**WeatherApp: Detailed Report**

**1. Introduction**

The **WeatherApp** is an Android application developed using **Jetpack Compose** for UI and **Retrofit** for network communication. The app fetches historical weather data using the **WeatherAPI** and displays the maximum and minimum temperatures for a specified date and year.

**2. Technologies Used**

* **Kotlin** (for Android development)
* **Jetpack Compose** (for UI design)
* **Retrofit** (for API calls)
* **Coroutines** (for asynchronous operations)
* **Gson** (for JSON parsing)
* **OpenWeatherMap API** (for fetching historical weather data)

**3. Application Structure**

**3.1 Manifest Configuration**

The AndroidManifest.xml includes:

* Internet permission: <uses-permission android:name="android.permission.INTERNET" />
* MainActivity declaration with intent filter for launcher category

**3.2 UI Implementation (MainActivity.kt)**

The MainActivity.kt contains a composable function WeatherApp() which provides:

* Input fields for **Date** and **Year**
* A **Button** to fetch weather data
* **Text views** to display max and min temperature or an error message

**3.3 API Integration**

The app uses **Retrofit** for API calls and follows an MVVM-like structure:

**RetrofitInstance.kt**

* Sets up the Retrofit client with **GsonConverterFactory**
* Base URL: http://api.weatherapi.com/v1
* Creates a singleton instance of WeatherApiService

**WeatherApiService.kt**

Defines a GET request method for fetching historical weather data:

@GET("history.json")

suspend fun getHistoricalWeatherData(

@Query("date") date: String,

@Query("year") year: String,

@Query("key") apiKey: String = API\_KEY

): Response<WeatherResponse>

**WeatherRepository.kt**

* Calls WeatherApiService to fetch data
* Handles HTTP and network exceptions
* Returns a WeatherData object containing max and min temperatures

**4. Data Models**

**WeatherResponse.kt**

Models the API response structure:

data class WeatherResponse(

@SerializedName("forecast") val forecast: ForecastData

)

data class ForecastData(

@SerializedName("forecastday") val forecastDay: List<ForecastDayData>

)

data class ForecastDayData(

@SerializedName("date") val date: String,

@SerializedName("day") val day: DayData,

@SerializedName("astro") val astro: AstroData

)

data class DayData(

@SerializedName("maxtemp\_c") val maxtempC: String,

@SerializedName("mintemp\_c") val mintempC: String

)

data class AstroData(

@SerializedName("sunrise") val sunrise: String,

@SerializedName("sunset") val sunset: String

)

**5. Error Handling**

The app handles various errors such as:

* **HTTP Errors** (e.g., 404 Not Found, 500 Server Error)
* **Network Issues** (IOException for no connectivity)
* **Invalid Responses** (null or missing data)
* **User Input Errors** (empty fields, invalid date format)

**6. API Key Configuration**

The API key is stored in WeatherApiService:

companion object {

const val API\_KEY = "965c14f6f9f3445b8024c27313466bb8"

}

It should be securely managed using **BuildConfig** or environment variables.

**7. Future Enhancements**

* **UI Improvements**: Add animations, themes, and better layouts.
* **Caching**: Store previous API responses for offline access.
* **Additional Weather Details**: Include humidity, wind speed, and precipitation.
* **Location-based Weather**: Fetch current location using GPS.

**8. Conclusion**

The **WeatherApp** provides a clean and functional interface to retrieve historical weather data using an API. It follows best practices in Android development with **Jetpack Compose**, **Retrofit**, and **Coroutines**, ensuring a smooth user experience. Future improvements will enhance its capabilities, making it more robust and user-friendly.