Take-Home Test: Space Ground Software Industry

Scenario: Satellite Telemetry Dashboard

You are tasked with building a system to display and manage telemetry data from satellites. Telemetry data includes information such as satellite ID, timestamp, altitude, velocity, and health status. The system should allow users to view, filter, and add new telemetry entries.

Backend Task: REST API for Satellite Telemetry

Objective

Design and implement a REST API to manage satellite telemetry data.

Requirements

1. **Endpoints**
   Implement the following API endpoints:
   - **`GET /telemetry`**: Retrieve all telemetry data.
     - Optional query parameters:
       - `satelliteId`: Filter by satellite ID.
       - `status`: Filter by health status (e.g., "healthy", "critical").
   - **`POST /telemetry`**: Add a new telemetry entry.
     - Request body should include:

```
{
  "satelliteId": "string",
  "timestamp": "ISO 8601 datetime",
  "altitude": "number",
  "velocity": "number",
  "status": "string"
}
```

   - **`GET /telemetry/:id`**: Retrieve a specific telemetry entry by ID.
   - **`DELETE /telemetry/:id`**: Delete a specific telemetry entry.
2. **Data Storage**
   - Use an in-memory database (e.g., SQLite, or a simple array for simplicity).
   - Ensure data persistence during runtime.
3. **Validation**
   - Validate incoming data:
     - `timestamp` must be a valid ISO 8601 datetime.

- `altitude` and `velocity` must be positive numbers.

4. **Implementation Details**
   - Use a backend framework of your choice (e.g., Node.js with Express, Python with Flask/FastAPI, etc.).
   - Write clean, modular code with comments explaining key decisions.

Bonus Features

- Add pagination support to the `GET /telemetry` endpoint.
- Implement unit tests for the API endpoints.
- Include Docker support for easy deployment.

Frontend Task: Satellite Telemetry Dashboard

Objective

Design and implement a web interface to display and manage satellite telemetry data.

Requirements

1. **Features**
   Build a web application with the following functionality:
   - Display a table of telemetry data with columns: Satellite ID, Timestamp, Altitude, Velocity, Health Status.
   - Allow users to filter telemetry data by Satellite ID and Health Status.
   - Provide a form to add new telemetry entries.
   - Allow users to delete telemetry entries.
2. **Mock API or Backend Integration**
   - You may use the backend API provided in the test or create a mock API for the frontend task.
   - If using a mock API, simulate the following endpoints:
     - `GET /telemetry`: Returns a list of telemetry data.
     - `POST /telemetry`: Accepts a new telemetry entry.
     - `DELETE /telemetry/:id`: Deletes a telemetry entry.
3. **Design**
   - Use a modern frontend framework (e.g., React, Angular, Vue).
   - Ensure the UI is responsive and user-friendly.
4. **Implementation Details**
   - Use state management (e.g., React's `useState` or Redux) to manage telemetry data.
   - Validate form inputs before sending data to the backend or mock API.

Bonus Features

- Add sorting functionality to the table (e.g., sort by Timestamp, Altitude, or Velocity).
- Implement error handling for API requests (e.g., display error messages if the backend is unreachable).
- Include unit tests for key components.
- Add a loading spinner while fetching data from the backend or mock API.

Submission Guidelines

Backend Task

- Provide a link to the repository containing the code.
- Include instructions for running the API locally (e.g., `npm install && npm start` or `python app.py`).
- If Docker is implemented, include a `Dockerfile` and instructions for running the container.

Frontend Task

- Provide a link to the repository containing the code.
- Include instructions for running the app locally (e.g., `npm install && npm start`).
- If applicable, provide a live demo link (e.g., deployed on Vercel, Netlify, etc.).

Documentation

- Include a short README file explaining your approach, any assumptions made, and how to test the application.