Name: Pallavi Shirodkar

Student ID : 801200666

ITIS 6177 Final Project

User Manual for Azure Translator API

https://azure.microsoft.com/en-us/products/cognitive-services/translator/

Introduction.

Project objective is to create and implement our own API endpoint so that a simpleton user can bypass all the complex procedures that Azure API(s) require and rather use the simple API endpoint provided by the end of this project.
The project has been designed in a way that the same API endpoint has been exposed for fair use by Postman as well as a Website.

Azure API consumes a target language and a sample text and converts the entire provided text into the desired target language. Additionally the users can provide sample text in some other language than English if they can type the input in the other language.

Tech Stack.

Entire code base has been developed from scratch using the following technologies:

1. Node.js + Express.js for writing the API endpoint.
2. React.js for creating the front end. React Semantic UI for enhancing the CSS.
3. Azure Translator API for data processing. (Text translation)
4. Digital Ocean to host the entire application.
5. Git for Version Control.

How to use the Product.

There are three ways in which a user can use the product. One is by directly creating and setting up an Azure account, creating resource groups, generating keys, building the backend as directed in the Quick Start guides, etc (A quite tedious process actually).

Other two ways simpler and less-energy-sapping ways are mentioned below:

1. For users who don't prefer(/ like) my simple, intuitive front end and prefer the Postman:

It is simple! On the postman app create a GET request. On the following url:

http://165.22.191.215:3000/translate?from=en&to=de&text=Hello!%20This%20project%20has%20been%20really%20interesting%20so%20far!%20I%20have%20enjoyed%20and%20learnt%20a%20lot%20in%20the%20process.%20

The result of this url is the Project!
Basically 165.22.191.215 is the droplet I created on Digital ocean and 3000 is the port I am using. "/translate" is the endpoint that I have created.

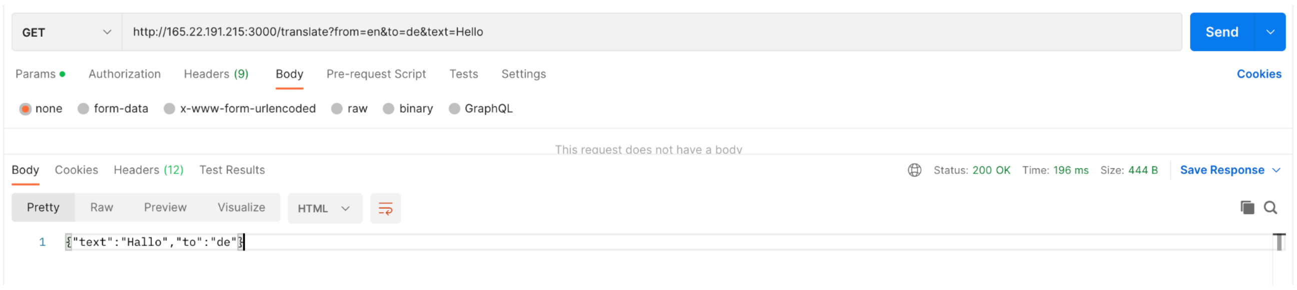
To form a request you will need to put together following things:

1. http://165.22.191.215:3000/translate?
2. mention a mandatory "to" in the query parameters, which is basically a target output language. Now in "to" codes for languages are expected - not the names of the language. (Language Name- Code conversion has been taken care in my front end) These codes can also be found on the following link: Language Codes
3. Another essential query parameter is a sample text which goes by "text". Which can be as simple as a "Hello!" in English. However if there are spaces between, it needs to be taken care of like in the above url. (Again.. It has been taken care of in my language…. Please use it! :))

So a simple query formed will be as easy as:

http://165.22.191.215:3000/translate?from=en&to=de&text=Hello

Here, from has been provided as in the front end, I am providing the option of choosing a starting language as well. “to” is pointing to “de” which is German Language.
This GET call returns the following simple result in the postman.



```
GET http://165.22.191.215:3000/translate?from=en&to=de&text=Hello
```

This request does not have a body

Status: 200 OK Time: 196 ms Size: 444 B

```
{"text":"Hallo","to":"de"}
```

In words:

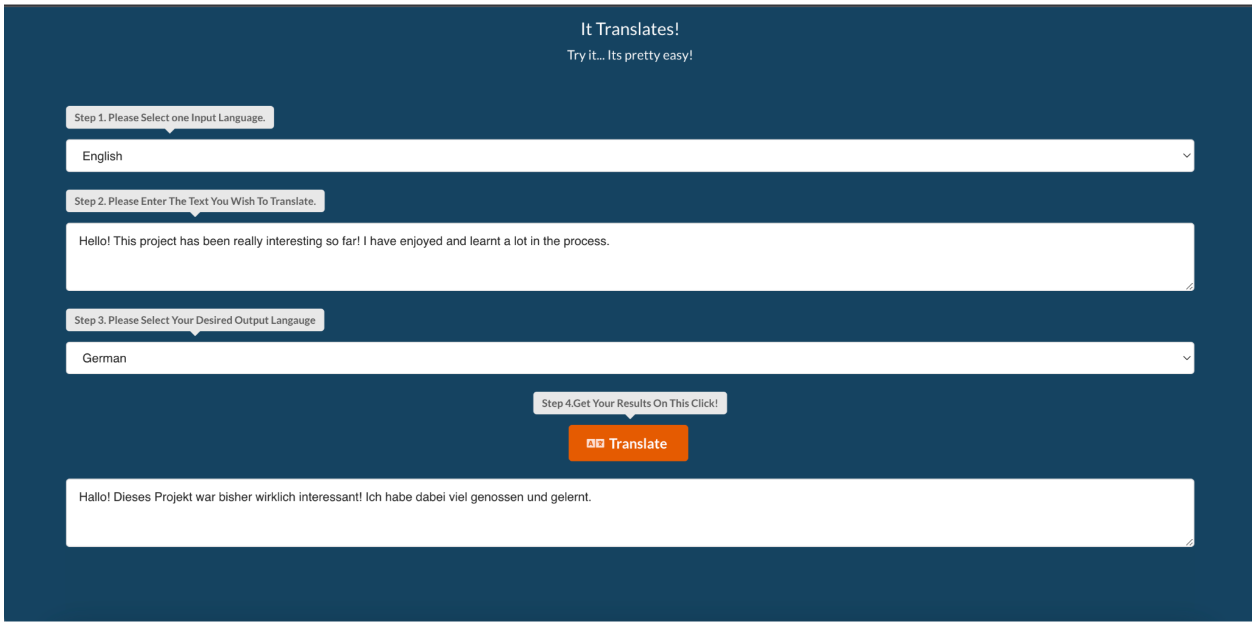
{"text":"Hallo","to":"de"}

Now this response is actually quite a simplified version of the actual monstrous big, complex response that we receive from Azure.

That's it! That's the project! In other words, you type the words you need in the text, choose a target language and all the magic in the backend gives you this finished product.

After covering the basic idea lets see the easier way to use this functionality!

2. Using the integrated frontend.



It Translates!

Try it... Its pretty easy!

Step 1. Please Select one Input Language.

English

Step 2. Please Enter The Text You Wish To Translate.

Hello! This project has been really interesting so far! I have enjoyed and learnt a lot in the process.

Step 3. Please Select Your Desired Output Langauge

German

Step 4.Get Your Results On This Click!

Hallo! Dieses Projekt war bisher wirklich interessant! Ich habe dabei viel genossen und gelernt.

It is a simple front end with simple prompts asking users to choose appropriately in 4 simple steps.

You can also check the same on : http://165.22.191.215:3000/

This static react build has been integrated with the Express server so that the front end will call the express backend which in turn will call the Azure APIs like it happens in the real world scenarios.

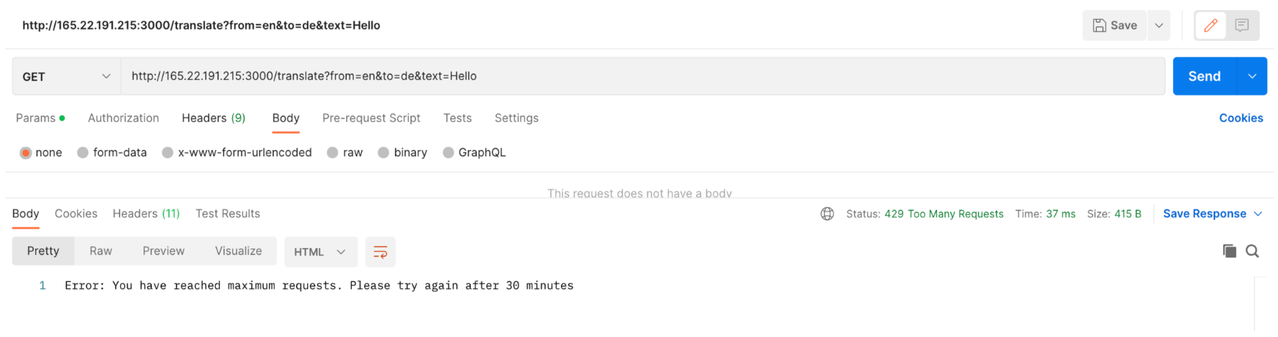
In 4 easy steps, the desired result can be obtained without bothering the create a request URL and taking care of all the parameters!

1. Choose the source language from the drop down.
2. Enter Sample Text to be translated
3. Choose the target language from the drop down.
4. Click the translate button so as to get the desired translated text.

UNIVERSITY OF NORTH CAROLINA CHARLOTTE THE GRADUATE SCHOOL

Validations

1. On postman:- Regarding Validations, little yet important has been achieved on the server, there is restriction placed on the number of times this service can be tested via postman! "30 times". After 30 attempts to run the translator service, for the 31st attempt through SAME machine, Error **429** of **"Too many requests"** will be shown indicating a user friendly message as shown below:



Status: 429 Too Many Requests Time: 37 ms Size: 415 B

```
Error: You have reached maximum requests. Please try again after 30 minutes
```

A **middleware** has been implemented for this purpose to limit

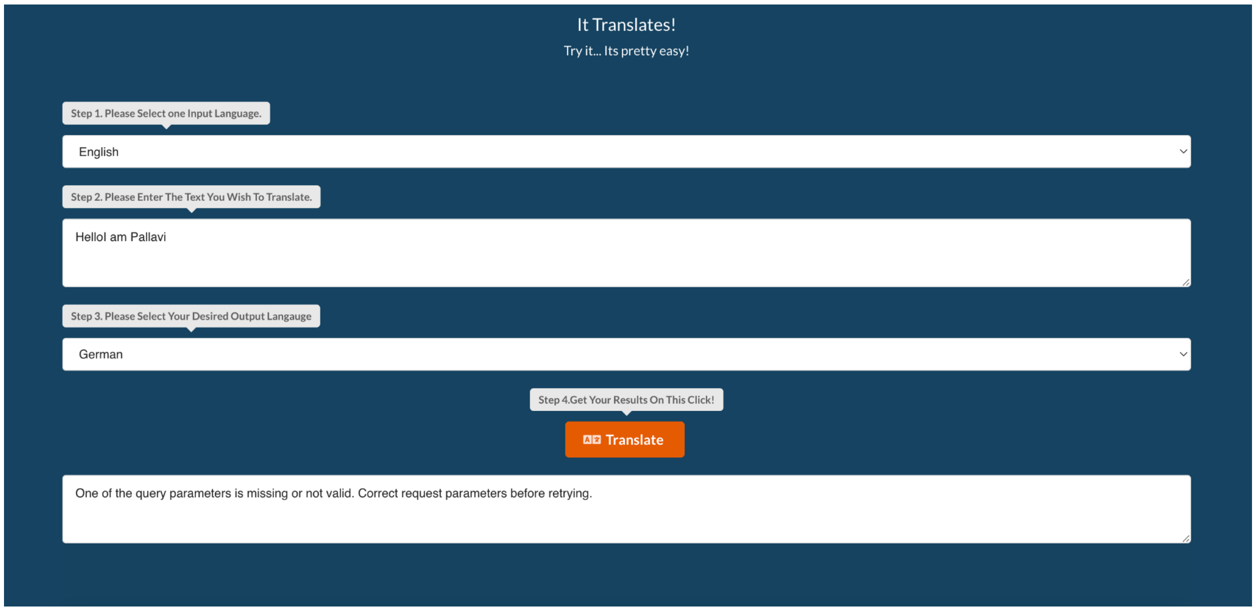
2. On the frontend however, more validations can be imposed because of the control over User Interface.
   a. There is a check for the same languages. If "**from**" and "**to**" languages are the same in the dropdown the translate button will remain disabled! Because what's the use of translation then!
   b. If both languages are the same yet **input text is empty** then the translate button will remain disabled.
   c. A local storage has been initialized for each machine - a simple attempt to limit user requests per session to **50**. This request count resets after session is refreshed.

Error Handling.

All the errors implemented are very user friendly. Following is the list of errors implemented in the project:

1. Error 400: When one of the parameter(s) is wrong. Which is the most common case.
2. Error 401,403: These errors will usually not arise because they are related to the Azure user authentication and will be shown when the user no longer is eligible to receive the services from Azure.
3. Error 429: As mentioned above, when too many requests are formed within a small period of time - which is usually a malpractice, this error will be shown.
4. Error 503: When server is down/ unable to serve the requests.
5. Error 500: A generic error.

In postman the errors will be in the form of the JSON response(as usual). In the UI, errors will be shown in the place of the output area as shown in the figure below.


It Translates!
Try it... Its pretty easy!
Step 1. Please Select one Input Language.
Step 2. Please Enter The Text You Wish To Translate.
Step 3. Please Select Your Desired Output Langauge
Step 4.Get Your Results On This Click!
Translate

English

HelloI am Pallavi

German

One of the query parameters is missing or not valid. Correct request parameters before retrying.

UNIVERSITY OF NORTH CAROLINA CHARLOTTE | THE GRADUATE SCHOOL

Recreating the errors.

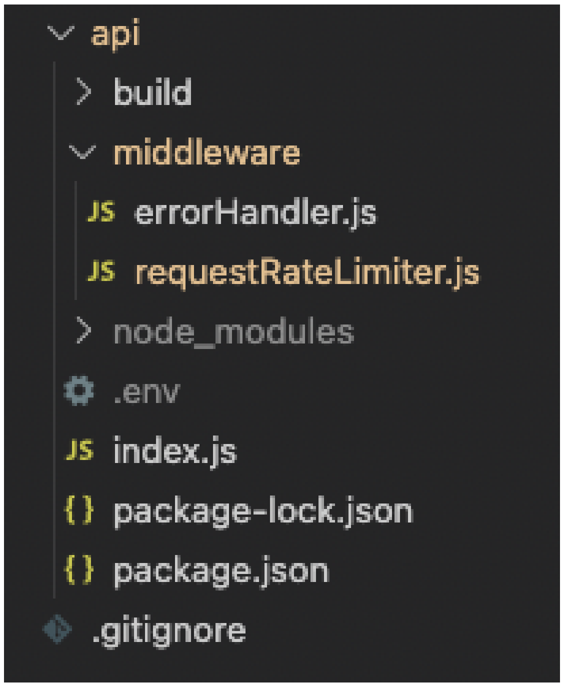
1. To recreate the errors in the postman:

   You simply have to alter the required parameter names

   eg: “to” becomes “xyz” or something else, “text” becomes “abc”.

2. Rest of the errors are server based. However the Error 429 can be obtained on postman by trying to request the same resources again and again over 30 times and on the UI over 30 times. Also on the UI after the 30th attempt the “translate” button will be disabled as well.

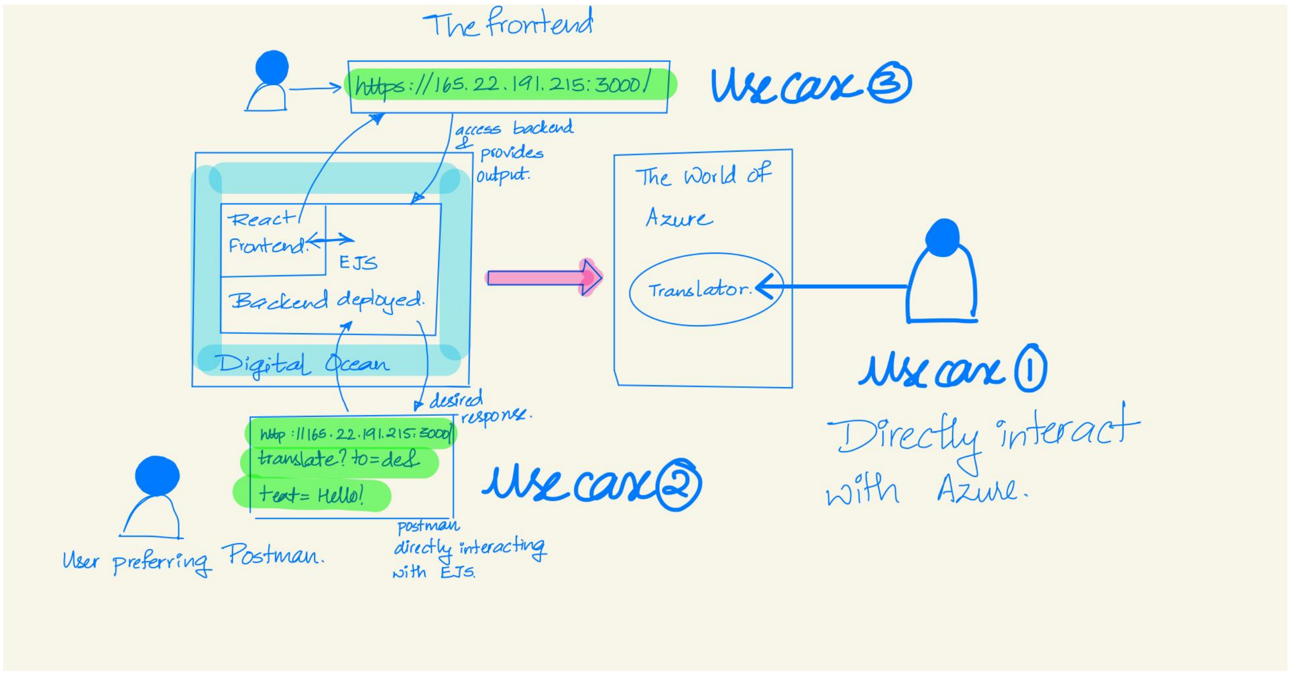
Folder Structure Of the Project.



```
api
  build
  middleware
    errorHandler.js
    requestRateLimiter.js
  node_modules
  .env
  index.js
  package-lock.json
  package.json
.gitignore
```

- The “api” is the root directory. “build” is the static part where react build stays. It is served on the default URL. The middle ware handles the request rate limiting and error handling. “.env” file is where the secret key for Azure resides which is hidden from all the git commits.
- The Git Repository for the entire code is: Backend Repository.
- I have also implemented the front end separately on my local machine and deployed a static build of the react on the Express server. The code can be found on: Frontend repository.
- As with the usual practice node modules have been ignored from any git commit.

Project Architecture in simple words:


The frontend
https://165.22.191.215:3000/
Usecase ③
access backend & provides output.
React Frontend.
EJS
Backend deployed.
Digital Ocean
The World of Azure
Translator.
Usecase ①
Directly interact with Azure.
desired response.
http://165.22.191.215:3000/
translate?to=de&
text= Hello!
Usecase ②
postman directly interacting with EJS.
User preferring Postman.