Take-Home Exercise: Orb Fleet Management Service

Welcome and thank you for your interest in joining the Fleet Management team. In this exercise, you will build a small microservice that simulates a core component of our system. The goal is to assess your ability to write clear, functional code and your familiarity with designing distributed systems.

We use AI tools heavily on our team, so if you'd like to use any, that's totally fine.

Overview

Your task is to create a service that manages telemetry update messages from Orb devices. Each update message will include at a minimum:

- **Device Identifier** (e.g., a string or UUID)
- **Geographic Location** (latitude, longitude)
- **Timestamp** (an ISO-8601 string or equivalent)

Your service should provide endpoints implementations for the devices to upload the telemetry messages to. Additionally, provide APIs to allow other clients to retrieve suitable summary reports for the telemetry data.

- Document your considerations for what makes your API a good design for the scenario
- Document some of the extra functionality you chose to add to the behavior of the API that would benefit how we manage the devices
- Document any considerations you put in place to handle issues you might encounter with clients

Important Notes:

- **Storage**: You do not need to use any specific database. You can use any in-memory or lightweight storage mechanism as long as the stored data can be retrieved later by the same service.
- **Demonstration**: You should be able to demonstrate that your service works by including unit/integration tests or by providing a test client that simulates the Orb functionality.

Requirements

- **Language**: Any compiled language (Rust, Go, Java, C#, etc.)
- **Testing**: Provide tests (unit or integration) or a small client to demonstrate your service's functionality.
- **Documentation**: Include brief comments or documentation to explain key design decisions.

Exercise Details

1. **Define and implement APIs** for the above design.

3. **Demonstrate Functionality**:
    - Provide tests or a test client (could be a command-line client or simple script) that shows:
        - Sending one or more telemetry messages in parallel.
        - Retrieving the latest telemetry for a given device.
        - Robust error handling

4. **Optional Enhancements**
    - Containerize your service with a Dockerfile.

Provide additional documentation on your design choices and any trade-offs you made.

Time Expectation

This exercise should take approximately **2–3 hours**. We do not expect production-ready code, but we do value clear, maintainable, and well-documented code.

Submission Instructions

- **Source Code and documents**: Please submit your source code in a compressed format (e.g., a ZIP file) or a link to a Git repository.
- **Instructions**: Ensure your project includes a `README.md` with build and run instructions.
- **Testing**: Include instructions on how to run your tests or demo client.

Evaluation Criteria

We will be evaluating your submission based on:

- **Correctness**: Does the service behave as specified?
- **Code Quality**: Is your code clear, maintainable, and well-structured?
- **Testing/Verification**: Did you provide a way to demonstrate that your service works as expected?
- **Documentation**: We are very interested in understanding the behaviors you chose for your APIs

Thank you for your time and good luck!