Digital Empowerment Pakistan

Name: Maryam Tariq

C++ Programming Internship

Task 1

Weather Data Retrieval Application

Table of Contents

Documentation and user manual

Introduction

This application provides a console user interface for managing weather data. Users can add and remove locations, retrieve weather forecasts, historical weather data, and air quality data, and export collected data in CSV, JSON, or TXT formats.

Addictions

C++ Compiler: Make sure you have a C++ compiler that supports C++11 or later.

Libraries:

cURL: For creating HTTP requests.

nlohmann/json: For handling JSON data.

Setup instructions

deInstall pendencies:

cURL: Install it via your package manager (eg apt-get install libcurl4-openssl-dev on Debian-based systems).

nlohmann/json: Download the json.hpp file from the nlohmann/json GitHub repository and place it in the project directory.

Compile the code:

To compile, use a command like the following:

g++ -std=c++11 -o weather_app weather_app.cpp -lcurl

Make sure the json.hpp file is in the same directory or adjust the include paths accordingly.

Launching the application

Run the executable:

./weatherapp

Console based user interface:

The application offers a menu with the following options:

Add Place: Add a new place by entering its name, latitude and longitude.

Remove location: Remove an existing location by entering its name.

Location list: List of all currently saved locations.

Fetch Weather Forecast: Get the weather forecast for a specific location.

Retrieve historical weather data: Retrieves historical weather data for a location between the specified start and end dates.

Get air quality data: Get air quality data for a given location.

Export data: Export weather data for a specific location to CSV, JSON and TXT formats.

Exit: Ending the application.

Functionality details

1. Add a location

It prompts the user to enter:

Location Name: The name of the location.

Latitude: latitude coordinates.

Longitude: Longitude coordinates

Adds the location to an internal list for future operations.

2. Remove location

It prompts the user to enter:

Location Name: The name of the location you want to delete.

Removes a location from the internal list.

3. List the locations

Displays all currently saved locations with their names, latitudes, and longitudes.

4. Load the weather forecast

It prompts the user to enter:

Location Name: The name of the location to retrieve the forecast for.

Loads and displays the hourly temperature and precipitation forecast.

5. Load historical weather data

It prompts the user to enter:

Location Name: Location name for historical data.

Start Date: Start date for historical data (format: YYYY-MM-DD).

End date: End date for historical data (format: YYYY-MM-DD).

Retrieves and displays the maximum daily temperatures for the specified time period.

6. Load air quality data

It prompts the user to enter:

Location Name: Location name for air quality data.

Loads and displays the air quality index (AQI).

7. Data export
It prompts the user to enter:

Location Name: The name of the location to export the data to.

Temperature: The temperature value to be included in the export.

Wind Speed: The wind speed value to include in the export.

Exports weather data in three formats:

CSV: Contains location information and weather variables.

JSON: Contains structured data with location and weather variables.

TXT: Contains formatted text data.

8. The end
Terminates the application.

Code overview
Location Class: Represents a geographic location with a name, latitude, and longitude.

WeatherVariable Class: Represents a weather variable with a name and a value.

WeatherForecastingSystem Class: Retrieves and displays weather forecasts using the Open Meteo API.

HistoricalWeatherSystem Class: Retrieves and displays historical weather data.

AirQualityForecastingSystem Class: Gets and displays air quality data using the WAQI API.

Export function: Function to export weather data to CSV, JSON and TXT formats.

Main Features: Provides a console-based user interface for user interaction.

API keys
Replace "YOUR_WEATHER_API_KEY", "YOUR_HISTORICAL_API_KEY" and "YOUR_AIR_QUALITY_API_KEY" with your actual API keys in the main() function.

Troubleshooting
Failed to open file: Make sure the file path is correct and you have write permission.

API errors: If you encounter problems loading data, check your API keys and network connection.

Contributions and support

You can find contributions or support in the documentation of the libraries used or in the community forums

This documentation should help users set up, run, and understand the application. Adjust any specific paths or configurations as needed for your environment.

CODE:

```
#include <iostream>
#include <vector>
#include <fstream>
#include <string>
#include <map>
#include <nlohmann/json.hpp>
#include <curl/curl.h> // For making HTTP requests

using json = nlohmann::json;
using namespace std;

// Helper function to handle HTTP responses
size_t WriteCallback(void* contents, size_t size, size_t nmemb, void* userp) {
    ((string*)userp)->append((char*)contents, size * nmemb);
    return size * nmemb;
}

// Define the Location class
class Location {
public:
    string name;
    double latitude;
    double longitude;

    Location(string name, double lat, double lon) : name(name), latitude(lat), longitude(lon) {}
};

// Define the WeatherVariable class
```

```
class WeatherVariable {
public:
    string name;
    double value;

    WeatherVariable(const string& n, double v) : name(n), value(v) {}
};

// Define the WeatherForecastingSystem class
class WeatherForecastingSystem {
private:
    string apiKey;

public:
    WeatherForecastingSystem(const string& api_key) : apiKey(api_key) {}

    json fetchForecast(const Location& location) {
        CURL* curl;
        CURLcode res;
        string readBuffer;
        json data;

        curl = curl_easy_init();
        if (curl) {
            string url = "https://api.open-meteo.com/v1/forecast?latitude=" + to_string(location.latitude) +
"&longitude=" + to_string(location.longitude) + "&hourly=temperature_2m,precipitation";
            curl_easy_setopt(curl, CURLOPT_URL, url.c_str());
            curl_easy_setopt(curl, CURLOPT_WRITEFUNCTION, WriteCallback);
            curl_easy_setopt(curl, CURLOPT_WRITEDATA, &readBuffer);
```

```
        res = curl_easy_perform(curl);
        if (res != CURLE_OK) {
            cerr << "curl_easy_perform() failed: " << curl_easy_strerror(res) << endl;
        }
        curl_easy_cleanup(curl);

        // Parse the response
        data = json::parse(readBuffer);
    }
    return data;
  }

  void displayForecast(const json& forecastData) const {
    cout << "Temperature: ";
    for (const auto& temp : forecastData["hourly"]["temperature_2m"]) {
        cout << temp << " ";
    }
    cout << endl;

    cout << "Precipitation: ";
    for (const auto& precip : forecastData["hourly"]["precipitation"]) {
        cout << precip << " ";
    }
    cout << endl;
  }
};

// Define the HistoricalWeatherSystem class
class HistoricalWeatherSystem {
```

```
private:
    string apiKey;

public:
    HistoricalWeatherSystem(const string& api_key) : apiKey(api_key) {}

    json fetchHistoricalData(const Location& location, const string& startDate, const string& endDate) {
        CURL* curl;
        CURLcode res;
        string readBuffer;
        json data;

        curl = curl_easy_init();
        if (curl) {
            string url = "https://api.open-meteo.com/v1/forecast?latitude=" + to_string(location.latitude) +
"&longitude=" + to_string(location.longitude) +
"&daily=temperature_2m_max,temperature_2m_min&start_date=" + startDate + "&end_date=" +
endDate;
            curl_easy_setopt(curl, CURLOPT_URL, url.c_str());
            curl_easy_setopt(curl, CURLOPT_WRITEFUNCTION, WriteCallback);
            curl_easy_setopt(curl, CURLOPT_WRITEDATA, &readBuffer);
            res = curl_easy_perform(curl);
            if (res != CURLE_OK) {
                cerr << "curl_easy_perform() failed: " << curl_easy_strerror(res) << endl;
            }
            curl_easy_cleanup(curl);

            // Parse the response
            data = json::parse(readBuffer);
        }
```

```
        return data;
    }

    void displayHistoricalData(const json& historicalData) const {
        cout << "Historical Temperature Data: ";
        for (const auto& temp : historicalData["daily"]["temperature_2m_max"]) {
            cout << temp << " ";
        }
        cout << endl;
    }
};

// Define the AirQualityForecastingSystem class
class AirQualityForecastingSystem {
private:
    string apiKey;

public:
    AirQualityForecastingSystem(const string& api_key) : apiKey(api_key) {}

    json fetchAirQuality(const Location& location) {
        CURL* curl;
        CURLcode res;
        string readBuffer;
        json data;

        curl = curl_easy_init();
        if (curl) {
```

```
        string url = "https://api.waqi.info/feed/geo:" + to_string(location.latitude) + ";" +
to_string(location.longitude) + "/?token=" + apiKey;
        curl_easy_setopt(curl, CURLOPT_URL, url.c_str());
        curl_easy_setopt(curl, CURLOPT_WRITEFUNCTION, WriteCallback);
        curl_easy_setopt(curl, CURLOPT_WRITEDATA, &readBuffer);
        res = curl_easy_perform(curl);
        if (res != CURLE_OK) {
            cerr << "curl_easy_perform() failed: " << curl_easy_strerror(res) << endl;
        }
        curl_easy_cleanup(curl);

        // Parse the response
        data = json::parse(readBuffer);
      }
      return data;
   }

   void displayAirQuality(const json& airQualityData) const {
      cout << "Air Quality Index: " << airQualityData["data"]["aqi"] << endl;
   }
};

// Define data export functions
void exportToCSV(const Location& location, const vector<WeatherVariable>& weatherVariables, const
string& filename) {
   ofstream file(filename);
   if (file.is_open()) {
      file << "Location: " << location.name << ", Latitude: " << location.latitude << ", Longitude: " <<
location.longitude << endl;
      file << "Weather Variable,Value" << endl;
```

```
        for (const auto& var : weatherVariables) {
            file << var.name << "," << var.value << endl;
        }
        file.close();
    }
    else {
        cout << "Failed to open file for writing." << endl;
    }
}

void exportToJSON(const Location& location, const vector<WeatherVariable>& weatherVariables, const string& filename) {
    json j;
    j["Location"]["Name"] = location.name;
    j["Location"]["Latitude"] = location.latitude;
    j["Location"]["Longitude"] = location.longitude;

    for (const auto& var : weatherVariables) {
        json varJson;
        varJson["name"] = var.name;
        varJson["value"] = var.value;
        j["WeatherData"].push_back(varJson);
    }

    ofstream file(filename);
    if (file.is_open()) {
        file << j.dump(4);
        file.close();
    }
```

```
    else {
        cout << "Failed to open file for writing." << endl;
    }
}

void exportToTXT(const Location& location, const vector<WeatherVariable>& weatherVariables, const string& filename) {
    ofstream file(filename);
    if (file.is_open()) {
        file << "Location Information:\n";
        file << "  Name: " << location.name << "\n";
        file << "  Latitude: " << location.latitude << "\n";
        file << "  Longitude: " << location.longitude << "\n\n";
        file << "Weather Data:\n";
        for (const auto& var : weatherVariables) {
            file << "  " << var.name << ": " << var.value << "\n";
        }
        file.close();
    }
    else {
        cout << "Failed to open file for writing." << endl;
    }
}

// Define the main function with a console-based UI
int main() {
    LocationManager locationManager;
    WeatherForecastingSystem weatherSystem("YOUR_WEATHER_API_KEY");
    HistoricalWeatherSystem historicalSystem("YOUR_HISTORICAL_API_KEY");
```

```
AirQualityForecastingSystem airQualitySystem("YOUR_AIR_QUALITY_API_KEY");

while (true) {
    cout << "\n1. Add Location" << endl;
    cout << "2. Remove Location" << endl;
    cout << "3. List Locations" << endl;
    cout << "4. Fetch Weather Forecast" << endl;
    cout << "5. Fetch Historical Weather Data" << endl;
    cout << "6. Fetch Air Quality Data" << endl;
    cout << "7. Export Data" << endl;
    cout << "8. Exit" << endl;

    int choice;
    cin >> choice;

    switch (choice) {
    case 1: {
        string name;
        double latitude, longitude;
        cout << "Enter location name: ";
        cin >> name;
        cout << "Enter latitude: ";
        cin >> latitude;
        cout << "Enter longitude: ";
        cin >> longitude;
        Location location(name, latitude, longitude);
        locationManager.addLocation(location);
        break;
    }
```

```
case 2: {
    string name;
    cout << "Enter location name to remove: ";
    cin >> name;
    locationManager.removeLocation(name);
    break;
}
case 3: {
    locationManager.listLocations();
    break;
}
case 4: {
    string name;
    cout << "Enter location name for forecast: ";
    cin >> name;
    Location* location = locationManager.findLocation(name);
    if (location) {
        json forecastData = weatherSystem.fetchForecast(*location);
        weatherSystem.displayForecast(forecastData);
    }
    else {
        cout << "Location not found." << endl;
    }
    break;
}
case 5: {
    string name, startDate, endDate;
    cout << "Enter location name for historical data: ";
    cin >> name;
```

```
    cout << "Enter start date (YYYY-MM-DD): ";
    cin >> startDate;
    cout << "Enter end date (YYYY-MM-DD): ";
    cin >> endDate;
    Location* location = locationManager.findLocation(name);
    if (location) {
        json historicalData = historicalSystem.fetchHistoricalData(*location, startDate, endDate);
        historicalSystem.displayHistoricalData(historicalData);
    }
    else {
        cout << "Location not found." << endl;
    }
    break;
}
case 6: {
    string name;
    cout << "Enter location name for air quality data: ";
    cin >> name;
    Location* location = locationManager.findLocation(name);
    if (location) {
        json airQualityData = airQualitySystem.fetchAirQuality(*location);
        airQualitySystem.displayAirQuality(airQualityData);
    }
    else {
        cout << "Location not found." << endl;
    }
    break;
}
case 7: {
```

```
        string name;
        double temperature, windSpeed;
        cout << "Enter location name for export: ";
        cin >> name;
        cout << "Enter temperature: ";
        cin >> temperature;
        cout << "Enter wind speed: ";
        cin >> windSpeed;
        Location* location = locationManager.findLocation(name);
        if (location) {
            vector<WeatherVariable> weatherVariables = {
                WeatherVariable("Temperature", temperature),
                WeatherVariable("Wind Speed", windSpeed)
            };
            exportToCSV(*location, weatherVariables, "data.csv");
            exportToJSON(*location, weatherVariables, "data.json");
            exportToTXT(*location, weatherVariables, "data.txt");
        }
        else {
            cout << "Location not found." << endl;
        }
        break;
    }
    case 8:
        return 0;
    default:
        cout << "Invalid choice. Please try again." << endl;
    }}
return 0;}
```