

PROJECT REPORT

ON

**LIBRARY MANAGEMENT SYSTEM**

Submitted By: Submitted To:

Rajat Singh Chauhan Pawandeep Sharma

UID: 24MCI10086 Assistant Professor

Class: MCA AI/ML E2916

Section: 24MAM 2-A

# University Institute of Computing Chandigarh University, Gharuan, Mohali

**Project Report**

**Library Management System**

**Student Name: Rajat Singh Chauhan**  **UID :24MCI10086**

**Branch: MCA(AI & ML) Section/Group: 24MAM2(A)**

**Semester : 1st** **Date of Performance:21/10/2024**

**Subject Name: Python Programming Subject Code :**24**CAH606**

***1. Introduction***

The Weather Data Dashboard is a GUI-based application developed using Python’s Tkinter library for real-time weather data retrieval, display, and visualization. The application allows users to view and interact with weather data for selected cities, with data sourced from the OpenWeatherMap API. Users can select or enter a city, fetch forecasted weather data, listen to an audio description of the weather via text-to-speech (TTS), view temperature and humidity graphs, and clear the displayed data.

***2. Objective***

The purpose of this project is to develop an interactive, user-friendly weather dashboard that provides users with essential weather information and visualizes it for easy understanding.- To maintain an accurate **book availability** status, ensuring that only available books can be borrowed and returned books are updated in the system.

- To provide a foundation for further expansion, including advanced features such as overdue tracking, fines, and more sophisticated search functionalities.

***3.Functionalities***

City Selection: Users can choose a city from a dropdown list or manually enter one.

Data Retrieval: On clicking "Fetch Weather Data," the app fetches the 5-day forecast for the chosen city from OpenWeatherMap’s API.

Data Display: Weather data, including date, temperature, humidity, and conditions, are displayed in a scrollable text box.

Text-to-Speech (TTS): The application uses a TTS engine to read out the weather forecast.

Data Visualization: Graphs showing temperature and humidity trends over the next 5 days.

Data Clearing: An option to reset the displayed data and inputs.

***4.Technical Design and Implementation***

**Libraries Used:**

**Tkinter**: For GUI components.

**Requests**: For HTTP requests to fetch weather data.

**Pyttsx3:** To enable TTS capabilities.

**Matplotlib**: For data visualization of weather parameters over time.

**API Integration**:

The OpenWeatherMap API provides weather data in JSON format. The application extracts temperature, humidity, and weather description for the forecast.

**Multi-threading:**

The TTS feature is run in a separate thread to avoid freezing the GUI while reading weather data.

**Event-Driven Programming:**

The application uses Tkinter’s button and command bindings to create interactive elements.

***5.Class and Method Descriptions***

Class WeatherDashboard:

Initializes GUI components and TTS engine.

Manages user interaction and triggers weather data fetch, TTS playback, data clearing, and data visualization.

Methods:

fetch\_weather(): Fetches weather data, parses JSON response, extracts relevant data, and populates the GUI.

start\_speaking(): Starts TTS in a separate thread.

speak\_weather(): Reads out the weather forecast through the TTS engine.

stop\_speaking(): Stops the TTS engine mid-sentence if required.

clear\_all\_data(): Clears data from the GUI and resets input fields.

plot\_data(): Generates a dual-axis plot for temperature and humidity over the next 5 days.

***6.CODE FOR THE PROJECT:***

import tkinter as tk

from tkinter import messagebox, scrolledtext

from tkinter import ttk

import requests

import matplotlib.pyplot as plt

from datetime import datetime

from matplotlib.dates import DateFormatter

import pyttsx3

import threading

class WeatherDashboard:

    def \_\_init\_\_(self, root):

        self.root = root

        self.root.title("Weather Data Dashboard")

        self.root.configure(bg='lightblue')

        # Initialize TTS engine

        self.tts\_engine = pyttsx3.init()

        self.tts\_thread = None

        self.stop\_tts\_flag = threading.Event()

        # Full screen dimensions

        scrn\_width = 800 # self.root.winfo\_screenwidth()

        scrn\_height = 600 #self.root.winfo\_screenheight()

        self.root.geometry(f"{scrn\_width}x{scrn\_height}+0+0")

        # Label and Combobox for City selection or entry

        self.city\_label = tk.Label(root, text="Select or Enter City Name:", font=("Arial", 12), bg='#C4D7FF')

        self.city\_label.pack(pady=5)

        self.cities = [

            "Delhi", "Mumbai", "Chennai", "Kolkata", "Bengaluru",

            "Hyderabad", "Pune", "Ahmedabad", "Jaipur", "Chandigarh"

        ]

        self.city\_combo = ttk.Combobox(root, values=self.cities, font=("Arial", 12))

        self.city\_combo.set("Enter or select a city")

        self.city\_combo.pack(pady=5)

        # Fetch Weather Data Button

        self.fetch\_button = tk.Button(root, text="Fetch Weather Data", command=self.fetch\_weather, font=("Arial", 12))

        self.fetch\_button.pack(pady=1)

        # ScrolledText widget to display weather data

        self.data\_display = scrolledtext.ScrolledText(root, width=50, height=30, font=("Arial", 10))

        self.data\_display.pack(pady=1)

        # Speak and Stop Speaking Buttons

        self.speak\_button = tk.Button(root, text="Speak Weather", command=self.start\_speaking, font=("Arial", 12), bg='gray')

        self.speak\_button.pack(pady=1)

        self.stop\_button = tk.Button(root, text="Stop Speaking", command=self.stop\_speaking, font=("Arial", 12), bg='red')

        self.stop\_button.pack(pady=1)

        # Clear Data Button

        self.clear\_button = tk.Button(root, text="Clear All Data", command=self.clear\_all\_data, font=("Arial", 12), bg='yellow')

        self.clear\_button.pack(pady=5)

    def fetch\_weather(self):

        """Fetch weather data for the selected city."""

        city = self.city\_combo.get().strip()

        if not city or city == "Enter or select a city":

            messagebox.showwarning("Warning", "Please enter or select a valid city name.")

            return

        api\_key = "a6f0ee5717157a4a27657b79c5d913cb"  # Replace with your API key

        url = f"http://api.openweathermap.org/data/2.5/forecast?q={city}&appid={api\_key}&units=metric"

        try:

            response = requests.get(url)

            response.raise\_for\_status()

            data = response.json()

            if data["cod"] != "200":

                messagebox.showerror("Error", f"City not found: {city}")

                return

            self.data\_display.delete(1.0, tk.END)

            self.weather\_info = ""

            dates, temperatures, humidities = [], [], []

            unique\_days = set()

            # Limit to only the next 5 days

            for item in data['list']:

                date\_obj = datetime.fromtimestamp(item['dt'])

                day\_str = date\_obj.strftime('%Y-%m-%d')

                if day\_str not in unique\_days:

                    temp = item['main']['temp']

                    humidity = item['main']['humidity']

                    description = item['weather'][0]['description']

                    entry = f"Date: {day\_str}, Temp: {temp}°C, Humidity: {humidity}%, Condition: {description}\n"

                    self.data\_display.insert(tk.END, entry)

                    self.weather\_info += f"On {day\_str}, the temperature is {temp} degrees Celsius with {description}. "

                    dates.append(date\_obj)

                    temperatures.append(temp)

                    humidities.append(humidity)

                    unique\_days.add(day\_str)

                    # Stop if we have collected data for 5 days

                    if len(unique\_days) == 5:

                        break

            self.plot\_data(city, dates, temperatures, humidities)

        except requests.exceptions.HTTPError as http\_err:

            messagebox.showerror("HTTP Error", f"HTTP error occurred: {http\_err}")

        except Exception as e:

            messagebox.showerror("Error", f"An error occurred: {e}")

    def start\_speaking(self):

        """Start speaking weather data in a separate thread."""

        if hasattr(self, 'weather\_info') and self.weather\_info:

            self.speak\_button.config(state=tk.DISABLED)

            self.stop\_tts\_flag.clear()

            self.tts\_thread = threading.Thread(target=self.speak\_weather)

            self.tts\_thread.start()

        else:

            messagebox.showwarning("Warning", "No weather data to speak.")

    def speak\_weather(self):

        """Speak the weather condition using TTS."""

        for sentence in self.weather\_info.split('. '):

            if self.stop\_tts\_flag.is\_set():

                break

            self.tts\_engine.say(sentence)

            self.tts\_engine.runAndWait()

        # Re-enable the Speak button after speech completes

        self.speak\_button.config(state=tk.NORMAL)

    def stop\_speaking(self):

        """Stop the TTS engine from speaking."""

        if self.tts\_thread and self.tts\_thread.is\_alive():

            self.stop\_tts\_flag.set()

            self.tts\_engine.stop()

        # Re-enable the Speak button after stopping

        self.speak\_button.config(state=tk.NORMAL)

    def clear\_all\_data(self):

        """Clear all displayed data."""

        self.data\_display.delete(1.0, tk.END)

        self.city\_combo.set("Enter or select a city")

        self.weather\_info = ""

    def plot\_data(self, city, dates, temperatures, humidities):

        """Plot temperature and humidity data."""

        fig, ax1 = plt.subplots(figsize=(12, 6))

        ax1.set\_xlabel("Date")

        ax1.set\_ylabel("Temperature (°C)", color='tab:red')

        ax1.plot(dates, temperatures, color='tab:red', marker='o', linewidth=2, label='Temperature')

        ax1.tick\_params(axis='y', labelcolor='tab:red')

        ax2 = ax1.twinx()

        ax2.set\_ylabel("Humidity (%)", color='tab:blue')

        ax2.plot(dates, humidities, color='tab:blue', marker='o', linewidth=2, label='Humidity')

        ax2.tick\_params(axis='y', labelcolor='tab:blue')

        date\_format = DateFormatter("%d-%m")

        ax1.xaxis.set\_major\_formatter(date\_format)

        plt.setp(ax1.get\_xticklabels(), rotation=45, ha='right')

        plt.title(f"Weather Data for {city} (Next 5 Days)")

        fig.tight\_layout()

        ax1.legend(loc='upper left')

        ax2.legend(loc='upper right')

        plt.grid()

        plt.show()

# Run the application

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

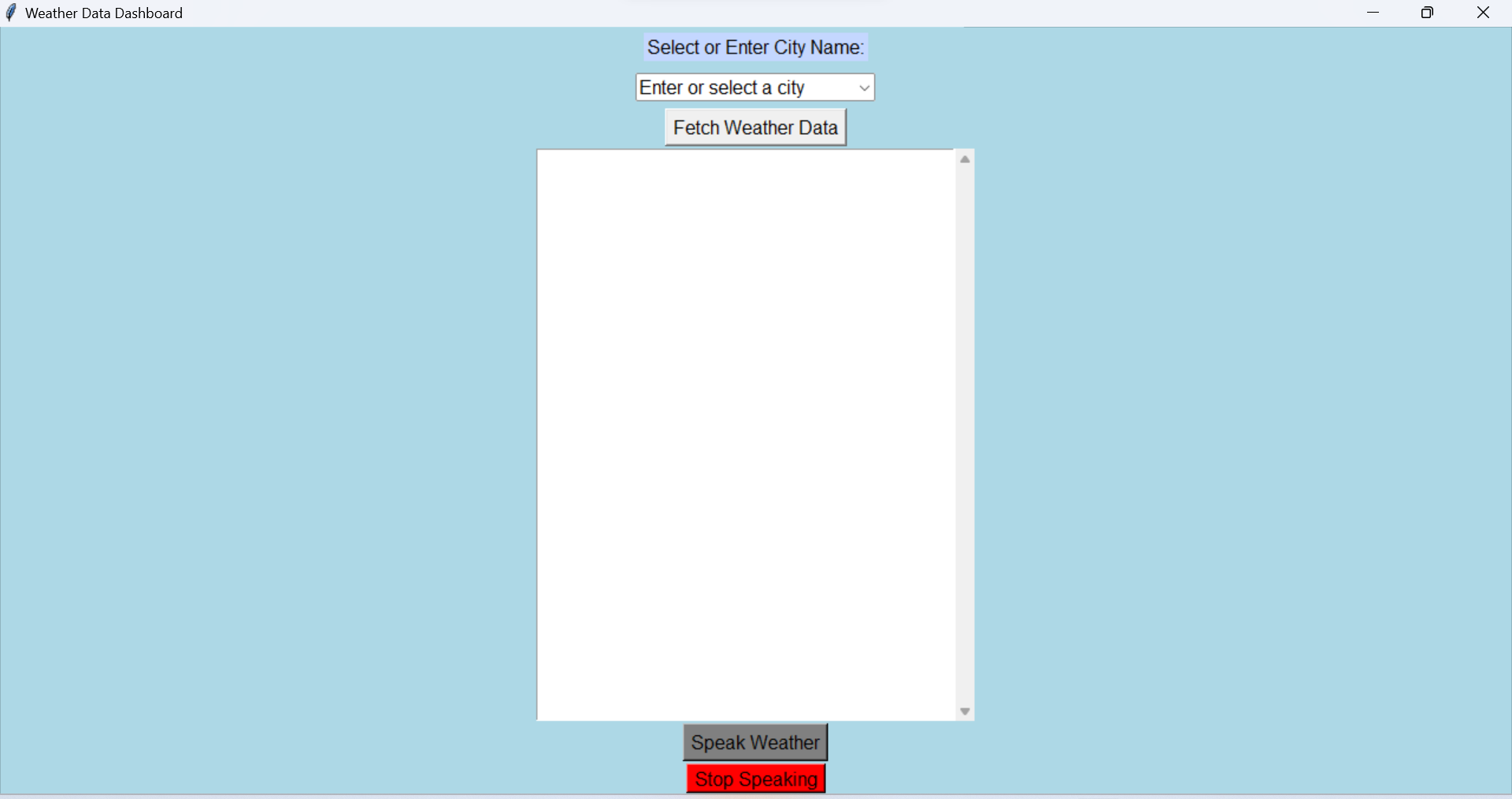
    root = tk.Tk()

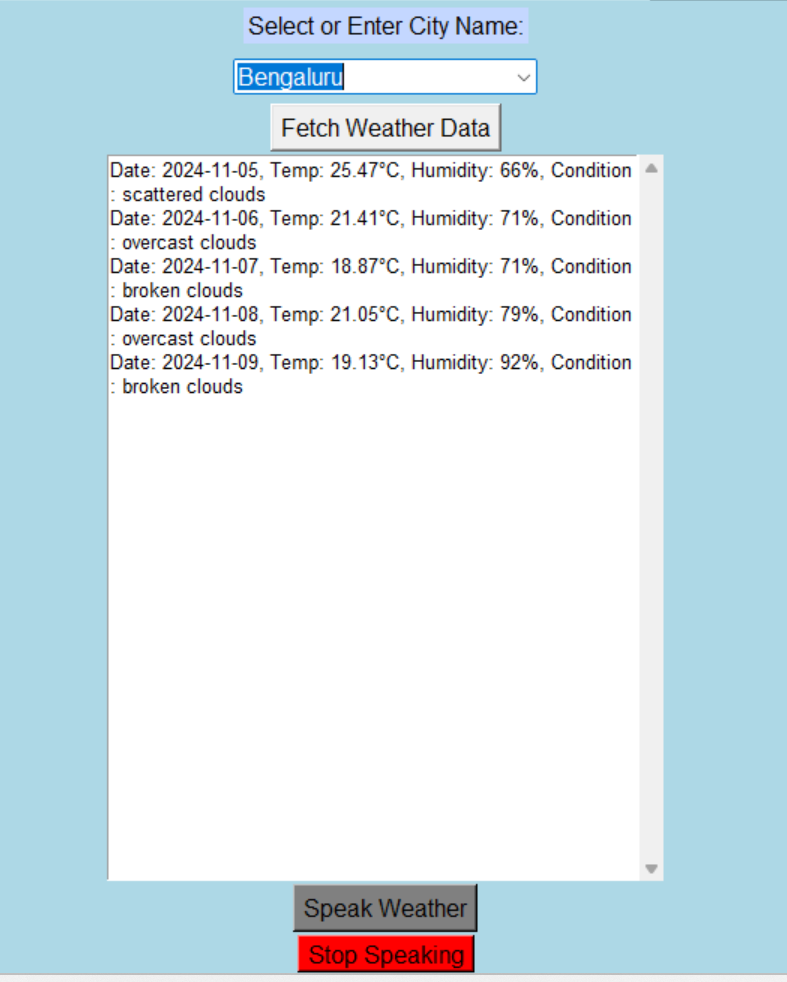
    app = WeatherDashboard(root)

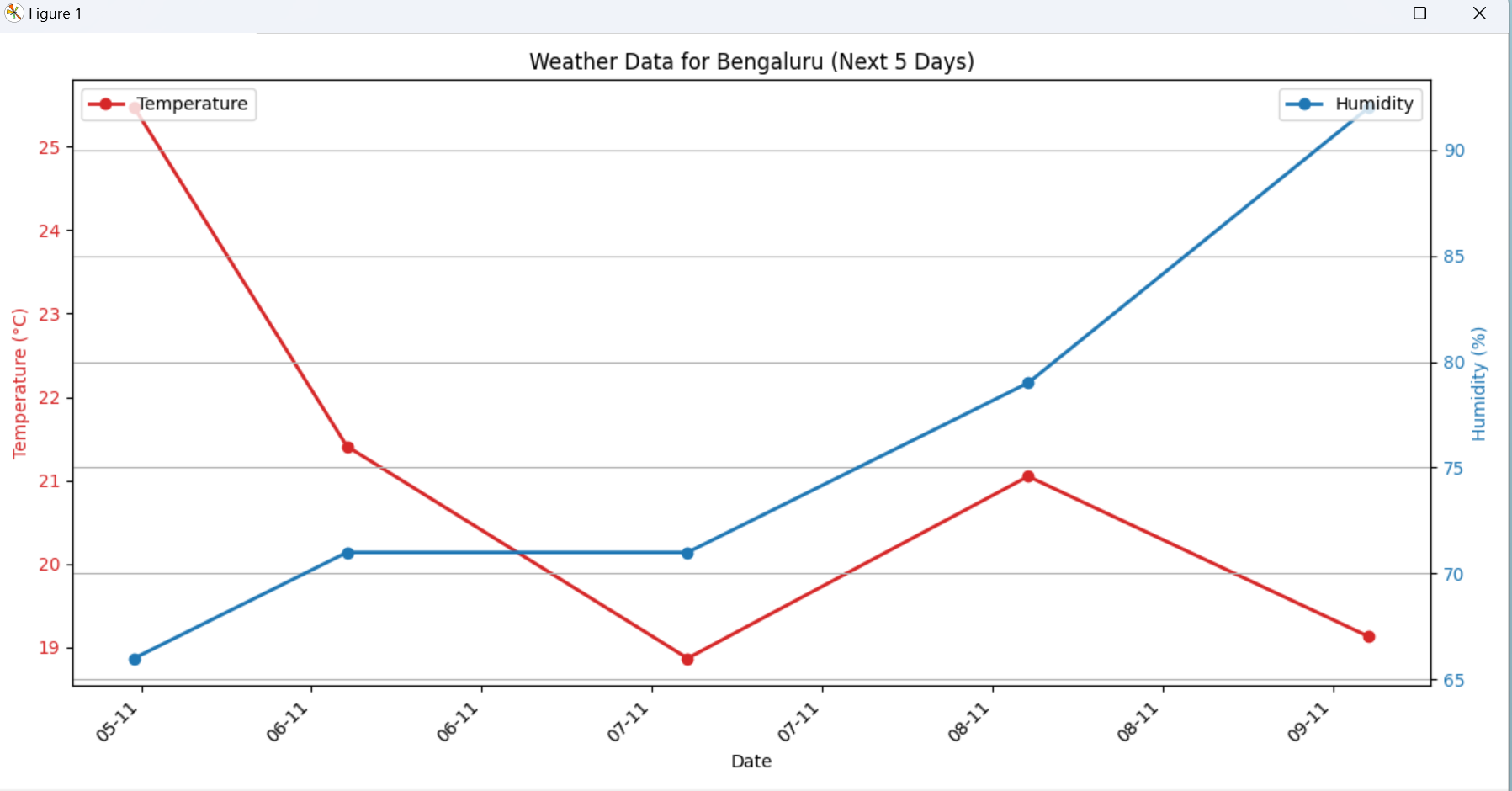
   # root.iconbitmap("Weather.ico")

    root.mainloop()

***7.RESULT***

******

******

******

***8.******Challenges and Solutions***

GUI Freezing during TTS:

To prevent the GUI from freezing during TTS playback, multi-threading was implemented for the speak\_weather() function, allowing asynchronous TTS playback.

Handling Invalid City Names:

Added validation to check if the user has selected a city or entered a valid city name, displaying error messages if input is incorrect.

Plotting with Matplotlib:

Dual-axis plotting was used to display temperature and humidity simultaneously. A custom date formatter was added for readability on the x-axis.

***9.Future Improvements***

Error Handling and Input Validation: Further enhance input validation to handle edge cases such as network failures or API rate limits.

Real-Time Data Updates: Integrate real-time updates by automatically refreshing the data at intervals.

Enhanced TTS Features: Add voice control features, allowing users to start, stop, or customize the TTS playback rate and voice.

Styling and User Interface: Improve the UI with enhanced styling, layout adjustments, and possibly custom icons.

***10. Conclusion***

**The Weather Data Dashboard application is an effective tool for users to access weather data in a visually and audibly accessible format. It demonstrates effective use of GUI components, API integration, TTS, data visualization, and multi-threading. This project provides a solid foundation for further enhancement and scalability for real-world applications.**