Sprint 2: Initial Setup

1. System Overview

The WeatherSensor AI aims to provide users with real-time weather information enhanced by natural-language summaries generated through a large language model.

2. Functional Requirements

User Story: As a user, I want the application to retrieve real-time weather data, handle errors smoothly, and generate a clear, natural-language summary so I can quickly understand current weather conditions without reading raw data.

2.1 - Format/Structure Weather API Information – Jackson Yanek

- FR 2.1.1: Parse API response to pull out key weather values (temperature, wind, humidity, etc.).
- FR 2.1.2: Implement key weather values into the website.
- FR 2.1.3: Store extracted data in a clean, labeled JSON format.

Progress Summary: Completed all Function Requirements

2.2 - Implement Error Handling for Weather API – Jackson Yanek

- FR 2.2.1: Detect and report API request errors (timeouts, bad responses).
- FR 2.2.2: Print clear messages for missing or invalid data.
- FR 2.2.3: Add simple retry logic if a request fails once.

Progress Summary: Most errors are handled correctly, need to revise and refine them now.

2.3 - Add Validation for Missing or Invalid Data Values – Jackson Yanek

- FR 2.3.1: Reject out-of-range values.
- FR 2.3.2: Replace invalid/missing values with null and tag a validation note.
- FR 2.3.3: Emit a single, human-readable validation summary for logs.

Progress Summary: Most errors are handled correctly, need to revise and refine them now.

2.4 - Test Cases / Verify Correctness of API Responses – Riley England

- FR 2.4.1: Create test cases to check that valid data is parsed correctly.
- FR 2.4.2: Test with both normal and error responses.
- FR 2.4.3: Confirm output matches expected weather values.
- FR 2.4.4: Collect group feedback and adjust the prompt or parameters as needed.

Progress Summary: Created test cases for both pass and fail cases, executed them on the website, and took notes of behavior for each test case.

2.5 - Choose LLM API and Configure LLM Provider – Manu Redd

- FR 2.5.1: Research available LLM APIs and select one (e.g., Gemini).
- FR 2.5.2: Set up environment variables for API key and model name.
- FR 2.5.3: Verify connection with a simple test prompt.

Progress Summary: I researched the most optimal gemini models for our use-case and learned how to use an API key for sending requests for text generation. I used sample code from google and the API requests are working.

2.6 - Verify Connection with Test Prompt Response – Manu Redd

- FR 2.6.1: Send a minimal test prompt and confirm non-empty LLM reply.
- FR 2.6.2: Log model name, latency, tokens (if available).

Progress Summary: Executed a test prompt through Google Gemini and confirmed valid response. Model version, token usage, and latency were recorded for future optimization.

2.7 - Design Prompt Structure for Weather Summary – Evans Chigweshe

- FR 2.7.1: Write a basic prompt format that includes weather data fields.
- FR 2.7.2: Keep the prompt concise and easy to read.

- FR 2.7.3: Add examples for different weather types (clear, rain, storm).

Progress Summary:

2.8 - Create Example Prompts for Different Weather Types – Evans Chigweshe

- FR 2.8.1: Create examples for clear, rain, storm, heat, cold.
- FR 2.8.2: Store prompts alongside expected tone.

Progress Summary:

2.9 - Setup of Google Gemini API Pipeline –

- FR 2.9.1: Install and configure Google Gemini API client.
- FR 2.9.2: Build a function to send prompts and receive responses.
- FR 2.9.3: Log the results and confirm that output text is returned.

Progress Summary:

2.10 - Implement Summary within Gemini, Return Generated Text –

- FR 2.10.1: Combine extracted weather data with the summary prompt.
- FR 2.10.2: Generate a text summary through Gemini and display it.
- FR 2.10.3: Test several examples to confirm correct output.

Progress Summary:

2.11 - Return formatted summary string to console –

- FR 2.11.1: Print a single-paragraph summary ≤ 120 words.
- FR 2.11.2: Ensure only imperial units appear in the output text.

Progress Summary:

2.12 - Verify Summary Quality with Test Cases – Riley England

- FR 2.12.1: Create sample weather inputs (clear, rain, storm, extreme heat) to test summaries.
- FR 2.12.2: Check that each summary accurately reflects key weather values (temperature, wind, precipitation).
- FR 2.12.3: Review summaries for clarity, grammar, and tone consistency.

- FR 2.12.4: Compare generated text to a short “ideal” reference summary for quality scoring.
- FR 2.12.5: Collect group feedback and adjust the prompt or parameters as needed.

Progress Summary:

2.13 - Create and Complete Sprint 1 Artifacts - Riley England

- FR 2.13.1: Compile Requirements for Sprint 2 into formatted documentation.
- FR 2.13.2: Write functional requirements for each requirement, leaving space for progress summaries.
- FR 2.13.3: Verify artifact alignment with course rubric and project objectives.
- FR 2.13.4: Upload all Sprint 2 documents to github.

Progress Summary: Compiled all Sprint 2 Requirements into the specified format of the Artifacts document. Broke each requirement down into its Functional Requirements, and gave a description for each. Finally, compiled all Sprint 2 deliverables and uploaded them to the github, ensuring the final version was pushed.