Python Project 11: Weather API Application

1. Project Overview

Build a simple desktop **Weather API Application** that allows the user to enter a city and view its current weather.

The project uses the **Open Meteo API**, which provides weather data through JSON HTTP requests and does not require an API key for non commercial use. Its Geocoding API converts a city name into latitude and longitude, which are then used by the Weather Forecast API.

2. Features

- Search weather by city
- Convert city name to coordinates
- Current temperature
- Feels like temperature
- Humidity
- Wind speed
- Weather condition
- API error handling
- Loading/status messages
- Simple Tkinter interface
- No external Python packages

3. Technologies

- Python 3
- Tkinter
- `urllib`
- `json`
- Open Meteo API
- Object Oriented Programming

Python's urllib.request can open HTTP URLs and retrieve data directly, making it suitable for this small API project without installing an additional HTTP library.

4. Folder Structure

```
weather_api_application/
□□□ main.py
□□□ ui.py
□□□ weather_api.py
□□□ README.md
```

5. How the Project Works

The application follows this flow:

```
User enters city
    □
Geocoding API
    □
Latitude + Longitude
    □
Weather API
    □
JSON weather data
    □
Python processes response
    □
Tkinter displays weather
```

Open Meteo's geocoding endpoint accepts a location name and returns information including latitude, longitude, timezone, and country.

The forecast endpoint accepts latitude and longitude and can return current variables such as temperature, apparent temperature,

humidity, precipitation, weather code, and wind speed.

6. Complete Backend Code

weather_api.py

```
import json
from urllib.parse import urlencode
from urllib.request import urlopen
from urllib.error import HTTPError, URLError


class WeatherAPI:

    GEOCODING_URL = (
        "https://geocoding-api.open-meteo.com/v1/search"
    )

    WEATHER_URL = (
        "https://api.open-meteo.com/v1/forecast"
    )

    WEATHER_CODES = {
        0: "Clear Sky",
        1: "Mainly Clear",
        2: "Partly Cloudy",
        3: "Overcast",
        45: "Fog",
        48: "Depositing Rime Fog",
        51: "Light Drizzle",
        53: "Moderate Drizzle",
        55: "Dense Drizzle",
        61: "Slight Rain",
        63: "Moderate Rain",
```

```
      65: "Heavy Rain",
      71: "Slight Snow",
      73: "Moderate Snow",
      75: "Heavy Snow",
      80: "Slight Rain Showers",
      81: "Moderate Rain Showers",
      82: "Violent Rain Showers",
      95: "Thunderstorm",
      96: "Thunderstorm With Hail",
      99: "Thunderstorm With Heavy Hail"
    }

    def request_json(self, url, params):
      query_string = urlencode(params)
      full_url = f"{url}?{query_string}"

      try:
        with urlopen(full_url, timeout=10) as response:
          return json.loads(response.read().decode("utf-8"))

      except HTTPError as error:
        raise Exception(
          f"API request failed: HTTP {error.code}"
        )

      except URLError:
        raise Exception(
          "Could not connect to the weather service."
        )

      except TimeoutError:
        raise Exception(
          "The weather request timed out."
        )
```

```
    def get_coordinates(self, city):
        params = {
            "name": city,
            "count": 1,
            "language": "en",
            "format": "json"
        }

        data = self.request_json(
            self.GEOCODING_URL,
            params
        )

        results = data.get("results")

        if not results:
            raise ValueError(
                f"Could not find the city: {city}"
            )

        location = results[0]

        return {
            "name": location["name"],
            "country": location.get("country", ""),
            "latitude": location["latitude"],
            "longitude": location["longitude"]
        }

    def get_weather(self, city):
        location = self.get_coordinates(city)

        params = {
```

```
            "latitude": location["latitude"],
            "longitude": location["longitude"],
            "current": (
                "temperature_2m,"
                "relative_humidity_2m,"
                "apparent_temperature,"
                "weather_code,"
                "wind_speed_10m"
            ),
            "temperature_unit": "celsius",
            "wind_speed_unit": "kmh"
        }

        data = self.request_json(
            self.WEATHER_URL,
            params
        )

        current = data.get("current", {})

        weather_code = current.get("weather_code")

        return {
            "city": location["name"],
            "country": location["country"],
            "temperature": current.get(
                "temperature_2m"
            ),
            "feels_like": current.get(
                "apparent_temperature"
            ),
            "humidity": current.get(
                "relative_humidity_2m"
            ),
```

```
            "wind_speed": current.get(
                "wind_speed_10m"
            ),
            "condition": self.WEATHER_CODES.get(
                weather_code,
                "Unknown"
            )
        }
```

7. Complete Frontend Code

ui.py

```
import tkinter as tk
from tkinter import messagebox

from weather_api import WeatherAPI


class WeatherAppUI:

    def __init__(self, root):
        self.root = root
        self.root.title("Weather API Application")
        self.root.geometry("600x520")
        self.root.configure(bg="#0f172a")

        self.weather_api = WeatherAPI()

        self.city_var = tk.StringVar()

        self.create_ui()

    def create_ui(self):
```

```
        title = tk.Label(
            self.root,
            text="Weather App",
            font=("Arial", 26, "bold"),
            bg="#0f172a",
            fg="white"
        )
        title.pack(pady=(35, 5))

        subtitle = tk.Label(
            self.root,
            text="Search current weather by city",
            font=("Arial", 11),
            bg="#0f172a",
            fg="#94a3b8"
        )
        subtitle.pack()

        search_frame = tk.Frame(
            self.root,
            bg="#0f172a"
        )
        search_frame.pack(
            fill="x",
            padx=60,
            pady=30
        )

        city_entry = tk.Entry(
            search_frame,
            textvariable=self.city_var,
            font=("Arial", 12),
            bg="#1e293b",
            fg="white",
```

```
            insertbackground="white",
            relief="flat"
        )
        city_entry.pack(
            side="left",
            fill="x",
            expand=True,
            ipady=10,
            padx=(0, 10)
        )

        city_entry.bind(
            "<Return>",
            lambda event: self.search_weather()
        )

        search_button = tk.Button(
            search_frame,
            text="Search",
            command=self.search_weather,
            font=("Arial", 11, "bold"),
            bg="#2563eb",
            fg="white",
            relief="flat",
            padx=20,
            pady=9
        )
        search_button.pack(side="right")

        self.location_label = tk.Label(
            self.root,
            text="Enter a city",
            font=("Arial", 20, "bold"),
            bg="#0f172a",
```

```
            fg="white"
        )
        self.location_label.pack(pady=10)

        self.condition_label = tk.Label(
            self.root,
            text="",
            font=("Arial", 14),
            bg="#0f172a",
            fg="#94a3b8"
        )
        self.condition_label.pack()

        self.temperature_label = tk.Label(
            self.root,
            text="--°C",
            font=("Arial", 42, "bold"),
            bg="#0f172a",
            fg="white"
        )
        self.temperature_label.pack(pady=15)

        details_frame = tk.Frame(
            self.root,
            bg="#1e293b",
            padx=30,
            pady=20
        )
        details_frame.pack(
            fill="x",
            padx=60
        )

        self.feels_label = tk.Label(
```

```
            details_frame,
            text="Feels Like: --°C",
            font=("Arial", 11),
            bg="#1e293b",
            fg="white"
        )
        self.feels_label.pack(pady=4)

        self.humidity_label = tk.Label(
            details_frame,
            text="Humidity: --%",
            font=("Arial", 11),
            bg="#1e293b",
            fg="white"
        )
        self.humidity_label.pack(pady=4)

        self.wind_label = tk.Label(
            details_frame,
            text="Wind: -- km/h",
            font=("Arial", 11),
            bg="#1e293b",
            fg="white"
        )
        self.wind_label.pack(pady=4)

        self.status_label = tk.Label(
            self.root,
            text="",
            font=("Arial", 10),
            bg="#0f172a",
            fg="#94a3b8"
        )
        self.status_label.pack(pady=20)
```

```
    def search_weather(self):

        city = self.city_var.get().strip()

        if not city:
            messagebox.showwarning(
                "Missing City",
                "Please enter a city name."
            )
            return

        self.status_label.config(
            text="Fetching weather..."
        )

        self.root.update_idletasks()

        try:
            weather = self.weather_api.get_weather(city)

            self.location_label.config(
                text=f"{weather['city']}, "
                     f"{weather['country']}"
            )

            self.condition_label.config(
                text=weather["condition"]
            )

            self.temperature_label.config(
                text=f"{weather['temperature']}°C"
            )
```

```
            self.feels_label.config(
                text=f"Feels Like: "
                    f"{weather['feels_like']}°C"
            )

            self.humidity_label.config(
                text=f"Humidity: "
                    f"{weather['humidity']}%"
            )

            self.wind_label.config(
                text=f"Wind: "
                    f"{weather['wind_speed']} km/h"
            )

            self.status_label.config(
                text="Weather updated successfully."
            )

        except Exception as error:

            self.status_label.config(
                text="Unable to fetch weather."
            )

            messagebox.showerror(
                "Weather Error",
                str(error)
            )
```

8. Main Code

main.py

```
import tkinter as tk
```

```
from ui import WeatherAppUI


def main():
    root = tk.Tk()
    WeatherAppUI(root)
    root.mainloop()


if __name__ == "__main__":
    main()
```

9. README

README.md

```
# Weather API Application

A simple Python desktop application that fetches
current weather information for a city.

## Features

- City search
- Current temperature
- Feels like temperature
- Humidity
- Wind speed
- Weather condition

## Requirements

Python 3
```

No external packages are required.

API

Open Meteo Weather API

Run

python main.py

10. How Frontend + Backend Connect

ui.py

WeatherAPI

get_coordinates()

Open Meteo Geocoding API

latitude + longitude

get_weather()

Open Meteo Weather API

JSON response

ui.py

Display weather

The API response is JSON, so Python's json module converts the response into Python dictionaries that the application can use.

11. How to Run

Open the project directory:

```
cd weather_api_application
```

Run:

```
python main.py
```

Enter a city such as:

```
Lahore
```

or:

```
London
```

Then click **Search**.

12. Basic Testing

Test these cases:

```
Lahore
Islamabad
Karachi
London
Tokyo
```

Also test:

- Empty city input

- Invalid city name
- Internet unavailable
- API timeout
- Pressing Enter instead of clicking Search

Expected result:

```
Lahore, Pakistan

Clear Sky

32°C

Feels Like: 34°C
Humidity: 45%
Wind: 12 km/h
```

The exact values will change because the application retrieves live weather data from the API.

13. Small Improvements

- Add 7 day forecast
- Add weather icons
- Add sunrise and sunset
- Add hourly forecast
- Add recent city history
- Add Celsius/Fahrenheit switch
- Add automatic location detection
- Add dark/light theme switch
- Add background changes based on weather

Visit haas.dev for more resources and guides.