WEATHER INFORMATION SYSTEM

An

Object-Oriented Programming through Java Course Project Report

in partial fulfillment of the degree

Bachelor of Technology
in
Computer Science & Engineering

By

DINGARI SRINIDHI (2103A52079)

EMMADI BHAVANI (2103A52080)

Under the guidance of

T. SAMPATH KUMAR

(Asst professor)

Submitted to

SCHOOL OF COMPUTER SCIENCE & ARTIFICIAL INTELLIGENCE

SR UNIVERSITY

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

CERTIFICATE

This is to certify that the Object Oriented Programming through Java - Course Project Report entitled "**WEATHER INFORMATION SYSTEM**" is a record of bonafide work carried out by the student **DINGARI SRINIDHI, EMMADI BHAVANI** bearing Roll No(s) **2103A52079, 2103A52080** during the academic year 2023-24 in partial fulfillment of the award of the degree of **Bachelor of Technology** in **Computer Science & Engineering** by the SR University , Ananthsagar ,Warangal.

Lab In-charge **Head of the Department**

Table of Contents

ABSTRACT

The Weather Information System (WIS) is a comprehensive and integrated platform designed to collect, process, analyze, and disseminate accurate and up-to-date weather information. In an era where climate variability and extreme weather events are becoming more prevalent, the need for a reliable and efficient weather information system is paramount. The WIS serves as a crucial tool for various sectors, including agriculture, transportation, disaster management, and everyday planning.

The system incorporates a network of advanced weather monitoring instruments such as satellites, radar systems, weather stations, and other sensors to capture real-time atmospheric data. These data sources feed into a centralized processing unit equipped with sophisticated algorithms for weather prediction and analysis. The WIS employs machine learning techniques to enhance the accuracy of short-term and long-term weather forecasts, taking into account factors such as historical data, topography, and global climate patterns.

The user interface of the Weather Information System is user-friendly, providing stakeholders with easy access to a wide range of weather parameters, including temperature, humidity, wind speed, precipitation, and atmospheric pressure. The system offers customizable alert mechanisms, ensuring timely dissemination of critical information to end-users. Decision-makers can utilize the WIS to make informed choices related to resource allocation, disaster preparedness, and risk mitigation.

Furthermore, the Weather Information System supports data visualization tools, allowing users to interpret complex weather patterns through interactive maps, graphs, and charts. This visual representation aids in better understanding and interpretation of weather trends, fostering proactive decision-making.

1. INTRODUCTION

The Weather Information System (WIS) stands at the forefront of modern technological solutions aimed at comprehensively understanding, monitoring, and predicting atmospheric conditions. In an era where the impacts of climate change are increasingly felt, having a robust and efficient system for gathering and disseminating weather information is essential for a wide array of applications, ranging from agriculture and transportation to disaster management and everyday decision-making.

The WIS is a sophisticated platform that harnesses the power of cutting-edge technology, including satellites, radar systems, weather stations, and advanced sensors, to capture real-time data from the Earth's atmosphere. This wealth of information is then processed through state-of-the-art algorithms, incorporating machine learning techniques, to provide accurate and reliable short-term and long-term weather forecasts. Weather affects various aspects of our lives, from the crops we cultivate to the clothes we wear, making timely and precise weather information a critical asset. The WIS not only serves as a data repository but also as a dynamic tool for analysis, interpretation, and visualization. Through user-friendly interfaces and intuitive data representation, the WIS empowers individuals, businesses, and government agencies to make informed decisions in the face of changing weather conditions.

This introduction aims to highlight the pivotal role of the Weather Information System in addressing the challenges posed by a dynamic climate. As we delve deeper into the system's components and functionalities, it becomes evident that the WIS is not merely a collection of weather data but a comprehensive solution fostering resilience, adaptability, and proactive decision-making in the face of evolving atmospheric conditions.

OBJECTIVES

Accurate Forecasting: Develop and implement advanced algorithms, including machine learning techniques, to enhance the accuracy of short-term and long-term weather forecasts. This objective aims to provide reliable and timely information for proactive decision-making.

Real-time Monitoring: Establish a network of weather monitoring instruments, such as satellites, radar systems, and weather stations, to capture real-time data on various atmospheric parameters. This objective ensures a continuous and up-to-date stream of information for analysis.

Data Integration: Facilitate the integration of diverse data sources, including meteorological, geographical, and historical data, to create a comprehensive and holistic understanding of weather patterns. This objective enhances the system's ability to provide contextually rich information.

User-Friendly Interface: Design an intuitive and user-friendly interface for stakeholders to access weather information easily. This objective ensures that end-users, including government agencies, businesses, and the general public, can interpret and utilize the data effectively.

DEFINITIONS OF THE ELEMENTS USED IN THE PROJECT

About Java:

Java is a general-purpose, class-based, object-oriented programming language designed for have lesser implementation dependencies. It is a computing platform for application development. Java is fast, secure, and reliable, therefore. It is widely used for developing Java applications in laptops, data centers, game consoles, scientific supercomputers, cell phones, etc.

Here are some important Java applications:

- It is used for developing Android Apps
- Helps you to create Enterprise Software
- Wide range of Mobile java Applications
- Scientific Computing Applications
- Use for Big Data Analytics
- Java Programming of Hardware devices

Java swings:

Swings in Java refers to the Java Swing framework, which is a set of graphical user interface (GUI) components for building desktop applications. Swing provides a rich set of tools for creating windows, buttons, menus, and other GUI elements, allowing developers to create interactive and visually appealing applications.

- Swing is a part of the Java Foundation Classes (JFC) and is included in the Java Standard Edition (SE) library.
- Swing components are platform-independent, which means that Swing-based applications look and behave consistently on different operating systems.
- Swing components are more flexible and customizable compared to the earlier Abstract Window Toolkit (AWT) components in Java.

- Swing supports a wide range of GUI components, including buttons, labels, text fields, tables, and more.
- Swing applications are typically event-driven, responding to user interactions like button clicks or mouse events.

Graphical User Interface(GUI):

Graphical User Interface (GUI) programming in Java typically involves using the Java Swing or JavaFX libraries to create user-friendly interfaces for your applications.

- Swing and JavaFX: These are the two main libraries for building GUI applications in Java. Swing is the older of the two, while JavaFX is more modern and provides richer features. JavaFX is often preferred for new projects.
- Components: GUI applications consist of various graphical components such as buttons, labels, text fields, and more. These components are provided by Swing or JavaFX libraries and can be customized to suit your application's needs.
- Layout Managers: Layout managers are used to arrange and manage the positioning of components within a GUI. Common layout managers include BorderLayout, FlowLayout, and GridLayout.
- Event Handling: In GUI applications, you need to handle user interactions, such as button clicks or mouse events. This is done by registering event listeners and implementing event handling code.

APPLICATION PROGRAMMING INTERFACE(API):

An API, or Application Programming Interface, is a set of rules and protocols that allows different software applications to communicate with each other. It defines the methods and data formats that applications can use to request and exchange information.

GEOGRAPHICAL INFORMATION SYSTEM(GIS):

A Geographical Information System (GIS) is a system designed to capture, store, manipulate, analyze, manage, and present spatial or geographic data. GIS applications are tools that allow users to create interactive queries, analyze spatial information, edit data, maps, and present the results of all these operations.

IMPLEMENTATION:

APP LAUNCHER :

```
package weatherforecast;
import javax.swing.SwingUtilities;
public class AppLauncher {
    public static void main(String[] args){
        SwingUtilities.invokeLater(new Runnable(){
            @Override
            public void run(){
                new WeatherAppGui().setVisible(true);
            }
        });

    }
}
```

WEATHER APP:

```
package weatherforecast;

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&time
zone=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;
    }
}
```

WEATHER APP GUI:

```
package weatherforecast;

import org.json.simple.JSONObject;

import javax.imageio.ImageIO;
import javax.swing.*;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.image.BufferedImage;
import java.io.File;
import java.io.IOException;

public class WeatherAppGui extends JFrame {
    private JSONObject weatherData;

    public WeatherAppGui(){
```

```
        // setup our gui and add a title
        super("Weather App");

        // configure gui to end the program's process once it has been closed
        setDefaultCloseOperation(EXIT_ON_CLOSE);

        // set the size of our gui (in pixels)
        setSize(450, 650);

        // load our gui at the center of the screen
        setLocationRelativeTo(null);

        // make our layout manager null to manually position our components within the
gui
        setLayout(null);

        // prevent any resize of our gui
        setResizable(false);

        addGuiComponents();
    }

    private void addGuiComponents(){
        // search field
        JTextField searchTextField = new JTextField();

        // set the location and size of our component
        searchTextField.setBounds(15, 15, 351, 45);

        // change the font style and size
        searchTextField.setFont(new Font("Dialog", Font.PLAIN, 24));

        add(searchTextField);
```

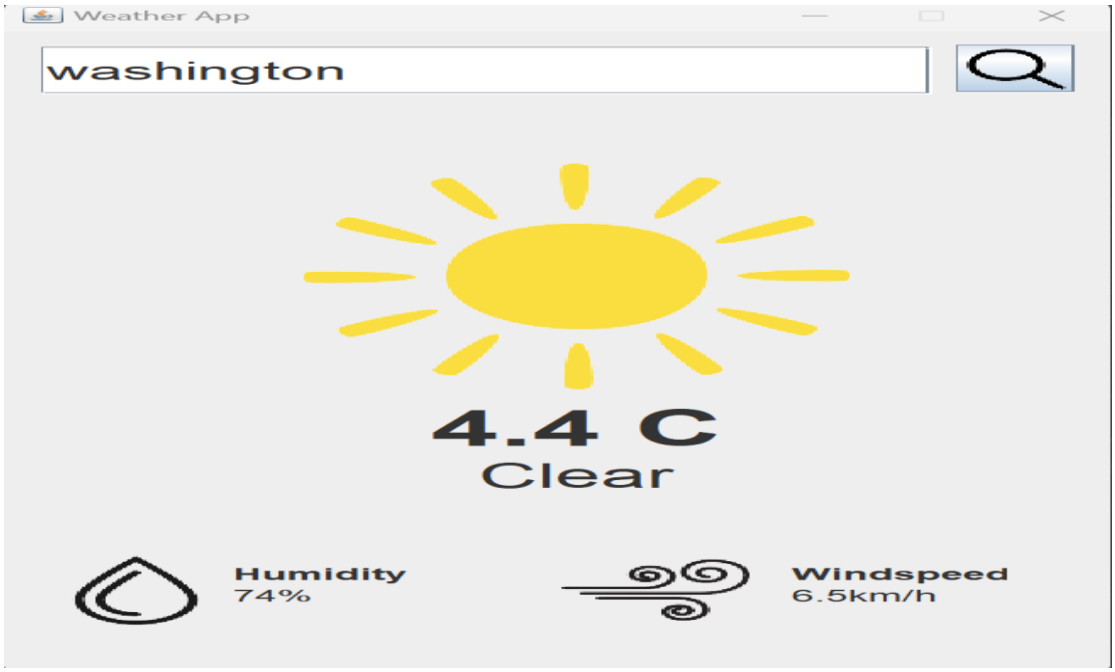
```
        // weather image
        JLabel weatherConditionImage = new
JLabel(loadImage("src/assets/cloudy.png"));
        weatherConditionImage.setBounds(0, 125, 450, 217);
        add(weatherConditionImage);

        // temperature text
        JLabel temperatureText = new JLabel("10 C");
        temperatureText.setBounds(0, 350, 450, 54);
        temperatureText.setFont(new Font("Dialog", Font.BOLD, 48));

        // center the text
        temperatureText.setHorizontalAlignment(SwingConstants.CENTER);
        add(temperatureText);

        // weather condition description
        JLabel weatherConditionDesc = new JLabel("Cloudy");
        weatherConditionDesc.setBounds(0, 405, 450, 36);
        weatherConditionDesc.setFont(new Font("Dialog", Font.PLAIN, 32));
        weatherConditionDesc.setHorizontalAlignment(SwingConstants.CENTER);
        add(weatherConditionDesc);

        // humidity image
        JLabel humidityImage = new JLabel(loadImage("src/assets/humidity.png"));
        humidityImage.setBounds(15, 500, 74, 66);
        add(humidityImage);

        // humidity text
        JLabel humidityText = new JLabel("<html><b>Humidity</b> 100%</html>");
        humidityText.setBounds(90, 500, 85, 55);
        humidityText.setFont(new Font("Dialog", Font.PLAIN, 16));
        add(humidityText);

        // windspeed image
```

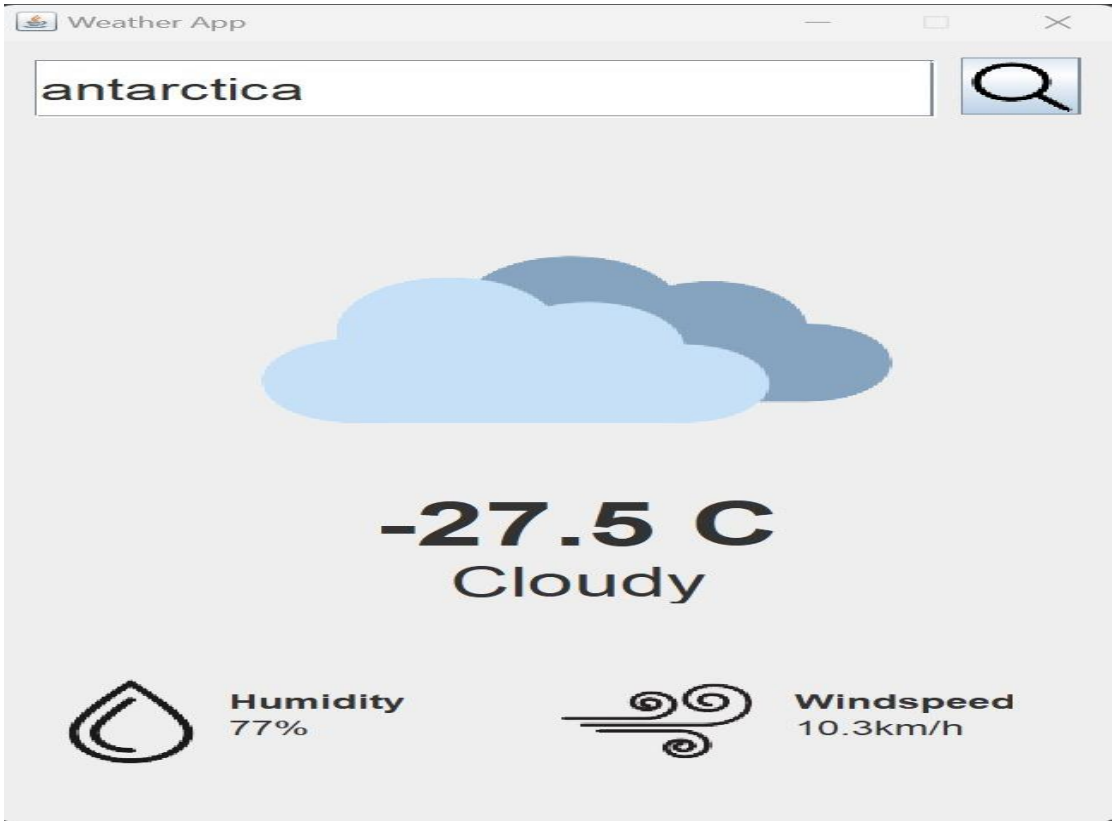
```
        JLabel windspeedImage = new JLabel(loadImage("src/assets/windspeed.png"));
        windspeedImage.setBounds(220, 500, 74, 66);
        add(windspeedImage);

        // windspeed text
        JLabel windspeedText = new JLabel("<html><b>Windspeed</b>
15km/h</html>");
        windspeedText.setBounds(310, 500, 85, 55);
        windspeedText.setFont(new Font("Dialog", Font.PLAIN, 16));
        add(windspeedText);

        // search button
        JButton searchButton = new JButton(loadImage("src/assets/search.png"));

        // change the cursor to a hand cursor when hovering over this button
        searchButton.setCursor(Cursor.getPredefinedCursor(Cursor.HAND_CURSOR));
        searchButton.setBounds(375, 13, 47, 45);
        searchButton.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                // get location from user
                String userInput = searchTextField.getText();

                // validate input - remove whitespace to ensure non-empty text
                if(userInput.replaceAll("\\s", "").length() <= 0){
                    return;
                }

                // retrieve weather data
                weatherData = WeatherApp.getWeatherData(userInput);

                // update gui

                // update weather image
```

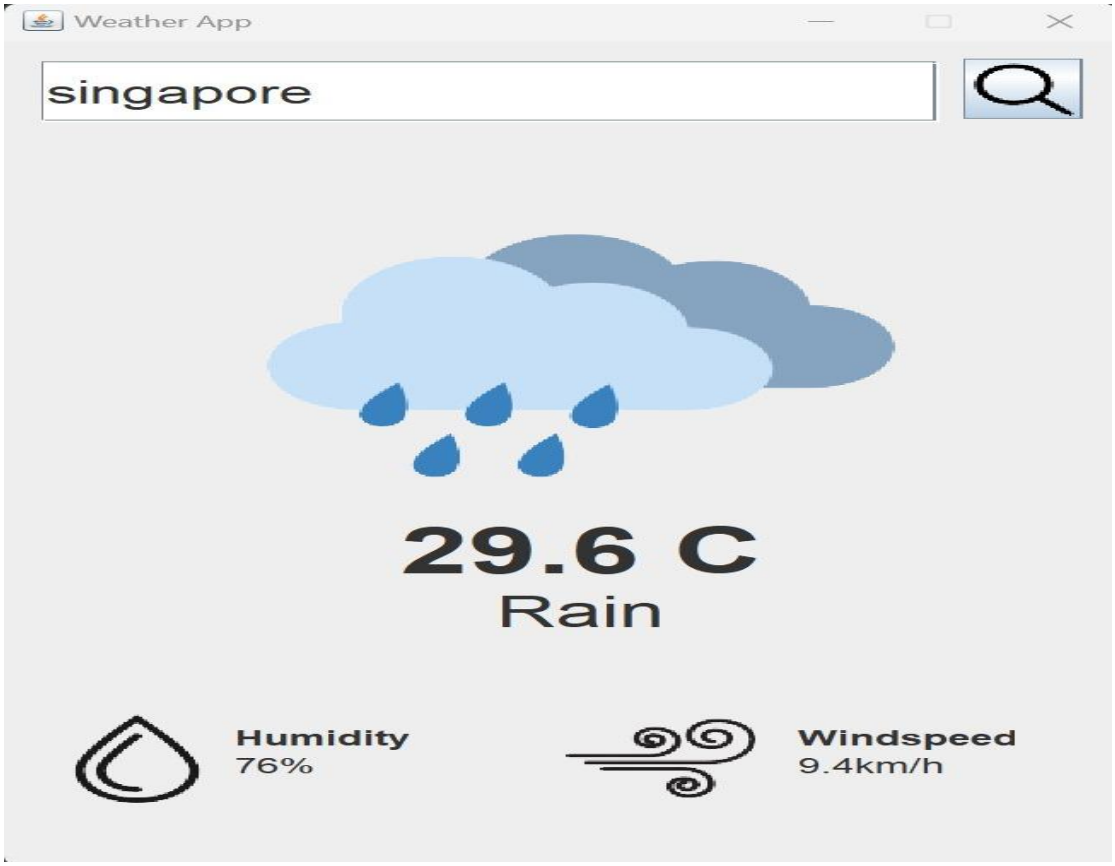
```
            String weatherCondition = (String) weatherData.get("weather_condition");

            // depending on the condition, we will update the weather image that
corresponds with the condition
            switch(weatherCondition){
               case "Clear":
                  weatherConditionImage.setIcon(loadImage("src/assets/clear.png"));
                  break;
               case "Cloudy":
                  weatherConditionImage.setIcon(loadImage("src/assets/cloudy.png"));
                  break;
               case "Rain":
                  weatherConditionImage.setIcon(loadImage("src/assets/rain.png"));
                  break;
               case "Snow":

weatherConditionImage.setIcon(loadImage("src/assets/snow.pngImage"));
                  break;
            }

            // update temperature text
            double temperature = (double) weatherData.get("temperature");
            temperatureText.setText(temperature + " C");

            // update weather condition text
            weatherConditionDesc.setText(weatherCondition);

            // update humidity text
            long humidity = (long) weatherData.get("humidity");
            humidityText.setText("<html><b>Humidity</b> " + humidity +
"%</html>");

            // update windspeed text
            double windspeed = (double) weatherData.get("windspeed");
```

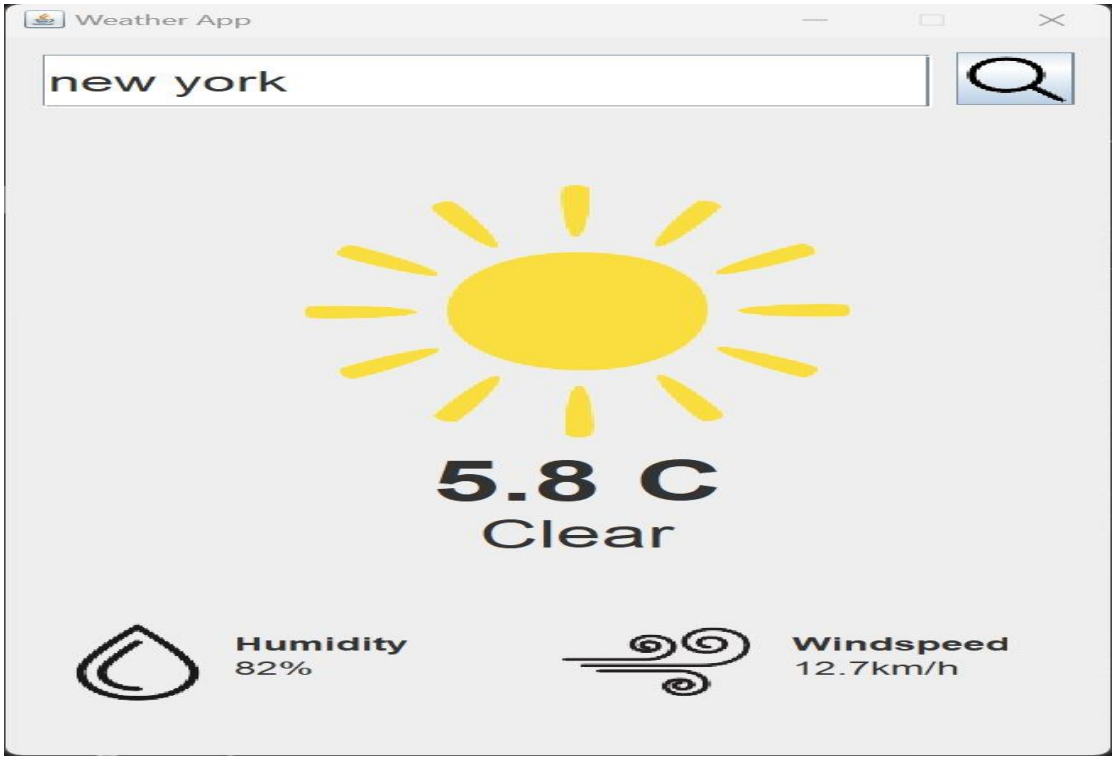
```
                windspeedText.setText("<html><b>Windspeed</b> " + windspeed +
"km/h</html>");
            }
        });
        add(searchButton);
    }

    // used to create images in our gui components
    private ImageIcon loadImage(String resourcePath){
        try{
            // read the image file from the path given
            BufferedImage image = ImageIO.read(new File(resourcePath));

            // returns an image icon so that our component can render it
            return new ImageIcon(image);
        }catch(IOException e){
            e.printStackTrace();
        }

        System.out.println("Could not find resource");
        return null;
    }
}
```

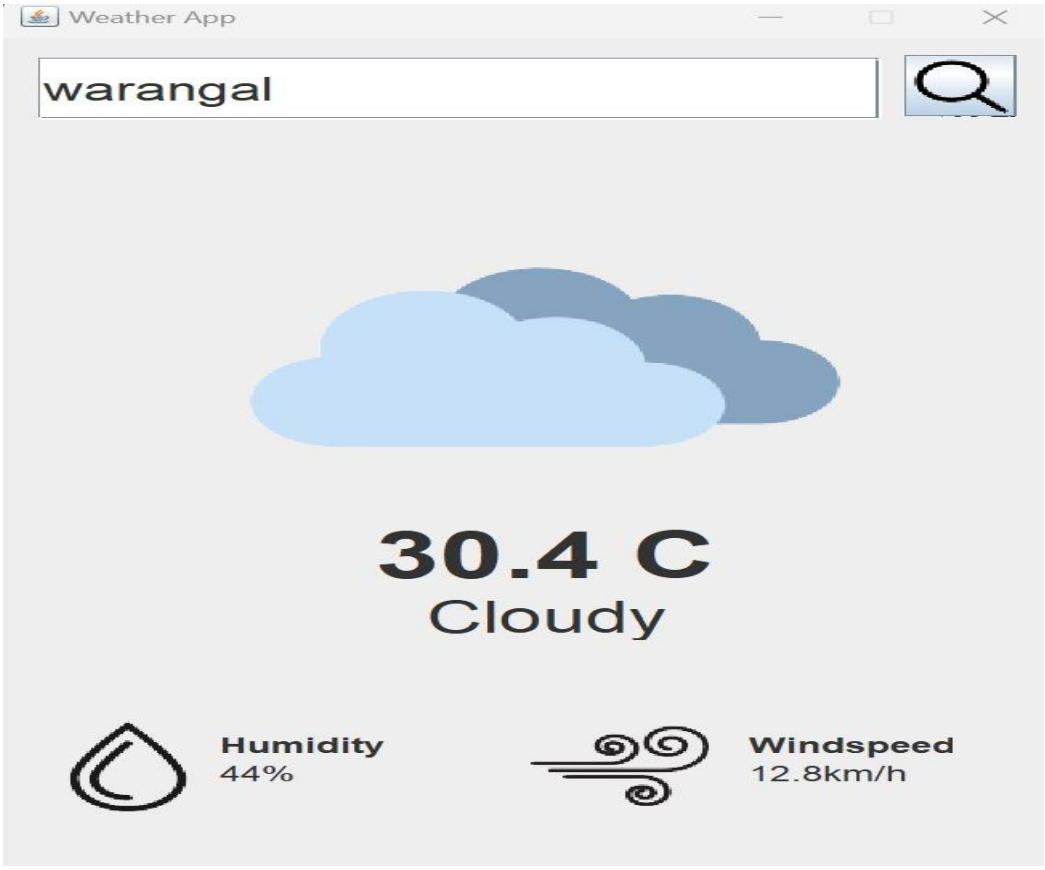
RESULTS :


Weather App
washington
4.4 C
Clear
Humidity
74%
Windspeed
6.5km/h


Weather App
antarctica
-27.5 C
Cloudy
Humidity
77%
Windspeed
10.3km/h


Weather App
singapore
29.6 C
Rain
Humidity
76%
Windspeed
9.4km/h


Weather App
new york
5.8 C
Clear
Humidity
82%
Windspeed
12.7km/h


Weather App
warangal
30.4 C
Cloudy
Humidity
44%
Windspeed
12.8km/h

CONCLUSION

A weather information system in Java is a useful tool for obtaining weather forecasts and other meteorological data. One such project is the Weather Report project [1]It is a web-based application that provides weather reports for any location. The application uses a geographical locator to identify the user's location and present weather details such as temperature, wind direction, humidity, and more. The user can change the location by selecting the options provided below to get its details. The application also has new avatars and feed burner that allow users to get weather reports directly to their mail when they are not able to access the domain or even when the server is down. Its weather watch gadgets in animated form will notify about weather for a particular date and time. It will also focus on critical weather conditions for a particular gadget through this gadget [1]

FUTURE SCOPE

A real-time weather monitoring system can be developed using Java. The system can use sensors, deep learning technologies, and information processing to provide real-time weather monitoring in buses and stations and achieve weather forecasts through predictive models

The user interface of the weather information system can be improved to make it more user-friendly. The system can be designed to provide weather reports in a more interactive and engaging way. For example, the system can use animations and graphics to display weather information