Sprint 1: Initial Setup

Requirements Artifact for WeatherSensor AI

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 fetch and display current weather data for a specific location to the console.

   1.1 - Define System Architecture - Evans Chigweshe

   - FR 1.1.1: The system architecture documentation shall provide a synopsis of the project.
   - FR 1.1.2: The system architecture documentation shall describe an overview of the system.
   - FR 1.1.3: The system architecture documentation shall provide a summary of the three layers.
   - FR 1.1.4: The system architecture documentation shall describe the data flow when a user interacts with the site.
   - FR 1.1.5: The system architecture documentation shall describe the component architecture with descriptions for each.
   - FR 1.1.6: The system architecture documentation shall provide UML Diagrams for Use Case, Components, and Data Flow.

   Progress Summary: Created a comprehensive architecture document of WeatherSensorAI, including the architectural layers, data flow, component architecture, and three UML diagrams showing user interaction, components, and data flow.

1.2 - Create UML Diagrams for Architecture – Evans Chigweshe

- FR 1.2.1: Produce Use Case diagram showing user → system interactions.
- FR 1.2.2: Produce Component diagram of front end, backend (chat), weather/LLM services.
- FR 1.2.3: Produce Data Flow diagram from location input → API fetch → LLM → UI.

Progress Summary: Created three clear UML diagrams displaying user interaction, component relationships, and data flow between modules. These diagrams were added to the architecture document for reference in later sprints.

1.3 - Create Repository and Initial File Structure - Jackson Yanek

- FR 1.3.1: The repository must be publicly accessible.

Progress Summary: Public Repository Created

1.4 - Set Up Environment Variables for API keys - Manu Redd

- FR 1.4.1: Generate Gemini Pro API key through KU education account
- FR 1.4.2: Create an environment variable to keep API key private
- FR 1.4.3: Write sample code to test API request

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.

1.5 - Create and complete the README file in github - Jackson Yanek

- FR 1.5.1: The README file must have a clean and readable format.

Progress Summary: Basic Readme skeleton pushed to Github Repository.

1.6 - Create simple skeleton website (github hosting) - Jackson Yanek

- FR 1.6.1: Create HTML Skeleton providing base
- FR 1.6.2: Create CSS file providing style
- FR 1.6.3: Enable Github Pages

- FR 1.6.4: Push to Github and ensure basic functionality

Progress Summary: Website created and pushed to GitHub

1.7 - Add CSS Styling and Color Theme - Jackson Yanek

- FR 1.7.1: Define a 2–3 color palette and base typography scale.
- FR 1.7.2: Implement shared CSS variables (colors, spacing, radius).
- FR 1.7.3: Style header, cards, buttons, and chat bubbles to use the theme.

Progress Summary: Created a consistent style sheet using cool blue-gray tones and rounded card layouts. All pages use the same fonts and color variables for uniform design.

1.8 - Verify Deployment of Github Pages – Riley England

- FR 1.8.1: Confirm site builds without errors and is publicly accessible.
- FR 1.8.2: Document deploy steps in README (branch, path, custom domain if any).

Progress Summary: Successfully deployed the static site to GitHub Pages and verified access through the public URL. Deployment instructions were added to the README for team reference.

1.9 - Evaluate and Select Weather Data Source - Riley England

- FR 1.9.1: Research available APIs (OpenWeatherMap, Visual Crossing, WeatherAPI).
- FR 1.9.2: Compare Data fields (temperature, winds, UV)
- FR 1.9.3: Choose a provider supporting all required parameters.

Progress Summary: Evaluated multiple weather APIs and selected Open-Meteo for initial integration due to its coverage, documentation, and free support. No API key needed, validated a simple query.

1.10 - Setup weather database API to retrieve realtime weather information - Manu Redd and Jackson Yanek

- FR 1.10.1: Write a script to send API requests to the selected weather provider.
- FR 1.10.2: Retrieve and print live weather data for a static location to the console.
- FR 1.10.3: Include key parameters such as temperature, wind speed, humidity, cloud cover, and UV index.
- FR 1.10.4: Validate that the response updates dynamically with current data at runtime.
- FR 1.10.5: Log raw API responses for verification and debugging.

Progress Summary: Implemented a script to call the Open-Meteo API and retrieve real-time weather data. Verified that temperature, wind, and humidity values matched current readings. The system prints results to the console in structured JSON format, confirming successful data retrieval and live updates.

1.11 - Verify data fields (temp, wind, UV) - Riley England

- FR 1.11.1: Check that required fields are present in the provider response.
- FR 1.11.2: Validate units are metric (°C, m/s, hPa, mm, km, index).
- FR 1.11.3: Record any missing fields and propose alternatives or derived values.

Progress Summary: Compared Open-Meteo's response schema to required fields and confirmed coverage of temperature, humidity, wind, and UV index. Logged minor omissions and documented how to approximate missing data values.

1.12 - Format API Response - Jackson Yanek

- FR 1.12.1: Extract core fields (temperature, wind speed/direction, humidity, cloud cover, UV, precip).
- FR 1.12.2: Convert timestamps to ISO-8601 with timezone.
- FR 1.12.3: Output a minimal, labeled dictionary ready for UI/LLM.

Progress Summary: Implemented a parser to convert Open-Meteo responses into structured JSON with normalized keys and ISO-formatted timestamps. Tested on multiple weather samples.

1.13 - Create and Complete Sprint 1 Artifacts - Riley England

- FR 1.13.1: Compile Requirements for Sprint 1 into formatted documentation.
- FR 1.13.2: Write functional requirements for each requirement, leaving space for progress summaries.
- FR 1.13.3: Verify artifact alignment with course rubric and project objectives.
- FR 1.13.4: Upload all Sprint 1 documents to github.

Progress Summary: Compiled all Sprint 1 Requirements into the specified format of the Artifacts document. Broke each requirement down into its Functional Requirements, and gave a description for each. Ensured consistent formatting and completeness across all team members' sections before submission. Finally, compiled all Sprint 1 deliverables and uploaded them to the github, ensuring the final version was pushed.