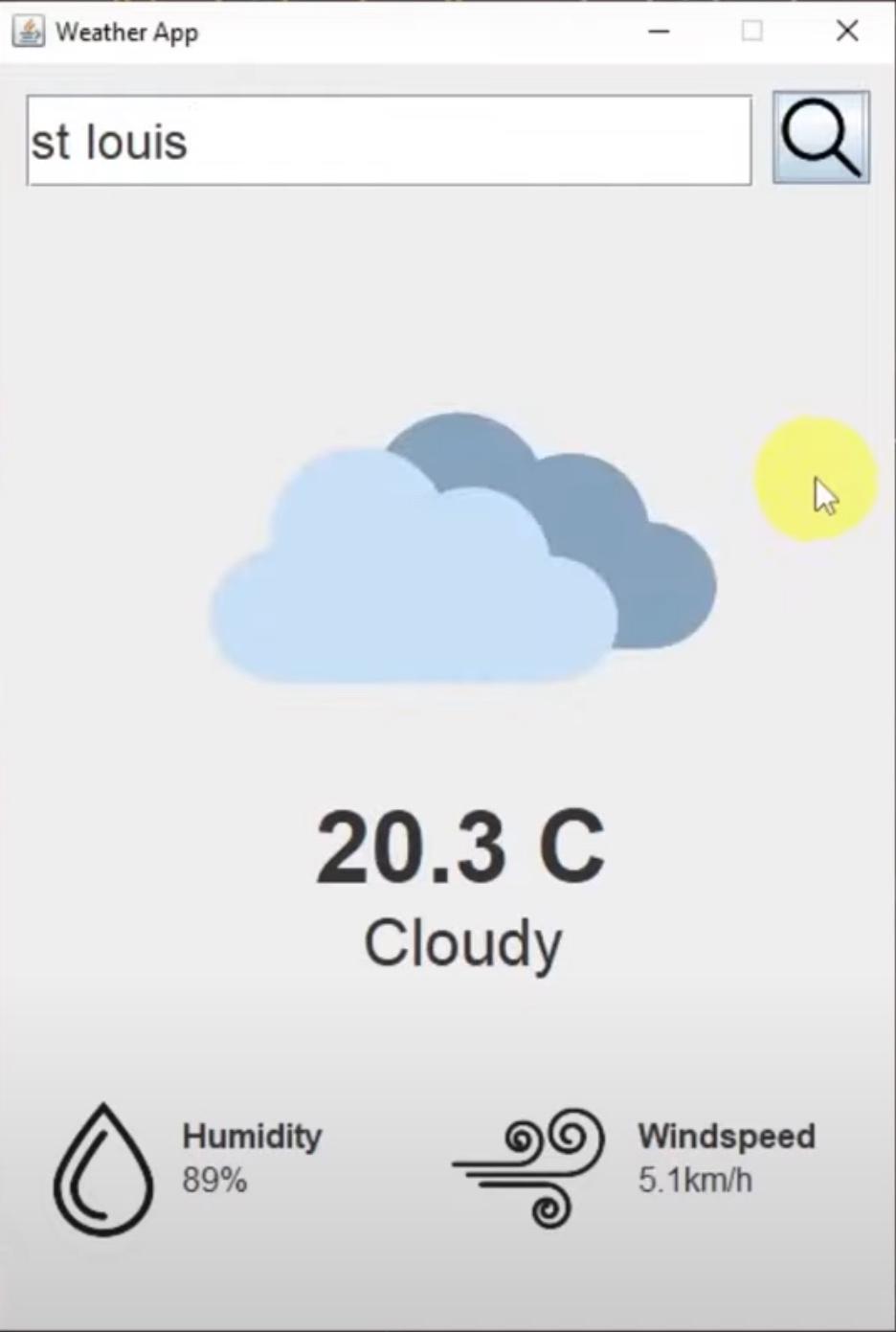
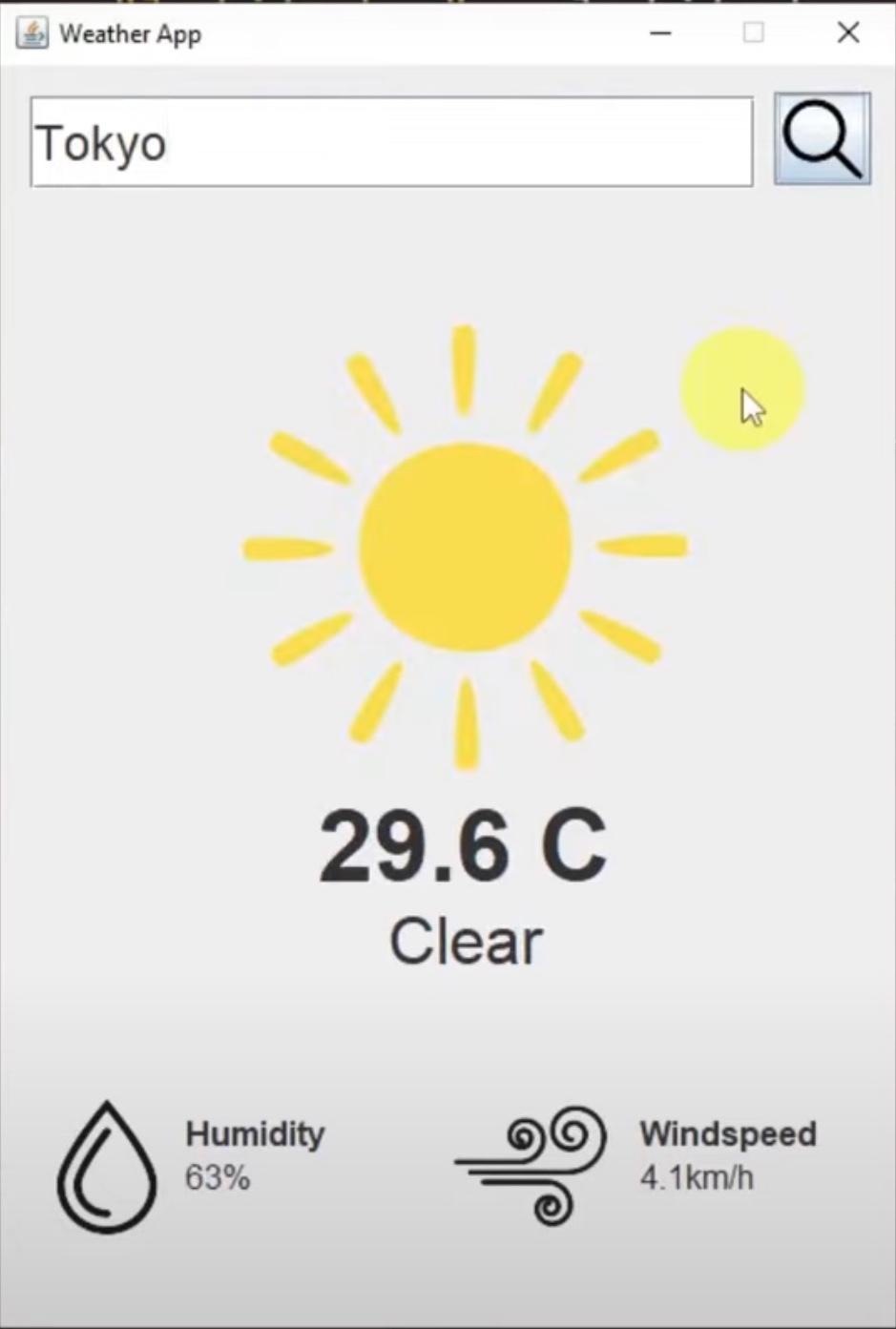
## **Program Screenshot**



## **Program Code**

### **AppLauncher.java**

import javax.swing.\*;

public class AppLauncher {

public static void main(String[] args) {

SwingUtilities.invokeLater(new Runnable(){

@Override

public void run(){

new WeatherAppGui().setVisible(true);

}

});

}

}

### **WeatherApp.java**

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;

}

}

### **WeatherAppGui.java**

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;

}

}

## **Program Documentation**

The Weather App is a Java-based application that provides users with real-time weather information for a specified location. It fetches weather data from an external API and displays it in a graphical user interface (GUI). Users can enter a location, and the app retrieves and presents weather details, including temperature, weather condition, humidity, and wind speed. This documentation outlines the project's architecture, technologies used, and the functionality of each class within the application.

#### **Technology used**

The Weather App utilizes the following technologies and libraries:

* Java 18
* JSON Simple - Used to parse and read through JSON data
* HTTPURLConnection - Java's built-in library for making HTTP requests to fetch data from external APIs.

### **File Summaries**

#### **AppLauncher.java**

This file serves as the entry point for the Weather App. It initializes the GUI and displays the main application window.

#### **WeatherAppGui.java**

This file represents the graphical user interface (GUI) of the Weather App. It is responsible for displaying weather information for a specified location.

It also handles the layout and display of GUI components, including text fields, labels, buttons, and images. It also implements the user interface for entering a location and updating the weather information based on user input.

#### **WeatherApp.java**

The file contains the backend logic for fetching weather data from an external API. It retrieves geographic coordinates for a location, fetches weather data for that location, and provides methods to convert weather codes.

It encapsulates the core functionality of the Weather App. It includes methods to fetch weather data and location coordinates, convert weather codes into readable weather conditions, and manage API requests. It also acts as the bridge between the GUI and the external weather data source, ensuring that weather information is retrieved and displayed accurately.