CSN-254
Software Engineering

SignBot

Design Document

Group 17:
Aniket Umesh Kathare (20114010)
Arnav Vyas (20114015)
Deepesh Garg (20114032)
Shaurya Rana (20114086)
Shaurya Semwal (20119048)

Summary

This design document has been developed to elucidate the design and workflow implementation of crucial features of SignBot. This design document's primary goals are to increase operational efficiency and provide a sustainable base and framework around which the coding phase can be structured. Both the logical and physical design components have been mentioned throughout the document. The low-level and high-level aspects of the design have been detailed clearly.

This document identifies the points of contact, lists all the implementation requirements, and shows all the deliverables and an overview of the implementation process for SignBot. The document covers all the features we have included in our project, and its application to the user is specified later in the document. For a better understanding of the client, several UML diagrams are appended to the document.

A Use-case diagram is used to elucidate the workflow of the use cases and deliverables. All the use-cases have been accurately represented in a concise form which gives the developers an initial idea about the overall working of the system.

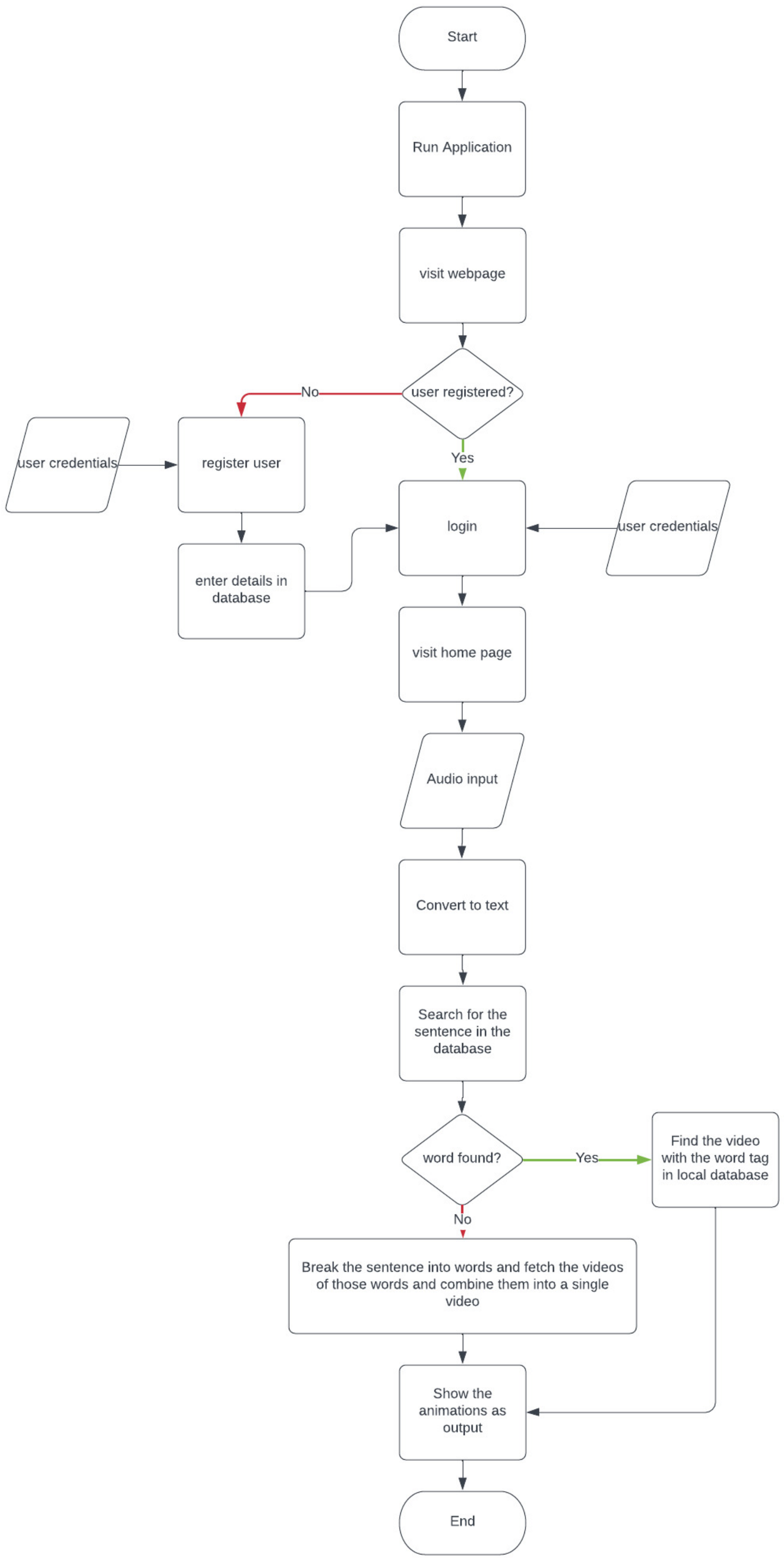
Class-diagram to show the relation between the various components/ objects within the system and what these objects do along with their services. It is highly beneficial for the developers and, if referenced and understood correctly, simplifies the coding part decently.

Sequence diagrams to show the method and order in which the processes are executed. It also shows how objects in the current environment interact and are used to document the behavior of an improved future system.

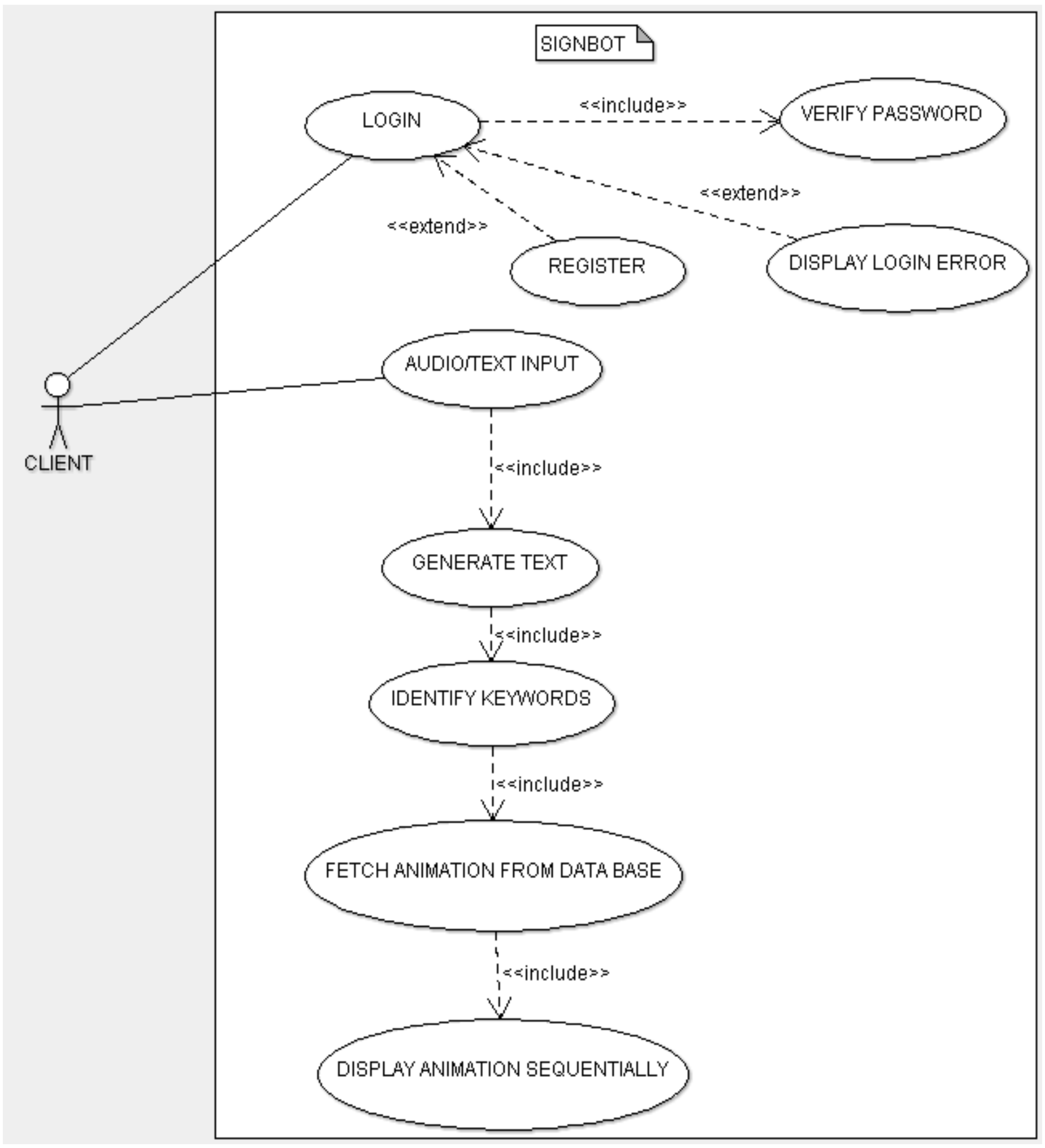
Furthermore, a flowchart diagram has been subsumed in the document to describe the user workflow or the project plan along with the interaction patterns of the website with the frontend and the databases.

High Level Design

1. Flowchart


Start
Run Application
visit webpage
user registered?
No
Yes
user credentials
register user
enter details in database
login
user credentials
visit home page
Audio input
Convert to text
Search for the sentence in the database
word found?
Yes
Find the video with the word tag in local database
No
Break the sentence into words and fetch the videos of those words and combine them into a single video
Show the animations as output
End

2. **Use Case Diagram**

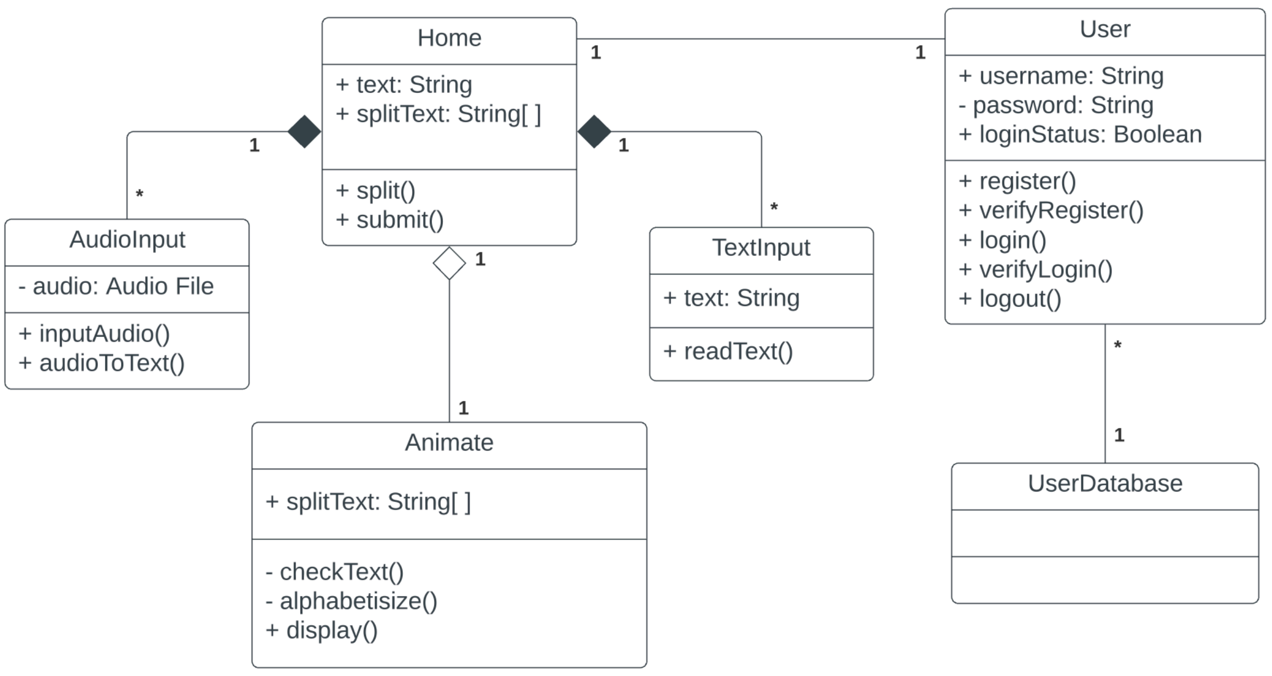


a. Register: a user must register on our site, filling in information like name, email address, phone number and password.

b. Login: once registration is done, a user will have to login using email address and password.

c. View Animation: on giving the audio Input, the software displays the corresponding Animation on the screen.

3. **Class Diagram**



a. User

i. Attributes:

1. **username(*public*):** stores the username entered by the user.
2. **password(*private*):** stores the password set by the user.
3. **loginStatus(*public*):** stores the the status (logged in or logged out) of the current user.

ii. Methods:

1. **register(*public*):** method to handle Sign Up request after user enters the details
2. **verifyRegister(*public*):** method for verifying successful Sign Up of user into the application.
3. **login(*public*):** method for handling the Login request after the user enters the Username and Password.
4. **verifyLogin(*public*):** method for verifying successful Login of the user.
5. **logout(*public*):** method for handling the Logout request.

b. Home

i. Attributes:

1. **text(*public*):** stores the input string entered by the user either through audio or typed text.

2. **splitText(*public*):** stores the words present in **text** in the form of an array.

ii. Methods:

1. **split(*public*):** method to split the input **text** into words resulting in **splitText**.

2. **submit(*public*):** method for submitting the user input for converting to sign language.

c. AudioInput

i. Attributes:

1. **audio(*private*):** stores audio recorded by the user.

ii. Methods:

1. **inputAudio(*public*):** method for recording user audio using the microphone present in user's system.

2. **audioToText(*public*):** method for converting audio to text.

d. TextInput

i. Attributes:

1. **text(*public*):** stores text entered by the user.

ii. Methods:

1. **readText(*public*):** A function which is used to read the text entered by the user.

e. Animate

i. Attributes:

1. **splitText(*public*):** stores the words present in user entered text in the form of an array.

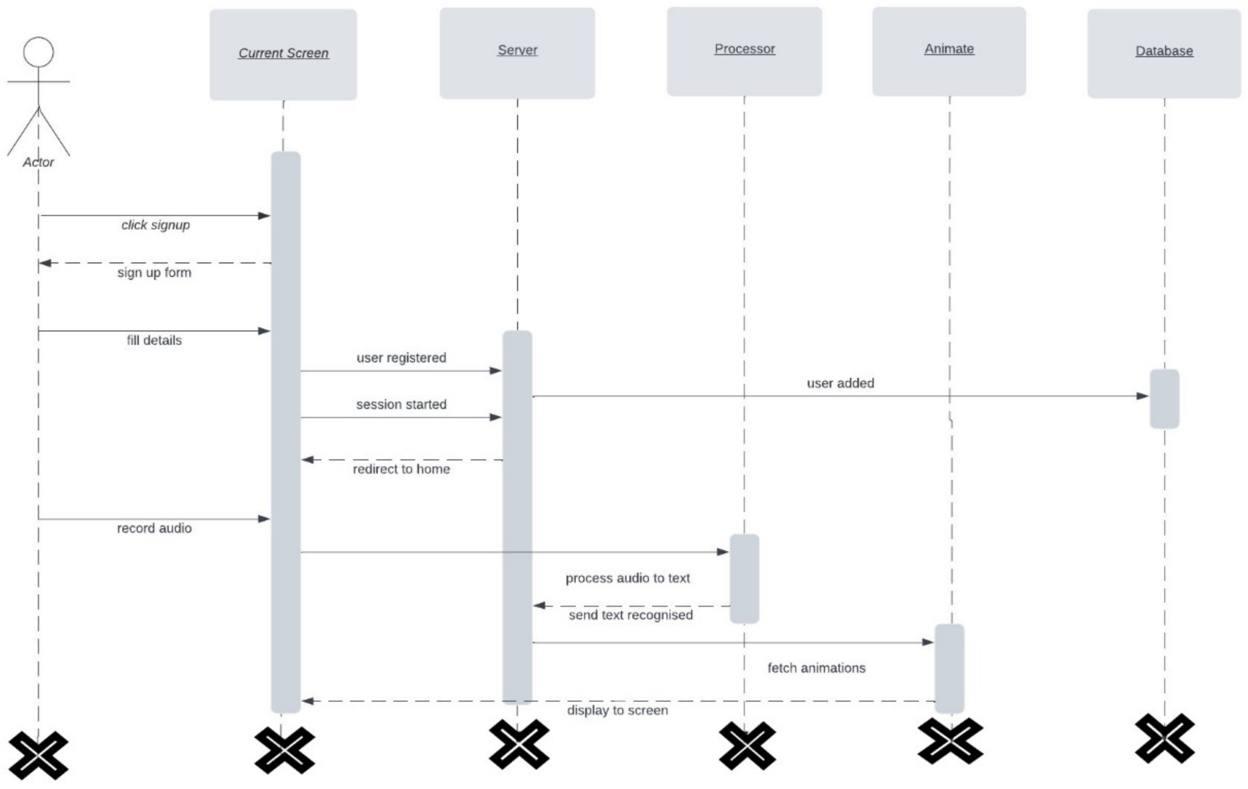
ii. Methods:

1. **checkText(*private*):** method to check if direct animation exists for the word.

2. **alphabetisize(*private*):** A function to break the word into its alphabets in case direct animation doesn't exist.

3. **display(*public*):** method to display the final result of the conversion of audio/text to sign language.

4. Sequence Diagram



Low Level Design

Workflow of the implemented system:

1. The front-end requests the user for login/register credentials. The information is sent to the back end which verifies the username and password if the user already exists. Otherwise, a new ID is created.
2. Then it requests the user for audio/text input. In case of audio input, JavaScript Web speech API is used to convert it to text.
3. Then the keywords are identified from the text using Natural Language Toolkit (NLTK).
4. The identified keywords are then checked in the database. The corresponding animations are displayed if they exist. Otherwise, the keywords are split into alphabets and the corresponding animations of the alphabets are displayed, respectively.
5. After displaying each animation, the operation is verified.

Following are the code-snippets and algorithms-

a. Login

```
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})
```

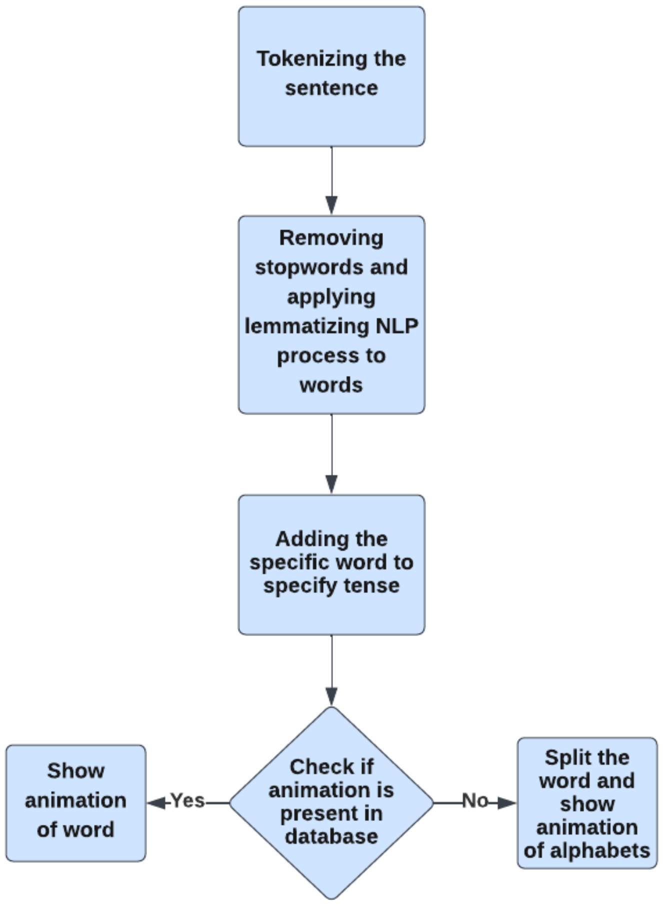
b. Register

```
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})
```

c. View Animation

- *Tokenize the sentence;*
- *Categorize the words into categories depending upon the form of tense i.e. (future, past, present, present continuous);*
- *Identify the stop words (such as 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') ;*
- *Remove the Identified stop words and apply lemmatizing nlp process to words;*
- *Add the specific word to specify tense;*
- *Display the animation of identified words from animation database. If not found, then split the word into alphabets and display the corresponding animation sequentially;*

Figure flow chart:



Tokenizing the sentence
Removing stopwords and applying lemmatizing NLP process to words
Adding the specific word to specify tense
Check if animation is present in database
Yes
Show animation of word
No
Split the word and show animation of alphabets

Comments

Through our web application we are trying to reduce the communication gap between disabled and non-disabled persons. Easy to understand signs and instant conversion from audio to signs are some of the features of our application. It is easy to access and user friendly. The design document is very helpful in understanding the details required before beginning the coding phase. Using this document even the customer can understand the functionalities of the application