

A PROJECT REPORT  
ON

# INDIAN SIGN LANGUAGE TRANSLATOR

Submitted in partial fulfillment of the requirement for the award of Degree of  
Bachelor of Technology in Computer Science & Engineering

Submitted to:



Rajasthan Technical University, Kota (Raj.)

Submitted By:

Rushikesh Bodke (17EARCS091)  
Suraj Dhara (17EARCS104)  
Rahul Pareek (17EARCS080)  
Yuvraj Gupta (17EARCS123)

Under the supervision of

PROJECT GUIDE  
**Dr. Vishal Shrivastava**  
Professor

PROJECT COORDINATOR  
**Dr. Vishal Shrivastava**  
Professor



Session: 2020-21

Department of Computer Science & Engineering  
**ARYA COLLEGE OF ENGINEERING & I.T.**  
**SP-42, RIICO INDUSTRIAL AREA, KUKAS, JAIPUR**



## **Department of Computer Science & Engineering**

### **CERTIFICATE OF APPROVAL**

The Major Project Report entitled “**Indian Sign language Translator**” submitted by “Rushikesh Bodke (17EARCS091), Suraj Dhara (17EARCS104), Rahul Pareek (17EARCS080) and Yuvraj Gupta(17EARCS123)” has been examined by us and is hereby approved for carrying out the project leading to the award of degree “**Bachelor of Technology in Computer Science & Engineering**”. By this approval the undersigned does not necessarily endorse or approve any statement made, opinion expressed or conclusion drawn therein, but approve the pursuance of project only for the above mentioned purpose.

Project Guide  
**Dr. Vishal Shrivastava**  
Professor

Project Coordinator  
**Dr. Vishal Shrivastava**  
Professor



## **Department of Computer Science & Engineering**

### **CERTIFICATE**

This is to certify that the work embodies in this Project entitled “**Indian Sign Language Translator**” being submitted by “**Rushikesh Bodke (17EARCS091), Suraj Dhara (17EARCS104), Rahul Pareek (17EARCS080) and Yuvraj Gupta(17EARCS123)**” in partial fulfillment of the requirement for the award of “**Bachelor of Technology in Computer Science & Engineering**” to Rajasthan Technical University, Kota (Raj.) during the academic year 2020-21 is a record of bonafide piece of work, carried out by them under our supervision and guidance in the “**Department of Computer Science & Engineering**”, Arya College of Engineering & Information Technology, Jaipur.

Project Guide  
**Dr. Vishal Shrivastava**  
Professor

Project-coordinator  
**Dr. Vishal Shrivastava**  
Professor

Approved by

**Dr. Akhil Pandey**  
Head, Department of  
Computer Science & Engineering



## **Department of Computer Science & Engineering**

### **DECLARATION**

We “**Rushikesh Bodke (17EARCS091), Suraj Dhara (17EARCS104), Rahul Pareek (17EARCS080) and Yuvraj Gupta(17EARCS123)**”, students of **Bachelor of Technology in Computer Science & Engineering, session 2020-21, Arya College of Engineering & Information Technology, Jaipur**, here by declare that the work presented in this Project entitled “**INDIAN SIGN LANGUAGE TRANSLATOR**” is the outcome of our own work, is bonafide and correct to the best of our knowledge and this work has been carried out taking care of Engineering Ethics. The work presented does not infringe any patented work and has not been submitted to any other University or anywhere else for the award of any degree or any professional diploma.

Rushikesh Bodke (17EARCS091)  
Suraj Dhara (17EARCS104)  
Rahul Pareek (17EARCS080)  
Yuvraj Gupta(17EARCS123)

Date : 11-08-2021

## **ACKNOWLEDGMENT**

I would like to express my sincere and profound gratitude to our project coordinator Dr. Vishal Srivastava sir for his simulating guidance, continuous encouragement and supervision throughout the course of present work.

I would like to place on record my deep sense of gratitude to our project guide Dr. Vishal Shrivastava sir for his generous guidance, help and useful suggestions.

I also wish to extend my thanks to Head of CS/IT Department Dr. Akhil Pandey sir for his insightful comments and constructive suggestions to improve the quality of this project.

I am extremely thankful to the whole CS/IT Department for providing us infrastructural facilities to work in, without which this work would not have been possible.

## **ABSTRACT**

Sign language is a visual language used by deaf and dumb people as their mother tongue. Unlike Acoustically conveyed sound patterns, sign language use body language and manual communication to fluidly convey the thoughts of a person. It can be used by a person who has difficulties in speaking or by a person who can hear but can not speak and also, by normal people to communicate with hearing disabled people. As far as a deaf person is concerned, having access to a sign language is very important for their social, emotional and linguistic growth.

Our project aims to bridge the gap between these Deaf people and normal people with the advent of new technologies of web applications, Machine Learning and Natural Language Processing. The main purpose of this project is to build an interface which accepts Audio/Voice as input and converts them to corresponding Sign Language for Deaf people. It is achieved by simultaneously combining hand shapes, orientation and movement of the hands, arms or body. The interface works in two phases, first converting Audio to Text using speech to text API (python modules or Google API) and secondly, represent the text using Parse Trees and applying the semantics of Natural Language Processing (NLTK specifically) for the lexical analysis of Sign Language Grammar.

The work builds upon the rules of ISL (Indian Sign Language) and follows the ISL rules of Grammar.

## **TABLE OF CONTENTS**

- i. Objective & Scope of the Project.
- ii. Theoretical Background.
- iii. Definition of Problem.
- iv. System Analysis & User Requirements.
- v. System Planning (PERT Chart).
- vi. Methodology adopted & Details of Hardware & Software used.
- vii. Detailed Life Cycle of the Project.
- viii. ERD, DFD.
- ix. Process involved, Algorithm, Flowchart, Database diagram.
- x. Input and Output Screen Design(Screen Shots)
- xi. Printout of the Code Sheet (Coding).
- xii. Testing.
- xiii. User/Operational Manual (including security aspects, access rights, back up, controls, etc.)
- xiv. Conclusions
- xv. Future enhancement.
- xvi. References.

## **Objective and Scope**

Communication is one of the basic requirement for survival in society. Deaf and dumb people communicate among themselves using sign language but normal people find it difficult to understand their language. Extensive work has been done on American sign language recognition but Indian Sign Language (ISL) differs significantly from American sign language.

ISL uses two hands for communicating (20 out of 26) whereas ASL uses single hand for communicating. Using both hands often leads to obscurity of features due to overlapping of hands. In addition to this, lack of datasets along with variance in sign language with locality has resulted in restrained efforts in ISL gesture detection. Our project aims at taking the basic step in bridging the communication gap between normal people and deaf and dumb people using Indian Sign Language. Effective extension of this project to words and common expressions may not only make the deaf and dumb people communicate faster and easier with outer world, but also provide a boost in developing autonomous systems for understanding and aiding them.

A sign language (SL) is a natural visual-spatial language that uses the three-dimensional space to articulate linguistic utterances instead of sound to convey meaning, simultaneously combining hand-shapes, orientation and movement of the hands, arms, upper body and facial expressions to express the linguistic message. The language came into existence because of the deaf, dumb and hard of hearing people in India. All around the world there are different communities of deaf and dumb people and thus the language of these communities are different. Just like there are many spoken languages in the world like English, French, and Chinese etc. Similarly there are different sign languages and different expressions used by hearing disabled people worldwide. The Sign Language used in USA is American Sign Language (ASL); British Sign Language (BSL) is used in Britain; and Indian Sign Language (ISL) is used in India for expressing thoughts and communicating with each other. The interactive systems are already developed for many sign language e.g. for ASL and BSL etc.



The main problem that our project aims to solve is the communication problem between speech-impaired people and the others. As those people cannot express themselves with the words, they have many difficulties during their daily life. Since almost all of the normal people do not know sign language and cannot understand what speechless people mean by their special language, tasks such as shopping, settling affairs at a public or private spaces are so difficult that speech-impaired people cannot handle it their own.

This problem is very broad and many solutions and implementation can be raised. A solution could be teaching sign language to everyone, yet it is very obvious to be an inefficient and even non-applicable one. Since speechless people can understand other people by lip-reading (or even hearing since being speechless does not mean being deaf also) the main problem is that normal people do not understand them. Thus a more suitable solution should be in the manner that makes speechless people's language understandable by the other people. For the language of speechless people to be understood by others. We need common people to understand them and communicate them in a better way and not only this but also learn some simple sign languages by seeing this.

INDIAN SIGN LANGUAGE TRANSLATOR will help us to bridge the gap between speechless people and common people. It would help us to be medium between them by converting their audio into sign language. It can breakdown your audio into words through Google API and can convert each word to sign language from its vast library. Not only can this words which can be recognized be directly converted into sign language without breaking them into alphabets.

Secondly, represent the text using Parse Trees and applying the semantics of Natural Language Processing (NLTK specifically) for the lexical analysis of Sign Language Grammar.

## **Theoretical Background**

### **Direct Translation Systems**

These systems perform their processing on the individual words of the source language string; translating is achieved without performing any form of syntactic analysis on the original input primer. Generally the word order of the sign language remains the same as that of the English primer. But in the case of English to Indian Sign Language, target sign language may not allow the same word order. With this, system also assumes a strong knowledge of both the English as well as the target sign language. Systems rested on this approach are TESSA and SignSynth system.

### **Transfer Systems**

These systems break down the input manual to some syntactic or semantic ranking, according to that a special set of . transfer rules are employed which read the information of the source language structure and produce a syntactic or semantic structure in the target language. Thereafter, a generation element is used to . convert this verbal structure into a target language outside form. The transfer elements approach is used in manual to SL MT systems and also in text to text MT systems. Systems hung on this approach are Crew, ASL workbench and ViSiCAST translator.

### **Existing Research**

Researchers all over the world have been working on automatic generation of sign language. These automated systems take text or speech as input and produce animation for it.

### **TESSA**

Cox *et al.* (2002) had developed TESSA system based on direct translation approach. It is a speech to British Sign Language translation system which provides communication between a deaf person and a post office clerk. TESSA takes English as input text, look up each word of the English string in the English-to-Sign dictionary, concatenates those signs together, and blends them into an animation.

In this system, formulaic grammar approach is used in which a set of predefined phrases are stored for translation and translated by using a phrase look up table.

The post office clerk uses headset microphone and speech recognizer. In speech recognizer, legal phrases from the grammar are stored. When clerk speaks a phrase, speech recognizer matches it with legal stored phrases. Clerk's screen displays topics available corresponding to uttered phrase, for example, "Postage", "Bill Payments" and "Passports". From these phrases clerk select one phrase according to requirement and sign of that phrase is displayed on the screen. Because of use of a small set of sentences as templates TESSA is a very domain specific system. Currently there are around 370 phrases stored in this system .

### **SignSynth Project**

It is a prototype sign synthesis application which is under development at the University of New Mexico. It converts input text into American Sign Language. Sign synthesis and speech synthesis performs almost same task. The only difference is in the form of outputs. Thus the architectures of both of these are also almost same. Sign synthesis uses Perl scripts through the common gateway interface (CGI) for performing signing animation.

It has three main interfaces. First interface, MENU CGI offers menus for signs by which user can specify the phonological parameters. Additional menus help such users who know nothing about ASCII-Stokoe. They can directly select the hand shape, hand location and hand orientation for each hold. Second interface, ASCII-Stokoe mini-parser is for more advanced users to type with additions for timing and non-manuals. The finger spelling module helps the user to type in the Roman alphabet. This module outputs an ASCII-Stokoe tree which becomes as input to the conversion module. Conversion module further produces Web3D rotations for joints which are used. After the creation of rotations, they become the input for SignGen module. SignGen module integrates them with Web3D humanoid for creating complete web file with animation data. Then with the help of plug-in, animation is played. SignSynth is free and open-source. It has simple humanoid and Perl CGI which runs on any web server.

## **TEAM**

Zhao *et al.* (2002) had developed a system TEAM based on machine translation. It is a translation system which converts English text into American Sign Language. In this system TAG parser analyzes the input text. It involves two major steps. First step includes the translation of input English sentence into intermediate representation. It considers syntactic, grammatical and morphological information. In second step, its interpretation is performed. Its representation is done as motion representation which actually controls the human model and produce ASL signs.

This system uses gloss notations for generating intermediate representation. Firstly it analyzes the word order and then generates glosses which represent facial expressions, sentence types, and morphological information. It uses a synchronous tree adjoining grammar (STAG) to map information from English text to ASL. It is first translation system which considers visual and spatial information along with linguistic information associated with American Sign Language. It is not limited to ASL; it is expandable to other signed languages because of its flexibility.

## **The ASL Workbench**

Speers (2001) had proposed and implemented a system ASL workbench. It is a Text to ASL Machine Translation system. It analyses the input text up to an LFG-style f- structure. Its representation abstracts some of syntactic specifics from input text and replaces them with linguistic features of the text. Architecture of ASL workbench is shown in figure 2.4. Workbench system included a set of specially written rules for translating an f-structure representation of English into one for ASL.

It implements transfer module and generation module. Input for translation module is an English LFG f- structure. It is converted into an ASL f-structure using structural correspondence and performing lexical selection. The ASL f-structure becomes input to the generation module.

Generation module creates American Sign Language c-structure and p-structure corresponding to the sentence. ASL workbench performs fingerspell of the word if the lexical element is noun.

If element is other than noun then translation fails, although translator can create entry corresponding to the word in ASL lexicon. It can also create entry in transfer lexicon if necessary and re attempt the translation.

### **ViSiCAST Translator**

Safar and Marshall (2001) had developed English to British Sign Language translation system. It uses semantic level of representation for performing English analysis to the BSL generation. It includes investigation of sign language delivery using different technologies. The architecture of ViSiCAST is shown in figure 2.5. This system is user friendly. User enters English text into the system, at this stage user can change or alter the text according to the requirement. After that, at the syntactic stage inputted text is parsed with CMU (Carnegie Mellon University) link grammar parser. From this parser, an intermediate semantic representation is made in the form of Discourse Representation Structure (DRS). From this representation, morphology and syntax of sign generation is defined in Head Driven Phrase Structure Grammar (HPSG). Here, signs are shown in form of HamNoSys and can be edited.

After this, signing symbols are linked with SiGML, which describes signing notations into XML form. SiGML is easily understandable by 3D rendering software, which plays animation corresponding to inputted text.

### **Machine Translation System from Text-To-Indian Sign Language**

Dasgupta et al. (2008) had developed a system based on machine translation approach. It takes English text as input and generates signs corresponding to the inputted text. Architecture of system is illustrated in figure. Figure shows four essential modules of the system which are, input text preprocessor and parser, LFG f-structure representation, Transfer Grammar Rules, ISL Sentence Generation and ISL synthesis.

Simple English sentence is inputted to the parser. Simple sentence means which sentence has only one main verb in it. Minipar parser parses the sentence and make dependency tree. A phrase lookup table of around 350 phrases and temporal expressions is made before parsing. English morphological analyzer identifies plurality of nouns.

LFG functional structure (f-structure) encodes grammatical relation of the input sentence. It also includes the higher syntactic and functional information representation of a sentence. This information is represented as a set of attribute-value pairs. Where attribute is for name of a grammatical symbol and value is for feature possessed by the constituent. It becomes as input to the generation module which apply transfer grammar rules on it so that it could transfer source sentence into target structure. Lexical selection and word order correspondence are two main operations that are performed during generation phase. Lexical selection is done using English to ISL bilingual lexicon. For example, word like “BREAKFAST” in English is replaced by “MORNING FOOD” in ISL. ISL uses Subject-Object-Verb (SOV) word order.

**English** “I have a LAPTOP”

**ISL** “I LAPTOP HAVE”

The final Indian Sign Language structure is achieved by the addition or deletion of words and restructuring of source representation.

## **INGIT**

A project being worked upon in the Indian Institute Of Technology, Kanpur, It is a prototype system designed as a proof-of-concept for a Hindi to ISL translator in the railway reservation domain, a common public need for citizens. The system,named INGIT 1 translates input from the reservation clerk into Indian Sign Language, which can then be displayed to the ISL user. INGIT currently accepts transcribed spoken language strings as input and generates ISL-gloss strings which are converted to a graphical display via HamNoSys (Prillwitz et al 1989)

simulation. Only the utterances of the reservation clerk are translated since the deaf client can respond via the paper form.

INGIT works on strings of transcribed Hindi spoken text. A domain-specific construction grammar for Hindi, implemented in FCG, converts the input into a thin semantic structure which is input to ellipsis resolution, after which we obtain a saturated semantic structure. Depending on the type of utterance (statement, query, negation, etc) a suitable ISL-tag structure is generated by the ISL generator. This is then passed to a HamNoSys converter to generate the graphical simulation.

It is a cross model translation system from Hindi strings to Indian Sign Language for possible use in the Indian Railways reservation counters. The system translates input from the reservation clerk into Indian Sign Language, which can be then displayed to the ISL user. They have used Fluid Construction Grammar (FCG) [3], for constructing the grammar for Sign language. In this the domain-specific construction grammar for Hindi converts the input into a thin semantic structure which is an input to ellipsis resolution, after which a saturated semantic structure is obtained. Depending on the type of utterance (statement, query, negation, etc.) a suitable ISL-tag structure is generated by the ISL generator.

This is then passed to a HamNoSys [4] converter to generate the graphical simulation. For validating the system, they collected small corpus on six different days. This corpus was based on interaction with speaking clients at a computer reservation counter. They after evaluation found the interaction constituted 230 words, of which many were repeated. The vocabulary of 90 words included 10 verbs in various morphological forms (e.g. work, worked, working etc.), 9 words related to time, 12 words specific to the domain (e.g. ticket, tatkal, etc.), Other words were numerals (15), names of months (12), cities (4) and trains (4) as well as digits particles etc.

The INGIT system has three main modules:

1. Input parser
2. Ellipsis Resolution Module
3. ISL Generator (including ISL lexicon with Domain Bounded English to Indian Sign Language Translation Model)



## **Definition of Problem**

Communication is one of the basic requirement for survival in society. Deaf and dumb people communicate among themselves using sign language but normal people find it difficult to understand their language. Extensive work has been done on American sign language recognition but Indian sign language differs significantly from American sign language.

A sign language (SL) is a natural visual-spatial language that uses the three-dimensional space to articulate linguistic utterances instead of sound to convey meaning, simultaneously combining handshapes, orientation and movement of the hands, arms, upper body and facial expressions to express the linguistic message. The language came into existence because of the deaf, dumb and hard of hearing people in India. All around the world there are different communities of deaf and dumb people and thus the language of these communities will be different. Just like there are many spoken languages in the world like English, French, and Urdu etc. Similarly there are different sign languages and different expressions used by hearing disabled people worldwide. The Sign Language used in USA is American Sign Language (ASL); British Sign Language (BSL) is used in Britain; and Indian Sign Language (ISL) is used in India for expressing thoughts and communicating with each other. The interactive systems are already developed for many sign language e.g. for ASL and BSL etc.

The main problem that our project aims to solve is the communication problem between speech-impaired people and the others. As those people cannot express themselves with the words, they have many difficulties during their daily life. Since almost all of the normal people do not know sign language and cannot understand what speechless people mean by their special language, tasks such as shopping, settling affairs at a government office are so difficult that speech-impaired people cannot handle by their own.

**Solution:** INDIAN SIGN LANGUAGE TRANSLATOR will help us to bridge the gap between speechless people and common people. It would help us to be medium between them by converting their audio into sign language. It can breakdown your audio into words through Google API and can convert each word to sign language from its vast library. Not only can this words which can be recognized be directly converted into sign language without breaking them into alphabets.

Secondly, represent the text using Parse Trees and applying the semantics of Natural Language Processing (NLTK specifically) for the lexical analysis of Sign Language Grammar.

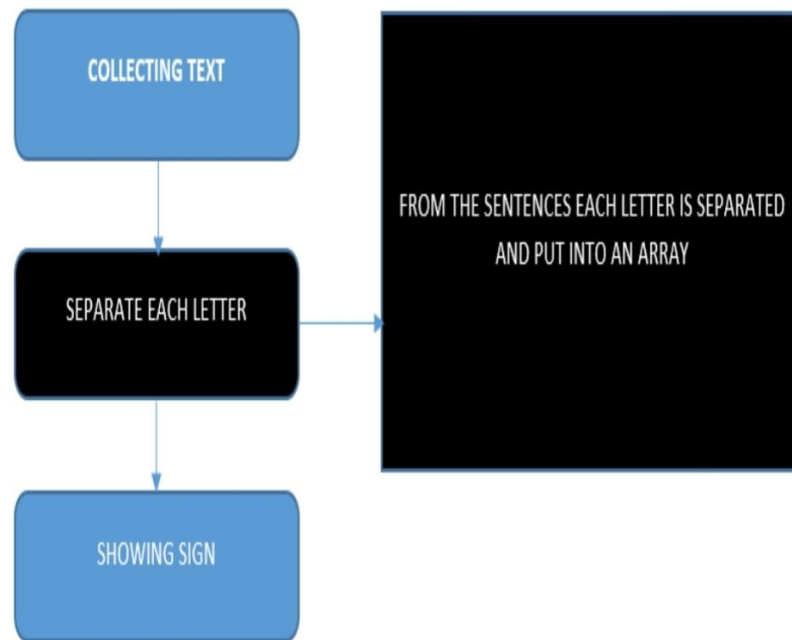
Let dive into more in deep procedure to understand the working behind our system. In this it takes audio as input, search that recording using google api, display the text on screen and finally it gives sign code of given input using ISL (Indian Sign Language) generator. All the words in the sentence are then checked against the words in the dictionary containing images and GIFs representing the words. If the words are not found, its corresponding synonym is replaced. Set of gestures are predefined in the system.

#### Procedure

1. Audio to Text Conversion: Audio input is taken using python PyAudio module. Conversion of audio to text using microphone Dependency parser is used for analyzing grammar of the sentence and obtaining relationship between words.

2. Text to Sign Language:

- Speech recognition using Google Speech API.
- Text Preprocessing using NLP.
- Dictionary based Machine Translation.
- ISL Generator: ISL of input sentence using ISL grammar rules.
- Generation of Sign language with signing Avatar



## **System Analysis & User Requirements**

Communication is one of the basic requirement for survival in society. Deaf and dumb people communicate among themselves using sign language but normal people find it difficult to understand their language.

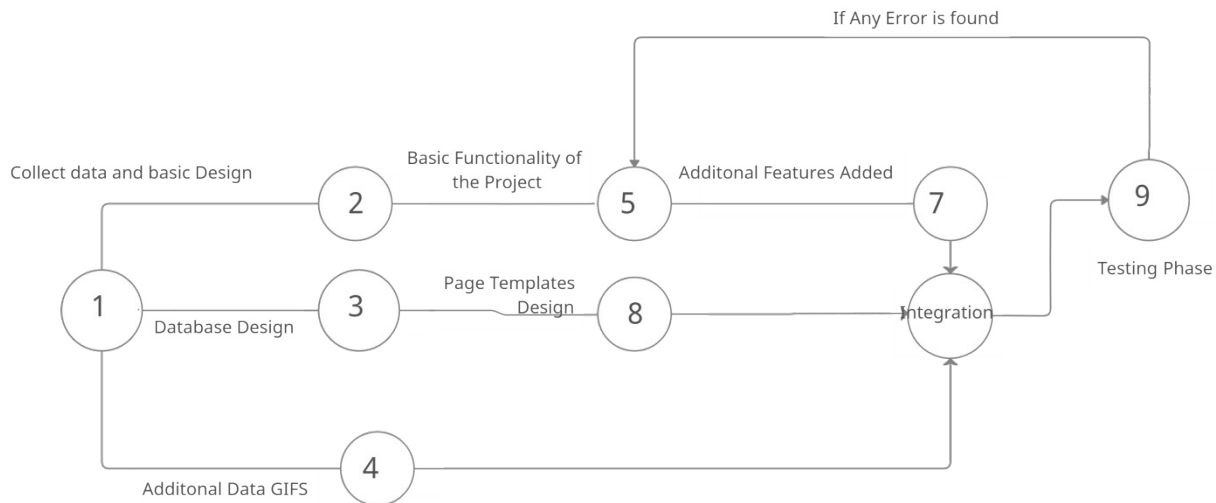
This problem is very broad and many solutions and implementation can be raised. A solution could be teaching sign language to everyone, yet it is very obvious to be an inefficient and even non-applicable one. Since speechless people can understand other people by lip-reading (or even hearing since being speechless does not mean being deaf also) the main problem is that normal people do not understand them. Thus a more suitable solution should be in the manner that makes speechless people's language understandable by the other people.

Our solution will help us to bridge the gap between speechless people and common people. It would help us to be medium between them by converting their audio into sign language. It can breakdown your audio into words through Google API and can convert each word to sign language from its vast library. Not only can this words which can be recognized be directly converted into sign language without breaking them into alphabets.

It will help us to bridge the gap between speechless people and common people. It would help us to be medium between them by converting their audio into sign language. It can breakdown your audio into words through Google API and can convert each word to sign language from its vast library. Not only can this words which can be recognized be directly converted into sign language without breaking them into alphabets.

## SYSTEM PLANNING & PERT CHART

1. Determine did you want to run it or not sign up if you are not an existing user
2. You don't need internet connection to run it
3. Login and use either Text or Audio to convert your speeches



## METHODOLOGY USED

1. **NLTK:** The **Natural Language Toolkit**, or more commonly **NLTK**, is a suite of libraries and programs for symbolic and statistical natural language processing (NLP) for English written in the Python programming language. It was developed by Steven Bird and Edward Loper in the Department of Computer and Information Science at the University of Pennsylvania. NLTK includes graphical demonstrations and sample data. It is accompanied by a book that explains the underlying concepts behind the language processing tasks supported by the toolkit, plus a cookbook.
2. **Web Speech API:** The **WebSpeechAPI** provides two distinct areas of functionality — **speech** recognition, and **speech** synthesis (also known as text to **speech**, or tts) — which open up interesting new possibilities for accessibility, and control mechanisms. The **WebSpeechAPI** enables you to incorporate voice data into web apps. The **WebSpeechAPI** makes web apps able to handle voice data. The **SpeechRecognition** interface of the **WebSpeechAPI** is the controller interface for the recognition service; this also handles the **SpeechRecognitionEvent** sent from the recognition service.
3. **PYTHON – CGI:** The Common Gateway Interface, or CGI, is a set of standards that define how information is exchanged between the web server and a custom script. The Common Gateway Interface, or CGI, is a standard for external gateway programs to interface with information servers such as HTTP servers.

When we click a hyper link to browse a particular web page or URL,

- Browser contacts the HTTP web server and demands for the URL, i.e. filename.
- Web Server parses the URL and looks for the filename. If it finds that file then sends it back to the browser, otherwise sends an error message indicating that you requested a wrong file.

4. **Google Speech to Text:** Google's Speech-to-Text accurately converts speech into text using an API powered by Google's AI technologies. It can transcribe content in real time or from stored files and deliver a better user experience in products through voice commands. It can also gain insights from customer interactions to improve the service.

Some of the key features are Speech Adaptation, Domain-specific models, Streaming Speech recognition, Speech-to-Text On-Prem.

5. **HTML :** Hypertext Markup Language (HTML) is the standard markup language for documents designed to be displayed in a web browser. It can be assisted by technologies such as Cascading Style Sheets (CSS) and scripting languages such as JavaScript. HyperText is the method by which you move around on the web — by clicking on special text called hyperlinks which bring you to the next page. The fact that it is hyper just means it is not linear — i.e. you can go to any place on the Internet whenever you want by clicking on links — there is no set order to do things in.

Markup is what HTML tags do to the text inside them.

6. **CSS: Cascading Style Sheets (CSS)** is a style sheet language used for describing the presentation of a document written in a markup language such as HTML. CSS is a cornerstone technology of the World Wide Web, alongside HTML and JavaScript.

CSS is designed to enable the separation of presentation and content, including layout, colors, and fonts. This separation can improve content accessibility, provide more flexibility and control in the specification of presentation characteristics, enable multiple web pages to share formatting by specifying the relevant CSS in a separate .css file which reduces complexity and repetition in the structural content as well as enabling the .css file to be cached to improve the page load speed between the pages that share the file and its formatting.

7. **JAVASCRIPT:** **JavaScript** often abbreviated as **JS**, is a programming language that conforms to the ECMAScript specification. JavaScript is high-level, often just-in-time compiled, and multi-paradigm. It has curly-bracket syntax, dynamic typing, prototype-based object-orientation, and first-class functions.

Alongside HTML and CSS, JavaScript is one of the core technologies of the World Wide Web. Over 97% of websites use it client-side for web page behavior, often incorporating third-party libraries. All major web browsers have a dedicated JavaScript engine to execute the code on the user's device.

As a multi-paradigm language, JavaScript supports event-driven, functional, and imperative programming styles. It has application programming interfaces (APIs) for working with text, dates, regular expressions, standard data structures, and the Document Object Model (DOM).



## **HARDWARE and SOFTWARE used**

### **Software Interface**

- OS- Windows 10
- Tools- Sublime Text Editor, Blender 3D
- Framework: Django
- Database- Sqlite
- Technologies Used – NLTK , JS WebSpeech API , HTML , CSS , JS
- Backend: Python

### **Hardware Interface :**

- Desktop

## LIFECYCLE OF PROJECT

1. **Initiation Phase:** Communication is the most important difficulty deaf and mute people face with normal people. In order to solve this major issue, we have thought of system that can be used for communication between hearing impaired people and normal people.

Our system converts the audio message into the sign language. This system takes audio as input, converts this audio recording message into text and displays the relevant Indian Sign Language images or GIFs which are predefined. By using this system, the communication between normal and deaf people gets easier.

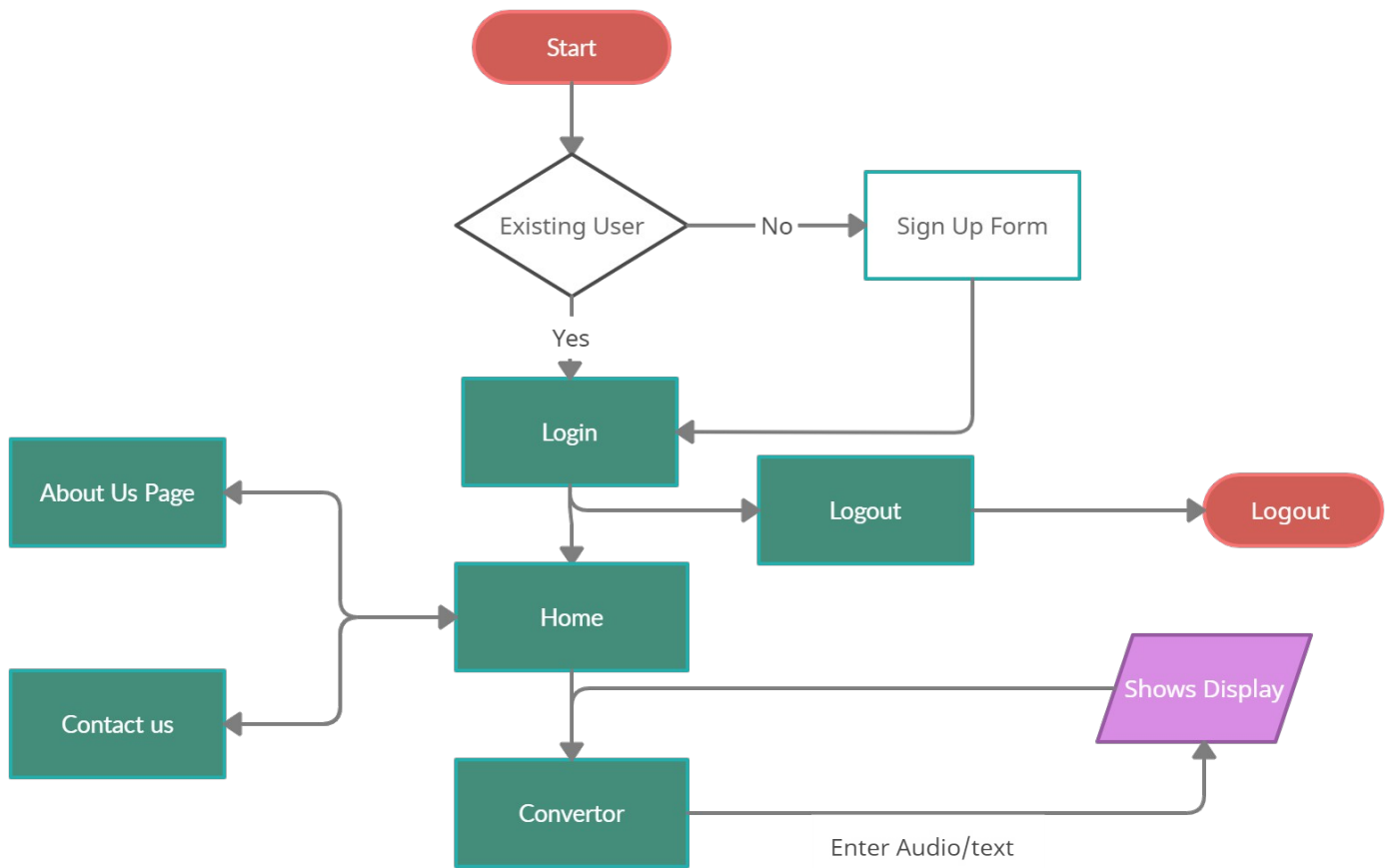
In this it takes audio as input, search that recording using google api, display the text on screen and finally it gives sign code of given input using ISL (Indian Sign Language) generator. All the words in the sentence are then checked against the words in the dictionary containing images and GIFs representing the words. If the words are not found, its corresponding synonym is replaced. Set of gestures are predefined in the system.

2. **Planning Phase:**

Sign language is communication language used by the deaf peoples using face, hands or eyes while using vocal tract. Sign language recognizer tool is used for recognizing sign language of deaf and dumb people. Gesture recognition is an important topic due to the fact that segmenting a foreground object from a cluttered background is a challenging problem.

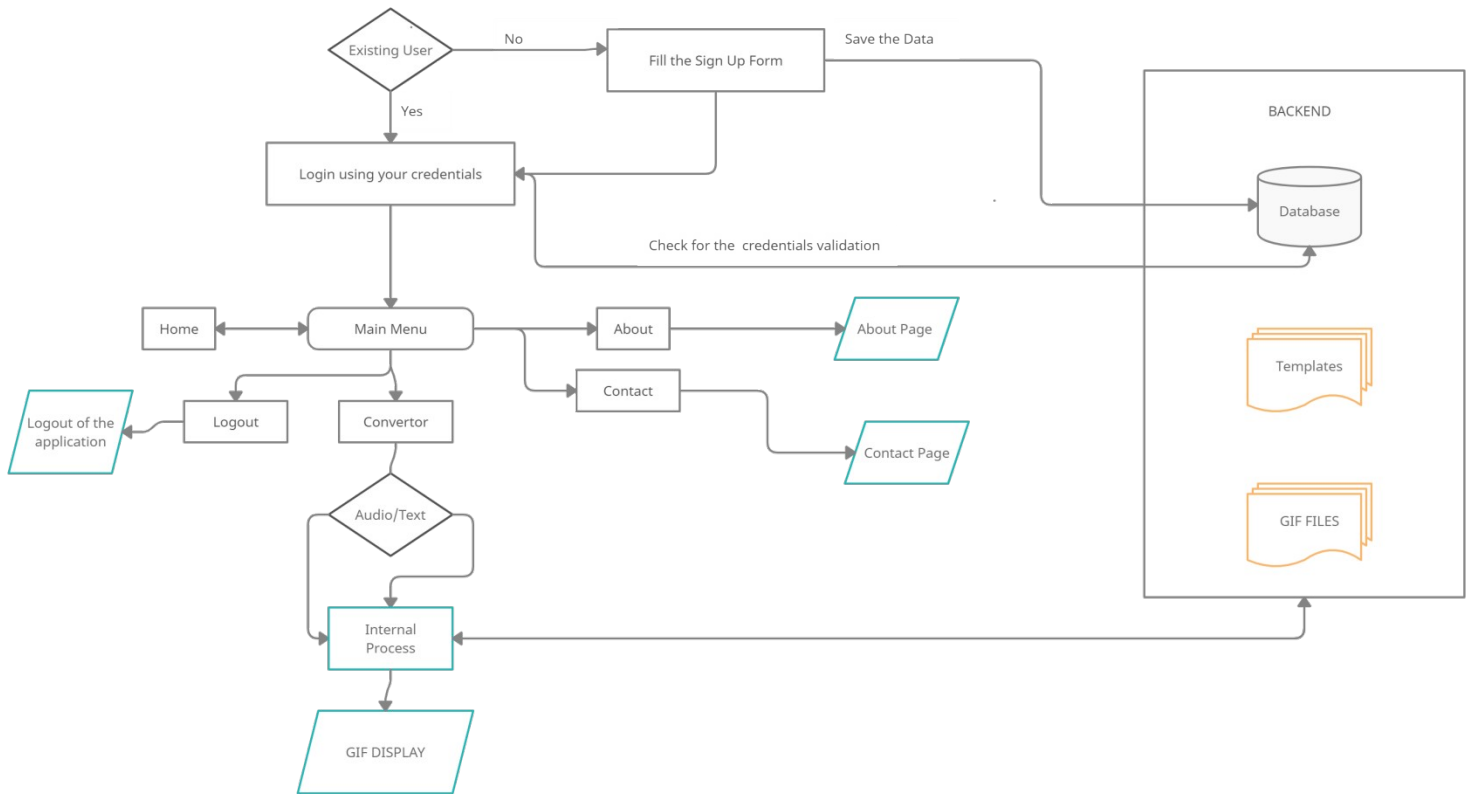
Output for a given English text is produced by generating its equivalent sign language depiction.

The output of this system will be a clip of ISL words. The predefined database will be having video for each and every separate words and the output video will be a merged video of suchwords.



Flowchart of the Project

## DATA FLOW DIAGRAM

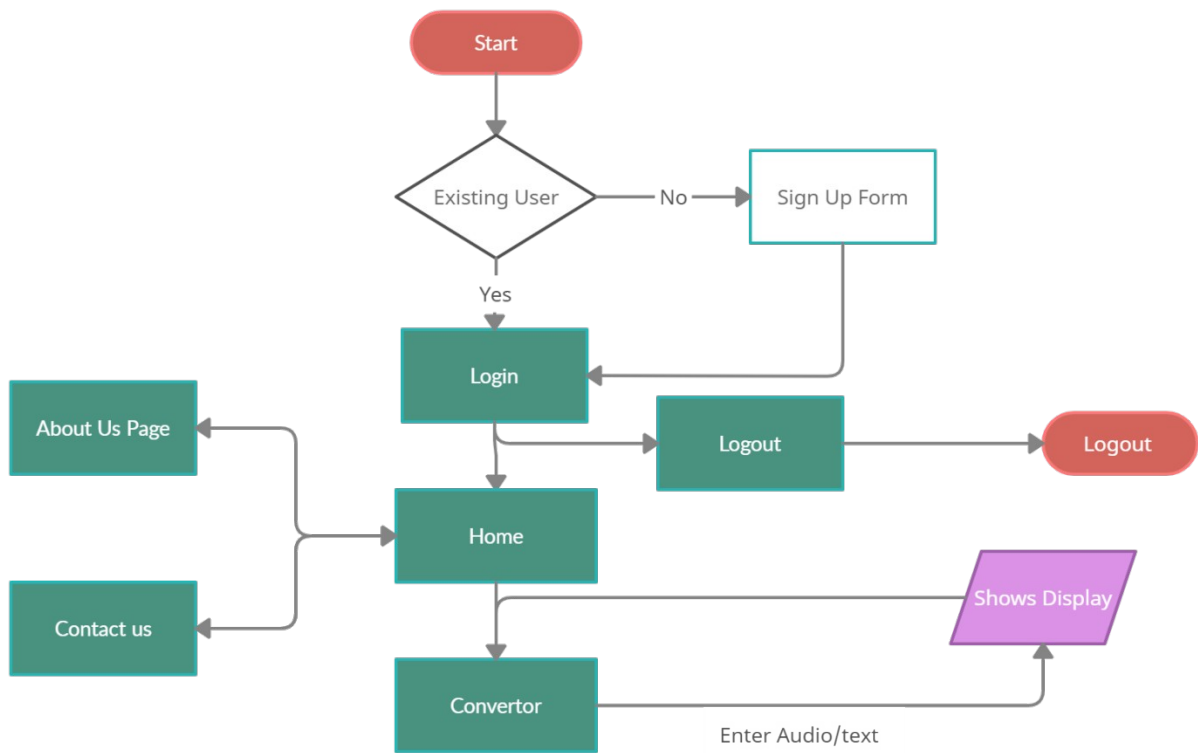


DFD

## PROCESS INVOLVED

1. The user first has to run the manage.py file in the runserver from the command line and open the localhost on a web browser using the localhost ip address/hostname.
2. File **index.html** or the homepage gets executed which displays a start button.
3. Upon clicking the start button, the user lands on the login page where the user has to enter the username and password in order to log in.
4. If the user is not already registered, then the user is taken to the sign up page where he has to enter credentials in order to sign up for the service.
5. After the user logs in or signs in, the user is taken to the main page where the user can give input in the form of either audio or text.
6. After giving the input, the user is required to submit that input to the system.
7. Then the Speech-to-Text engine converts the audio input into text.
8. After that the text is parsed by the nltk modules and converted into short Indian Sign Language words and letters.
9. Then the system plays the gif animation of hand signs and gestures which displays the input entered into a visual form.
10. If the user wants to watch the hand signs animation again, he can just click the play button again without having to enter the input again.

## FLOW CHART:



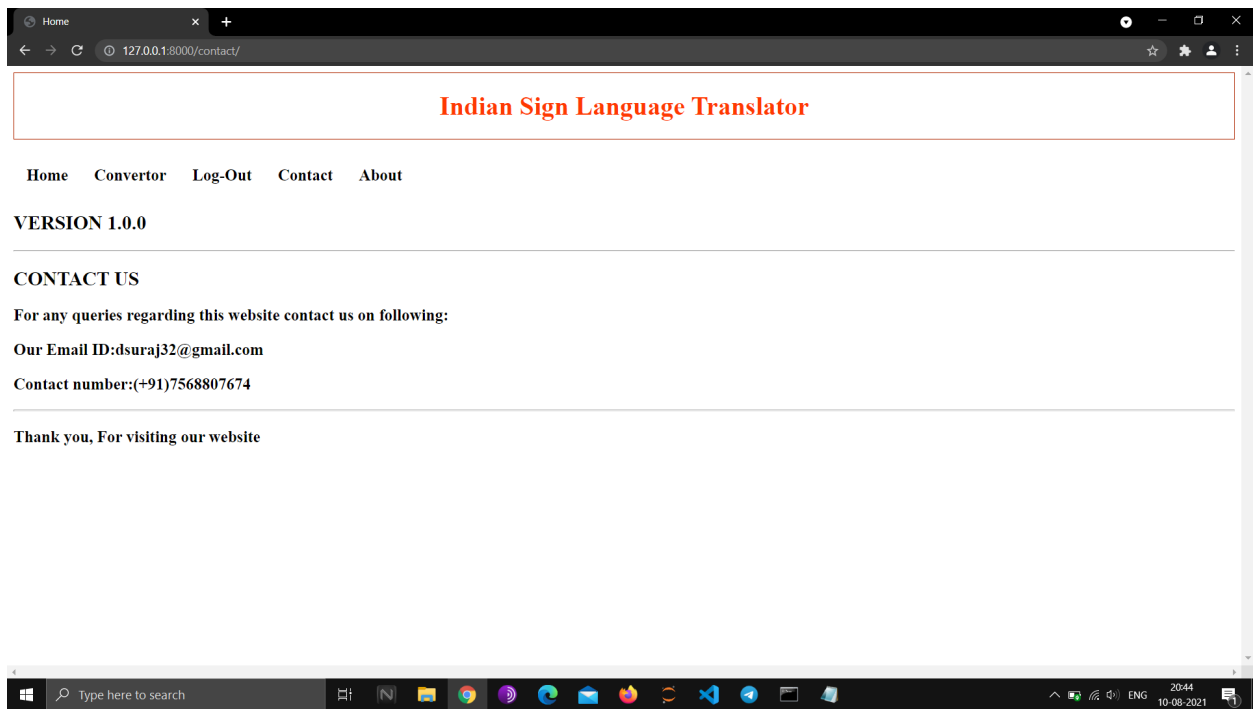
# INPUT/OUTPUT SCREEN DESIGN

The screenshot shows a web browser window with the address bar displaying "127.0.0.1:8000/login/". The page has a header with the title "Indian Sign Language Translator" in red. Below the header is a navigation menu with links: Home, Converter, Sign Up, Log-in, Contact, and About. The main content area is titled "Log in" in red. It contains two input fields: "Username:" and "Password:". Below these fields is a red "Log in" button. The Windows taskbar at the bottom shows the search bar and various application icons.

Figure 1: Login

The screenshot shows a web browser window with the address bar displaying "127.0.0.1:8000/signup/". The page has a header with the title "Indian Sign Language Translator" in red. Below the header is a navigation menu with links: Home, Converter, Sign Up, Log-in, Contact, and About. The main content area is titled "Sign Up" in red. It contains three input fields: "Username:", "Password:", and "Password confirmation:". Below these fields is a red "Sign Up" button. A note below the "Password confirmation:" field reads "Enter the same password as before, for verification." The Windows taskbar at the bottom shows the search bar and various application icons.

Figure 2: Sign Up



*Figure 3:Contact Us Page*



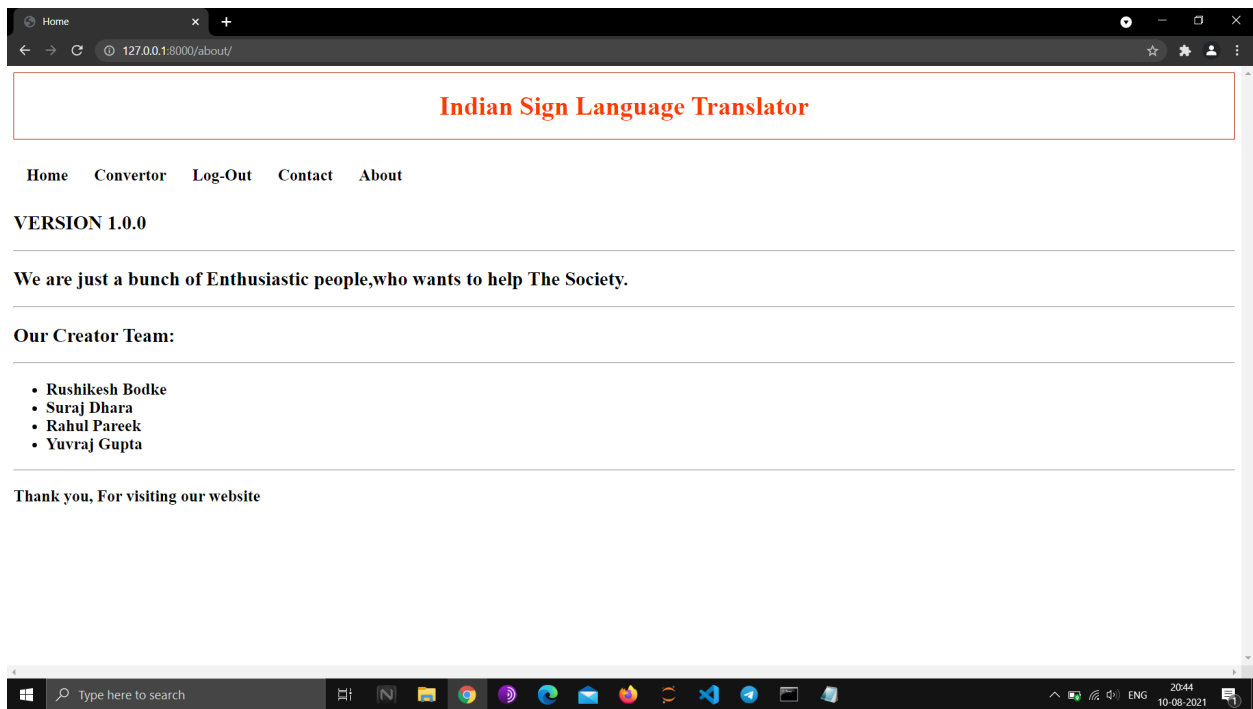


Figure 4: ABOUT us Page

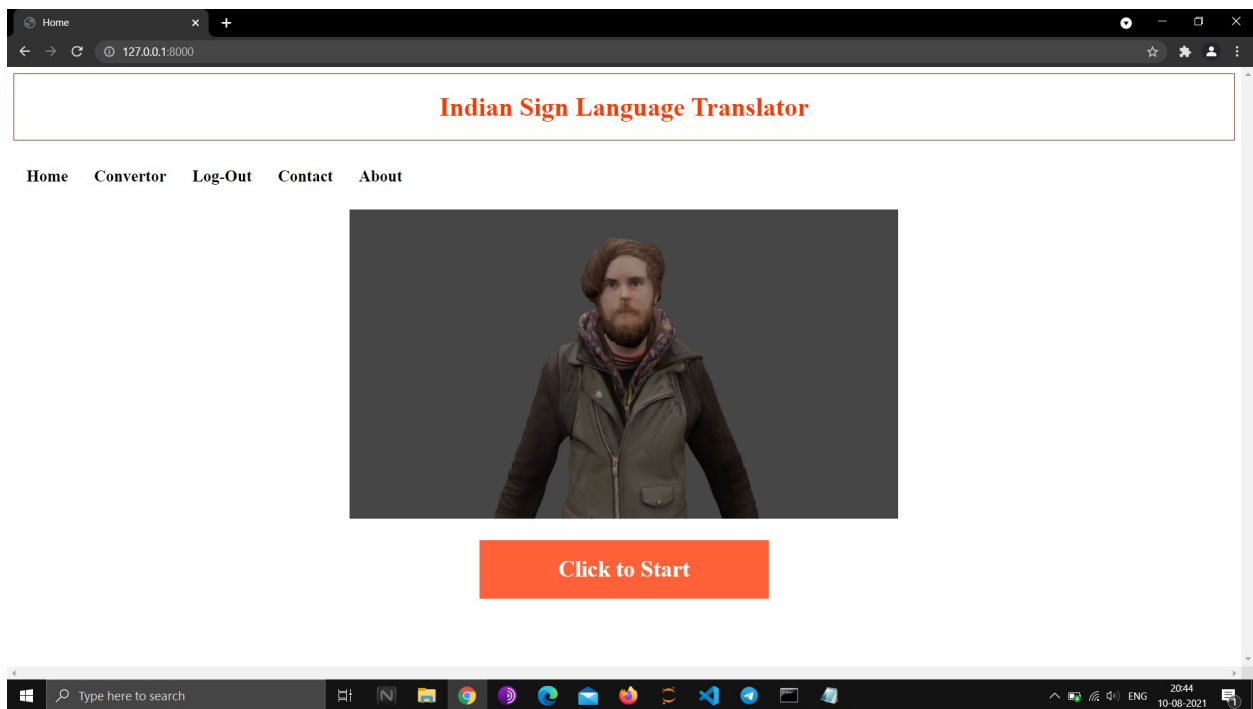
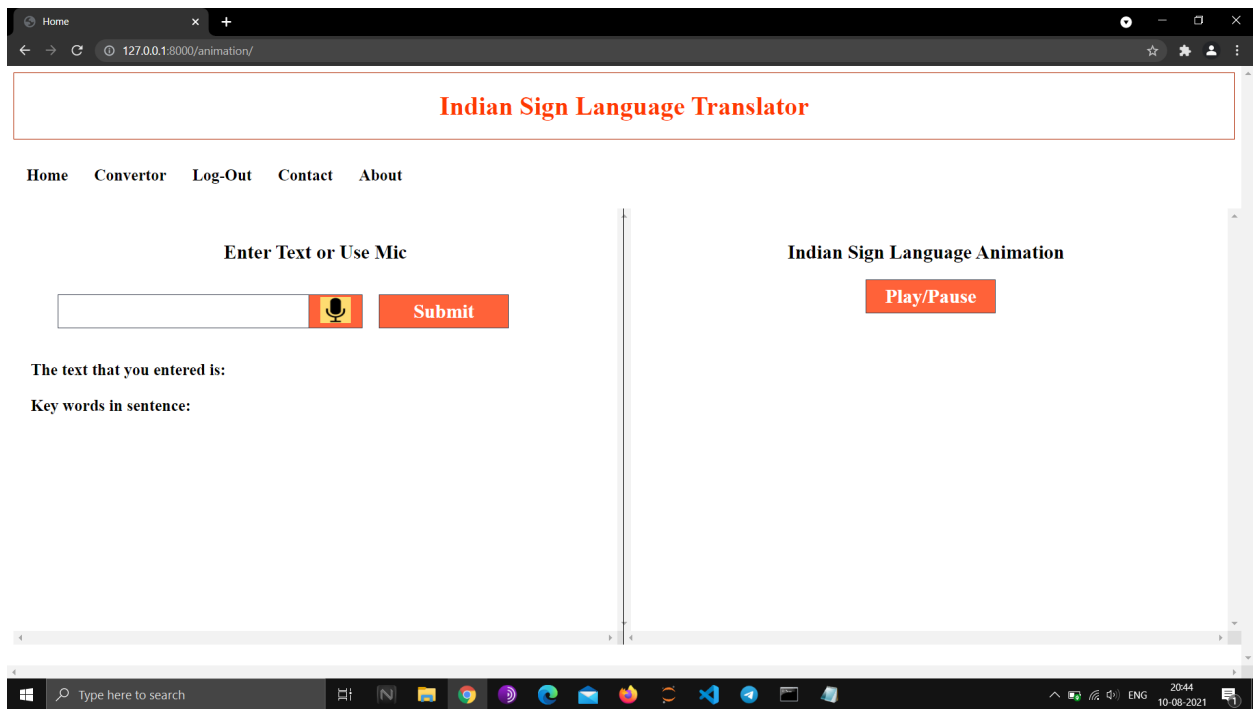
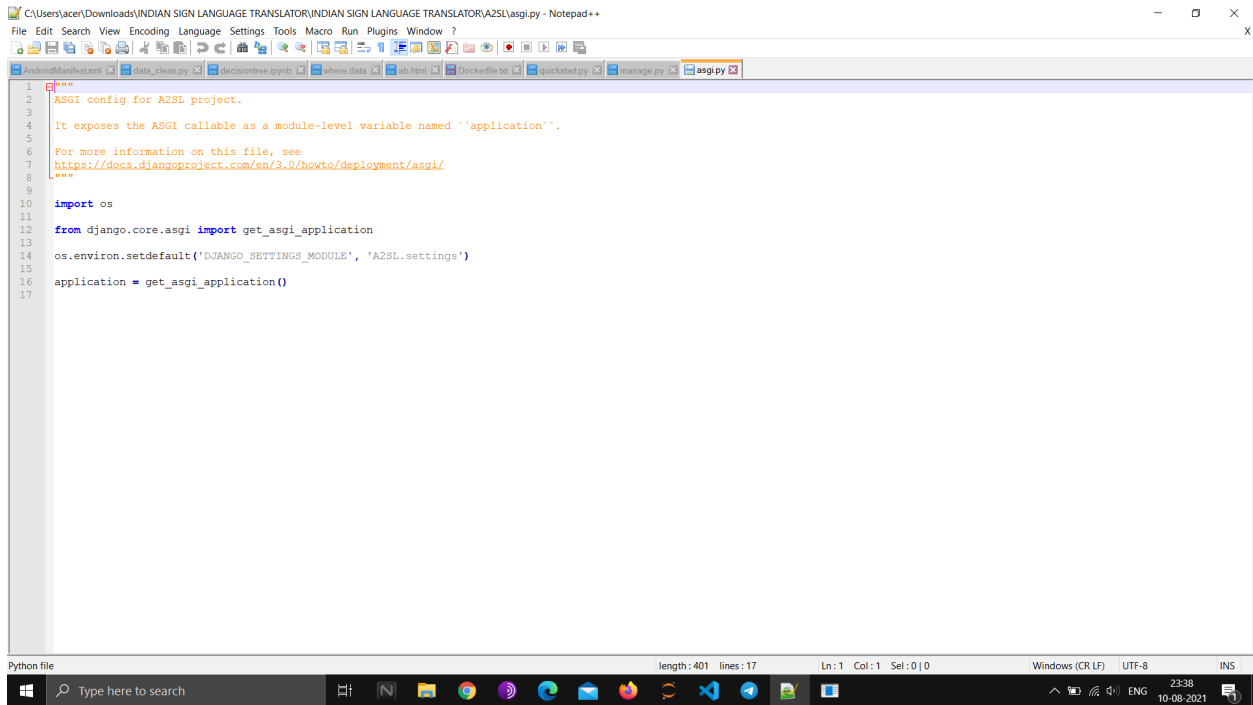


Figure 5: Home Page



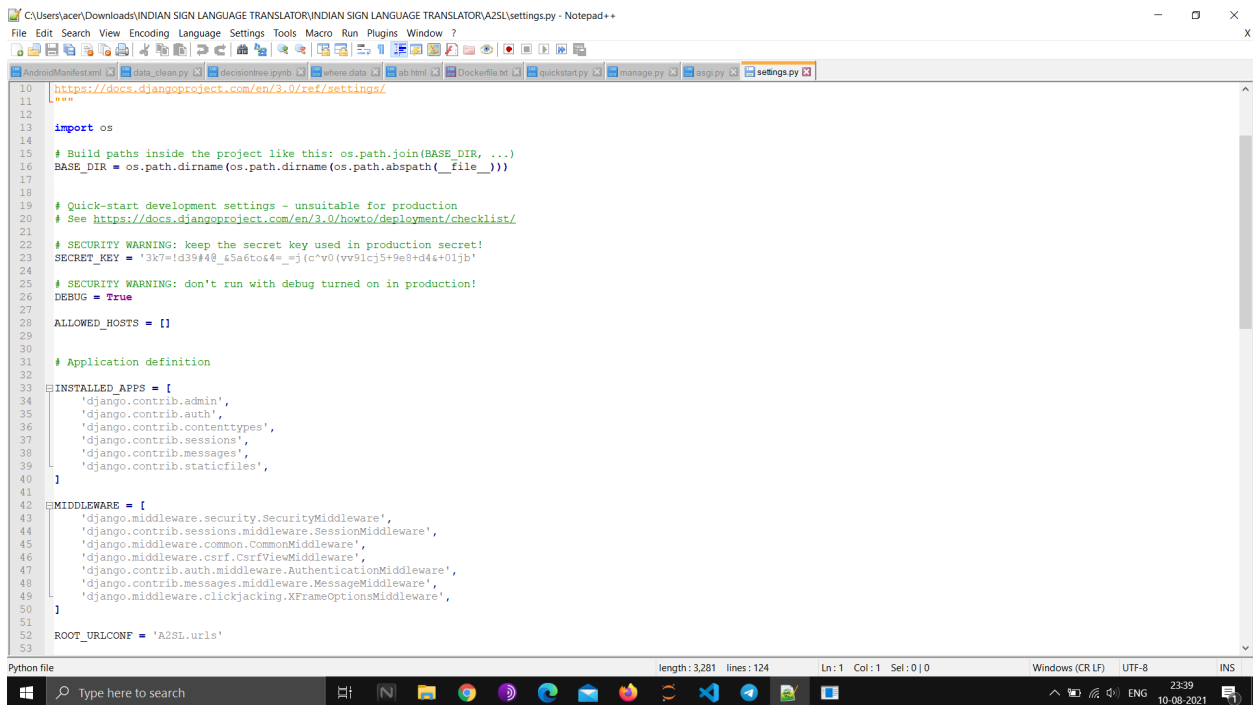
*Figure 6: Convertor*

## Code Snaps



```
1 # Django ASGI config for A2SL project.
2
3 # It exposes the ASGI callable as a module-level variable named ``application``.
4
5 # For more information on this file, see
6 # https://docs.djangoproject.com/en/3.0/howto/deployment/asgi/
7
8 """
9
10 import os
11
12 from django.core.asgi import get_asgi_application
13
14 os.environ.setdefault('DJANGO_SETTINGS_MODULE', 'A2SL.settings')
15
16 application = get_asgi_application()
17
18 """
```

Figure 7: Application initialization



```
10 https://docs.djangoproject.com/en/3.0/ref/settings/
11 """
12
13 import os
14
15 # Build paths inside the project like this: os.path.join(BASE_DIR, ...)
16 BASE_DIR = os.path.dirname(os.path.abspath(__file__))
17
18
19 # Quick-start development settings - unsuitable for production
20 # See https://docs.djangoproject.com/en/3.0/howto/deployment/checklist/
21
22 # SECURITY WARNING: keep the secret key used in production secret!
23 SECRET_KEY = '3k7e!d39#48_65a6to64=-_j(c^v0(vv91cj5+9e8+d44+01jb'
24
25 # SECURITY WARNING: don't run with debug turned on in production!
26 DEBUG = True
27
28 ALLOWED_HOSTS = []
29
30
31 # Application definition
32
33 INSTALLED_APPS = [
34     'django.contrib.admin',
35     'django.contrib.auth',
36     'django.contrib.contenttypes',
37     'django.contrib.sessions',
38     'django.contrib.messages',
39     'django.contrib.staticfiles',
40 ]
41
42 MIDDLEWARE = [
43     'django.middleware.security.SecurityMiddleware',
44     'django.contrib.sessions.middleware.SessionMiddleware',
45     'django.middleware.common.CommonMiddleware',
46     'django.middleware.csrf.CsrfViewMiddleware',
47     'django.contrib.auth.middleware.AuthenticationMiddleware',
48     'django.contrib.messages.middleware.MessageMiddleware',
49     'django.middleware.clickjacking.XFrameOptionsMiddleware',
50 ]
51
52 ROOT_URLCONF = 'A2SL.urls'
53
54 """
```

Figure 8: Allowed Hosts and Installed Apps and other middlewares

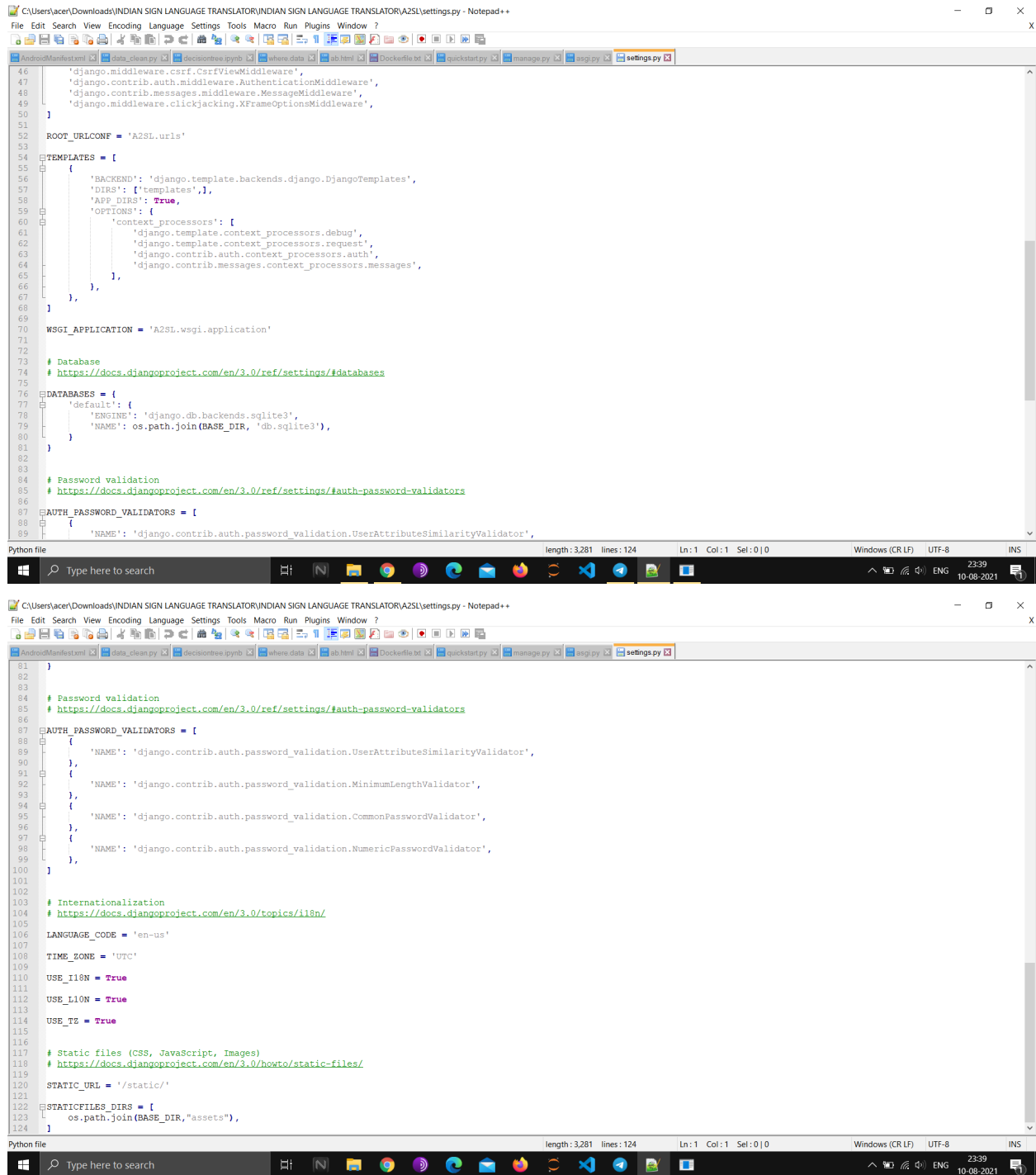


Figure 9: validations

```
C:\Users\acer\Downloads\INDIAN SIGN LANGUAGE TRANSLATOR\INDIAN SIGN LANGUAGE TRANSLATOR\A2SL\urls.py - Notepad++
File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?
AndroidManifest.xml data_clean.py decisiontree.pyib where data ab.html Dockerfile.be quickstart.py manage.py asgi.py settings.py urls.py
1 """A2SL URL Configuration
2
3 The 'urlpatterns' list routes URLs to views. For more information please see:
4     https://docs.djangoproject.com/en/3.0/topics/http/urls/
5 Examples:
6 Function views
7     1. Add an import: from my_app import views
8     2. Add a URL to urlpatterns: path('', views.home, name='home')
9 Class-based views
10    1. Add an import: from other_app.views import Home
11    2. Add a URL to urlpatterns: path('', Home.as_view(), name='home')
12 Including another URLconf
13    1. Import the include() function: from django.urls import include, path
14    2. Add a URL to urlpatterns: path('blog/', include('blog.urls'))
15 """
16 from django.contrib import admin
17 from django.urls import path
18 from .. import views
19
20 urlpatterns = [
21     path('admin/', admin.site.urls),
22     path('about/', views.about_view, name='about'),
23     path('contact/', views.contact_view, name='contact'),
24     path('login/', views.login_view, name='login'),
25     path('logout/', views.logout_view, name='logout'),
26     path('signup/', views.signup_view, name='signup'),
27     path('animation/', views.animation_view, name='animation'),
28     path('', views.home_view, name='home'),
29     path('animation/', views.animation_view, name='animation')
30 ]
31
Python file length: 1,223 lines: 31 Ln: 1 Col: 1 Sel: 0 | 0 Windows (CR LF) UTF-8 INS
Type here to search 23:39 10-08-2021
```

Figure 10: URLS

```
C:\Users\acer\Downloads\INDIAN SIGN LANGUAGE TRANSLATOR\INDIAN SIGN LANGUAGE TRANSLATOR\A2SL\views.py - Notepad++
File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?
AndroidManifest.xml data_clean.py decisiontree.pyib where data ab.html Dockerfile.be quickstart.py manage.py asgi.py settings.py urls.py views.py
1 from django.http import HttpResponse
2 from django.shortcuts import render, redirect
3 from django.contrib.auth.forms import UserCreationForm, AuthenticationForm
4 from django.contrib.auth import login, logout
5 from nltk.tokenize import word_tokenize
6 from nltk.corpus import stopwords
7 from nltk.stem import WordNetLemmatizer
8 import nltk
9 from django.contrib.staticfiles import finders
10 from django.contrib.auth.decorators import login_required
11
12 nltk.download('punkt')
13
14 def home_view(request):
15     return render(request, 'home.html')
16
17 def about_view(request):
18     return render(request, 'about.html')
19
20 def contact_view(request):
21     return render(request, 'contact.html')
22
23 @login_required(login_url='login')
24 def animation_view(request):
25     if request.method == 'POST':
26         text = request.POST.get('sen')
27         #tokenizing the sentence
28         text_lower = text.lower()
29         #tokenizing the sentence
30         words = word_tokenize(text)
31
32         tagged = nltk.pos_tag(words)
33         tense = {}
34         tense['future'] = len([word for word in tagged if word[1] == "MD"])
35         tense['present'] = len([word for word in tagged if word[1] in ["VBP", "VBZ", "VBG"]])
36         tense['past'] = len([word for word in tagged if word[1] in ["VBD", "VBN"]])
37         tense['present_continuous'] = len([word for word in tagged if word[1] in ["VBG"]])
38
39         #stopwords that will be removed
40
41
42
43
Python file length: 4,459 lines: 145 Ln: 1 Col: 1 Sel: 0 | 0 Windows (CR LF) UTF-8 INS
Type here to search 23:39 10-08-2021
```

Figure 11: Views

```

C:\Users\acer\Downloads\INDIAN SIGN LANGUAGE TRANSLATOR\INDIAN SIGN LANGUAGE TRANSLATOR\A2SL\views.py - Notepad++
File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?
AndroidManifest.xml data_clean.py decisiontree.py nb where data ab.html Dockerfile bt quickstart.py manage.py asgi.py settings.py urls.py views.py
28 text = request.POST.get('sen')
29 #tokenizing the sentence
30 text.lower()
31 #tokenizing the sentence
32 words = word_tokenize(text)
33
34 tagged = nltk.pos_tag(words)
35 tense = {}
36 tense["future"] = len([word for word in tagged if word[1] == "MD"])
37 tense["present"] = len([word for word in tagged if word[1] in ["VBP", "VBZ", "VBG"]])
38 tense["past"] = len([word for word in tagged if word[1] in ["VBD", "VBN"]])
39 tense["present_continuous"] = len([word for word in tagged if word[1] in ["VBG"]])
40
41
42
43 #stopwords that will be removed
44 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', 's
45
46
47
48 #removing stopwords and applying lemmatizing nlp process to words
49 lr = WordNetLemmatizer()
50 filtered_text = []
51 for w,p in zip(words,tagged):
52     if w not in stop_words:
53         if p[1]=='VBG' or p[1]=='VBD' or p[1]=='VBZ' or p[1]=='VBN' or p[1]=='NN':
54             filtered_text.append(lr.lemmatize(w,pos='v'))
55         elif p[1]=='JJ' or p[1]=='JJR' or p[1]=='JJS' or p[1]=='RBR' or p[1]=='RBS':
56             filtered_text.append(lr.lemmatize(w,pos='a'))
57         else:
58             filtered_text.append(lr.lemmatize(w))
59
60
61
62 #adding the specific word to specify tense
63 words = filtered_text
64 temp=[]
65 for w in words:
66     if w=="I":
67         temp.append('Me')
68     else:
69         temp.append(w)
70 words = temp

```

Figure 12 Adding Stopwords

```

C:\Users\acer\Downloads\INDIAN SIGN LANGUAGE TRANSLATOR\INDIAN SIGN LANGUAGE TRANSLATOR\A2SL\views.py - Notepad++
File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?
AndroidManifest.xml data_clean.py decisiontree.py nb where data ab.html Dockerfile bt quickstart.py manage.py asgi.py settings.py urls.py views.py
55 elif p[1]=='JJ' or p[1]=='JJR' or p[1]=='JJS' or p[1]=='RBR' or p[1]=='RBS':
56     filtered_text.append(lr.lemmatize(w,pos='a'))
57
58     else:
59         filtered_text.append(lr.lemmatize(w))
60
61
62 #adding the specific word to specify tense
63 words = filtered_text
64 temp=[]
65 for w in words:
66     if w=="I":
67         temp.append('Me')
68     else:
69         temp.append(w)
70 words = temp
71 probable_tense = max(tense,key=tense.get)
72
73 if probable_tense == "past" and tense["past"]>=1:
74     temp = ["before"]
75     temp = temp + words
76     words = temp
77 elif probable_tense == "future" and tense["future"]>=1:
78     if "will" not in words:
79         temp = ["will"]
80         temp = temp + words
81         words = temp
82     else:
83         pass
84 elif probable_tense == "present":
85     if tense["present_continuous"]>=1:
86         temp = ["Now"]
87         temp = temp + words
88         words = temp
89
90
91 filtered_text = []
92 for w in words:
93     path = w + ".mp4"
94     f = finders.find(path)
95     #splitting the word if its animation is not present in database
96     if not f:
97         for c in w:

```

Figure 13: Functions

```
C:\Users\acer\Downloads\INDIAN SIGN LANGUAGE TRANSLATOR\INDIAN SIGN LANGUAGE TRANSLATOR\A2S1\views.py - Notepad++
File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?
Python file length: 4,459 lines: 145 Ln: 1 Col: 1 Sel: 0 | 0 Windows (CR LF) UTF-8 INS
104         return render(request, 'animation.html', {'words': words, 'text': text})
105     else:
106         return render(request, 'animation.html')
107
108
109
110
111
112 def signup_view(request):
113     if request.method == 'POST':
114         form = UserCreationForm(request.POST)
115         if form.is_valid():
116             user = form.save()
117             login(request, user)
118             # log the user in
119             return redirect('animation')
120     else:
121         form = UserCreationForm()
122         return render(request, 'signup.html', {'form': form})
123
124
125
126 def login_view(request):
127     if request.method == 'POST':
128         form = AuthenticationForm(data=request.POST)
129         if form.is_valid():
130             # log in user
131             user = form.get_user()
132             login(request, user)
133             if 'next' in request.POST:
134                 return redirect(request.POST.get('next'))
135             else:
136                 return redirect('animation')
137     else:
138         form = AuthenticationForm()
139         return render(request, 'login.html', {'form': form})
140
141
142 def logout_view(request):
143     logout(request)
144     return redirect("home")
145
```

## *Sign up and Login Functions*

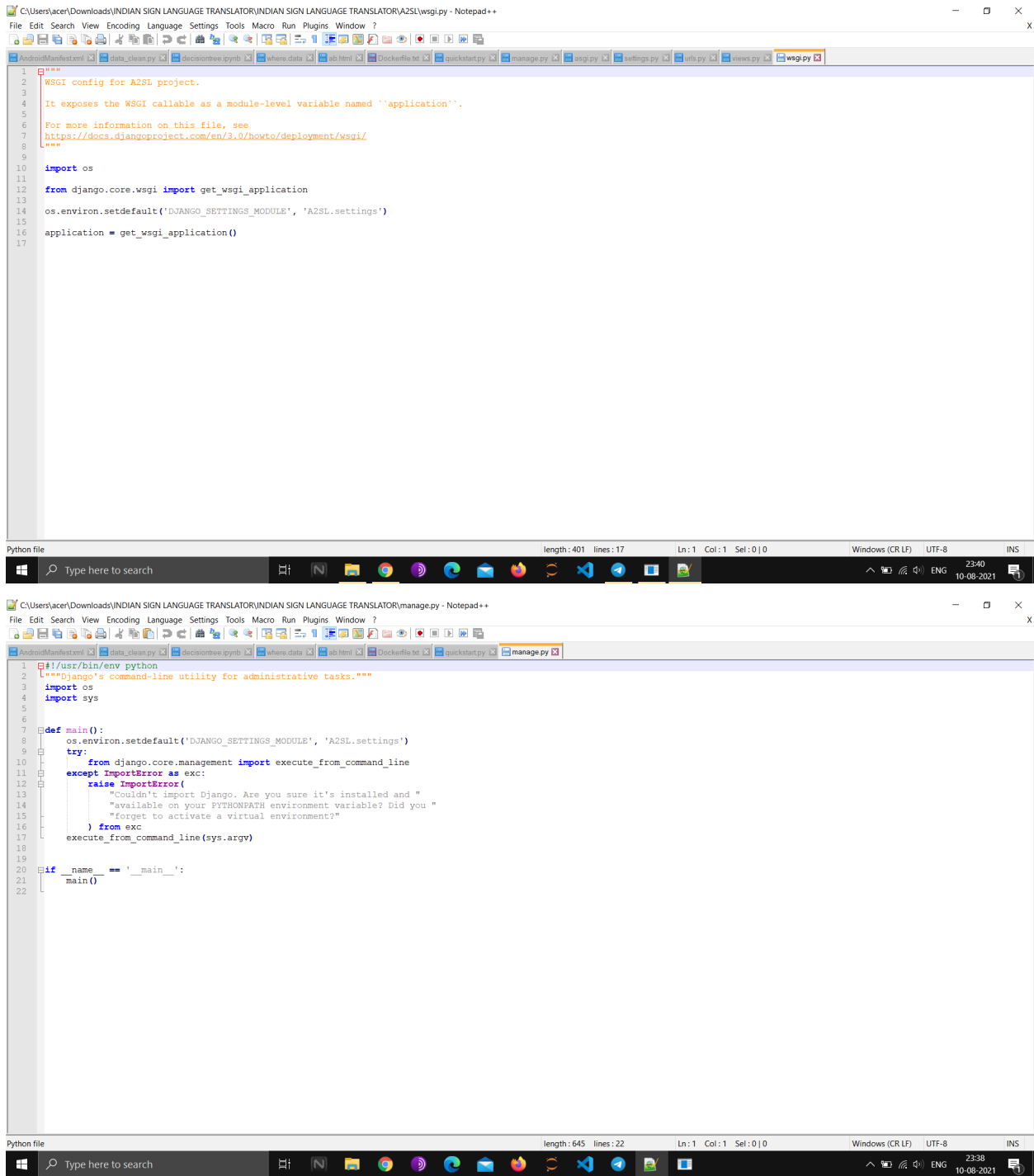


Figure 14: Error Pages



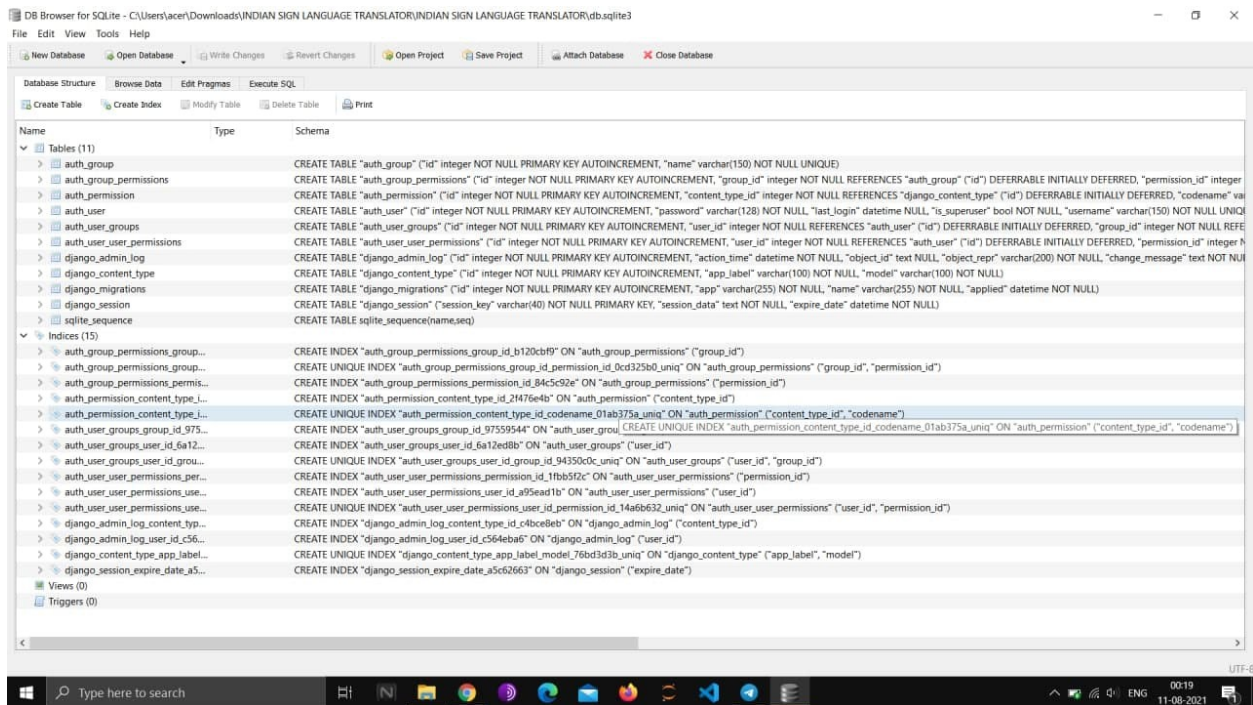


Figure 15: DB Tables and Indices

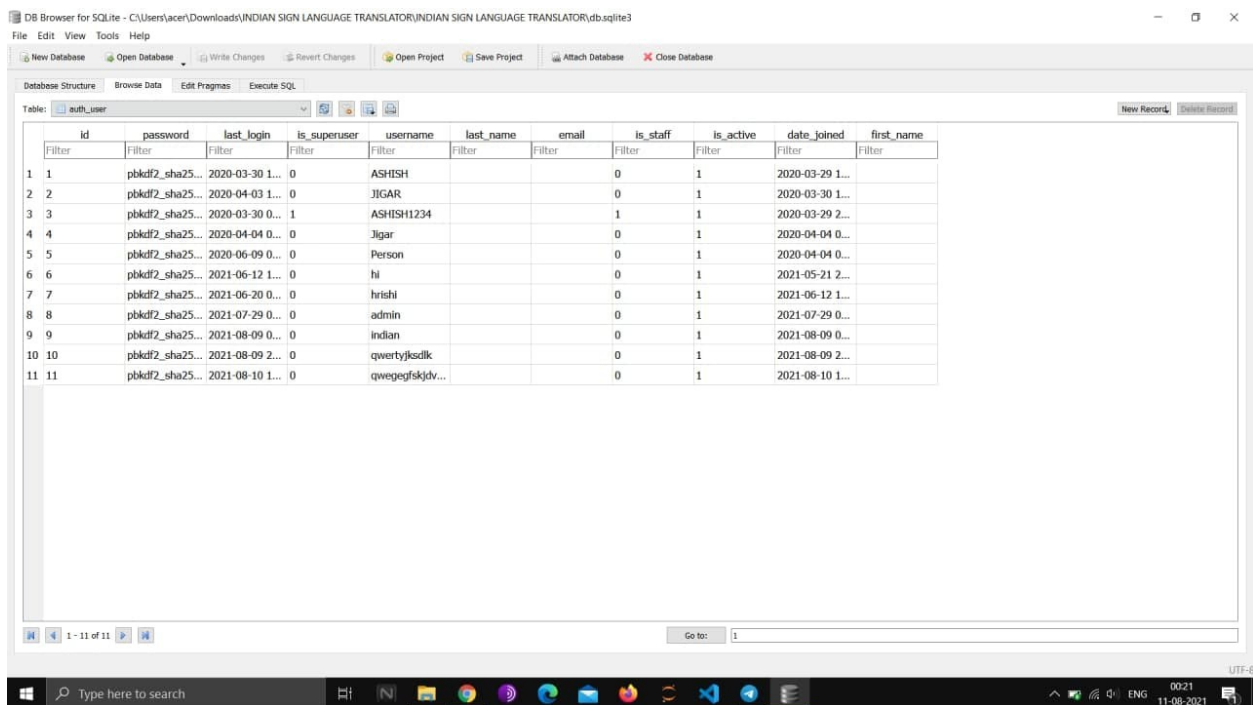


Figure 16: Existing Users DB User Table

## TESTING

**Test Plan:** A test plan can be defined as a document describing the scope, approach, resources, and schedule of intended testing activities. It identifies test items, the features to be tested, the testing tasks, who will do each task, and any risks requiring contingency planning. In software testing, a test plan gives detailed testing information regarding an upcoming testing effort, including

- \_ Scope of testing
- \_ Schedule
- \_ Test Deliverables
- \_ Release Criteria
- \_ Risks and Contingencies

It is also be described as a detail of how the testing will proceed, who will do the testing, what will be tested, in how much time the test will take place, and to what quality level the test will be performed.

The process of defining a test project so that it can be properly measured and controlled. The test planning process generates a high level test plan document that identifies the software items to be tested, the degree of tester independence, the test environment, the test case design and test measurement techniques to be used, and the rationale for their choice.

A testing plan is a methodological and systematic approach to testing a system such as a machine or software. It can be effective in finding errors and flaws in a system. In order to find relevant results, the plan typically contains experiments with a range of operations and values, including an understanding of what the eventual workflow will be.

Test plan is a document which includes, introduction, assumptions, list of test cases, list of features to be tested, approach, deliverables, resources, risks and scheduling. A test plan is a systematic approach to testing a system such as a machine or software. The plan typically contains a detailed understanding of what the eventual workflow will be. A record of the test planning process detailing the degree of tester independence, the test environment, the test case design techniques and test measurement techniques to be used, and the rationale for their choice.

## TESTING ACTIVITIES

Various testing activities are performed:

- Black box testing - Internal system design is not considered in this type of testing. Tests are based on requirements and functionality.
- White box testing - This testing is based on knowledge of the internal logic of an application's code. Also known as Glass box Testing. Internal software and code working should be known for this type of testing. Tests are based on coverage of code statements, branches, paths, conditions.
- Unit testing - Testing of individual software components or modules. Typically done by the programmer and not by testers, as it requires detailed knowledge of the internal program design and code. May require developing test driver modules or test harnesses.
- Incremental integration testing Bottom up approach for testing i.e. continuous testing of an application as new functionality is added; Application functionality and modules should be independent enough to test separately. Done by programmers or by testers.
- Integration testing - Testing of integrated modules to verify combined functionality after integration. Modules are typically code modules, individual applications, client and server applications on a network, etc. This type of testing is especially relevant to client/server and distributed systems.
- Functional testing - This type of testing ignores the internal parts and focuses on the output as per requirement or not. Black-box type testing geared to functional requirements of an application.
- System testing Entire system is tested as per the requirements. Black-box type testing that is based on overall requirements specifications, covers all combined parts of a system.
- End-to-end testing Similar to system testing, involves testing of a complete application environment in a situation that mimics real-world use, such as interacting with a database, using network communications, or interacting with other hardware, applications, or systems if appropriate.

- Acceptance testing - Normally this type of testing is done to verify if system meets the customer specified requirements. User or customer do this testing to determine whether to accept application.
  - Usability testing User-friendliness check. Application flow is tested, Can new user understand the application easily, Proper help documented whenever user stuck at any point. Basically system navigation is checked in this testing.

# **USER /OPERATIONAL MANUAL**

## **Procedure**

### **1. Audio to Sign Language Conversion:**

- Audio input is taken using python PyAudio module.
- Conversion of audio to text using microphone
- Dependency parser is used for analyzing grammar of the sentence and obtaining relationship between words.
- Text is converted to Sign Language

### **2. Text to Sign Language:**

- i. Speech recognition using Google Speech API.
- ii. Text Preprocessing using NLP.
- iii. Dictionary based Machine Translation.
- iv. Conversion of Text into Sign Language

## **CONCLUSION**

Sign language translator is very useful in various areas. In schools, colleges, hospitals, universities, airports, courts anywhere anyone can use this system for understanding the sign language to communicate. It makes communication between a normal hearing person and a hard to hearing person easier. The future work is to develop an application where in the news channels can use it while giving news, in one corner of the screen it will be displayed in sign language for deaf people. Write now only DD news is using this kind of presentation but they are using a human being showing signs according to the speech of the person giving news live. So this will be better idea which we can give to news channels. We look forward to expand the project by also including facial expressions into the system.

## **FUTURE SCOPE & ENHANCEMENT**

- Support for More Languages
- Better Translation of words to ISL
- Increasing the dictionary
- Adding more GIF's in the dictionary to get more accurate translation
- Enhancing the way project can be used
- Support for more platforms
- Improving its GUI for more better and friendly look
- Adding the education panel so people can learn the Sign languages and common words

## REFERENCES

- [1] Amit Kumar Shinde and Ramesh Khagalkar “sign language to text and vice versa recognition using computer vision in Marathi” International journal of computer Application (0975-8887) National conference on advanced on computing (NCAC 2015).
- [2] Neha Poddar, Shrushti Rao, Shruti Sawant, Vrushali Somavanshi, Prof.Sumita Chandak "Study of Sign Language Translation using Gesture Recognition" International Journal of Advanced Research in Computer and Communication Engineering Vol. 4, Issue 2, February 2015.
- [3] <https://www.slideshare.net/mobile/madhuriyellapu/signlanguage-translator-ieee-power-point>