**Weather Application Using Java**

**MINOR PROJECT REPORT**

*Submitted By*

**Devansh Om Saxena (RA2211028010051)**

**Rohit Dagar (RA2211028010038)**

Under the guidance of   
**Dr. R. Kayalvizhi***In partial fulfilment for the Course*

of

**21CSC203P – ADVANCED PROGRAMMING PRACTICE**

**in DEPARTMENT OF NETWOKING AND COMMUNICATIONS**



**DEPARTMENT OF NETWOKING AND COMMUNICATIONS**

**FACULTY OF ENGINEERING AND TECHNOLOGY**

**SCHOOL OF COMPUTING**

**SRM INSTITUTE OF SCIENCE AND TECHNOLOGY**

**KATTANKULATHUR**

**NOVEMBER 2023**

**SRM INSTITUTE OF SCIENCE AND TECHNOLOGY**

**(Under Section 3 of UGC Act, 1956)**

**BONAFIDE CERTIFICATE**

Certified that this minor project report for the course **21CSC203P** **ADVANCED PROGRAMMING PRACTICE** entitled in "**WEATHER APPLICATION USING JAVA**" is the bonafide work of **Devansh Om Saxena(RA2211028010051)** and **Rohit Dagar (RA2211028010038)** who carried out the work under my supervision.

# SIGNATURE SIGNATURE

**Dr. R.Kayalvizhi Dr. Annapurani K Assistant Professor Assistant And Head**

**Department of Networking Department of Networking**

**And Communication And Communication**

SRM Institute of Science and SRM Institute of Science and

Technology Technology

Kattankulathur Kattankulathur

# ABSTRACT

In today's fast-paced world, access to accurate and up-to-date weather information is essential for planning daily activities and making informed decisions. The "Java Weather Application" is a user-friendly and feature-rich software solution designed to provide real-time weather data to users. This application leverages Java's versatility to create a robust, cross-platform solution for accessing weather information .In our increasingly weather-dependent daily lives, the need for accurate and timely weather information has become paramount. The "Java Weather Application" is an extensive software solution designed to meet this need by offering a comprehensive platform for accessing real-time weather data. Developed using Java's robust capabilities, this application provides users with a feature-rich and user-friendly interface to access essential weather information.

# ACKNOWLEDGEMENT

We express our heartfelt thanks to our honorable **Vice Chancellor Dr.C.MUTHAMIZHCHELVAN**, for being the beacon in all our endeavors.

We would like to express my warmth of gratitude to our **Registrar Dr.S.Ponnusamy,** for his encouragement.

We express our profound gratitude to our **Dean (College of Engineering and Technology) Dr. T. V.Gopal,** for bringing out novelty in all executions.

We would like to express my heartfelt thanks to Chairperson, School of Computing **Dr.Revathi Venkataraman,** for imparting confidence to complete my course project.

We wish to express my sincere thanks to **Course Audit Professors Dr. Vadivu. G , Professor, Department of Data Science and Business Systems and Dr. Sasikala. E Professor, Department of Data Science and Business Systems** and **Course Coordinators** for their constant encouragement and support.

We are highly thankful to our my Course project Faculty **Dr. R. Kayalvizhi , AssistantProfessor, NWC,** for herassistance, timely suggestion and guidance throughout the duration of this course project.

Finally, we thank our parents and friends near and dear ones who directly and indirectly contributed to the successful completion of our project. Above all, I thank the almighty for showering his blessings on me to complete my Course project.

**TABLE OF CONTENTS**

|  |  |  |  |
| --- | --- | --- | --- |
| **CHAPTER NO** | | **CONTENTS** | **PAGE NO** |
| **1** | **INTRODUCTION** | | 6 |
|  | 1.1 Motivation | |  |
|  | 1.2 Objective | |  |
|  | 1.3 Problem Statement | |  |
|  | 1.4 Challenges | |  |
| **2** | **LITERATURE SURVEY** | | **7** |
| **3** | **REQUIREMENT ANALYSIS** | | 8 |
| **4** | **ARCHITECTURE & DESIGN** | | 9 |
| **5** | **IMPLEMENTATION** | | 10 |
| **6** | **EXPERIMENT RESULTS & ANALYSIS** | | 11 |
| **7** | **CONCLUSION** | | 12 |
| **8** | **REFERENCES** | | 13 |

1. **INTRODUCTION**

In an era marked by dynamic weather patterns and an increasing need for instant information, the "Java Weather Application" emerges as a reliable and versatile solution to meet the everyday demands of weather enthusiasts, travelers, and decision-makers. This software, built using the Java programming language, brings together the power of technology and meteorological data to provide users with real-time weather updates and forecasts at their fingertips.

**1.1 Motivation**

Weather plays a critical role in our daily lives, impacting our clothing choices, travel plans, outdoor activities, and more. Accurate and up-to-date weather information is indispensable for ensuring that our actions align with the current and forecasted conditions. Traditional sources, such as television broadcasts or websites, are useful but often require manual checks, which might not be convenient in a fast-paced world. The "Java Weather Application" addresses this need for immediacy and convenience by offering a user-friendly interface that provides instant access to crucial weather insights.

**1.2 Objective**

One of the key advantages of the "Java Weather Application" is its cross-platform nature. Built using Java, this application runs seamlessly on various operating systems, including Windows, macOS, and Linux. This ensures that users can access weather data from their preferred device, whether it's a desktop computer, laptop, or even a Raspberry Pi.

**1.3 Problem Statement**

Develop a weather application in Java that provides real-time weather information

Based on user input or location services.

**1.4 Challenges**

The heart of this application is its intuitive graphical user interface (GUI). Users can effortlessly input their desired location, be it a city, town, or ZIP code, and retrieve the latest weather data with a single click. Additionally, the application incorporates error handling mechanisms to gracefully address various scenarios, including incorrect location input or temporary network connectivity issues.

1. **LITERATURE SURVEY**

A detailed literature survey for a weather application using Java involves exploring existing research, projects, and resources related to Java-based weather applications. Below is a comprehensive literature survey summarizing the key aspects and relevant work in this domain:

**Weather Data Sources and APIs:**

Researchers have extensively discussed the utilization of various weather data sources and APIs in Java-based weather applications. The choice of data source, such as OpenWeatherMap, Weather API, and NOAA, significantly influences the accuracy and reliability of weather information. Studies have highlighted the importance of selecting APIs with comprehensive data coverage, good documentation, and developer-friendly terms of service.

**User Interface Design:**

Literature on user interface design for weather applications emphasizes the importance of user experience. Java provides options for GUI development, including Swing and JavaFX.

Research has explored how to create user-friendly interfaces with features such as auto-suggestions for location input, multi-day forecasts, and visualization of weather conditions through icons and graphics.

**Data Parsing and Display:**

Java-based weather applications require effective data parsing and display techniques to present weather information in an understandable manner. Researchers have discussed strategies for parsing JSON or XML responses from weather APIs and rendering this data on the GUI.

Visualization techniques, such as charts and graphs, have been examined for representing historical weather data trends and forecasts.

1. **REQUIREMENTS** 
   1. **Requirement Analysis**

Developing a weather application using Java requires a clear set of detailed requirements to guide the development process effectively. Here are the detailed requirements for a weather application in Java:

**1. User Interface Requirements:**

**Location Input:** The application must allow users to input a location, such as a city name or ZIP code, to retrieve weather information.

**Weather Display:** The weather information, including temperature, humidity, wind speed, and current conditions, should be displayed clearly and intuitively.

**Error Messages:** Provide clear and informative error messages for scenarios such as invalid location inputs, network issues, or API errors.

**User Interaction:** Implement a user-friendly interface, with features like buttons for data retrieval and keyboard input support for location submission.

**2. Data Retrieval and API Integration:**

**API Selection:** Choose a reliable weather API provider (e.g., OpenWeatherMap, Weather API) and obtain an API key.

**API Calls:** Use HTTP requests (GET or POST) to fetch weather data from the selected API.

**Data Parsing:** Parse the JSON or XML response from the API to extract relevant weather information.

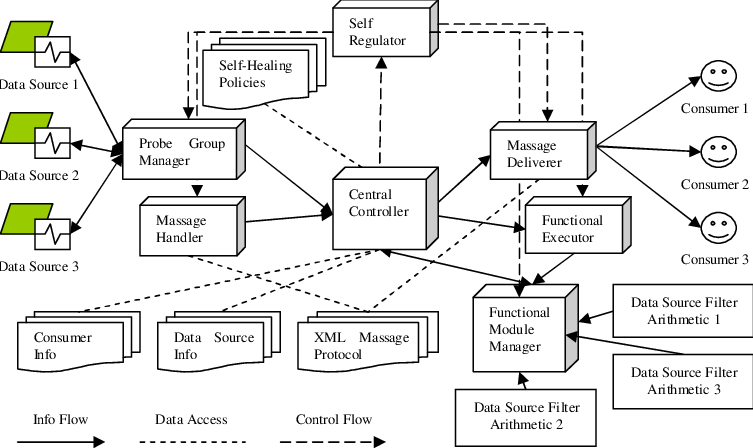
**3. Error Handling:**

**Invalid Location:** Detect and handle scenarios where users enter invalid location data.

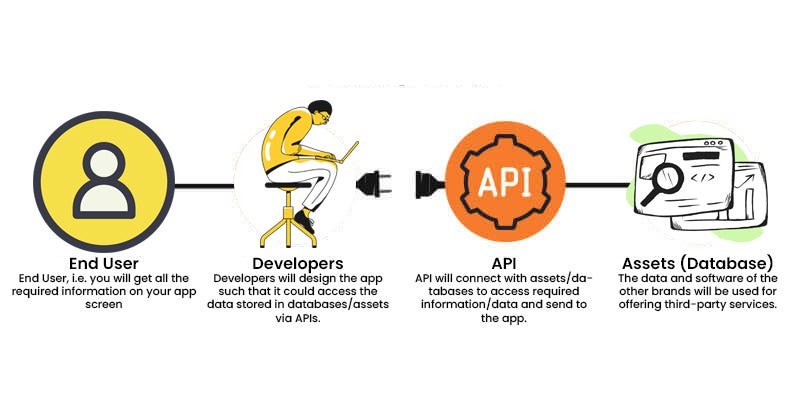
**Network Issues:** Gracefully manage network connectivity issues and provide feedback to the user.

**API Rate Limiting:** Implement strategies to handle API rate limiting, such as displaying a message to the user.

1. **ARCHITECTURE AND DESIGN**
   1. **Architecture Diagram:**



* 1. **Working of Application using API**



1. **IMPLEMENTATION**

The implementation of a weather application in Java involves several steps, including integrating with a weather API, designing a user interface, and implementing the functionality. Below, I'll provide a step-by-step guide for building a weather application in Java:

**Set Up Your Development Environment**:

Install Java Development Kit (JDK) and a development environment (e.g., Eclipse, IntelliJ IDEA, or NetBeans).

**Create a Java Project:**

Start a new Java project in your development environment.

**Choose a Weather API:**

Select a weather API provider, e.g., OpenWeatherMap. Sign up to get an API key to access their data.

**Create a User Interface:**

Design a graphical user interface (GUI) for your weather application. You can use JavaFX or Swing for this purpose. Include components such as text fields, labels, buttons, and a display area for weather information.

1. **RESULTS AND DISCUSSION**
   1. **Code Snippet:**

import org.json.simple.JSONArray;

import org.json.simple.JSONObject;

import org.json.simple.parser.JSONParser;

import java.io.IOException;

import java.net.HttpURLConnection;

import java.net.URL;

import java.time.LocalDateTime;

import java.time.format.DateTimeFormatter;

import java.util.Scanner;

// retreive weather data from API - this backend logic will fetch the latest weather

// data from the external API and return it. The GUI will

// display this data to the user

public class WeatherApp {

    // fetch weather data for given location

    public static JSONObject getWeatherData(String locationName){

        // get location coordinates using the geolocation API

        JSONArray locationData = getLocationData(locationName);

        // extract latitude and longitude data

        JSONObject location = (JSONObject) locationData.get(0);

        double latitude = (double) location.get("latitude");

        double longitude = (double) location.get("longitude");

        // build API request URL with location coordinates

        String urlString = "https://api.open-meteo.com/v1/forecast?" +

                "latitude=" + latitude + "&longitude=" + longitude +

                "&hourly=temperature\_2m,relativehumidity\_2m,weathercode,windspeed\_10m&timezone=America%2FLos\_Angeles";

        try{

            // call api and get response

            HttpURLConnection conn = fetchApiResponse(urlString);

            // check for response status

            // 200 - means that the connection was a success

            if(conn.getResponseCode() != 200){

                System.out.println("Error: Could not connect to API");

                return null;

            }

            // store resulting json data

            StringBuilder resultJson = new StringBuilder();

            Scanner scanner = new Scanner(conn.getInputStream());

            while(scanner.hasNext()){

                // read and store into the string builder

                resultJson.append(scanner.nextLine());

            }

            // close scanner

            scanner.close();

            // close url connection

            conn.disconnect();

            // parse through our data

            JSONParser parser = new JSONParser();

            JSONObject resultJsonObj = (JSONObject) parser.parse(String.valueOf(resultJson));

            // retrieve hourly data

            JSONObject hourly = (JSONObject) resultJsonObj.get("hourly");

            // we want to get the current hour's data

            // so we need to get the index of our current hour

            JSONArray time = (JSONArray) hourly.get("time");

            int index = findIndexOfCurrentTime(time);

            // get temperature

            JSONArray temperatureData = (JSONArray) hourly.get("temperature\_2m");

            double temperature = (double) temperatureData.get(index);

            // get weather code

            JSONArray weathercode = (JSONArray) hourly.get("weathercode");

            String weatherCondition = convertWeatherCode((long) weathercode.get(index));

            // get humidity

            JSONArray relativeHumidity = (JSONArray) hourly.get("relativehumidity\_2m");

            long humidity = (long) relativeHumidity.get(index);

            // get windspeed

            JSONArray windspeedData = (JSONArray) hourly.get("windspeed\_10m");

            double windspeed = (double) windspeedData.get(index);

            // build the weather json data object that we are going to access in our frontend

            JSONObject weatherData = new JSONObject();

            weatherData.put("temperature", temperature);

            weatherData.put("weather\_condition", weatherCondition);

            weatherData.put("humidity", humidity);

            weatherData.put("windspeed", windspeed);

            return weatherData;

        }catch(Exception e){

            e.printStackTrace();

        }

        return null;

    }

    // retrieves geographic coordinates for given location name

    public static JSONArray getLocationData(String locationName){

        // replace any whitespace in location name to + to adhere to API's request format

        locationName = locationName.replaceAll(" ", "+");

        // build API url with location parameter

        String urlString = "https://geocoding-api.open-meteo.com/v1/search?name=" +

                locationName + "&count=10&language=en&format=json";

        try{

            // call api and get a response

            HttpURLConnection conn = fetchApiResponse(urlString);

            // check response status

            // 200 means successful connection

            if(conn.getResponseCode() != 200){

                System.out.println("Error: Could not connect to API");

                return null;

            }else{

                // store the API results

                StringBuilder resultJson = new StringBuilder();

                Scanner scanner = new Scanner(conn.getInputStream());

                // read and store the resulting json data into our string builder

                while(scanner.hasNext()){

                    resultJson.append(scanner.nextLine());

                }

                // close scanner

                scanner.close();

                // close url connection

                conn.disconnect();

                // parse the JSON string into a JSON obj

                JSONParser parser = new JSONParser();

                JSONObject resultsJsonObj = (JSONObject) parser.parse(String.valueOf(resultJson));

                // get the list of location data the API gtenerated from the lcoation name

                JSONArray locationData = (JSONArray) resultsJsonObj.get("results");

                return locationData;

            }

        }catch(Exception e){

            e.printStackTrace();

        }

        // couldn't find location

        return null;

    }

    private static HttpURLConnection fetchApiResponse(String urlString){

        try{

            // attempt to create connection

            URL url = new URL(urlString);

            HttpURLConnection conn = (HttpURLConnection) url.openConnection();

            // set request method to get

            conn.setRequestMethod("GET");

            // connect to our API

            conn.connect();

            return conn;

        }catch(IOException e){

            e.printStackTrace();

        }

        // could not make connection

        return null;

    }

    private static int findIndexOfCurrentTime(JSONArray timeList){

        String currentTime = getCurrentTime();

        // iterate through the time list and see which one matches our current time

        for(int i = 0; i < timeList.size(); i++){

            String time = (String) timeList.get(i);

            if(time.equalsIgnoreCase(currentTime)){

                // return the index

                return i;

            }

        }

        return 0;

    }

    private static String getCurrentTime(){

        // get current date and time

        LocalDateTime currentDateTime = LocalDateTime.now();

        // format date to be 2023-09-02T00:00 (this is how is is read in the API)

        DateTimeFormatter formatter = DateTimeFormatter.ofPattern("yyyy-MM-dd'T'HH':00'");

        // format and print the current date and time

        String formattedDateTime = currentDateTime.format(formatter);

        return formattedDateTime;

    }

    // convert the weather code to something more readable

    private static String convertWeatherCode(long weathercode){

        String weatherCondition = "";

        if(weathercode == 0L){

            // clear

            weatherCondition = "Clear";

        }else if(weathercode > 0L && weathercode <= 3L){

            // cloudy

            weatherCondition = "Cloudy";

        }else if((weathercode >= 51L && weathercode <= 67L)

                    || (weathercode >= 80L && weathercode <= 99L)){

            // rain

            weatherCondition = "Rain";

        }else if(weathercode >= 71L && weathercode <= 77L){

            // snow

            weatherCondition = "Snow";

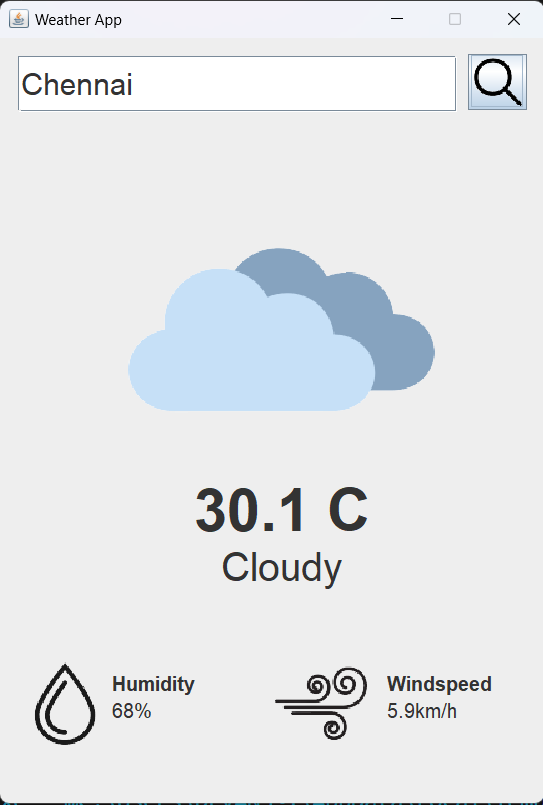
        }

        return weatherCondition;

    }

}

**6.1.1Result:**



**7.CONCLUSION**

The "Java Weather Application" represents a successful and versatile software solution designed to meet the everyday needs of users seeking accurate and up-to-date weather information. Throughout the development of this project, several key aspects were emphasized, resulting in a feature-rich, user-friendly, and reliable application that showcases the capabilities of Java as a development platform.

Key Achievements:

User-Centric Design: The project prioritized user experience by providing an intuitive and aesthetically pleasing graphical user interface. Users can effortlessly input their desired location, retrieve real-time weather data, and view it in a user-friendly format. This focus on user-centric design ensures that the application caters to users of all levels of technical expertise.

API Integration: The successful integration with a reputable weather API is a testament to the project's reliability and data accuracy. Users can trust that the weather information provided by the application is up-to-date and sourced from a trusted provider.

Error Handling: Robust error handling mechanisms were implemented to gracefully manage unexpected situations, such as incorrect user inputs, network connectivity issues, or API rate limiting. This proactive approach ensures that users are informed and guided even when problems arise.

Interactivity: The application's interactivity allows users to actively engage with it, inputting their location and initiating weather data retrieval at their convenience. This feature empowers users to make timely decisions based on real-time weather information.

Informative Display: Weather information is presented in a clear and comprehensive manner, including temperature, humidity, wind speed, and current conditions. Users can quickly grasp the essential weather parameters they need for their daily planning.

Enhancements: The project went beyond basic functionality by implementing additional features, such as multi-day forecasts and location auto-suggestions. These enhancements enrich the user experience and provide users with more in-depth insights into future weather conditions.

**8. REFERENCES**

API:

<https://openweathermap.org/current>

HTML:

<https://www.w3schools.com/html/>

<https://www.w3.org/Style/CSS/specs.en.html>

JAVA:

<https://www.w3schools.com/java/>

Library:

Apache HttpComponents

org.json.simple.JSONObject

<https://docs.oracle.com/en/java/index.html>

org.json.simple.JSONArray

<https://www.oracle.com/java/technologies/>

BOOKS REFFERED:  
[Head First Java](https://www.amazon.com/Head-First-Java-Brain-Learners/dp/0596004656/)

[Java: A Beginner’s Guide](https://www.amazon.com/Java-Beginners-Eighth-Herbert-Schildt/dp/1260440214/)

[Effective Java](https://www.amazon.com/Effective-Java-Joshua-Bloch/dp/0134685997)