CHAPTER 1

INTRODUCTION

In today's digital era, the demand for intuitive and versatile applications that cater to diverse user needs is ever-growing. One area of increasing interest is the integration of speech-related functionalities into web applications. Speech-to-text (STT) and text-to-speech (TTS) capabilities offer users convenient ways to interact with applications, especially in scenarios where typing may not be feasible or efficient.

This project introduces a web-based application developed using the Angular framework, aimed at providing seamless speech-to-text and text-to-speech functionalities. The application comprises two main components: the Translator Component and the Process Speech Component.

The Translator Component serves as a language translation tool, allowing users to input text, select source and target languages, and translate text between languages. Leveraging the MyMemory Translation API, users can obtain accurate translations of their text inputs. Additionally, the Translator Component integrates speech synthesis capabilities, enabling users to hear the translated text audibly.

Complementing the Translator Component is the Process Speech Component, which facilitates speech recognition within the application. Utilizing the webkitSpeechRecognition API, users can input text through speech, which is then processed to trigger specific actions based on predefined commands. This component enhances user interaction by allowing voice commands to execute actions such as opening URLs or providing responses.

By harnessing the capabilities of browser APIs for both speech-to-text and text-to-speech functionalities, this project showcases the seamless integration of speech-related features into a web application. The following sections delve deeper into the implementation details of each component, highlighting the features, functionalities, and potential areas for further enhancement.

1.1 Objectives

1. Enable Speech Interaction: Provide users with the ability to interact with the application using speech input, allowing for hands-free operation and accessibility for users with disabilities.

2. Facilitate Language Translation: Implement a language translation feature that allows users to input text in one language and obtain translations in another language, enhancing communication across linguistic barriers.

3. Enhance User Experience: Offer a seamless and intuitive user experience by integrating speech recognition and synthesis capabilities into the application's interface, ensuring ease of use and efficiency.

4. Ensure Cross-Platform Compatibility: Utilize browser APIs for speech recognition and synthesis to ensure compatibility across different platforms and browsers, enabling users to access the application from various devices.

5. Provide Customization Options: Allow users to customize their language preferences, input text, and adjust settings to suit their individual needs and preferences.

6. Promote Modularity and Scalability: Design the application architecture in a modular and scalable manner, facilitating the addition of new features and enhancements in the future.

1.2 Scope of the project

1. Speech Recognition:
   - Implement speech recognition functionality using the `webkitSpeechRecognition` API to enable users to input text through speech.
   - Process recognized speech to extract meaningful commands or input from users.

2. Language Translation:
   - Develop a language translation feature allowing users to input text in one language and obtain translations in another language.
   - Utilize the MyMemory Translation API or similar services to provide accurate translations.

3. Text-to-Speech Synthesis:
   - Incorporate text-to-speech synthesis capabilities to audibly output translated text or responses to user queries.
   - Utilize browser APIs such as `SpeechSynthesisUtterance` for text-to-speech functionality.

4. User Interface:
   - Design an intuitive and user-friendly interface for inputting text, selecting languages, and interacting with speech-related features.
   - Include options for customization and settings adjustment to accommodate user preferences.

5. Cross-Platform Compatibility:
   - Ensure compatibility across various devices and browsers by leveraging browser APIs for speech recognition and synthesis.
   - Optimize the application for responsiveness and performance on different screen sizes and resolutions.

6. Error Handling and Validation:

- Implement robust error handling mechanisms to gracefully handle exceptions and unexpected scenarios.
- Validate user inputs and provide feedback to ensure data integrity and prevent errors.

7. Documentation and Testing:

- Provide comprehensive documentation detailing the application's features, functionalities, and usage instructions.
- Conduct thorough testing to validate the correctness, reliability, and performance of the implemented features.

8. Scalability and Future Enhancements:

- Design the application architecture to be modular and extensible, facilitating the addition of new features and enhancements in the future.
- Consider potential enhancements such as multi-language support, voice commands for navigation, and integration with external services.

CHAPTER 2

SYSTEM REQUIREMENTS

The system requirements for this project involve both development environment prerequisites and end-user requirements:

2.1 Development Environment Requirements:

1. Operating System: Windows, macOS, or Linux.

2. Integrated Development Environment (IDE): Any IDE or text editor of choice, such as Visual Studio Code, WebStorm, or Atom.

3. Node.js and npm: Latest stable versions of Node.js and npm installed on the development machine.

4. Angular CLI: Latest version of Angular CLI installed globally for creating and managing Angular projects.

5. Browser: Latest versions of Google Chrome, Mozilla Firefox, Safari, or Microsoft Edge for testing and debugging.

2.2 End-User Requirements:

1. Web Browser: Users should have access to a modern web browser with support for HTML5, CSS3, and JavaScript.

2. Internet Connection: A stable internet connection is required for accessing external APIs used for language translation and speech recognition.

3. Microphone (Optional): For users intending to utilize speech input functionalities, a microphone connected to the device may be necessary.

4. Speakers or Headphones (Optional): Users may require speakers or headphones to listen to the text-to-speech synthesized output.

2.3 Additional Considerations:

1. Accessibility: Ensure the application is accessible to users with disabilities, including support for screen readers and keyboard navigation.

2. Localization: Consider providing language options for the application interface to cater to users from diverse linguistic backgrounds.

3. Data Privacy: Implement appropriate measures to safeguard user data and ensure compliance with data protection regulations.

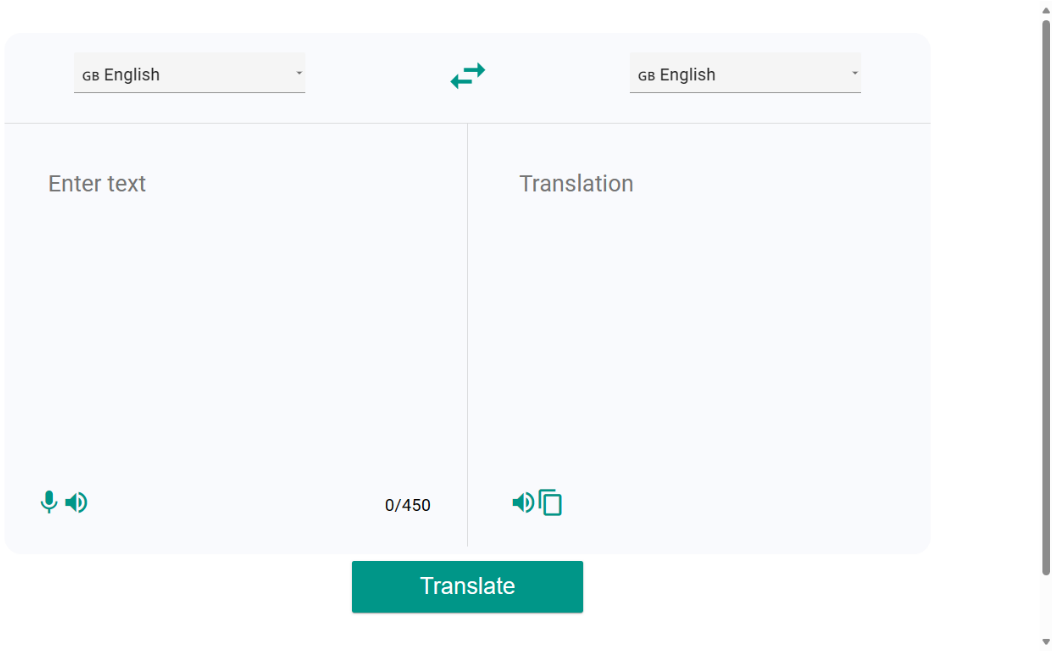
4. Scalability: Design the application to handle potential increases in user traffic and data volume as usage grows over time.

5. Documentation: Provide clear and comprehensive documentation for both developers and end-users, detailing installation instructions, usage guidelines, and troubleshooting steps.

CHAPTER 3

DESIGN

Designing the architecture for this project involves structuring the components, services, and modules in a way that promotes modularity, scalability, and maintainability. Below is a high-level overview of the design for this project:



3.1 Components:

1. Translator Component:

- Handles language selection, text input, translation functionality, and speech synthesis.
- Consists of UI elements such as dropdowns for language selection, text areas for input and translation, and buttons for actions.
- Interacts with translation and speech synthesis services to perform translations and generate synthesized speech.

2. Process Speech Component:

- Manages speech recognition functionality and processes recognized speech to trigger actions.
- Contains a button or interface element to initiate speech recognition.
- Utilizes the `webkitSpeechRecognition` API for speech recognition and processes recognized speech using predefined commands.

3.2 Services:

1. Translation Service:

- Handles communication with external translation APIs (e.g., MyMemory Translation API).
- Provides methods for translating text between languages based on source and target language selections.

2. Speech Synthesis Service:

- Integrates with browser APIs (e.g., `SpeechSynthesisUtterance`) to perform text-to-speech synthesis.
- Offers methods for generating synthesized speech from translated text and controlling speech output.

3.3 Modules:

1. Translator Module:

- Contains the Translator Component and related services.
- Provides a cohesive module for language translation functionality within the application.

2. Process Speech Module:

- Includes the Process Speech Component and any supporting services.
- Offers a module dedicated to speech recognition and processing functionalities.

3.3 Design Considerations:

1. Modularity: Divide the application into reusable and independent modules to facilitate code organization and maintenance.

2. Separation of Concerns: Maintain a clear separation between presentation logic (components) and business logic (services) to improve code readability and maintainability.

3. Service Dependency Injection: Utilize Angular's dependency injection mechanism to inject services into components, enabling loose coupling and facilitating unit testing.

4. Error Handling: Implement robust error handling mechanisms at both component and service levels to gracefully handle errors and provide meaningful feedback to users.

5. Optimized UI/UX: Design intuitive and user-friendly interfaces that guide users through the translation and speech interaction processes seamlessly.

6. Testing: Conduct comprehensive testing, including unit tests for individual components and services, as well as integration tests to ensure the proper functioning of the application as a whole.

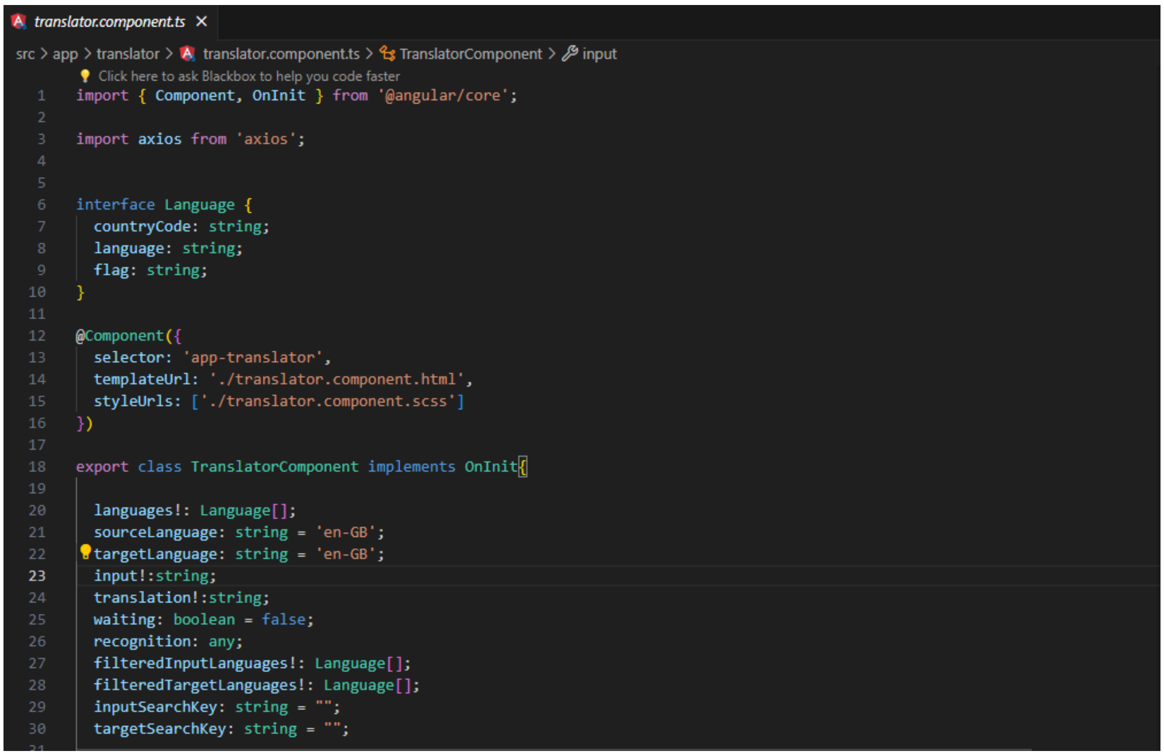
CHAPTER 4

IMPLEMENTATION

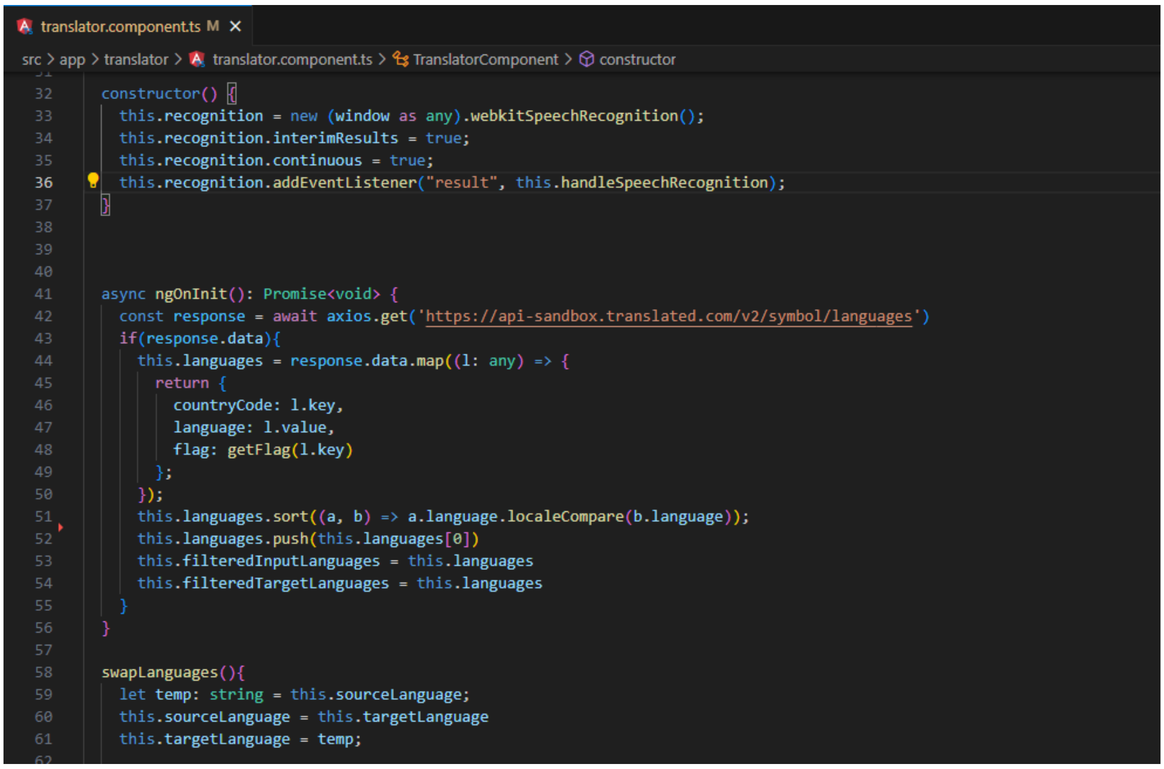
Translator Module:

TranslatorComponent, facilitates language translation and speech interaction within a web application. It retrieves language data from an external API, enables users to input text for translation, supports speech input via the webkitSpeechRecognition API, and offers features such as language filtering, translation swapping, and text-to-speech synthesis. Additionally, it handles translation requests via the MyMemory Translation API and provides functionality for copying translated text to the clipboard.

Translator.component.ts:


translator.component.ts
src > app > translator > translator.component.ts > TranslatorComponent > input
Click here to ask Blackbox to help you code faster

```
import { Component, OnInit } from '@angular/core';

import axios from 'axios';


interface Language {
  countryCode: string;
  language: string;
  flag: string;
}

@Component({
  selector: 'app-translator',
  templateUrl: './translator.component.html',
  styleUrls: ['./translator.component.scss']
})

export class TranslatorComponent implements OnInit{

  languages!: Language[];
  sourceLanguage: string = 'en-GB';
  targetLanguage: string = 'en-GB';
  input!:string;
  translation!:string;
  waiting: boolean = false;
  recognition: any;
  filteredInputLanguages!: Language[];
  filteredTargetLanguages!: Language[];
  inputSearchKey: string = "";
  targetSearchKey: string = "";
```

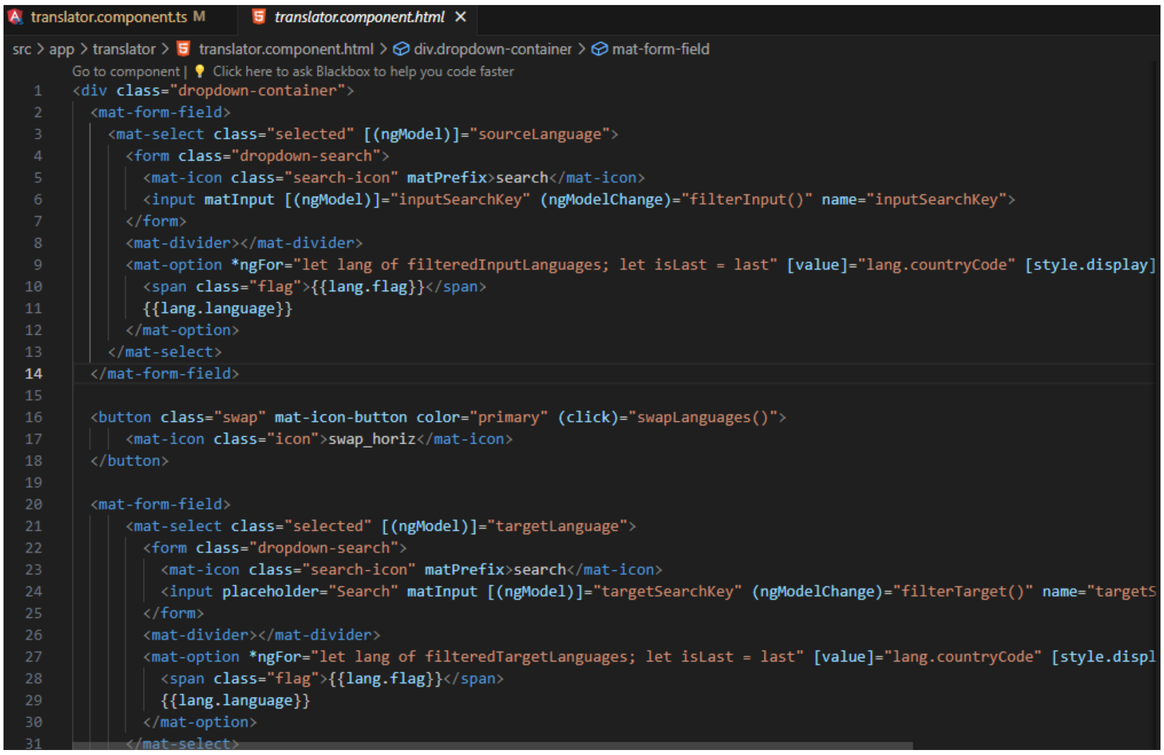


translator.component.ts M ×

src > app > translator > translator.component.ts > TranslatorComponent > constructor

```
  constructor() {
    this.recognition = new (window as any).webkitSpeechRecognition();
    this.recognition.interimResults = true;
    this.recognition.continuous = true;
    this.recognition.addEventListener("result", this.handleSpeechRecognition);
  }


  async ngOnInit(): Promise<void> {
    const response = await axios.get('https://api-sandbox.translated.com/v2/symbol/languages')
    if(response.data){
      this.languages = response.data.map((l: any) => {
        return {
          countryCode: l.key,
          language: l.value,
          flag: getFlag(l.key)
        };
      });
      this.languages.sort((a, b) => a.language.localeCompare(b.language));
      this.languages.push(this.languages[0])
      this.filteredInputLanguages = this.languages
      this.filteredTargetLanguages = this.languages
    }
  }

  swapLanguages(){
    let temp: string = this.sourceLanguage;
    this.sourceLanguage = this.targetLanguage
    this.targetLanguage = temp;
```

Translator.component.html:



translator.component.ts M | translator.component.html ×

src > app > translator > translator.component.html > div.dropdown-container > mat-form-field

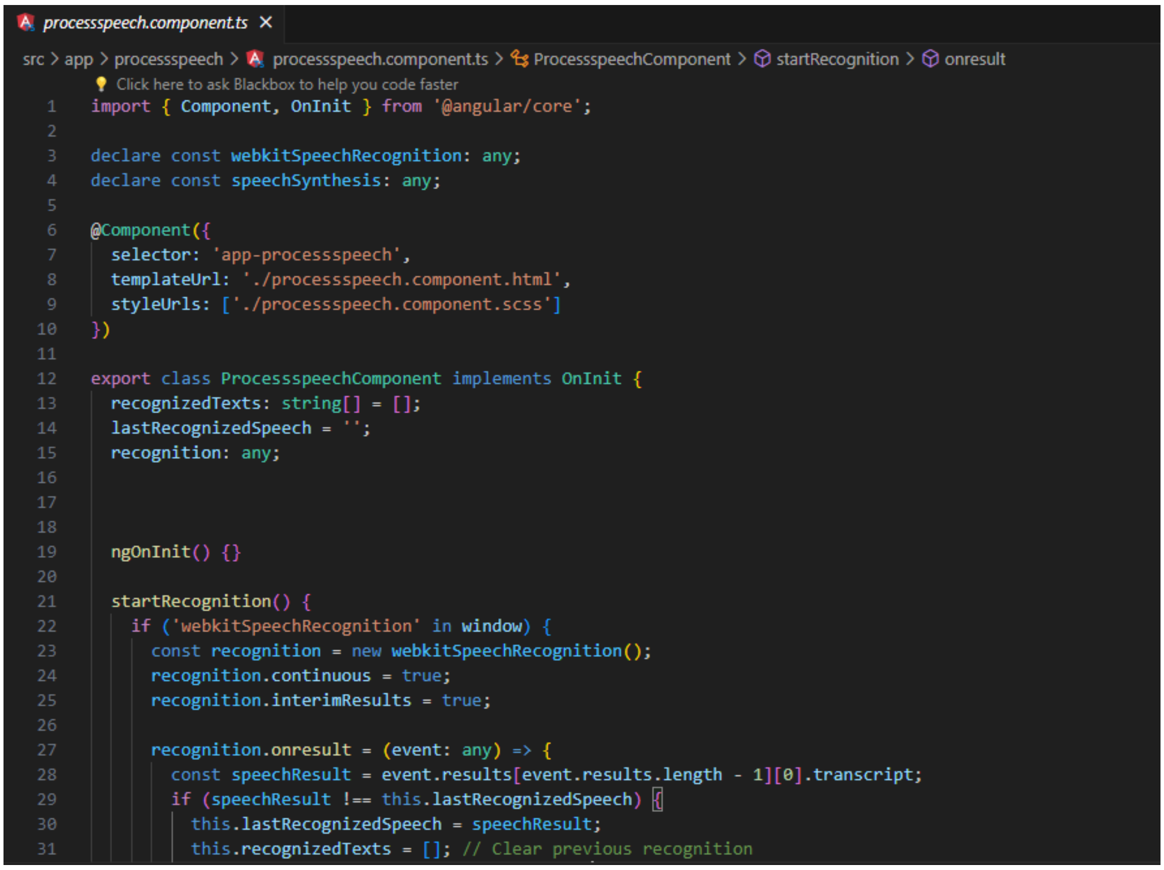
Go to component | Click here to ask Blackbox to help you code faster

```
<div class="dropdown-container">
  <mat-form-field>
    <mat-select class="selected" [(ngModel)]="sourceLanguage">
      <form class="dropdown-search">
        <mat-icon class="search-icon" matPrefix>search</mat-icon>
        <input matInput [(ngModel)]="inputSearchKey" (ngModelChange)="filterInput()" name="inputSearchKey">
      </form>
      <mat-divider></mat-divider>
      <mat-option *ngFor="let lang of filteredInputLanguages; let isLast = last" [value]="lang.countryCode" [style.display]
        <span class="flag">{{lang.flag}}</span>
        {{lang.language}}
      </mat-option>
    </mat-select>
  </mat-form-field>

  <button class="swap" mat-icon-button color="primary" (click)="swapLanguages()">
    <mat-icon class="icon">swap_horiz</mat-icon>
  </button>

  <mat-form-field>
      <mat-select class="selected" [(ngModel)]="targetLanguage">
        <form class="dropdown-search">
          <mat-icon class="search-icon" matPrefix>search</mat-icon>
          <input placeholder="Search" matInput [(ngModel)]="targetSearchKey" (ngModelChange)="filterTarget()" name="targetS
        </form>
        <mat-divider></mat-divider>
        <mat-option *ngFor="let lang of filteredTargetLanguages; let isLast = last" [value]="lang.countryCode" [style.displ
          <span class="flag">{{lang.flag}}</span>
          {{lang.language}}
        </mat-option>
      </mat-select>
```

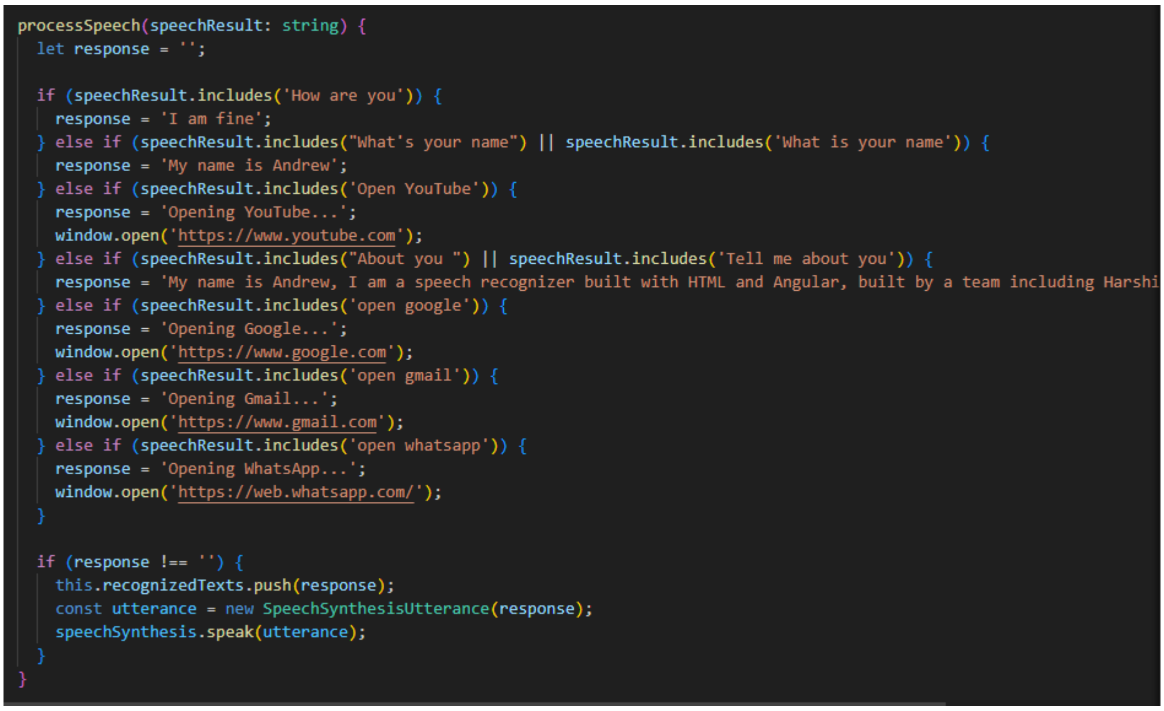
Processspeech module:

`ProcessspeechComponent`, handles speech recognition and processing within a web application. It utilizes the `webkitSpeechRecognition` API to recognize speech input, processes recognized speech to trigger predefined actions based on predefined commands, and utilizes `speechSynthesis` to generate synthesized speech responses. Additionally, it includes logic to respond to specific speech commands such as opening websites or providing information about the application.

Processspeech.component.ts:


processspeech.component.ts
src > app > processspeech > processspeech.component.ts > ProcessspeechComponent > startRecognition > onresult
Click here to ask Blackbox to help you code faster

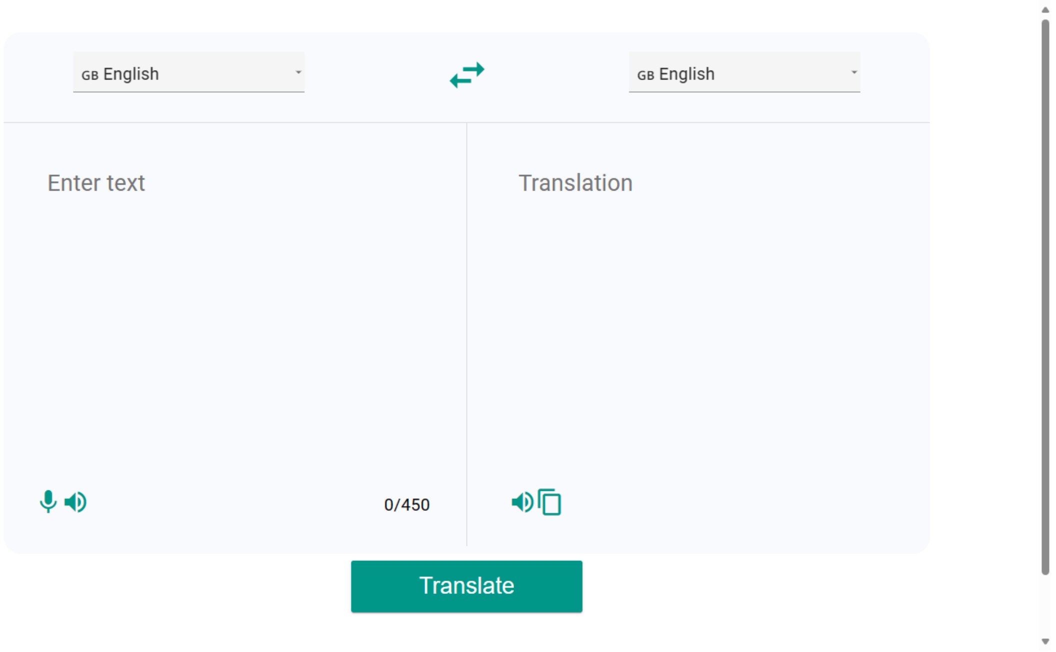
```
import { Component, OnInit } from '@angular/core';

declare const webkitSpeechRecognition: any;
declare const speechSynthesis: any;

@Component({
  selector: 'app-processspeech',
  templateUrl: './processspeech.component.html',
  styleUrls: ['./processspeech.component.scss']
})

export class ProcessspeechComponent implements OnInit {
  recognizedTexts: string[] = [];
  lastRecognizedSpeech = '';
  recognition: any;



  ngOnInit() {}

  startRecognition() {
    if ('webkitSpeechRecognition' in window) {
      const recognition = new webkitSpeechRecognition();
      recognition.continuous = true;
      recognition.interimResults = true;

      recognition.onresult = (event: any) => {
        const speechResult = event.results[event.results.length - 1][0].transcript;
        if (speechResult !== this.lastRecognizedSpeech) {
          this.lastRecognizedSpeech = speechResult;
          this.recognizedTexts = []; // Clear previous recognition
```



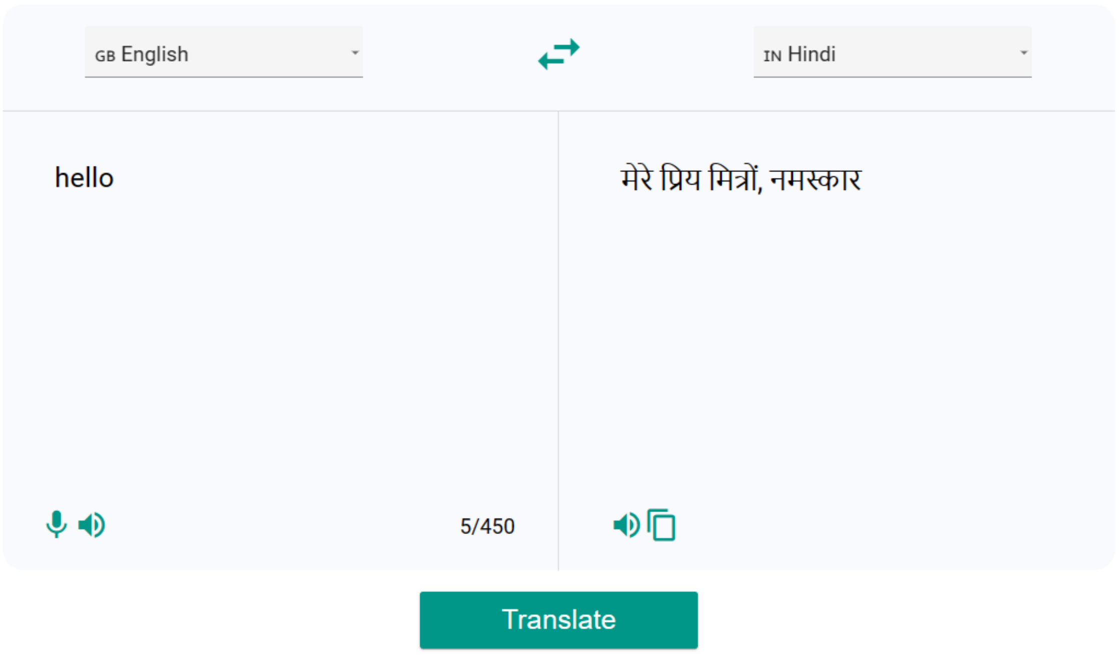
```
processSpeech(speechResult: string) {
  let response = '';

  if (speechResult.includes('How are you')) {
    response = 'I am fine';
  } else if (speechResult.includes("What's your name") || speechResult.includes('What is your name')) {
    response = 'My name is Andrew';
  } else if (speechResult.includes('Open YouTube')) {
    response = 'Opening YouTube...';
    window.open('https://www.youtube.com');
  } else if (speechResult.includes("About you ") || speechResult.includes('Tell me about you')) {
    response = 'My name is Andrew, I am a speech recognizer built with HTML and Angular, built by a team including Harshi
  } else if (speechResult.includes('open google')) {
    response = 'Opening Google...';
    window.open('https://www.google.com');
  } else if (speechResult.includes('open gmail')) {
    response = 'Opening Gmail...';
    window.open('https://www.gmail.com');
  } else if (speechResult.includes('open whatsapp')) {
    response = 'Opening WhatsApp...';
    window.open('https://web.whatsapp.com/');
  }

  if (response !== '') {
    this.recognizedTexts.push(response);
    const utterance = new SpeechSynthesisUtterance(response);
    speechSynthesis.speak(utterance);
  }
}
```

CHAPTER 5

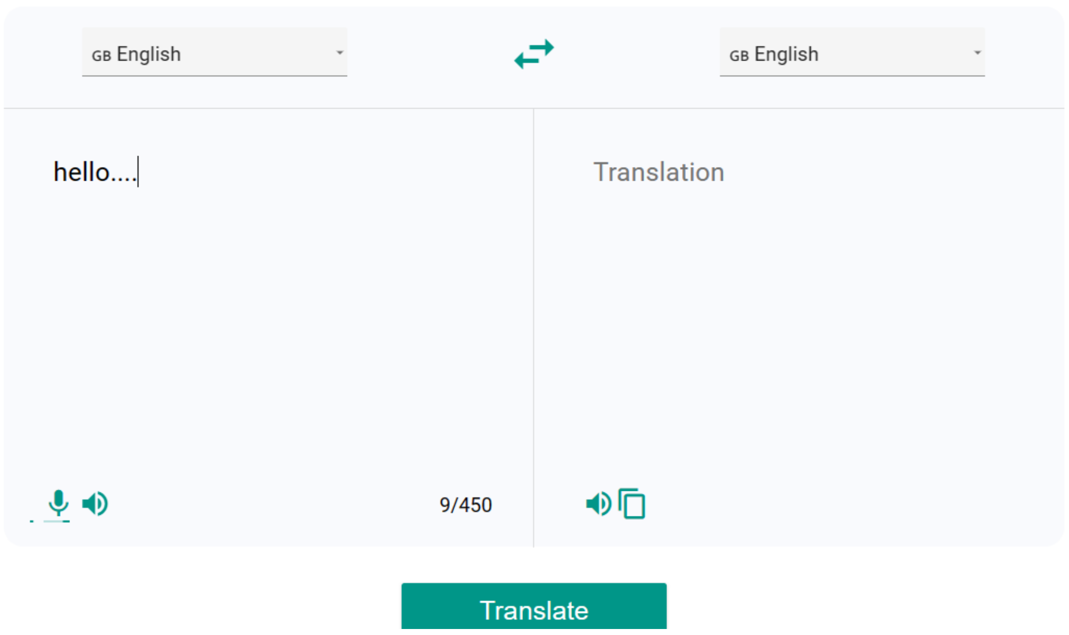
RESULT



HOME PAGE OF THE PROJECT



CONVERTING TEXT TO OTHER LANGUAGE



SPEECH TO TEXT CONVERTER

CHAPTER 6

CONCLUSION AND FUTURE SCOPE

6.1 Conclusion:

In conclusion, the implementation of the web-based application with speech-to-text and text-to-speech functionalities using Angular has provided users with a versatile and accessible platform for language translation and speech interaction. The project successfully achieved its objectives of enabling speech input, facilitating language translation, and enhancing user experience through speech synthesis. By leveraging Angular and browser APIs, the application offers a seamless and intuitive user interface while ensuring compatibility across different platforms and devices.

6.2 Future Scope:

Despite the successful implementation, there are several avenues for future enhancement and expansion of the project:

1. Enhanced Language Support: Expand language translation capabilities by integrating with additional translation APIs and supporting a broader range of languages.

2. Improved Speech Recognition: Enhance speech recognition accuracy and performance by implementing advanced algorithms or machine learning techniques.

3. Customization Options: Provide users with more customization options, such as adjusting speech recognition sensitivity or choosing different speech synthesis voices.

4. Accessibility Features: Further improve accessibility by incorporating features for users with disabilities, such as voice commands for navigation and compatibility with screen readers.

5. Real-time Collaboration: Implement real-time collaboration features, allowing multiple users to collaborate on translations or participate in voice conversations simultaneously.

6. Mobile Application: Develop a mobile application version of the project to extend its reach and accessibility to users on mobile devices.

7. Integration with Smart Assistants:Integrate the application with smart assistants like Google Assistant or Amazon Alexa to enable voice-controlled interactions.

REFERENCES

- AngularJS Tutorial (w3schools.com)
- Angular 2 Tutorial - Javatpoint
- Speech to text angular(SpeechRecognition) - Vangarsushil - Medium