Project Documentation: Intelligent FSM-Based IoT Sensor Node

1. Project Overview

This project implements a **Finite State Machine (FSM) Controlled IoT Sensor Node** designed for environmental monitoring. Unlike simple "always-on" sensors, this system uses an FSM to strictly manage power states, ensuring maximum battery efficiency and reliability. The system visualizes the internal logic of an IoT node, simulating the behavior of an ESP32 or similar microcontroller.

2. Pin-to-Pin System Logic

Although this is a simulation, it accurately mirrors the logic of a physical hardware implementation. Below is the pin-to-pin, state-by-state breakdown of the system's operation.

A. Core Components (Simulated Hardware)

1. **MCU (Microcontroller):** The Brain (Simulated by React State & Hooks).
2. **Sensors:**
    - **BME280:** Measures Temperature, Humidity, Pressure (Connected via I2C).
    - **SDS011:** Measures PM2.5 and PM10 (Connected via UART).
3. **Power Unit:** Lithium-Ion Battery (Simulated discharge logic).
4. **Comms Module:** WiFi/MQTT radio (Simulated async transmission). ### B. How This Simulation Mimics an ESP32 This project is an **Emulator**. It runs the *exact same logical control structures* that you would write in C++ for an ESP32, but executes them in JavaScript.

ESP32 Hardware Feature	Web Simulation Equivalent	How it Mimics Reality
Main Loop (`void loop()`)	`useEffect` Hook with Timer	Runs a continuous cycle checking for state transitions every X milliseconds.
System State (`enum State`)	React `useState<FSMState>`	Holds the current active mode (e.g., `SENSE`, `SLEEP`) just like a state variable in RAM.
Deep Sleep (`esp_deep_sleep`)	`setTimeout(.., delay)`	The system pauses execution and "waits" for the timer to expire, simulating low-power inactivity.
I2C Sensor Read		Requesting data from the cloud takes time

(`Wire.read()`)	`fetch(API)`	(wait states), simulating the I2C bus latency.
WiFi Radio (`WiFi.begin()`)	Browser Network Stack	Uses the device's actual network interface to connect to the MQTT broker.
Interrupts (`ISR`)	User Click / Async Events	External events (like clicking "Reset") act as hardware interrupts triggering immediate state changes.

---

3. Operational Workflow (The FSM Cycle)

The system operates in a continuous loop defined by the **State Machine Diagram**.

State 1: BOOT (Initialization)

- **Action:** System Power On.
- **Logic:**
  - Initialize variables (Battery=100%).
  - Check configuration.
- **Objective:** Ensures known starting state.
- **Power Draw:** Low (10mA).

State 2: SELF_TEST (Diagnostics)

- **Action:** Check sensor connectivity.
- **Logic:**
  - Ping BME280 & SDS011.
  - If Success (90% chance) -> Go to SLEEP.
  - If Failure -> Go to ERROR.
- **Objective: Reliability.** Prevents running broken hardware.
- **Power Draw:** Low (8mA).

State 3: SLEEP (Deep Sleep - The "Default" State)

- **Action:** Power down all sensors and radio.
- **Logic (Adaptive Duty Cycle):**
  - **Algorithm:** `SleepTime = BaseInterval * Multiplier`
  - **Condition 1 (Normal):** `Multiplier = 1.0` -> Sleep 3.0s.
  - **Condition 2 (Pollution Alert):** IF `PM2.5 > 35` THEN `Multiplier = 0.5` -> Sleep 1.5s (High Resolution).
  - **Condition 3 (Low Battery):** IF `Battery < 20%` THEN `Multiplier = 2.0` -> Sleep 6.0s (Power Saving).
- **Objective: Power Efficiency.** The node spends 90% of its life here to save battery.
- **Power Draw:** Ultra-Low (0.1mA).

State 4: WAKE (Startup)

- **Action:** Wake up interrupt received.
- **Logic:** Restore context, prepare I2C/UART buses.
- **Objective:** Safe transition from low power to active mode.
- **Power Draw:** Medium (5mA).

State 5: SENSE (Data Acquisition)

- **Action:** Power ON sensors and read values.
- **Logic:**
  - **Data Source 1 (BME280):** Fetches real-time Temperature, Humidity, Pressure.
  - **Data Source 2 (SDS011):** Fetches real-time PM10, PM2.5.
  - **Synchronization:** Uses `Promise.all()` to parallel fetch for speed.
- **Objective: Data Acquisition.** Get accurate data only when needed.
- **Power Draw:** High (15mA).

State 6: PROCESS (Decision Making)

- **Action:** Analyze data locally.
- **Logic (Smart Power Gating):**
  - **Check:** Is Battery critical?
  - **Code Implementation:** `javascript     if (sensorData.battery < 10) {         nextState = "SLEEP"; // Emergency Abort         message = "Battery Critical - Skipping TX";     } else {         nextState = "TRANSMIT"; // Proceed     }`
- **Objective: Fault Tolerance & Preservation.** Prioritize survival over reporting.
- **Power Draw:** High (20mA).

State 7: TRANSMIT (Communication)

- **Action:** Enable WiFi/Radio and send MQTT packet.
- **Logic:**
  - **Protocol:** MQTT (Lightweight IoT Protocol).
  - **Broker:** `wss://58071564a2bd44eeacfb16945302d2d6.s1.eu.hivemq.cloud:8884/mqtt`
  - **Topic:** `adld/sensor/data`
  - **Payload:** JSON `{ temp, hum, pressure, pm10, pm25, batt }`
- **Objective: Communication.** Cloud data sync.
- **Power Draw: MAXIMUM (50mA).** This is the most expensive state.

State 8: ERROR / REPAIR (Fault Handling)

- **Action:** Exception detected.
- **Logic:**
  - Log the error.
  - Wait `config.errorRecoveryTime` (Cool down).
  - Retry -> Go to SLEEP.
- **Objective: Reliability.** Self-healing capabilities.

---

4. Satisfying Project Objectives

Objective	Implementation Detail
Sensors Data Acquisition	Real-time fetching of 6 parameters (Temp, Hum, Pressure, PM10, PM2.5, Battery) during the `SENSE` state.
Low Power/Sleep Cycles	The system sits in `SLEEP` state by default. We implemented **Adaptive Duty Cycling** which actively changes sleep time based on battery and pollution levels.
Communication	Full MQTT implementation sending JSON payloads to a public broker during the `TRANSMIT` state.
Power Efficiency	**Smart Power Gating**: The system strictly refuses to transmit if battery < 10%. It also sleeps 2x longer when battery < 20%.
Reliable State Transitions	The `VALID_TRANSITIONS` table enforces a strict, deterministic flow. The `ERROR` state ensures no “hangs” occur; the system always self-heals.