Weather Coding Application

Abstract: A weather API is a service that allows developers access to weather data and integrate it into their applications and websites. Weather API’s provide real-time weather data forecasts, historical data and or a combination of information. This paper explores the development of an innovative weather application using the flask web framework, with the focus on strategic server environment configuration and the application of Object-Relational Mapping (ORM) system for python. Flask is a lightweight and flexible web framework for Python that allows users to construct an application. Flask-SQLAlchemy is a flask extension intended for ORM in Python. Choosing the right tools can significantly impact the efficiency and success of projects in web development.

1 Introduction:

Libraries provide pre-designed functionality that developers can reuse, compile, and test code for automation or for the purpose of expediting and augmenting an application. Libraries are also meant to enforce strong encapsulation of dependencies. (“Library Basics, n.d) This also saves time and energy by avoiding reinventing the wheel. For this project, flask dependencies coming together to build a robust custom API is important because it provides the foundation for the web application. Flask-restful simplifies building restful API’s by providing tools for request. Here is an example of an import dependency.

**from** **flask** **import** Flask

**from** **flask\_restful** **import** Resource, Api

app = Flask(\_\_name\_\_)

api = Api(app)

**class** **HelloWorld**(Resource):

**def** get(self):

**return** {'hello': 'world'}

api.add\_resource(HelloWorld, '/')

**if** \_\_name\_\_ == '\_\_main\_\_':

app.run(debug=**True**)

Flask is often referred to as the friendly web framework. In order to use flask, the system being used requires Python installation. Next, requires opening prompt command, to determine if a Python version is visible and ready to use. Then, installing Flask is usually done by the same process using; (**pip install flask**) but in the terminal of Visual Studio Code. These Python packages are installed next using ‘pip’ manager and running Python scripts.

**# to download a single package**

**pip3 install flask**

**# or to download all the packages from requirements.txt**

**pip3 install -r requirements.txt # (This should install all the modules in Python3)# you can run settings.py in python 3python3 settings/settings.py**

This process sounds simple, but it was not so easy when trying to install flask. Repeated attempts in trying to run the code can be frustrating if syntax errors are present. In this case, there are no syntax errors, yet the system showed implications. In Python programming there are three types of error that occur. If a syntax error occurs it means the interpreter has encountered code that does not conform to Python language rules. [[1]](#footnote-1)Many times, if the language and format are all done correctly, there isn’t an issue. For this design, the program application could not find the location of the files created, making it impossible to run the program. After further research, it was determined the application module wasn’t installed at all on the computer. However, this issue could never fully be resolved for the project, but some things were modified. Here is one of the repeated visual messages displayed in the terminal. (from flask import Flask ModuleNotFoundError:No module named 'flask’)

After days of trying to install flask in the environment to run the code, it was determined the Flask package wasn’t a part of the Python environment needed. The only option in solving this issue was to find a way to gain access directly to flask. The website itself contained detailed information on resolving the issue “ModuleNotFoundError No module named “flask” error in Python. There are several key points that aim to help users fix errors: Install Flask use “pip install flask,” Python Environment,” “Virtual Environment, (vitualenv),” and “IDE Settings.” The site recommended using a newer version of Python. (Pallets,2024). According to personal observation, Flask was manually installed via a .venvsetup which installed all the files for flask and its extensions, but never truly ran to full capacity. Some extensions such as Flask-SQLAlchemy worked, while others did not. One learning experience is that newer libraries for projects can break the compatibility in other projects that are not so similar. For this project, creating a weather app in Python came with many challenges, first trying to obtain 3 API keys from three different sites. The three sites used in creating the weather app for this project are OpenWeatherMap API, Weatherbit API, and AccuWeather API.

According to OpenWeatherMap, there are certain factors that influence the performance of their API’s, such as data accuracy, documentation quality, ease of use, customer support, cost effectiveness, feature sets, reliability and uptime are all pivotal for weather data sites aiming to offer their users a comprehensive weather service. (OpenWeatherMap, n.d.) For some, cost effectiveness may be detrimental to their project or business needs. For this project, to obtain a key requires just signing up with each official site with some offering a free 21-day account and a key is automatically generated in most cases. In this particular instance, it was an overstatement, because it nearly took 24 hours before one key was truly active. Once signed up though, the key will authenticate any requests to the API. According to the Weatherbit API documentation serves as a comprehensive tool for integrating data and the endpoints are crucial in accessing various types of data. Some of the main endpoints include: Current weather data, 5-day forecast, 16-day forecast, historical weather data, weather alerts, and weather maps (e.g., temperature, wind, precipitation). Each of these endpoints has its own URL, along with set parameters that can be used to customize a request. [[2]](#footnote-2) (Weatherbit. (n.d.)." Handling and receiving responses from the API are necessary for the functionality and security of the application. This ensures that communication is applied using external services and data is utilized to fulfill its intended purpose.

Handling is also critical for parsing API responses, oftentimes, comes in structured format such as, JSON or XML. Parsing allows data to be displayed accurately to users. In the provided code snippet for this design project for validating the request, parsing the request arguments permits users to define rules for each argument. In the weather input arguments parser in this situation defines arguments like “location,” “temperature,” and “humidity,” indicating that they are required and their data types. After defining the parser and its arguments, calling (parse\_args) extracts the arguments from the incoming HTTP request. What should happen for example, if a client sends a PUT request with JSON data involving location, temperature, and humidity ‘pars\_args ()’ will extract these values and make them available as ‘args . location,’ ‘args . temperature,’ args humidity respectfully.

( **Parser =reqparse.RequestsParser()** )

**parser.add\_argument('rate', type=int, help='Rate to charge for this resource')**

**args = parser. parse\_args()**

**Weather\_put\_args = reqparse.RequestParser()**

**Weather\_put\_args.add\_argument("location", type=str, help="Location of the weather data is required", required=True)**

**Weather\_put\_args.add\_argument("temperature", type=float, help="Temperature of the weather data is required", required=True)**

**Weather\_put\_args.add\_argument("humidity", type=float, help="Humidity of the weather data is required", required=True)**

Under the heading API documentation are details of how to construct API requests and the required parameters for the data types. Upon reading this information at the beginning of this project, things were a little murky as a newcomer not being familiar with structure, terminology, or specific concepts used in an API, can be overwhelming. Like many others, OpenWeatherMap’s API documentation is dense and filled with technical jargon, which initially is shocking when starting out.

Similarly, when exploring other weather API’s such as, Weatherbit API and Accuweather API, one encounters a similar challenge in navigating their respective documentation. Much like OpenWeatherMap, Weatherbit API and AccuWeather API provide detailed instructions on constructing API requests and specifying required parameters for various data types. Nevertheless, the learning curve for newcomers can still be steep due to the dense nature of documentation and the abundance of technical jargon. When comparing these APIs, one can assess factors such as the clarity of documentation, accessibility of information, and the level of support provided to users, which are decisive in determining the ease of integration and usability of the API in projects. After evaluating several API’s, together with OpenWeatherMap, Weatherbit API, Accuweather API, next is retrieving the API keys for each service to integrate them into the weather app design project. The decision to utilize these specific APIs are based on various factors, including the comprehensiveness of weather data, the availability of historical data, and the quality of documentation for support provided by each service. Collectively, acquiring the API keys was a straightforward process, requiring registration on each platform and generating unique access keys.

Weather is a constantly changing atmospheric process that is detected using a combination of both prediction and direct measurements. For farming: a foundational food source and having information on the weather is vital. Simultaneously, one’s decision on what to wear each day also is primarily a result of what one knows about the weather, so it is easy to take for granted how much one relies on weather data. Getting accurate weather forecasting nowadays is attributed to the advancement of weather apps. According to a *Time* article, due to the implementation of advanced weather prediction models led to a significant amelioration in the accuracy of our forecasts. [[3]](#footnote-3) (Henshall, 2023). This assertion is supported by the integration of vast meteorological models run on supercomputers by government agencies like the U.S. National Weather Service and the U.K. Meteorological Office (Henshall, 2023). The article further adds, providers mentioned here utilize data from government models and additional sources such as, proprietary weather stations to improve their forecasts. Furthermore, regarding accuracy of weather apps, the article points out the weather accuracy with features such as atmospheric circulation patterns are based on different regions. [[4]](#footnote-4) (Henshall, 2023). The technology used to depict the data utilizes APIs that coalesce the process of showing individual specific measurements of temperature, wind speed, and so forth. The output from this process are the websites, applications, and maps that all show the specific forms of data as well as how they progress over timespans.

2. **Background and Motivation**

In the following is a description of the terminology used in the paper, provided is a motivating example of the importance of “F-String, a powerful, and concise way to interpolate and format strings in Python.

2.1 **Terminology**

* **API’s**. API acronym for Application Programming Interface. An API call is any request sent to the API. Requests are generally sent in one of the two ways: First, manually using a web browser (such as navigating to the URL,) or
* secondly, programmatically sending the requests via executing code that sends the API call and process the response. (About the openFDA API, 2024)
* **Uniform Resources Locators(URLs**) URLs are addresses used to locate resources on the internet. They usually consist of several components including the protocol (http, https) domain name, path query parameters, etc.
* **REST.** The representational state transfer (REST) architecture is perhaps the most popular approach to building APIs. Rest relies on a client/server approach usually separating front and back ends of the API and provides plenty flexibility in development and implementation. Restful is a way of providing interoperability.

**Method definition**. F-string is also known as formatted literals that provide to the point and in a dynamic way to interpolate and format strings in Python. To create an f-string, simply prefix the string with the letter “F” or “f”. Users can position expressions within the curly brackets {} where they want to insert their values into the string. Remember that expressions can be variables, calculations, and function calls or any number of Python expressions. Python's F-strings provide a convenient way for string interpolation and formatting (Jablonski, 2023).[[5]](#footnote-5)

Motivating Example: For this part of the project the driver application is created: the elegance and practicality of f-string in the context of my weather application. I was able to move faster with f-string in Python versus (%) operator, and the (.format() method. According to an article in *Real Python*, the key takeaway is f-string has an (=specifier) to help with debugging a code by displaying the expression and its value result. [[6]](#footnote-6) (Real Python, n.d.) By leveraging this python feature, I seamlessly integrated my three API keys for the application driver to get accurate up-to-date weather data locally. F-strings can be very useful when dealing with sensitive information like APIs. By utilizing f-strings one can easily and securely integrate an API in a codebase without exposing them directly in the source code.[[7]](#footnote-7) Next is data storage and retrieval, a list of dictionaries (‘**data’**) is defined containing data using ‘PUT’ requests to the local server. This step was initially missing the rest of the code and later added. Then, another function is defined as ‘**get\_weather\_data ()**’ to retrieve weather data. This function enhances code readability and reusability. However, there is an interaction with the local server, using PUT and GET requests to store and retrieve data. The information displayed in the status terminal was the ‘country code:’ ‘US’ and the ‘region name’ South Carolina. The coordinates {‘coord’ : {‘lon’: -82.4653, ‘lat’: 34.6498}, the actual temperature and the heat index is what it feels like outside.

Whether you are a skilled developer or a Python enthusiast, f-strings are a valuable addition because Python expressions can be added directly inside the string literals. F-string is a feature introduced in Python 3.9. Using the f-string for this project resulted in readable, concise, and less error-prone string interpolation in the driver application. Ultimately, the goal was to create a restful API, by putting together a driver application into a custom API.

A screenshot of a computer program

Description automatically generated

Figure 1(a)

In fig. 1(a) shows the significance of accessing real-time weather data from various applications, for example, agriculture, tourism, and urban planning. Further, the method is Python F-strings used to dynamically construct an API request for retrieving weather data

from OpenWeatherMapAPI based on the user’s geographical location. First, the user’s IP address using ‘**requests.get** ‘function and the ‘**ipify’** API.[[8]](#footnote-8) Next, using the IP address is used to fetch location data (including latitude and longitude) from the **‘ip-api.com’** API. The third step is constructing the API request for OpenWeatherMap using and F-string, incorporating latitude, longitude, and the API key (**‘openweatherapi’**)**.** Soon after making the API requests, weather data is extracted from the JSON response focusing on the temperature. Finally, we print the current temperature along with the (‘city name’).

A screenshot of a computer program

Description automatically generated

figure 1(b)

In fig. 1(b) is a demonstration of how to integrate the WeatherBit API to retrieve weather data, offering an alternative to OpenweatherMap. The script here follows a similar approach to figure 1(a) but instead of using OpenweatherMap API, it integrates the WeatherBit API. The API requests URL is constructed using an F-string, incorporating latitude, longitude, and the WeatherBit API key (‘**weatherbitapi**’). After retrieving the weather data, the temperature is extracted from the JSON response. In conclusion, the current temperature is printed only this time in Celsius, followed by the (‘city name’).

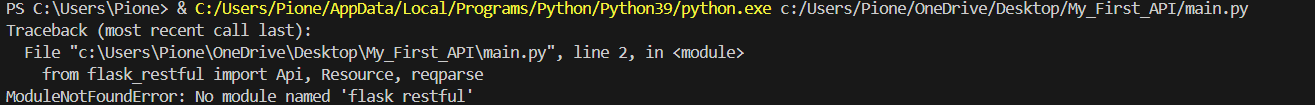
These precise demonstrations show how Python scripts can effectively integrate different APIs (OpenWeatherMap and WeatherBit) to retrieve real-time weather data based on the user's IP geolocation, showcasing the power and flexibility of API integration in Python programming.

**Proposed Approach**:

To recap the development process, Python was chosen as the programming language to write the code needed for the application. Across the software, ultimately, two primary files were created: the first titled main.py (Custom API) and the other project-weather.py (Driver Application). The custom API is responsible for handling specific functions related to the restful API application it interacts with, while its design functions, by making request to restful weather APIs. It performs tasks of handling any necessary authentication of API keys. Next, four other files were created to help make flask and its extensions properly run the code; those files being: Get.py, \_\_init\_\_.py, reqparse.py, and inputs.py. In class, a template was given on how to make main.py, and was accommodated to run the weather data. However, when the changes were made, they came with much difficulty, it was a challenge to determine which components of data would go into each of the arguments.

In the end, three components were part of the primary class function to create the server, which were location, temperature, and humidity. These served as backbones behind the arguments, the additional commands, and filtering. As per part of the main.py (Custom API), reqparse was called in parts of the code. The other major data called was located in the project-weather.py (Driver Application), which contained the data from the weather APIs themselves. The data was directly requested using direct coordinates of latitude, longitude, and city. Information was called from requests via f-strings and was used in tandem with the URL sources of the original websites where the APIs were found. The API keys played a vital role in calling the information due to the constantly changing nature of weather, meaning the data would need to be updated periodically. It was here where the Get.py came into play. Get.py was used for the update process, while also determining the app version of the WeatherbitAPI.

To close out, the project-weather.py ran without an incident, and was easily able to generate clear, accurate, and real time weather data. This resulting output data was put into a csv file, which was made available alongside the other files of this project. However, it failed to run in the base URL server <http://127.0.0.1:5000/>. A test message was done via another file, providing a hello message associated with it, but aside from this isolated test attempt, nothing else worked. Meanwhile, the main.py failed to work despite numerous attempts to make this code in the file work. The most common errors were with flask and its extensions. Figure 2 shows an example of the message seen during the failed attempts.



(Figure 2\*)

As stated earlier, this error was constantly present in the terminal, each time the code was run. Upon further research, it was found the reason for this error was due to the dependencies at the beginning of the code and the system failed to find the flask module in the same file location as main.py. A similar error occurred on line one for flask itself, and after the aforementioned manual installation was completed. This error was corrected and ran, only for the extension to fail. Many files for flask and its extensions were removed from the venv file to share direct space with the API python files. However, the error for the flask’s restful extension still remained.

According to Mike McGrath, “Whenever a user asks to view an online web page in their browser it requests the page from the web server, and receives the page in response, via the HTTP protocol. […] Python […] host[ed] the […] server [from scripts it recognized], […] call[ing] upon the Python interpreter to process script code before sending an HTML response to the web…” That setup is shown in figure three below.[[9]](#footnote-9)

As mentioned earlier, this book simplifies Python programmers for beginners: by understanding these concepts, user requests, web server, and HTTP protocols readers can gain insight into the inner workings of web servers and the role of Python. The figure three is possibly a depiction for visual learners to understand complex concepts.

Conclusion:

In conclusion, developing code for a weather API using Python involves navigating through various complexities and challenges. From understanding the intricacies of APIs with their endpoints, parameters, and response formats to obtaining and securely managing API keys, each step requires careful consideration and attention to detail. The availability and accessibility of API keys vary across different platforms, adding an additional layer of complexity to the process. Moreover, users must be mindful of rate limits to avoid disruptions to their service and adhere to API documentation guidelines. Despite these challenges, mastering the development of weather APIs can be immensely rewarding, offering opportunities to create innovative applications and solutions. By staying informed about best practices, leveraging available resources such as API documentation, and continually refining their coding skills, developers can overcome obstacles and unlock the full potential of weather data integration in their projects. In the ever-evolving landscape of software development, persistence, adaptability, and a commitment to continuous learning are essential traits for success. As technology continues to advance, so too will the capabilities and possibilities of weather API development with Python. By embracing these principles and embracing the journey of exploration and discovery, developers can chart a course towards achieving their goals and making meaningful contributions to the field.

References

AccuWeather. (n.d.). AccuWeather API. Retrieved April 5, 2024, from <https://developer.accuweather.com/apis>

Adams, A. (2024). Weather Project. Unpublished manuscript.

Henshall, W. (2023, July 3). Most Accurate Weather Forecast Apps of 2023. Time Magazine. https://time.com/6291479/most-accurate-weather-forecast-apps-2023/

Hjelle, G. A. (2020, January 20). Python f-strings: A complete guide. Retrieved from <https://realpython.com/python-f-strings/#key-takeaways>

Jablonski, J. (2023, October 18). Python's F-String for String Interpolation and Formatting. Real Python. Retrieved from https://realpython.com/f-strings-python/

McGrath, M. (2023*). Python in Easy Steps. In Easy Steps Ltd.*

Open Weather. (n.d.). Weather API. Retrieved April 5, 2024, from https://openweathermap.org/api

Shulka, S. (2024, January 2). Efficient String Formatting with Python f-Strings. Dzone. https://dzone.com/articles/efficient-string-formatting-with-python-f-strings

The Pallets Projects. (n.d.). Installing Flask. Retrieved April 12, 2024, from https://flask.palletsprojects.com/en/3.0.x/installation/#install-flask

U.S. Food & Drug Administration. (2024, March 19). About the openFDA API. Retrieved April 7, 2024, from https://open.fda.gov/apis/

Weatherbit. (n.d.). Weatherbit API Documentation. Retrieved April 5, 2024, from https://www.weatherbit.io/api

1. As stated by McGrath, M. (2023*). Python in Easy Steps. In Easy Steps Ltd*.

   Syntax Error occurs when the interpreter encounters code the does not conform to the Python language rules. For example, a missing quote mark around a string. The interpreter halts and reports the error without executing the program. [↑](#footnote-ref-1)
2. "The Weatherbit API offers a range of endpoints, including current weather data, forecasts, historical weather information, and weather maps, each providing access to comprehensive and up-to-date meteorological data for locations worldwide. [↑](#footnote-ref-2)
3. The article highlights the improvements in weather forecasting accuracy over the past decades, attributing it to advancements in meteorological models and the utilization of new sources of data. David Novak, Director of the Weather Prediction Center at the U.S National Weather Service, asserts that there has been a steady improvement in forecasting skill, with forecasts now being as accurate for five days ahead as they were for four days ahead in 2013 (Henshall, 2023). [↑](#footnote-ref-3)
4. Regarding the accuracy of weather apps, the article notes that geographic features and atmospheric circulation patterns can affect forecast accuracy in different regions. Floehr highlights those areas with more consistent weather, like California and Arizona, tend to have more accurate forecasts compared to regions with more complex weather patterns (Henshall, 2023). [↑](#footnote-ref-4)
5. Jablonski, J. (2023, October 18). Python's F-String for String Interpolation and Formatting. Real Python. https://realpython.com/f-strings-python/ [↑](#footnote-ref-5)
6. Real Python. (n.d.). Python f-strings: A Complete Guide. Retrieved from <https://realpython.com/python-f-strings/#key-takeaways> [↑](#footnote-ref-6)
7. Jablonski, J. (2023, October 18). Python's F-String for String Interpolation and Formatting. Real Python. https://realpython.com/f-strings-python/ [↑](#footnote-ref-7)
8. Adams, A. (2024). Weather Project. Unpublished manuscript. [↑](#footnote-ref-8)
9. See McGrath, M. (2023*). Python in Easy Steps. In Easy Steps Ltd*. [↑](#footnote-ref-9)