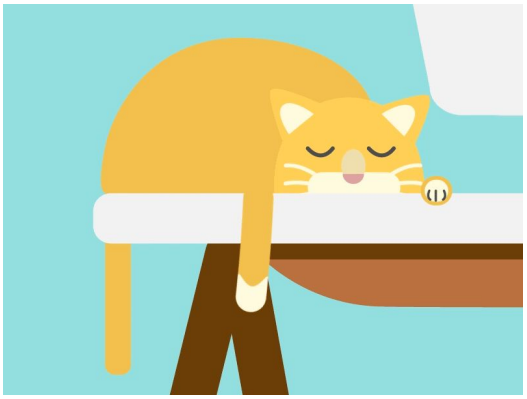


하루종일

자는건

고양이 밖에 없어

장화신은 고양이
한승구 이상민
최진석 손다은

목차

1. 프로젝트 개요

요즘 코로나 사태로 인해 코로나의 전파를 막기 위해서 대부분의 교육 과정이 온라인으로 진행되는 경우가 많다. 그렇게 온라인으로 전환한 것은 좋지만, 온라인 교육 환경의 특성상 오프라인 교육 보다 모니터를 오랫동안 바라보며 집중해야하는 시간이 점점 늘어나게 되었다. 그러다 보니 눈의 피로가 많이 쌓이게 되고 쉽게 졸음을 느낄 수 있는 환경이 되었다. 그래서 우리 장화신은 고양이 팀은 이러한 환경을 개선하기 위해 사용자가 졸면 미리 입력해둔 문장을 TTS를 통해 음성으로 변환하여 잠에서 깨워주는 하루종일 자는건 고양이밖에 없어(이하 하자고) 서비스를 준비했다.

이 서비스를 구현하기 위해서 우리가 생각한 시스템의 구조는 다음과 같다. 첫번째로 웹캠을 이용하여 졸음을 인식한다. 이 졸음을 인식하기 위해 opencv 를 활용하였다. 얼굴의 눈, 코, 입, 눈썹 등 68개의 포인트 지점을 인식하는것을 기준으로 얼굴을 인식하고 그 중에서 눈부분의 거리를 확인하여 사용자가 오랫동안 눈을 감고 있으면 졸음으로 판단한다. 졸음으로 판단되었으면 잠을 깨워주는 시스템으론 TTS를 이용하였다. TTS를 활용한 이유는 사람이 자신의 이름을 들으면 다른 알람보다는 더 확실히 깬다고 판단하였기 때문이다. TTS는 카카오톡의 open API를 활용하기로 하였으며 카카오 음성 변환 Host에 원하는 문장과 세부 설정을 저장하여 request한 후 response를 받아 이를 mp3파일로 저장한 후 원할 때 재생하는 방법을 사용하였다. 들려주는 것 만으로는 잠을 깨는데에 부족하다고 판단하여 oenpose를 통해 사용자에게 기지개를 펴는 등 잠을 깨게 하는 행동을 하도록 하여 좀 더 잠깨는 것이 효과적으로 일어나도록 하였다. 그리고 이를 웹으로 구현하였다. 클라이언트 부분은 html 과 javascript, css 를 이용하여 개발하였고 서버는 Django 기반으로 개발하였으며 프로젝트의 주 기술인 이미지 처리를 서버에서 opencv를 통해 처리, 주로 cv2 모듈을 활용하였다.

하자고 서비스를 활용함으로써 사용자는 잠에서 깨는것 뿐만 아니라 자신이 졸고있는 상태라는 것을 객관적으로 인지하는것 자체로도 대체 방안과 생활 패턴을 점검할 수 있고, 그로 인해 전체적인 생활의 생산성과 효율성을 증대할수 있으며, 코로나를 발판삼아 본격적으로 주류 교육과 학습의 플랫폼이 된 온라인에서 실용적이고 재미있는 툴의 경험을 제공는 역할이 될 수 있다. 더 나아가 딥러닝을 활용한 TTS의 정교화로 유명인 들의 목소리를 사용하여 비즈니스 모델을 구축하거나 제스처 사용으로 재생되는 유튜브 컨텐츠의 창작자들과 비즈니스 기회를 창출 할 수 있다고 예상된다.

2. 개발 내용

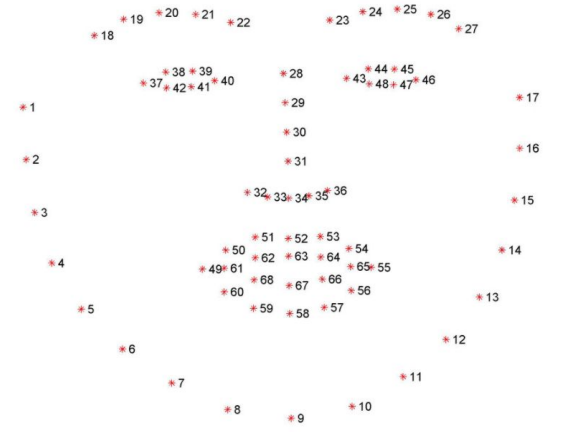
1) opencv 를 이용하여 졸음 인식

- 웹캠을 이용하여 입력받은 사용자의 영상이 조명으로 인해 영향을 받지 않기 위해 OpenCV에서 제공하는 cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)를 이용하여 grayscale로 변환한다. OpenCV는 사용자의 영상을 한 프레임씩 읽어와서 모델에 적용하기 위한 영상의 전반적인 처리를 수행한다.
- 입력된 영상에서 얼굴을 검출하기 위해 Haar feature 기반 Cascade Classifier 를 이용한다.

  OpenCV는 Haar-cascade트레이너와 검출기를 모두 제공하여 이미지에서 특정 객체를 검출하는데 유용하게 사용된다.

```
face_cascade = cv2.CascadeClassifier("haarcascades/haarcascade_frontalface_default.xml")
# opencv는 haar-cascade 트레이너와 검출기를 모두 제공.
```

- 그 후 dlib라이브러리를 이용하여, 얼굴에서 68개의 landmark를 찾아낸다. 68개의 landmark가 학습된 모델데이터는 다음과 같고, 해당 위치에 가져와 압축을 푼다.



<68개의 landmark는 다음과 같이 나온다.>

```
detector = dlib.get_frontal_face_detector()
# 얼굴 인식용 클래스 생성(기본 제공되는 얼굴 인식 모델 사용)
predictor = dlib.shape_predictor('shape_predictor_68_face_landmarks.dat')
# 인식된 얼굴에서 랜드마크 찾기위한 클래스 생성

(lStart, lEnd) = face_utils.FACIAL_LANDMARKS_IDXS['left_eye']
(rStart, rEnd) = face_utils.FACIAL_LANDMARKS_IDXS['right_eye']
```

- dlib 라이브러리를 사용하여 얼굴 인식용 클래스를 생성하고 인식된 얼굴에서 랜드마크를 찾기위한 클래스를 생성한다. 그 후, 사용자의 눈 위치를 뽑아내 변수로 저장한다.

```
def eye_aspect_ratio(eye):
    A = distance.euclidean(eye[1], eye[5])
    B = distance.euclidean(eye[2], eye[4])
    C = distance.euclidean(eye[0], eye[3])

    ear = (A+B) / (2*C)
    return ear
```

<사용자 눈 비율>

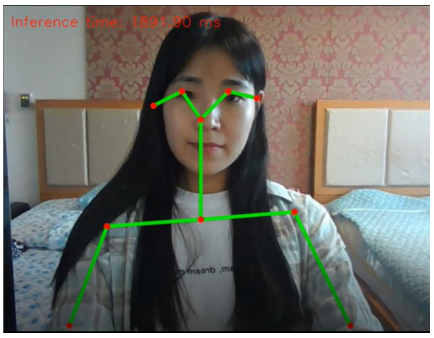
- 저장한 눈 위치의 변수를 통해 사용자 눈 비율을 계산하여 일정 경계값 이상 눈이 감기는 시간이 지속되면 졸음으로 인식한다.
- 이 때, 사용자마다 눈 모양이 다르기 때문에 졸음에 대한 정확도를 높이기 위해 입력영상의 첫 부분에 눈을 감고 뜸을 반복하여, 사용자의 눈에 맞춘 경계값을 설정한다.

2) Openpose 를 이용한 포즈 인식

- openpose는 단일 이미지에서 인체, 손, 얼굴 및 발 키포인트 (총 135 개 키포인트)를 동시에 감지 하는 최초의 실시간 다중 사람 인식 시스템이다.

- 관절 부위 (키 포인트) 를 먼저 감지하여 서로 연결해 사람의 자세를 추정하는 Bottom Up 방식을 사용했다.
- 입력 이미지를 VGG 망의 input으로 넣으면, feature들이 강조되어 나오게 되는데, 이 output을 2개로 분할하여 stage에 있는 각각의 branch에 다시 input으로 들어갑니다.
- branch1은 confidence map을 예측하는 망을 의미하고, branch2는 affinity field를 예측하는 망을 의미하고, 각각의 branch를 거쳐서 나온 결과물들은 합쳐서 다음 stage로 들어갑니다.
- 이러한 과정을 반복하면서 키포인트를 학습하고 정확도가 올라가게 됩니다.
- 데이터 학습을 위해서는 Caffe를 사용하므로 설치하고, 빌드를 해야하며 위의 과정들을 통해 얻을 키포인트를 입력합니다.

```
# 모델 & 설정 파일
model = 'openpose/pose_iter_440000.caffemodel'
config = 'openpose/pose_deploy_linevec.prototxt'

# 포즈 점 개수, 점 연결 개수, 연결 점 번호 쌍
nparts = 18
npairs = 17
pose_pairs = [(1, 2), (2, 3), (3, 4),  # 왼팔
              (1, 5), (5, 6), (6, 7),  # 오른팔
              (1, 8), (8, 9), (9, 10),  # 왼쪽다리
              (1, 11), (11, 12), (12, 13),  # 오른쪽다리
              (1, 0), (0, 14), (14, 16), (0, 15), (15, 17)]  # 얼굴
```



<opencv를 통해 입력영상에 대한 사전처리를 진행한 후, 다음과 같은 결과를 얻을 수 있다>

- 하지만 openpose를 real-time 으로 돌리면 연산량의 증가로 인해 영상의 딜레이가 생겨서 이를 해결하기위해 GPU를 이용해 모듈을 돌려보기로 하였다.
- CUDA toolkit과 CUDNN을 설치한후, CMAKE를 설정하여 필요한 라이브러리들을 설치하여 시도하였지만, 버전 오류 등의 문제와 기존모델과 합치는 과정에서 실시간으로 동작을 인식하여 알람을 해제하는 기능을 하기에 매끄럽지 않을 것 같다고 판단되어 OPENPOSE는 기능 추가를 하지 않기로 결정하였다.

3) TTS 구현

```python
from gtts import gTTS
import os
myText = input()

language = 'ko'

output = gTTS(text = myText,lang = language, slow = False)

output.save("output.mp3")


os.system("start output.mp3")
```

- TTS를 구현하기 위해 첫번째로 사용한 것은 gTTS 이다. gTTS는 가장 기본적인 TTS모듈 중 하나로 파이썬에 등록되어 있어 pip install gTTS를 통해 바로 다운로드 받아서 파이썬 코드에 적용할 수 있다. 다양한 언어를 지원하지만 목소리가 선택 언어에 따라 고정되어 있어 변경이 불가능하다. input 값은 str 타입으로 input하면 되며 출력값은 mp3파일로 저장된다. 다만 gTTS는 성능이 그다지 좋다고 하기는 부족하다. 억양이 굉장히 부자연스럽고 정말로 기본적인 컴퓨터가 읽어주는 느낌에 가깝다

```python
import requests
import json

kakao_tts_url = "https://kakaoi-newtone-openapi.kakao.com/v1/synthesize"
rest_api_key = "your rest_api_key"

headers = {
    "Content-Type": "application/xml",
    "Authorization": "KakaoAK " + rest_api_key,
}
with open('data.xml', 'rb') as fp:
    data = fp.read()

res = requests.post(kakao_tts_url, headers=headers, data = data)
rescode = res.status_code
if(rescode==200):
    print("TTS mp3 저장")
    response_body = res.content
    with open('1111.mp3', 'wb') as f:
        f.write(response_body)
else:
    print("Error Code:" + rescode)
```

- 억양이 굉장히 부자연스러운 gTTS를 대신해 국내에서 지원하는 TTS를 찾아본 결과 카카오와 네이버에서 지원하는 TTS가 있다. 다만 네이버는 유료서비스만 지원했기 때문에 Kakao TTS 를 이용하였다. Kakao TTS는 모듈 설치는 물가능하고 원하는 input 값을

카카오의 서버에 request 하여 그 받아온 값을 mp3파일로 저장해서 실행하는 방법이다. 카카오 서버에 보낼 값은 특이하게도 다른 TTS와는 달리 xml 문서 구조로 전송해야 하는데, xml.etree.ElementTree모듈을 사용해서 xml파일로 만드는 함수를 별도로 사용하여 작성한 후 이를 카카오 서버에 보낸다. 목소리는 총 4종류 있으며 속도조절 크기조절 등등 상당히 다양한 기능을 지원한다.

- 세번째는 Tensorflow를 이용한 TTS 이다. Tensorflow를 통해 모델을 학습시킨 후 사용하는 TTS 이다. 기본적으로 지원하는 모델이 정해져 있으며 최초의 TTS를 실행시키면 모델을 학습시키는 과정이 있기 때문에 처음에는 상당히 오래걸린다. 하지만 그만큼 가장 자연스러운 발성을 보여준다. 다만 위의 두가지 TTS와는 달리 설치해야하는 모듈이 상당히 많으며 처음 실행시에 학습시간까지 존재하다 보니 웹에서 가동될때는 어떠한 문제가 발생할지 모르겠다.
- 딥러닝을 이용한 TTS. 위에서 Tensorflow를 이용한 TTS를 보고 우리가 직접 모델을 학습시켜서 TTS를 만들수 있지 않을까 라는 발상에서 나온 아이디어이다. 하지만 모델을 학습시키는데 필요한 데이터를 만들기 위해선 해당 사람의 목소리를 배경음이나 다른 사람의 목소리랑 섞이지 않은 부분만 잘라내여 문장단위로 자른 후 그걸 음성인식을 통해 각 문장마다 어떠한 내용인지를 저장하여 학습을 시켜야하는데 그래도 괜찮은 TTS를 하기 위해선 적어도 4시간 이상의 데이터를 학습시켜야 된다고 한다. 하지만 이렇게 많은 양의 데이터를 만드는 것 자체가 엄청난 시간이 필요하며 학습시키는 데도 한참 걸린다고 되어있었기 때문에 시도하기전에 하지 않기로 결정하였다.
- 기존의 카카오 TTS가 약간 잠을 깨우는데는 부적합한 목소리라는 의견이 있어 이를 목소리 변조를 통해 좀더 자극적인 소리로 바꿀려고 시도하였다. 하지만 파이썬에서 기본적으로 지원하는 것은 구간 자르기나 속도증감, 크기조절과 같은 기본적인 시스템 밖에 존재하지 않았고, 음성 변조는 대부분 전용 프로그램을 이용하거나 특정 사이트에서만 가능했다. 하지만 이 사이트를 request를 통해 변조하여 내 시스템으로 가져오는 것은 불가능 하다고 판단하여 보류되었다.

4) 웹으로 구현

- 웹으로 구현하기 전에 파이참을 통해 모델이 제대로 작동하는지 확인한다.
- 로컬 환경에서 동작할 수 있도록 view파일에 모델을 적용시켜 준다.

```
@gzip_page #콘텐츠를 압축하는 용도로 사용
def livefe(requests):
    drow = Drowsiness()
    try:#이미 연속된 값을 가지고 있는 VideoCamera()를 gen이라는 generator에 보내서 계속해서 값을 yield를 통해서 리
        return StreamingHttpResponse(drow.main_run(), content_type="multipart/x-mixed-replace;boundary=frame")
    except:  # This is bad! replace it with proper handling
        exit()
```

- 위의 livefe 함수를 이용해 웹캠을 통해 들어오는 프레임을 StreamingHttpResponse을 통해서 클라이언트 쪽에 지속적으로 전달해준다.
-
```
<img src="{% url 'livefe'%}">
```
- 클라이언트에서는 이미지 태그의 src속성을 통해 매번 전달되는 프레임을 입력 받는다.

```
def gen(camera):
    while True:
        frame = camera.get_frame()
        yield(b'--frame\r\n'
            b'Content-Type: image/jpeg\r\n\r\n' + frame + b'\r\n\r\n')
```

- 위의 gen함수 frame = camera.get_frame()을 통해서 class VideoCamera에서 while 문을 통해서 지속적으로 전달되는 프레임을 가져온다. 그리고 yield문을 통해서 반복문에서 처리된 frame을 반복 루틴마다 리턴해준다.

```
def __init__(self):
    self.video = cv2.VideoCapture(0)
    (self.grabbed, self.frame) = self.video.read()
    threading.Thread(target=self.update, args=()).start()
```

- 위의 self.video = cv2.VideoCapture(0)을 통해서 사용자 기기 중에서 웹캠을 실행할 수 있고, 제일 마지막 threading.Thread문을 통해서 쓰레드를 생성해서 프레임을 매번 갱신해줄 수 있다.
- 위 코드까지 작성하면 로컬에서 웹 스트리밍이 잘 이루어 질거라고 예상했으나 웹캠을 한번 키면 끌 수 없는 문제가 발생하였다. 웹캠을 켤 때는 각 유저가 서버에 접속해서 새로 인스턴스를 만들고 그때마다 쓰레드가 생성되어 start가 진행되기 때문에 문제가 없으나 각 유저에게 생긴 인스턴스의 쓰레드를 멈출수 있는 방법도, 얻은 카메라를 릴리스 할 수 있는 방법이 존재하지 않는다는 치명적인 문제가 발생했다.
- 따라서 위의 문제를 해결하기위해 socket.io를 사용하였다. 소켓을 통해 클라이언트와 연결을 하면 각 유저가 클라이언트에서 소켓을 통해서 서버와 통신을 하고, 이를 통해서 웹캠을 켜고 끄는 기능을 만들수 있게 된다. 이 소켓의 개념이 디장고에선 channels라는 모듈로 별로도 존재하며 이 모듈은 주로 채팅을 구현할 때 사용된다. 이 모듈을 사용하기 위해선 설치해야하는 파일과 설정해줘야하는 요소가 있지만 channels를 통해서 구현한 서버를 에코 서버로만 사용할 것이기 때문에 컨슈머[1] 까지만 만들어 주었다.
- connect 함수를 통해서 클라이언트 쪽에서 요청한 소켓 연결과 관련된 일을 처리하고 receive 함수를 통해서 클라이언트에서 전달된 메시지를 처리할 수 있다. 여기서 이 기능을 통해 웹캠을 켜고 끄는 기능을 구현하였다. 웹캠을 켤때는 쓰레드를 하나 생성해서 circle이라는 함수를 통해 frame을 Drowsiness 클래스로부터 계속 얻어와 소켓을 통해 클라이언트에게 전달하도록 하였다.

[1] Consumer, 소켓 통신을 할 때 소켓을 연결, 차단, 메시지를 받아주는 역할을 하는 클래스이다.

```
<img id='web_cam' class='item' src="{% static '/images/cat.gif' %}" alt='대신 이미지'>
```

```
chatSocket.onmessage = (e) => {

    var reader = new FileReader();
    let data = e.data;
    reader.onload = function(event) {
        document.querySelector('#web_cam').src = reader.result;
    };
    reader.readAsDataURL(data);

};
```

- 위의 코드를 통해 클라이언트 측에서 web_cam이라는 아이디를 가진 이미지 태그의 src 값을 서버로부터 소켓을 통해 가져온 프레임으로 채워주면 웹 스트리밍이 가능해진다.

```
chatSocket = new WebSocket(
    'ws://'+window.location.host+'/ws/stream/'+roomName+'/'
);
```

- disconnect 함수를 통해 소켓과의 연결이 끊어졌을 때 현재의 circle을 돌리고 있는 쓰레드를 끝내기 위해 self.camera.stop[2]과 self.camera = None을 호출한다.
- 다음으로 본 서비스에서 가장 중요한 Drowsiness 클래스를 만들었다. 서비스의 시나리오 대로 init_open_ear와 init_close_ear을 통해서 눈을 뜨고 있을 때와 감았을 때 각각 5개의 샘플을 구해서 이에 대한 평균 값을 구해서 눈을 떴을 때와 감았을 때를 구분할 수 있는 경계값을 구한다.
- 쓰레드를 통해 main_run에서 프레임을 계속해서 갱신해주고 감지한 데이터를 통해서 현재 눈을 뜨고있는지 감고있는 지 구분하고 졸고있다면 TTS 음성이 나오도록 설정하였다.

[2] Drowsiness에서 돌아가는 쓰레드를 멈추기 위해 이에 해당하는 조건 값을 변경하고, 캠을 꺼주고, 알람음을 None으로 입력해주는 역할을 한다. 즉 클라이언트와의 연결이 끊어졌을 때 안전하게 해당서비스를 종료시켜주는 역할이다.

```python
class User_info(models.Model):
    objects=models.Manager()

    user = models.OneToOneField(
        User, on_delete=models.CASCADE, related_name='streamer'
    )

    name=models.CharField(max_length=20)
    mail=models.CharField(max_length=20)

class TTS_text(models.Model):
    objects=models.Manager()
    text=models.TextField()
    user_info = models.OneToOneField(
        User_info, on_delete=models.CASCADE, related_name='tts'
    )

class Eye_threshold(models.Model):
    objects=models.Manager()
    eye_info=models.FloatField()
    user_info = models.OneToOneField(
        User_info, on_delete=models.CASCADE, related_name='eye_threshold'
    )
```

- 하자고 서비스를 위해 총 세가지 모델을 사용했다. User와 User_info는 모델 간에 1:1 로 설계했고 User_info와 TTS_text, Eye_threshold 모델 사이도 1:1 관계가 성립되어 모델간에 필요한 정보가 있으면 로그인 된 변수를 통해서 이들에 대한 정보를 편리하게 사용하기 쉽게 설계하였다. 하지만 사용자의 눈 사이 경계값을 저장해주는 Eye_threshold 모델을 시간이 부족하여 완성하지 못하였다.

```python
def tts(request, user_pk):

    if request.method == 'POST':

        # 데이터베이스에 추가하기

        # print(user_pk)

        content = request.POST['text']
        name = TTS.make_tts(content)

        user = User.objects.get(pk=user_pk)

        TTS_text.objects.create(
            text=name,
            user_info = user.streamer
        )

        return redirect('../stream/green/')

    else : return render(request, 'tts.html', {'user_pk':user_pk})
```

- 이제 눈을 감았을 때 알림음을 위해 TTS를 만드는 과정을 만들었다. TTS 클래스를 선언하여 tts의 이름, mp3파일을 만들어 준다. 여기서 mp3의 이름이 중복되게 만들어지지 않기 위해서 time함수를 통해 만든 후 이 파일을 경로명을 return 해준다. 그리고 이 return 된 파일의 경로를 TTS_text 모델을 저장해준다.

```html
<body>
    <br/>
    Listen to your alarm sound?<br/>
    <p id='path' style='display: none;'>{{results.text}}</p>
    <script type="text/javascript">

        var audio = new Audio();

        audio.src = '../' + document.querySelector('#path').innerText

    </script>
    <input type="button" onClick="audio.play();" value="PLAY"/>
    <input type="button" onClick="audio.pause();" value="PAUSE"/>
    <!-- <input type="number" onChange="audio.volume=this.value"/> -->
```

-

- 자바스크립트의 Audio를 통해서 만들어진 TTS를 들을 수 있게 구현했으며, alarm 이라는 path로 사용자가 접속을 하면 user.pk를 통해서 User 모델에 저장되어있는 해당 사용자의 tts_url을 가져온다.

```
user = User.objects.get(pk=user_pk)

context = {
    'results' : user.streamer.tts
}

return render(request, 'alarm.html',context)
```

- 그리고 가져온 서버 변수를 자바스크립트에서 사용하기 위해 우선 p태그를 사용자들에게 보이지 않게 만들고 그 p태그 안에 tts_url을 저장한다. 그 다음 audio.src 값을 지정할 때 가져와 사용한다. 이 과정을 통해 사용자들은 웹에서 자신이 설정한 TTS 를 들을 수 있다.

```
def change_tts(request, user_pk):

    user = User.objects.get(pk=user_pk)
    text = user.streamer.tts.text

    content = request.POST['text']

    name = TTS.change_tts(text, content)

    tts_val = TTS_text.objects.filter(text = text)
    tts_val.update(
            text = name
        )

    return HttpResponse(name)
```

- 회원 가입할 때 처음에 만든 TTS가 맘에 들지 않을 경우를 위해서 change_tts 함수를 만들었다. 이 함수를 통해서 사용자는 변경된 TTS를 사용중인 화면에서 alarm 버튼을 눌러 새로 만들어진 TTS를 미리 들어보는 것이 가능하다.

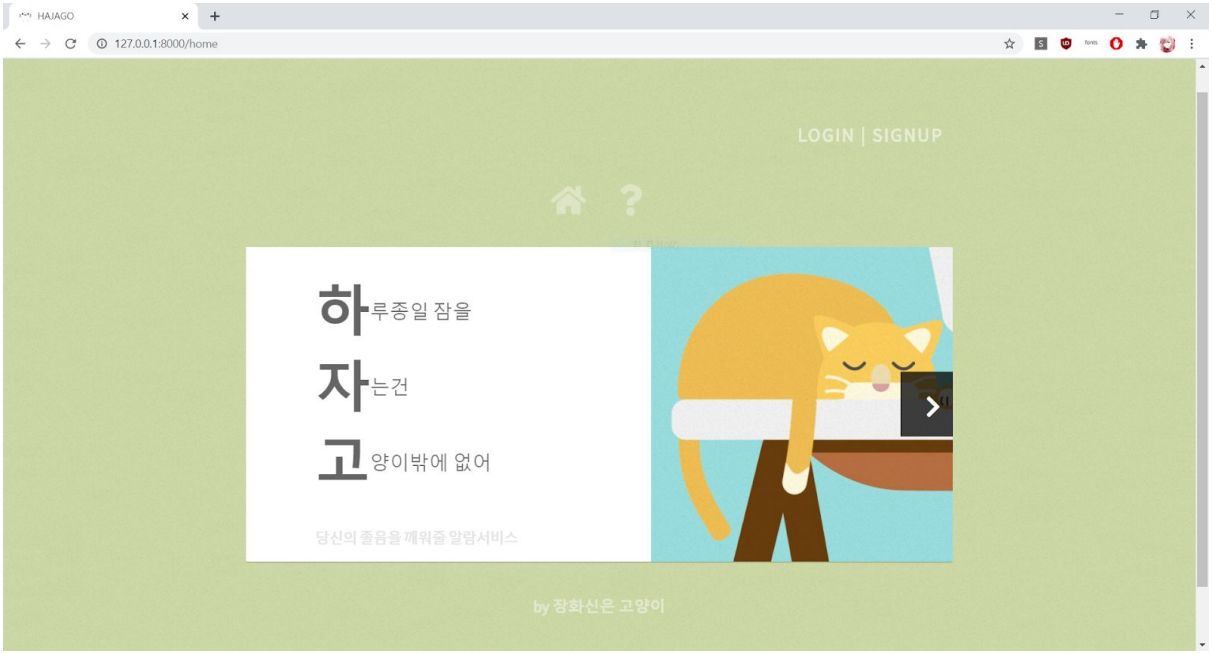
```
$.ajax({
    url:'{% url "get_tts_url" %}',
    type:"POST",
    data:{'pk':pk, 'csrfmiddlewaretoken': '{{ csrf_token }}'},
    dataType : "json",
    success: function(result) {
        if (result) {
            // console.log(result.tts_url);

            if (!check){
                check = true;

                setTimeout(function() {
                    chatSocket.send(JSON.stringify({
                        'message': result.tts_url
                    }));
                }, 700);
```

- change_tts 함수만으로는 새롭게 바뀐 TTS를 적용시킬 수 없기 때문에 위의 과정을 통해서 바꾸는 기능을 추가하였다.

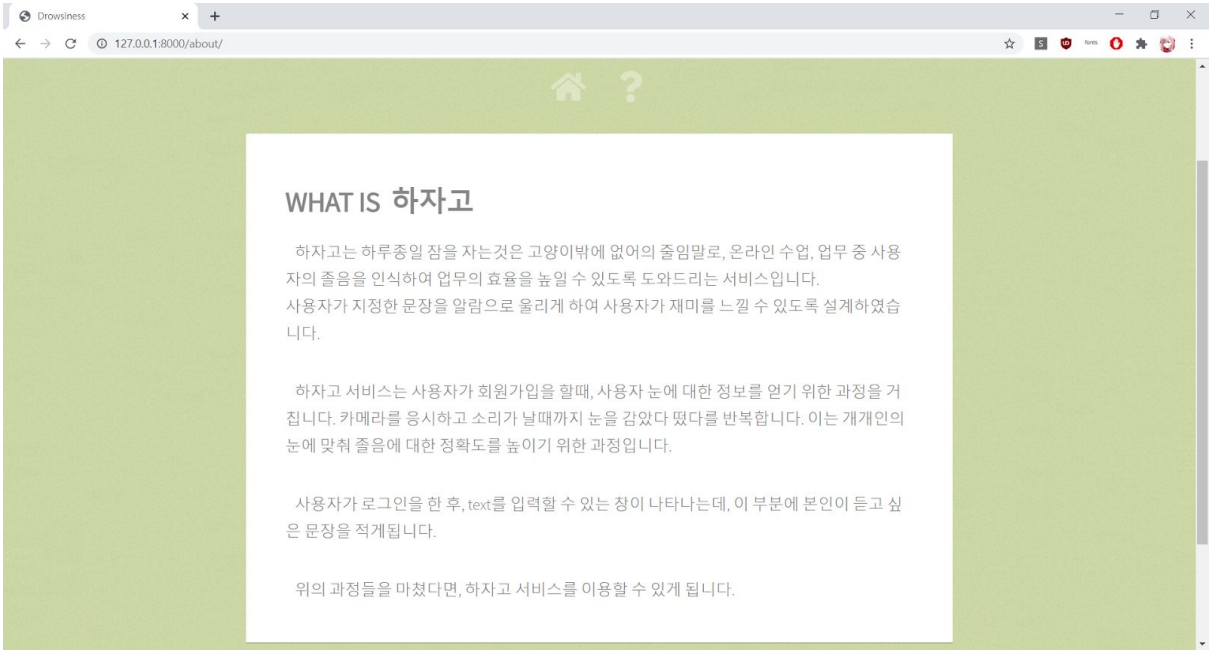
```
def get_tts_url(request):

    user_pk = request.POST.get('pk', None)
    print(user_pk)
    user = User.objects.get(pk=int(user_pk))
    context = {
        'tts_url': user.streamer.tts.text
    }

    return HttpResponse(json.dumps(context), content_type="application/json")
```

- start 버튼을 누르면 ajax를 통해서 알림음으로 지정한 TTS의 url을 서버에 요청함으로써 현재 로그인 된 사용자의 user.pk를 얻어서 User.object.get(pk=int(user_pk))를 통해서 user 객체를 얻은 후, user.streamer.tts.text을 통해서 현재 로그인 중인 사용자의 tts_url을 반환받도록 하였다. 이 과정을 통해 하자고 서비스에서 사용자가 설정한 TTS가 알람음으로 적용될 수 있도록 설계 하였다.

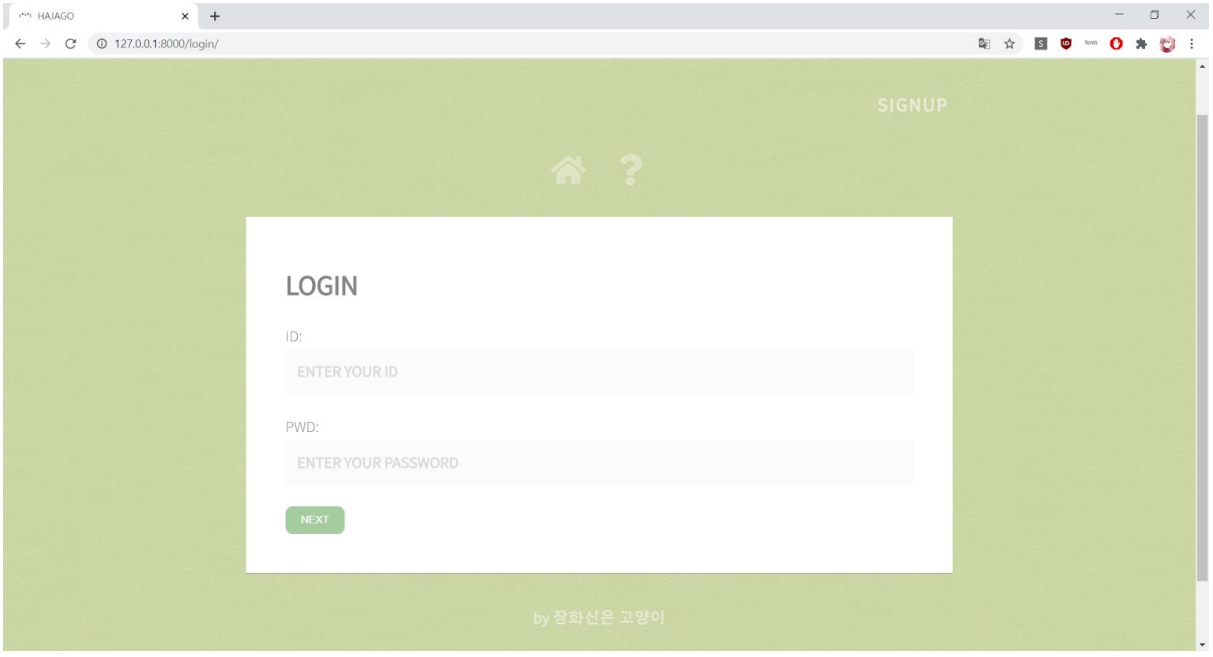
3. 결과



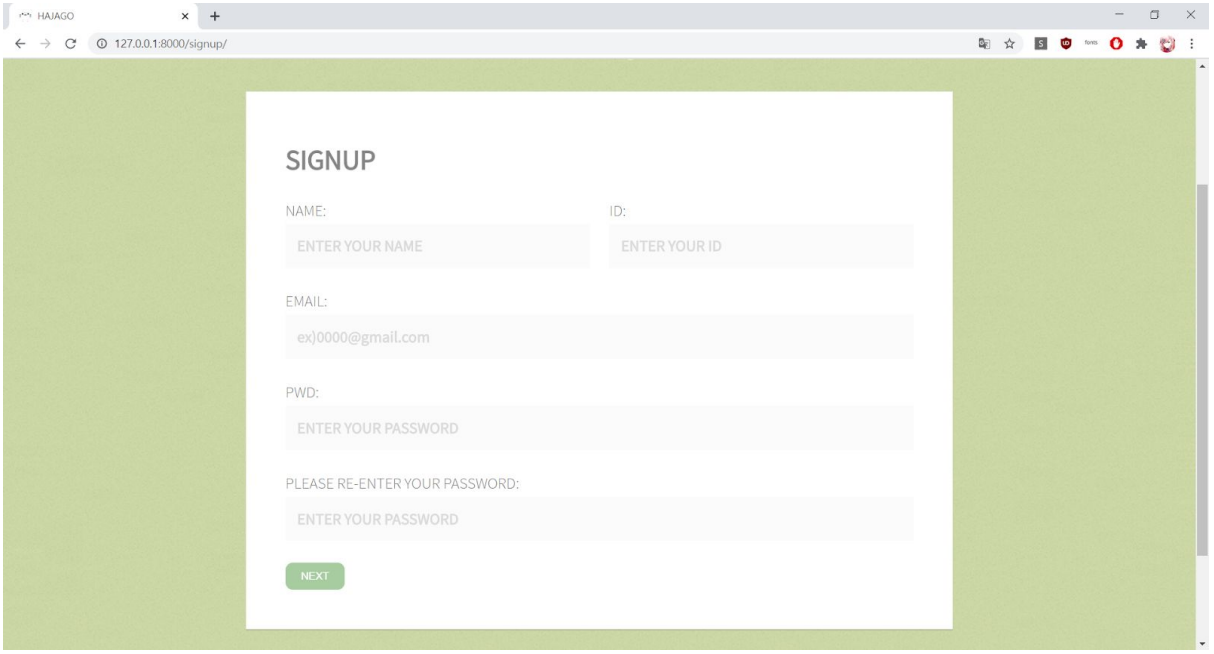
- 하자고 서비스의 메인 홈 화면이다. 이 화면에서 로그인이나 회원가입 후 졸음 감지 서비스 화면으로 넘어갈 수 있다.



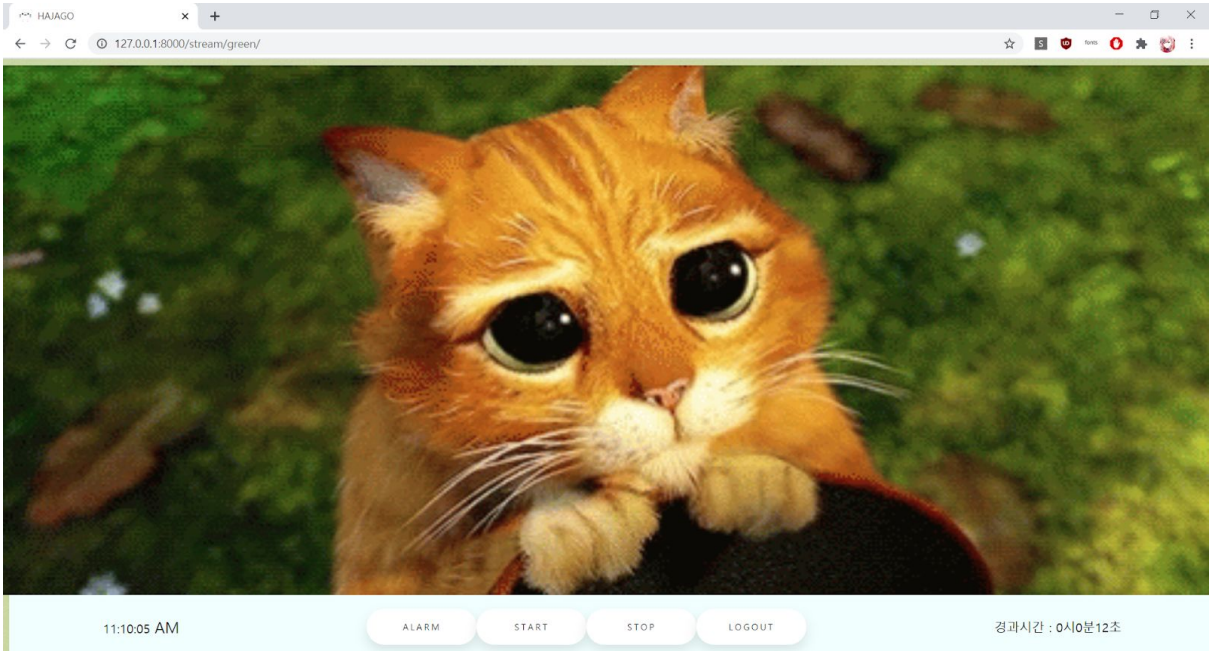
- 이해를 돕기 위해 이 사이트에 대한 설명이 간략하게 적혀있다.



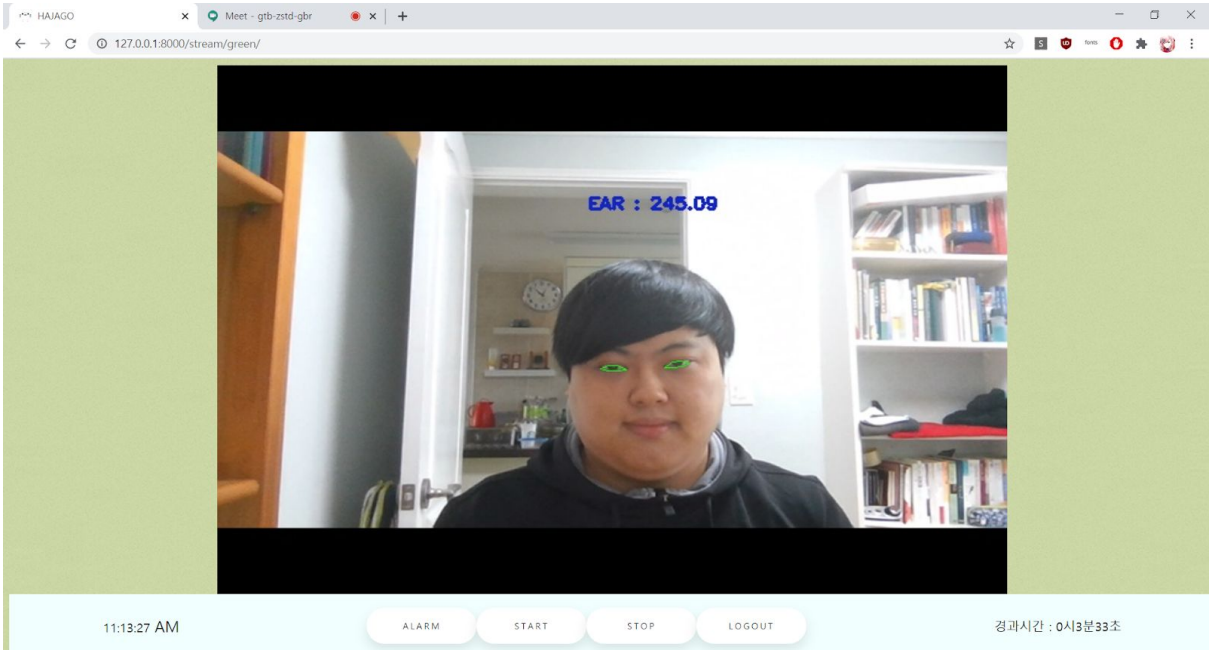
<로그인 화면>



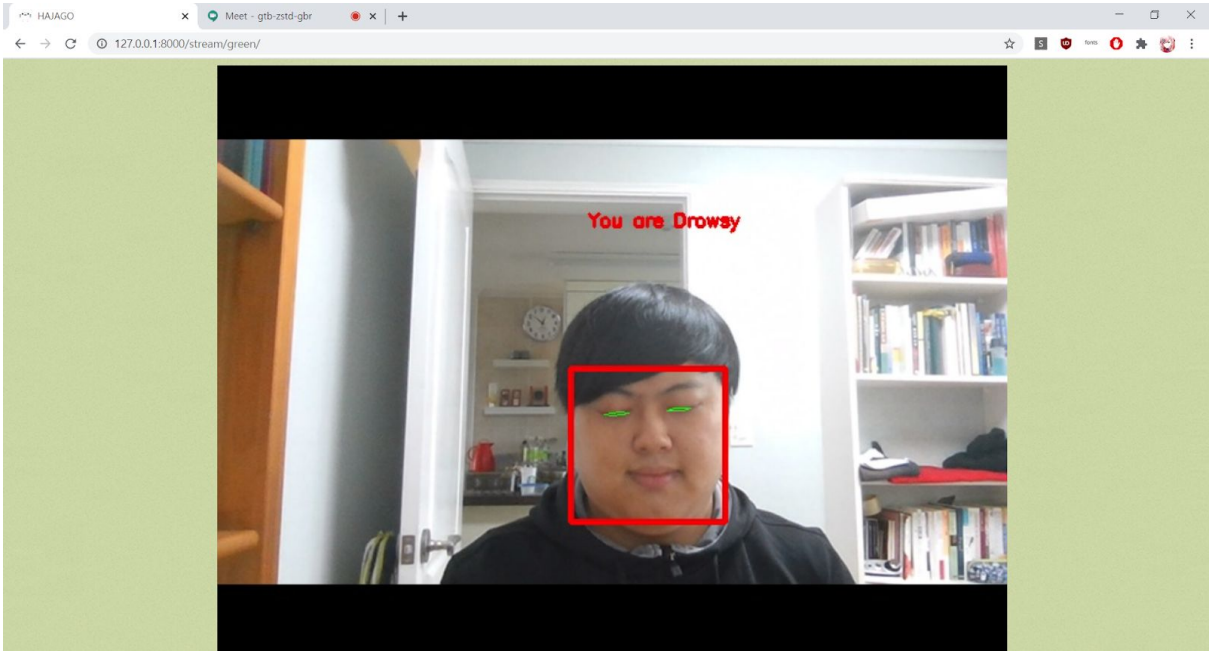
<회원 가입 화면>



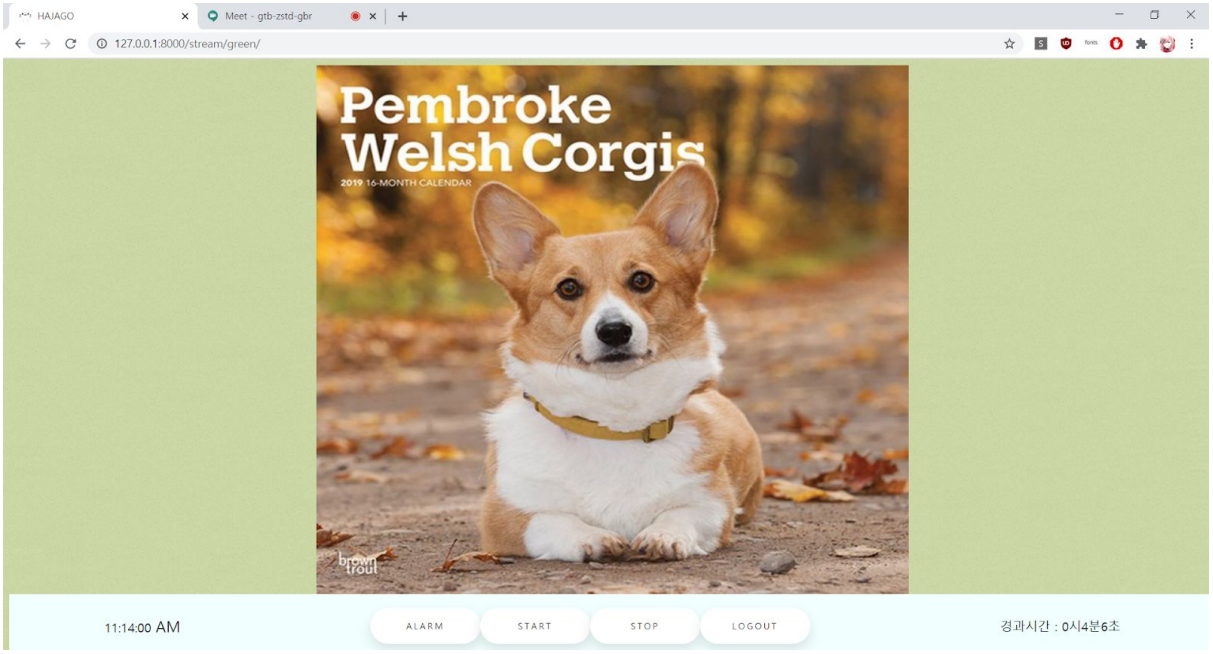
- 로그인 후 메인 서비스를 이용하는 화면이다. 밑의 alarm 버튼을 통해 현재 TTS를 들어보거나 바꿀 수 있고 start 버튼을 통해 졸음 방지 서비스를 이용할 수 있다.



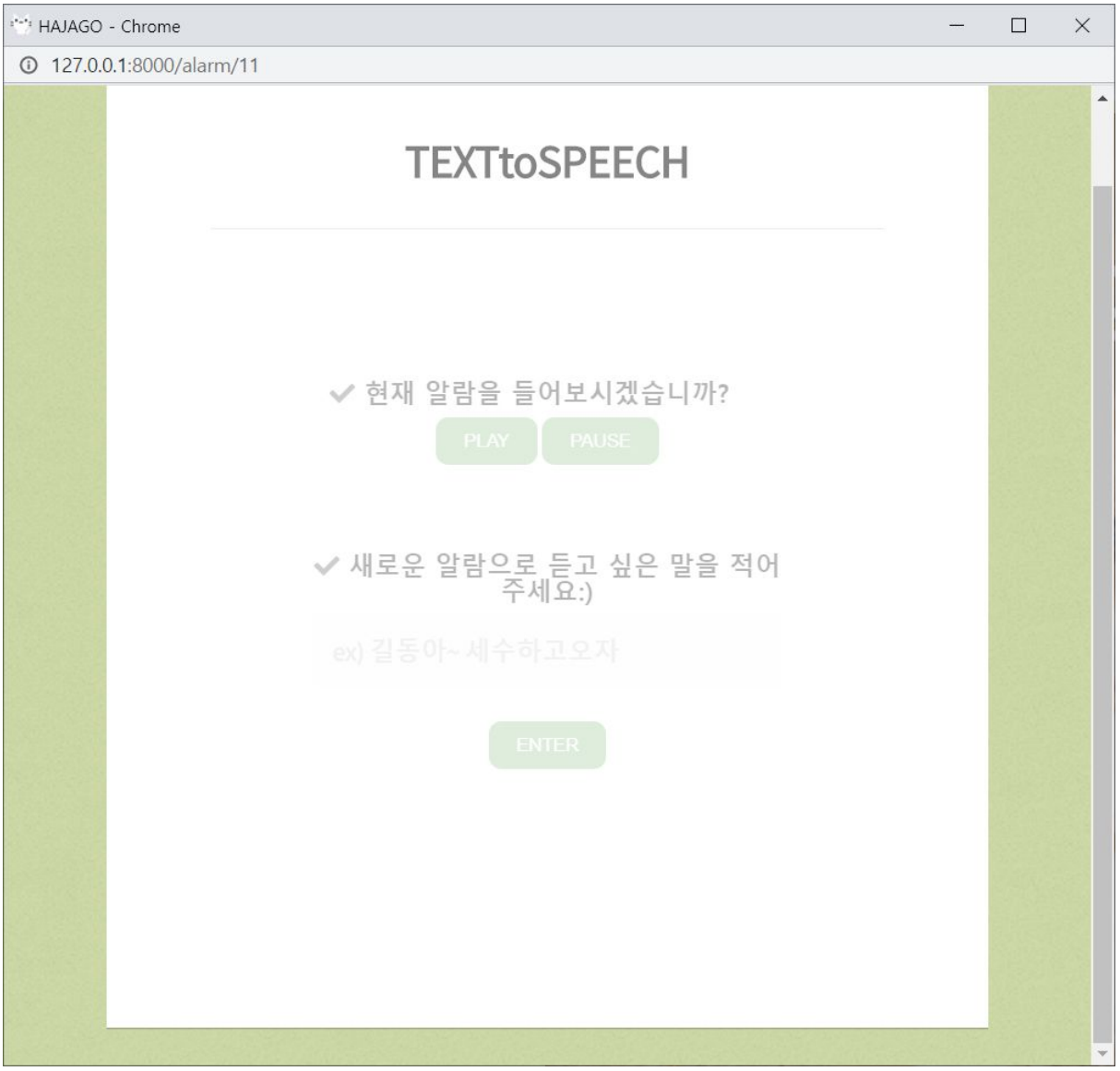
- 서비스를 이용하기 시작하면 맨 처음 얼굴을 제대로 인식하기 위해 안내 문구가 나온 후 본격적으로 시작된다.



- 눈을 감았을 때 정상적으로 작동되는 것을 확인할 수 있다.



- stop버튼을 통해 잠시 멈출 수 있고 logout 버튼을 통해 서비스를 종료할 수 있다.



- alarm 버튼을 눌렀을 때 뜨는 창이다. 이 창에서 현재 TTS 를 들어보거나 다른 TTS로 변경할 수 있다.

4. 참고 문서

Opencv 기반 졸음감지 : https://github.com/mohitwildbeast/Driver-Drowsiness-Detector

dlib 라이브러리: http://dlib.net/files/shape_predictor_68_face_landmarks.dat.bz2

Openpose: https://github.com/CMU-Perceptual-Computing-Lab/openpose

kakao developer: https://developers.kakao.com/docs/latest/ko/voice/rest-api

Tensorflow를 이용한 TTS: https://github.com/TensorSpeech/TensorflowTTS

딥러닝을 이용한 TTS: https://blog.crux.cx/iu-siri-1/