RESTful API Project

Course: INFO U762 – 01F

Semester: Spring 2024

Instructor: Dr. Grover Walters

Date: April 3, 2024

David C. Miller

**RESTful API Project**

Introduction

I started looking at the provided public API list. I did not want to deal with Authentication, so I copied the list to a spreadsheet and identified the ~700 that did not need Authentication. I started narrowing these down looking for three that would work together to tell a story. I started to focus on travel, currency, weather, and food, as possibilities. After reviewing the types of data that I have access to, I decided to build a travel advisory API for the City of Los Angeles, USA. This type of API, could be added to the City’s homepage and could be modified to support other cities.

My next task was to dig into the specific APIs for each type of data that I could collect. For weather, I started with the National Weather Service (NWS) API, provided by National Oceanic and Atmospheric Administration (NOAA) (API Web Service, n.d.). The NWS API site had extensive documentation on how to pull data based on various input values. Based on my use case of provided Weather related data for Los Angeles, I chose the input value for station ID. Another part of the website provided me with a map of weather station IDs and I confirmed that Los Angeles was KCQT.



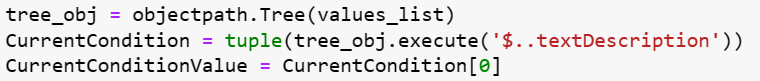
I moved this information to my Weather Rest API Jupyter Notebook at and confirmed that I would be able to pull the data correctly. Out of the data required, I decided that the two values of particular interest would be the “current weather”, which appears in the ‘testDescription’ value and “current temperature” value, which appears in the ‘temperature’ value. Since the temperature value is provided in Celsius I will need to convert that to Fahrenheit so that I can provide both values to visitors. With this information in hand, I began writing my Python to access this data and pull out the two values that I wanted to use in my application.

Initially, I was unable to parse the JSON correctly with the NWS API. After struggling for hours, I discovered that the data being returned is not JSON, but instead it is GeoJSON data. I researched how to query this data and explored the use of the ‘geojson’ python module (geojson 3.1.0, n.d.). I was able to install the module into Anaconda, but after testing the module it would not return the data in a format that I could use. Next I found the ‘geopandas’ module and found a way to convert the JSON into a data frames with the following code sample (Taras, 2023).

A screenshot of a computer

Description automatically generated

This provided me with my data in a Python list format. Next, I needed to parse the list for the list of values for the ‘textDescription’ and ‘temperature’. I found another code process using the ‘objectpath’ module. This required me to install that module into Anaconda and add the following code to my query (H., 2017).



With this code, I was able to parse the ‘textDescription’ as a string value, but I still needed to isolate the ‘temperature’ value from the tuple string. I used the python slice function to truncate the front and the back of the string to isolate the temperature value. Then used the python float function to convert this back into a number that would allow mathematical operations. Specifically, I wanted to convert the Celsius value into a Fahrenheit value. At the end of this process, I was left with two values that I wanted to pass on to my Custom API, the Current Conditions and Current Temperature in Fahrenheit.

Next I moved onto my plan to convert foreign currency into US Dollars. For this example, I decided to use the Euro for a simulation of a person that will be traveling to Los Angeles, USA and wanted to know how much they would receive for $1000 Euro’s . After reviewing the list of Currency API’s I decided to try the ExchangeRate-API.com API.



I was able to retrieve the JSON output and it appears to be in a relatively flat format. Next, I will build my query to isolate one of the exchange rates for my application. To accomplish this, I first flattened the data frame to a single nested dictionary entry and then queried that dictionary entry for the ‘EUR’ value as follows.

A black text on a white background

Description automatically generated

Once I had the Euro Exchange rate I was able to calculate a hypotectical scenario where the person visting Los Angeles has 1000 Euros and wants to know how much they would have in US Dollars. I also needed to modify the result of this calculation to round to the nearest secound decimal. I am not sure that this is techincally correct, but I went with it since it worked. After calculating this in my sample the visitor was left with $940.38 and I had one value, USD, to pass on to my Custom API.

For my last API, I wanted to offer tourist a night life option so I am using the Open Brewery DB API located at [Open Brewery DB | Documentation](https://www.openbrewerydb.org/documentation/#search-breweries). This open database allows for queries for over 8250 breweries located in cities around the world. The API allows queries for a variety of location data including City Name and Zip Code. For the API query I ran a simple query that pulled all of the results for the city name of ‘las angeles’. This data contains 25 breweries in Las Angeles. In the data, some of the results are breweries only and I wanted Breweries that also served food. Based on the API data, the most relevant brewery type would be ‘brewpub’.



Next I was able to filter the 25, Los Angelese based breweries, based on the data type for “brewpub”, which returned five entries. I also want to filter out businesses that do not have a website URL and phone number. There are two that meet both criteria from the data, but I am having a hard time filtering out the URL and phone numbers. The issue that I am having is how to eliminate “null” values using python. During my research I found the DROPNA option, which appears to work well (Saturn Cloud, 2023).

A close-up of a website

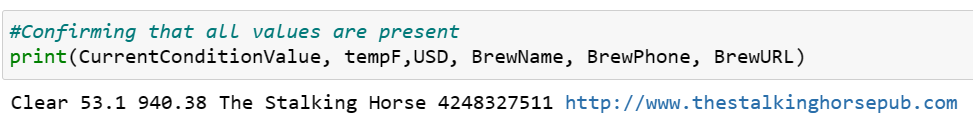
Description automatically generated

At this point I had two entries left. To keep my custom API return simple I wanted to only deal with a single entry. To accomplish this, I looked for a way to randomly pick a single entry from a list of data. The most straightforward method appears to be with the “DataFrame.sample” method (Collins, 2024). One I applied this method, I was left with a single set of keys that I was able to extract to be used in my Custom API.

A math equation with red text

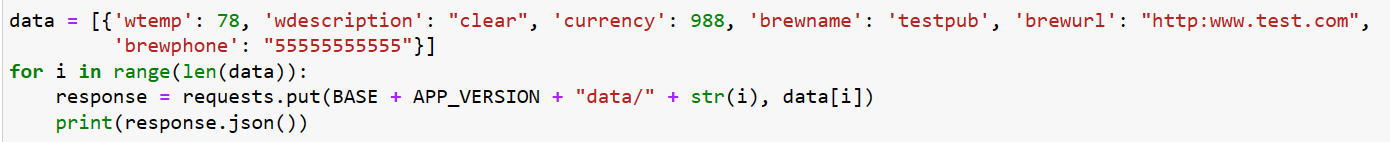
Description automatically generated with medium confidence

With the three public API defined and tested I next moved to the creation of my driver application. The driver application will consolidate the pull of the 3 public API’s, transform part of the data, and push the data to the custom API. First I defined the new “[API Driver App v1.ipynb](http://localhost:8888/notebooks/RestFull/API%20Driver%20App%20v1.ipynb)” python file and began the work of moving the three previous api python files into a single doc. The required cleaning up duplicate data entries between the three files. Once this was completed, I was left with five values that I wanted to pass onto my Custom API via a push, which was my next task.

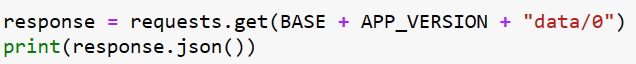


Before I could add the PUSH API code to my Driver Application, I needed to modify my new custom api code to accept the new data types that I would be sending to its database. I used the class example, main.py, to start this process and began by modifying the database fields to hold my new value names and data types. Once this was completed, I tested the code, which executed and created a new database file.

To confirm that the database was created correctly, I researched how to read an SQLITE database file. It appears that the best free tool is the DB Browser for SQLite located at sqlitebrowser.org (sqlitebrowser.org, n.d.). With this tool I confirmed that the database was created correctly and had the correct fields. Next, I wanted to test my PUT code, so I used the class example again. In the test-put.py, I hardcoded data into the data value, which successfully added the data to the API and the database.



Once I confirmed that the data was located in the database correctly, I next tested my GET code, which also worked correctly to the single data entry.



Since the expectation of my Custom API is that the data will always be current, there will never be more than a single data entry so going forward my Driver Application will only use the API Update of the “Index 0” item. Now that the Custom API appears to be working correctly, I move to add the correct API commands to the Driver Application. I was able to add the API Update commands and tie them into the existing variables. Update worked successfully, but I am getting an error return. It appears that the code is working correctly on the Update, but the error handling it broken and incorrectly reporting an error. The API Get code also worked correctly.

Finalized the API document that I have been working on and started working on reviewing, formatting, and completing my Narrative Document.