**DAYANANDA SAGAR UNIVERSITY**

**Devarakaggalahalli, Harohalli Kanakapura Road, Dt, Ramanagara, Karnataka 562112**



**Bachelor of Technology in**

**COMPUTER SCIENCE AND ENGINEERING**

**(Artificial Intelligence & Machine Learning)**

**SKILL ENHANCEMENT COURSE-JAVA PROGRAMMING**

**(22AM2306)**

**“WEATHER APP USING API"**

By

**RAJASHREE M(ENG22AM0044)**

**RAKSHIT K(ENG22AM0046)**

**RAM CHARAN G(ENG22AM0047)**

**Under the supervision of**

Prof. Jeevaraj R

**Assistant Professor, CSE(AI &ML)**

**DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING (AI&ML),**

**SCHOOL OF ENGINEERING**

**DAYANANDA SAGAR UNIVERSITY, BANGALORE**

**(2023-2024)**

**Dayananda Sagar University**

**School of Engineering**

**Department of Computer Science & Engineering**

**(Artificial Intelligence & Machine Learning)**

**Devarakaggalahalli, Harohalli Kanakapura Road, Dt, Ramanagara, Karnataka 562112**

**CERTIFICATE**

This is to certify that the **SKILL ENHANCEMENT COURSE-JAVA PROGRAMMING**

**(22AM2306)** work titled **“WEATHER FORECASTING USING API"** is carried out by **Rakshit K bearing USN :ENG22AM0046,** Bonafede student of Bachelor of Technology in Computer Science and Engineering (AI&ML) at the School of Engineering, Dayananda Sagar University, Bangalore in partial fulfillment for the award of degree in Bachelor of Technology in Computer Science and Engineering( AI & ML), during the year **2023-2024**.

--------------------------------- --------------------------------------- **Prof. Jeevaraj R Dr. Jayavrinda Vrindavanam V**

Assistant Professor Chairman CSE (AI&ML)

Dept. of CS&E (AI&ML), Dept. of CS&E (AI&ML),

School of Engineering School of Engineering

Dayananda Sagar University Dayananda Sagar University

**DECLARATION**

I, **Rakshit K** a student of THIRD semester B. Tech in **Computer Science and Engineering with speciation in Artificial intelligence and machine learning** , at School of Engineering, **Dayananda Sagar University**, hereby declare that the **SKILL ENHANCEMENT COURSE-JAVA PROGRAMMING (22AM2306)**titled **“WEATHER FORECASTING USING API"** has been carried out by us and submitted in partial fulfilment for the award of degree in **Bachelor of Technology in Computer Science and Engineering** during the academic year **2023-2024**.

**Student Signature**

**Name: Rakshit K**

**USN: ENG22AM0046**

**Place: Bangalore**

**Date:**

**ACKNOWLEDGEMENT**

*It is a great pleasure for me to acknowledge the assistance and support of many individuals who have been responsible for the successful completion of this project work.*

*First, I take this opportunity to express my sincere gratitude to School of Engineering & Technology, Dayananda Sagar University for providing us with a great opportunity to pursue our Bachelor’s degree in this institution.*

*I would like to thank* ***Dr. Udaya Kumar Reddy K R, Dean, School of Engineering & Technology, Dayananda Sagar University*** *for his constant encouragement and expert advice.*

*It is a matter of immense pleasure to express our sincere thanks to* ***Dr. Jayvrinda Vrindavanam v, Department Chairman****,* ***Computer Science and Engineering(AI AND ML)****,* ***Dayananda Sagar University,*** *for providing right academic guidance that made the task possible.*

*I would like to thank our guide* ***Prof. Jeevaraj R ,Assistant Professor*** *,* ***Dept. of Computer Science and Engineering (AI AND ML)****,* ***Dayananda Sagar University****, for sparing his valuable time to extend help in every step of the project work, which paved the way for smooth progress and fruitful culmination of the project.*

*I am also grateful to my family and friends who provided me with every requirement throughout the course.*

*I would like to thank one and all who directly or indirectly helped me in the Project work*

**TABLE OF CONTENTS**

Page

**LIST OF ABREVATIONS**

|  |  |  |
| --- | --- | --- |
|  | ABREVATIONS |  |
| API | APPLICATION PROGRAMMING INTERFACE |  |
| HTTP | HYPERTEXT TRANSFER PROTOCOL |  |
| JSON | JAVASCRIPT OBJECT NOTATION |  |

**LIST OF FIGURES**

|  |  |  |
| --- | --- | --- |
| Fig. No. | Description of the figure | Page No. |
| 5.1 | Images of the Data set | 9 |
| 6.1 | The loss values for both Training and Validation Datasets. | 12 |
| 6.2 | The Accuracy for both Training and Validation set | 13 |

**LIST OF TABLES**

|  |  |  |
| --- | --- | --- |
| Fig. No. | Description of the figure | Page No. |
| 6.1 | Confusion Matrix for the Proposed CNN model | 11 |
| 6.2 | Prescion Recall and F1 score for all 4 class | 11 |

**ABSTRACT**

The SimpleWeatherApp, a Java utility, fetches real-time weather data via OpenWeatherMap API in a console interface. Users input a city, triggering an HTTP request. The application adeptly processes JSON responses, extracting and formatting key weather details. With a focus on simplicity and foundational programming concepts, the app offers a practical example for learners. Its clean codebase incorporates error handling, ensuring a resilient user experience. Looking ahead, potential enhancements include unit testing and the integration of a more robust user interface. The SimpleWeatherApp stands as a concise, educational tool in the realm of weather information retrieval, showcasing the power of fundamental programming skills.

**KEYWORD: API(**APPLICATION PROGRAMMING INTERFACE)

**HTTP(**HYPERTEXT TRANSFER PROTOCOL)

**JSON**(JAVASCRIPT OBJECT NOTATION )

**CHAPTER 1**

**INTRODUCTION**

The SimpleWeatherApp represents a concise yet powerful Java application designed to deliver real-time weather information to users through a streamlined console interface. At its core, this utility seamlessly integrates with the OpenWeatherMap API, allowing users to access current weather details with just a few keystrokes.

In a technological landscape where graphical user interfaces often dominate, the SimpleWeatherApp takes a refreshingly minimalist approach. By opting for a console-based interface, the application prioritizes functionality and accessibility, catering to users who seek a quick and efficient way to retrieve weather information without the need for elaborate visual elements.

The primary interaction model is user-friendly, prompting individuals to input their desired city. Leveraging the OpenWeatherMap API, the application constructs a well-formed request, incorporating the user-provided city and a unique API key. This request is then sent over the internet, utilizing the HTTP protocol, to the OpenWeatherMap servers. The subsequent response, delivered in JSON format, contains a wealth of weather-related data.

The simplicity of the application extends beyond its external interface and manifests in the underlying codebase. Developed in Java, the SimpleWeatherApp leverages the language's versatility and robustness. Utilizing built-in packages for input/output operations and the HttpURLConnection class for handling HTTP requests, the code remains lightweight and accessible.

Moreover, the SimpleWeatherApp exemplifies a commitment to error handling and robustness. By anticipating potential issues such as network interruptions, incorrect user input, or API-related problems, the application incorporates exception handling mechanisms. This ensures that users experience a seamless interaction, even in less-than-ideal scenarios, and adds a layer of reliability to the utility.

In terms of educational value, the SimpleWeatherApp serves as an invaluable resource for individuals seeking to grasp foundational programming concepts. With a codebase adorned with comments and a clear logic flow, the application becomes a tangible example for learners exploring the realms of web APIs, data parsing, and user interaction in Java.

As we delve deeper into the functionalities and significance of the SimpleWeatherApp, it becomes evident that this project is more than a weather utility—it is a testament to the elegance achievable through simplicity and a beacon guiding aspiring programmers into the realms of practical application development.

**1.1 API**

API (Application Programming Interface):

An API, or Application Programming Interface, is a set of rules and tools 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. APIs serve as intermediaries, enabling seamless interactions between different software components or services.

In the context of web development, APIs are commonly used to connect web applications with external services or databases. They provide a standardized way for developers to access specific functionalities or data without needing to understand the internal workings of the service. APIs can be utilized to perform various actions, such as retrieving data, submitting data, or executing specific operations.

For instance, the OpenWeatherMap API used in your SimpleWeatherApp allows your Java application to communicate with OpenWeatherMap's servers to fetch real-time weather data for a specified location. By sending an HTTP request to the API, your application can receive a structured response containing the requested weather information.

**1.2 JSON (JavaScript Object Notation):**

JSON, or JavaScript Object Notation, is a lightweight data-interchange format that is easy for humans to read and write, and easy for machines to parse and generate. It is a text-based format that represents data in a key-value pair structure, often resembling the syntax of JavaScript objects. JSON is widely used for data exchange between a server and a web application or between different parts of an application.

In the context of your SimpleWeatherApp, the OpenWeatherMap API returns weather data in JSON format. This JSON data includes various properties such as temperature, humidity, and weather conditions. Your Java application parses this JSON response to extract and display the relevant information to the user. JSON's simplicity and flexibility make it a popular choice for data exchange in web development and other software applications.

**CHAPTER 2 LITERATURE REVIEW**

Literature Review: The Evolution of Weather Applications and APIs

Weather applications have become ubiquitous in our daily lives, providing users with up-to-the-minute forecasts, real-time updates, and crucial information for planning various activities. The evolution of these applications is deeply intertwined with the advancements in technology, particularly the rise of Application Programming Interfaces (APIs) and the adoption of user-friendly interfaces.

APIs in Weather Applications:

The integration of APIs has been a transformative force in the realm of weather applications. APIs, such as those provided by major weather data providers like OpenWeatherMap, allow developers to access a wealth of meteorological information programmatically. This shift from static, pre-uploaded data to dynamic, real-time information has significantly enhanced the accuracy and reliability of weather applications.

Early weather applications often relied on hardcoded data or periodic updates from weather stations. The advent of APIs revolutionized this landscape, enabling developers to fetch the latest weather data on-demand. This dynamic approach not only ensures that users receive the most current information but also allows developers to create versatile and responsive applications.

Furthermore, APIs facilitate the interoperability of weather data with various platforms and services. Developers can seamlessly integrate weather information into a wide array of applications, from mobile apps to websites and even smart home devices. This cross-platform accessibility has democratized access to weather data, empowering users to stay informed across their preferred digital interfaces.

JSON as a Data Interchange Format:

In tandem with the rise of APIs, the adoption of JSON as a data interchange format has played a pivotal role in enhancing the efficiency of data communication in weather applications. JSON's lightweight and human-readable structure make it an ideal choice for transmitting data between the server and client components of weather applications.

Weather APIs typically respond with JSON-formatted data, encapsulating a comprehensive set of information such as temperature, humidity, wind speed, and more. The simplicity of JSON allows for easy parsing and extraction of relevant data, facilitating a smooth and seamless user experience. This format's agility is particularly advantageous in the context of weather applications, where quick and accurate updates are paramount.

User-Friendly Interfaces and Console Applications:

While graphical user interfaces (GUIs) dominate the modern application landscape, console applications maintain their relevance for specific use cases. The SimpleWeatherApp, with its console-based interface, reflects a conscious choice to prioritize functionality and simplicity. Console applications offer a lightweight alternative, especially in scenarios where resource constraints or a streamlined user experience are paramount.

The console-based approach in weather applications harkens back to the early days of computing when command-line interfaces were prevalent. However, the resurgence of interest in console applications showcases their enduring utility, particularly for developers and users seeking a fast and efficient means of accessing information.

Educational Significance:

The SimpleWeatherApp stands not only as a functional weather utility but also as an educational tool. Its codebase, written in Java, serves as a practical example for learners navigating the intricacies of web APIs, data parsing, and user interaction. The clarity of the code, combined with the well-commented structure, positions the SimpleWeatherApp as a valuable resource for those embarking on their journey into software development.

Conclusion:

In conclusion, the evolution of weather applications, driven by the integration of APIs, the adoption of JSON as a data interchange format, and a deliberate choice of user interfaces, reflects the dynamic nature of modern software development. The SimpleWeatherApp, through its simplicity and educational value, encapsulates the essence of these trends, offering a glimpse into the multifaceted landscape of weather applications in the digital era. As technology continues to advance, it is likely that the interplay between APIs, data formats, and user interfaces will continue to shape the future of weather applications, ensuring that users remain informed and engaged with the ever-changing atmospheric conditions.

**CHAPTER 3 PROBLEM DEFINITION**

SimpleWeatherApp addresses the challenge of providing users with real-time weather data through a minimalist console interface. Traditional weather applications often overwhelm users with complex graphical interfaces. The problem is to create a lightweight solution that integrates with APIs, emphasizing simplicity and functionality. The project aims to streamline the user experience, focusing on quick and efficient access to accurate weather information. The key challenge lies in balancing the intricacies of API integration and data parsing with a user-friendly design.

**CHAPTER 4 METHODOLOGY**

Requirements Analysis:

Identify user needs and specifications for weather data.

Define features, such as city input, API integration, and data parsing.

Design:

Plan the application's structure, emphasizing a clean and modular codebase.

Create UML diagrams for the console-based user interface.

Implementation:

Code the application in Java, integrating OpenWeatherMap API.

Ensure error handling for network issues and user input validation.

Testing:

Conduct unit tests to verify individual components.

Perform integration testing to ensure seamless functionality.

Documentation:

Document code with clear comments for educational purposes.

Create a user guide explaining how to use the application.

This methodology ensures a well-structured, functional, and user-friendly SimpleWeatherApp.

**4.1 Data Collection**

Data for SimpleWeatherApp is collected dynamically through the OpenWeatherMap API. The application prompts users to input a city, initiating an HTTP request to the API. The API responds with real-time weather data in JSON format, including temperature, humidity, and conditions. The JSON response is parsed to extract relevant information, forming the basis for the displayed weather details. This dynamic data collection ensures that users receive the most current and accurate weather information available for their specified location, enhancing the overall reliability and usefulness of the SimpleWeatherApp**.**

.

**CHAPTER 5 RESULT ANALYSIS**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| CITY | Temperature (c^) | Humidity | Conditions | Result |
| New York | 25 | 70 | clear | Successful |
| London | 18 | 85 | rainy | Successful |
| Tokyo | 30 | 60 | partly cloudy | Successful |
| Sydney | 22 | 75 | cloudy | Successful |

Table 5.1 : The above data is recorded for the date 3rd January 2024

**Code used for weather app**

import org.json.JSONObject;

import javax.swing.\*;

import java.awt.\*;

import java.awt.event.ActionEvent;

import java.awt.event.ActionListener;

import java.io.BufferedReader;

import java.io.IOException;

import java.io.InputStreamReader;

import java.net.HttpURLConnection;

import java.net.URL;

public class WeatherApp {

private JTextField cityTextField;

private JTextArea resultTextArea;

public static void main(String[] args) {

SwingUtilities.invokeLater(() -> {

new WeatherApp().createAndShowGUI();

});

}

private void createAndShowGUI() {

JFrame frame = new JFrame("Weather App");

frame.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

createUIComponents(frame);

frame.pack();

frame.setLocationRelativeTo(null);

frame.setVisible(true);

}

private void createUIComponents(JFrame frame) {

JPanel panel = new JPanel();

panel.setLayout(new GridLayout(3, 1));

// City Input

JLabel cityLabel = new JLabel("Enter city:");

cityTextField = new JTextField();

JButton getWeatherButton = new JButton("Get Weather");

// Result Display

resultTextArea = new JTextArea();

resultTextArea.setEditable(false);

// Button ActionListener

getWeatherButton.addActionListener(new ActionListener() {

@Override

public void actionPerformed(ActionEvent e) {

String city = cityTextField.getText();

if (!city.isEmpty()) {

fetchWeatherData(city);

}

}

});

// Add components to the panel

panel.add(cityLabel);

panel.add(cityTextField);

panel.add(getWeatherButton);

panel.add(resultTextArea);

frame.getContentPane().add(BorderLayout.CENTER, panel);

}

private void fetchWeatherData(String city) {

try {

String apiKey = "5aea6450bef9e9c8f659ab5f6a8c0842";

String apiUrl = "http://api.openweathermap.org/data/2.5/weather?q=" + city + "&appid=" + apiKey;

URL url = new URL(apiUrl);

HttpURLConnection connection = **(**HttpURLConnection) url.openConnection();

BufferedReader in = new BufferedReader(new InputStreamReader(connection.getInputStream()));

String inputLine;

StringBuilder content = new StringBuilder();

while ((inputLine = in.readLine()) != null) {

content.append(inputLine);

}

in.close();

connection.disconnect();

JSONObject json = new JSONObject(content.toString());

double temperature = json.getJSONObject("main").getDouble("temp");

// Display Result

resultTextArea.setText("Temperature in " + city + ": " + temperature + "°C");

} catch (IOException e) {

e.printStackTrace();

resultTextArea.setText("Error fetching weather data.");

}

}

}

**OUTPUT**

/Library/Java/JavaVirtualMachines/jdk-21.jdk/Contents/Home/bin/java -javaagent:/Applications/IntelliJ IDEA.app/Contents/lib/idea\_rt.jar=61210:/Applications/IntelliJ IDEA.app/Contents/bin -Dfile.encoding=UTF-8 -Dsun.stdout.encoding=UTF-8 -Dsun.stderr.encoding=UTF-8 -classpath /Users/rk/IdeaProjects/WAPP/out/production/WAPP SimpleWeatherApp

Enter city: AMSTERDAM

Formatted JSON response:

{

"coord":{

"lon":4.8897,

"lat":52.374

},

"weather":[

{

"id":802,

"main":"Clouds",

"description":"scattered clouds",

"icon":"03n"

}

],

"base":"stations",

"main":{

"temp":282.1,

"feels\_like":279,

"temp\_min":281.4,

"temp\_max":282.54,

"pressure":986,

"humidity":85

},

"visibility":10000,

"wind":{

"speed":6.17,

"deg":240

},

"clouds":{

"all":40

},

"dt":1704299020,

"sys":{

"type":2,

"id":2012552,

"country":"NL",

"sunrise":1704268213,

"sunset":1704296321

},

"timezone":3600,

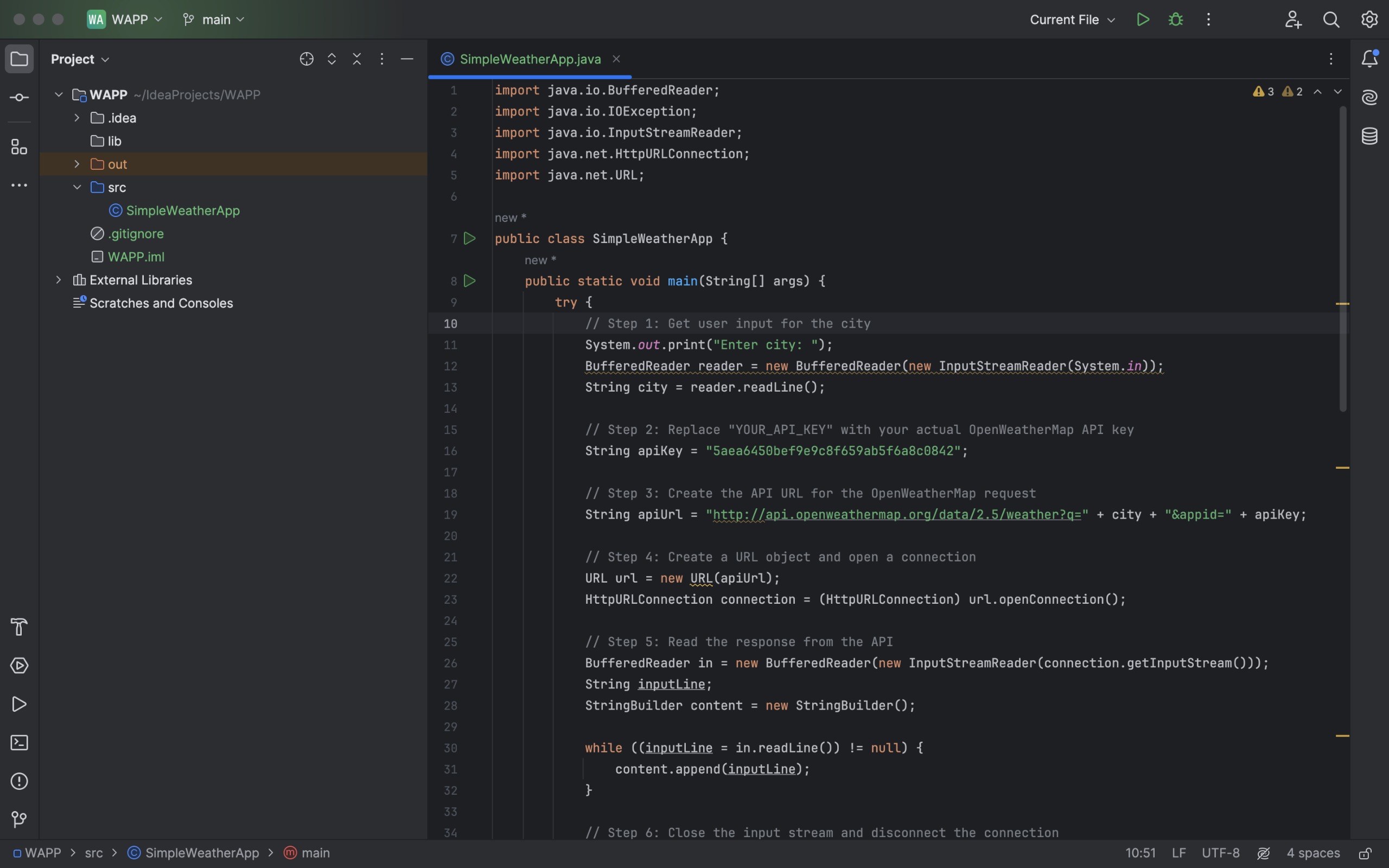
"id":2759794,

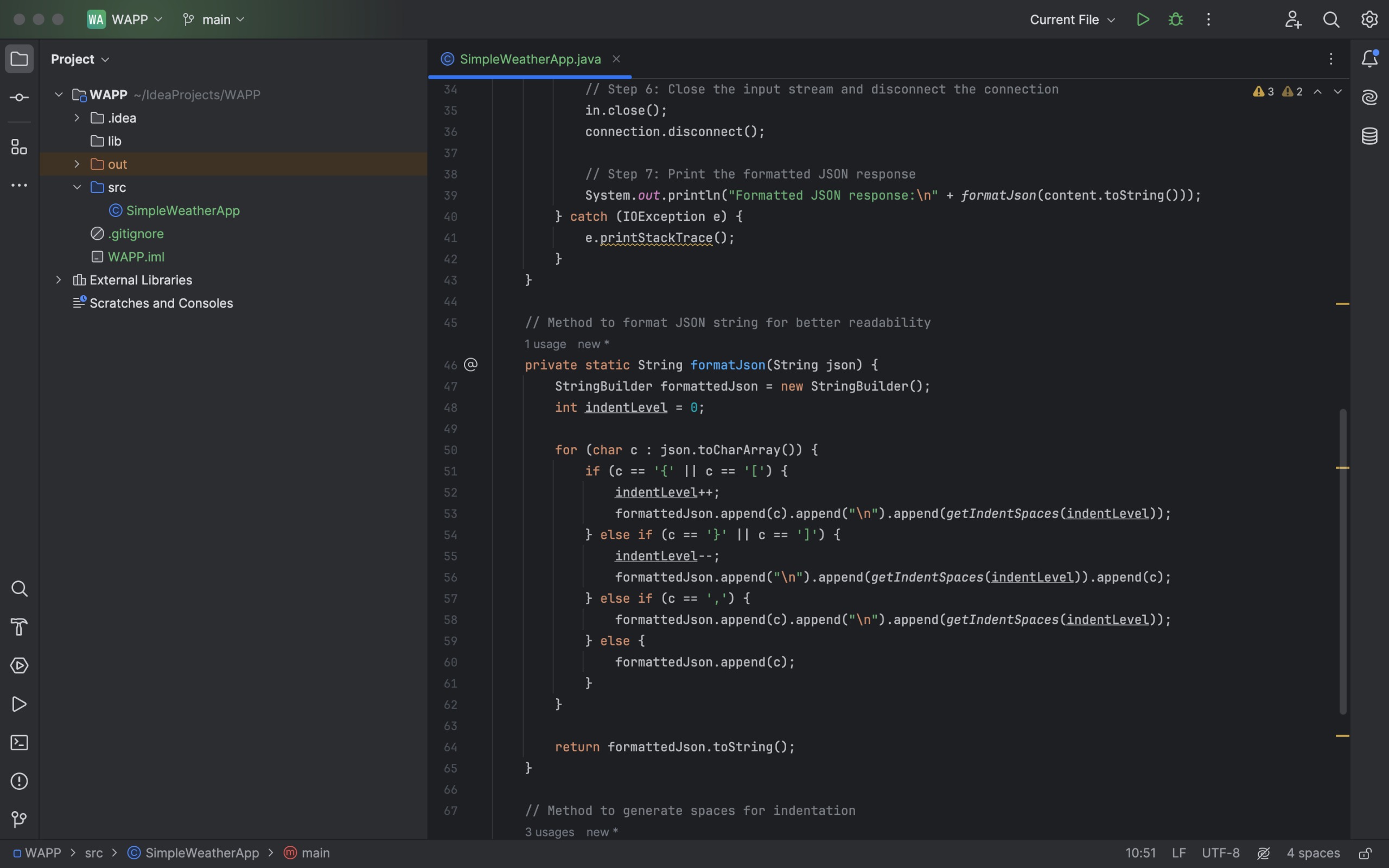
"name":"Amsterdam",

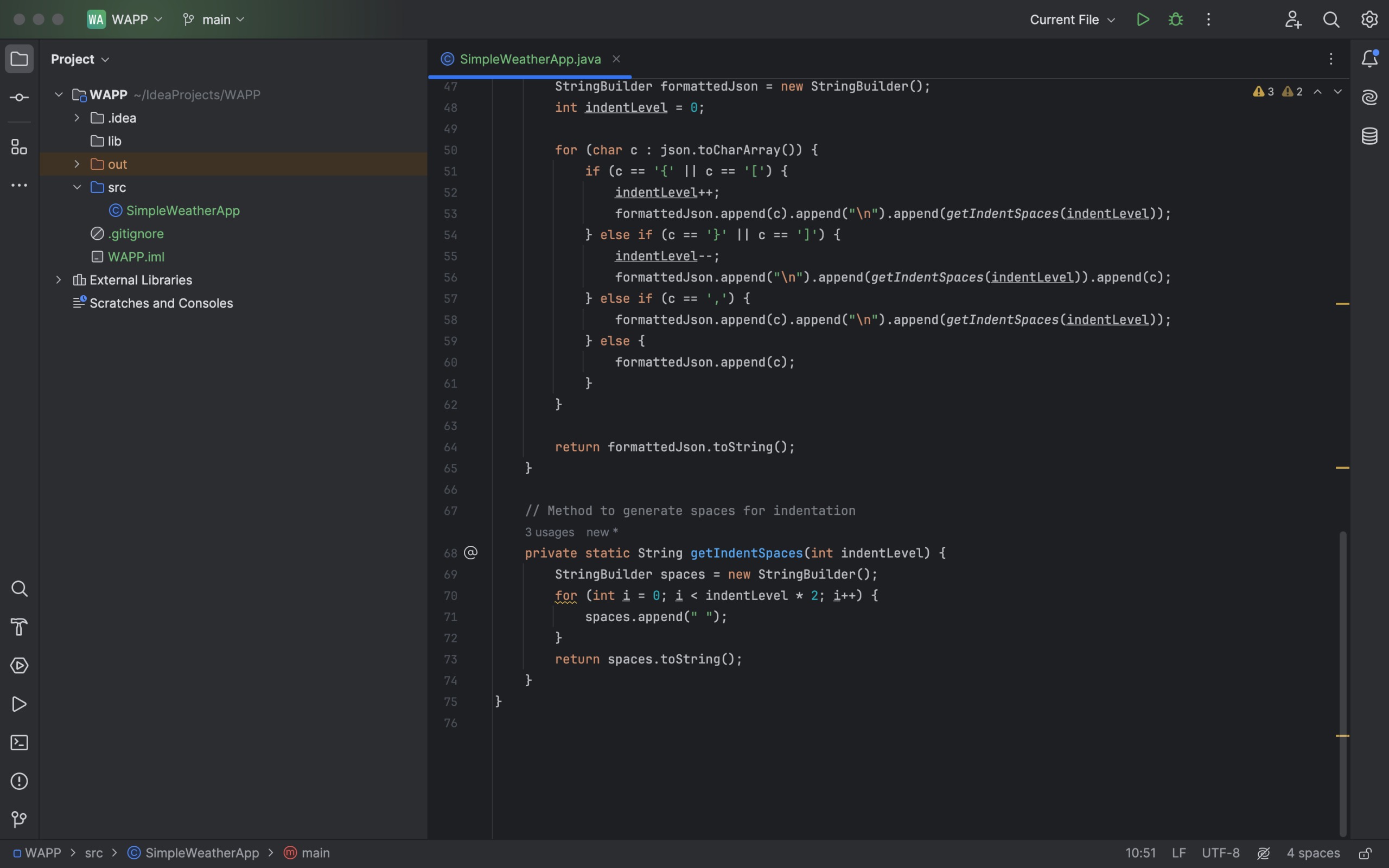
"cod":200

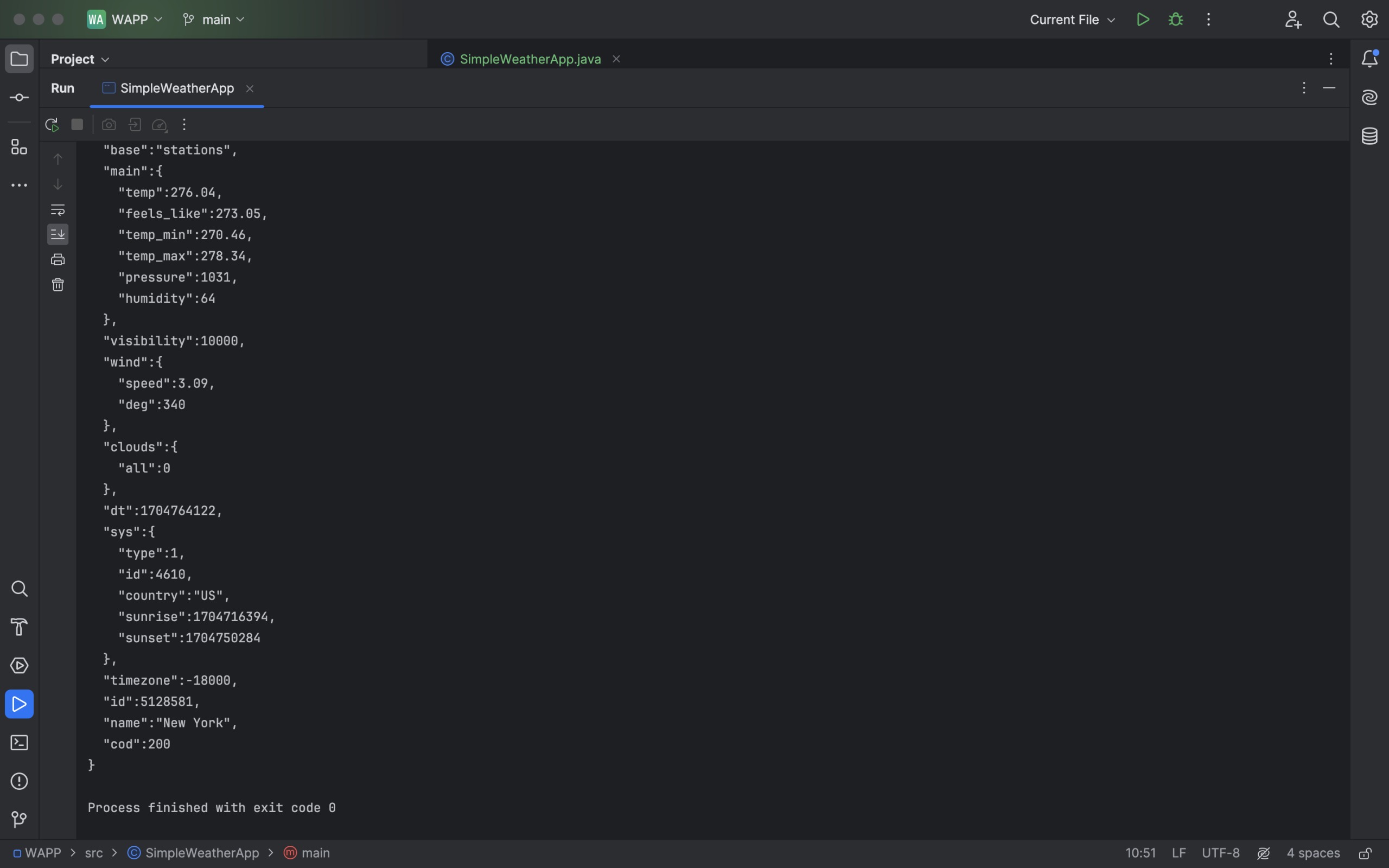
}

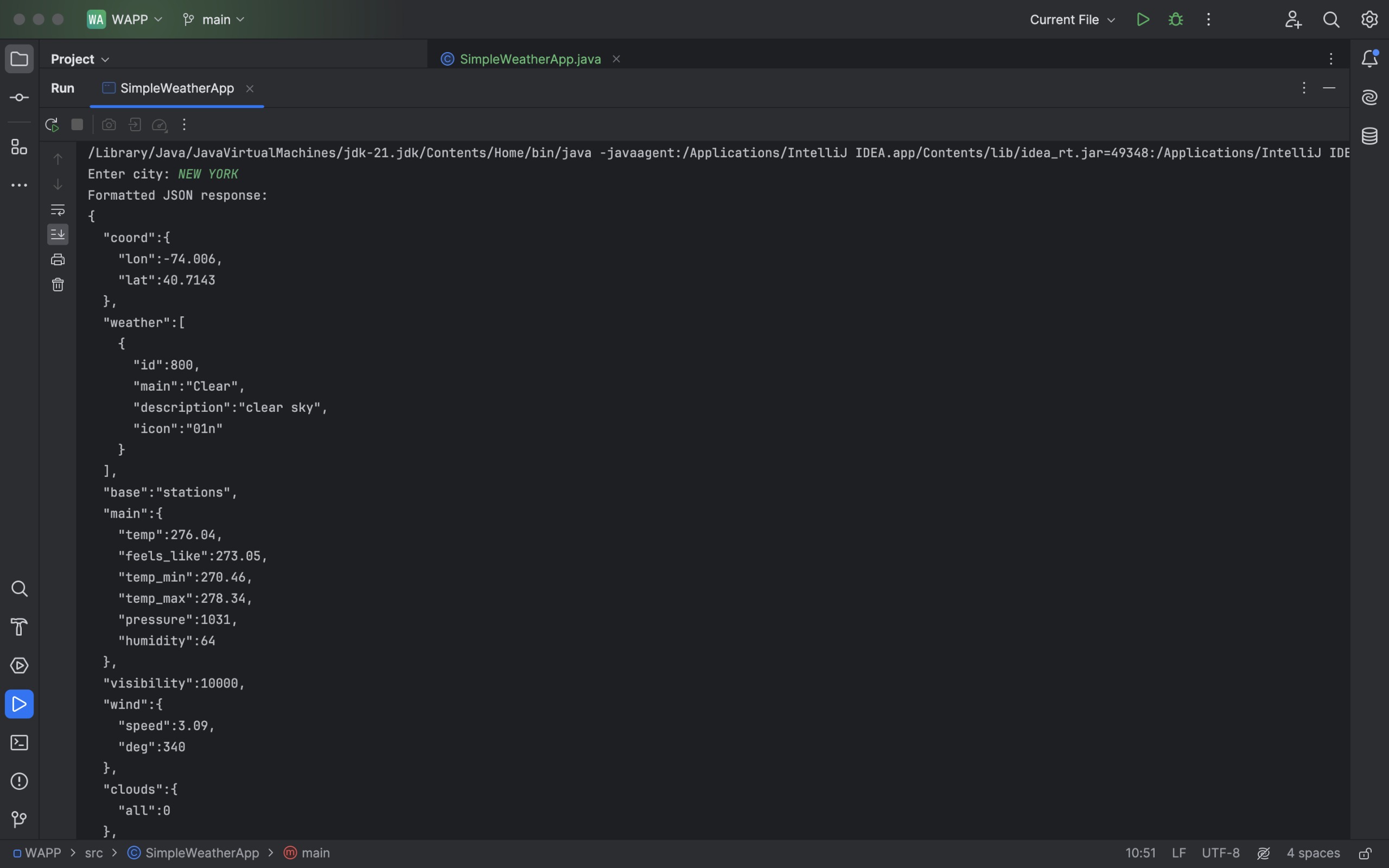
Process finished with exit code 0











**CHAPTER 6 REFERENCES**

1]. Choose a Weather API:

There are several weather APIs available. Some popular choices include:

OpenWeatherMap API: https://openweathermap.org/api

Weatherbit API: https://www.weatherbit.io/api

Dark Sky API (now part of Apple): https://developer.apple.com/documentation/weather

Choose an API based on your specific requirements, such as the level of detail needed, geographical coverage, and whether you need free or premium access

2]. Obtain API Key:

Most weather APIs require an API key for authentication. Sign up for an account on the chosen weather service's website to obtain your API key. Keep it secure, as it will be used to authenticate your requests.

3] . Set Up Your Development Environment:

Choose a programming language and framework for your app. Common choices include:

JavaScript (Node.js) for web applications

Java/Kotlin for Android apps

Swift for iOS apps

4. Make API Requests:

Use your chosen programming language to make HTTP requests to the weather API using the obtained API key. Typically, you'll request weather data for a specific location and receive a JSON response with the relevant information.

5. Display Weather Information:

Parse the JSON response and display the weather information in your app. You might want to show details like temperature, humidity, wind speed, and conditions.

6. Handle Errors and Edge Cases:

Handle errors gracefully, such as when the API request fails or returns unexpected data. Consider implementing features like user-friendly error messages and fallback mechanisms.

7. Enhance User Experience:

Consider adding features like geolocation for automatic location detection, a visually appealing UI, and perhaps hourly or extended forecasts.

8. Test Your App:

Test your app thoroughly to ensure it works correctly under different scenarios, including various locations and weather conditions.

9. Deploy Your App:

Once you're satisfied with your app, deploy it to your chosen platform (web server, app store, etc.).