**Weather‑Forecast**

A weather forecasting application that shows current weather and forecast for a user‑specified location.

**Table of Contents**

1. Introduction
2. Features
3. Technologies & Dependencies
4. Architecture & Module Structure
5. Setup & Installation
6. Configuration
7. Usage & Examples
8. API / Data Sources
9. Error Handling & Validation
10. Limitations & Future Improvements
11. Contributing
12. License

**1. Introduction**

**Weather‑Forecast** is a project that allows users to fetch weather data (current, forecasts) based on a location (city name, coordinates, etc.). It integrates with a third‑party weather API, processes the data, and displays the results in a user‑friendly format (web / CLI / UI, depending on your implementation).

The goal is to provide accurate, timely, and clean weather information, with a focus on usability and stability.

**2. Features**

Here are the key features your project might include (modify to fit what you actually implemented):

* Get **current weather** (temperature, humidity, wind, etc.)
* Get **forecast** (e.g. next 5 days, hourly)
* Support for **search by city name** or **geolocation / coordinates**
* Unit switching (°C / °F)
* Beautiful / responsive UI (if web)
* Error handling (invalid city, network error)
* Caching or limiting API calls to avoid exceeding rate limits
* Clean UI / CLI output

**3. Technologies & Dependencies**

List the languages, frameworks, and libraries your project uses.

Example:

| **Layer** | **Technology / Library** | **Purpose** |
| --- | --- | --- |
| Frontend / UI | HTML / CSS / JavaScript (React / Vue / Vanilla) | To build web interface |
| Backend / Logic | Python / Node.js / Java | Weather data fetching & processing |
| HTTP client | axios, fetch, requests | For making API calls |
| Environment / Config | dotenv (Node) / python-dotenv | For API keys, environment variables |
| Build tools | webpack, Babel, or similar | To bundle / transpile code |
| Testing (optional) | Jest, Mocha, Pytest, etc. | For unit / integration tests |

Also, mention minimum versions (e.g. Node 14+, Python 3.8+).

**4. Architecture & Module Structure**

Explain how your code is organized (folders / modules) and how they interact.

Example:

/Weather‑Forecast

│

├── /src

│ ├── api/

│ │ └── weatherClient.js # logic to call weather API

│ ├── components/ # UI components (if web)

│ ├── utils/

│ │ └── helpers.js # utility functions, formatting, unit conversions

│ ├── views/ # pages or view templates

│ └── index.js # main entry point

│

├── config/

│ └── default.env # environment template

│

├── tests/ # test cases

│

├── package.json (or requirements.txt)

└── README.md

You should also explain the flow:

1. User inputs location (city or lat/lon)
2. The UI calls your **weatherClient**
3. weatherClient sends HTTP request to the external weather API
4. Response JSON parsed & transformed into internal data model
5. UI or CLI renders the weather data
6. Errors handled gracefully

If you have caching, rate limiting, or retry logic, mention where in the flow that lies.

**5. Setup & Installation**

Step‑by‑step instructions so someone can clone and run your project locally.

Example:

# 1. Clone the repo

git clone https://github.com/irfan-khan12/Weather-Forecast.git

cd Weather-Forecast

# 2. Install dependencies

# If using Node.js

npm install

# If using Python

pip install -r requirements.txt

If there is a build step:

npm run build

**6. Configuration**

Explain how the user sets up API keys or other configuration.

* Sign up with the weather service provider (e.g. OpenWeatherMap) and obtain an API key
* Create a .env file (or config.json) in project root:

API\_KEY=your\_api\_key\_here

DEFAULT\_UNIT=metric

DEFAULT\_CITY=Kurnool

* (If applicable) mention environment variables used, e.g.:
  + API\_KEY — your weather API key
  + PORT — server port (if web back end)
  + UNITS — default unit (metric or imperial)

Mention that **do not commit** your .env file (add to .gitignore).

**7. Usage & Examples**

Show how to run and use the project (command line or via browser).

**Web / UI usage**

1. Start the server (if needed):
2. npm start
3. Open browser at http://localhost:3000 (or whichever port)
4. Enter a city name (e.g. “Hyderabad”) and click **Search**
5. The app shows current weather and forecast

Show sample screenshots or sample JSON responses.

**Command-line usage (if applicable)**

python weather.py --city "Kurnool" --units metric

Output:

City: Kurnool

Temperature: 32.5 °C

Humidity: 75%

Wind: 3.2 m/s

Forecast:

Wed: min 28 °C — max 34 °C, rain chance 40%

**8. API / Data Sources**

Document which weather API(s) you use, endpoints, and data format.

Example:

* **API provider**: OpenWeatherMap
* **Endpoints used**:
  + GET https://api.openweathermap.org/data/2.5/weather?q={city}&appid={API\_KEY}&units={units} — for current weather
  + GET https://api.openweathermap.org/data/2.5/forecast?q={city}&appid={API\_KEY}&units={units} — for forecast
* **Request parameters**:

| **Parameter** | **Description** |
| --- | --- |
| q | City name (e.g. “Kurnool,IN”) |
| appid | API key |
| units | metric or imperial |

* **Sample JSON Response**:

{

"weather": [

{

"id": 800,

"main": "Clear",

"description": "clear sky",

"icon": "01d"

}

],

"main": {

"temp": 30.5,

"feels\_like": 32.0,

"humidity": 70

},

"wind": {

"speed": 3.5

},

"name": "Kurnool"

}

Explain how you map fields (e.g. weather[0].description → “clear sky”).

**9. Error Handling & Validation**

Describe how your project handles errors, invalid input, or API failures.

* If the user enters a non-existent city, show a friendly message: “City not found, please try again.”
* If network error / timeout, show message: “Could not fetch data, please check your internet.”
* Validate inputs (e.g. reject empty strings)
* Use try/catch (or promise .catch) blocks around HTTP calls
* Possibly retry logic or fallback defaults

Mention the cases you tested (offline, invalid key, rate limit exceeded).

**10. Limitations & Future Improvements**

No project is perfect. It’s good to acknowledge limitations and propose enhancements.

**Current limitations**

* API usage is limited by rate limits
* No offline caching (if not implemented)
* Only supports a single weather provider
* Forecast granularity may be coarse (every 3 hours, daily)
* UI may not handle long city names or edge cases

**Future improvements**

* Add **caching / local storage** to reduce API calls
* Support multiple weather APIs (fallback)
* Add **geolocation** (auto‑detect user location)
* Add **more forecast types**, e.g. hourly, weekly, alerts
* Improve UI / themes / animations
* Add **unit tests / integration tests**
* Deploy to cloud / make live (Netlify / Vercel / AWS)

**11. Contributing**

How others can help improve your project:

* Fork the repository
* Create a new branch (git checkout -b feature/your-feature)
* Make changes, write tests
* Push and open a Pull Request
* Ensure coding style, format, linting
* Provide clear commit messages
* Report issues / bugs

**12. License**

State the license under which your project is released (e.g. MIT, Apache, GPL). Example:

MIT License

Copyright (c) 2025 [Your Name]

Permission is hereby granted, free of charge, to any person obtaining a copy

...

Place a LICENSE file in the repository root.