Fullstack Software Developer Test

This test represents a slice of AltoTech's mission to harness IoT in the building/hotel sector. While there's a clear framework, innovative approaches are always appreciated. Should you have any questions, please reach out. Let's get started!

Objective: Create a simplified version of a Smart Hotel system with the following goals:

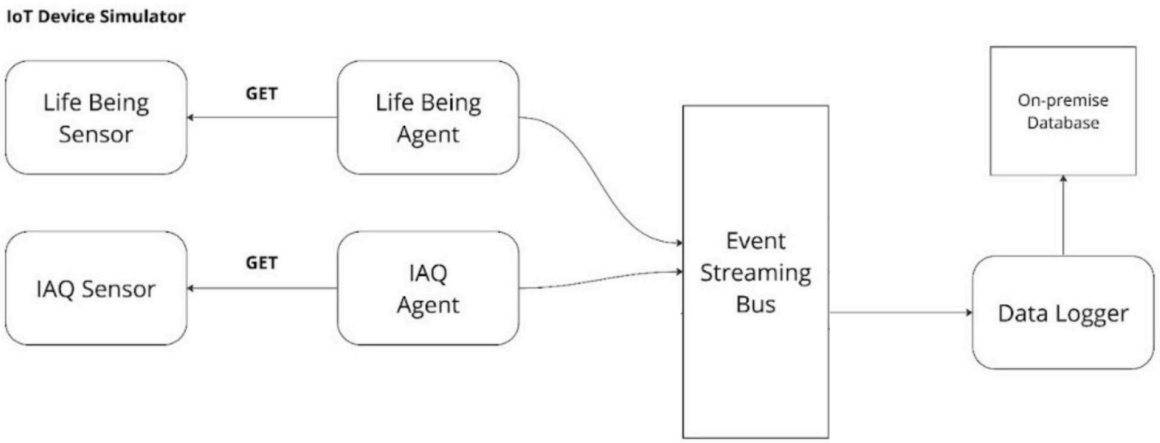
- Stimulate IoT data from each hotel room into the Smart Hotel platform
- Offer transparent data on hotel room usage from website interface and downloadable datalog.
- Enhance the guest experience with the Large Language Model (LLM) interface.

How?

- **IoT Integration in Rooms**: Equip hotel rooms with:
    - Life Being Sensor: Detects motion (occupied, unoccupied)
    - IAQ Sensor: Measures indoor environment data (temp, rH, CO2)
- **Energy Consumption**: Using data from power meters (power: kW) installed within the hotel to calculate total energy consumption by system.
- **Enhance Guest Experience**: Allow guests to access room data via a web app on their mobile devices. To simplify the user interaction and enhance their experience, a chatbot-like interface is needed.

1. On-premise IoT Infrastructure

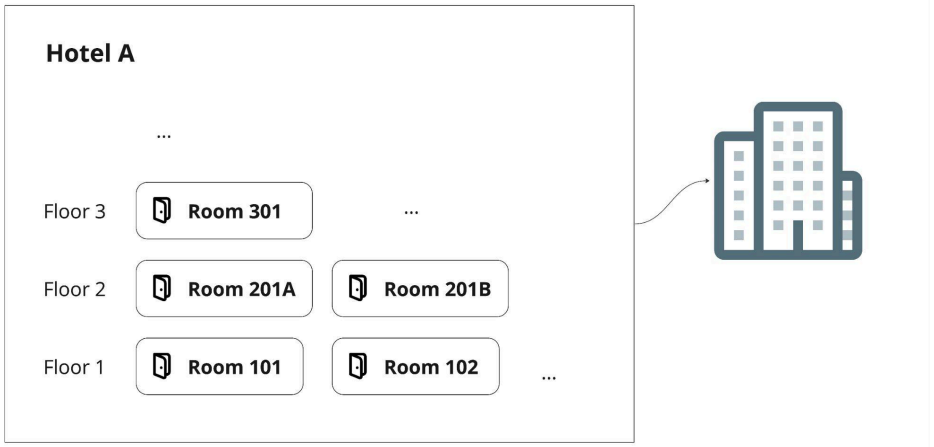
Develop an IoT system for a Hotel Room to stimulate sensor data and store to local databases using PUB-SUB approach though Event Streaming Bus.



- **Agent - IAQ Sensor:** Develop Python agents that stimulate IoT sensor data to PUBLISH to Event Streaming Bus at every 5 seconds. To simulate these mock data, please use provided CSV files as reference.
    - IAQ Sensor datapoint: co2, humidity, temperature
- **Agent - LifeBeing Sensor:** Develop Python agents that stimulate IoT sensor data to PUBLISH to Event Streaming Bus at every 5 seconds. To simulate these mock data, please use provided CSV files as reference.
    - Life Being Sensor datapoint: presence_state, sensitivity
- **Agent - Datalogger:** Develop Python agent that SUBSCRIBE to IoT data payload from IoT Sensor Agents and store them to Timeseries database (TimescaleDB) and Realtime database (Supabase).
- **Event Streaming Bus:** Use RabbitMQ service as medium for Agents to PUB-SUB data payloads.
- **Timeseries database:** Use TimescaleDB database to store historical data of IoT Sensors. Please design **"raw_data" table** as follows:
    - timestamp (INTEGER): Unix timestamp of the data point
    - datetime (TIMESTAMPTZ): Datetime of the data point
    - device_id (TEXT): Unique identifier of the device
    - datapoint (TEXT): Type of measurement (e.g., temperature, humidity)
    - value (TEXT): Measured value (use TEXT type because some value is string)

- **Prepare Timeseries Database**: Please fill Power Meter (kW) data into the TimescaleDB database in the correct "raw_data" table format. There's 6 power meter data provided in CSV files. (required for Backend tasks)
- **Realtime database:** Use Supabase database to store latest sensor readings with device-level indexing for efficient querying.
- **Relational database:** Use PostgreSQL database to store hierarchical relationships between hotels, floors, rooms, and IoT devices with proper foreign key constraints and metadata fields.

2. Backend Services (Django-Python)



The backend service requires RESTful APIs to provide efficient access to IoT data across multiple hotels. The system must handle unique room/floor identifiers and device management. Required endpoints include:

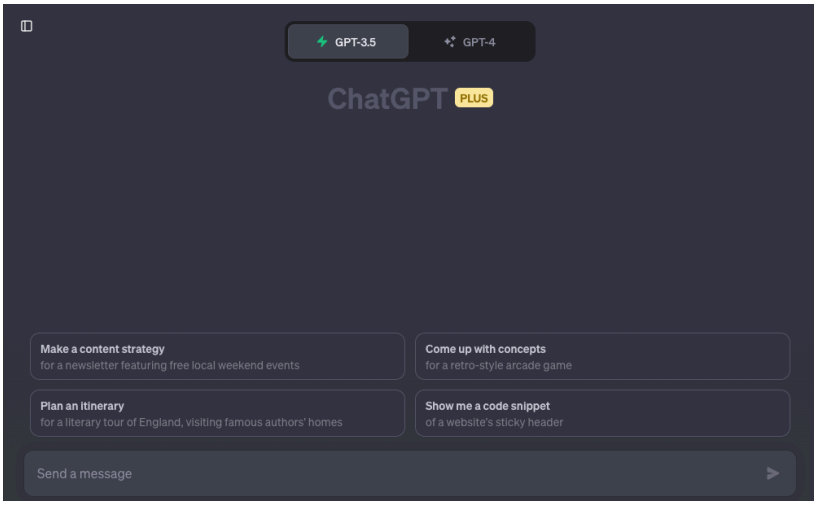
- **API - get relation, get latest data:**
  - url/hotels/: Retrieve all hotels.
  - url/hotels/<hotel_id>/floors/: Access floors in a hotel.
  - url/floors/<floor_id>/rooms/: List rooms on a floor.
  - url/rooms/<room_id>/data/: Latest IoT data of a room.
  - url/rooms/<room_id>/data/life_being/: Latest Life Being sensor data of a room.
  - url/rooms/<room_id>/data/iaq/: Latest IAQ sensor data of a room.
- **API - Energy Consumption Summary:**

- endpoint: url/hotels/<hotel_id>/energy_summary/
- Retrieves Power consumption data (kW) from TimeScaleDB database and calculates Energy consumption (kWh) for specified hotel subsystems (AC, lighting, plug load) based on time resolution.
- **Technique**: Please use Timescale's time bucket aggregation or Python-Pandas only.
- **Equation**: energy_consumption = average(kW) x runhour
- **Returns** CSV with aggregated energy consumption (kWh) per timestamp across all subsystems.
  CSV format:

timestamp	ac	lighting	plug_load	
2024-01-01 00:00:00	12.5	8.3	5.2	<-- kWh value
2024-01-01 00:15:00	13.1	8.5	5.4	
...				

- Supports filtering by subsystem and custom date ranges.
  Query parameters:
  - resolution: [1hour, 1day, 1 month] (required)
  - subsystem: [ac, lighting, plug_load] (optional: default as included all subsystem)
  - start_time: ISO timestamp (optional)
  - end_time: ISO timestamp (optional)
- Power Meter included in each subsystem:
  - AC system: power_meter_1, power_meter_2, power_meter_3
  - Lighting system: power_meter_4, power_meter_5
  - Plug Load system: power_meter_6

3. LLM System and Chat Interface



- **Chatbot-like Interface**: The system should have an easy-to-use interface for hotel guests to interact with IoT devices within their room through Text interactions. (you can use UI framework ex. Gradio)
- **Smart Functionality**: The guest interface should provide smart functionality to enhance the guest experience. Essential features include:
    1. Access real-time sensor data
    2. Access historical sensor data
    3. Utilize the Backend's EnergyAPI to request and provide CSV to the user
- **Anthropic's Clade Function Calling Feature**: One big challenge in the development of an LLM system is for the LLM to prepare the required parameters and call our custom Python functions correctly. Multiple providers have a feature called 'Function Calling,' which allows this event to happen with ease. Please apply **Anthropic's** 'Tool Use' method to the system to seamlessly connect our user requests to our software.

Note: You can also try other provider ex. OpenAI's "Function Calling" feature.
Note: You're free to suggest alternative strategies on LLM technique or framework.

4. Docker-Compose Deployment

- **Ease of Deployment:** The infrastructure should be able to deploy easily with minimal effort. Please use Docker-Compose script to enable users to setup and run the entire infrastructure.
- **Documentation**: Please provide instructions on how to setup and run your deployment implementation.

Guidelines:

- **Tools:** While Django, Docker, and GitHub are mandatory, you have freedom in the choice of tools.
- **Submission:**
    - **GitHub Repository:** This should house all code. The README must contain clear instructions for reproducing your setup.
    - **Github Wiki:**
        - System Architecture: Visual representation of connectivity between modules.
        - Flow Charts and Sequence Diagrams: Visual representation of processes.
        - API & Technical Documentation: Detailed explanation of your APIs and technicalities.
        - ER Diagram: Showcase the database relationships.
        - Reasoning: Justify the tools and methodologies you adopted.
        - Provide a demo in the form of screenshots or videos.
    - Feel free to use any presentation material. We expected a real DEMO.

Remember, at AltoTech, collaboration is key. Consider me as your teammate and don't hesitate to reach out.

We recognize the depth of this task, but it mirrors the challenges we tackle at AltoTech. It's not just about your current expertise but also your adaptability and learning pace. Your efforts will resonate with AltoTech's mission and enhance your professional journey. Good Luck !!

Resources:

- Antropic Clade documentation: https://docs.anthropic.com/en/api/getting-started
- Anthropic's Clade ToolUse: https://docs.anthropic.com/en/docs/build-with-claude/tool-use
- Alternative LLM solutions:
  - OpenAI documentation: https://platform.openai.com/docs/introduction
  - OpenAI's Function Calling: https://openai.com/blog/function-calling-and-other-api-updates
  - Youtube on OpenAI's Function Calling: https://www.youtube.com/watch?v=0lOSvOoF2to
  - Pydantic - Advanced LLM framework: https://ai.pydantic.dev/
- Gradio document: https://www.gradio.app/docs/interface