****

**School of Mechanical & Manufacturing Engineering, SMME**

**National University of Science and Technology**

**NUST H-12**

**“Weather Application Project”**

Name: Syed Amna Hasnain

CMS ID: 467234

Course Title: Fundamentals of Programming

Course Code: CS-223

Program: BE Aerospace

Section: AE-01

Semester: 3rd

**Project Overview:**

This project is a **Python-based Weather App** built using the **PyQt5** library for the GUI and the **OpenWeatherMap API** for retrieving real-time weather data. The application allows users to input the name of a city and fetches the current weather conditions, including temperature, an emoji representation of the weather, and a description. It is user-friendly, visually appealing, and handles errors gracefully.

**Objectives:**

1. Develop a functional weather app with an interactive GUI.
2. Retrieve and display weather information using an API.
3. Implement error handling for various network and input-related issues.
4. Utilize visual elements like emojis and formatting for better user experience.

**Features:**

1. **City Input**:  
   Users can enter a city name in the input field.
2. **Weather Display**:  
   Displays the following information after fetching data:

* Temperature in Fahrenheit.
* Weather description (e.g., clear sky, light rain).
* Emoji representing the current weather condition.

1. **Error Handling**:  
   Provides user-friendly error messages for various issues like:

* Invalid city name.
* Invalid or expired API key.
* Connection errors or server issues.

1. **Stylized GUI**:

* Modern font and formatting using stylesheets.
* Well-aligned and centrally positioned labels and input boxes.

**How to Run:**

**1. Prerequisites:**

* Make sure you have **Python 3.x** installed on your system. If you don't have it, you can download it from the official [Python website](https://www.python.org/downloads/).
* Install the required Python libraries. Open a terminal (or command prompt) and run the following command to install the dependencies:



**2. API Key:**

* Before running the application, you'll need a valid **OpenWeatherMap API key**. If you don't have one, you can sign up for a free account at OpenWeatherMap and get your API key.
* Replace the api\_key variable in the code with your **actual API key.**

**Code Explanation:**

1. **Imports:**

### 

* **Purpose:**

These libraries and modules enable us to design the GUI, make HTTP requests, and handle system-level operations.

* **Explanation:**
* sys: Manages the application's control flow and helps terminate the program cleanly.
* requests: Facilitates communication with the OpenWeatherMap API to fetch weather data.
* PyQt5.QtWidgets: Provides tools to build the GUI components, such as buttons, labels, and text boxes.
* PyQt5.QtCore: Offers alignment features to organize the layout visually.

1. **Class Definition:**

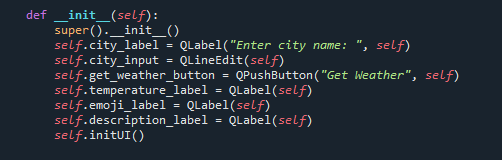
### 

* **Purpose:**

Encapsulates the application logic and interface within a reusable class structure.

* **Explanation:**
* QWidget: Serves as the parent class for the application window. It ensures access to all GUI-related features.
* By inheriting from QWidget, WeatherApp gains all of the functionality of a QWidget, which allows it to act as the main window for the application.

1. **Constructor Initialization:**

****

* **Purpose:**

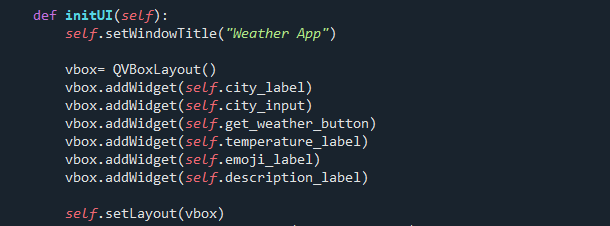
Initializes the application with essential components like input fields, buttons, and labels for displaying results.

* **Explanation:**
* **Calling super().\_\_init\_\_():**

It ensures that all the necessary initialization for the parent class (like setting up the GUI window) is done properly.

* **QLabel**: Used to display static or dynamic text, like instructions or output.
* **QLineEdit**: A text input field where users type the city name.
* **QPushButton**: A button that triggers the weather-fetching logic.
* **initUI()**: A method to set up the graphical interface layout and design. In order to arrange and style these components.
* **Self**: The second argument (self) specifies that the label is a child of the WeatherApp window. This makes the label part of the overall window and allows it to be displayed when the window is shown.

1. **User Interface Initialization:**

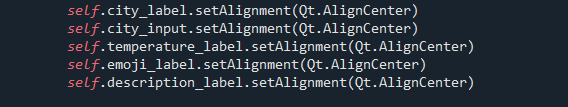
****

* **Purpose:**

Arranges GUI components in a vertical order for a clean and user-friendly layout.

* **Explanation:**
* **setWindowTitle("Weather App")**: Sets the title of the application window. After calling setWindowTitle(), you will typically continue by adding widgets (like buttons, labels, etc.), arranging them, and setting other properties like window size, styles, and so on.
* **QVBoxLayout**: A layout manager that arranges components vertically.
* **addWidget()**: Adds individual components (e.g., labels, buttons) to the layout.
* **setLayout()**: Finalizes the layout for the main window.
* **The initUI() method** is used to define the layout and other visual properties of the window.

1. **Widget Alignment:**

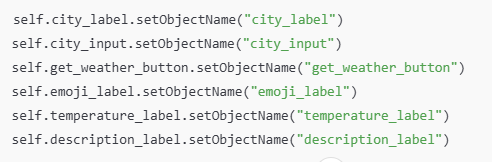
****

* **Purpose:**

Ensures all widgets are horizontally centered within the window for a professional appearance.

* **Explanation:**
* Qt.AlignCenter: A PyQt5 property that centers a widget horizontally.

1. **Object Names:**



* **Purpose:**

It is used to assign object names to various widgets in the PyQt5 application.

* **Explanation:**

Object names are used to reference and identify individual widgets for styling, signals, or testing purposes.

1. **Widget Styling:**

****

* **Purpose:**

Enhances the visual appeal of the application by applying consistent styles to all widgets .

* **Explanation:**
* The stylesheet uses CSS-like syntax to define properties like font size, color, and padding for individual widgets.
* Consistency: The stylesheet ensures a consistent and professional appearance across all widgets.
* Visual Hierarchy: Larger font sizes for key elements like the temperature and emoji draw the user's attention to the most critical information.
* User Experience: The clear and intuitive styling makes the app more user-friendly and visually appealing.

1. **Button Signal Connection:**

****

* **Purpose:**

Links the "Get Weather" button to the get\_weather function.

* **Explanation:**
* **clicked.connect():** Binds a button click event to a specific function, allowing user interaction to trigger functionality**.**
* **self.get\_weather\_button:** This is the QPushButton widget defined earlier in the code. It represents the "Get Weather" button that users interact with.
* **clicked:**

A signal emitted by the button whenever it is clicked.

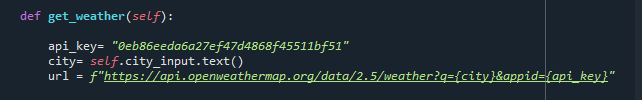
In PyQt, signals are events that widgets emit to notify about changes or user actions.

* **connect(self.get\_weather):**

This connects the clicked signal to the get\_weather method, establishing a link between the action (button click) and the response (calling the method).

* The **get\_weather method** is responsible for fetching weather data from the OpenWeatherMap API based on the city name entered by the user.

1. **Fetching Weather Data:**

****

* **Purpose:**

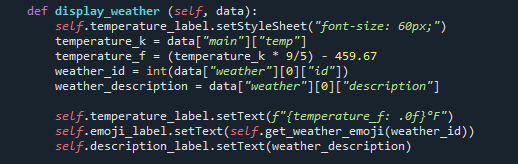
Communicates with the OpenWeatherMap API to retrieve weather data for the entered city.

* **Explanation:**
* **api\_key**: The unique key to authenticate requests with the API. It ensures only authorized requests are processed.
* **city**: The user-provided city name for which weather data is fetched.
* **url**: The endpoint for accessing the OpenWeatherMap API, formatted with the city name and API key.

1. **API Response and Error Handling:**

* The purpose of the API response and error handling is to ensure that the application retrieves weather data reliably and gracefully handles any issues that may occur during the request process. This includes:
* Verifying the API's response.
* Parsing valid data for use in the application.
* Providing clear feedback to the user in case of errors, such as incorrect input, connection issues, or server problems.

1. **Displaying Weather Data:**

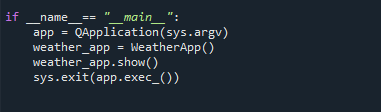
****

* **Purpose:**

Parses the API response and displays relevant data in the GUI.

* **Explanation:**
* Converts temperature from Kelvin to Fahrenheit.
* Retrieves and displays weather descriptions and emojis for better visualization.

1. **Running the Application:**

****

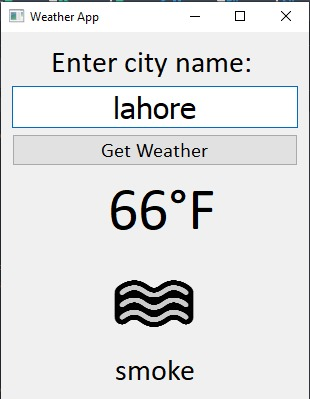
* **Purpose:**

Starts the application and displays the main window.

* **Explanation:**
* **QApplication**: Initializes the application environment.
* **sys.exit()**: Ensures a clean termination of the program.

**Output:**

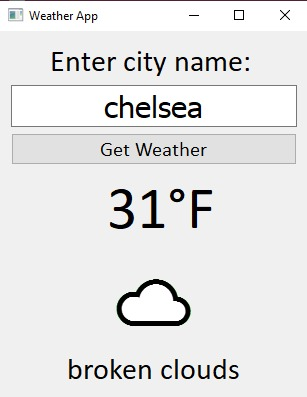
* **Weather Displayed of city:**

****

* **Can Handle Errors:**

****

* **Can handle lowercase:**

****

****

**Conclusion:**

The Weather App developed in this project demonstrates the practical application of PyQt5 for creating a graphical user interface (GUI) and integrating it with external data sources, specifically the OpenWeatherMap API. This app allows users to input a city name, retrieve real-time weather data, and display the results in a visually intuitive format. The use of weather emojis based on specific weather conditions enhances the user experience, making the app both functional and engaging.

Throughout the project, key concepts such as API requests, error handling, and dynamic UI updates were implemented to ensure that the app provides accurate weather information while handling various potential issues (e.g., network errors or incorrect city names). Additionally, the app’s design was optimized for user-friendliness with clear input fields, buttons, and labels, ensuring that users can easily interact with the app to obtain the weather information they need.

This project has not only enhanced my understanding of working with PyQt5 and APIs but has also allowed me to explore how to create responsive and visually appealing applications. The ability to integrate external data into a GUI application opens up possibilities for further enhancements, such as adding more detailed weather information or supporting additional languages. Overall, the Weather App serves as a solid foundation for building more complex applications that rely on real-time data and interactive interfaces.

**Appendix:**

Code For the Project:

import sys

import requests

from PyQt5.QtWidgets import (QApplication, QWidget, QLabel,

QLineEdit, QPushButton, QVBoxLayout)

from PyQt5.QtCore import Qt

class WeatherApp (QWidget):

def \_\_init\_\_(self):

super().\_\_init\_\_()

self.city\_label = QLabel("Enter city name: ", self)

self.city\_input = QLineEdit(self)

self.get\_weather\_button = QPushButton("Get Weather", self)

self.temperature\_label = QLabel(self)

self.emoji\_label = QLabel(self)

self.description\_label = QLabel(self)

self.initUI()

def initUI(self):

self.setWindowTitle("Weather App")

vbox= QVBoxLayout()

vbox.addWidget(self.city\_label)

vbox.addWidget(self.city\_input)

vbox.addWidget(self.get\_weather\_button)

vbox.addWidget(self.temperature\_label)

vbox.addWidget(self.emoji\_label)

vbox.addWidget(self.description\_label)

self.setLayout(vbox)

self.city\_label.setAlignment(Qt.AlignCenter)

self.city\_input.setAlignment(Qt.AlignCenter)

self.temperature\_label.setAlignment(Qt.AlignCenter)

self.emoji\_label.setAlignment(Qt.AlignCenter)

self.description\_label.setAlignment(Qt.AlignCenter)

self.city\_label.setObjectName("city\_label")

self.city\_input.setObjectName("city\_input")

self.get\_weather\_button.setObjectName("get\_weather\_button")

self.emoji\_label.setObjectName("emoji\_label")

self.temperature\_label.setObjectName("temperature\_label")

self.description\_label.setObjectName("description\_label")

self.setStyleSheet("""

QLabel, QPushButton {

font-family: Calibri;

}

QLabel#city\_label{

font-size: 30px;

}

QLineEdit#city\_input{

font-size: 30px;

}

QPushButton#get\_weather\_button{

font-size: 20px;

}

QLabel#temperature\_label{

font-size: 40px;

}

QLabel#emoji\_label{

font-size: 70px;

font-family: Segoe UI emoji;

}

QLabel#description\_label {

font-size:30px;

}

""")

self.get\_weather\_button.clicked.connect(self.get\_weather)

def get\_weather(self):

api\_key= "0eb86eeda6a27ef47d4868f45511bf51"

city= self.city\_input.text()

url = f"https://api.openweathermap.org/data/2.5/weather?q={city}&appid={api\_key}"

try:

response = requests.get(url)

response.raise\_for\_status()

data = response.json()

if data["cod"] == 200:

self.display\_weather(data)

except requests.exceptions.HTTPError as http\_error:

match response.status\_code:

case 400:

self.display\_error("Bad request:\nPlease check your input")

case 401:

self.display\_error("Unauthorized:\nInvalid API key")

case 403:

self.display\_error("Forbidden:\nAccess is denied")

case 404:

self.display\_error("Not found:\nCity not found")

case 500:

self.display\_error("Internal Server Error:\nPlease try again later")

case 502:

self.display\_error("Bad Gateway:\nInvalid response from the server")

case 503:

self.display\_error("Service Unavailable:\nServer is down")

case 504:

self.display\_error("Gateway Timeout:\nNo response from the server")

case \_:

self.display\_error(f"HTTP error occured:\n{http\_error}")

except requests.exceptions.ConnectionError:

self.display\_error("Connection Error:\nCheck your internet connection")

except requests.exceptions.Timeout:

self.display\_error("Timout Error:\nThe request timed out")

except requests.exceptions.TooManyRedirects:

self.display\_error("Too many Redirects:\nCheck the URL")

except requests.exceptions.RequestException as req\_error:

self.display\_error(f"Request Error:\n{req\_error}")

def display\_error (self, message):

self.temperature\_label.setStyleSheet("font-size: 25px;")

self.temperature\_label.setText(message)

self.emoji\_label.clear()

self.description\_label.clear()

def display\_weather (self, data):

self.temperature\_label.setStyleSheet("font-size: 60px;")

temperature\_k = data["main"]["temp"]

temperature\_f = (temperature\_k \* 9/5) - 459.67

weather\_id = int(data["weather"][0]["id"])

weather\_description = data["weather"][0]["description"]

self.temperature\_label.setText(f"{temperature\_f: .0f}°F")

self.emoji\_label.setText(self.get\_weather\_emoji(weather\_id))

self.description\_label.setText(weather\_description)

@staticmethod

def get\_weather\_emoji(weather\_id):

if 200 <= weather\_id <= 232:

return "⛈️"

elif 300 <= weather\_id <= 321:

return "🌦️"

elif 500 <= weather\_id <= 531:

return "🌧️"

elif 600 <= weather\_id <= 622:

return "❄️"

elif 701 <= weather\_id <= 741:

return "🌫️"

elif weather\_id == 762:

return "🌋"

elif weather\_id == 771:

return "💨"

elif weather\_id == 781:

return "🌪️"

elif weather\_id == 800:

return "☀️"

elif 801 <= weather\_id <= 804:

return "☁️"

else:

return ""

#change the emoji font here

if \_\_name\_\_== "\_\_main\_\_":

app = QApplication(sys.argv)

weather\_app = WeatherApp()

weather\_app.show()

sys.exit(app.exec\_())