RESTful API Project

Project Narrative Document

Course: INFO U762 – 01F

Semester: Spring 2024

Instructor: Dr. Grover Walters

Date: April 22, 2024

David C. Miller

**Custom Travel REST API Application to Support City Tourism Websites**

David Miller1

1University of South Carolina - Upstate, 800 University Way, Spartanburg, SC 29303

**Abstract**

Most cities maintain a city tourism website that promotes different aspects of that city to perspective tourist. These pieces of data may be unique to that location but many of these data points are common for all cities, such as weather or dining options. This includes data points which can be collected from Public REST APIs. The use of Public REST APIs allows cities to collect and use the data that has already been provided by a trusted party. This paper describes a Custom Travel REST API Application that collects three types of data, stores them for retrieval, and then provides them on-demand to a website. The sample location that I chose for this demonstration is Los Angeles, California, but this code can be reused for any city’s website.

**Keywords**

RESTful Web APIs, Tourism Website

1. **Introduction**

This paper describes a Custom Travel REST API Application that collects three types of data, stores them for retrieval, and then provides them on-demand to a hypothetical website. I started the project by looking at the class provided public API list. After reviewing this list, I knew that to keep things simple I did not want to use API Authentication or REST API KEYs. Based on this, I copied the list to an Excel spreadsheet and identified the ~700 Public APIs that did not need Authentication. Next, I started narrowing this list down and looking for three that would work together to tell my use case 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. I was able to reuse many of the class examples for my Python code to build my API Driver Application and my Custom API. The hardest part of this project was extracting the values that I needed out of the massive amount of data retrieved from the three public API’s.

1. **Related Work**

To begin working on the Driver Application and Custom API I needed to confirm the installation of several tools on my computer. First, I needed to confirm that Anaconda Navigator was still working. Anaconda Navigator allows me to run the Jupyter Notebook environment that allows me to test my code. Secondly, I needed to confirm that the Anaconda Environment software was installed correctly. This allows me to test my code in Visual Studio Code. Third, I needed to confirm that Visual Studio Code was installed correctly and setup to run Python in the Anaconda Environment. Lastly, I needed to install a series of Python Modules to be Imported into my code to allow me to parse and process the API data correctly. Once I was satisfied that the tools were installed on my computer correctly, I began working on my code.

1. **Benchmark Requirements and Tasks**

Once I had decided on the types of APIs that I wanted to use, my next task was to dig into the specific APIs for each type of data that I wanted to 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 <https://www.weather.gov/documentation/services-web-api> API site had extensive documentation on how to pull data based on various input values. Based on my use case of providing Weather related data for Los Angeles, I chose the input value of Station ID. Another part of the website provided me with a map of weather station IDs, and I confirmed that Los Angeles was KCQT.



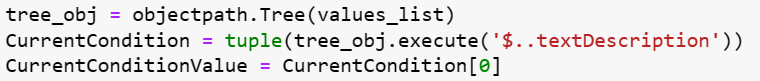
I moved this information to my Weather REST API Jupyter Notebook and confirmed that I would be able to pull the data correctly. Out of the data acquired, I decided that the two values of particular interest would be the “current weather”, which appears in the ‘textDescription’ value and “current temperature” 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 Fahrenheit values to visitors. With this information in hand, I began writing my Python code 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 from 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 extract 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 frame 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 values for the ‘textDescription’ and ‘temperature’ fields. I found another code sample 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 a 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 from a country in Europe and wanted to know how much they would receive for 1000 Euro’s. After reviewing the list of Currency API’s, I decided to use the <https://www.exchangerate-api.com/> API.



I was able to retrieve the JSON output that 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 hypothetical scenario where the person visiting 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 second decimal place. I am not sure that this is technically 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 <https://www.openbrewerydb.org/documentation/#search-breweries>. This open database allows for queries of over 8250 breweries located in cities around the world. This 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 tough time filtering out the blank URLs 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

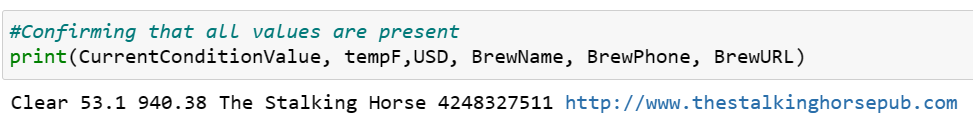
Description automatically generated

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). Once 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

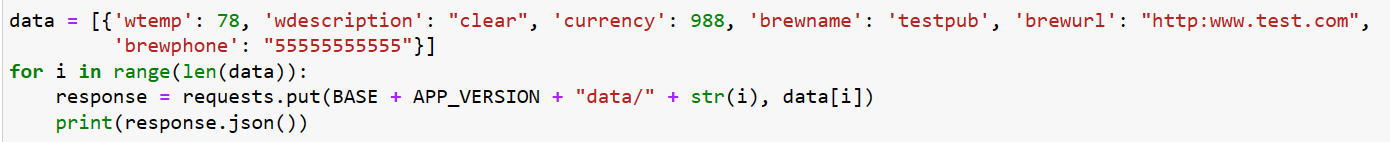
Description automatically generated with medium confidence

With the three public APIs 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 the new Driver App. This required cleaning up duplicate code between the three files. Once this was completed, I was left with the five values that I wanted to pass onto my Custom API via a REST PUT or UPDATE, which was my next task.

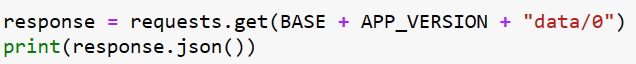


Before I could add the PUT API code to my Driver Application, I needed to modify my new custom API code to accept the new data types that I would be sending to its database. I used the class example, main.py, to start this process and began by modifying the database fields to hold my new value names 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 <https://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 stored in the database correctly, I next tested my GET code, which also worked correctly to pull from 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. The Update worked successfully, but I am getting an error return statement. It appears that the code is working correctly on the Update, but the error handling is broke and incorrectly reporting an error. The API Get code also worked correctly. I will leave this non-fatal error for v2 of my Custom API.

1. **Usage Example**

In this section we review how the Custom Travel REST API Application would be used in a real-world example. To be used more generically the custom API would need to be modified to accept a series of inputs from a city’s website. One input could be a simple zip code. The custom API would then take this input value to determine the correct city to pull weather from. The custom API would also use the zip code to find a Brew Pub to recommend to visitors. Lastly the website could ask the visitor if they are traveling from another country and use that input into the custom API to show the visitor how much 100 of a local currency would be in US Dollars. With these modifications the Custom Travel REST API Application could be used by city’s anywhere in the United States.

1. **Conclusion**

By using Python and publicly available APIs I was able to demonstrate the ability to create a Custom Travel REST API Application. This application would allow cities to add tourist information to their existing city webpages without the need to do extensive custom coding. The API also allows for modifications that would allow cities to add this API code to their websites with only the use of a Zip Code. One outstanding question would be where this code would be hosted so that cities could access it and if this code could ultimately be monetized.

# References

*API Web Service*. (n.d.). Retrieved March 25, 2024, from National Weather Service: https://www.weather.gov/documentation/services-web-api#/default/station\_observation\_latest

Collins, E. R. (2024, March 7). *blogfinxter.com*. Retrieved from 5 Best Ways to Select a Random Row from a DataFrame in Python: https://blog.finxter.com/5-best-ways-to-select-a-random-row-from-a-dataframe-in-python/#:~:text=One%20of%20the%20most%20straightforward%20ways%20to%20select,row%20by%20setting%20the%20n%20parameter%20to%201.

*geojson 3.1.0*. (n.d.). Retrieved April 6, 2024, from pypi.org: https://pypi.org/project/geojson/

H., S. (2017, January 5). *Parsing json and searching through it*. Retrieved from stackoverflow.com: https://stackoverflow.com/questions/8383136/parsing-json-and-searching-through-it

Saturn Cloud. (2023, June 19). *saturncloud.io*. Retrieved from How to filter out records with null or empty strings in python pandas: https://saturncloud.io/blog/how-to-filter-out-records-with-null-or-empty-strings-in-python-pandas/#:~:text=To%20filter%20out%20records%20with%20null%20or%20empty,columns%29%20and%20the%20subset%20of%20columns%20to%20consider.

*sqlitebrowser.org*. (n.d.). Retrieved April 13, 2024, from DB Browser for SQLite: https://sqlitebrowser.org/

Taras. (2023, August 24). *Reading single feature as dict into GeoDataFrame with GeoPandas*. Retrieved from gis.stackexchange.com: https://gis.stackexchange.com/questions/465741/reading-single-feature-as-dict-into-geodataframe-with-geopandas