Text-to-Speech Documentation

**Introduction:**

The Text-to-Speech is an API that allows users to enter in text which is read aloud to them in the form of a neural voice. This is possible by using the Microsoft cognitive services. Specifically, the text-to-speech API that this current API is connecting to. To summarize, it allows text to be converted into human-like voice using prebuilt neural voices that Microsoft Azure provides.

**The API: http://137.184.143.83:3000**

There are three parts to this current API.

* The UI
* The JavaScript jQuery code
* Backend NodeJS JavaScript code

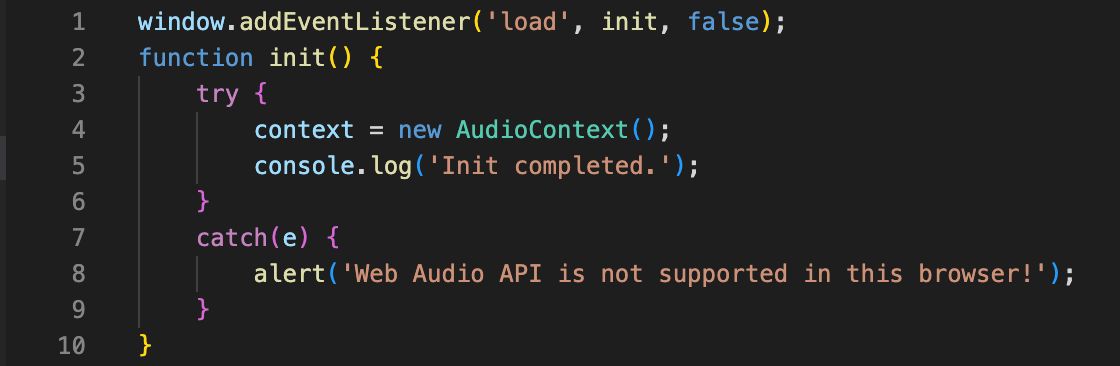
**The UI**

* **Audience: Managers and Directors** 
  + **People who need a general understanding of this part of the application.**
* The UI consists of two files. These two files are combined to show the following UI to the user. In the big text box labeled “1”, the user enters the text they would like to convert to speech. The thing to note here is that:
* The text needs to be in English. This is because it is not a translating tool, rather a text-to-speech tool.
* The text cannot be extensively long. Anything that is longer then 10 minutes, will not be read. This is hard to determine so it is suggested to divided any big parts into smaller paragraphs or sections and then convert to speech in order.
* Once the user enters in the text and they are ready to convert to speech, they hit the button on the bottom titled “Speak” and labeled as “2.” Once they do that the device which the user is using, will play the converted text speech. Essentially, it will read out the text to the user. Therefore, it is important that the user:
* Makes sure that their device volume is not on mute or low.

2

1

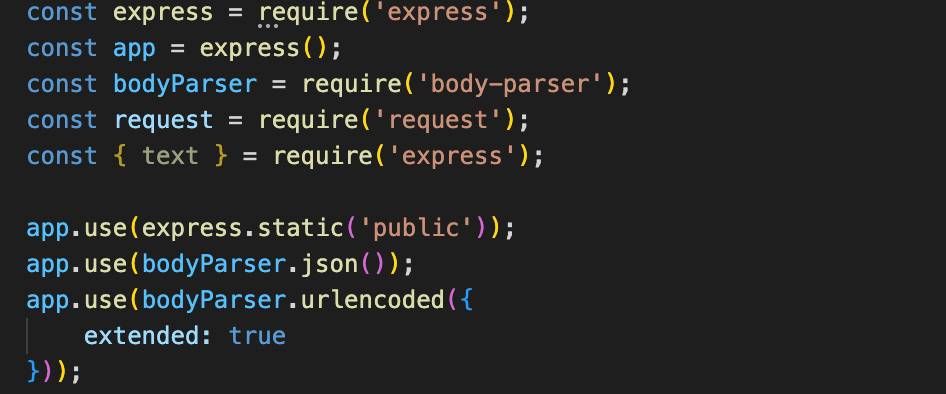
**jQuery Code**

* **Audience: Managers and Directors** 
  + People who need a general understanding of this part of the application.
  + The jQuery code essentially does two things. The first, is that is it pulls the text that the user enters and sends it Azure to convert to text. The second, is that it creates an audio context for the sound to be played on the browser. To play the sound, it utilizes the Web Audio API.
  + For more information, on the Web Audio API, please refer to the following article:
    - <https://web.dev/webaudio-intro/>
* **Audience: Technical Side**
  + Looking at the first half of the code, this where we create an audio context to handle the incoming audio file that Azure returns. Since an audio context, can handle multiple sounds, we only need to create the one.
  + The second half of the code does a couple of things. The play function is triggered when the user hits, the “Speak” button after inputting the text. The code explained in order:
    1. We send a XMLHttp request to retrieve the sound file.
    2. We then, retrieve the entered text from the **“textarea”** and assign it to a variable called **“input\_text”.**
    3. We create a variable called **“convert\_url”** and assign it the text value of the endpoint we have created on our backend, which will be discussed in the backend section, along with the ability to add a query (the inputted text). We do this using “?input=” and the **“input\_text”** variable which houses the user’s inputted text.
    4. Then we create a new get request and enter the **“convert\_url”** variable as our endpoint.
    5. Since, the response type is binary we set the **“request.responseType”** as **“arraybuffer”.**
    6. We create a function to decode the audio file.
    7. We again create the audio context to handle our sound.
    8. The **“audiosource”** variable is created to create a buffer source, which creates a sound source.
    9. The **“audiosource”** is then connected to the destination, which are the speakers of the device.
    10. Finally, we use the **“decodeAudioData”** method to decode the file and play the sound.

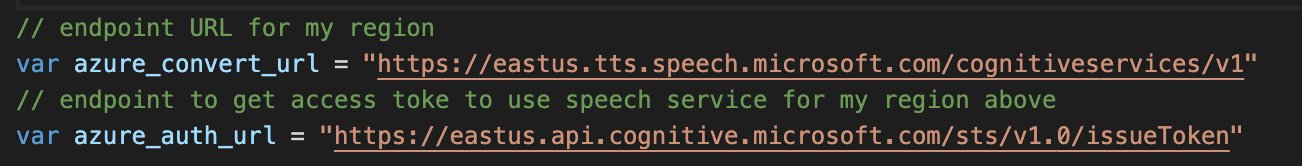


**NodeJS Backend**

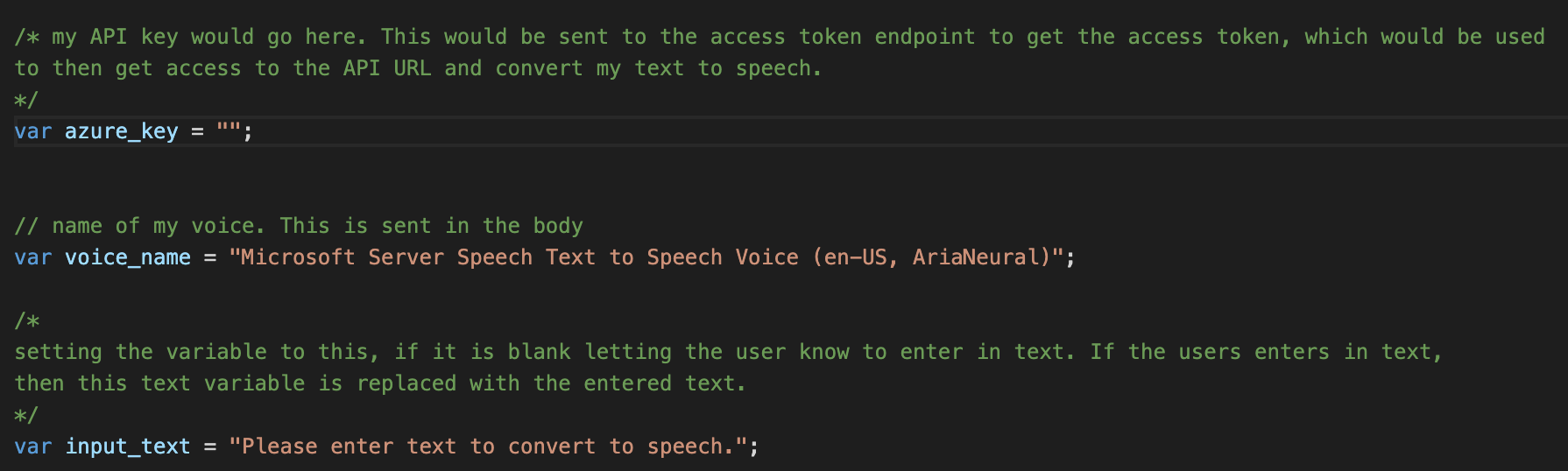
* **Audience: Managers and Directors** 
  + People who need a general understanding of this part of the application.
  + The backend code uses NodeJS to create endpoints and use those endpoints to connect to the Azure endpoints in order to send textual data and get back audio. Essentially, we use the Express framework to create REST APIs. Please see the technical section to understand how the code works.
* **Audience: Technical Side**
  + This is where we install the dependencies and the packages which are used in the application.

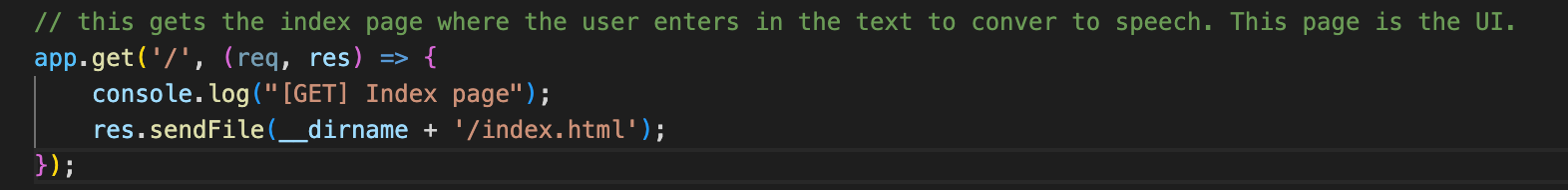
****

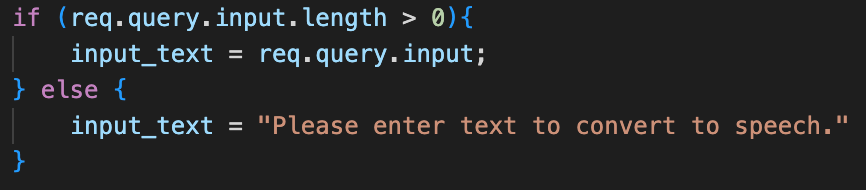
* The two variables below store two links.
  + The first variable stores the endpoint, which is used to send the text to be converted to speech. This endpoint is specific to a region. You can check the following link to get more information on supported regions.
    1. [**https://learn.microsoft.com/en-us/azure/cognitive-services/Speech-Service/regions**](https://learn.microsoft.com/en-us/azure/cognitive-services/Speech-Service/regions)
    2. Link to the prebuilt neural voices by region and endpoint.
       - https://learn.microsoft.com/en-us/azure/cognitive-services/Speech-Service/rest-text-to-speech?tabs=streaming
  + The second endpoint is the one that is used to get the access token by sending in the subscription ID, which is generated when you subscribe to Azure.

****

* There are three variables here:
  + The first is the place where the subscription key would be entered as a text value. This is blank in order to not disclose it.
  + The second, is the name of the voice that is being used. There are a number of different voices. The way that this is written has to be this way because it goes into the request that is going to be sent to Azure, in the request body. This allows Azure to know the sound that is required.
    1. List of voice styles: <https://learn.microsoft.com/en-us/azure/cognitive-services/speech-service/language-support?tabs=stt-tts#prebuilt-neural-voices>
  + The third, is a variable that sets the input text to “Please enter text to convert to speech” if the user does not input any text. Therefore, this sound is played by the device letting the user know to enter in text.

****

* The following is the endpoint to present the index.html page. This is where the user enters in the text.
* The “convert” endpoint is the one connects to the Azure API and sends the inputted text to be converted to speech. Here we look at it in different parts:
  + The first set of lines for this code is an if-else statement. It checks whether there is any query text attached to the **“convert”** endpoint. If you remember from the jQuery code, we pass the input as a query within the URL. Therefore, the if statement says that if the query length is greater than zero then we set the **“input\_text”** variable equal to the query. Otherwise, we keep the input text to the pre-set variable letting the user know that they need to enter text.



* + Next, we need to get the access token to utilize the Azure software to convert the text to speech. Therefore, we need to make a request to the issue token endpoint. The endpoint formatting is below and is based on the region identifier. This is already stored in the “azure\_auth\_url” variable. A post request is sent to the endpoint with the proper formatting for the headers. This is where the subscription key is added. Here is the list of region identifier if not known:
    1. <https://learn.microsoft.com/en-us/azure/cognitive-services/Speech-Service/regions>
    2. https://<REGION\_IDENTIFIER>.api.cognitive.microsoft.com/sts/v1.0/issueToken



* + Once the authentication is complete and successful, the endpoint sends an access token as a response. This is done if the subscription key is valid and correct otherwise it sends back in error. Once we retrieve the access token, we need to send it to the API endpoint, mentioned above along with the text to be converted to speech. The access token works as a key to unlock the Azure software. To do this:
    1. We create a function to handle the response from the request that we sent above. We need to handle two types of response. The first being a success response with sends back the access token. The second being an error in case the subscription key was incorrect. To handle a success response, we create an if statement where we say, if the response status code is equal to 200 (success status code), then we create another request inside this if statement. This request is a post request to the “azure\_convert\_url” variable. The formatting of this request can be found in the below link under the **“Sample Request”** section. To summarize: for the headers we have to send the access token with the correct title formatting, the Microsoft Output Format for the sound output, and the content type. Cache control has also been added to control allow no caching. For the body, we have to include the formatting along with the type of voice we have selected and the input text. This information is stored in two variables **“voice\_name”** and **“input\_text.”**
       - <https://learn.microsoft.com/en-us/azure/cognitive-services/speech-service/rest-text-to-speech?tabs=streaming>
    2. Since we are creating another request, we need a nested "if-else" statement within our parent if statement, to handle the response. If there is an error, we want to print the error in the console log. Otherwise, if the response code is 200 (successful) then in the console log, we print successful and our browser automatically "plays" the text, which Azure converted to speech. This set of function and if-else statements is inside our parent if statement.
    3. If there is an error, from the access token endpoint, then we create the else statement we want to print the error in the console log.



3

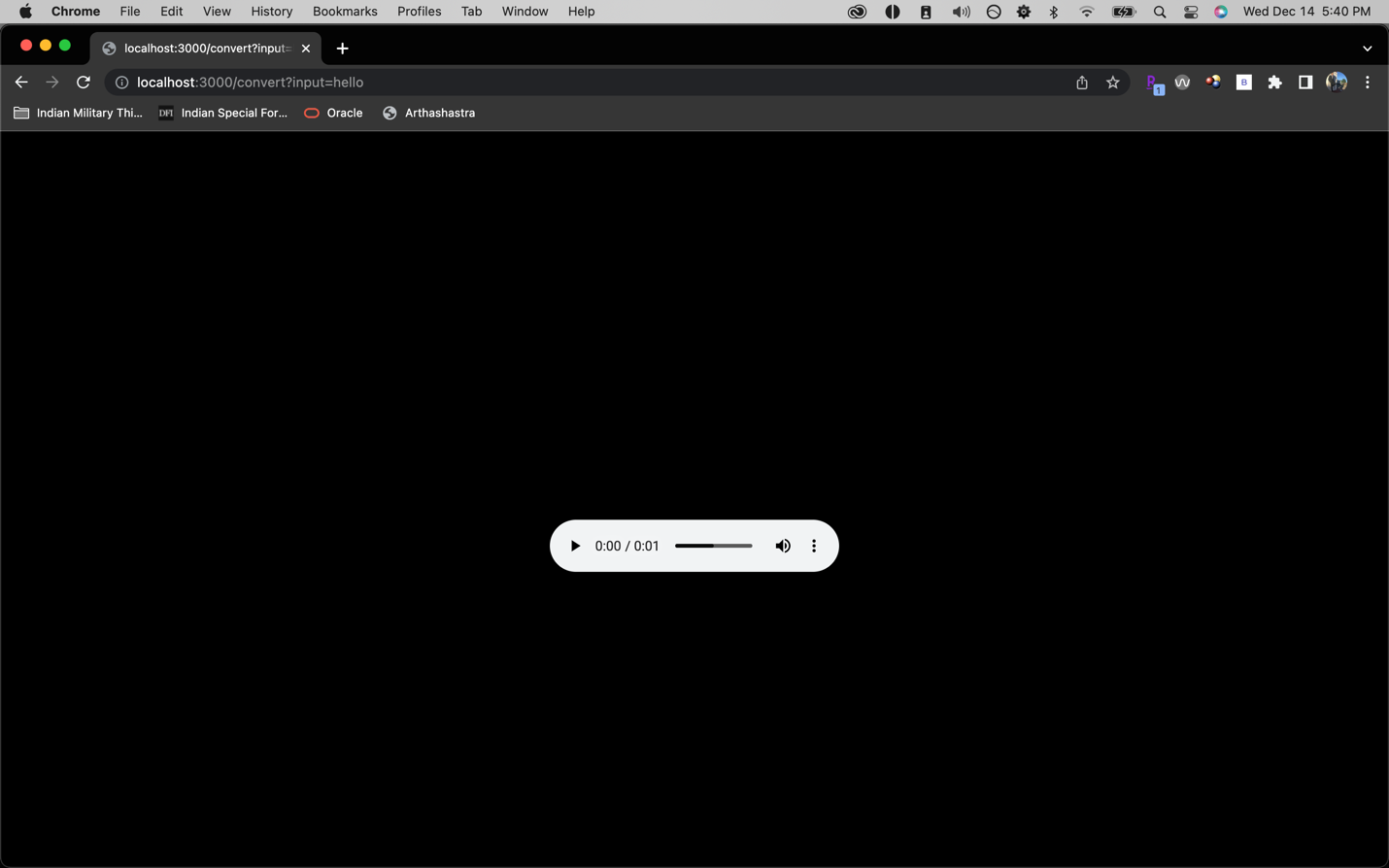
2

1

That’s it! That is how the Text-to-Speech API works!

Note: For those who want to access the API, there are three ways to do so.

* First, you can use the UI to enter text and hit the “Speak” button, this is the easiest option.
* You can create your own UI page. This UI would need to send the inputted text to the endpoint as a query, which would then handle the rest:
  + /convert?input= and add the text after the equal sign.
* The third, does not work with Safari because of its tight privacy and content settings, however, you can just access the convert endpoint like above, in Chrome. This would then lead to a page like below. This page would allow more controls, like play and pause, the ability to download, and also control the playback speed.



Also, note that the domain name is not registered so right now it is just the IP address. http://137.184.143.83:3000