Tech Analyst Intern Assessment

Exercise: Smart Building IoT Integration

Estimated Time Limit: Approximately 6 hours

Objective: COOi Studios has entrusted you with the responsibility of designing and implementing a simplified IoT system for a smart building that monitors temperature and controls HVAC systems. The solution must integrate with a mock legacy HVAC system via REST API.

Tasks

1. **System Architecture (Design)**
    - Design an IoT architecture for a building with 5 rooms, each equipped with:
        - A temperature sensor (simulated).
        - An HVAC actuator (simulated).
    - Include communication protocols (e.g., MQTT, HTTP), data flow, and integration with the legacy HVAC API.
2. **Sensor/Actuator Simulation (Coding)**

- Write a Python script to simulate temperature sensors (random values between 15°C–35°C) publishing data to an MQTT broker.
- Write a Python script to simulate actuators subscribing to MQTT topics to receive HVAC commands (ON/OFF).

3. **Legacy System Integration (Coding)**
   - The legacy HVAC system has a REST API with two endpoints:
     - GET /api/hvac/status: Returns {"status": "active"} or {"status": "inactive"}.
     - POST /api/hvac/command: Accepts {"command": "activate"} or {"command": "deactivate"}.
   - Create a middleware service (in Python/Node.js) to:
     - Poll temperature data from the MQTT broker.
     - Trigger HVAC commands via the legacy API if any room's temperature exceeds 30°C.
     - Log actions to a file.
4. **Security & Error Handling**
   - Add basic authentication to the legacy HVAC API.
   - Handle edge cases (e.g., API downtime, invalid sensor data).
5. **Documentation**
   - Write a README explaining:
     - How to run the system.
     - Design decisions (protocols, security, error handling).
     - Describe

Rubric

Category	**Criteria**	**Points (1–20)**

IoT Understanding	- Correct use of protocols (MQTT/HTTP). - Efficient sensor/actuator simulation.	20
Systems Integration	- Seamless interaction between MQTT and REST API. - Error-free legacy system integration.	20
Code Quality	- Clean, modular code. - Proper error handling and security practices.	20
Problem-Solving	- Logical edge-case handling (e.g., API retries, data validation).	20
Documentation	- Clear setup instructions. - Justification of design choices.	20

Bonus Points (Optional):

- Real-time dashboard for monitoring (e.g., using Flask/WebSocket).
- Unit tests for critical components.

Expected Deliverables:

1. Architecture diagram (PDF/image).
2. Code files (Python/Node.js you can use your language choice but explain the rationale for your decision on the README file).
3. README.md. (Give insights and your thinking on your design choices)