Weather Forecast App

1. Software Requirements Specification (SRS)

1.1 Introduction

- **Purpose**: The Weather Forecast App provides real-time weather information using the OpenWeatherMap API, including temperature, humidity, wind speed, sunrise/sunset times, and other relevant weather data. This app is designed to be a simple, user-friendly platform to check weather details based on the city name.
- **Scope**: The app will display weather data for any city, given its name, and will show data such as current weather, temperature, humidity, wind speed, sunrise and sunset times, and location coordinates. The data will be updated in real-time using the OpenWeatherMap API.
- **Objectives**: The goal is to build a scalable, secure, and interactive web app that provides weather details based on user input. The app will be developed using Django and hosted on a cloud platform (e.g., Render).

1.2 Overall Description

- **System Context**: The app fetches weather information from the OpenWeatherMap API using city names. The front-end interface allows users to search for weather information, and the back-end processes the API calls.
- **Constraints**:
  - The app must be responsive and mobile-friendly.
  - The app should handle invalid city names gracefully, showing an error message.
- **Assumptions**:
  - Users will have internet access to fetch weather data.
  - The OpenWeatherMap API key is valid and available.

1.3 Functional Requirements

1. **City Search**: The user must be able to input the name of a city to fetch weather data.

2. **Weather Data**: The app should display current weather conditions, including temperature, humidity, wind speed, and a brief description.
3. **Sunrise/Sunset**: Display the sunrise and sunset times in UTC.
4. **Error Handling**: If a city name is incorrect or the data cannot be fetched, an error message should be shown.
5. **Responsive UI**: The interface should adapt to various screen sizes and provide an intuitive experience.

1.4 Non-Functional Requirements

1. **Performance**: The app should provide weather data within a few seconds after the user enters a city name.
2. **Usability**: The interface should be easy to use with clear instructions and labels.
3. **Security**: Ensure the API key is securely stored and not exposed in the codebase.

1.5 Appendix

- **Glossary**:
    - **API**: Application Programming Interface, used to fetch weather data.
    - **UTC**: Coordinated Universal Time, used to show sunrise and sunset times.

---

2. High-Level Design (HLD)

2.1 Introduction

- The system will be a Django-based web app integrated with the OpenWeatherMap API to fetch weather data and display it in a user-friendly format.

2.2 Architecture Diagram

lua

Copy code

```
+-------------+    +---------------+    +---------------------+

|  Frontend  | <-- |  Django App  | <-- |  OpenWeatherMap API  |

+-------------+    +---------------+    +---------------------+

     |              |
      v              v
+------------------------------------+
|    Render Hosting & Database      |
+------------------------------------+
```

2.3 Module Descriptions

- **Frontend (HTML, CSS, JS)**: Displays the weather information and handles user input.
- **Django Backend**: Fetches weather data from the OpenWeatherMap API and serves it to the frontend.
- **OpenWeatherMap API**: Provides weather data in JSON format.

2.4 Database Design

The app does not require a persistent database for storing weather data, as the data is fetched in real-time from the OpenWeatherMap API.

---

3. Low-Level Design (LLD)

3.1 Introduction

The low-level design outlines the structure of the app's components, detailing their interactions, flow, and data management.

3.2 Flowcharts / Sequence Diagrams

- **Weather Data Flow**:
    1. The user enters a city name in the search bar.
    2. The backend sends a request to the OpenWeatherMap API using the entered city name.
    3. The backend receives the weather data and sends it to the frontend.
    4. The frontend displays the weather data, including temperature, humidity, wind speed, and sunrise/sunset times.

3.3 Class and Entity-Relationship Diagrams

- **WeatherInfo Class**:
    - city_name: String
    - temperature: Float

    - humidity: Integer
    - wind_speed: Float
    - sunrise: Integer (timestamp)
    - sunset: Integer (timestamp)
    - description: String

3.4 Algorithms and Pseudocode

- **Fetching Weather Data**:

python

Copy code

```
def get_weather_data(city_name):
  try:
    response =
requests.get(f'https://api.openweathermap.org/data/2.5/weather?q={city_name}&appi
d={API_KEY}')
    data = response.json()
    return data
  except Exception as e:
    return {"error": str(e)}
```

---

4. Test Plan

4.1 Introduction

This test plan covers the testing strategy for the Weather Forecast App, including functional, security, and usability tests.

4.2 Test Scope and Objectives

- **Features to be tested**: City search functionality, API response, error handling, UI responsiveness.
- **Features not to be tested**: Integration with other third-party services.

4.3 Test Approach

- **Unit Testing**: Test individual functions like API calls.

- **Integration Testing**: Test the integration between the frontend, Django backend, and OpenWeatherMap API.
- **Usability Testing**: Test the ease of use of the user interface.

---

5. Test Case Template

5.1 Test Case Example

- **Test Case ID**: TC001
- **Description**: Test the weather data retrieval for a valid city.
- **Preconditions**: The app is running, and the user is on the homepage.
- **Steps to Execute**:
    1. Enter a valid city name in the search bar (e.g., "New York").
    2. Press the "Get Weather" button.
- **Expected Result**: Weather data for New York is displayed.
- **Actual Result**: [To be filled during testing]
- **Status**: Pass/Fail

---

6. Deployment Plan

6.1 Deployment Strategy

- The app will be deployed on Render.com using Gunicorn to serve the Django application.

6.2 Pre-Deployment Requirements

- Create an account on Render.com.
- Obtain a valid OpenWeatherMap API key.
- Set environment variables for sensitive information (e.g., API key).

6.3 Deployment Steps

1. Set up a GitHub repository for the app.
2. Link the repository to Render.com.
3. Configure Render's settings to use Gunicorn with the Procfile.
4. Deploy the app.

6.4 Rollback Plan

If any errors occur during deployment, rollback to the previous stable version in the GitHub repository.

---

7. API Documentation

7.1 Endpoint Description

- **GET /weather**: Retrieves the weather data for a given city.
    - **URL**: /weather?q={city_name}&appid={API_KEY}
    - **Parameters**: q (city name), appid (OpenWeatherMap API key).
    - **Response**:

json

Copy code

```json
{
  "temperature": 25.0,
  "humidity": 60,
  "wind_speed": 5.0,
  "sunrise": 1633544800,
  "sunset": 1633582800
}
```

---

8. Traceability Matrix

8.1 Mapping Requirements to Test Cases

Requirement ID	Requirement Description	Test Case ID	Test Case Description
REQ001	City search functionality	TC001	Test weather data retrieval for a valid city
REQ002	API response handling	TC002	Test API failure handling for an invalid city

---

9. Security Test Plan and Report

9.1 Introduction

This section outlines the security testing approach to ensure the app is secure.

9.2 Security Requirements

- Ensure the OpenWeatherMap API key is securely stored and not exposed in the codebase.
- Validate user inputs to prevent malicious data injection.

9.3 Testing Tools

- **OWASP ZAP**: Used for testing security vulnerabilities in the app.

9.4 Test Scenarios

- Test for **SQL Injection** in case of any form submissions.
- Test for **XSS (Cross-Site Scripting)** vulnerabilities in the search input fields.