

Department of Artificial Intelligence and Data Learning

AUDIO TO COMMON SIGN LANGUAGE USING DJANGO AND NLTK

**N GOWRI VENKAT
(221801013)
KEERTHEVASAN TS
(221801026)**

Introduction

- This project is a **Django-based web application** that combines **natural language processing (NLP)** with **user interaction**.
- Users can input text, and the system analyzes it using **NLTK tokenization and lemmatization** techniques.
- The **tense of the sentence** is detected and adjusted accordingly.
- The application maps words to corresponding **MP4 animations**.
- It incorporates **user authentication** for secure access.

Existing Systems

- ❑ Existing systems focus on **text processing and speech generation**, such as Google TTS and IBM Watson.
- ❑ They provide **natural language understanding** but lack integration with **real-time animation**.
- ❑ These models primarily rely on **cloud infrastructure**, limiting offline capabilities.
- ❑ **Visual output** is minimal and not dynamically linked to text input.
- ❑ **High costs** and complexity reduce accessibility for basic applications.

Advantages and Disadvantages of Existing System

Limitations of Existing System:

- ❑ **No Animation Integration:** They lack support for real-time word-to-animation mapping.
- ❑ **Cloud Dependency:** Requires constant internet access, limiting offline use.
- ❑ **High Cost:** Usage costs can be prohibitive for smaller projects.
- ❑ **Complexity:** Overly complex for basic text-to-animation applications.

Features of Existing System:

- ❑ **Advanced NLP Capabilities:** Models like Google TTS and Azure LUIS offer robust natural language understanding and processing.
- ❑ **Cloud-Based:** These systems are scalable and accessible from anywhere with internet connectivity.
- ❑ **High Accuracy:** They provide accurate speech synthesis, text processing, and language understanding.
- ❑ **Well-Supported:** These models come with extensive documentation and support for developers.

Proposed System

- ❑ Text Processing: Utilizing NLTK for tokenization, POS tagging, and lemmatization of input sentences.
- ❑ Animation Rendering: Matches processed words with available animation files, displaying them on the webpage.
- ❑ Modular Design: Following Django's MVT architecture for scalability and maintainability

Advantages and Disadvantages of Proposed System

Features of Proposed System:

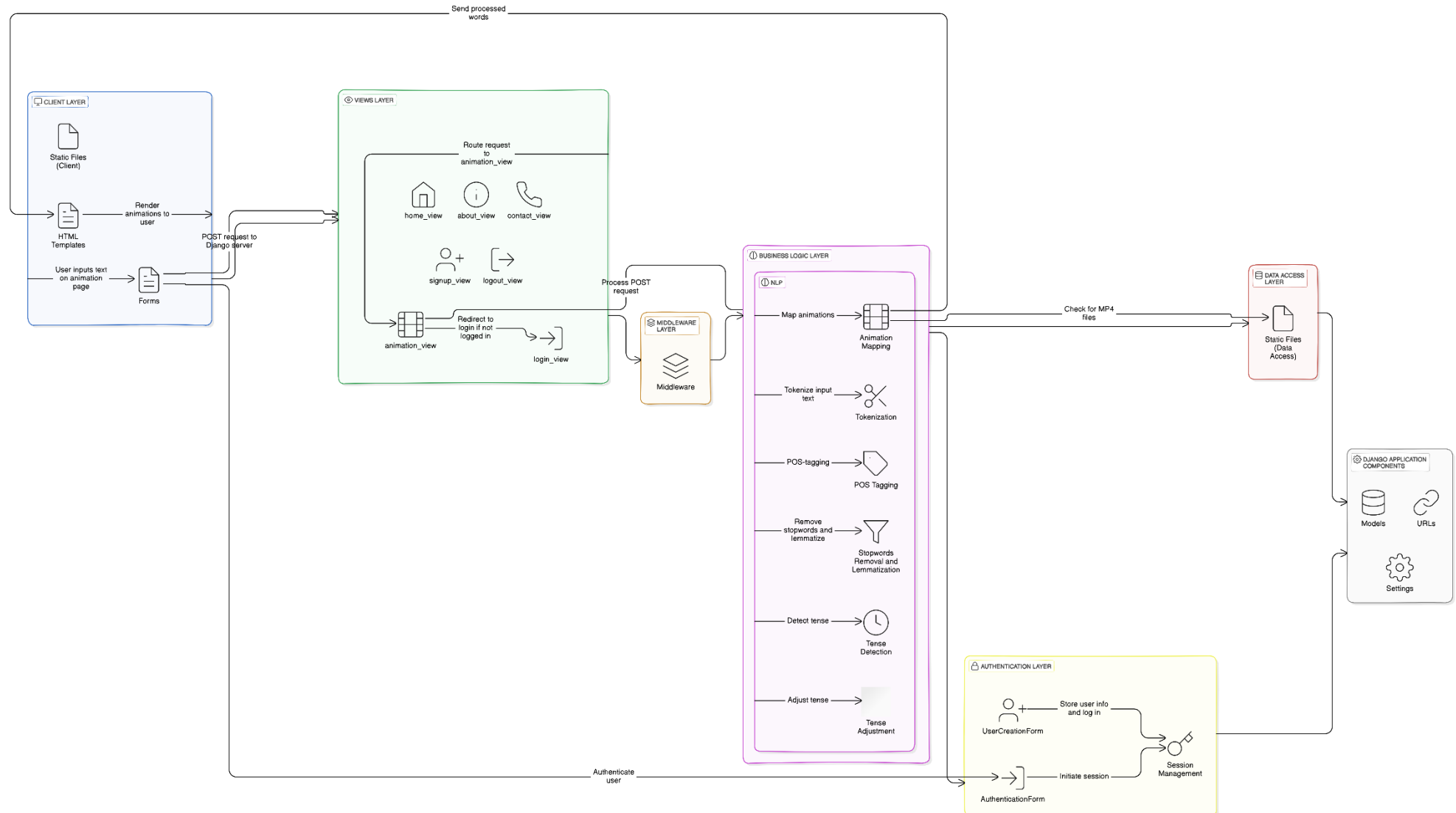
- ❑ **Modularity:** The system is divided into distinct components (views, forms, NLP processing), promoting maintainability and scalability.
- ❑ **User Authentication:** Incorporates Django's robust authentication system, ensuring secure access to protected resources.
- ❑ **Natural Language Processing :** Utilizes NLTK for sophisticated text analysis, enhancing the accuracy of text processing.
- ❑ **Customization:** The system allows for easy customization and extension, including the addition of new animations or text processing rules.

Limitations of Proposed System:

- ❑ **Resource Intensive:** NLP tasks can be computationally expensive, potentially leading to slower response times.
- ❑ **Limited Animation Availability:** The system's effectiveness relies on the availability of corresponding animations, which may not cover all words or characters.
- ❑ **Scalability Challenges:** The system may face challenges in scaling, particularly with large user bases or complex text inputs.
- ❑ **Dependency on Static Files:** The reliance on static animation files may limit dynamic content generation, reducing flexibility.

Architecture Diagram

Django Project Architecture



Software Development Model (Agile Model)

Goal	Output	Tasks
Setup and Planning	- Define user stories, MVP	- Setup Django, NLTK, and version control
	- Project structure defined	- Define core modules and requirements
Basic App Setup and Text Input	- Basic form for text input	- Set up Django project and create text input form
Text Preprocessing with NLTK	- Preprocessed text (tokenized, stopwords removed)	- Implement NLTK text preprocessing (tokenization, stopwords)
Map Text to Sign Language Animations	- Animation for each character	- Implement logic to map text to sign language animations
Display Animations in User Interface	- Animation displayed in web UI	- Integrate animations into Django views
Testing and Feedback Collection	- User feedback collected	- Conduct user testing, gather feedback, improve design
Feature Enhancements and Optimization	- Additional features and improvements	- Add features based on feedback (e.g., multi-language)

Module Description

Module 1: Django Setup and Text Input Interface

Description:

The first module involves setting up the Django web framework and creating a user interface that allows users to input text. This module focuses on handling user interaction and forms the front-end part of the system. The goal is to build a simple web page where users can type in the text they wish to convert to sign language animations.

Key Features:

- Set up a Django project and app structure.
- Create HTML templates to receive text input from the user.
- Configure URLs and views to handle the text submission.

Expected Output:

- A web page that displays a text box for users to input text.
- A button to submit the text for conversion.

Module Description

Module 2: Preprocessing Text with NLTK

Description:

This module preprocesses the text input using the Natural Language Toolkit (NLTK). The purpose of preprocessing is to clean up the text data, making it easier to map each character to its corresponding sign language animation. It involves tokenizing the input, removing special characters, and standardizing the text to lower case.

Key Features:

- Tokenize the input text to break it down into words.
- Remove punctuation and special characters.
- Convert all characters to lowercase to ensure consistency.

Expected Output:

- Cleaned and preprocessed text that is ready for the next stage of conversion.

Module Description

Module 3: Mapping Text to Sign Language Animations

Description:

In this module, the preprocessed text is mapped to a set of pre-stored sign language animations. Each character in the input text (A-Z and 0-9) has a corresponding GIF animation stored in the project's static folder. The module identifies the appropriate animation for each character and prepares them for display.

Key Features:

- Access and load pre-existing GIF animations for each character.
- Map the cleaned text to the corresponding animation files.
- Generate a sequence of animations based on the input text.

Expected Output:

- A list of file paths to the animations that correspond to each character in the input text.

Module Description

Module 4: Displaying Animations on the Web Interface

Description:

The final module handles the presentation layer by displaying the mapped animations on the web interface. Once the text has been processed and mapped to animations, the system sequentially displays each animation on the page. This provides the user with a visual representation of the text in sign language.

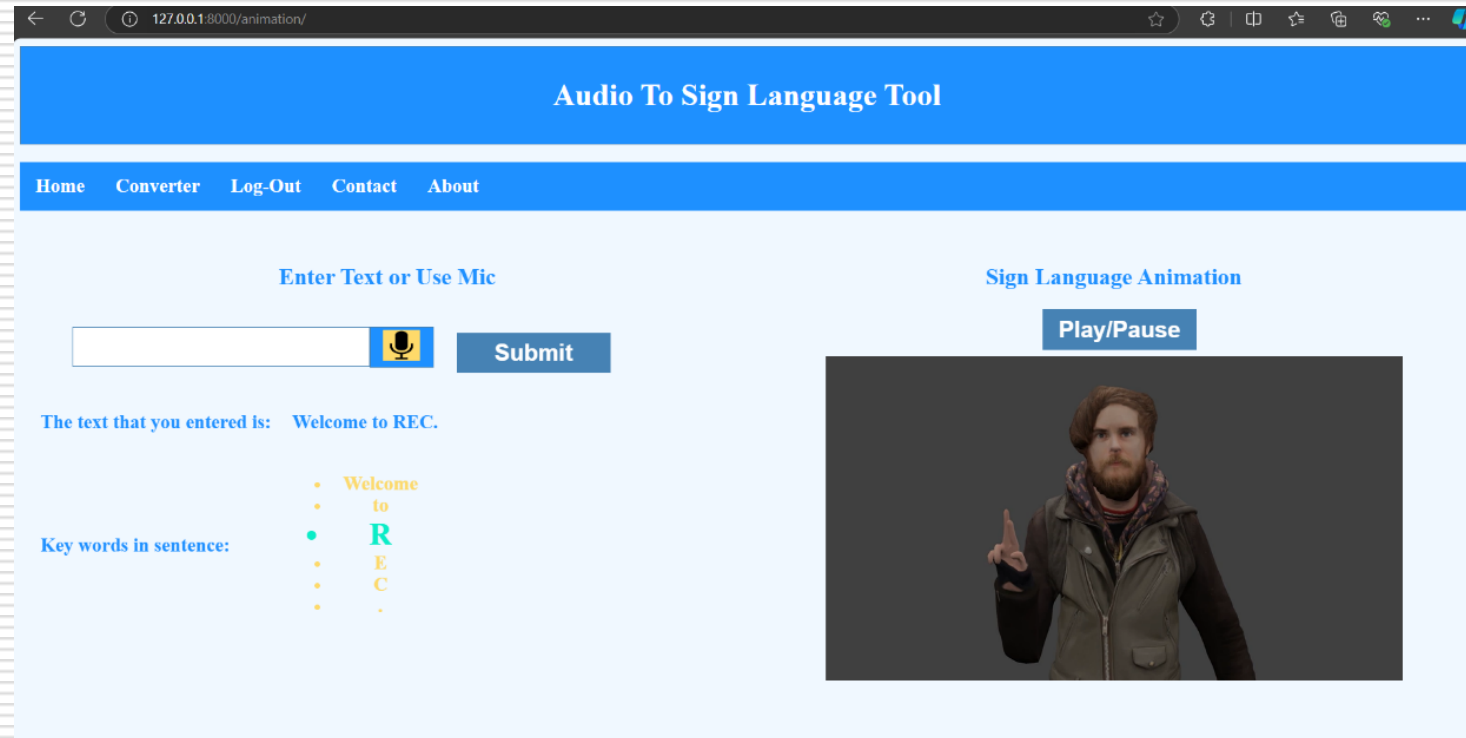
Features:

- Render the sequence of sign language animations on the web page.
- Use Django's templating system to dynamically load the animations.
- Create a visually engaging interface for users to view the converted text.

Expected Output:

- A web page that displays the sign language animations in the correct order based on the input text.
- Smooth transitions between animations for better user experience.
-

OUTPUT



Unit Testing (Authentication)

Audio To Sign Language Tool

Home Converter Sign Up Log-in Contact About

Log in

Username:

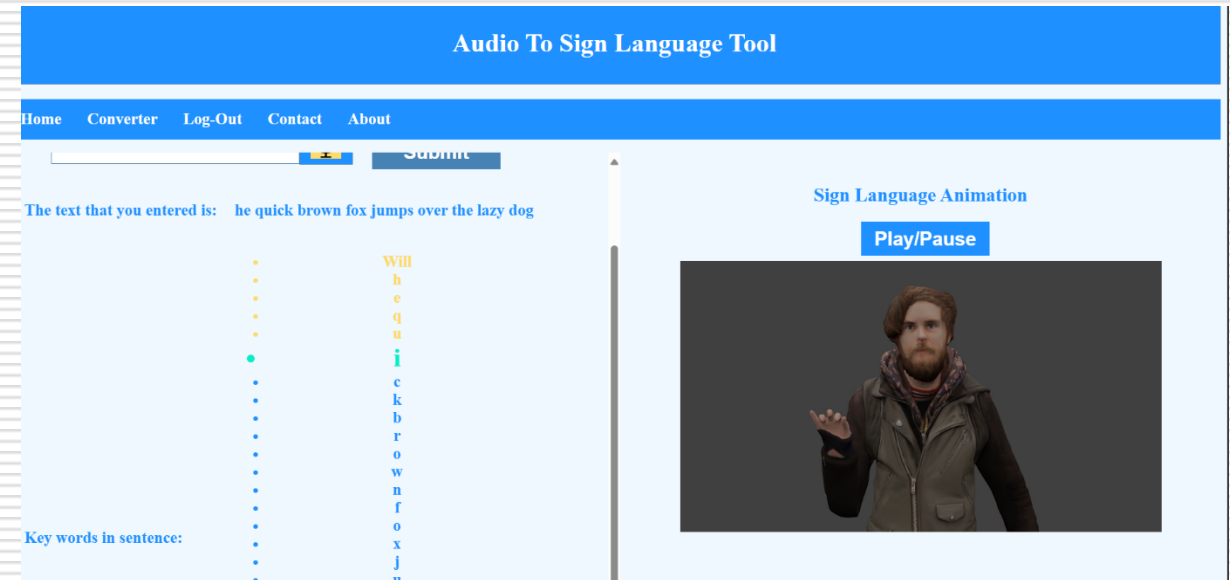
Password:

Log in

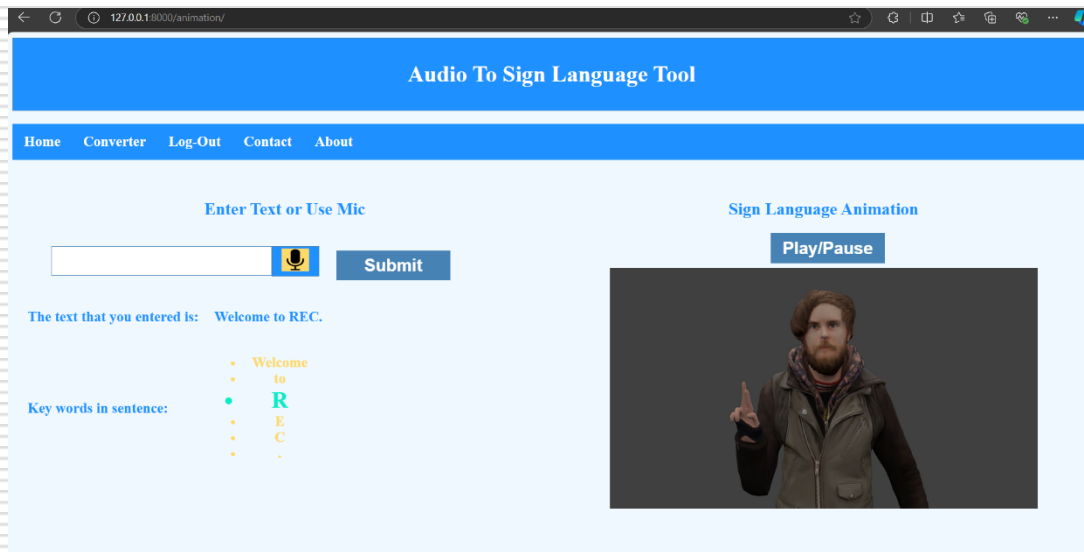
```
> Users > gowri > Downloads > test.py > AuthenticationTests > test_login
1 # tests/test_authentication.py
2 from django.test import TestCase
3 from django.contrib.auth.models import User
4
5 class AuthenticationTests(TestCase):
6     def test_user_creation(self):
7         user = User.objects.create_user(username="testuser", password="testpassword")
8         self.assertIsNotNone(user)
9
10    def test_login(self):
11        user = User.objects.create_user(username="testuser", password="testpassword")
12        login = self.client.login(username="testuser", password="testpassword")
13        self.assertTrue(login) # Verify login was successful
14
```

Unit Testing (NLTK)

```
1 # tests/test_nlp_processing.py
2 from django.test import TestCase
3 from nltk import word_tokenize, pos_tag, WordNetLemmatizer
4 from app.name.nlp module import process_text # assuming you have an NLP processing module
5
6 class NLPProcessingTests(TestCase):
7     def test_tokenization(self):
8         sentence = "The quick brown fox jumps over the lazy dog."
9         tokens = word_tokenize(sentence)
10        self.assertEqual(len(tokens), 9) # Check that we get the right number of tokens
11
12    def test_pos_tagging(self):
13        tokens = word_tokenize("I am running.")
14        tags = pos_tag(tokens)
15        self.assertIn(('running', 'VBG'), tags) # Verify that the word is tagged as verb
16
17    def test_lemmatization(self):
18        lemmatizer = WordNetLemmatizer()
19        word = lemmatizer.lemmatize("running", pos="v")
20        self.assertEqual(word, "run") # Verify correct lemmatization for verbs
21
```




Integration testing



```
1 # tests/test_integration.py
2 from django.test import TestCase
3 from app_name.models import Animation
4 from app_name.views import process_and_render_animation # hypothetical view to handle NLP and animations
5
6 class IntegrationTests(TestCase):
7     def test_nlp_to_animation_mapping(self):
8         response = self.client.post('/process_text/', {'text': 'Hello'}) # Replace URL accordingly
9         self.assertEqual(response.status_code, 200)
10        # Check if the correct animation URL is returned
11        self.assertIn("hello.mp4", response.json().get("animation_url"))
12
13    def test_authenticated_access(self):
14        # Test if authentication is required to access protected resources
15        response = self.client.get('/protected_view/')
16        self.assertEqual(response.status_code, 302) # Redirect to login
17        # Simulate login and access protected view
18        self.client.login(username="testuser", password="testpassword")
19        response = self.client.get('/protected_view/')
20        self.assertEqual(response.status_code, 200) # Access successful
21
```


OUTPUT(Module 1)

Enter Text or Use Mic




Submit

```
views.py x
A2SL > views.py > ...
26 def animation_view(request):
27
28     #removing stopwords and applying lemmatizing nlp process to words
29     lr = WordNetLemmatizer()
30     filtered_text = []
31     for w,p in zip(words,tagged):
32         if w not in stop_words:
33             if p[1]=='VBG' or p[1]=='VBD' or p[1]=='VBZ' or p[1]=='VBN' or p[1]=='NN':
34                 filtered_text.append(lr.lemmatize(w,pos='v'))
35             elif p[1]=='JJ' or p[1]=='JJR' or p[1]=='JJS' or p[1]=='RBR' or p[1]=='RBS':
36                 filtered_text.append(lr.lemmatize(w,pos='a'))
37             else:
38                 filtered_text.append(lr.lemmatize(w))
39
40     #adding the specific word to specify tense
41     words = filtered_text
42     temp=[]
43     for w in words:
44         if w=='I':
45             temp.append('Me')
46         else:
47             temp.append(w)
48     words = temp
49     probable_tense = max(tense,key=tense.get)
50
51     if probable_tense == "past" and tense["past"]>=1:
52         temp = ["Before"]
53         temp = temp + words
54         words = temp
55     elif probable_tense == "future" and tense["future"]>=1:
56         if "Will" not in words:
57             temp = ["Will"]
58             temp = temp + words
59             words = temp
60         else:
61             pass
62     elif probable_tense == "present":
63
```

OUTPUT(Module 2)

Enter Text or Use Mic

a,a,,l 

Submit

The text that you entered is: aa

Key words in sentence:

- a
- a

```
from django.http import HttpResponseRedirect
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
nltk.download('punkt_tab')
nltk.download('averaged_perceptron_tagger_eng')

def home_view(request):
    return render(request, 'home.html')

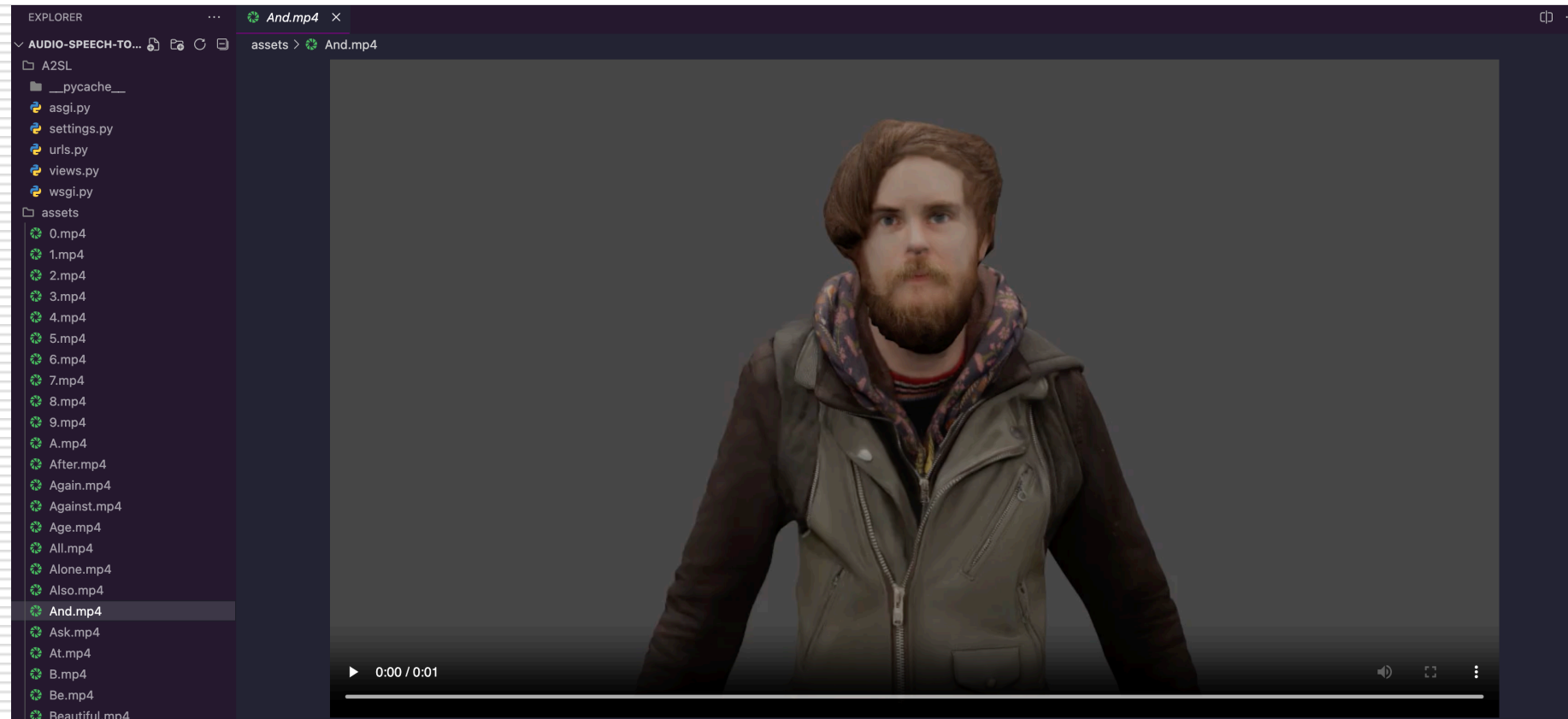
def about_view(request):
    return render(request, 'about.html')

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"]])
```

OUTPUT(Module 3)




OUTPUT(Module 4)

Audio To Sign Language Tool

[Home](#) [Converter](#) [Log-Out](#) [Contact](#) [About](#)

Enter Text or Use Mic




Submit

The text that you entered is: Walk

Key words in sentence:

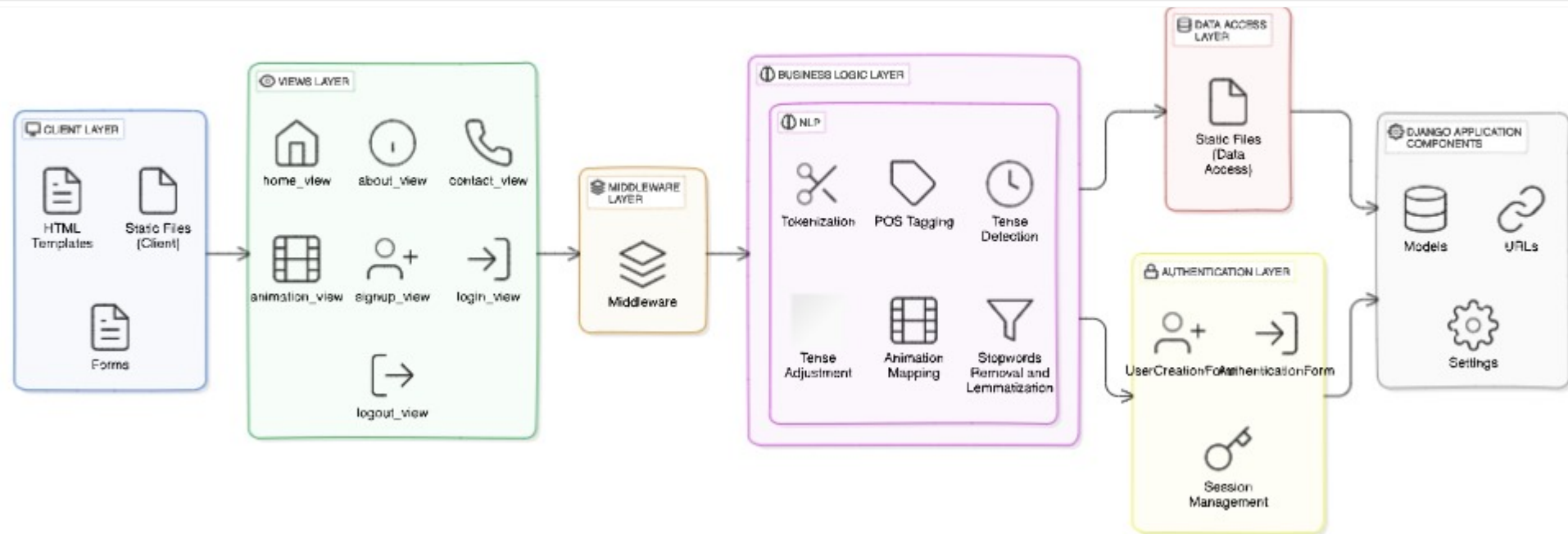
- Walk

Sign Language Animation



Play/Pause

Data Flow Diagram



References:

- [Audio to Sign Language Translation using NLP | IEEE Conference Publication | IEEE Xplore](#)
- [Audio to Indian and American Sign Language Converter using Machine Translation and NLP Technique | IEEE Conference Publication | IEEE Xplore](#)
- [NLTK Book](#)
- [Web Development with Django: A definitive guide to building modern Python web applications using Django 4 | Packt Publishing books | IEEE Xplore](#)



Thank You