Automated Greenhouse Control System: Operational and Architectural Document

1. Introduction

This document details the features, functions, and architecture of the Automated Greenhouse Control System. This system uses real-time, external weather data to make intelligent, predictive decisions regarding the internal climate of the greenhouse, optimizing conditions for crop growth while minimizing resource consumption.

The system is currently demonstrated using a mock API server (Mockoon) to simulate real-world data feeds, ensuring the control logic is fully testable and functional. Transitioning to a live, external weather API requires only a minor change to the API endpoint URL.

2. System Architecture Overview

The system follows a classic IoT control loop: **Sense** (API Data) → **Decide** (Control Logic) → **Act** (Actuator Command).

Component	Function	Status (in Simulation)
External Weather API	Provides real-time data: outdoorTemp, windSpeed, isRaining, simulatedRainfall.	Mockoon Server (Simulated)
Control Logic	Executes decision-making (JavaScript in iot_tester.html).	Live and Active
Actuators	Physical devices: **Ventilation Fan** and **Sprinkler System**.	Simulated Output (UI/Log)
User Interface	Displays system status and command logs.	Live and Active

3. Key System Features and Functions

The greenhouse automates two critical functions using external weather data: **Climate Regulation (Fan)** and **Precision Irrigation (Sprinklers)**.

3.1. Feature 1: Climate Regulation (Ventilation Fan)

The primary goal is to prevent overheating inside the greenhouse by integrating the **outdoor temperature** and **wind speed** with a predefined internal threshold.
Function: Automatically toggle the fan state (ON/OFF) based on temperature and air flow conditions.

Data Point Used	Threshold / Logic	Actuator Command
greenhouseTemp (Internal)	If temperature exceeds a target threshold (e.g., 28°C or 82°F).	**Primary Trigger**
outdoorTemp (External API)	Used to confirm the external air is cool enough to provide cooling via ventilation. If internal temp is too high, but outdoor temp is also very high (e.g., $>32^\circ\text{C}$), the system might log a warning instead of relying solely on ventilation, or wait for better conditions.	**Logic Check**
windSpeed (External API)	If wind speed is high (e.g., $>15\text{ mph}$), the fan may be controlled more cautiously or	**Safety / Predictive**

	not at all, to avoid stress on the greenhouse structure, **or** high wind is considered beneficial for natural cooling.	**Input**

Example Logic Rule (Implemented in checkGreenhouseTemp):
$$\text{IF} \left(\text{GreenhouseTemp} > T_{\text{max}} \right) \land \left(\text{OutdoorTemp} < T_{\text{outdoor\_limit}} \right) \land \left(\text{FanState} = \text{OFF} \right) \\ \rightarrow \text{Set Fan State} = \text{ON} $$**Note:** For the simulation in `iot_tester.html`, a fixed internal temperature of 30°C is used to demonstrate the fan control logic. ### 3.2. Feature 2: Precision Irrigation (Sprinklers) The system ensures the plants are watered optimally by preventing watering when it is already raining (conserving water) or when a large amount of rain is immediately predicted (predictive optimization). **Function:** Automatically control the sprinkler state (ON/OFF) based on current and predicted precipitation. | Data Point Used | Threshold / Logic | Actuator Command | | :--- | :--- | :--- | | **`isRaining` (External API)** | Boolean check: If `true`, immediately override any watering schedule. | **Primary Safety Lock** | | **`simulatedRainfall` (External API)** | Quantitative check: A value representing predicted or recent rainfall (e.g., millimeters). If this value exceeds a threshold (e.g., $>5\text{ mm}$), the sprinkler command is canceled. | **Predictive Override** | **Example Logic Rule (Implemented in `checkSprinklers`):** $$\text{IF} \left(\text{SprinklerSchedule} = \text{DUE} \right) \\ \text{AND NOT} \left(\text{isRaining} = \text{TRUE} \right) \\ \text{AND} \left(\text{SimulatedRainfall} \< R\_{\text{threshold}} \right) \\ \rightarrow \text{Set Sprinkler State} = \text{ON} $$\#\# 4. API Specification and Simulation Details The system relies on a single API endpoint to retrieve all necessary external environmental data. ### 4.1. Mockoon API Endpoint Details | Parameter | Value | Notes | | :--- | :--- | :--- | | **Base URL** | `http://localhost:3000` | The Mockoon server URL defined in your `iot_tester.html`. | | **Endpoint** | `/weather/realtime` | The specific path for the data. | | **Method** | `GET` | Standard request type for fetching data. | ### 4.2. Response Data Structure (JSON) The external API must return a JSON object with the following fields: ```json { "outdoorTemp": 25.5, "windSpeed": 7.2, "isRaining": false, "simulatedRainfall": 0.5 } ``` | Field Name | Type | Unit (Example) | Purpose | | :--- | :--- | :--- | :--- | | **`outdoorTemp`** | Number | Degrees Celsius | External temperature for fan decision. | | **`windSpeed`** | Number | MPH or km/h | External wind conditions. | | **`isRaining`** | Boolean | N/A | Immediate rain detection for sprinkler lock. | | **`simulatedRainfall`** | Number | Millimeters (mm) | Recent/predicted precipitation

for water conservation. | ## 5\. Transitioning to Real-World Use The current implementation using Mockoon (`http://localhost:3000/weather/realtime`) is functionally identical to a live API call. **To transition to a real-world system, only the following steps are required:** 1. **Select a Real Weather API:** Choose a public weather service (e.g., OpenWeatherMap, WeatherAPI) that provides the required data points (`temp`, `wind`, `rain`). 2. **Update the Endpoint:** Change the `REAL_WORLD_API_URL` variable in the JavaScript section of your application to point to the new, live API endpoint. * **Simulated URL:** `const REAL_WORLD_API_URL = 'http://localhost:3000/weather/realtime';` * **Real URL (Example):** `const REAL_WORLD_API_URL = 'https://api.openweathermap.org/data/2.5/weather?lat=...&appid=...';` 3. **Adapt Data Parsing:** Modify the `fetchAndCheckGreenhouseTemp` and `fetchAndCheckSprinklers` functions to correctly parse and extract the four required values (`outdoorTemp`, `windSpeed`, `isRaining`, `simulatedRainfall`) from the new API's response structure. No changes to the core control logic (`checkGreenhouseTemp` or `checkSprinklers`) would be necessary, as the system is designed to act upon the four abstracted data variables.$$