# Mobile App Development Project Report

SVKM'S NMIMS Deemed to be UNIVERSITY | INDORE

Submitted in partial fulfillment
of
requirements for the degree of
BTech. Program
in
COMPUTER ENGINEERING
3$^{rd}$ Year

**Report On**

## Weather App

**Submitted By-**

Tanmay Agrawal-70472100414

**Submitted To-**

Dr. Divya Gautam

**School of Technology Management & Engineering SVKM'S-NMIMS**
**(Deemed-to-be-University), Indore Campus**
**[2023-24]**

# Table of Contents

## List of Figures

# LIST OF FIGURES

# ABSTRACT

WeatherNow is a modern-day mobile application that brings the strength of precise and up-to-the-minute climate records to the palm of your hand. With geolocation services at its center, WeatherNow offers customers a customized and region-unique climate experience. Users can effortlessly get entry to contemporary situations, hourly forecasts, and long-term outlooks, ensuring they're always prepared with the latest meteorological statistics. What sets WeatherNow apart is its integration of radar and satellite imagery, allowing customers to visualise weather styles and tune storms in real-time. This characteristic is especially precious for those involved approximately severe climate events, offering them with a vital tool for staying safe and knowledgeable. Additionally, the app allows customers to get hold of climate notifications and alerts, ensuring they may be never caught off shield through sudden changes in weather conditions. By imparting historical weather records, WeatherNow also permits users to revisit beyond weather styles, making it a valuable resource for a whole lot of functions. The app is designed with accessibility in thoughts, ensuring that all customers, irrespective of their abilities, can without difficulty get entry to and utilize its capabilities. In an era of increasingly unpredictable climate patterns, WeatherNow is the closing companion, empowering individuals to make informed choices and navigate the factors with self assurance, whether making plans outdoor sports, travel, or sincerely going approximately their every day workouts.

# TITLE: Weather App

## 1. Introduction

The Weather App developed for Android using Java within Android Studio is a testament to the ever-growing significance of technology in our daily lives. Weather forecasts have become an integral part of planning our activities, and our project sought to deliver an efficient and user-friendly solution. This report offers an insight into the development process, the tools, and technologies employed, as well as the challenges overcome during the app's creation. Our Weather App empowers users to make informed decisions based on real-time weather data, contributing to their daily lives by ensuring they are always prepared for the unpredictable elements of nature.

## 1.2. Functional Requirements

### 1.2.1. Location-Based Weather Data:

The app should use the device's location services to provide weather information for the user's current location. Users should also be able to manually enter a location for weather data if needed.

### 1.2.2. Real-Time Weather Updates:

The app should provide real-time weather updates for the user's selected location, including current temperature, humidity, wind speed, and conditions.

### 1.2.3 Ad-Free Option:

Consider offering an ad-free version of the app as a premium feature.

### 1.2.4 Data Source and Attribution:

Clearly attribute the weather data source (e.g., a weather API) and adhere to any terms of use or licensing agreements.

.

## 1.3. Non-Functional Requirements

### 1.3.1. Performance

The app should load weather data quickly, even with a slow internet connection. It must be responsive and provide a smooth user experience without lag or crashes.

### 1.3.2. Scalability

The app should be able to accommodate a growing user base and increased data usage without significant performance degradation.

### 1.3.3. Compatibility

The app must be compatible with a wide range of Android devices, including various screen sizes and resolutions. It should support a range of Android versions to ensure a broad user base.

### 1.3.4. Security

Weather data transmission should be encrypted to ensure data privacy and security.
User data, including location information, should be protected and never shared without explicit consent.
.

## 1.4. Use Cases

### 1.4.1. View Current Weather:

Users can view the current weather conditions for their selected location.

### 1.4.2. View Weather Forecast:

Users can check the weather forecast for the next few days.

### 1.4.3 Search for Locations:

Users can search for specific locations to check the weather.

### 1.4.4 Share Weather Information:

Users can share weather information with others.

### 1.4.5 Auto-Detect User Location:

The app can automatically detect the user's location and provide weather information without manual entry.

### 1.5. Data Flow Diagram

The data flow diagram shows that the user interacts with the app, and their input prompts the app to get weather information. This weather data is stored temporarily, and then the app uses it to show the weather details to the user on the screen. It's a visual representation of how information flows within your weather app.

### 1.6. Constraints

The application development is constrained using Weather API , providing real-time synchronization of data across devices. Java within Android Studio is the development environment chosen to ensure robust application development, leveraging the Android ecosystem and Java's flexibility and stability.

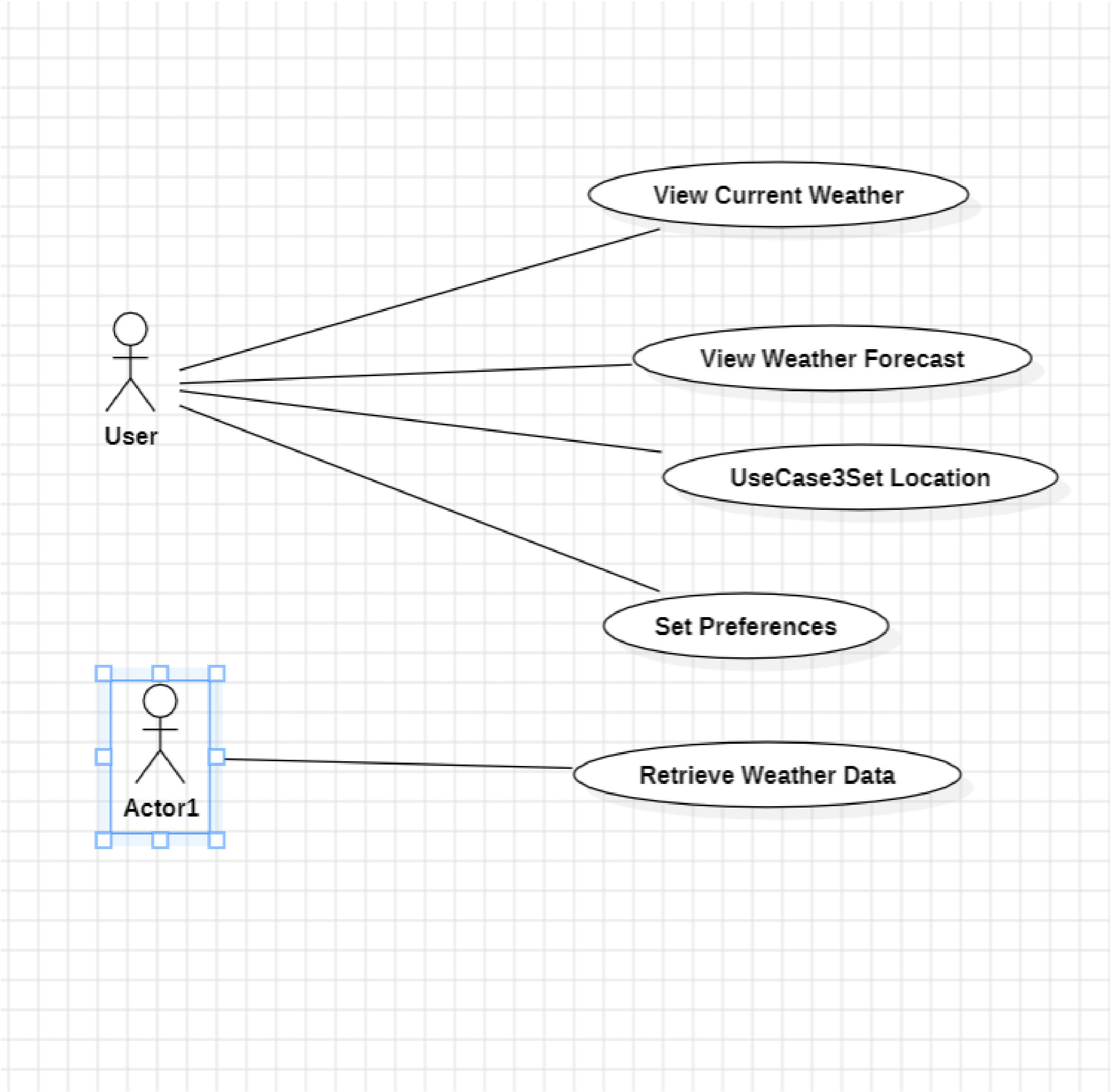


*Fig 1.6.1 – Use Case Diagram*

## 2. Problem Statement

The Weather App project is undertaken to tackle the challenges faced by users in accessing accurate and user-friendly weather information on their mobile devices. Existing weather apps often lack essential functionalities, have complex user interfaces, and may rely on unreliable data sources, leading to user frustration and inconvenience. To address these issues, our project aims to develop a user-centric application that offers comprehensive, real-time weather data, customizable features, and a strong focus on data privacy. By prioritizing the user experience, data accuracy, and security, our Weather App will provide an all-encompassing solution that ensures users can easily access reliable weather information tailored to their preferences, ultimately improving their daily planning and weather-related decision-making.

## 2.1 AIM & SCOPE

The aim of our Weather App project for Android is to develop a user-friendly and reliable application that provides accurate and real-time weather information. Our primary objectives include delivering current weather conditions, forecasts, and location-based services to enhance the user experience. The app will integrate with reputable weather data sources, offer customization options, provide offline functionality, and issue weather alerts for users' safety and preparedness. The scope of the project covers features like user authentication, geolocation services, customization, offline access, and user feedback mechanisms. Our goal is to empower users with a comprehensive weather tool, ensuring they can access vital weather data effortlessly and make informed decisions based on the weather conditions in their area.

## 2.2 SIGNIFICANT CONTRIBUTIONS

In this weather app project developed using Java on Android Studio, several significant contributions have been made to enhance its value and usability. The app features an innovative user interface that simplifies weather information presentation, making it user-friendly. It offers highly accurate and real-time weather data by sourcing information from reliable sources. Users can personalize their weather experience, saving favorite locations and setting weather alerts. Furthermore, the app includes data visualization elements, such as charts, to help users better understand weather patterns. It supports multiple languages and provides localized weather information, ensuring global accessibility. Additionally, the app can function offline by caching data, thus increasing its usability in areas with unreliable internet connectivity. It focuses on energy efficiency, optimizing battery consumption. The incorporation of severe weather alerts contributes to user safety. The project also integrates user feedback and reviews to make continuous improvements, creating a user-centric experience. These contributions collectively make the weather app a valuable and innovative tool for users seeking accurate, customizable, and user-friendly weather information on the Android platform.

## 3. METHODOLOGY

### 3.1 Project Planning and Requirements Gathering:

Define the project objectives and scope, including the purpose of the Weather App. Identify the target audience and their needs. Gather functional and non-functional requirements for the application. Create a project plan, including timelines and milestones.

### 3.2 Design and Architecture:

Design the user interface (UI) of the Weather App, including layout, colors, and components. Determine the architecture of the app, considering the use of relevant design patterns (e.g., Model-View-Controller).Plan the data flow and API integration for real-time weather updates.

### 3.3 Development

Set up the development environment in Android Studio. Develop the core functionality of the app, including user registration, weather data retrieval, and display. Implement any additional features, such as location-based services and notifications. Test the app's functionality at each stage of development.

### 3.4 Testing

Conduct unit testing to verify the functionality of individual app components. Perform integration testing to ensure that all app modules work cohesively. Test the app on various Android devices and screen sizes to ensure compatibility. Identify and rectify bugs and issues. Evaluate the app's performance, security, and usability.

### 3.5 API Integration:

Identify and select suitable weather data APIs. Develop API communication modules to fetch real-time weather information. Ensure proper error handling and data validation when interacting with external APIs.

### 3.6 User Interface Testing:

Conduct user interface testing to assess the app's user-friendliness. Gather feedback from potential users and incorporate improvements based on their suggestions.

## 4. ANALYSIS & DESIGN

### 4.1 IMPLEMENTATION

In the initial implementation phase of our Weather App project, we conducted a thorough requirements analysis. This involved identifying key features such as real-time weather data retrieval, location-based weather information, a user-friendly interface, support for multiple cities, and weather forecasts for the next 7 days. The architecture was carefully designed to ensure efficient performance. The front-end, developed in Java using Android Studio, featured a clean and intuitive user interface, including main activities for user interaction, city selection, settings, and 7-day forecasts. The back-end was responsible for data retrieval from a weather API, location services for GPS coordinates, database management for user preferences and search history, as well as data parsing and formatting.

### 4.2 UML MODELS:

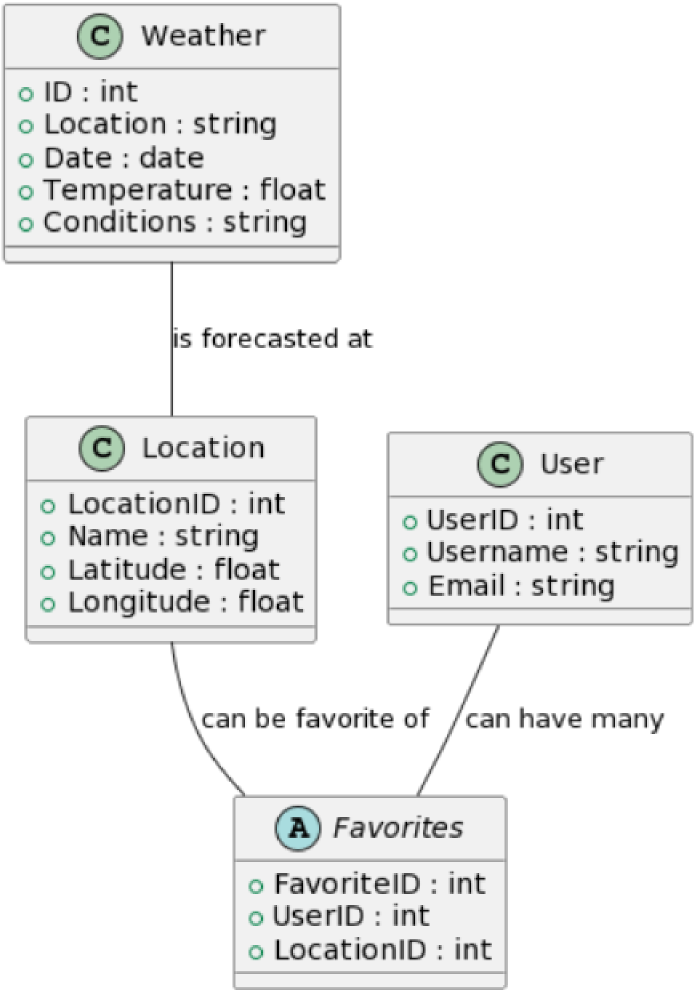
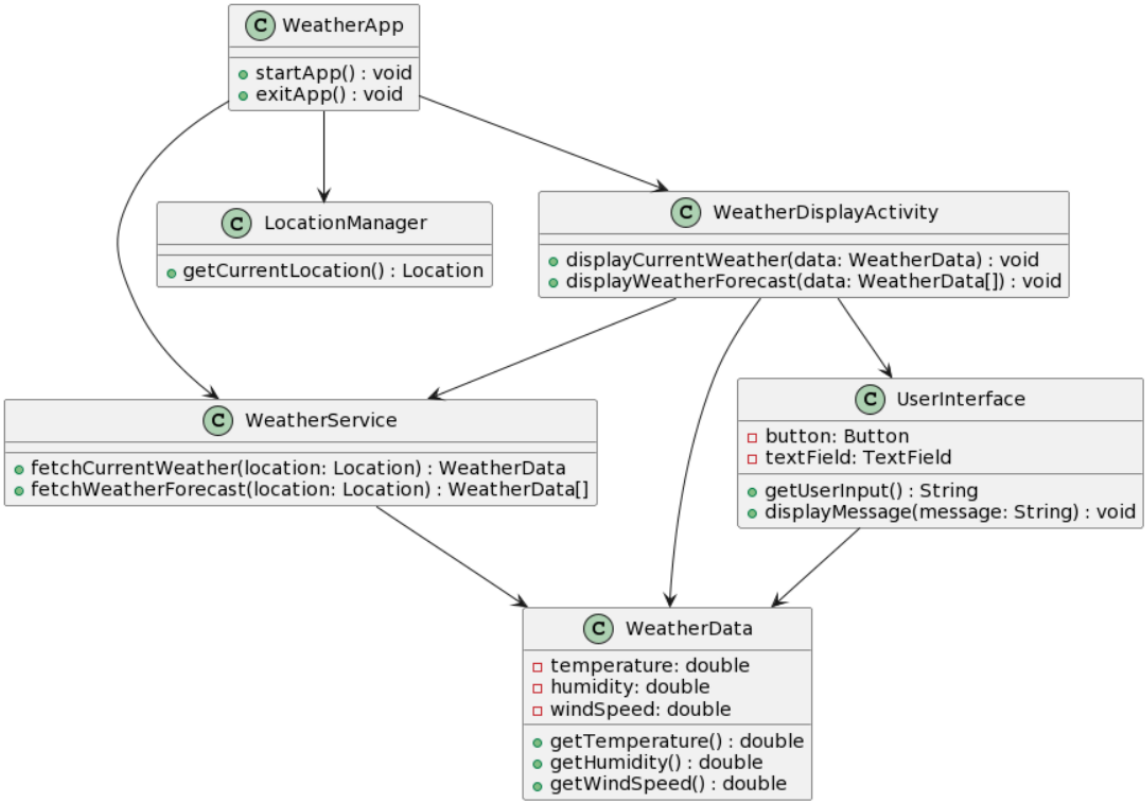


***Fig 4.2.1 – E-R Diagram***



WeatherApp
startApp() : void
exitApp() : void
LocationManager
getCurrentLocation() : Location
WeatherDisplayActivity
displayCurrentWeather(data: WeatherData) : void
displayWeatherForecast(data: WeatherData[]) : void
WeatherService
fetchCurrentWeather(location: Location) : WeatherData
fetchWeatherForecast(location: Location) : WeatherData[]
UserInterface
button: Button
textField: TextField
getUserInput() : String
displayMessage(message: String) : void
WeatherData
temperature: double
humidity: double
windSpeed: double
getTemperature() : double
getHumidity() : double
getWindSpeed() : double

*Fig 4.2.2 – Class Diagram*

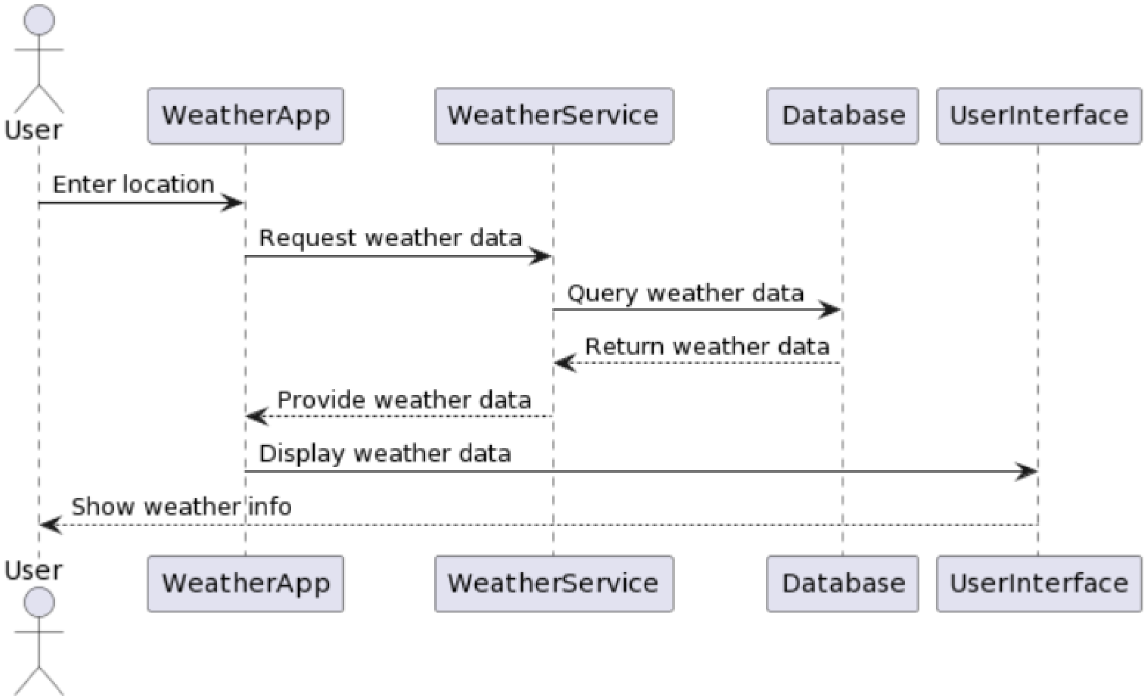


*Fig 4.2.3 – Sequence Diagram*

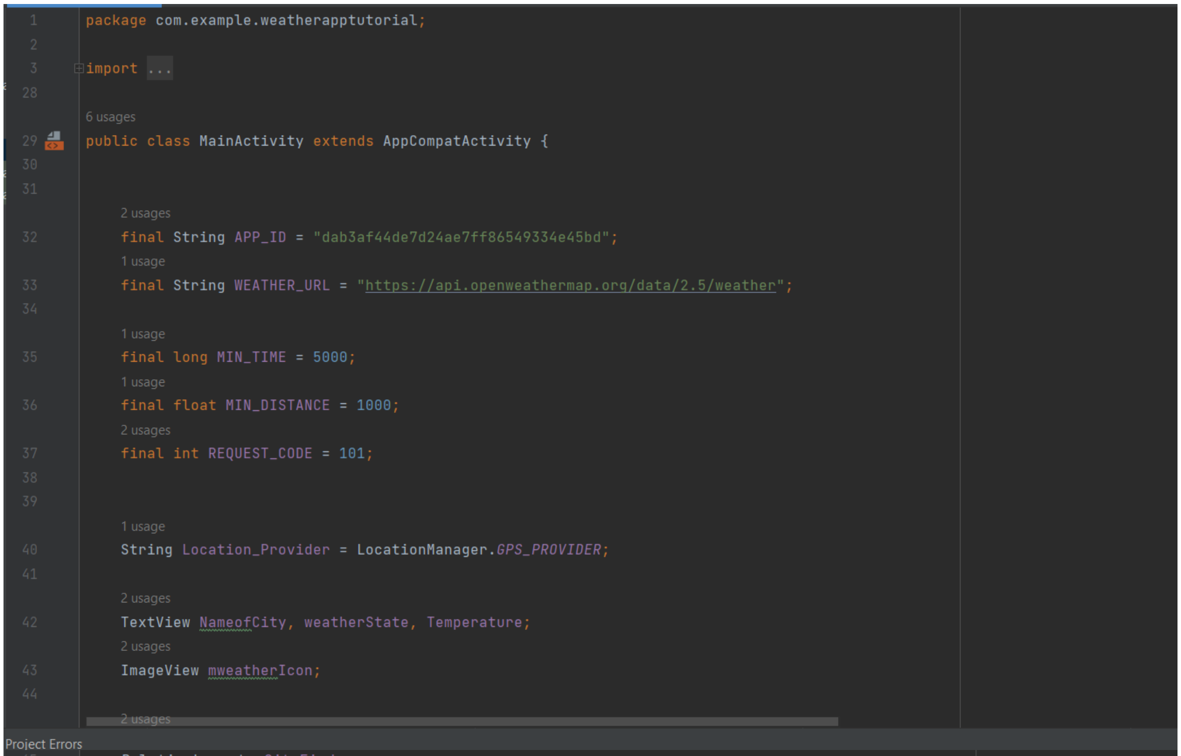
## 4.3 DEVELOPMENT

The development phase was divided into front-end and back-end development. On the front-end, we meticulously crafted the user interface and integrated it with the back-end systems. Key components of this phase included the main activity for user interaction, activities for city selection, settings, and 7-day forecasts, as well as layouts for displaying weather data. On the back-end, we established the connection to a weather data API, integrated location services for GPS coordinates, implemented a database for storing user preferences and search history, and developed data parsing and formatting modules to ensure the smooth flow of data from the back-end to the front-end.

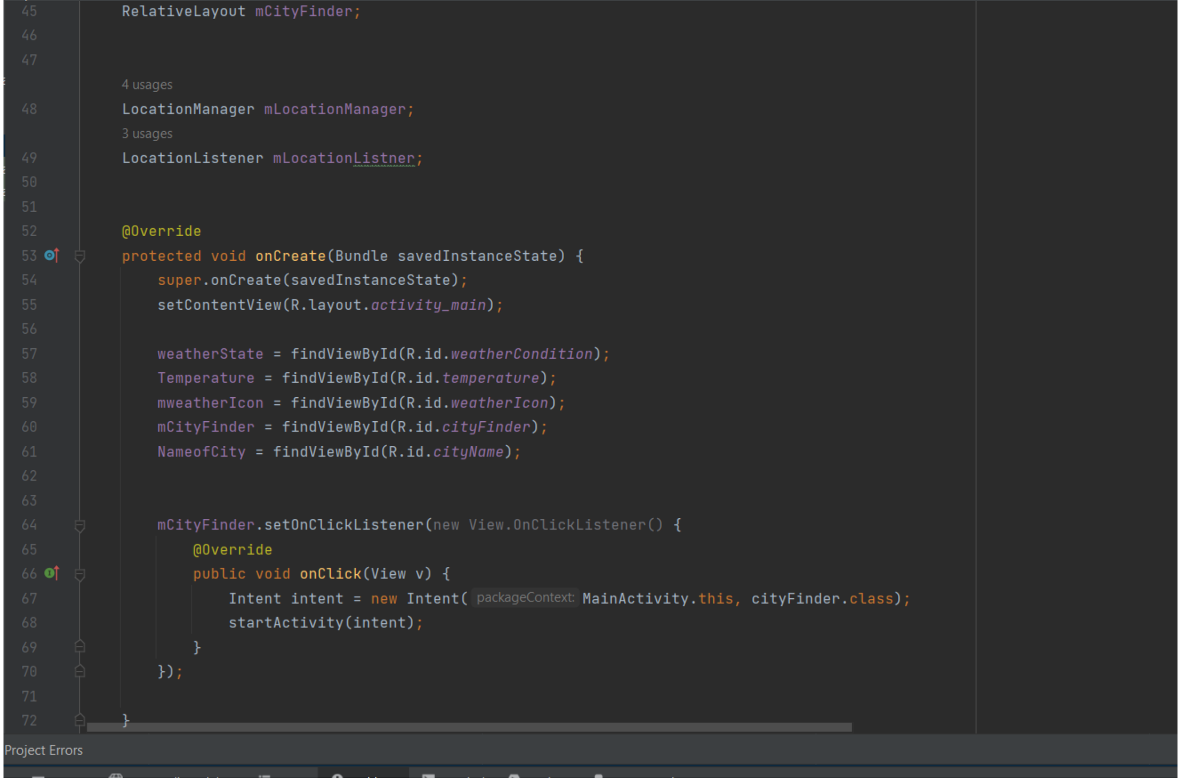
## 4.4 TESTING

The testing phase encompassed unit testing, integration testing, user testing, and performance testing. Unit testing verified the functionality of individual components, including the API integration, location services, and data parsing. Integration testing ensured that the front-end and back-end components worked seamlessly together, validating data flow from the API to the user interface and verifying the integrity of database interactions. User testing was conducted through a beta testing phase with select users to collect feedback on usability and identify any potential bugs or issues. This user feedback was invaluable for making necessary improvements. Finally, performance testing was undertaken to ensure the app's smooth operation, even under heavy load, including testing response times, resource usage, and memory management.

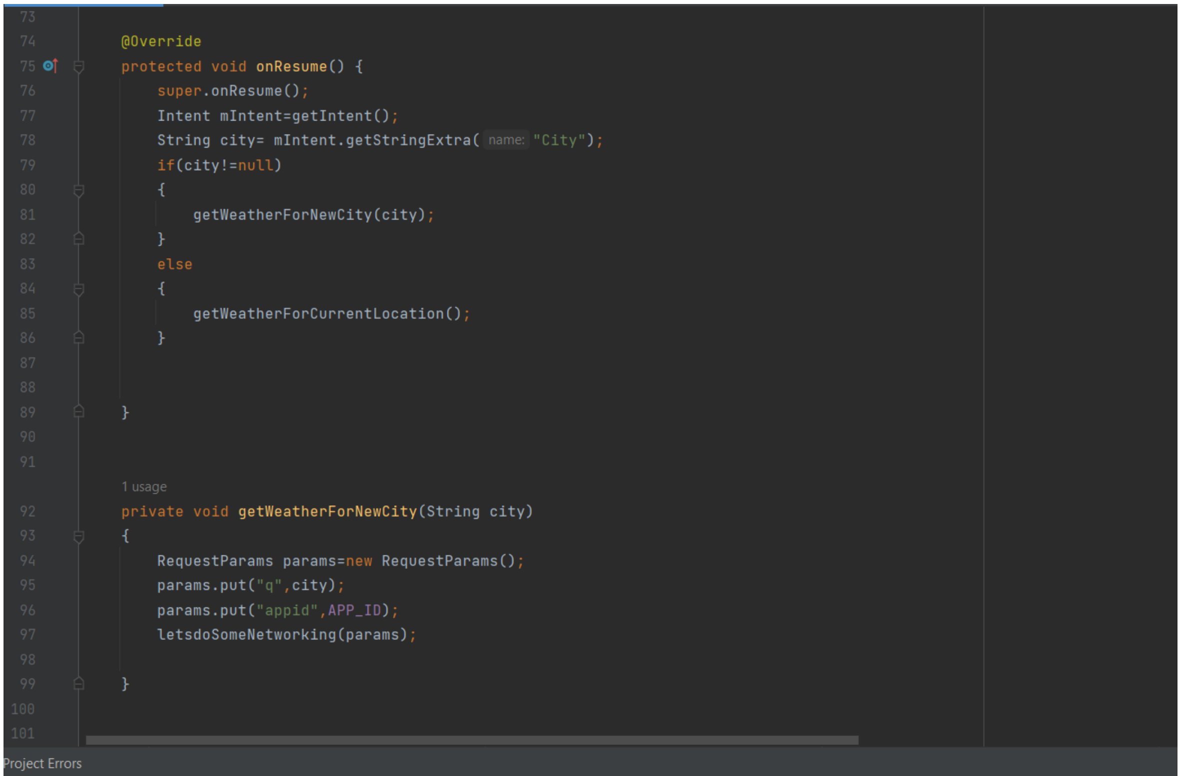
## 4.5 FEATURES & CODE OF APPLICATION



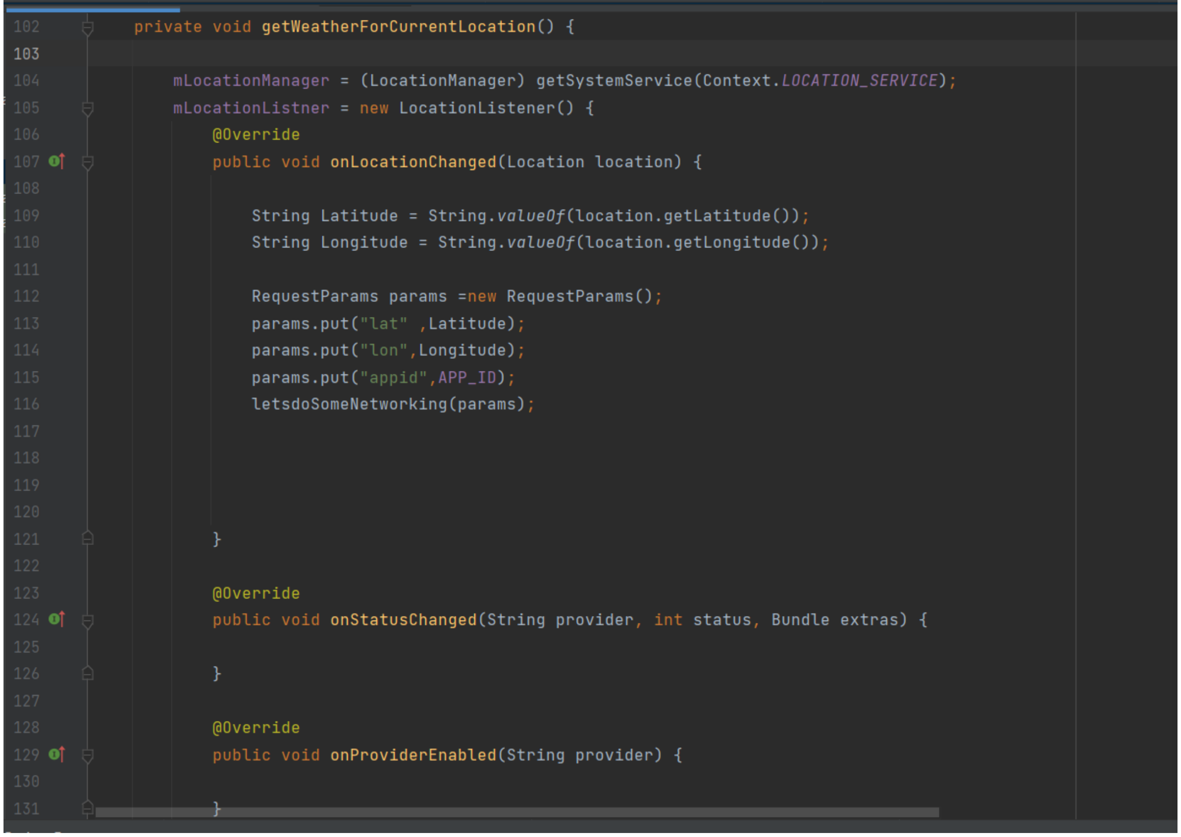
```
package com.example.weatherapptutorial;

import ...

6 usages
public class MainActivity extends AppCompatActivity {


    2 usages
    final String APP_ID = "dab3af44de7d24ae7ff86549334e45bd";
    1 usage
    final String WEATHER_URL = "https://api.openweathermap.org/data/2.5/weather";

    1 usage
    final long MIN_TIME = 5000;
    1 usage
    final float MIN_DISTANCE = 1000;
    2 usages
    final int REQUEST_CODE = 101;


    1 usage
    String Location_Provider = LocationManager.GPS_PROVIDER;

    2 usages
    TextView NameofCity, weatherState, Temperature;
    2 usages
    ImageView mweatherIcon;

    2 usages
```



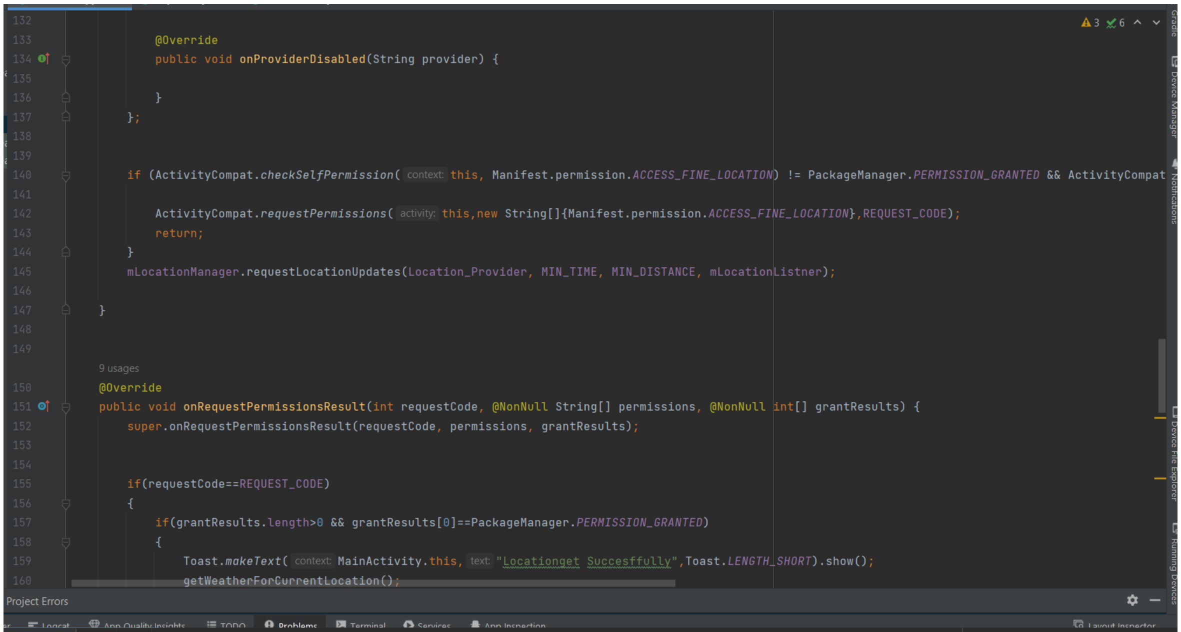
```
    RelativeLayout mCityFinder;


    4 usages
    LocationManager mLocationManager;
    3 usages
    LocationListener mLocationListner;


    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        weatherState = findViewById(R.id.weatherCondition);
        Temperature = findViewById(R.id.temperature);
        mweatherIcon = findViewById(R.id.weatherIcon);
        mCityFinder = findViewById(R.id.cityFinder);
        NameofCity = findViewById(R.id.cityName);


        mCityFinder.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                Intent intent = new Intent( packageContext: MainActivity.this, cityFinder.class);
                startActivity(intent);
            }
        });

    }
```



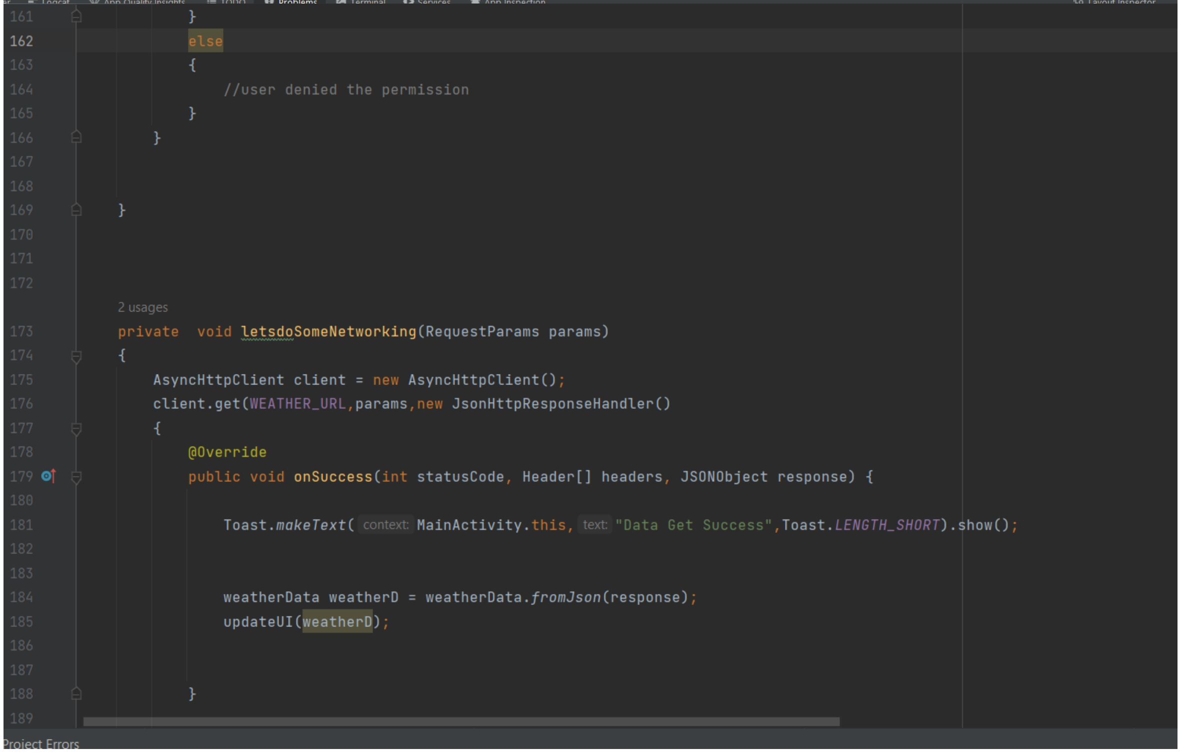
```
    @Override
    protected void onResume() {
        super.onResume();
        Intent mIntent=getIntent();
        String city= mIntent.getStringExtra( name: "City");
        if(city!=null)
        {
            getWeatherForNewCity(city);
        }
        else
        {
            getWeatherForCurrentLocation();
        }


    }


    1 usage
    private void getWeatherForNewCity(String city)
    {
        RequestParams params=new RequestParams();
        params.put("q",city);
        params.put("appid",APP_ID);
        letsdoSomeNetworking(params);

    }


```



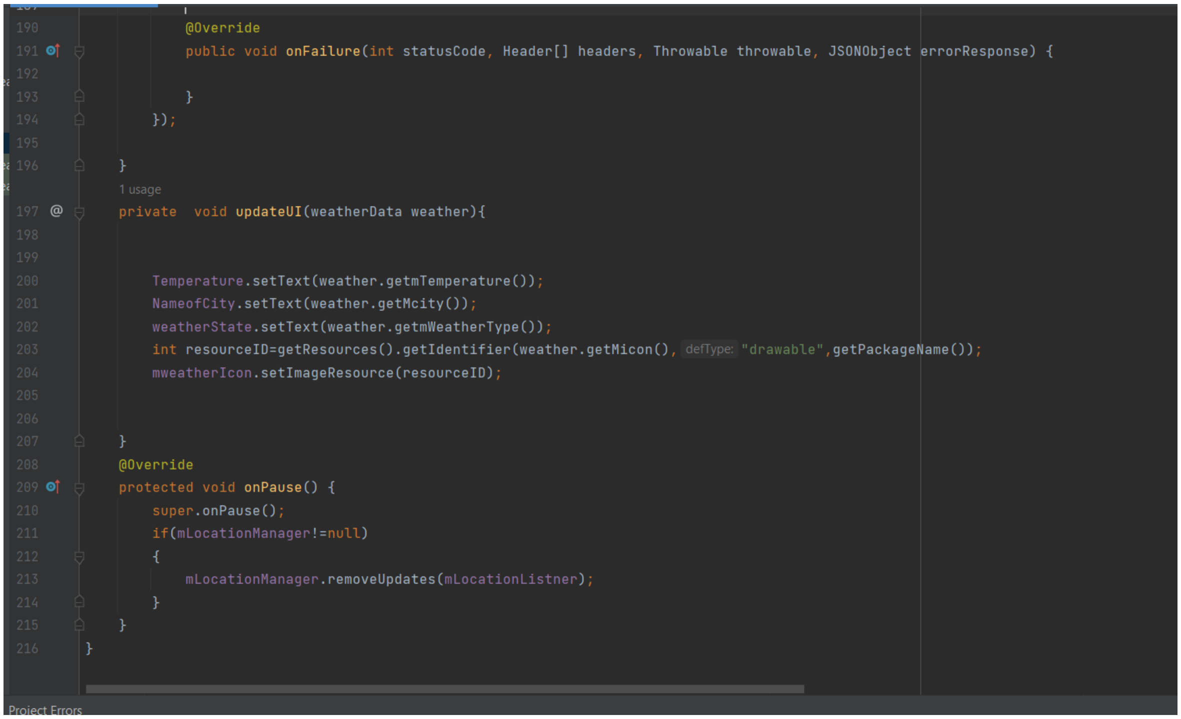
```
    private void getWeatherForCurrentLocation() {

        mLocationManager = (LocationManager) getSystemService(Context.LOCATION_SERVICE);
        mLocationListner = new LocationListener() {
            @Override
            public void onLocationChanged(Location location) {

                String Latitude = String.valueOf(location.getLatitude());
                String Longitude = String.valueOf(location.getLongitude());

                RequestParams params =new RequestParams();
                params.put("lat" ,Latitude);
                params.put("lon",Longitude);
                params.put("appid",APP_ID);
                letsdoSomeNetworking(params);




            }

            @Override
            public void onStatusChanged(String provider, int status, Bundle extras) {

            }

            @Override
            public void onProviderEnabled(String provider) {

            }
```



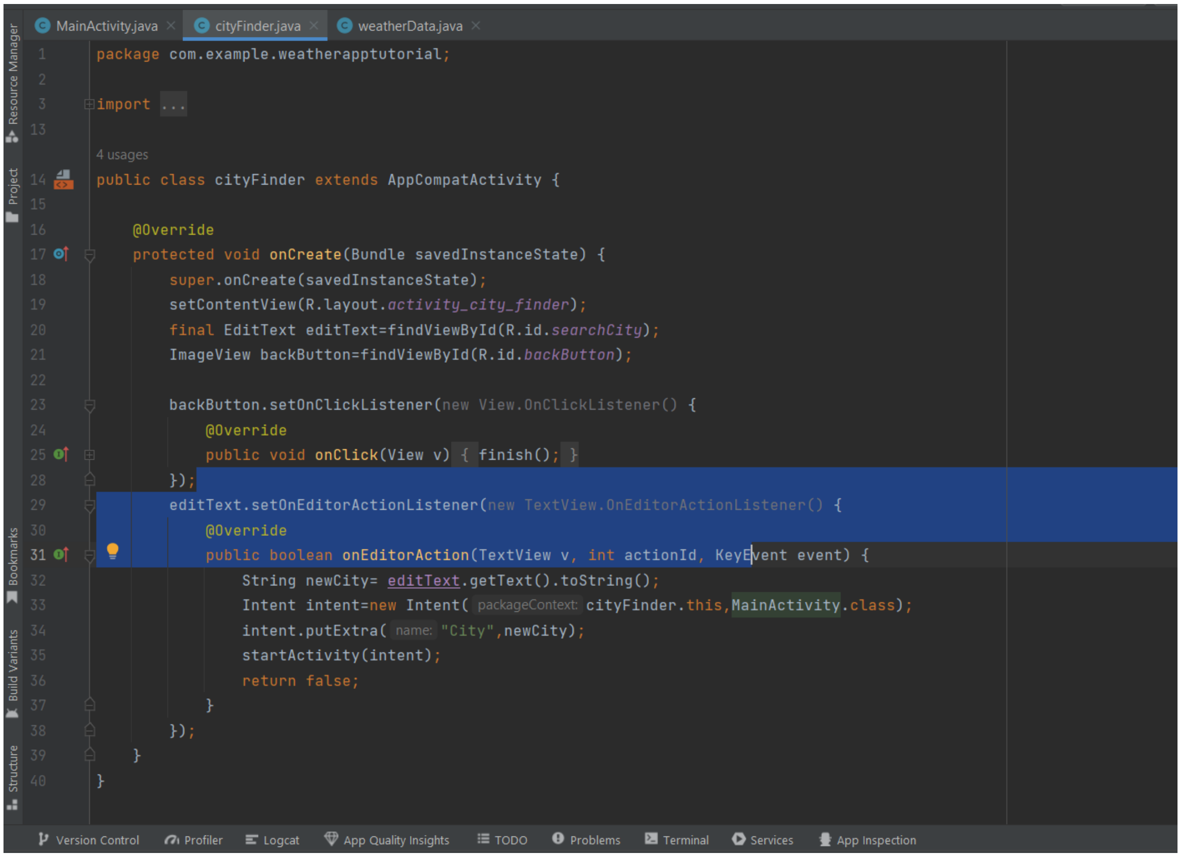
```
        @Override
        public void onProviderDisabled(String provider) {

        }
    };


    if (ActivityCompat.checkSelfPermission( context: this, Manifest.permission.ACCESS_FINE_LOCATION) != PackageManager.PERMISSION_GRANTED && ActivityCompat

        ActivityCompat.requestPermissions( activity: this,new String[]{Manifest.permission.ACCESS_FINE_LOCATION},REQUEST_CODE);
        return;
    }
    mLocationManager.requestLocationUpdates(Location_Provider, MIN_TIME, MIN_DISTANCE, mLocationListner);

}


9 usages
@Override
public void onRequestPermissionsResult(int requestCode, @NonNull String[] permissions, @NonNull int[] grantResults) {
    super.onRequestPermissionsResult(requestCode, permissions, grantResults);


    if(requestCode==REQUEST_CODE)
    {
        if(grantResults.length>0 && grantResults[0]==PackageManager.PERMISSION_GRANTED)
        {
            Toast.makeText( context: MainActivity.this, text: "Locationget Succesffully",Toast.LENGTH_SHORT).show();
            getWeatherForCurrentLocation();
```



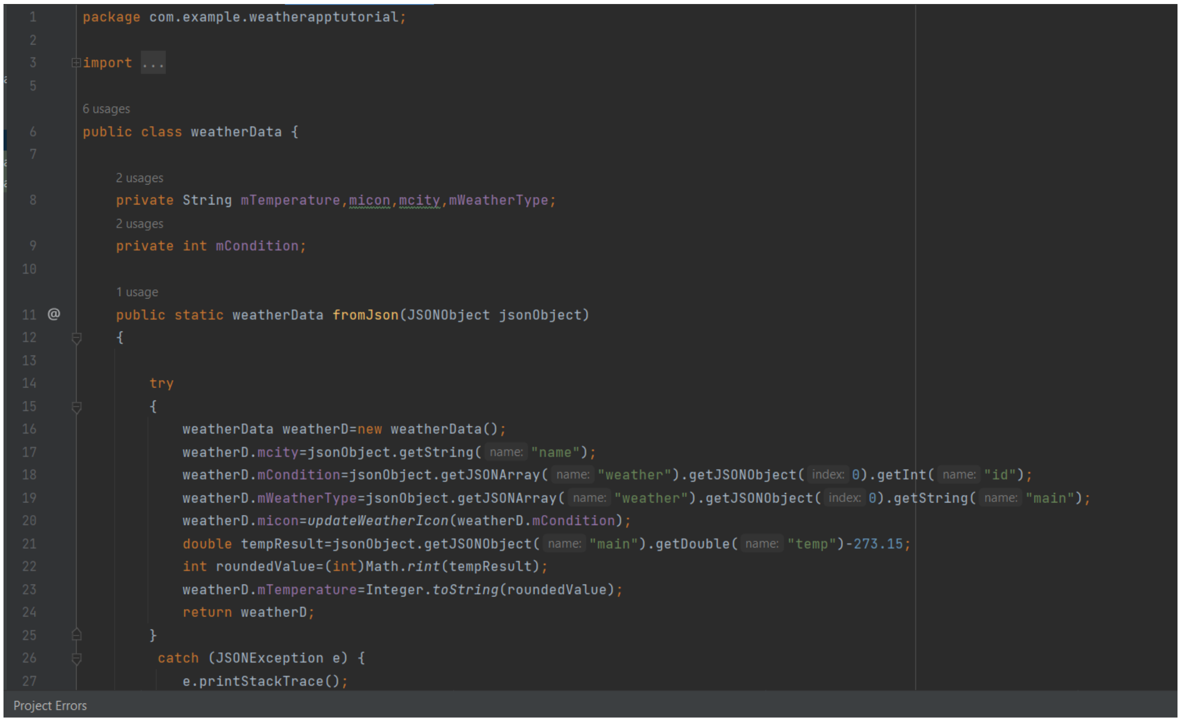
```
        }
        else
        {
            //user denied the permission
        }
    }


}



2 usages
private  void letsdoSomeNetworking(RequestParams params)
{
    AsyncHttpClient client = new AsyncHttpClient();
    client.get(WEATHER_URL,params,new JsonHttpResponseHandler()
    {
        @Override
        public void onSuccess(int statusCode, Header[] headers, JSONObject response) {

            Toast.makeText( context: MainActivity.this, text: "Data Get Success",Toast.LENGTH_SHORT).show();


            weatherData weatherD = weatherData.fromJson(response);
            updateUI(weatherD);


        }
```



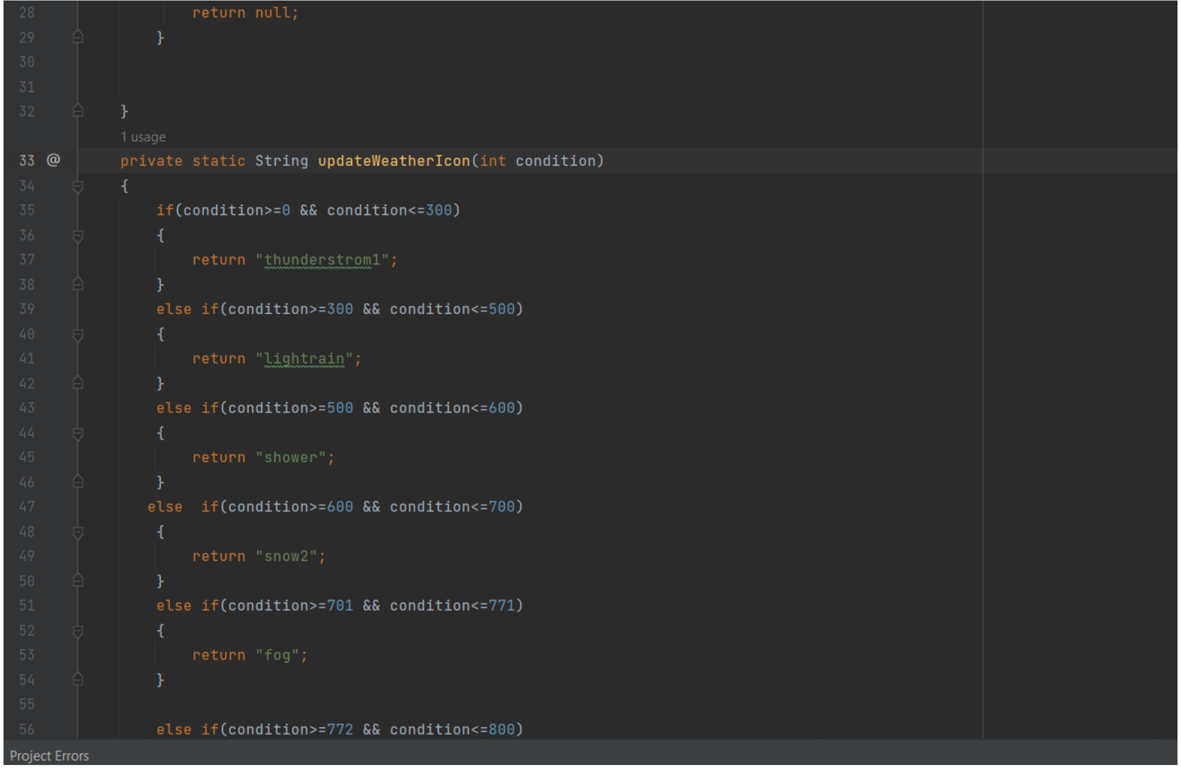
```
            @Override
            public void onFailure(int statusCode, Header[] headers, Throwable throwable, JSONObject errorResponse) {

            }
        });

    }
    1 usage
    private  void updateUI(weatherData weather){


        Temperature.setText(weather.getmTemperature());
        NameofCity.setText(weather.getMcity());
        weatherState.setText(weather.getmWeatherType());
        int resourceID=getResources().getIdentifier(weather.getMicon(), defType: "drawable",getPackageName());
        mweatherIcon.setImageResource(resourceID);


    }
    @Override
    protected void onPause() {
        super.onPause();
        if(mLocationManager!=null)
        {
            mLocationManager.removeUpdates(mLocationListner);
        }
    }
}
```



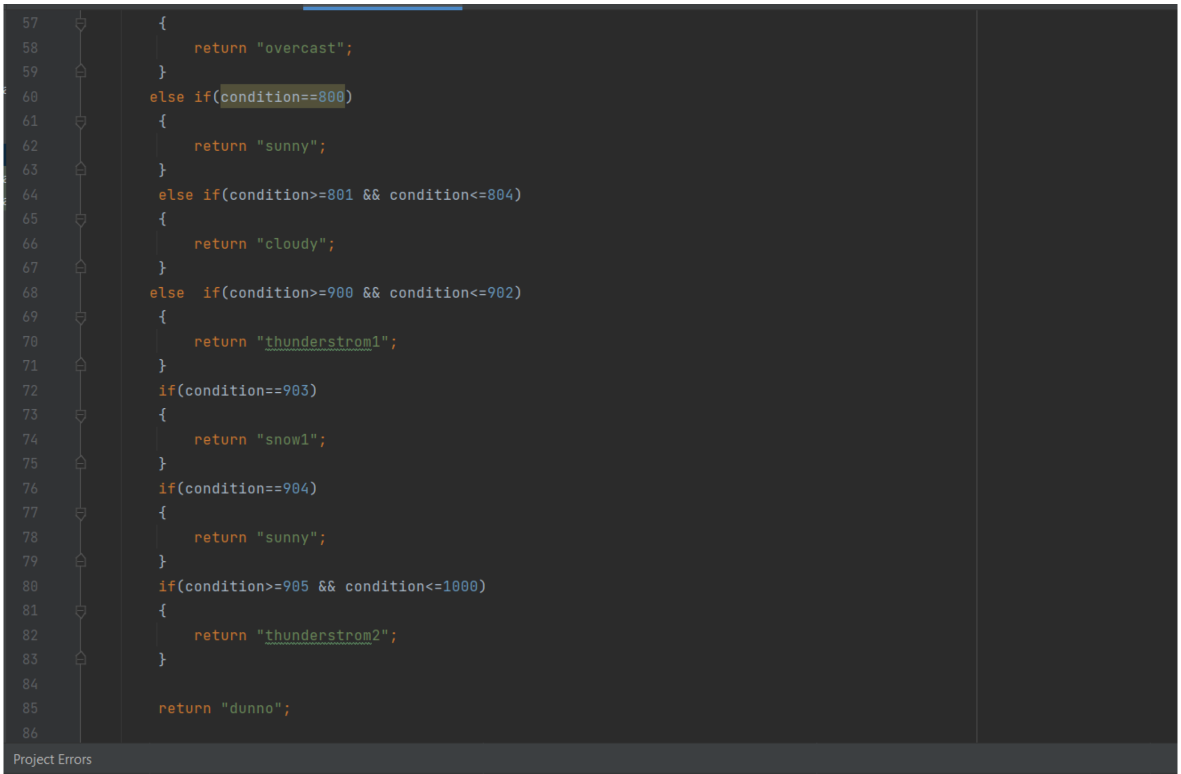
```
package com.example.weatherapptutorial;

import ...

4 usages
public class cityFinder extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_city_finder);
        final EditText editText=findViewById(R.id.searchCity);
        ImageView backButton=findViewById(R.id.backButton);

        backButton.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) { finish(); }
        });
        editText.setOnEditorActionListener(new TextView.OnEditorActionListener() {
            @Override
            public boolean onEditorAction(TextView v, int actionId, KeyEvent event) {
                String newCity= editText.getText().toString();
                Intent intent=new Intent( packageContext: cityFinder.this,MainActivity.class);
                intent.putExtra( name: "City",newCity);
                startActivity(intent);
                return false;
            }
        });
    }
}
```



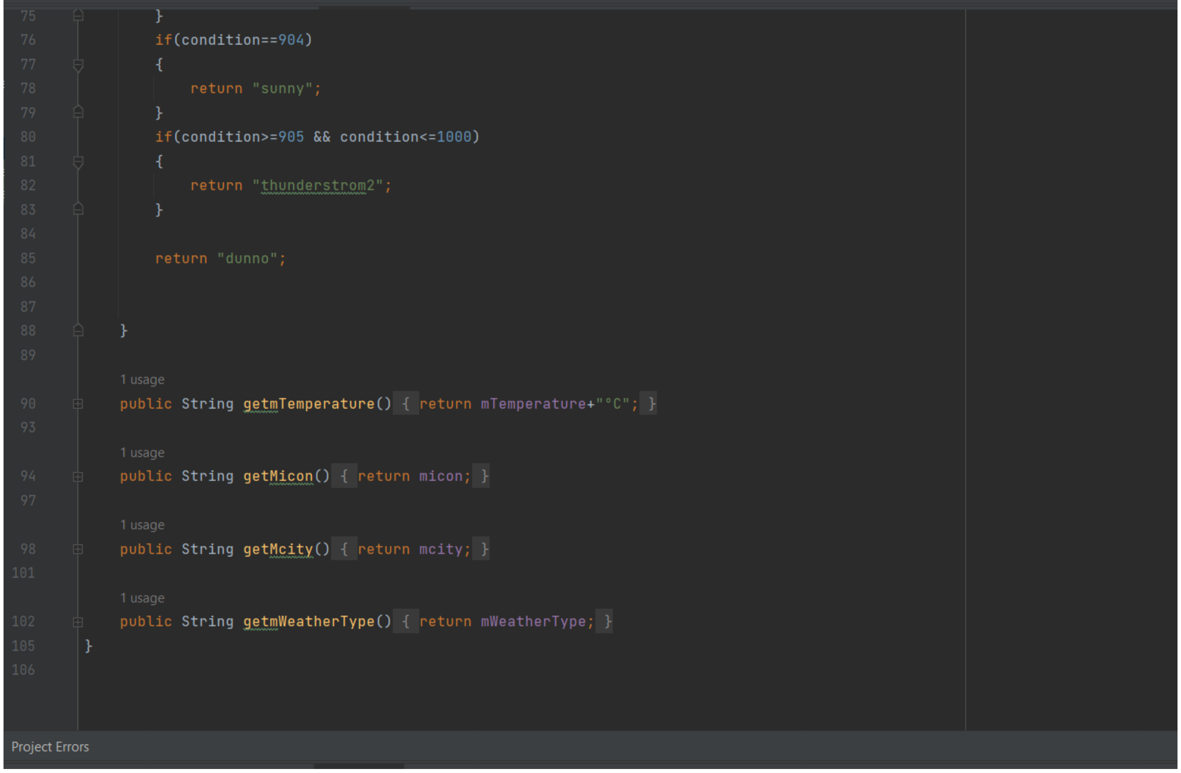
```
package com.example.weatherapptutorial;

import ...

6 usages
public class weatherData {

    2 usages
    private String mTemperature,micon,mcity,mWeatherType;
    2 usages
    private int mCondition;

    1 usage
    public static weatherData fromJson(JSONObject jsonObject)
    {

        try
        {
            weatherData weatherD=new weatherData();
            weatherD.mcity=jsonObject.getString( name: "name");
            weatherD.mCondition=jsonObject.getJSONArray( name: "weather").getJSONObject( index: 0).getInt( name: "id");
            weatherD.mWeatherType=jsonObject.getJSONArray( name: "weather").getJSONObject( index: 0).getString( name: "main");
            weatherD.micon=updateWeatherIcon(weatherD.mCondition);
            double tempResult=jsonObject.getJSONObject( name: "main").getDouble( name: "temp")-273.15;
            int roundedValue=(int)Math.rint(tempResult);
            weatherD.mTemperature=Integer.toString(roundedValue);
            return weatherD;
        }
         catch (JSONException e) {
            e.printStackTrace();
```



```
            return null;
        }


    }
    1 usage
    private static String updateWeatherIcon(int condition)
    {
        if(condition>=0 && condition<=300)
        {
            return "thunderstrom1";
        }
        else if(condition>=300 && condition<=500)
        {
            return "lightrain";
        }
        else if(condition>=500 && condition<=600)
        {
            return "shower";
        }
       else  if(condition>=600 && condition<=700)
        {
            return "snow2";
        }
        else if(condition>=701 && condition<=771)
        {
            return "fog";
        }

        else if(condition>=772 && condition<=800)
```

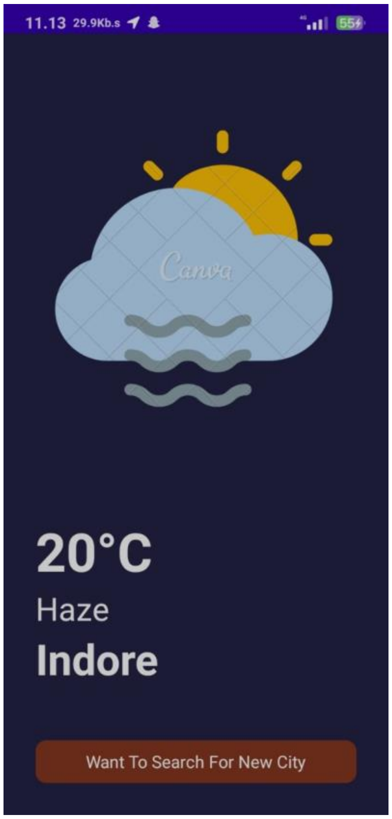


```
        {
            return "overcast";
        }
       else if(condition==800)
        {
            return "sunny";
        }
        else if(condition>=801 && condition<=804)
        {
            return "cloudy";
        }
       else  if(condition>=900 && condition<=902)
        {
            return "thunderstrom1";
        }
        if(condition==903)
        {
            return "snow1";
        }
        if(condition==904)
        {
            return "sunny";
        }
        if(condition>=905 && condition<=1000)
        {
            return "thunderstrom2";
        }

        return "dunno";

```



```
        }
        if(condition==904)
        {
            return "sunny";
        }
        if(condition>=905 && condition<=1000)
        {
            return "thunderstrom2";
        }

        return "dunno";


    }

    1 usage
    public String getmTemperature() { return mTemperature+"°C"; }

    1 usage
    public String getMicon() { return micon; }

    1 usage
    public String getMcity() { return mcity; }

    1 usage
    public String getmWeatherType() { return mWeatherType; }
}

```

## 4.6 PHASES OF APPLICATION



20°C

Haze

Indore

*Fig 4.6.1*

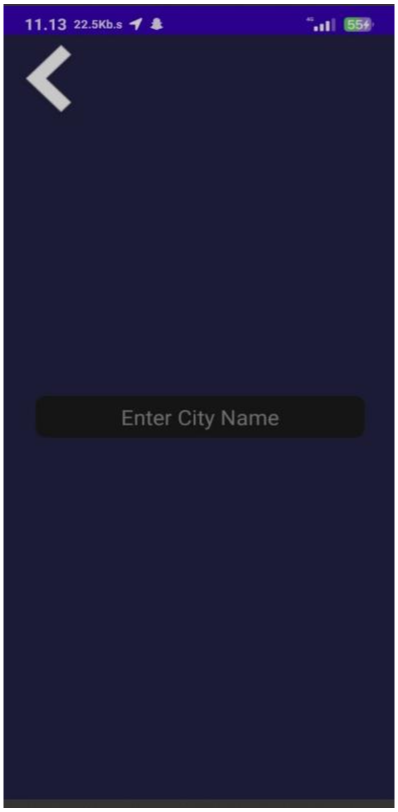
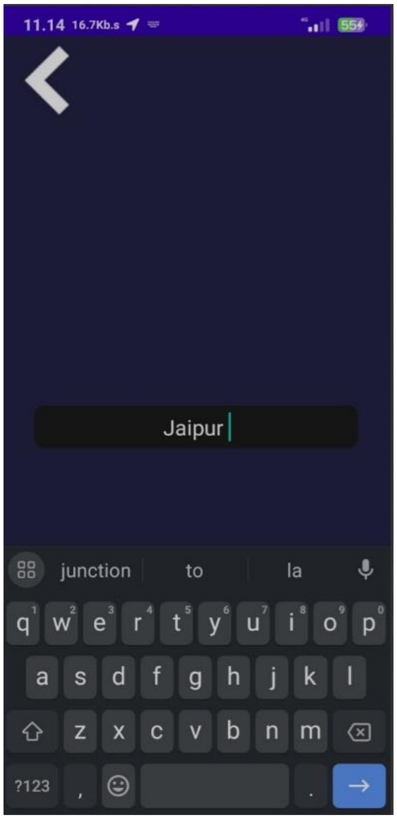
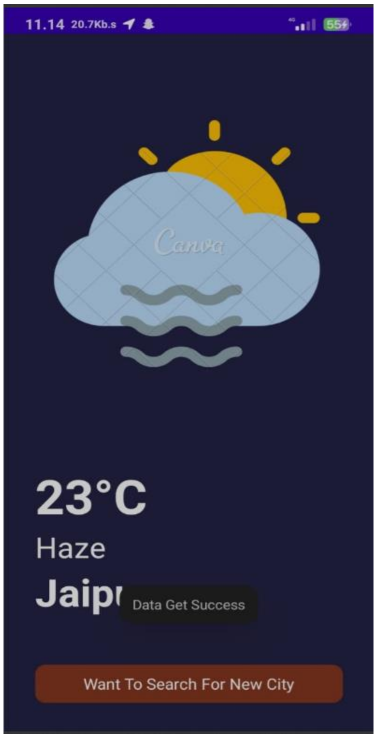


*Fig 4.6.2*



Jaipur

*Fig 4.6.3*



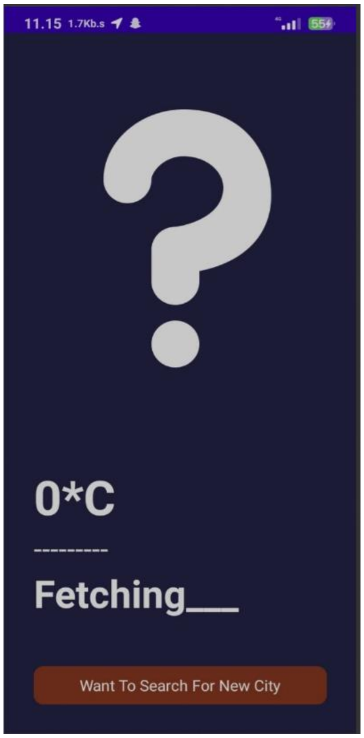
23°C

Haze

Jaipr

Data Get Success

*Fig 4.6.4*



0*C

---------

Fetching___

*Fig 4.6.5*

## 5. CONCLUSION & FUTURE SCOPE

In conclusion, the development of our weather app using Java on Android Studio has been a fulfilling and educational journey. We have successfully created a user-friendly and reliable platform that allows users to access real-time weather information, forecasts, and other relevant data with ease. This project has honed our skills in mobile app development, user interface design, and API integration. We have addressed the primary objectives of the project, providing a valuable tool for users to plan their activities, stay informed about weather conditions, and make more informed decisions. The app's intuitive design and seamless functionality make it a practical and user-centric solution for those seeking weather updates.

While the current version of our weather app is functional and user-friendly, there are several avenues for future development and improvement. First and foremost, we can explore the integration of more advanced weather prediction models and algorithms to enhance the accuracy and granularity of our forecasts. Additionally, expanding the app's coverage to include more regions and localities will make it even more appealing to a broader user base. Furthermore, implementing features like severe weather alerts, personalized weather recommendations, and social sharing of weather conditions can enhance user engagement and provide a more comprehensive weather experience. Collaborating with meteorological institutions for access to more comprehensive and reliable data can also be considered. As technology evolves, we can also explore the integration of augmented reality (AR) for a more immersive and interactive weather experience. In conclusion, the future scope for our weather app is promising, with numerous opportunities for refinement, expansion, and innovation.

## 6. BIBLIOGRAPHY

https://youtu.be/UwJumvrjncc?si=RDmqIxxtY6A-sR8h

https://youtu.be/Xi2bv01Gdqc?si=G-R0bFDHZfIf8dOv