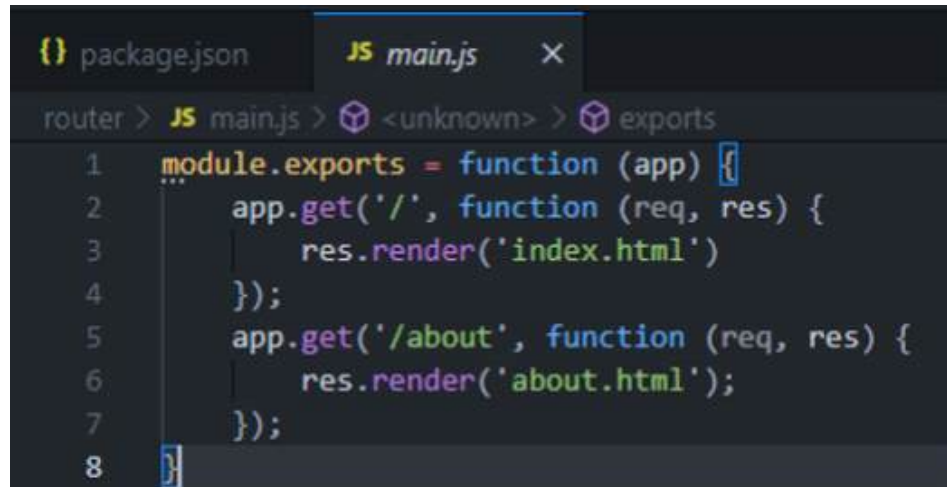


```
var express = require('express');
var app = express();
var router = require('./router/main')(app);

app.set('views', __dirname + '/views');
app.set('view engine', 'ejs');
app.engine('html', require('ejs').renderFile);

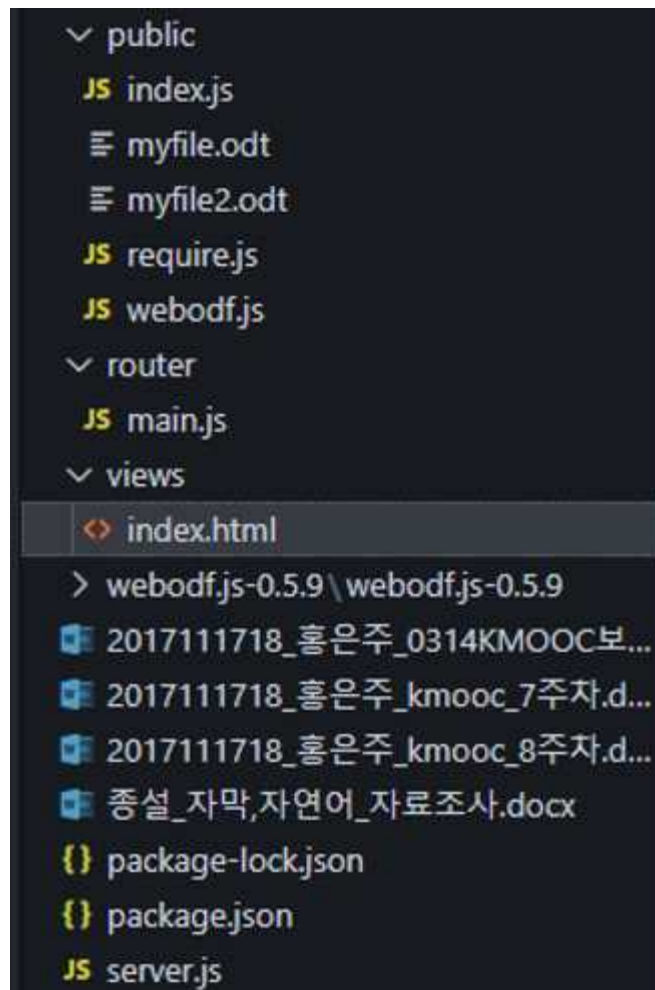
var server = app.listen(3000, function () {
  console.log("Express server has started on port 3000")
});
app.use(express.static('public'));
```

라. router 폴더 생성 후 main.js 생성



```
{} package.json JS main.js X
router > JS main.js > <unknown> > exports
1 module.exports = function (app) {
2   app.get('/', function (req, res) {
3     res.render('index.html')
4   });
5   app.get('/about', function (req, res) {
6     res.render('about.html');
7   });
8 }
```

마. server.js에 맞게 html 파일의 dirname을 views로 지정



```

  public
    JS index.js
    myfile.odt
    myfile2.odt
    JS require.js
    JS webodf.js
  router
    JS main.js
  views
    index.html
> webodf.js-0.5.9\webodf.js-0.5.9
2017111718_홍은주_0314KMOOC보...
2017111718_홍은주_kmooc_7주차.d...
2017111718_홍은주_kmooc_8주차.d...
종설_자막,자연어_자료조사.docx
{} package-lock.json
{} package.json
JS server.js
```

바. 각종 정적파일을 public 폴더에 저장 후 실행 (npm start)  
로컬 서버가 구동되며 http://localhost:3000/으로 실행 가능하다.

## 사. 결과화면



텍스트, 이미지, 표가 모두 정상적으로 출력되는 것을 확인할 수 있다.

## 회의내용

### #멘토링 진행 후 구현 방향성 논의

- 멘토님과의 멘토링을 통해 구현 방향성 논의
- 앞으로의 계획 논의

### #관련 프로그램 실습 진행

- ODF 파일 사용을 위한 WebODF 실습
- 텍스트 추출을 위한 textract 실습
- 음성인식을 위한 Naver CLOVA 실습