**Weather App - Project Documentation**

# Project Overview

The Weather App is an Android application developed using **Kotlin** and **Jetpack Compose**. It allows users to enter a city name and fetch the current temperature using the **OpenWeatherMap API**.

# Technology Stack

* Kotlin (Programming Language)
* Jetpack Compose (UI Framework)
* Retrofit (For making network requests)
* Gson (For parsing JSON responses)
* OpenWeatherMap API (To get weather data)

# Application Flow Diagram

Start App

↓

User Enters City Name

↓

Clicks "Get Weather" Button

↓

Call OpenWeatherMap API with City Name + API Key

↓

If Success:

- Show Weather (Temperature, Condition)

Else:

- Show Error Message ("City not found" or "Network error")

↓

END

# Project Structure

WeatherApp/

├── app/

│ ├── manifests/

│ │ └── AndroidManifest.xml

│ ├── java/

│ │ └── com/abmn/weatherapp/

│ │ ├── MainActivity.kt

│ │ ├── WeatherScreen.kt

│ │ ├── WeatherApiService.kt

│ │ ├── WeatherResponse.kt

│ │ ├── RetrofitInstance.kt

│ │ └── Constants.kt

└── build.gradle.kts / settings.gradle.kts

# Complete Code Explanation Word by Word

## 1. Constants.kt

This file holds all constant values like the API base URL and API key.

object Constants {

const val BASE\_URL = "https://api.openweathermap.org/data/2.5/"

const val API\_KEY = "YOUR\_API\_KEY\_HERE"

}

* object keyword is used to create a singleton.
* BASE\_URL is the root URL for all API calls.
* API\_KEY is your unique key from OpenWeatherMap.

## 2. WeatherResponse.kt

This file defines the structure of the data received from the API.

data class WeatherResponse(

val name: String,

val main: Main

)

data class Main(

val temp: Double

)

* data class automatically creates getter, setter, equals, and hashCode.
* WeatherResponse represents the JSON response.
* Inside it, Main holds the temp field which is the temperature.

## 3. WeatherApiService.kt

This file defines an interface for making API requests using Retrofit.

interface WeatherApiService {

@GET("weather")

suspend fun getWeather(

@Query("q") cityName: String,

@Query("appid") apiKey: String

): Response<WeatherResponse>

}

* @GET("weather") makes a GET request to the "weather" endpoint.
* suspend fun means it is a coroutine function (runs asynchronously).
* @Query("q") and @Query("appid") add parameters to the URL.
* Response<WeatherResponse> defines the type of expected response.

## 4. RetrofitInstance.kt

This file provides a singleton object to create the Retrofit service.

object RetrofitInstance {

val api: WeatherApiService by lazy {

Retrofit.Builder()

.baseUrl(Constants.BASE\_URL)

.addConverterFactory(GsonConverterFactory.create())

.build()

.create(WeatherApiService::class.java)

}

}

* Retrofit.Builder() builds a Retrofit instance.
* .baseUrl(Constants.BASE\_URL) sets the base API URL.
* .addConverterFactory(GsonConverterFactory.create()) converts JSON to Kotlin objects.
* .create(WeatherApiService::class.java) creates an implementation of the API interface.
* by lazy ensures it's only created when needed.

## 5. WeatherScreen.kt

This file defines the UI and handles user input and API interactions.

@Composable

fun WeatherScreen() {

var cityName by remember { mutableStateOf("") }

var temperature by remember { mutableStateOf<String?>(null) }

var error by remember { mutableStateOf<String?>(null) }

val coroutineScope = rememberCoroutineScope()

Column(

modifier = Modifier

.fillMaxSize()

.padding(16.dp),

horizontalAlignment = Alignment.CenterHorizontally,

verticalArrangement = Arrangement.Center

) {

OutlinedTextField(

value = cityName,

onValueChange = { cityName = it },

label = { Text("Enter City Name") }

)

Spacer(modifier = Modifier.height(16.dp))

Button(onClick = {

coroutineScope.launch {

try {

val response = RetrofitInstance.api.getWeather(cityName, Constants.API\_KEY)

if (response.isSuccessful && response.body() != null) {

val weather = response.body()!!

val tempInCelsius = weather.main.temp - 273.15

temperature = "Temperature in ${weather.name}: ${String.format("%.2f", tempInCelsius)}°C"

error = null

} else {

temperature = null

error = "City not found or network error."

}

} catch (e: Exception) {

temperature = null

error = "City not found or network error."

}

}

}) {

Text(text = "Get Weather")

}

Spacer(modifier = Modifier.height(24.dp))

if (temperature != null) {

Text(text = temperature!!)

}

if (error != null) {

Text(text = error!!, color = MaterialTheme.colorScheme.error)

}

}

}

* @Composable defines a UI component.
* OutlinedTextField lets the user enter the city name.
* Button triggers the API call.
* CoroutineScope runs the API request asynchronously.
* Shows either the temperature or error based on API response.

## 6. MainActivity.kt

This is the entry point of the application.

class MainActivity : ComponentActivity() {

override fun onCreate(savedInstanceState: Bundle?) {

super.onCreate(savedInstanceState)

setContent {

WeatherAppTheme {

WeatherScreen()

}

}

}

}

* ComponentActivity is a basic Android Activity.
* setContent {} loads the WeatherScreen composable.
* WeatherAppTheme {} wraps it in Material3 design.

## 7. AndroidManifest.xml

This file defines app permissions and activities.

<manifest xmlns:android="http://schemas.android.com/apk/res/android"

package="com.abmn.weatherapp">

<uses-permission android:name="android.permission.INTERNET" />

<application

android:allowBackup="true"

android:icon="@mipmap/ic\_launcher"

android:label="WeatherApp"

android:roundIcon="@mipmap/ic\_launcher\_round"

android:supportsRtl="true"

android:theme="@style/Theme.WeatherApp">

<activity android:name=".MainActivity"

android:exported="true">

<intent-filter>

<action android:name="android.intent.action.MAIN" />

<category android:name="android.intent.category.LAUNCHER" />

</intent-filter>

</activity>

</application>

</manifest>

* <uses-permission android:name="android.permission.INTERNET" /> allows the app to access the internet.
* Defines MainActivity as the app launcher.

# How the App Works (Step-by-Step)

1. App starts and displays an input field.
2. User types a city name.
3. User clicks "Get Weather" button.
4. App sends an API request to OpenWeatherMap.
5. If city found, temperature is displayed.
6. If city not found, error is shown.

# Final Output Example

* **Input**: "London"
* **Output**: "Temperature in London: 13.45°C"

# Future Improvements

* Add weather icons based on condition (rain, sunny, etc.).
* Show more data (humidity, pressure, wind speed).
* Add location detection.
* Implement dark mode support.

# Built with ❤️ using Kotlin + Jetpack Compose