**MUHAMMAD SHAHZAIB**

**DEP TASK 1**

**User Manual**

Table of Contents

[Air Quality Weather Forecast App 3](#_Toc171541308)

[1. Introduction 3](#_Toc171541309)

[1.1. How to Use: 3](#_Toc171541310)

[2. Classes and Functionalities (Application Breakdown): 3](#_Toc171541311)

[2.1: Classes 3](#_Toc171541312)

[2.2. Functionalities 3](#_Toc171541313)

[Example Workflow: 4](#_Toc171541314)

[CODE: 5](#_Toc171541315)

# Air Quality Weather Forecast App

## 1. Introduction

Welcome to the Weather Forecasting System user manual. This application allows you to manage locations, weather forecasts, and air quality forecasts, providing functionalities to export data to various formats for offline use.

### 1.1. How to Use:

* The program prompts you to enter a location's name.
* The application fetches weather forecasts, historical weather data, and air quality forecasts for the specified location.
* The fetched data is displayed and can be exported to CSV, JSON, and TXT files for later use.

## 2. Classes and Functionalities (Application Breakdown):

### 2.1: Classes

* **Location**: Represents a geographical location with a name, latitude, and longitude.
* **WeatherVariable**: Holds weather data, including temperature and wind speed.
* **WeatherForecastingSystem**: Fetches and displays current weather data.
* **HistoricalWeatherSystem**: Fetches and displays historical weather data.
* **AirQualityForecastingSystem**: Fetches and displays air quality data.

### 2.2. Functionalities

* **Fetching Weather Forecasts**:
  + Method: fetchWeatherForecast(location)
  + Description: Fetches current weather data for a given location.
* **Fetching Historical Weather Data**:
  + Method: fetchHistoricalWeather(location)
  + Description: Fetches historical weather data for a given location.
* **Fetching Air Quality Data**:
  + Method: fetchAirQualityForecast(location)
  + Description: Fetches air quality data for a given location.
* **Displaying Weather Variables**:
  + Method: display()
  + Description: Displays the fetched weather variables for a given location.
* **Exporting Data to CSV**:
  + Function: exportAllToCSV(weatherData, filename)
  + Description: Exports all weather data to a CSV file.
* **Exporting Data to JSON**:
  + Function: exportAllToJSON(weatherData, filename)
  + Description: Exports all weather data to a JSON file.
* **Exporting Data to TXT**:
  + Function: exportAllToTXT(weatherData, filename)
  + Description: Exports all weather data to a TXT file.

## Example Workflow:

1. **Start the Application**:
   * The application prompts you to enter a location's name.
2. **Fetch Data**:
   * The application fetches weather forecasts, historical weather data, and air quality forecasts for the specified location.
3. **Display Data**:
   * The fetched data is displayed on the console.
4. **Export Data**:
   * The data is exported to CSV, JSON, and TXT files for offline use.

## CODE:

#include <iostream>

#include <string>

#include <vector>

#include <fstream>

#include <sstream>

#include "curl/curl.h"

#include "json/json.h"

using namespace std;

// Location class

class Location {

public:

    string name;

    double latitude;

    double longitude;

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

    void display() {

        cout << "Location: " << name << ", Latitude: " << latitude << ", Longitude: " << longitude << endl;

    }

};

// WeatherVariable class

class WeatherVariable {

public:

    string name;

    double value;

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

    void display() {

        cout << "Weather Variable: " << name << ", Value: " << value << endl;

    }

};

// WeatherForecastingSystem class

class WeatherForecastingSystem {

public:

    string apiKey;

    string apiUrl;

    WeatherForecastingSystem(string apikey, string apiurl) : apiKey(apikey), apiUrl(apiurl) {}

    vector<WeatherVariable> fetchWeatherForecast(Location location) {

        CURL\* curl;

        CURLcode res;

        string readBuffer;

        curl\_global\_init(CURL\_GLOBAL\_DEFAULT);

        curl = curl\_easy\_init();

        if (curl) {

            string locationKey = getLocationKey(location);

            string url = apiUrl + "/currentconditions/v1/" + locationKey + "?apikey=" + apiKey;

            curl\_easy\_setopt(curl, CURLOPT\_URL, url.c\_str());

            curl\_easy\_setopt(curl, CURLOPT\_WRITEFUNCTION, writeMemoryCallback);

            curl\_easy\_setopt(curl, CURLOPT\_WRITEDATA, &readBuffer);

            res = curl\_easy\_perform(curl);

            if (res != CURLE\_OK) {

                cout << "cURL error: " << curl\_easy\_strerror(res) << endl;

            }

            curl\_easy\_cleanup(curl);

        }

        curl\_global\_cleanup();

        vector<WeatherVariable> weatherVariables;

        Json::CharReaderBuilder builder;

        Json::Value root;

        istringstream s(readBuffer);

        string errs;

        if (Json::parseFromStream(builder, s, &root, &errs)) {

            double temperature = root[0]["Temperature"]["Metric"]["Value"].asDouble();

            double windSpeed = root[0]["Wind"]["Speed"]["Metric"]["Value"].asDouble();

            WeatherVariable temperatureVar("Temperature", temperature);

            WeatherVariable windSpeedVar("Wind Speed", windSpeed);

            weatherVariables.push\_back(temperatureVar);

            weatherVariables.push\_back(windSpeedVar);

        } else {

            cout << "Error parsing JSON: " << errs << endl;

        }

        return weatherVariables;

    }

    static size\_t writeMemoryCallback(void\* ptr, size\_t size, size\_t nmemb, void\* data) {

        string& readBuffer = \*(static\_cast<string\*>(data));

        readBuffer.append((char\*)ptr, size \* nmemb);

        return size \* nmemb;

    }

private:

    string getLocationKey(Location location) {

        CURL\* curl;

        CURLcode res;

        string readBuffer;

        string locationKey;

        curl\_global\_init(CURL\_GLOBAL\_DEFAULT);

        curl = curl\_easy\_init();

        if (curl) {

            string url = apiUrl + "/locations/v1/cities/geoposition/search?apikey=" + apiKey + "&q=" + to\_string(location.latitude) + "," + to\_string(location.longitude);

            curl\_easy\_setopt(curl, CURLOPT\_URL, url.c\_str());

            curl\_easy\_setopt(curl, CURLOPT\_WRITEFUNCTION, writeMemoryCallback);

            curl\_easy\_setopt(curl, CURLOPT\_WRITEDATA, &readBuffer);

            res = curl\_easy\_perform(curl);

            if (res != CURLE\_OK) {

                cout << "cURL error: " << curl\_easy\_strerror(res) << endl;

            }

            curl\_easy\_cleanup(curl);

        }

        curl\_global\_cleanup();

        Json::CharReaderBuilder builder;

        Json::Value root;

        istringstream s(readBuffer);

        string errs;

        if (Json::parseFromStream(builder, s, &root, &errs)) {

            locationKey = root["Key"].asString();

        } else {

            cout << "Error parsing JSON: " << errs << endl;

        }

        return locationKey;

    }

};

// HistoricalWeatherSystem class

class HistoricalWeatherSystem {

public:

    string apiKey;

    string apiUrl;

    HistoricalWeatherSystem(string apikey, string apiurl) : apiKey(apikey), apiUrl(apiurl) {}

    vector<WeatherVariable> fetchHistoricalWeather(Location location) {

        // Implement similar to fetchWeatherForecast

        return vector<WeatherVariable>();

    }

    static size\_t writeMemoryCallback(void\* ptr, size\_t size, size\_t nmemb, void\* data) {

        string& readBuffer = \*(static\_cast<string\*>(data));

        readBuffer.append((char\*)ptr, size \* nmemb);

        return size \* nmemb;

    }

};

// AirQualityForecastingSystem class

class AirQualityForecastingSystem {

public:

    string apiKey;

    string apiUrl;

    AirQualityForecastingSystem(string apikey, string apiurl) : apiKey(apikey), apiUrl(apiurl) {}

    vector<WeatherVariable> fetchAirQualityForecast(Location location) {

        // Implement similar to fetchWeatherForecast

        return vector<WeatherVariable>();

    }

    static size\_t writeMemoryCallback(void\* ptr, size\_t size, size\_t nmemb, void\* data) {

        string& readBuffer = \*(static\_cast<string\*>(data));

        readBuffer.append((char\*)ptr, size \* nmemb);

        return size \* nmemb;

    }

};

// Function to export all data to CSV file

void exportAllToCSV(const vector<vector<WeatherVariable>>& weatherData, const string& filename, const Location& location) {

    ofstream file(filename);

    if (file.is\_open()) {

        // Write location information

        file << "Location: " << location.name << ", Latitude: " << location.latitude << ", Longitude: " << location.longitude << endl;

        // Write weather variables

        file << "Weather Variable,Value" << endl;

        for (const auto& weatherVariables : weatherData) {

            for (const auto& weatherVariable : weatherVariables) {

                file << weatherVariable.name << "," << weatherVariable.value << endl;

            }

        }

        file.close();

    }

    else {

        cout << "Unable to open file" << endl;

    }

}

// Function to export all data to JSON file

void exportAllToJSON(const vector<vector<WeatherVariable>>& weatherData, const string& filename, const Location& location) {

    Json::Value root;

    root["Location"]["Name"] = location.name;

    root["Location"]["Latitude"] = location.latitude;

    root["Location"]["Longitude"] = location.longitude;

    for (const auto& weatherVariables : weatherData) {

        for (const auto& weatherVariable : weatherVariables) {

            Json::Value var;

            var["name"] = weatherVariable.name;

            var["value"] = weatherVariable.value;

            root["WeatherData"].append(var);

        }

    }

    ofstream file(filename);

    if (file.is\_open()) {

        file << root;

        file.close();

    }

    else {

        cout << "Unable to open file" << endl;

    }

}

// Function to export all data to TXT file

void exportAllToTXT(const vector<vector<WeatherVariable>>& weatherData, const string& filename, const Location& location) {

    ofstream file(filename);

    if (file.is\_open()) {

        file << "Location: " << location.name << ", Latitude: " << location.latitude << ", Longitude: " << location.longitude << endl;

        for (const auto& weatherVariables : weatherData) {

            for (const auto& weatherVariable : weatherVariables) {

                file << "Weather Variable: " << weatherVariable.name << ", Value: " << weatherVariable.value << endl;

            }

        }

        file.close();

    }

    else {

        cout << "Unable to open file" << endl;

    }

}

int main() {

    string apiKey = "YOUR\_API\_KEY";

    string apiUrl = "http://dataservice.accuweather.com";

    WeatherForecastingSystem weatherForecastingSystem(apiKey, apiUrl);

    HistoricalWeatherSystem historicalWeatherSystem(apiKey, apiUrl);

    AirQualityForecastingSystem airQualityForecastingSystem(apiKey, apiUrl);

    cout << "Enter location: ";

    string locationName;

    double lat1, lon1;

    cin >> locationName;

    cout << "Enter Latitude: ";

    cin >> lat1;

    cout << "Enter Longitude: ";

    cin >> lon1;

    Location location(locationName, lat1, lon1);

    vector<WeatherVariable> weatherForecast = weatherForecastingSystem.fetchWeatherForecast(location);

    vector<WeatherVariable> historicalWeather = historicalWeatherSystem.fetchHistoricalWeather(location);

    vector<WeatherVariable> airQualityForecast = airQualityForecastingSystem.fetchAirQualityForecast(location);

    cout << "Weather Forecast:" << endl;

    for (auto& weatherVariable : weatherForecast) {

        weatherVariable.display();

    }

    cout << "Historical Weather:" << endl;

    for (auto& weatherVariable : historicalWeather) {

        weatherVariable.display();

    }

    cout << "Air Quality Forecast:" << endl;

    for (auto& weatherVariable : airQualityForecast) {

        weatherVariable.display();

    }

    vector<vector<WeatherVariable>> allWeatherData = {weatherForecast, historicalWeather, airQualityForecast};

    exportAllToCSV(allWeatherData, "all\_weather\_data.csv");

    exportAllToJSON(allWeatherData, "all\_weather\_data.json", location);

    exportAllToTXT(allWeatherData, "all\_weather\_data.txt", location);

    return 0;

}