Coding Challenge

Mission

You've just been recruited by an elite team of Jedi engineers to implement a Spaceship Power Service for the Rebel Alliance's fleet of space vehicles. The Spaceship Power Service is needed so that rebel coordinators can keep track of the energy usage of different ships. Your task is to implement the Spaceship Power Service that handles POST and GET requests to an API endpoint.

The Spaceship Power Service minimum viable prototype (MVP) should carry out the following logic each time the following endpoints are called. See accompanying request and response examples.

`POST /data`

1. Given a spaceship_id and a set of energy consumption data in the request body, parse data and translate to a standard 15-minute energy format, following formatting guidelines below
2. Persist data

`GET /data?spaceship_id=x&start=y&end=z`

1. Parse spaceship_id, start, and end query parameters
2. Return the set of 15-minute energy data corresponding to the spaceship and time period if it exists

Data Formatting Guidelines

- The Spaceship Power Service should store all data as energy in 15-minute intervals, in units of kWh
- Three different formats are permitted for POSTed energy consumption data. Each file contains an example POST request of energy consumption timeseries data, but in different formats
- Depending on the POST format, some values may be in kW (power) or kWh (energy)
    - kWh = kW * hours
    - Ex: 6 kW consumed for 5 minutes = 0.5 kWh
- The interval of the timestamps in POSTed data may vary: i.e., 5 minute, 15 minute, 1 hour
    - All timestamps are hour-beginning, meaning that hour 0 = 12:00am
    - When aggregating kW, use the mean. For example, aggregating three 5-minute values (1 kW, 4 kW, 1 kW) should be 2 kW for the 15-minute period
    - If disaggregating kW (i.e. 1 hour to 15 minutes), assume the same value for each 15-minute sub-interval
    - When aggregating kWh, take the sum across sub-intervals
- Timestamps for all data are in UTC, ISO8601 format
- Handle missing data; it is up to you how you do so
- In the event that multiple values are supplied for the same spaceship and timestamp, the most recent values should be used and old values overwritten
- Handle any non-numeric values erroneously entered in the raw dataset; it is up to you how you do so

The MVP application must include:

- Unit tests
- Request validation logic
- Storage/repository layer for persisting POSTed timeseries data using your database of choice

You can implement this service in any language you choose, but it should run without error on Linux. In addition to your code, please submit a design document of no more than 800 words that describes your design decisions, tradeoffs, assumptions, and edge cases that may be handled or unhandled. Please send both code and design doc in a compressed folder. They will be reviewed during the technical portion of the phone/on-site interview. This exercise should take anywhere from 4 to 8 hours to complete. We please ask that you spend no longer.

Bonus Points if your implementation includes any of the following suggestions:

- Display/visualization that allows the user to interact with the data and/or alerts the user when there are data anomalies (outliers, missing data, etc.); feel free to be creative
- Deployment spec using container runtime/orchestration technologies of your choice
- Integration tests
- Data model/schema migration solution for storage layer
- Any other creative elements you'd like to incorporate :)