

# Milestone 2 - Report

CMPT 276 @ Simon Fraser University

Fall 2023

[GitHub-Repo-Link](#)

<b>Group 21</b>
<b>Zheyuan Ou - 301464403</b>
<b>Damir Zharikessov - 301541028</b>
<b>Victor Nguyen - 301458739</b>
<b>Tegvaran Sooch - 301418178</b>

# Table of Contents

<b>Overview.....</b>	<b>1</b>
<b>Choice of SDLC Model:.....</b>	<b>1</b>
What worked:.....	2
What did not work:.....	2
<b>Features.....</b>	<b>2</b>
Language Translation:.....	2
Text Summarization:.....	4
Language Detection:.....	5
Multilingual Text-To-Speech.....	6
Multilingual Speech-To-Text.....	7
Speech Voice Customization.....	8
<b>Tests.....</b>	<b>9</b>
Unit Tests.....	9
Integration Tests.....	10
<b>CI/CD infrastructure.....</b>	<b>12</b>
<b>Data Flow Diagram.....</b>	<b>13</b>
<b>Reflections.....</b>	<b>13</b>
How work was divided up.....	14



## Overview

Our project's primary goal is to develop a web-based application that offers valuable support to students who are non-native English speakers and those with visual impairments. The core concept revolves around harnessing the capabilities of the OpenAI API for language translation and the SpeechSynthesis API for text-to-speech conversion to enhance the accessibility and comprehensibility of academic assignments.

For non-native English speakers, we envision a tool that enables them to copy text from their assignments and gain a clear understanding of the content in their native language. This facilitates improved comprehension of their academic tasks. Additionally, our application will extend its utility to translating lecture notes into the user's preferred language, further enhancing learning experiences.

Furthermore, we are introducing a speech-to-speech translation feature that allows students to engage in seamless conversations with their peers. This feature ensures effective communication and mutual understanding between individuals working on collaborative projects, bridging language barriers and promoting teamwork.

Now that we have presented an overview of our project and its core objectives, it is essential to delve into the methodology that will guide our development process. In the following section, we will explain our choice of the Software Development Life Cycle (SDLC) model.

## Choice of SDLC Model:

We have opted for the Extreme Programming (XP) methodology for our project due to our team's composition. Specifically, we have two team members experienced in JavaScript, while the other two are unfamiliar with it.

To address this, we are leveraging XP's pair programming practice, where each experienced JavaScript developer is paired with a less experienced counterpart. This ensures that the learning curve is smoother for those new to JavaScript, as they have immediate guidance and problem-solving support. Pair programming not only accelerates learning but also enhances code quality and issue resolution. It aligns with our goal of enabling every team member to contribute to the project actively.

In addition to pair programming, we are incorporating other XP practices like test-driven development, continuous integration, and small, iterative releases. Our choice of XP emphasizes collaboration, flexibility, and efficient knowledge sharing to deliver a high-quality application.

### **What worked:**

Pair programming worked really well. We were able to ask each other questions and seek help easily because we had set the expectation that mutual assistance would be crucial. Additionally, getting on calls with each other and screen sharing expedited our coding process. Without these collaborative sessions, we would have encountered longer delays in problem-solving.

### **What did not work:**

We encountered several unexpected merge conflicts, highlighting the importance of improved communication needed. A more proactive approach to communication about individual goals and progress could have alleviated these issues. For instance, implementing weekly meetings at the beginning and end of each week to discuss upcoming tasks, accomplishments, and potential roadblocks could have enhanced our overall collaboration and project workflow.

## **Features**

Here are the features we implemented in our application:

### **OpenAI API:**

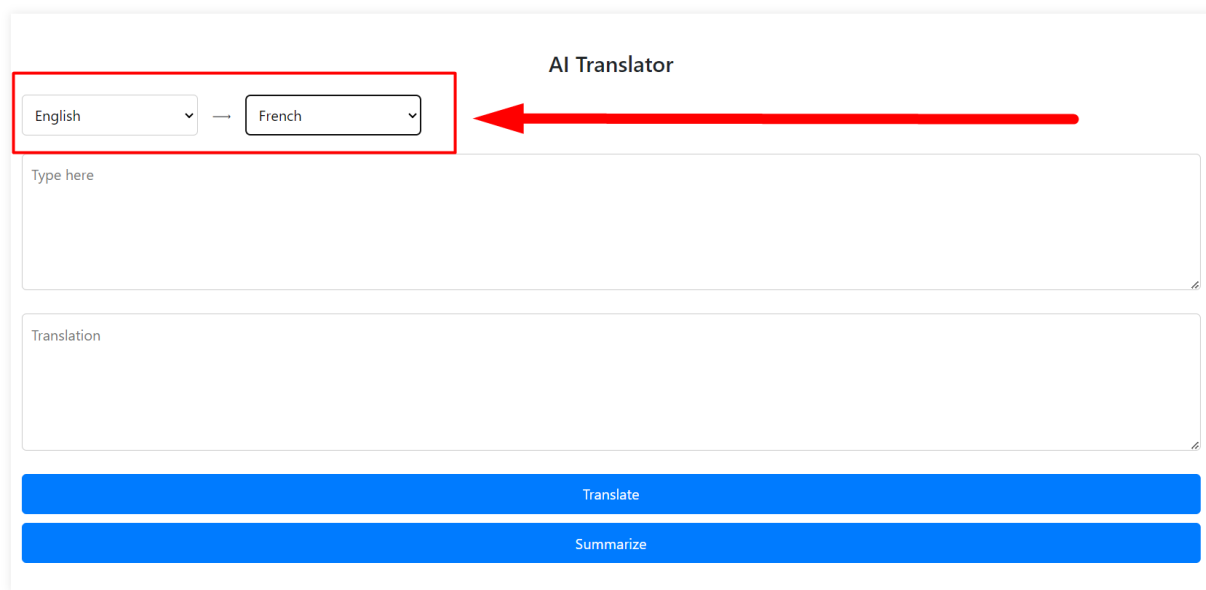
- 1) Language Translation
- 2) Text Summarization
- 3) Language Detection

### **SpeechSynthesis API:**

- 1) Multilingual Text-to-Speech
- 2) Multilingual Speech-to-text
- 3) Customization of Speech Voice

### **Language Translation:**

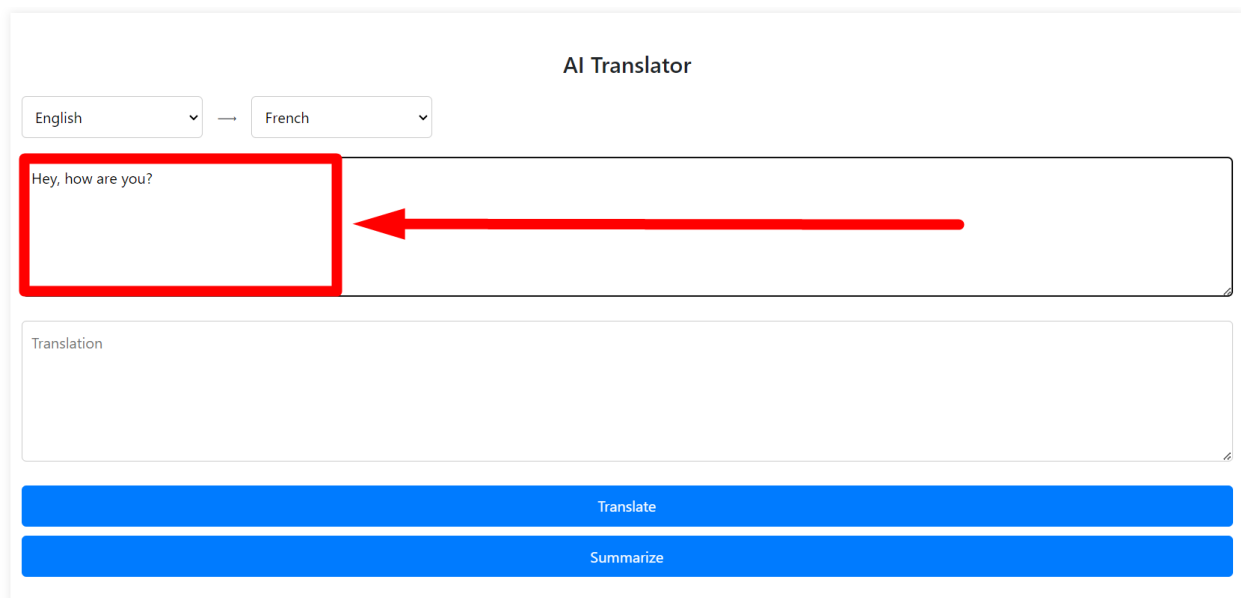
This feature allows the user to translate text from one language to another. Users begin by choosing the source language and the target language through the language dropdown menu (refer to Figure 1).



The image shows a web interface for an AI Translator. At the top, the title "AI Translator" is centered. Below it, there are two dropdown menus for language selection. The first dropdown is set to "English" and the second is set to "French", with a double-headed arrow between them. A red rectangular box highlights these two dropdowns, and a red arrow points from the right towards this box. Below the language selection, there is a large text input area labeled "Type here" with a placeholder text "Type here". Below the input area is a text output area labeled "Translation" with a placeholder text "Translation". At the bottom, there are two blue buttons: "Translate" and "Summarize".

*Figure 1 - Language Selection*

Upon selecting languages, users input the text they wish to translate in the "Type here" input box (refer to Figure 2). For example, entering "Hey, how are you?"



The image shows the same AI Translator interface as Figure 1, but with the text "Hey, how are you?" entered into the "Type here" input box. A red rectangular box highlights the input box, and a red arrow points from the right towards this box. The "Translation" output box is still empty. The "Translate" and "Summarize" buttons remain at the bottom.

*Figure 2 - Type here input box*

After inputting the text, users initiate the translation process by clicking the "Translate" button located at the bottom of the screen (refer to Figure 3.1). The translated text appears in the "Translation" output box, situated below the input box (refer to Figure 3.2).

The screenshot shows the 'AI Translator' interface. At the top, there are two dropdown menus for language selection, currently set to 'English' and 'French'. Below these is a text input field containing the English text 'Hey, how are you?'. Underneath the input field is a red rectangular box labeled '3.2' containing the French translation 'Salut, comment ça va ?'. To the right of this box is a red label '3.1'. Below the translation box are two blue buttons: 'Translate' (highlighted with a red box) and 'Summarize'.

*Figure 3 - Translate button and Translation output*

After a few seconds, the translation of the inputted text will appear in the “Translation” output box. For this example, the translation for “hey, how are you?” comes out to be “Salut, comment ça va?”.

Users can choose from a diverse range of language options, with a selection pool encompassing 30 different languages for both source and target translations.

### **Text Summarization:**

This feature allows users to summarize translated text. If the user wishes to translate longer text, they can click on the 'Summarize' button (Figure 4.1) to obtain a summary of the translated text. The summarized text is displayed below the initial translation in the 'Translation' output (refer to Figure 4.2).

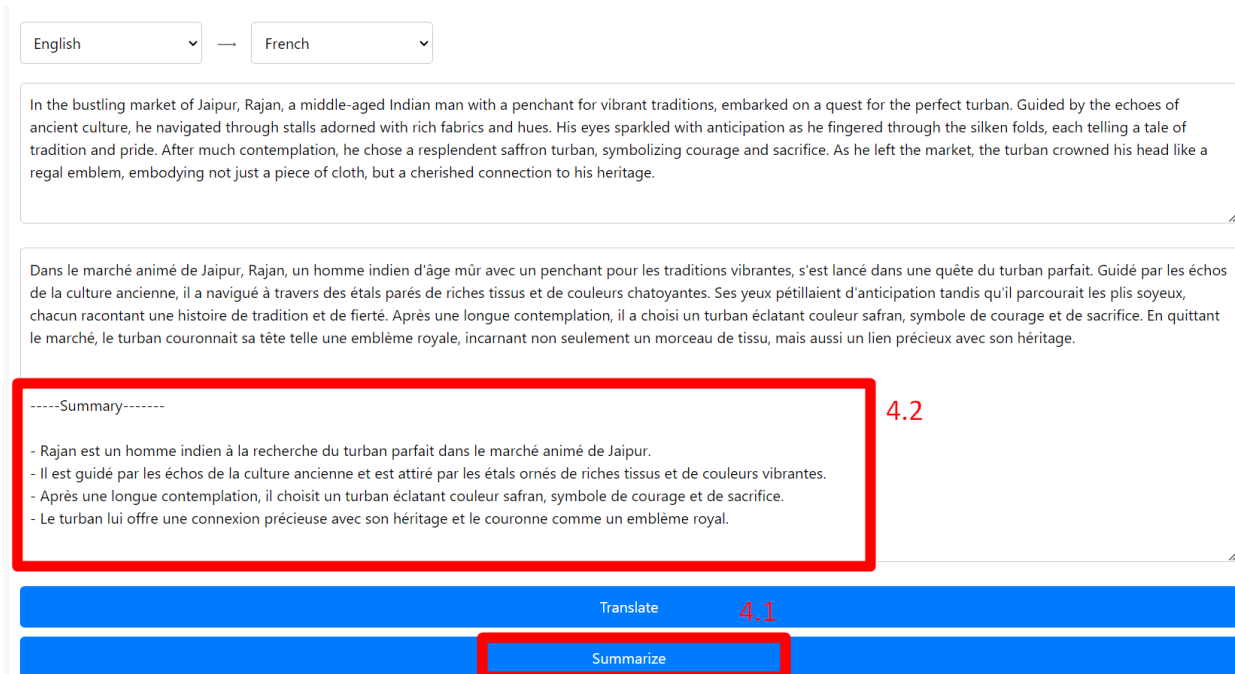


Figure 4 - Summarize button and Summarization

The summary takes a bit of time to load if the text is long.

### Language Detection:

This feature automatically detects the language the user is trying to translate from. The user can select the "Detect Language" option (Figure 5.1) from the "From Language" dropdown. The user can then paste the text they want to translate into the "Type here" input box (Figure 5.2).

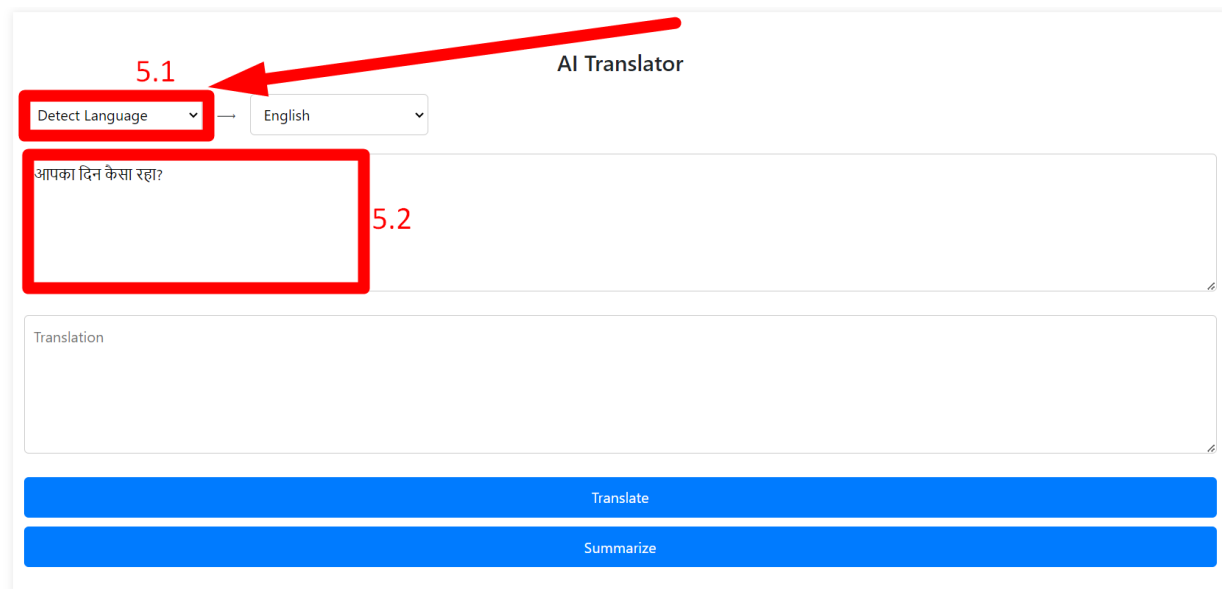
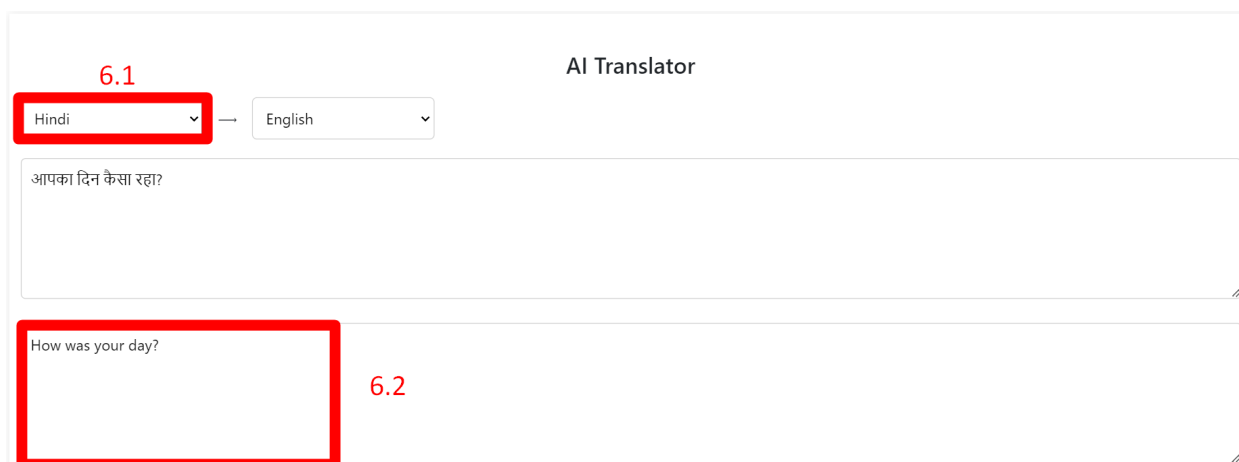


Figure 5 - Detect Language option and Type here input box



When the user hits the translate button, the application automatically detects the language (Figure 6.1) of the text and translates it into the desired language (Figure 6.2).



*Figure 6 - Detected Language and Translated Output*

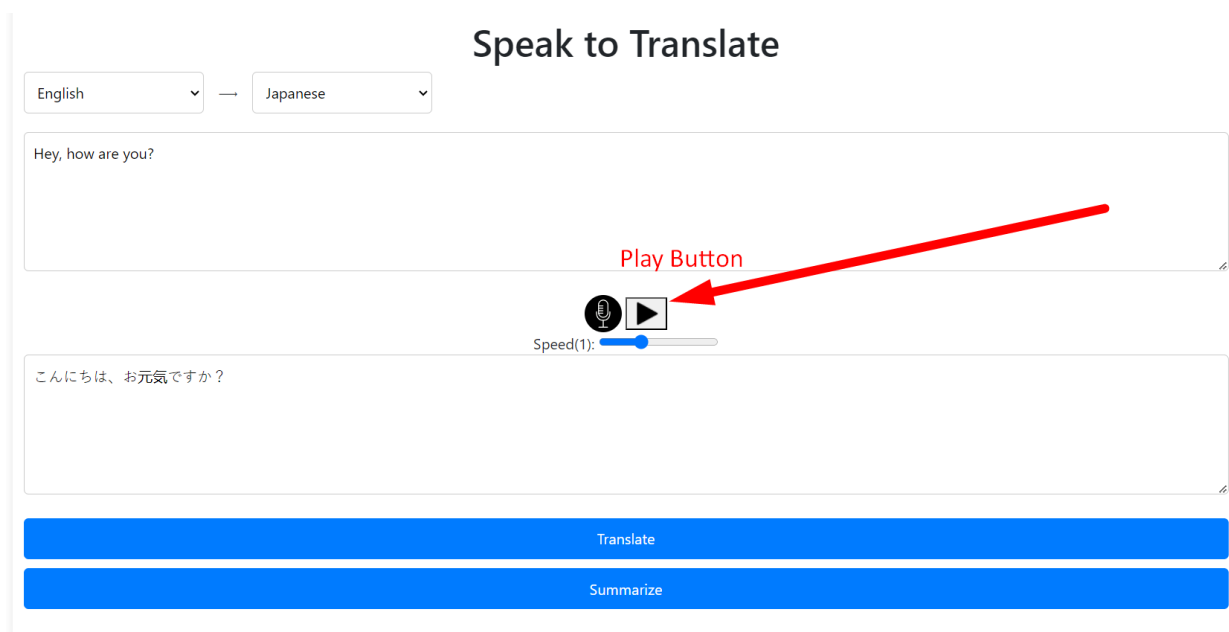
This feature is only available in the "Type to Translate" page and not the "Speak to Translate" page.

#### **Bug:**

- 1) One of the bugs is that a language is detected successfully and is translated properly but a blank is displayed on the from language.
- 2) Sometimes it does not work on the first click, if this happens, select a different "to" language and press "translate" button again. Should work now.

#### **Multilingual Text-To-Speech**

This feature, present in the "Speak to Translate" page of the application, allows the user to listen to the translated text in the language it was translated to. After the user clicks the "Translate" button, they can click the "Play" button (refer to Figure 7) to listen to the translated text in the target language with the correct pronunciations.



*Figure 7 - Play button*

For example, if the user translates “Hey, how are you?” into Japanese, pressing the “Play Button” will render the translated text as Japanese speech.

### **Multilingual Speech-To-Text**

This feature present in the 'Speech to Text' page, allows users to choose the language they are going to speak in using the "From Language" drop-down (refer to Figure 8.1). The user can then simply click the "Mic Button" (refer to Figure 8.2) to initiate the process of vocalizing their content. The transcription of their speech is also displayed in real time in the "Type here" input box (refer to Figure 8.3).

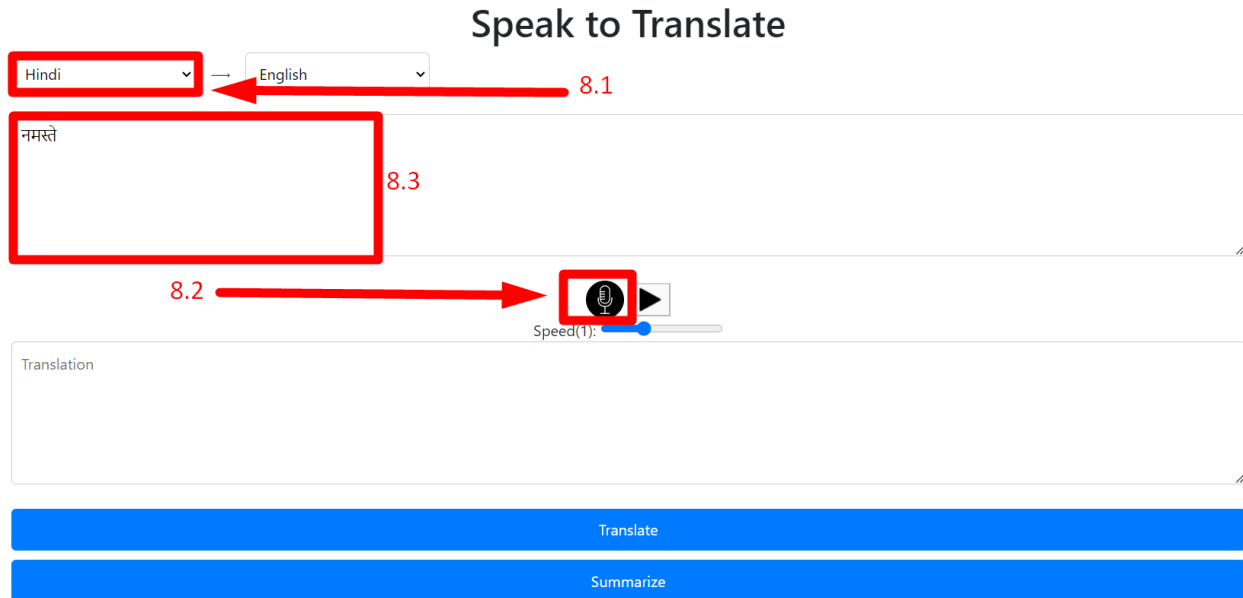


Figure 8 - Speech-To-Text

### Speech Voice Customization

This feature allows users to tailor their text-to-speech experience by adjusting the speech's speed. Once the user has translated their text, they can utilize the "Speech Speed" slider (Refer to Figure 9) to modify the pace of the text-to-speech voice. The speed can be fine-tuned from 0.5x (for slower speech) to 2x (for faster speech). Subsequently, users can click the "Play Button" to listen to their text at the selected speed.

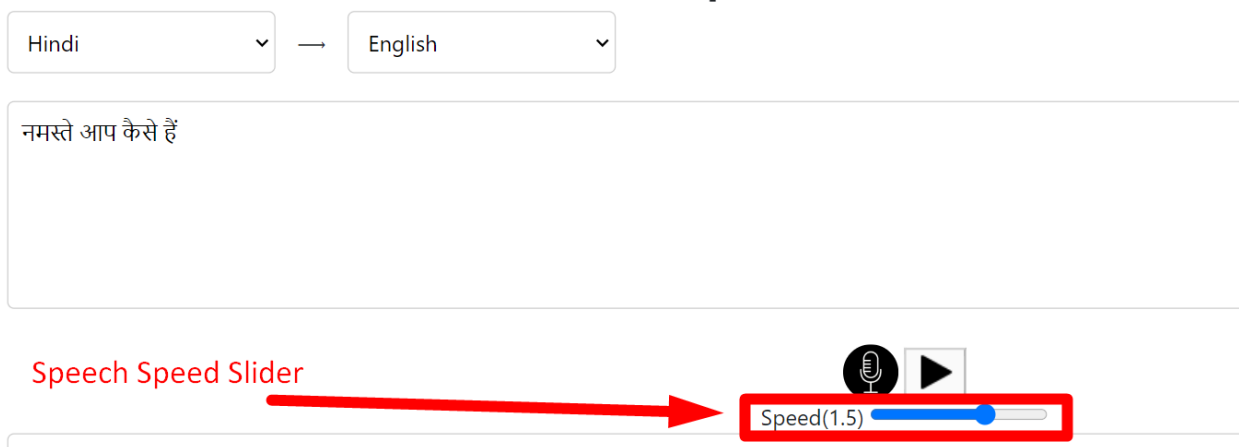


Figure 9 - Voice Speech Slider

## Tests

We employed the Jest framework to design a suite of unit and integration tests for our code. Our testing strategy comprised 8 unit tests, each assessing specific functions in isolation, and 6 integration tests, corresponding to distinct features.

### Unit Tests

#### 1) Is text provided by the user empty?

- This test validates the `isEmpty(input)` function, ensuring the "Type here" input box is empty after the user clicks the "Translate" button. It verifies the correct output of "true" when the input box is empty.

#### 2) Is text provided by the user filled?

- This test verifies that the `isEmpty(input)` function correctly outputs "false" when the input box is not empty

#### 3) Are languages chosen by the user the same?

- This test examines the `isLanguageSame(fromLanguage, toLanguage)` function to ensure it accurately identifies when the "from" and "to" languages are the same, preventing translation. It confirms the correct output of "true" in such cases.

#### 4) Are languages chosen by the user different?:

- Similar to the previous test, this one ensures the `isLanguageSame(fromLanguage, toLanguage)` function outputs "false" when the selected languages are different, allowing translation.

#### 5) Check if the language separator function separates the language code and displays the name properly:

- The language separator function accepts an array of language codes and names (e.g., `['hi-IN Hindi', 'en-EN English', ...]`) and divides them into two distinct arrays (`[hi-IN, en-EN]` and `[Hindi, English]`). This test ensures the function produces accurate outputs for the given parameters.

#### 6) Check if the helper function “`createOptionElement(languageCode, displayName)`” correctly creates an ‘option’ element with the right display name and value:

- The `createOptionElement(languageCode, displayName)` function accepts the language code and display name, creating an 'option' element with the value set to `languageCode` and the `innerText` set to `displayName`. This test ensures that the function generates the correct element with the appropriate value and `innerText` by utilizing assertions such as `expect(optionElement.nodeName).toBe('OPTION')`, etc.

**7) Check if the text input, from language and to language are the same as last time:**

- This tests the `isUserInputSame(input, fromLanguage, toLanguage)` function to ensure that the function correctly outputs true when all of the parameters are the same as the previous input the user provided. For example, if the user translates 'Hi' from English to French and does not change any of the parameters, an alert is triggered, stating, 'Please make sure you are not translating the same text in the same languages as the last time.'

**8) Check if the `showTranslatedText` function correctly displays the translated text in the output area:**

- This function checks if the `showTranslatedText(translatedText)` properly displays the correct text in the appropriate output area. This test uses JSDOM to create a test DOM environment and verifies whether the `showTranslatedText` function changes the innerHTML of the correct element.

## Integration Tests

We utilized the 'mock' feature provided by Jest to conduct integration tests for our functions, assessing their collective functionality. The 'mock' feature allowed us to create simulated API calls, eliminating the need to invoke the actual API during testing. This approach ensured that our tests focused on evaluating our code's performance rather than the API's. Below, you'll find detailed descriptions of the integration tests conducted for each feature.

**1) Language Translation:**

- The integration test for this feature mocks the output that the openAI API gives after sending a POST request. The mock returns a mocked version of the response and that is used to test if all the functions like `isEmpty()`, `isLanguageSame()`, `isUserInputSame()` work together by a single function called `getTranslation()`.

**2) Text Summarization:**

- This integration test also uses a mocked API call like the last one and checks if the `getSummarization()` button gets the summary of the correct text and if it correctly appends the summary to the innerHTML of the "translation" output box.

**3) Language Detection:**

- This integration test evaluates the functionality of the language detection feature within our translation application. The test scenario involves

translating the text "Hey, how are you?" into French while specifying the source language as "Detect Language." The expected outcome is an array containing the detected language and the corresponding translation. Calling the "getTranslation" function checks the integration of the detect feature with the isEmpty, isLanguageSame, isInputSame, and checkIfDetectLanguageUsed functions.

#### **4) Multilingual Text-to-Speech:**

- This integration test focuses on the interaction between the application and the SpeechSynthesis feature. The objective is to verify that the listenPlayPause function is appropriately called when the play button is clicked, ensuring the initiation of speech synthesis. The test begins by creating a spy, using Jest's jest.spyOn, on the listenPlayPause function within the global window object. Subsequently, it simulates a user action by enabling the play button (using JSDOM) and triggering a click event. The assertion validates whether the listenPlayPause function was invoked during this process, ensuring that the play button click indeed initiates the speech synthesis functionality. This test contributes to the comprehensive assessment of the application's integration with the SpeechSynthesis feature, ensuring that user interactions trigger the expected functionalities.

#### **5) Multilingual Speech-to-text**

- This integration test focuses on mocking the SpeechRecognition function to ensure that its start method is called as expected. The purpose is to verify that the speech recognition functionality initiates correctly when the start method is invoked. The test involves creating a spy using Jest's jest.spyOn on the start method of the SpeechRecognition prototype. Subsequently, it instantiates a SpeechRecognition object and calls its start method. The assertion validates whether the spy was invoked, confirming that the start method was indeed called during the process. By mocking the SpeechRecognition function and verifying the invocation of its start method, this test ensures the proper integration and functioning of the speech recognition feature within the application.

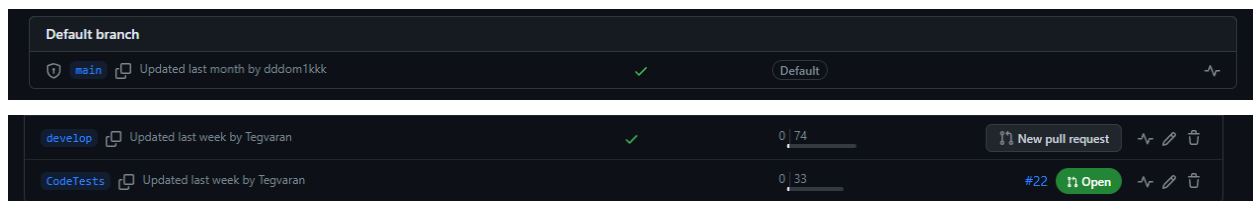
#### **6) Customization of Speech Voice**

- This integration test focuses on the seamless interaction between the changeVoiceSpeed and listenPlayPause functions. The primary goal is to ensure that updating the speech rate through the rate input triggers the expected changes in the application. The test scenario involves setting up event listeners (using JDOM), particularly for changes in the rate input. Upon simulating an input change event by modifying the rate value and dispatching the corresponding event, the test verifies whether the

associated elements, specifically the `rateText` element, are updated accordingly. The assertion checks that the displayed speed information (`rateText.innerHTML`) accurately reflects the modified rate value. This confirms that the integration between the voice speed customization and speech synthesis functionalities functions as intended.

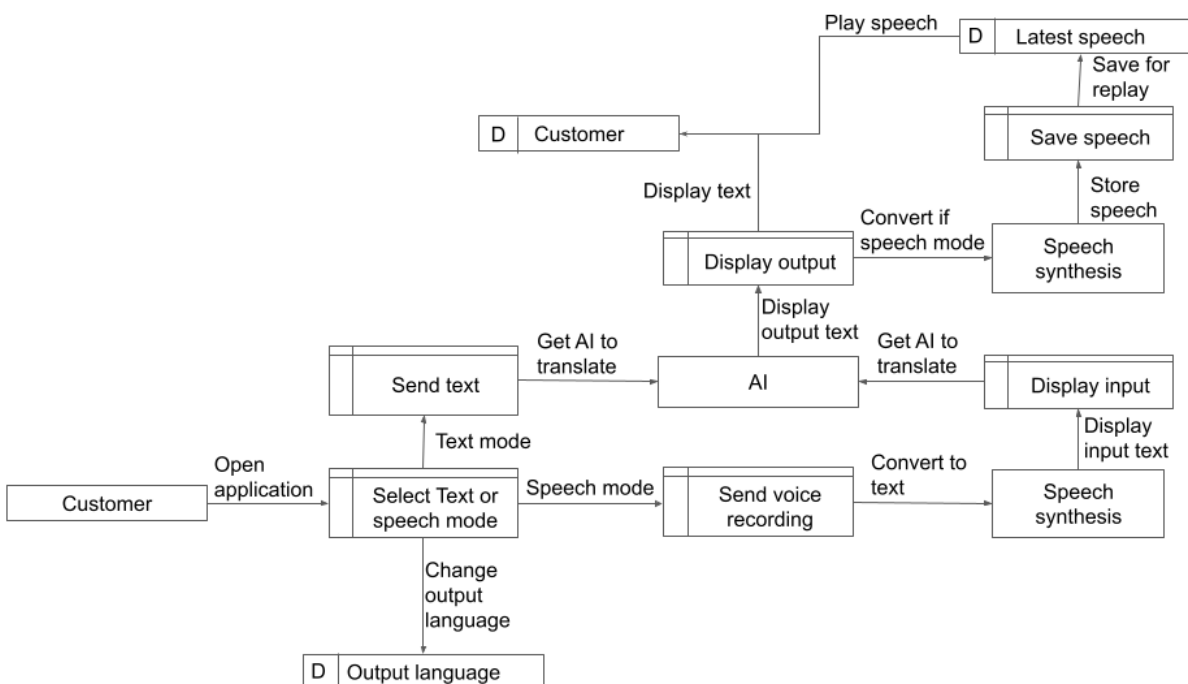
## CI/CD infrastructure

For our CI/CD infrastructure, we decided to have 3 branches to replicate the build, test, and deploy process. Our “develop” branch is where we all pull request our feature branches onto. There does exist the problem that one might accidentally overwrite/delete another member’s work while merging their changes onto develop. To counteract this, we have merge reviews to ensure not only the changes but the resulting merge will not cause issues. The “CodeTests” branch is where we sync our develop branch to ensure our code is completely functioning. And lastly, the main branch acts as a production branch, where changes that pass our quality assurance in CodeTests gets synced there and published.



We have also chosen to utilize Github Actions to perform automated test, build, and deployment. For our tests, we chose to use Jest due to its compatibility with testing our Javascript applications. This test should run for every pull request into a branch with this workflow. For deployment, we chose to use Github Pages for its ease of use and our universal understanding of it with the help of Assignment 3. The deployment step should run every time a branch pulls requests into main.

## Data Flow Diagram



*Figure 8 - Level 1 DFD*

Our level 1 diagram shows the data process with a much closer inspection relating to our system. The data starts on the customer on the bottom left opening the application and choosing speech mode or text mode. After doing so, the customer could also change the output language to the language they want their message to be translated to. As for the modes themselves, text mode can immediately be sent to the AI and quickly sent to the customer as text. Speech mode on the other hand has to be converted into text in order to be fed to the AI. Along with displaying the output as text, we will also convert the message to a speech with the speech synthesis and save it in order to be replayed by the customer.

## Reflections

One of the main challenges for our project is finding suitable APIs that fit with the criteria of our project. Because our application is focused on working with different languages, it became much more important to find a speech synthesizer that can adapt to different languages than just converting simple speech to text. This could've been fixed if we spent more time considering practical API that actually work with our application than just a general idea. Another problem we quite frequently run into are merge conflicts.



With more people in the group, each working on their own features, the likelihood of having a merge conflict skyrockets. While it may be difficult to completely mitigate due to the nature of code frequently changing, we could commit and pull often to ensure our code is up to date. The noticeable challenge we had was organizing times around everyone's schedules. We're all taking many different classes on top of this one, which can affect when and how frequently we can talk with each other. Not only that but the amount of assignments being handed out during the last few weeks even made some of us too busy to work on the project. There isn't much we can do to counteract problems with our schedules but it can be useful to perform risk management in case something goes wrong.

## How work was divided up

### Tegvaran:

- Worked on the functionality of the openAI API
  - Type to Translate feature
  - Detect Language Feature
  - Summarize text feature
- Wrote all the unit tests
- Wrote all the integration test for OpenAI API
- Worked the features part of the report
- Recorded the demo for presentation

### Damir:

- Worked on the functionality of the SpeechSynthesis API
  - Speech-to-Text feature
  - Text-to-Speech feature
  - Change speech speed feature
- Wrote all the integration tests for SpeechSynthesis API
- Worked on bug fixes

### Victor:

- Level 0 and 1 data flow diagram
- Presentation slides 5 and 6
- Write up for project challenges, data flow diagram, and CI/CD in report
- Solving merge conflicts when others are busy

### Zheyuan:

- CSS for the whole website
- Worked on the integration and unit tests part of report
- Review the documentation