**Weather Forecasting API Testcases** :

1. Request and Response Validation

| **Test Case ID** | **Scenario** | **Request** | **Expected Response** | **Status Code** |
| --- | --- | --- | --- | --- |
| TC\_01 | Valid request with city and date | { "city": "New York", "date": "2025-03-19" } | { "max\_temp": 20, "min\_temp": 10, "humidity": 60 } | 200 OK |
| TC\_02 | Missing city parameter | { "date": "2025-03-19" } | { "error": "City is required" } | 400 Bad Request |
| TC\_03 | Missing date parameter | { "city": "New York" } | { "error": "Date is required" } | 400 Bad Request |
| TC\_04 | Invalid city name (non-existent) | { "city": "XYZ123", "date": "2025-03-19" } | { "error": "City not found" } | 404 Not Found |
| TC\_05 | Invalid date format | { "city": "New York", "date": "03-19-2025" } | { "error": "Invalid date format, use YYYY-MM-DD" } | 400 Bad Request |
| TC\_06 | Future date beyond supported range | { "city": "New York", "date": "2035-03-19" } | { "error": "Forecast unavailable for the given date" } | 422 Unprocessable Entity |
| TC\_07 | Past date (historical data not available) | { "city": "New York", "date": "2020-03-19" } | { "error": "Historical data not available" } | 422 Unprocessable Entity |

2. Error Code Handling

| **Test Case ID** | **Scenario** | **Expected Response** | **Status Code** |
| --- | --- | --- | --- |
| TC\_08 | Empty request body | { "error": "Request body is required" } | 400 Bad Request |
| TC\_09 | Unauthorized access (API key missing/invalid) | { "error": "Unauthorized access" } | 401 Unauthorized |
| TC\_10 | Server error simulation | { "error": "Internal server error" } | 500 Internal Server Error |
| TC\_11 | Request limit exceeded | { "error": "Too many requests" } | 429 Too Many Requests |

3. Data Accuracy Verification

| **Test Case ID** | **Scenario** | **Validation Method** |
| --- | --- | --- |
| TC\_12 | Compare API response with another reliable weather source | Validate API response against third-party sources like OpenWeatherMap, Weather.com |
| TC\_13 | Ensure the temperature is within realistic range | Validate temperatures fall between -50°C and 60°C |
| TC\_14 | Check humidity range | Humidity values should be between 0 and 100% |

4. Automated Test Script (Postman)

// Parse the response

**let** jsonData = pm.response.json();

// Check for successful response

pm.test("Status code is 200", **function** () {

    pm.response.to.have.status(200);

});

// Check if response contains required fields

pm.test("Response has temp\_min, temp\_max, and humidity", **function** () {

    pm.expect(jsonData.main).to.have.property("temp\_min");

    pm.expect(jsonData.main).to.have.property("temp\_max");

    pm.expect(jsonData.main).to.have.property("humidity");

});

// Validate temperature range (-50°C to 60°C)

pm.test("Temperature values are realistic", **function** () {

    pm.expect(jsonData.main.temp\_max).to.be.within(-50, 60);

    pm.expect(jsonData.main.temp\_min).to.be.within(-50, 60);

});

// Validate humidity range (0% to 100%)

pm.test("Humidity is in valid range", **function** () {

    pm.expect(jsonData.main.humidity).to.be.within(0, 100);

});

// Test Case: Missing City Parameter

**if** (pm.request.body) {

**let** reqBody = JSON.parse(pm.request.body.raw);

**if** (!reqBody.city) {

        pm.test("Error when city is missing", **function** () {

            pm.response.to.have.status(400);

            pm.expect(jsonData.error).to.eql("City is required");

        });

    }

}

// Test Case: Missing Date Parameter

**if** (pm.request.body) {

**let** reqBody = JSON.parse(pm.request.body.raw);

**if** (!reqBody.date) {

        pm.test("Error when date is missing", **function** () {

            pm.response.to.have.status(400);

            pm.expect(jsonData.error).to.eql("Date is required");

        });

    }

}

// Test Case: Invalid Date Format

pm.test("Error when date format is incorrect", **function** () {

**if** (pm.request.body) {

**let** reqBody = JSON.parse(pm.request.body.raw);

**if** (reqBody.date && !/^\d{4}-\d{2}-\d{2}$/.test(reqBody.date)) {

            pm.response.to.have.status(400);

            pm.expect(jsonData.error).to.eql("Invalid date format, use YYYY-MM-DD");

        }

    }

});

// Test Case: Future Date Beyond Supported Range

pm.test("Error for forecast beyond supported range", **function** () {

**if** (pm.request.body) {

**let** reqBody = JSON.parse(pm.request.body.raw);

**if** (reqBody.date && **new** Date(reqBody.date) > **new** Date("2030-01-01")) {

            pm.response.to.have.status(422);

            pm.expect(jsonData.error).to.eql("Forecast unavailable for the given date");

        }

    }

});

// Test Case: Past Date (Historical Data Not Available)

pm.test("Error when requesting past weather data", **function** () {

**if** (pm.request.body) {

**let** reqBody = JSON.parse(pm.request.body.raw);

**if** (reqBody.date && **new** Date(reqBody.date) < **new** Date("2024-01-01")) {

            pm.response.to.have.status(422);

            pm.expect(jsonData.error).to.eql("Historical data not available");

        }

    }

});

// Test Case: Invalid City Name

pm.test("Error for invalid city name", **function** () {

**if** (pm.request.body) {

**let** reqBody = JSON.parse(pm.request.body.raw);

**if** (reqBody.city === "XYZ123") {

            pm.response.to.have.status(404);

            pm.expect(jsonData.error).to.eql("City not found");

        }

    }

});

// Test Case: Unauthorized Access (API Key Missing)

pm.test("Unauthorized access when API key is missing", **function** () {

**if** (!pm.request.headers.has("Authorization")) {

        pm.response.to.have.status(401);

        pm.expect(jsonData.error).to.eql("Unauthorized access");

    }

});

// Test Case: Rate Limit Exceeded

pm.test("Too many requests error", **function** () {

**if** (pm.response.code === 429) {

        pm.expect(jsonData.error).to.eql("Too many requests");

    }

});

// Test Case: Server Error

pm.test("Internal Server Error Handling", **function** () {

**if** (pm.response.code === 500) {

        pm.expect(jsonData.error).to.eql("Internal server error");

    }

});