MyRadar – A Modern Android Weather Application

1. Introduction

Weather information is essential in daily life, from planning outdoor activities to travel and disaster preparedness. Traditional weather apps often require API keys or come with heavy interfaces.

MyRadar was developed to provide:

- A **lightweight**, **free-to-use** weather application.
- A clean, modern UI built with **Jetpack Compose**.
- A scalable codebase using **MVVM architecture** and **Dependency Injection (Hilt)**.
- Instant access to **current weather** and **short-term forecasts** via the **Open-Meteo API** (which requires no authentication).

2. Objectives

The main objectives of the project were:

1. To design and develop a **functional Android weather app** using **modern Android frameworks**.
2. To implement **MVVM (Model-View-ViewModel)** for maintainability and separation of concerns.
3. To integrate **Retrofit with Coroutines** for efficient API calls.
4. To apply **Hilt (Dependency Injection)** for clean dependency management.
5. To present weather data using **Jetpack Compose** with Material 3 styling.
6. To ensure the solution is **extendable** for future features such as caching, location services, and weather icons.

3. System Design

3.1 Architecture

The project follows the **MVVM pattern**:

- **Model Layer (Data):**
- `WeatherRepository` fetches and prepares weather data.
- Retrofit interfaces (`GeocodingApi`, `ForecastApi`) handle network requests.
- **ViewModel Layer:**
- `WeatherViewModel` exposes `StateFlow` for the UI to observe.
- Manages state transitions (Idle → Loading → Success/Error).
- **View Layer (UI):**

- Built entirely with **Jetpack Compose**.
- Composables render search input, current conditions, and forecast cards.

3.2 Dependencies

- **Retrofit + Gson + OkHttp** for networking.
- **Coroutines** for asynchronous operations.
- **Hilt** for dependency injection.
- **Jetpack Compose** for UI.
- **Material 3** for theming.

4. Implementation

4.1 Data Sources

- **Open-Meteo Geocoding API:**
  Converts city names into latitude/longitude.
  Example:

```
https://geocoding-api.open-meteo.com/v1/search?name=Mumbai&count=1
```

- **Open-Meteo Forecast API:**
  Provides current weather and daily forecasts.
  Example:

```
https://api.open-
meteo.com/v1/forecast?latitude=19.07&longitude=72.87&current=...
```

4.2 Core Features

- **City Search:** User inputs a city, which is geocoded.
- **Weather Fetch:** Repository fetches forecast data based on coordinates.
- **UI Binding:** StateFlow ensures real-time UI updates.
- **Forecast View:** Next days shown in horizontally scrollable cards.

4.3 Error Handling

- Invalid city → “City not found.”
- Network failure → Graceful error message.
- Empty input → No API call triggered.

5. Results

The final application, **MyRadar**, successfully meets all objectives:

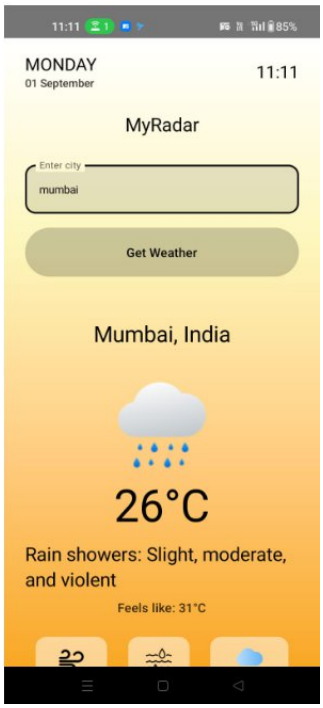
- Fetches real-time weather for any city worldwide.

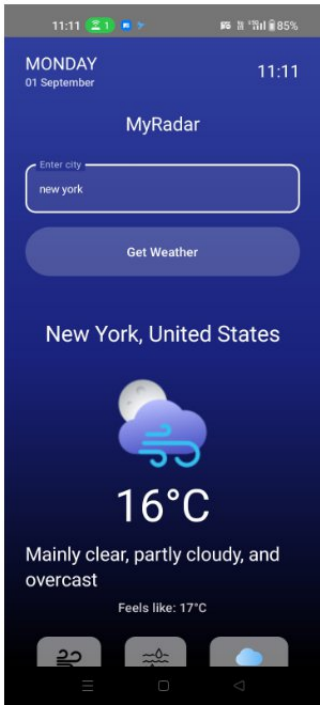
- Displays **current temperature, feels-like, humidity, wind speed, and conditions**.
- Shows **daily forecast** in a user-friendly card layout.
- Implements a **reactive, responsive UI** with Jetpack Compose.
- Achieves a **clean architecture** with MVVM and DI.

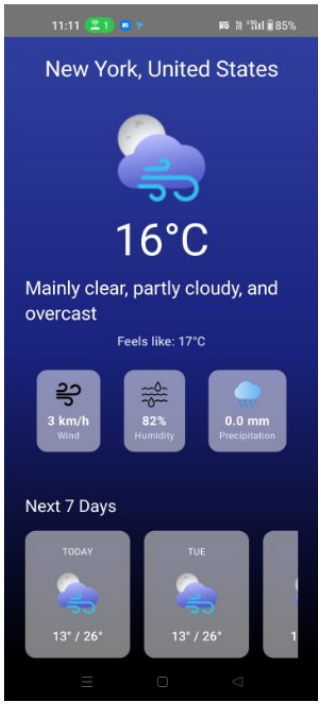
6. Features Snapshot

- **Enter a city → Get weather instantly**.
- **Now screen**: current conditions in bold, readable format.
- **Forecast section**: scrollable daily forecast cards.
- **Material 3 UI** for consistent theming.
- **Lightweight and fast**, with no login or API keys.

7. Screenshots







8. Limitations & Future Enhancements

Current Limitations

- No offline caching (requires active internet).
- Minimal visuals (text-based weather codes).

Future Enhancements

1. **Location Auto-Detect** – Fetch weather for user's current GPS location.
2. **Room Database Caching** – Store last fetched weather for offline use.
3. **Charts/Graphs** – Visualize temperature trends.
4. **Unit Tests** – Add coverage for Repository & ViewModel.

9. Conclusion

MyRadar demonstrates how to build a **modern, efficient, and scalable Android weather application** using:

- Jetpack Compose for declarative UI,
- MVVM for separation of concerns,
- Hilt for dependency injection, and
- Retrofit + Coroutines for smooth networking.

The app is lightweight, extendable, and free from API key restrictions, making it both **user-friendly** and **developer-friendly**.