I first started off the project by reading up on the Google translate API. I read the forms they offered on getting started and how the request to the API would be handled. I continued to read the documentation of their API and looked over some examples in various languages on how to make a request to the API. This took some time, but when I finished I went to make an account with Google to obtain an API key so that I could call the API when I wanted. I made an account with them and had my card linked with it so that I could make use of the service.

From there, I decided to use the CS50 Finance problem set as a backbone for my project. What I intended to do was to have the website built from scratch, but after about a day or so of trying to get the website the way I wanted, I still was not satisfied and realized that I would then still need to implement most of the features from CS50 Finance into my project. So after some time, I decided to use the problem set as a starting point. I copied the problem set and renamed it cs50project. From here, I decided to make the index page the home page, so I would work from there. I then began to create the aesthetics of the page. I looked online on how to create a PNG logo for my website and ultimately decided upon a design and size.

From there, I needed to create a textbox for the user to input their text. I searched online for how I might style the textbox and after some time ended with the current design. I then created tree of these to have a space for the the user to input a text and then have a two other textboxes for Google API's translation of the text and Microsoft API's translation of the text. Then, I went online to find images for Google and Microsoft respectively to place above these fields to associate them with the translation. I finally came across PNGs for Google and Microsoft and used these as the images.

After having these basics done, I set to trying to get the Google API to work. I started off with just trying to load the page and have a button that would translate a hardcoded given translation from a hardcoded given language to a hardcoded given language. When trying to do this, however, I ran into some problems with calling the Google API service so I contacted them and saw that there was a problem with my billing. Once I got that taken care of, I could call the API with all of the hardcoded texts. I then moved to remove the hardcoded text. I looked into jQuery commands that would let me access the values in the textbox and then pass this value to the function that would call Google's API.

Once this was done, I realized the best next step would be to implement a dropdown menu with the different languages supported by Google. I then go create the dropdown menu and stylize it with some help from online stylesheets. I then search through Google's API again to look up the various language codes by which the API uses to refer to a language. I grab all these values for the languages and set them within the dropdown menu. (I ultimately had to alter these to support languages that were supported by both Google and Microsoft of course.) I also set a default to no language. (I attempted to make the default language the language of the user visiting the page, but the codes did not match up and would not work.) From here then, I used jQuery commands to obtain the value (the language code) of the dropdown menu for the language from which translate should be done and the language to which the translation should be done. From here then, I was able to factor out the hardcoded language codes and was able to dynamically make a request to Google's API. This, however, was all done using the REST method of the Google API, which I would later remove. What I did not find appealing about the REST method was that it had a function which had an outside callback. I did not like this because that callback is what ultimately placed the translated text in the desired location, so if I called the translator from a different page I would not get it into the desired location. I left this for the moment to move onto the Microsoft translator API.

I began working on the Microsoft API by reading up on the translator and how it worked, i.e. how to call it and what it hands back. I then created my account to access the API and got myself the secret needed to use the API. Once I had all that non-interesting stuff out the way I saw that the Microsoft API was a lot more different and seemingly difficult because instead of just sending off the request to Microsoft with my secret key, as would be the analogous case with Google, Microsoft's API works differently in that it requires the use of an access token that is only food for ten minutes at a time. Thus, you need to constantly be asking Microsoft for an access token to have that on hand so that when the Microsoft translate API is called it has a verifiable access token.

I read over the examples of how to get an access token from Microsoft in different languages, but since I was working in JavaScript, one of the languages not offered in full by example by Microsoft, it made things a lot more difficult. Nevertheless, I read through the example codes of the other languages to get a feel about how to call Microsoft to obtain an access token. After some time, I thought about how I might implement the code in JavaScript. I began with tackling the refreshing of the access token and storing that in a global variable to be on hand for the later Microsoft translate API call function to work. I made this function work when the page loaded and then request another function to be written to grab the access token. This function would then call that function to get the token every nine minutes instead of ten just to be safe. I then created a PHP controller to handle the request, which I thankfully found the Microsoft sample code to be of help. This controller would then be called by the get token function with an ajax call to, on success, update the global access token variable. From there, I began work on the Microsoft translate function to call Microsoft with the access token in much the same way as the process of the Google API.

I found the best way to work with this API was to use an ajax call to Microsoft from looking at their documentation. I began working on implementing the API with hardcoded values as in the Google API before working on the ajax method though. Once I had that done and got back a response from Microsoft, I began working on removing the hardcoded values out, first the text value and the languages. Once I had that done, I wanted to implement basic user compliance checking to make sure a text was provide, a from language was chosen, and a to language was chosen. I then saw to having the place where the translated text should go out from being hardcoded.

Ultimately, the way I saw that I would do this is to place both the functions of the translators and their callbacks all within one function. I first got started on Google's translator. What I I ended realizing is that the REST method of this translator makes it a bit difficult in that it has the callback function within a url so another way of implementing the translator would be needed. I began looking into the translator API for Google again and saw to implementing an ajax call to Google. Once I had the ajax call working but having the callback on the outside, I was able to bring the callback on the inside to alter text in a text area by having the function of the callback run on success of the ajax call. This allowed me to pass a variable of where the translated text should go to the Google translate APU function call.

When working the Microsoft translator API ajax call, I assumed it would be relatively easier to implement than the Google translate variant because I had already implemented an ajax call so all I would need is to have an on success function replace the callback function. This, however, failed to work. I found why it was not working by adding an on error call of the ajax call in the Microsoft translate API function. After a bit of digging around again, I found that there are different ways of implementing a similar feature to on success in an ajax call. I saw that

there were things like .done and .fail which would help me out, so I implemented them this way and managed to have everything for the Microsoft translate API function within one function, which allowed me to pass a in a variable to where the Microsoft translate function should place its response.

After getting this done, I moved on to comparing the accuracy of the two translators against each other by creating a page for them to show their responses. The ultimate goal was to have the interface accept English text and have this text run through all the supported languages, printing out the translated text of the language, printing out this translated text back to English, then have the previous translated text run though the next language and have this repeated throughout all the supported languages for both Google and Microsoft.

Ultimately, I had all the code written down for implementing this, but what I failed to realize is the asynchronous nature of the callbacks of the two APIs. I had seen that the responses were not getting back to me in the correct order when using console.log to check this, but I thought the fix might be easy. I posted on cs50 discus, but got back no answer for a while, so I posted on stack overflow and again got no answer for a while. Finally, after I had done some more searching around online on how to get these functions to get back to me in order and get the loop I had going to run through all the languages in order, I saw an answer coming up more and more on implementing a promise. Ultimately, this is the same answer as I got back from cs50 discuss. I read up online about JavaScript and promises, but it seemed really confusing. Nevertheless, I tried on implementing this anyways. I tried and tried, but I could not get my head around this. I was hoping then that I would receive some help at the hackathon, but ultimately got none, as some tfs would not know what promises were or would give me different suggestions on how to run the code. Ultimately, I tried to implement the code using promises; i tried event listeners, i tried recursive loops, i tried entering a loop that would only break when the callback was executed by changing a global variable; i tried a library called waterfall to help make promises relatively easier. Ultimately, I was not able to get the code to work; the closes I got was to have my original code and having the loop that would only break on completion of the callback, but this proved to freeze the browser. Defeated, I moved on implementing the code to how I originally proposed and have the translators translate once and back.

Even with this method though, I ran into the problem of not having the callbacks return their response in time. To get around this, I had the second calls to the translator APIs on completing of the first in their ajax calls. To do this though, I had to create global variables of the from language and of the to language, as well as the method in which the translator should work, in that method differs in that one method would be for the first page to have the translated text replace the text in a textbox and the other method two methods dealing with the first and second translations of the text. Then, I had to decide to include two method variable for Google and Microsoft instead of only one so that if one translator got back faster with its response that the method would not be affected.

As the last of the implementation, I integrated the current date and the current time, and about me page with some cool links, and the more difficult feature of history in passing the variables of the from language, the to language, the original provided text, and the translations from both Google and Microsoft from JavaScript to PHP.

Given that this has been quite long, I will end it hear and ay that I am still working on trying to get the accuracy page to be even more robust. We will say what I further accomplish.