Video to Subtitle:

import os

import logging

import threading

import time

import whisper

import moviepy.editor as mp

from flask import Flask, request, jsonify, render\_template, send\_from\_directory

from flask\_socketio import SocketIO

from flask\_cors import CORS

app = Flask(\_name\_, static\_folder="uploads")

socketio = SocketIO(app, cors\_allowed\_origins="\*")

CORS(app)

UPLOAD\_FOLDER = "uploads"

if not os.path.exists(UPLOAD\_FOLDER):

    os.makedirs(UPLOAD\_FOLDER)

# Load Whisper model once for faster processing

whisper\_model = whisper.load\_model("base")

# Ensure the logging is set up

logging.basicConfig(level=logging.INFO)

def extract\_audio(video\_path, audio\_path="uploads/audio.wav"):

    """Extracts audio from a video file."""

    video = mp.VideoFileClip(video\_path)

    video.audio.write\_audiofile(audio\_path)

    return audio\_path

def stream\_subtitles(audio\_path):

    """Streams subtitles in real-time using Whisper."""

    result = whisper\_model.transcribe(audio\_path, word\_timestamps=True)

    for segment in result["segments"]:

        start\_time = segment["start"]

        end\_time = segment["end"]

        text = segment["text"]

        # Emit subtitle data with proper start and end times

        socketio.emit("subtitle", {"start": start\_time, "end": end\_time, "text": text})

        time.sleep(end\_time - start\_time)  # Simulate real-time streaming

@app.route("/")

def index():

    return render\_template("index.html")

@app.route("/upload", methods=["POST"])

def upload\_video():

    """Handles video upload and starts real-time subtitle generation."""

    if "video" not in request.files:

        logging.error("No video file part in the request")

        return jsonify({"error": "No video uploaded"}), 400

    file = request.files["video"]

    if file.filename == "":

        logging.error("No selected file")

        return jsonify({"error": "No file selected"}), 400

    try:

        video\_path = os.path.join(UPLOAD\_FOLDER, file.filename)

        file.save(video\_path)

        logging.info(f"Video saved at: {video\_path}")

        audio\_path = extract\_audio(video\_path)

        # Run subtitle streaming in a separate thread

        threading.Thread(target=stream\_subtitles, args=(audio\_path,)).start()

        return jsonify({"video\_url": f"/uploads/{file.filename}"}), 200

    except Exception as e:

        logging.error(f"Error saving video: {str(e)}")

        return jsonify({"error": "Error saving video"}), 500

@app.route("/uploads/<filename>")

def uploaded\_file(filename):

    """Serves uploaded videos."""

    return send\_from\_directory(UPLOAD\_FOLDER, filename)

if \_name\_ == "\_main\_":

    socketio.run(app, debug=True)

**Text to Signlanguage:**

**Views.py**

from django.http import HttpResponse

from django.shortcuts import render, redirect

from django.contrib.auth.forms import UserCreationForm, AuthenticationForm

from django.contrib.auth import login,logout

from nltk.tokenize import word\_tokenize

from nltk.corpus import stopwords

from nltk.stem import WordNetLemmatizer

import nltk

from django.contrib.staticfiles import finders

from django.contrib.auth.decorators import login\_required

def home\_view(request):

    return render(request,'home.html')

def about\_view(request):

    return render(request,'about.html')

def contact\_view(request):

    return render(request,'contact.html')

@login\_required(login\_url="login")

def animation\_view(request):

    if request.method == 'POST':

        text = request.POST.get('sen')

        #tokenizing the sentence

        text.lower()

        #tokenizing the sentence

        words = word\_tokenize(text)

        tagged = nltk.pos\_tag(words)

        tense = {}

        tense["future"] = len([word for word in tagged if word[1] == "MD"])

        tense["present"] = len([word for word in tagged if word[1] in ["VBP", "VBZ","VBG"]])

        tense["past"] = len([word for word in tagged if word[1] in ["VBD", "VBN"]])

        tense["present\_continuous"] = len([word for word in tagged if word[1] in ["VBG"]])

        #stopwords that will be removed

        stop\_words = set(["mightn't", 're', 'wasn', 'wouldn', 'be', 'has', 'that', 'does', 'shouldn', 'do', "you've",'off', 'for', "didn't", 'm', 'ain', 'haven', "weren't", 'are', "she's", "wasn't", 'its', "haven't", "wouldn't", 'don', 'weren', 's', "you'd", "don't", 'doesn', "hadn't", 'is', 'was', "that'll", "should've", 'a', 'then', 'the', 'mustn', 'i', 'nor', 'as', "it's", "needn't", 'd', 'am', 'have',  'hasn', 'o', "aren't", "you'll", "couldn't", "you're", "mustn't", 'didn', "doesn't", 'll', 'an', 'hadn', 'whom', 'y', "hasn't", 'itself', 'couldn', 'needn', "shan't", 'isn', 'been', 'such', 'shan', "shouldn't", 'aren', 'being', 'were', 'did', 'ma', 't', 'having', 'mightn', 've', "isn't", "won't"])

        #removing stopwords and applying lemmatizing nlp process to words

        lr = WordNetLemmatizer()

        filtered\_text = []

        for w,p in zip(words,tagged):

            if w not in stop\_words:

                if p[1]=='VBG' or p[1]=='VBD' or p[1]=='VBZ' or p[1]=='VBN' or p[1]=='NN':

                    filtered\_text.append(lr.lemmatize(w,pos='v'))

                elif p[1]=='JJ' or p[1]=='JJR' or p[1]=='JJS'or p[1]=='RBR' or p[1]=='RBS':

                    filtered\_text.append(lr.lemmatize(w,pos='a'))

                else:

                    filtered\_text.append(lr.lemmatize(w))

        #adding the specific word to specify tense

        words = filtered\_text

        temp=[]

        for w in words:

            if w=='I':

                temp.append('Me')

            else:

                temp.append(w)

        words = temp

        probable\_tense = max(tense,key=tense.get)

        if probable\_tense == "past" and tense["past"]>=1:

            temp = ["Before"]

            temp = temp + words

            words = temp

        elif probable\_tense == "future" and tense["future"]>=1:

            if "Will" not in words:

                    temp = ["Will"]

                    temp = temp + words

                    words = temp

            else:

                pass

        elif probable\_tense == "present":

            if tense["present\_continuous"]>=1:

                temp = ["Now"]

                temp = temp + words

                words = temp

        filtered\_text = []

        for w in words:

            path = w + ".mp4"

            f = finders.find(path)

            #splitting the word if its animation is not present in database

            if not f:

                for c in w:

                    filtered\_text.append(c)

            #otherwise animation of word

            else:

                filtered\_text.append(w)

        words = filtered\_text;

        return render(request,'animation.html',{'words':words,'text':text})

    else:

        return render(request,'animation.html')

def signup\_view(request):

    if request.method == 'POST':

        form = UserCreationForm(request.POST)

        if form.is\_valid():

            user = form.save()

            login(request,user)

            # log the user in

            return redirect('animation')

    else:

        form = UserCreationForm()

    return render(request,'signup.html',{'form':form})

def login\_view(request):

    if request.method == 'POST':

        form = AuthenticationForm(data=request.POST)

        if form.is\_valid():

            #log in user

            user = form.get\_user()

            login(request,user)

            if 'next' in request.POST:

                return redirect(request.POST.get('next'))

            else:

                return redirect('animation')

    else:

        form = AuthenticationForm()

    return render(request,'login.html',{'form':form})

def logout\_view(request):

    logout(request)

    return redirect("home")

**urls.py:**

from django.contrib import admin

from django.urls import path

from . import views

urlpatterns = [

    path('admin/', admin.site.urls),

    path('about/',views.about\_view,name='about'),

    path('contact/',views.contact\_view,name='contact'),

    path('login/',views.login\_view,name='login'),

    path('logout/',views.logout\_view,name='logout'),

    path('signup/',views.signup\_view,name='signup'),

    path('animation/',views.animation\_view,name='animation'),

    path('',views.home\_view,name='home'),

    path('animation/',views.animation\_view,name='animation')

]

**Settings.py:**

"""

Django settings for A2SL project.

Generated by 'django-admin startproject' using Django 3.0.4.

For more information on this file, see

https://docs.djangoproject.com/en/3.0/topics/settings/

For the full list of settings and their values, see

https://docs.djangoproject.com/en/3.0/ref/settings/

"""

import os

# Build paths inside the project like this: os.path.join(BASE\_DIR, ...)

BASE\_DIR = os.path.dirname(os.path.dirname(os.path.abspath(\_\_file\_\_)))

# Below code added by RanjeetKumbhar01

import nltk

NLTK\_DATA\_DIR = os.path.join(BASE\_DIR, 'nltk\_data')

nltk.data.path.append(NLTK\_DATA\_DIR)

# download nltk utilities

nltk.download('averaged\_perceptron\_tagger')

nltk.download('wordnet')

nltk.download('omw-1.4')

# Quick-start development settings - unsuitable for production

# See https://docs.djangoproject.com/en/3.0/howto/deployment/checklist/

# SECURITY WARNING: keep the secret key used in production secret!

SECRET\_KEY = '3k7=!d39#4@\_&5a6to&4=\_=j(c^v0(vv91cj5+9e8+d4&+01jb'

# SECURITY WARNING: don't run with debug turned on in production!

DEBUG = True

ALLOWED\_HOSTS = []

# Application definition

INSTALLED\_APPS = [

    'django.contrib.admin',

    'django.contrib.auth',

    'django.contrib.contenttypes',

    'django.contrib.sessions',

    'django.contrib.messages',

    'django.contrib.staticfiles',

]

MIDDLEWARE = [

    'django.middleware.security.SecurityMiddleware',

    'django.contrib.sessions.middleware.SessionMiddleware',

    'django.middleware.common.CommonMiddleware',

    'django.middleware.csrf.CsrfViewMiddleware',

    'django.contrib.auth.middleware.AuthenticationMiddleware',

    'django.contrib.messages.middleware.MessageMiddleware',

    'django.middleware.clickjacking.XFrameOptionsMiddleware',

]

ROOT\_URLCONF = 'A2SL.urls'

TEMPLATES = [

    {

        'BACKEND': 'django.template.backends.django.DjangoTemplates',

        'DIRS': ['templates',],

        'APP\_DIRS': True,

        'OPTIONS': {

            'context\_processors': [

                'django.template.context\_processors.debug',

                'django.template.context\_processors.request',

                'django.contrib.auth.context\_processors.auth',

                'django.contrib.messages.context\_processors.messages',

            ],

        },

    },

]

WSGI\_APPLICATION = 'A2SL.wsgi.application'

# Database

# https://docs.djangoproject.com/en/3.0/ref/settings/#databases

DATABASES = {

    'default': {

        'ENGINE': 'django.db.backends.sqlite3',

        'NAME': os.path.join(BASE\_DIR, 'db.sqlite3'),

    }

}

# Password validation

# https://docs.djangoproject.com/en/3.0/ref/settings/#auth-password-validators

AUTH\_PASSWORD\_VALIDATORS = [

    {

        'NAME': 'django.contrib.auth.password\_validation.UserAttributeSimilarityValidator',

    },

    {

        'NAME': 'django.contrib.auth.password\_validation.MinimumLengthValidator',

    },

    {

        'NAME': 'django.contrib.auth.password\_validation.CommonPasswordValidator',

    },

    {

        'NAME': 'django.contrib.auth.password\_validation.NumericPasswordValidator',

    },

]

# Internationalization

# https://docs.djangoproject.com/en/3.0/topics/i18n/

LANGUAGE\_CODE = 'en-us'

TIME\_ZONE = 'UTC'

USE\_I18N = True

USE\_L10N = True

USE\_TZ = True

# Static files (CSS, JavaScript, Images)

# https://docs.djangoproject.com/en/3.0/howto/static-files/

STATIC\_URL = '/static/'

STATICFILES\_DIRS = [

    os.path.join(BASE\_DIR,"assets"),

]