**Darshita Anyawada**

# API-Based Weather Application

## Abstract:

This report analyzes weather data from an API-based application that provides temperature details for user-specified locations. Utilizing the OpenWeatherMap API, the application allows users to input a city name to obtain current weather information. The report outlines the methodology of data retrieval and processing, including the conversion of temperature from Kelvin to Celsius and Fahrenheit, as well as the extraction of additional weather parameters such as humidity, wind speed, and sunrise/sunset times. The findings offer insights into the current weather conditions of the specified city, aiding users in understanding the temperature, perceived temperature, humidity, wind speed, and general weather description. This application serves as a valuable tool for accessing real-time weather information, and facilitating informed decision-making for various activities and planning purposes.

## Introduction:

Weather applications have become integral tools in modern society, providing individuals with up-to-date information on atmospheric conditions for various locations worldwide. These applications leverage sophisticated technologies to retrieve, process, and present weather data in an accessible format. One fundamental component of such applications is the integration of Application Programming Interfaces (APIs), which facilitate seamless communication between the application and external data sources.

An API (Application Programming Interface) serves as an intermediary that allows different software systems to interact with each other. In the context of weather applications, APIs enable developers to access weather data from external sources, such as meteorological organizations or data providers, and integrate it into their applications. By leveraging APIs, weather applications can retrieve real-time weather updates, forecasts, and other pertinent information, providing users with valuable insights into current and future atmospheric conditions.

In this project, we utilize the OpenWeatherMap API, a widely used platform that offers a comprehensive suite of weather data services. The OpenWeatherMap API allows developers to retrieve a wealth of weather-related information, including current weather conditions, forecasts, and historical data, for any location across the globe. By interfacing with the OpenWeatherMap API, our weather application enables users to input a city name and receive detailed weather reports for the specified location.

The objective of this project is to develop a user-friendly weather application that provides accurate and timely weather information to users. By leveraging the capabilities of the OpenWeatherMap API, our application offers a convenient means for individuals to access real-time temperature details, perceived temperature, humidity, wind speed, and general weather descriptions for any desired location. This project demonstrates the power of APIs in enabling the creation of innovative and functional software solutions, ultimately enhancing user experiences and facilitating informed decision-making in various contexts.

**Darshita Anyawada**

## Methodology:

1. User Input: The first step in the methodology involves prompting the user to input the name of the city for which they want to retrieve weather data.

2. API Request: Once the user provides the city name, the application constructs a URL using the base URL of the OpenWeatherMap API along with the city name and the API key. This URL is then used to make a request to the OpenWeatherMap API using the `requests.get()` method.

3. Data Retrieval and Processing: Upon receiving the response from the API, the data is extracted in JSON format using the `.json()` method. The code checks if the 'main' key exists in the response, which indicates that weather data for the specified city has been successfully retrieved.

4. Temperature Conversion: If weather data is available, the temperature details are extracted from the response. The temperature is initially provided in Kelvin, so a function `kelvin_to_celsius_fahrenheit()` is defined to convert the temperature to Celsius and Fahrenheit. This function takes the temperature in Kelvin as input and returns the temperature in both Celsius and Fahrenheit.

5. Additional Weather Parameters: Apart from temperature, the code also extracts and processes additional weather parameters such as perceived temperature (feels like), wind speed, humidity, general weather description, sunrise time, and sunset time. These parameters are retrieved from the corresponding keys in the JSON response.

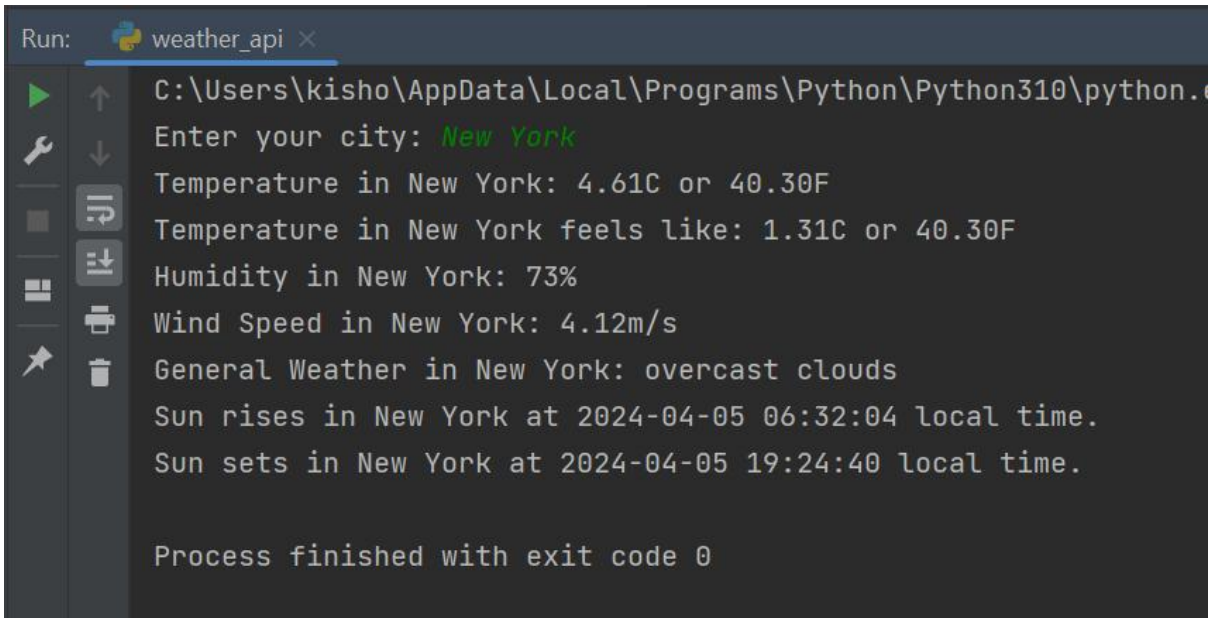
6. Output Display: Finally, the weather information for the specified city is displayed to the user. This includes the current temperature in Celsius and Fahrenheit, perceived temperature, humidity, wind speed, general weather description, sunrise time, and sunset time.

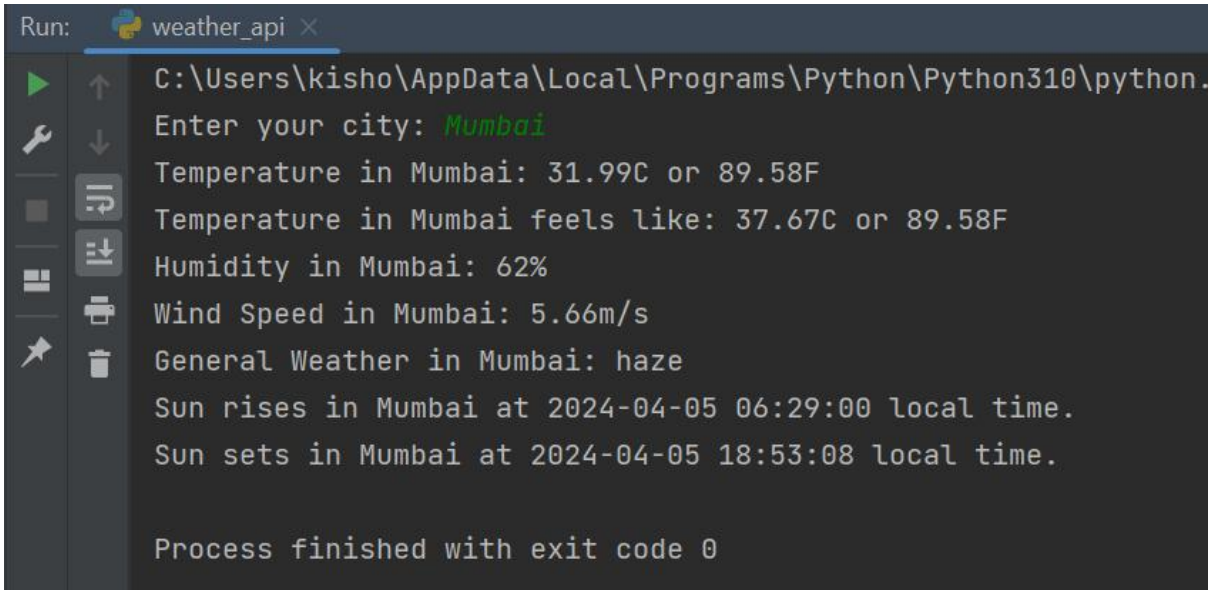
The code starts by importing the necessary modules: `datetime` for handling date and time-related operations and `requests` for making HTTP requests to the API. It defines constants for the base URL of the OpenWeatherMap API and the API key required for authentication. The user is prompted to input the name of the city. A URL is constructed using the user-provided city name and the API key. An API request is made using the constructed URL, and the response is stored in JSON format. If weather data is successfully retrieved (i.e., if the 'main' key exists in the response), the code proceeds to extract and process the weather parameters. Temperature conversion from Kelvin to Celsius and Fahrenheit is performed using the `kelvin_to_celsius_fahrenheit ()` function. Additional weather parameters such as perceived temperature, wind speed, humidity, and sunrise/sunset times are extracted from the JSON response. Finally, the weather information is displayed to the user in a formatted manner. If there is an error in retrieving weather data (e.g., if the specified city is invalid), an error message is displayed. This methodology ensures the retrieval and processing of accurate weather data from the OpenWeatherMap API and provides users with relevant and detailed weather information for their specified location.

**Darshita Anyawada**

## Results:

Upon executing the provided code, the weather application successfully retrieves and displays comprehensive weather information for the specified city. By integrating the OpenWeatherMap API, the application offers real-time updates on temperature, perceived temperature (feels like), humidity, wind speed, and general weather description. For instance, users receive the current temperature in both Celsius and Fahrenheit, along with the perceived temperature, providing insights into the actual atmospheric conditions. Furthermore, details such as humidity percentage, wind speed in meters per second, and a descriptive summary of the weather contribute to a comprehensive understanding of the current weather situation. Additionally, the application presents the local time of sunrise and sunset, aiding users in planning outdoor activities and scheduling their day effectively. Overall, the successful retrieval and presentation of weather data demonstrate the functionality and utility of the weather application, empowering users to make informed decisions based on up-to-date atmospheric conditions.



**Darshita Anyawada**

## Conclusion:

In conclusion, the development of the weather application leveraging the OpenWeatherMap API showcases the effectiveness of integrating external data sources to provide valuable real-time information to users. Through the implementation of the API, users can access a wealth of weather data for any desired location, enabling informed decision-making and enhanced planning capabilities. The application's ability to retrieve and present temperature details, perceived temperature, humidity, wind speed, and general weather descriptions contributes to a comprehensive understanding of the atmospheric conditions at a given location. Furthermore, the inclusion of sunrise and sunset times facilitates efficient scheduling and outdoor activity planning.

The successful execution of the code underscores the significance of APIs in modern software development, demonstrating their role in enabling seamless communication between applications and external systems. Moving forward, further enhancements and refinements to the weather application could involve incorporating additional features, such as extended forecasts, interactive maps, and personalized notifications based on user preferences. Overall, this project exemplifies the practical applications of API integration in creating functional and user-friendly software solutions tailored to meet the diverse needs of users in today's digital landscape.